Supplemental Code

Metadata and postprocessing utilities 2
- clade_utils.py 2
- classes.py 11
- config.py 11
- core_gene_utils.py 13
- diversity_utils.py 18
- gene_diversity_utils.py 54
- midas_db_utils.py 62
- parse_midas_data.py 66
- parse_patric.py 85
- plot_utils.py 90
- sample_utils.py 92
- sfs_utils.py 117
- snps_utils.py 121
- species_phylogeny_utils.py 125
- stats_utils.py 129
- substitution_rates_utils.py 132
- temporal_changes_utils.py 134

Scripts to store data for plotting 137
- pickle_core_genes_sharing.py 137
- pickle_details.py 138
- pickle_everything.py 141
- pickle_ferretti_snp_changes.py 151
- pickle_mom_seeding.py 154
- pickle_pnps.py 158
- pickle_qp.py 160
- pickle_transmission.py 161

Plotting scripts 164
- plot_figure_1.ipynb 164
- plot_figure_2.ipynb 193
- plot_figure_3.ipynb 299
- plot_figure_4-5.ipynb 319
- plot_metadata.ipynb 361
- plot_qp.ipynb 370
- plot_qp_pairs.ipynb 387
- plot_revisions.ipynb 392
- plot_sfs.ipynb 458

Metadata and postprocessing utilities

clade_utils.py

```
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import cophenet
from scipy.cluster.hierarchy import fcluster
from numpy.random import shuffle, normal

import diversity_utils

import parse_midas_data
import numpy

# min_d = pick only a single sample per cluster with distance below this value
# max_d = cut tree at this distance
def cluster_samples(distance_matrix, min_d=0, max_ds=[1e09]):

    # calculate compressed distance matrix suitable for agglomerative clustering
    Y = []
    for i in xrange(0,distance_matrix.shape[0]):
        for j in xrange(i+1,distance_matrix.shape[1]):
            Y.append(distance_matrix[i,j])
    Y = numpy.array(Y)

    Z = linkage(Y, method='average')

    # First coarse-grain things less than min_d apart:
    #NRG: what does it mean to coarse-grain?
    subcluster_assignments = fcluster(Z, min_d, criterion='distance')

    coarse_grained_idxs = []
    subcluster_idx_map = {}
    for i in xrange(0,len(subcluster_assignments)):
        if subcluster_assignments[i] not in subcluster_idx_map:
            subcluster_idx_map[subcluster_assignments[i]] = i
            coarse_grained_idxs.append(True)
        else:
            coarse_grained_idxs.append(False)

    coarse_grained_idxs = numpy.array(coarse_grained_idxs)


    sorted_final_clusterss = []
    for max_d in max_ds:

        cluster_assignments = fcluster(Z, max_d, criterion='distance')

        cluster_idx_map = {}

        for i in xrange(0,len(cluster_assignments)):

            if not coarse_grained_idxs[i]:
                continue

            if cluster_assignments[i] not in cluster_idx_map:
                cluster_idx_map[cluster_assignments[i]] = []
            cluster_idx_map[cluster_assignments[i]].append(i)

        cluster_labels = set(cluster_idx_map.keys())
        cluster_idxss = [set(cluster_idx_map[cluster_label]) for cluster_label in
cluster_labels]
```

```
        cluster_sizes = [len(cluster_idxs) for cluster_idxs in cluster_idxss]

        # only return ones with more than one individual
        final_clusters = []
        final_cluster_sizes = []

        for cluster_idx_set in cluster_idxss:

            if len(cluster_idx_set)>1:

                cluster_idxs = numpy.array([(i in cluster_idx_set) for i in
xrange(0,len(cluster_assignments))])

                final_clusters.append(cluster_idxs)
                final_cluster_sizes.append((cluster_idxs*1.0).sum())

        if len(final_cluster_sizes) > 0:

            final_cluster_idxs = [i for i in xrange(0,len(final_cluster_sizes))]

            final_cluster_sizes, final_cluster_idxs = zip(*sorted(zip(final_cluster_sizes,
final_cluster_idxs),reverse=True))


            sorted_final_clusters = [final_clusters[idx] for idx in final_cluster_idxs]
            sorted_final_clusterss.append(sorted_final_clusters)
        else:
            sorted_final_clusterss.append([])

    return coarse_grained_idxs, sorted_final_clusterss


# Perform hierarchical clustering respecting the clade boundaries in clade_idxss
def cluster_samples_within_clades(distance_matrix, clade_idxss=[], d=1e09):


    if len(clade_idxss)==0:
        clade_idxss = [numpy.array([True for i in xrange(0,distance_matrix.shape[0])])]

    subcluster_sets = []

    for clade_idxs in clade_idxss:

        numeric_clade_idxs = numpy.nonzero(clade_idxs)[0]

        # get subset distance matrix
        sub_distance_matrix = distance_matrix[numpy.ix_(numeric_clade_idxs, numeric_clade_idxs)]

        # calculate compressed distance matrix suitable for agglomerative clustering
        Y = []
        for i in xrange(0,sub_distance_matrix.shape[0]):
            for j in xrange(i+1,sub_distance_matrix.shape[1]):
                Y.append(sub_distance_matrix[i,j])
        Y = numpy.array(Y)

        Z = linkage(Y, method='average')

        # First coarse-grain things less than min_d apart:
        subcluster_assignments = fcluster(Z, d, criterion='distance')

        new_subcluster_sets = {}
```

```
        for i in xrange(0,len(subcluster_assignments)):

            if subcluster_assignments[i] not in new_subcluster_sets:
                new_subcluster_sets[subcluster_assignments[i]] = set([])

            new_subcluster_sets[subcluster_assignments[i]].add( numeric_clade_idxs[i] )

        for subcluster_set in new_subcluster_sets.values():
            if len(subcluster_set) > 1:
                subcluster_sets.append(subcluster_set)

    return subcluster_sets




def calculate_phylogenetic_consistency(allele_counts_map, passed_sites_map, proposed_clusters,
allowed_variant_types=set(['1D','2D','3D','4D']), allowed_genes=set([])):

    clusters = []
    anticlusters = []
    for cluster_idxs in proposed_clusters:

        #print cluster_idxs.sum(), numpy.logical_not(cluster_idxs).sum()

        if cluster_idxs.sum() > 1.5: # Need at least two guys in a cluster to look for
polymorphisms

            anticluster_idxs = numpy.logical_not(cluster_idxs)

            if anticluster_idxs.sum() > 1.5: # Likewise for the anticluster

                clusters.append(cluster_idxs)
                anticlusters.append( anticluster_idxs )

    total_genes = set(passed_sites_map.keys())

    if len(allowed_genes)==0:
        allowed_genes = set(passed_sites_map.keys())

    allowed_genes = (allowed_genes & total_genes)

    singleton_freqs = [] # actual freq value is meaningless..
    polymorphic_freqs = [] # non-singleton freqs -- only ones that can be inconsistent!
    inconsistent_freqs = []
    null_inconsistent_freqs = []

    singleton_variant_types = {variant_type: 0 for variant_type in allowed_variant_types}
    polymorphic_variant_types = {variant_type: 0 for variant_type in allowed_variant_types}
    inconsistent_variant_types = {variant_type: 0 for variant_type in allowed_variant_types}
    null_inconsistent_variant_types = {variant_type: 0 for variant_type in
allowed_variant_types}

    if len(clusters)>0: # Can only do stuff if there are clusters!


        for gene_name in allowed_genes:

            for variant_type in passed_sites_map[gene_name].keys():
```

```
                if variant_type not in allowed_variant_types:
                    continue

                allele_counts = allele_counts_map[gene_name][variant_type]['alleles']
                if len(allele_counts)==0:
                    continue

                # good to go, let's get calculating

                # take consensus approximation
                genotype_matrix, passed_sites_matrix =
diversity_utils.calculate_consensus_genotypes(allele_counts)

                population_prevalence = (genotype_matrix*passed_sites_matrix).sum(axis=1)
                population_max_prevalence = (passed_sites_matrix).sum(axis=1)

                population_minor_prevalence = numpy.fmin(population_prevalence,
population_max_prevalence - population_prevalence)

                population_freqs =
population_prevalence*1.0/(population_max_prevalence+10*(population_max_prevalence<0.5))
                population_freqs = numpy.fmin(population_freqs, 1-population_freqs)

                is_polymorphic = numpy.zeros(genotype_matrix.shape[0])
                is_inconsistent = numpy.zeros(genotype_matrix.shape[0])

                for cluster_idxs,anticluster_idxs in zip(clusters,anticlusters):


                    cluster_prevalence =
(genotype_matrix[:,cluster_idxs]*passed_sites_matrix[:,cluster_idxs]).sum(axis=1)
                    cluster_min_prevalence = 1-1e-09
                    cluster_max_prevalence = (passed_sites_matrix[:,cluster_idxs]).sum(axis=1)-
1+1e-09

                    cluster_freqs =
cluster_prevalence*1.0/(cluster_max_prevalence+10*(cluster_max_prevalence<0.5))
                    cluster_freqs = numpy.fmin(cluster_freqs, 1-cluster_freqs)

                    anticluster_prevalence =
(genotype_matrix[:,anticluster_idxs]*passed_sites_matrix[:,anticluster_idxs]).sum(axis=1)
                    anticluster_min_prevalence = 1-1e-09
                    anticluster_max_prevalence =
(passed_sites_matrix[:,anticluster_idxs]).sum(axis=1) -1+1e-09

                    # Those that are polymorphic in the clade!
                    polymorphic_sites =
(cluster_prevalence>=cluster_min_prevalence)*(cluster_prevalence<=cluster_max_prevalence)

                    # Those that are also polymorphic in the remaining population!
                    inconsistent_sites =
polymorphic_sites*(anticluster_prevalence>=anticluster_min_prevalence)*(anticluster_prevalence<=
anticluster_max_prevalence)

                    is_polymorphic = numpy.logical_or(is_polymorphic, polymorphic_sites)
                    is_inconsistent = numpy.logical_or(is_inconsistent, inconsistent_sites)

                if is_polymorphic.sum() > 0:

                    is_singleton = (numpy.fabs(population_minor_prevalence-1)<1e-
08)*is_polymorphic
```

```
                    is_polymorphic = (population_minor_prevalence>1.5)*is_polymorphic

                    singleton_freqs.extend( population_freqs[is_singleton] )
                    singleton_variant_types[variant_type] += is_singleton.sum()

                    polymorphic_freqs.extend( population_freqs[is_polymorphic] )
                    polymorphic_variant_types[variant_type] += is_polymorphic.sum()

                    if is_inconsistent.sum() > 0:
                        #inconsistent_freqs.extend( cluster_freqs[is_inconsistent] )
                        inconsistent_freqs.extend( population_freqs[is_inconsistent] )
                        inconsistent_variant_types[variant_type] += is_inconsistent.sum()

                    # now try to compute a null expectation for a completely unlinked genome
                    polymorphic_idxs = numpy.arange(0,genotype_matrix.shape[0])[is_polymorphic]
                    # Loop over sites that were polymorphic, generate a "null" draw for them
                    for site_idx in polymorphic_idxs:

                        genotypes = genotype_matrix[site_idx,:]
                        passed_sites = passed_sites_matrix[site_idx,:]
                        population_freq = population_freqs[site_idx]

                        permuted_idxs = numpy.arange(0,len(genotypes))

                        is_polymorphic = False
                        is_inconsistent = False
                        # loop until we find a polymorphic site
                        while not is_polymorphic:

                            # permute indexes
                            shuffle(permuted_idxs)

                            permuted_genotypes = genotypes[permuted_idxs]
                            permuted_passed_sites = passed_sites[permuted_idxs]

                            # loop through clusters
                            is_inconsistent = False
                            for cluster_idxs,anticluster_idxs in zip(clusters,anticlusters):


                                cluster_prevalence =
(permuted_genotypes[cluster_idxs]*permuted_passed_sites[cluster_idxs]).sum()
                                cluster_min_prevalence = 0.5
                                cluster_max_prevalence =
(permuted_passed_sites[cluster_idxs]).sum()-0.5


                                anticluster_prevalence =
(permuted_genotypes[anticluster_idxs]*permuted_passed_sites[anticluster_idxs]).sum()
                                anticluster_min_prevalence = 0.5
                                anticluster_max_prevalence =
(permuted_passed_sites[anticluster_idxs]).sum() - 0.5

                                polymorphic_in_cluster =
((cluster_prevalence>cluster_min_prevalence)*(cluster_prevalence<cluster_max_prevalence))
                                inconsistent_in_cluster =
(polymorphic_in_cluster*(anticluster_prevalence>anticluster_min_prevalence)*(anticluster_prevale
nce<anticluster_max_prevalence))

                                is_polymorphic = is_polymorphic or polymorphic_in_cluster
```

```
                                is_inconsistent = is_inconsistent or inconsistent_in_cluster

                        if is_inconsistent:
                            null_inconsistent_freqs.append(population_freq)
                            null_inconsistent_variant_types[variant_type] += 1

    singleton_freqs = numpy.array(singleton_freqs)
    polymorphic_freqs = numpy.array(polymorphic_freqs)
    inconsistent_freqs = numpy.array(inconsistent_freqs)
    null_inconsistent_freqs = numpy.array(null_inconsistent_freqs)

    return singleton_freqs, polymorphic_freqs, inconsistent_freqs, null_inconsistent_freqs,
singleton_variant_types, polymorphic_variant_types, inconsistent_variant_types,
null_inconsistent_variant_types

def calculate_clade_allele_freqs(allele_counts_map, passed_sites_map, clusters,
allowed_variant_types=set(['1D','2D','3D','4D']), allowed_genes=set([])):

    anticlusters = []
    for cluster_idxs in clusters:
        anticlusters.append( numpy.logical_not(cluster_idxs) )

    total_genes = set(passed_sites_map.keys())

    if len(allowed_genes)==0:
        allowed_genes = set(passed_sites_map.keys())

    allowed_genes = (allowed_genes & total_genes)

    ktotals = []
    ntotals = []

    clade_ks = [[] for cluster in clusters]
    clade_ns = [[] for cluster in clusters]

    for gene_name in allowed_genes:
        for variant_type in passed_sites_map[gene_name].keys():
            if variant_type not in allowed_variant_types:
                continue

            allele_counts = allele_counts_map[gene_name][variant_type]['alleles']
            if len(allele_counts)==0:
                continue

            # good to go, let's get calculating

            # take consensus approximation
            genotype_matrix, passed_sites_matrix =
diversity_utils.calculate_consensus_genotypes(allele_counts)

            ktots = (genotype_matrix*passed_sites_matrix).sum(axis=1)
            ntots = (passed_sites_matrix).sum(axis=1)
            kminortots =  numpy.fmin(ktot, ntot - ktot)

            polymorphic_sites = (kminortots>0.5)

            genotype_matrix = genotype_matrix[polymorphic_sites,:]
            passed_sites_matrix = passed_sites_matrix[polymorhpic_sites,:]

            ktotals.extend(ktots)
            ntotals.extend(ntots)
```

```
            for cluster_idx in xrange(0,len(clusters)):

                cluster_idxs = clusters[cluster_idx]
                anticluster_idxs = anticlusters[cluster_idx]


                k0s =
(genotype_matrix[:,cluster_idxs]*passed_sites_matrix[:,cluster_idxs]).sum(axis=1)
                n0s = cluster_max_prevalence = (passed_sites_matrix[:,cluster_idxs]).sum(axis=1)

                clade_ks.extend(k0s)
                clade_ns.extend(n0s)

    return ktotals, ntotals, clade_ks, clade_ns

def calculate_phylogenetic_inconsistency_from_sfs(ktotals, ntotals, clade_ks, clade_ns):

    # first coarse-grain allele frequencies
    n = long(numpy.median(ntotals))
    frequency_bins = numpy.arange(1,n+1)*1.0/n
    frequency_bins = frequency_bins - (frequency_bins[1]-frequency_bins[0])/2.0
    frequency_bins[0] = -1
    frequency_bins[-1] = 2
    fs = numpy.arange(1,n)*1.0/n
    pfs = numpy.histogram(ktotals*1.0/ntotals, bins=frequency_bins)[0]
    pfs = (pfs+pfs[::-1])/2

def load_manual_clade_divergence_threshold(species_name):
    file = open(parse_midas_data.scripts_directory+"manual_clade_thresholds.txt","r")
    file.readline()
    divergence_threshold = 1e-02
    for line in file:
        items=line.split("\t")
        print items[0].strip(), species_name
        if items[0].strip()==species_name:
            divergence_threshold = float(items[1])
            print "Setting divergence threshold!", divergence_threshold
            break
    file.close()
    return divergence_threshold

def load_manual_clades(species_name):
    file = open(parse_midas_data.scripts_directory+"manual_clade_definitions.txt","r")

    line = file.readline().strip()

    # Just put some default values there in case of issue
    current_species = ""
    clades = {"":set()}

    while line!="":

        items = line.split()

        if items[0].isdigit():
            # is a sample entry
            sample_name = items[1].strip()
            clades[current_species][-1].add(sample_name)
        elif items[0].startswith('-'):
            # delimits a clade
```

```
            clades[current_species].append(set())
        else:
            # new species
            current_species = items[0].strip()
            clades[current_species] = [set()]

        line = file.readline().strip()

    file.close()

    if species_name in clades:
        return clades[species_name]
    else:
        return []

def calculate_clade_idxs_from_clade_sets(samples, clade_sets):

    clade_idxss = []
    for clade_set in clade_sets:

        clade_idxs = numpy.array([sample in clade_set for sample in samples])
        clade_idxss.append(clade_idxs)

    return clade_idxss

def permute_idxs_within_clades(cluster_idxss):

    permuted_idxs = numpy.arange(0,cluster_idxss[0].shape[0])

    for cluster_idxs in cluster_idxss:
        idxs = list(permuted_idxs[cluster_idxs])
        shuffle(idxs)
        permuted_idxs[cluster_idxs] = numpy.array(idxs)

    return permuted_idxs


############
#
# Here is another set of algorithms that does it without tree structure
#
############

###
#
# Finds SNPs.
# Calculates min distance between the two alleles (i.e., approximate "date" if asexual)
# Then calculates average and max distance within each allele.
#
###
def calculate_snp_distances(allele_counts_map, passed_sites_map, distance_matrix,
allowed_variant_types=set(['1D','2D','3D','4D']), allowed_genes=set([])):

    total_genes = set(passed_sites_map.keys())

    if len(allowed_genes)==0:
        allowed_genes = set(passed_sites_map.keys())

    allowed_genes = (allowed_genes & total_genes)

    snp_data = []
```

```
    for gene_name in allowed_genes:

        for variant_type in passed_sites_map[gene_name].keys():

            if variant_type not in allowed_variant_types:
                continue

            allele_counts = allele_counts_map[gene_name][variant_type]['alleles']
            if len(allele_counts)==0:
                continue
            locations = allele_counts_map[gene_name][variant_type]['locations']

            # good to go, let's get calculating

            # take consensus approximation
            genotype_matrix, passed_sites_matrix =
diversity_utils.calculate_consensus_genotypes(allele_counts)

            derived_samples = numpy.logical_and(genotype_matrix, passed_sites_matrix)
            ancestral_samples = numpy.logical_and(numpy.logical_not(genotype_matrix),
passed_sites_matrix)

            derived_allele_counts = derived_samples.sum(axis=1)
            ancestral_allele_counts = ancestral_samples.sum(axis=1)

            # Get SNVs to look at (no singletons)
            polymorphic_idxs =
numpy.nonzero( (derived_allele_counts>1.5)*(ancestral_allele_counts>1.5) )[0]

            for snp_idx in polymorphic_idxs:

                # Get cross snp distance
                between_distances =
distance_matrix[derived_samples[snp_idx,:],:][:,ancestral_samples[snp_idx,:]]

                min_between_d = between_distances.min()

                # Get within snp distance
                within_derived_distances =
distance_matrix[derived_samples[snp_idx,:]][:,derived_samples[snp_idx,:]]
                within_derived_distances =
within_derived_distances[numpy.triu_indices(within_derived_distances.shape[0], k = 1)]

                max_derived_d = within_derived_distances.max()
                avg_derived_d = within_derived_distances.mean()


                within_ancestral_distances =
distance_matrix[ancestral_samples[snp_idx,:],:][:,ancestral_samples[snp_idx,:]]
                within_ancestral_distances =
within_ancestral_distances[numpy.triu_indices(within_ancestral_distances.shape[0], k = 1)]

                max_ancestral_d = within_ancestral_distances.max()
                avg_ancestral_d = within_ancestral_distances.mean()


snp_data.append((locations[snp_idx],variant_type,derived_allele_counts[snp_idx],ancestral_allele
_counts[snp_idx], min_between_d, avg_derived_d, max_derived_d, avg_ancestral_d,
max_ancestral_d))
```

```
    return snp_data
```

classes.py

```
class Interval:

	include_strings = ['i', 'inclusive', 'include']
	exclude_strings = ['e', 'exclusive', 'exclude']
	valid_type_strings = include_strings + exclude_strings

	def __init__(self, *args, **kwargs):

		if len(args) == 1:
			lower, upper = args[0].split(',') # Should error out if bad
formatting
			self.lower_bound_type = 'e' if lower[0] == '(' else 'i'
			self.lower_bound = float(lower.strip()[1:])
			self.upper_bound_type = 'e' if upper[-1] == ')' else 'i'
			self.upper_bound = float(upper.strip()[:-1])
		else:
			# Otherwise, assume 4 arguments. This should error out if there
			# are an incorrect number of arguments. Tacky but yea
			lbound_type, lbound, ubound, ubound_type = args

			self.lower_bound = lbound
			self.upper_bound = ubound

			if lbound_type in self.valid_type_strings:
				self.lower_bound_type = lbound_type
			else:
				print("Invalid bound type! Defaulting to inclusive")
				self.lower_bound_type = 'inclusive'

			if ubound_type in self.valid_type_strings:
				self.upper_bound_type = ubound_type
			else:
				print("Invalid bound type! Defaulting to inclusive")
				self.upper_bound_type = 'inclusive'

	def contains(self, val):
		if (self.lower_bound_type in self.include_strings):
			lower_good = (val >= self.lower_bound)
		else:
			lower_good = (val > self.lower_bound)

		if (self.upper_bound_type in self.include_strings):
			upper_good = (val <= self.upper_bound)
		else:
			upper_good = (val < self.upper_bound)

		return (lower_good and upper_good)
```

config.py

```
##############################################################################
#
# Set up default source and output directories
#
##############################################################################
import os.path
```

```
from math import log10

data_directory = os.path.expanduser("~/mother_infant/data/")
metadata_directory = os.path.expanduser("~/mother_infant/scripts/metadata/")
analysis_directory = os.path.expanduser("~/mother_infant/analysis/")
scripts_directory = os.path.expanduser("~/mother_infant/scripts/")
patric_directory = os.path.expanduser("~/patric_db/")
midas_directory = os.path.expanduser("~/midas_db/")

# We use this one to debug because it was the first one we looked at
debug_species_name = 'Bacteroides_uniformis_57318'

good_species_min_coverage = 10
good_species_min_prevalence = 10

min_median_coverage = 20

consensus_lower_threshold = 0.2
consensus_upper_threshold = 0.8
temporal_lower_threshold = 0.35
temporal_upper_threshold = 0.65

fixation_min_change = 0.3 # Originally consensus_upper_threshold -
consensus_lower_threshold
fixation_log10_depth_ratio_threshold = log10(3)

threshold_within_between_fraction = 0.1
threshold_pi = 1e-03

min_opportunities = 100000

modification_difference_threshold = 20
replacement_difference_threshold = 500

twin_modification_difference_threshold = 1000
twin_replacement_difference_threshold = 1000

gainloss_max_absent_copynum = 0.05
gainloss_min_normal_copynum = 0.6
gainloss_max_normal_copynum = 1.2

core_genome_min_copynum = 0.3
core_genome_max_copynum = 3 # BG: should we use a maximum for "core genome"? I'm
going to go w/ yes for now
core_genome_min_prevalence = 0.9
shared_genome_min_copynum = 3

# Default parameters for pipe snps
# (Initial filtering for snps, done during postprocessing)
pipe_snps_min_samples=4
pipe_snps_min_nonzero_median_coverage=5
pipe_snps_lower_depth_factor=0.3
pipe_snps_upper_depth_factor=3

parse_snps_min_freq = 0.05

between_host_min_sample_size = 33
between_host_ld_min_sample_size = 10
within_host_min_sample_size = 3
within_host_min_haploid_sample_size = 10
between_low_divergence_threshold = 2e-04
```

core_gene_utils.py

```
import numpy, os.path, gzip
import config, parse_midas_data

# ============================================================================
# This module (core_gene_utils) contains the following utilities:
#
#      parse_core_genes
#      parse_shared_genes
#      parse_non_shared_reference_genes
#      parse_non_shared_pangenome_genes
#      get_good_pangenome_samples
#      parse_gene_freqs
#
# ============================================================================

# Prevalence cohorts: adult, hmp, infant
# where all is union of core genes from the other three

def get_filename(type, prev_cohort, external=False, species=None):

       if external:
              core_genes_directory = "%s/core_genes/external/" %
(config.data_directory)
       else:
              core_genes_directory = "%s/core_genes/" % (config.data_directory)

       prev_cohort_directory = '%s/%s/' % (core_genes_directory, prev_cohort)

       # Type must be one of shared_genes, core_genes, core_genes_stringent, or
gene_freqs
       # default to core
       if type == 'gene_freqs' and species != None:
              return '%s/prev_%s/%s_%s.txt.gz' % (prev_cohort_directory,
prev_cohort, species, type)
       elif type in ['shared_genes', 'core_genes', 'core_genes_stringent']:
              return '%s/%s.txt.gz' % (prev_cohort_directory, type)
       else:
              raise ValueError("Bad arguments for core genes filename")

# ============================================================================
# Returns set of core genes for specified set of species
# ============================================================================

def parse_core_genes(desired_species_name = None, prev_cohort='all',
external_filtering=True):

              core_genes = set() # Core genes for the specified species

              if prev_cohort == 'all':
                     # Take union of core genes from adult, hmp, infant
                     for sub_prev_cohort in ['adult', 'hmp', 'infant']:
                            core_gene_filename = get_filename('core_genes',
sub_prev_cohort)
                            core_gene_file = gzip.GzipFile(core_gene_filename,"r")
                            for line in core_gene_file:
                                   items = line.split(':')
                                   if len(items)<2:
                                          continue
```

```
                                species_name = items[0].strip()
                                gene_names = [subitem.strip() for subitem in
items[1].split(",")]
                                if (species_name == desired_species_name) or
(desired_species_name == None):
                                        core_genes.update(gene_names)

                        core_gene_file.close()

                else:
                        # Otherwise only consider specified prevalence cohort
                        core_gene_filename = get_filename('core_genes', prev_cohort)
                        core_gene_file = gzip.GzipFile(core_gene_filename,"r")
                        for line in core_gene_file:
                                items = line.split(":")
                                if len(items)<2:
                                        continue
                                species_name = items[0].strip()
                                gene_names = [subitem.strip() for subitem in
items[1].split(",")]
                                if (species_name == desired_species_name) or
(desired_species_name == None):
                                        core_genes.update(gene_names)

                        core_gene_file.close()

                # Account for externally provided core genes if available
                external_core_gene_filename = get_filename('core_genes', prev_cohort,
external=True)
                external_core_genes = set()

                if os.path.isfile(external_core_gene_filename):

                        external_core_gene_file =
gzip.GzipFile(external_core_gene_filename,"r")
                        for line in external_core_gene_file:

                                items = line.split(":")
                                if len(items)<2:
                                        continue

                                species_name = items[0].strip()
                                gene_names = [subitem.strip() for subitem
in items[1].split(",")]
                                if (species_name == desired_species_name)
or (desired_species_name == None):

        external_core_genes.update(gene_names)
                        external_core_gene_file.close()

                if external_filtering and len(external_core_genes)>0:
                        core_genes = (core_genes & external_core_genes)

                return core_genes

# ============================================================================
# Returns set of shared genes for specified set of species
# ============================================================================

def parse_shared_genes(desired_species_name = None, prev_cohort='all',
external_filtering = True):
```

```
            shared_genes = set()

            if prev_cohort == 'all':

                  # Take union of core genes from adult, hmp, infant
                  for sub_prev_cohort in ['adult', 'hmp', 'infant']:

                        shared_gene_filename = get_filename('shared_genes',
sub_prev_cohort)
                        shared_gene_file =
gzip.GzipFile(shared_gene_filename,"r")

                        for line in shared_gene_file:
                              items = line.split(":")

                              if len(items)<2:
                                    continue

                              species_name = items[0].strip()
                              gene_names_str = items[1].strip()

                              # N/A means wasn't enough pangenome data to detect
shared genes
                              gene_names = [] if
gene_names_str.startswith('N/A') else [subitem.strip() for subitem in
gene_names_str.split(",")]

                              if (species_name==desired_species_name) or
(desired_species_name==""):
                                          shared_genes.update(gene_names)

                        shared_gene_file.close()

            else:
                  # Otherwise only consider specified prevalence cohort
                  shared_gene_filename = get_filename('shared_genes',
prev_cohort)
                  shared_gene_file = gzip.GzipFile(shared_gene_filename,"r")

                  for line in shared_gene_file:
                        items = line.split(":")

                        if len(items)<2:
                              continue

                        species_name = items[0].strip()
                        gene_names_str = items[1].strip()

                        # N/A means wasn't enough pangenome data to detect
shared genes
                        gene_names = [] if gene_names_str.startswith('N/A') else
[subitem.strip() for subitem in gene_names_str.split(",")]

                        if (species_name==desired_species_name) or
(desired_species_name==""):
                              shared_genes.update(gene_names)

                  shared_gene_file.close()
```

```
            external_shared_gene_filename = get_filename('shared_genes',
prev_cohort, external=True)
            external_shared_genes = set()

            if os.path.isfile(external_shared_gene_filename):

                        external_shared_gene_file =
gzip.GzipFile(external_shared_gene_filename,"r")

                        for line in external_shared_gene_file:

                                    items = line.split(":")
                                    if len(items)<2:
                                                continue

                                    species_name = items[0].strip()
                                    gene_names_str = items[1].strip()
                                    if gene_names_str.startswith('N/A'): #
Wasn't enough pangenome data to detect shared genes
                                                gene_names = []
                                    else:
                                                gene_names = [subitem.strip()
for subitem in gene_names_str.split(",")]

                                    if (species_name==desired_species_name) or
(desired_species_name==""):

      external_shared_genes.update(gene_names)

                        external_shared_gene_file.close()

            if external_filtering and len(external_shared_genes)>0:
                        # some externally provided core genes
                        shared_genes = (shared_genes | external_shared_genes)

            return shared_genes

# ============================================================================
# Returns set of reference genes which are not shared
# ============================================================================

def parse_non_shared_reference_genes(desired_species_name="", prev_cohort='all',
external_filtering=True):

            from utils import parse_midas_data

            shared_genes = parse_shared_genes(desired_species_name, prev_cohort,
external_filtering)
            reference_genes =
parse_midas_data.load_reference_genes(desired_species_name)
            non_shared_reference_genes = set(reference_genes)-shared_genes

            return non_shared_reference_genes

# ============================================================================
# Returns set of pangenome genes which are not shared
# ============================================================================

def parse_non_shared_pangenome_genes(desired_species_name="", prev_cohort='all',
external_filtering=True):
```

```
            from utils import parse_midas_data

            shared_genes = parse_shared_genes(desired_species_name, prev_cohort,
external_filtering)
            pangenome_genes, pangenome_centroid_genes =
parse_midas_data.load_pangenome_genes(desired_species_name)
            # TODO: Not sure if I should be using the first or second
            non_shared_pangenome_genes = set(pangenome_centroid_genes)-
shared_genes
            return non_shared_pangenome_genes

# ====================================================================
# Returns indices for samples which have enough present genes (copy number
# exceeds a low threshold) of which not too many are high-copynum
# ====================================================================

def get_good_pangenome_samples(marker_coverages, gene_copynum_matrix,
species_name):

            cmin = config.core_genome_min_copynum
            cmax = config.core_genome_max_copynum

            # For each sample, get number of genes which are "present" (copynum >
0.3)
            # and which are "high" (copynum > 3)
            num_present_genes = (gene_copynum_matrix>cmin).sum(axis=0)
            num_high_genes = (gene_copynum_matrix>cmax).sum(axis=0)

            # Get number of reference genes
            # TODO: does not match number of genes in copynum matrix
            num_reference_genes =
len(parse_midas_data.load_reference_genes(species_name))

            # Want at least 30% of all reference genes to be present, and want no
more
            # than 30% of present genes to be high
            min_present_genes = 0.3*num_reference_genes
            max_high_genes = 0.3*num_present_genes

            good_sample_idxs =
(num_present_genes>min_present_genes)*(num_high_genes<max_high_genes)

            return good_sample_idxs

# ====================================================================
# Returns gene frequency map (gene name -> prevalence)
# ====================================================================

def parse_gene_freqs(desired_species_name, prev_cohort='hmp', use_external=False):

            filename = get_filename('gene_freqs', prev_cohort,
external=use_external, species=desired_species_name)

            if not os.path.isfile(filename):
                        return {}

            file = gzip.open(filename,"r")
            gene_freq_map = {}
            for line in file:
                        items = line.split()
                        gene_name = items[0]
```

```
                        f = float(items[1])
                        gene_freq_map[gene_name] = f
            file.close()

            return gene_freq_map
```

diversity_utils.py

```
import numpy
from scipy.linalg import eigh
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import cophenet
from scipy.cluster.hierarchy import fcluster
from numpy.random import shuffle, normal
import scipy.stats
from scipy.stats import binom
import config
from scipy.special import betainc
import sys
import parse_midas_data
import sample_utils
import sample_utils
import stats_utils
import os.path
import sfs_utils

# ====================================================================
# Calls consensus genotypes from matrix of allele counts
#
# Returns: genotype matrix, passed_sitse matrix for polymorphic sites
# ====================================================================

def calculate_consensus_genotypes(allele_counts_matrix,lower_threshold=0.2,upper_threshold=0.8):

            num_sites, num_samples, num_alleles = allele_counts_matrix.shape

            depths = allele_counts_matrix.sum(axis=2)
            freqs = allele_counts_matrix[:,:,0]*1.0/(depths+(depths==0))
            passed_sites_matrix =
(depths>0)*numpy.logical_or(freqs<=lower_threshold,freqs>=upper_threshold)
            # consensus approximation
            genotype_matrix = numpy.around(freqs)*passed_sites_matrix


            return genotype_matrix, passed_sites_matrix


def
calculate_consensus_polymorphic_genotypes(allele_counts_matrix,lower_threshold=0.2,upper_thresho
ld=0.8):

            genotype_matrix, passed_sites_matrix =
      calculate_consensus_genotypes(allele_counts_matrix,lower_threshold,upper_threshold)

            prevalences = (genotype_matrix*passed_sites_matrix).sum(axis=1)
            min_prevalences = 0.5
            max_prevalences = (passed_sites_matrix).sum(axis=1)-0.5

            polymorphic_sites = (prevalences>min_prevalences)*(prevalences<max_prevalences)
```

```
            return genotype_matrix[polymorphic_sites,:],
passed_sites_matrix[polymorphic_sites,:]

# ========================================================================
# Calculates first two PCA coordinates for samples in allele_counts
# using the normalization scheme outlined in McVean (PLoS Genet, 2009).
#
# Returns: (vector of pca1 coords, vector of pca2 coords), (percent variance 1, percent variance
2)
# ========================================================================

def calculate_pca_coordinates(genotype_matrix, passed_sites_matrix):

            Zl =
(genotype_matrix*passed_sites_matrix).sum(axis=1)/(passed_sites_matrix).sum(axis=1)

            Zli = (genotype_matrix-Zl[:,None])*passed_sites_matrix

            Mij = numpy.einsum('li,lj',Zli,Zli)/numpy.einsum('li,lj',passed_sites_matrix,
passed_sites_matrix)

            # calculate eigenvectors & eigenvalues of the covariance matrix
            # use 'eigh' rather than 'eig' since R is symmetric,
            # the performance gain is substantial
            evals, evecs = eigh(Mij)

            # sort eigenvalue in decreasing order
            idx = numpy.argsort(evals)[::-1]
            evals = evals[idx]
            evecs = evecs[:,idx]

            variances = evals/evals.sum()

            pca1_coords = evals[0]**0.5*evecs[:,0]
            pca2_coords = evals[1]**0.5*evecs[:,1]

            return (pca1_coords, pca2_coords), (variances[0],variances[1])



def calculate_rsquared_condition_freq(allele_counts_1, allele_counts_2, low_freq, high_freq):
            # Note: should actually be sigma_squared!
            # sigma_squared= E[X]/E[Y], where X=(p_ab-pa*pb)^2 and Y=(pa*(1-pa)*pb*(1-pb))
            # rsquared=E[X/Y]
            # see McVean 2002 for more notes on the difference.

            # allele counts = 1 x samples x alleles vector

            depths_1 = allele_counts_1.sum(axis=2)
            freqs_1 = allele_counts_1[:,:,0]*1.0/(depths_1+(depths_1==0))
            depths_2 = allele_counts_2.sum(axis=2)
            freqs_2 = allele_counts_2[:,:,0]*1.0/(depths_2+(depths_2==0))


            # consensus approximation
            freqs_1 = numpy.around(freqs_1)
            freqs_2 = numpy.around(freqs_2)

            # condition on allele frequency in the pooled population:
            pooled_freqs_1=freqs_1[:,:].sum(axis=1)/len(freqs_1[0])
            pooled_freqs_2=freqs_2[:,:].sum(axis=1)/len(freqs_2[0])
```

```
            # check if any freqs >0.5, if so, fold:
            pooled_freqs_1=numpy.where(pooled_freqs_1 > 0.5, 1-pooled_freqs_1, pooled_freqs_1)
            pooled_freqs_2=numpy.where(pooled_freqs_2 > 0.5, 1-pooled_freqs_2, pooled_freqs_2)

            # this asks which pairs of sites have depths >0 at BOTH sites as well as which
paris of sites both have pooled frequencies within the low_freq and high_freq ranges.
            # None here takes the product of the elements in the two vectors and returns a
matrix.


            passed_sites_1=(depths_1>0)*(pooled_freqs_1 >= low_freq)[:,None]*(pooled_freqs_1
<=high_freq)[:,None]
            passed_sites_2=(depths_2>0)*(pooled_freqs_2 >= low_freq)[:,None]*(pooled_freqs_2 <=
high_freq)[:,None]
            joint_passed_sites=passed_sites_1[None,:,:]*passed_sites_2[:,None,:]
            # sites x sites x samples matrix

            joint_freqs = freqs_1[None,:,:]*freqs_2[:,None,:]
            # sites x sites x samples_matrix

            # this tells us what the denominator is for the computation below for
joint_pooled_freqs
            total_joint_passed_sites = joint_passed_sites.sum(axis=2)
            # add 1 to denominator if some pair is 0.
            total_joint_passed_sites = total_joint_passed_sites+(total_joint_passed_sites==0)

            # compute p_ab
            joint_pooled_freqs =
(joint_freqs*joint_passed_sites).sum(axis=2)/total_joint_passed_sites
            # floting point issue
            joint_pooled_freqs *= (joint_pooled_freqs>1e-10)

            # compute p_a
            marginal_pooled_freqs_1 =
(freqs_1[None,:,:]*joint_passed_sites).sum(axis=2)/total_joint_passed_sites
            marginal_pooled_freqs_1 *= (marginal_pooled_freqs_1>1e-10)

            # compute p_b
            marginal_pooled_freqs_2 =
(freqs_2[:,None,:]*joint_passed_sites).sum(axis=2)/total_joint_passed_sites
            marginal_pooled_freqs_2 *= (marginal_pooled_freqs_2>1e-10)

            # (p_ab-p_a*p_b)^2
            rsquared_numerators = numpy.square(joint_pooled_freqs-
marginal_pooled_freqs_1*marginal_pooled_freqs_2)

            # (p_a*(1-p_a)*pb*(1-p_b))
            rsquared_denominators = marginal_pooled_freqs_1*(1-
marginal_pooled_freqs_1)*marginal_pooled_freqs_2*(1-marginal_pooled_freqs_2)


            rsquareds = rsquared_numerators/(rsquared_denominators+(rsquared_denominators==0))

            return rsquared_numerators, rsquared_denominators


#######################################################################


#######################################################################
```

```
def calculate_sigmasquared(allele_counts_1, allele_counts_2):
            # A standard measure of linkage disequilibrium:
            #
            # sigma_squared= E[X]/E[Y], where X=(p_ab-pa*pb)^2 and Y=(pa*(1-pa)*pb*(1-pb))
            # rsquared=E[X/Y]
            # see McVean 2002 for more notes on the difference.

            # allele counts = 1 x samples x alleles vector

            freqs_1, passed_sites_1 = calculate_consensus_genotypes(allele_counts_1)
            freqs_2, passed_sites_2 = calculate_consensus_genotypes(allele_counts_2)

            # this asks which pairs of sites have depths >0 at BOTH sites
            # None here takes the product of the elements in the two vectors and returns a
matrix.
            joint_passed_sites = (passed_sites_1)[None,:,:]*(passed_sites_2)[:,None,:]
            # sites x sites x samples matrix

            joint_freqs = freqs_1[None,:,:]*freqs_2[:,None,:]
            # sites x sites x samples_matrix

            # this tells us what the denominator is for the computation below for
joint_pooled_freqs
            total_joint_passed_sites = joint_passed_sites.sum(axis=2)
            # add 1 to denominator if some pair is 0.
            total_joint_passed_sites = total_joint_passed_sites+(total_joint_passed_sites==0)

            # compute p_ab
            joint_pooled_freqs =
(joint_freqs*joint_passed_sites).sum(axis=2)/total_joint_passed_sites
            # floting point issue
            joint_pooled_freqs *= (joint_pooled_freqs>1e-10)

            # compute p_a
            marginal_pooled_freqs_1 =
(freqs_1[None,:,:]*joint_passed_sites).sum(axis=2)/total_joint_passed_sites
            marginal_pooled_freqs_1 *= (marginal_pooled_freqs_1>1e-10)

            # compute p_b
            marginal_pooled_freqs_2 =
(freqs_2[:,None,:]*joint_passed_sites).sum(axis=2)/total_joint_passed_sites
            marginal_pooled_freqs_2 *= (marginal_pooled_freqs_2>1e-10)

            # (p_ab-p_a*p_b)^2
            rsquared_numerators = numpy.square(joint_pooled_freqs-
marginal_pooled_freqs_1*marginal_pooled_freqs_2)

            # (p_a*(1-p_a)*pb*(1-p_b))
            rsquared_denominators = marginal_pooled_freqs_1*(1-
marginal_pooled_freqs_1)*marginal_pooled_freqs_2*(1-marginal_pooled_freqs_2)

            rsquareds = rsquared_numerators/(rsquared_denominators+(rsquared_denominators==0))

            return rsquared_numerators, rsquared_denominators


#########################################################################


#########################################################################
def calculate_unbiased_sigmasquared(allele_counts_1, allele_counts_2):
```

```
            # An alternate version of a standard measure of linkage disequilibrium:
            #
            # sigma_squared= E[X]/E[Y], where X=(p_ab-pa*pb)^2 and Y=(pa*(1-pa)*pb*(1-pb))
            # rsquared=E[X/Y]
            # see McVean 2002 for more notes on the difference.
            #
            # where we have corrected for finite sample effects

            genotypes_1, passed_sites_1 = calculate_consensus_genotypes(allele_counts_1)
            genotypes_2, passed_sites_2 = calculate_consensus_genotypes(allele_counts_2)


            # this asks which pairs of sites have depths >0 at BOTH sites
            # None here takes the product of the elements in the two vectors and returns a
matrix.
            joint_passed_sites = (passed_sites_1)[None,:,:]*(passed_sites_2)[:,None,:]
            # sites x sites x samples matrix

            # allele counts
            ns = joint_passed_sites.sum(axis=2)

            n11s =
((genotypes_1[None,:,:])*(genotypes_2[:,None,:])*joint_passed_sites).sum(axis=2)
            n10s = (genotypes_1[None,:,:]*(1-
genotypes_2[:,None,:])*joint_passed_sites).sum(axis=2)
            n01s = ((1-
genotypes_1[None,:,:])*(genotypes_2[:,None,:])*joint_passed_sites).sum(axis=2)
            n00s = ((1-genotypes_1[None,:,:])*(1-
genotypes_2[:,None,:])*joint_passed_sites).sum(axis=2)

            #print "Gene:"
            #print n11s
            #print n10s
            #print n01s
            #print n00s
            #print "--"


            # First calculate numerator
            rsquared_numerators = n11s*(n11s-1)*n00s*(n00s-1)
            rsquared_numerators -= 2*n10s*n01s*n11s*n00s
            rsquared_numerators += n10s*(n10s-1)*n01s*(n01s-1)

            #print "Before divide:"
            #print rsquared_numerators

            rsquared_numerators = rsquared_numerators*(ns>3.5)*1.0/(ns*(ns-1)*(ns-2)*(ns-
3)+10*(ns<3.5))

            #print "After divide:"
            #print rsquared_numerators
            #print "---"
            # Now calculate denominator
            # (more annoying... there are 16 terms rather than 4, so we will write them
separately)

            #1
            rsquared_denominators = n10s*(n10s-1)*n01s*(n01s-1)
            #2
            rsquared_denominators += n10s*n01s*(n01s-1)*n00s
            #3
```

```
            rsquared_denominators += n10s*(n10s-1)*n01s*n11s
            #4
            rsquared_denominators += n10s*n01s*n11s*n00s
            #5
            rsquared_denominators += n10s*(n10s-1)*n01s*n00s
            #6
            rsquared_denominators += n10s*n01s*n00s*(n00s-1)
            #7
            rsquared_denominators += n10s*(n10s-1)*n11s*n00s
            #8
            rsquared_denominators += n10s*n11s*n00s*(n00s-1)
            #9
            rsquared_denominators += n10s*n01s*(n01s-1)*n11s
            #10
            rsquared_denominators += n01s*(n01s-1)*n11s*n00s
            #11
            rsquared_denominators += n10s*n01s*n11s*(n11s-1)
            #12
            rsquared_denominators += n01s*n11s*(n11s-1)*n00s
            #13
            rsquared_denominators += n10s*n01s*n11s*n00s
            #14
            rsquared_denominators += n01s*n11s*n00s*(n00s-1)
            #15
            rsquared_denominators += n10s*n11s*(n11s-1)*n00s
            #16
            rsquared_denominators += n11s*(n11s-1)*n00s*(n00s-1)

            # divide by sample size
            rsquared_denominators = rsquared_denominators*(ns>3.5)*1.0/(ns*(ns-1)*(ns-2)*(ns-
3)+10*(ns<3.5))

            return rsquared_numerators, rsquared_denominators




###################################
def generate_haplotype(allele_counts_4D, allele_counts_1D, location_dictionary, species_name):

            freqs={}
            depths={}

            depths['4D'] = allele_counts_4D.sum(axis=2)
            freqs['4D'] = allele_counts_4D[:,:,0]*1.0/(depths['4D']+(depths['4D']==0))

            depths['1D'] = allele_counts_1D.sum(axis=2)
            freqs['1D'] = allele_counts_1D[:,:,0]*1.0/(depths['1D']+(depths['1D']==0))

            #explanation of numpy commands above:
            # allele_counts_1.sum(axis=2) this returns a sum over all sites alt + ref counts.
            #(depths_1+(depths_1==0) this is done because if depths_1==0, then we've have a
division error. addition of 1 when depths_1==0.
            #allele_counts_1[:,:,0] means that the alt allele is grabbed. Multiply by 1.0 to
convert to float

            # consensus approximation
            consensus={}
            consensus['4D'] = numpy.around(freqs['4D'])
            consensus['1D'] = numpy.around(freqs['1D'])
```

```
            locations=location_dictionary.keys()
            locations=sorted(locations)

            #s_consensus='' # store the haplotypes in a string for printing out later
            #s_annotation=''

      outFile_consensus=open(os.path.expanduser('~/tmp_intermediate_files/tmp_consensus_%s.txt'
) % species_name ,'w')
            outFile_anno=open(os.path.expanduser('~/tmp_intermediate_files/tmp_anno_%s.txt') %
species_name ,'w')

            for loc in range(0, len(locations)):
                        location=str(int(locations[loc]))
                        index=location_dictionary[locations[loc]][0]
                        variant_type=location_dictionary[locations[loc]][1]
                        alleles=consensus[variant_type][index].tolist()
                        annotation=freqs[variant_type][index].tolist()
                        coverage=depths[variant_type][index].tolist() # if coverage ==0, then
set to 'N' in both the consensus and annotation files.

                        for person in range(0, len(alleles)):
                                    alleles[person]=str(int(alleles[person]))
                                    if coverage[person] == 0.0:
                                                alleles[person]='N'
                                                annotation[person]='N'
                                    else:
                                                if annotation[person] ==0:
                                                            annotation[person]=str(0) # no
difference from ref
                                                elif annotation[person] ==1:
                                                            if variant_type=='4D':

      annotation[person]=str(1) # fixed syn diff from ref
                                                            else:

      annotation[person]=str(2) # fixed nonsyn diff from ref
                                                else:
                                                            if variant_type=='4D':

      annotation[person]=str(3) # polymorphic syn within host
                                                            else:

      annotation[person]=str(4) # polymorphic nonsyn within host
                        s_consensus = location + ',' + ','.join(alleles) +'\n'
                        s_annotation = location + ',' + ','.join(annotation) + '\n'
                        outFile_consensus.write(s_consensus)
                        outFile_anno.write(s_annotation)

###################################

def calculate_sample_freqs(allele_counts_map, passed_sites_map, variant_type='4D',
allowed_genes=None, fold=True):

            if allowed_genes == None:
                        allowed_genes = set(passed_sites_map.keys())

            sample_freqs = [[] for i in
xrange(0,allele_counts_map[allele_counts_map.keys()[0]][variant_type]['alleles'].shape[1])]
```

```
            passed_sites =
numpy.zeros(passed_sites_map[passed_sites_map.keys()[0]][variant_type]['sites'].shape[0])*1.0

            for gene_name in allowed_genes:

                        allele_counts = allele_counts_map[gene_name][variant_type]['alleles']

                        if len(allele_counts)==0:
                                    continue

                        depths = allele_counts.sum(axis=2)
                        freqs = allele_counts[:,:,0]/(depths+(depths==0))
                        if fold == True:
                                    freqs = numpy.fmin(freqs,1-freqs) #fold
                        for sample_idx in xrange(0,freqs.shape[1]):
                                    gene_freqs = freqs[:,sample_idx]

      sample_freqs[sample_idx].extend( gene_freqs[gene_freqs>0])

                        passed_sites +=
numpy.diagonal(passed_sites_map[gene_name][variant_type]['sites'])


            return sample_freqs, passed_sites




####################################


def calculate_temporal_sample_freqs(allele_counts_map, passed_sites_map, initial_sample_idx,
final_sample_idx, allowed_variant_types=set(['1D','2D','3D','4D']), allowed_genes=None):

            desired_samples = numpy.array([initial_sample_idx, final_sample_idx])

            initial_freqs = []
            final_freqs = []
            gene_names = []
            chromosomes = []
            positions = []
            marginal_initial_depths = []
            marginal_final_depths = []


            if allowed_genes == None:
                        allowed_genes = set(passed_sites_map.keys())


            for gene_name in allowed_genes:
                        for variant_type in allele_counts_map[gene_name].keys():

                                    if variant_type not in allowed_variant_types:
                                                continue

                                    allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

                                    if len(allele_counts)==0:
                                                continue
```

```
                                        chunk_chromosomes = numpy.array([chromosome for
chromosome, position in allele_counts_map[gene_name][variant_type]['locations']])

                                        chunk_positions = numpy.array([position for chromosome,
position in allele_counts_map[gene_name][variant_type]['locations']])

                                        allele_counts = allele_counts[:,desired_samples,:]

                                        depths = allele_counts.sum(axis=2)
                                        freqs = allele_counts[:,:,0]*1.0/(depths+(depths==0))

                                        initial_depths = depths[:,0]
                                        final_depths = depths[:,1]

                                        marginal_passed_sites =
numpy.logical_or((initial_depths>0), (final_depths>0))
                                        joint_passed_sites =
numpy.logical_and((initial_depths>0), (final_depths>0))

                                        initial_depths = initial_depths[marginal_passed_sites]
                                        final_depths = final_depths[marginal_passed_sites]

                                        unpolarized_initial_freqs =
freqs[marginal_passed_sites,0]
                                        unpolarized_final_freqs = freqs[marginal_passed_sites,1]

                                        unpolarized_dfs = unpolarized_final_freqs-
unpolarized_initial_freqs+normal(0,1e-06,unpolarized_initial_freqs.shape)

                                        polarized_initial_freqs = unpolarized_initial_freqs +
(1-2*unpolarized_initial_freqs)*(unpolarized_dfs<=0)
                                        polarized_final_freqs = unpolarized_final_freqs + (1-
2*unpolarized_final_freqs)*(unpolarized_dfs<=0)


                                        #polarized_initial_freqs = unpolarized_initial_freqs +
(1-2*unpolarized_initial_freqs)*(unpolarized_initial_freqs>0.5)
                                        #polarized_final_freqs = unpolarized_final_freqs + (1-
2*unpolarized_final_freqs)*(unpolarized_initial_freqs>0.5)

                                        initial_freqs.extend(polarized_initial_freqs)
                                        final_freqs.extend(polarized_final_freqs)

                                        marginal_initial_depths.extend(initial_depths)
                                        marginal_final_depths.extend(final_depths)


      gene_names.extend([gene_name]*len(polarized_final_freqs))

      chromosomes.extend(chunk_chromosomes[marginal_passed_sites])
                                        positions.extend(chunk_positions[marginal_passed_sites])

            print len(gene_names), len(chromosomes), len(initial_freqs), len(initial_depths)

            return numpy.array(gene_names), numpy.array(chromosomes), numpy.array(positions),
numpy.array(initial_freqs), numpy.array(final_freqs), numpy.array(marginal_initial_depths),
numpy.array(marginal_final_depths)

def calculate_triplet_sample_freqs(allele_counts_map, passed_sites_map, i, j, k,
allowed_variant_types=set(['1D','2D','3D','4D']), allowed_genes=None):
```

```
		desired_samples = numpy.array([i, j, k])

		initial_freqs = []
		middle_freqs = []
		final_freqs = []
		gene_names = []
		chromosomes = []
		positions = []

		if allowed_genes == None:
				allowed_genes = set(passed_sites_map.keys())


		for gene_name in allowed_genes:
				for variant_type in allele_counts_map[gene_name].keys():

						if variant_type not in allowed_variant_types:
								continue

						allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

						if len(allele_counts)==0:
								continue

						chunk_chromosomes = numpy.array([chromosome for
chromosome, position in allele_counts_map[gene_name][variant_type]['locations']])

						chunk_positions = numpy.array([position for chromosome,
position in allele_counts_map[gene_name][variant_type]['locations']])

						allele_counts = allele_counts[:,desired_samples,:]

						depths = allele_counts.sum(axis=2)
						freqs = allele_counts[:,:,0]*1.0/(depths+(depths==0))

						initial_depths = depths[:,0]
						middle_depths = depths[:,1]
						final_depths = depths[:,2]

						joint_passed_sites =
(initial_depths>0)*(middle_depths>0)*(final_depths>0)

						unpolarized_initial_freqs = freqs[joint_passed_sites,0]
						unpolarized_middle_freqs = freqs[joint_passed_sites,1]
						unpolarized_final_freqs = freqs[joint_passed_sites,2]

						# polarize sites
						flipped_sites = (unpolarized_initial_freqs>0.5)

						polarized_initial_freqs = unpolarized_initial_freqs +
(1-2*unpolarized_initial_freqs)*(flipped_sites)
						polarized_middle_freqs = unpolarized_middle_freqs + (1-
2*unpolarized_middle_freqs)*(flipped_sites)
						polarized_final_freqs = unpolarized_final_freqs + (1-
2*unpolarized_final_freqs)*(flipped_sites)

						# changed sites
						changed_sites =
(polarized_initial_freqs<=0.2)*numpy.logical_or(polarized_final_freqs>=0.8,
polarized_middle_freqs>=0.8)
```

```
                                        if changed_sites.sum() > 0:


        initial_freqs.extend(polarized_initial_freqs[changed_sites])

        middle_freqs.extend(polarized_middle_freqs[changed_sites])

        final_freqs.extend(polarized_final_freqs[changed_sites])

        gene_names.extend( long(changed_sites.sum()) * [gene_name] )


                freqs = []
                for f0,f1,f2 in zip(initial_freqs, middle_freqs, final_freqs):
                        freqs.append((f0,f1,f2))

                return freqs


####################

def calculate_sample_freqs_2D(allele_counts_map, passed_sites_map, desired_samples,
variant_type='4D', allowed_genes=None, fold=True):


                if allowed_genes == None:
                        allowed_genes = set(passed_sites_map.keys())

                num_samples=sum(desired_samples)
                sample_freqs = [[] for i in xrange(0, num_samples)]
                joint_passed_sites= [[] for i in xrange(0, num_samples)]
                passed_sites = numpy.zeros((num_samples, num_samples))*1.0


                for gene_name in allowed_genes:

                        allele_counts = allele_counts_map[gene_name][variant_type]['alleles']

                        if len(allele_counts)==0:
                                continue

                        allele_counts = allele_counts[:,desired_samples,:]

                        depths = allele_counts.sum(axis=2)
                        freqs = allele_counts[:,:,0]*1.0/(depths+(depths==0))
                        joint_passed_sites_tmp=(depths>0)[:,None,:]*(depths>0)[:,:,None]

                        if fold== True:
                                freqs = numpy.fmin(freqs,1-freqs)

                        for sample_idx in xrange(0,freqs.shape[1]):
                                gene_freqs = freqs[:,sample_idx]
                                sample_freqs[sample_idx].extend(gene_freqs)

        joint_passed_sites[sample_idx].extend(joint_passed_sites_tmp[:,0,sample_idx])
                                idx=numpy.where(desired_samples==True)
                        passed_sites +=
passed_sites_map[gene_name][variant_type]['sites'][:,idx[0]][idx[0],:]

                return sample_freqs, passed_sites, joint_passed_sites
```

```
####################


def calculate_pooled_freqs(allele_counts_map, passed_sites_map, allowed_sample_idxs=[],
allowed_variant_types = set(['1D','2D','3D','4D']), allowed_genes=set([]),
lower_threshold=0.2,upper_threshold=0.8):

            if len(allowed_sample_idxs)==0:
                        # all samples are allowed
                        allowed_sample_idxs = numpy.array([True for i in
xrange(0,allele_counts_map.values()[0].values()[0]['alleles'].shape[1])])

            if len(allowed_genes)==0:
                        allowed_genes = set(passed_sites_map.keys())
            allowed_genes = allowed_genes & set(passed_sites_map.keys())

            pooled_freqs = []

            for gene_name in allowed_genes:

                        for variant_type in allele_counts_map[gene_name].keys():

                                    if variant_type not in allowed_variant_types:
                                                continue


                                    allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

                                    if len(allele_counts)==0:
                                                continue

                                    #print
allele_counts_map[gene_name][variant_type]['alleles'].shape, allowed_sample_idxs.shape

                                    allele_counts = allele_counts[:,allowed_sample_idxs,:]

                                    genotype_matrix, passed_sites_matrix =
calculate_consensus_genotypes(allele_counts,lower_threshold,upper_threshold)
                                    prevalences =
(genotype_matrix*passed_sites_matrix).sum(axis=1)
                                    min_prevalences = 0.5
                                    max_prevalences = (passed_sites_matrix).sum(axis=1)-0.5

                                    polymorphic_sites =
(prevalences>min_prevalences)*(prevalences<max_prevalences)

                                    gene_pooled_freqs =
prevalences*1.0/(passed_sites_matrix).sum(axis=1)
                                    gene_pooled_freqs = gene_pooled_freqs[polymorphic_sites]
                                    gene_pooled_freqs = numpy.fmin(gene_pooled_freqs,1-
gene_pooled_freqs)
                                    pooled_freqs.extend(gene_pooled_freqs)

            pooled_freqs = numpy.array(pooled_freqs)
            return pooled_freqs

def calculate_pooled_counts(allele_counts_map, passed_sites_map, allowed_sample_idxs=[],
allowed_variant_types = set(['1D','2D','3D','4D']),
allowed_genes=set([]),pi_min_k=1,lower_threshold=0.2,upper_threshold=0.8):
```

```
            if len(allowed_sample_idxs)==0:
                        # all samples are allowed
                        allowed_sample_idxs = numpy.array([True for i in
xrange(0,allele_counts_map.values()[0].values()[0]['alleles'].shape[1])])

            if len(allowed_genes)==0:
                        allowed_genes = set(passed_sites_map.keys())
            allowed_genes = allowed_genes & set(passed_sites_map.keys())

            pi_weighted_number = 0
            pooled_counts = []

            for gene_name in allowed_genes:

                        for variant_type in allele_counts_map[gene_name].keys():

                                    if variant_type not in allowed_variant_types:
                                                continue

                                    allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

                                    if len(allele_counts)==0:
                                                continue

                                    #print
allele_counts_map[gene_name][variant_type]['alleles'].shape, allowed_sample_idxs.shape

                                    allele_counts = allele_counts[:,allowed_sample_idxs,:]

                                    genotype_matrix, passed_sites_matrix =
calculate_consensus_genotypes(allele_counts,lower_threshold,upper_threshold)
                                    prevalences =
(genotype_matrix*passed_sites_matrix).sum(axis=1)
                                    min_prevalences = 0.5
                                    max_prevalences = (passed_sites_matrix).sum(axis=1)-0.5

                                    polymorphic_sites =
(prevalences>min_prevalences)*(prevalences<max_prevalences)

                                    ks = prevalences[polymorphic_sites]
                                    ns = passed_sites_matrix.sum(axis=1)[polymorphic_sites]
                                    minor_ks = numpy.fmin(ks,ns-ks)
                                    pooled_counts.extend( minor_ks )

                                    pi_weighted_number += (ks*(ns-ks)*2.0/(ns*(ns-
1))*(minor_ks>=pi_min_k)).sum()

            pooled_counts = numpy.array(pooled_counts)
            return pooled_counts, pi_weighted_number

def calculate_private_snvs(samples, allele_counts_map, passed_sites_map,
allowed_variant_types=set([]), allowed_genes=set([]),
lower_threshold=config.consensus_lower_threshold,
upper_threshold=config.consensus_upper_threshold):

            # First
            sample_host_matrix, hosts = sample_utils.calculate_sample_subject_matrix(samples)

            total_genes = set(passed_sites_map.keys())
```

```
		if len(allowed_genes)==0:
				allowed_genes = set(passed_sites_map.keys())

		allowed_genes = (allowed_genes & total_genes)

		if len(allowed_variant_types)==0:
				allowed_variant_types = set(['1D','2D','3D','4D'])

		private_snvs = []

		for gene_name in allowed_genes:

				for variant_type in passed_sites_map[gene_name].keys():

						if variant_type not in allowed_variant_types:
								continue

						passed_sites =
passed_sites_map[gene_name][variant_type]['sites']

						allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

						if len(allele_counts)==0:
								continue

						locations =
allele_counts_map[gene_name][variant_type]['locations']

						depths = allele_counts.sum(axis=2)
						freqs = allele_counts[:,:,0]*1.0/(depths+(depths==0))

						derived_sites = (freqs>=upper_threshold)
						ancestral_sites = (freqs<=lower_threshold)

						# Sites where the major allele is at sufficiently high
frequency
						high_freq_sites = numpy.logical_or(ancestral_sites,
derived_sites)

						passed_depths = (depths>0)

						confident_sites = numpy.logical_and(high_freq_sites,
passed_depths)

						#print confident_sites.shape

						# goes from L x n to L x h (sites across all hosts)
						host_confident_sites = (numpy.einsum('ij,jk',
confident_sites, sample_host_matrix)>0.5)

						host_derived_sites =
(numpy.einsum('ij,jk',derived_sites, sample_host_matrix)>0.5)

						#print host_confident_sites.shape

						host_sample_sizes = host_confident_sites.sum(axis=1)
						host_derived_counts = host_derived_sites.sum(axis=1)

						#print host_sample_sizes.shape
```

```
                                        #print host_derived_counts.shape

                                        private_idxs =
numpy.nonzero((host_sample_sizes>3.5)*(host_derived_counts==1))[0]

                                        #print private_idxs.shape

                                        if len(private_idxs)>0:
                                                for snp_idx in private_idxs:
                                                        host =
hosts[numpy.nonzero(host_derived_sites[snp_idx])[0][0]]

                                                        contig, location =
allele_counts_map[gene_name][variant_type]['locations'][snp_idx]

                                                        private_snvs.append((contig,
location, gene_name, variant_type, host))

                return private_snvs


def calculate_singleton_matrix(allele_counts_map, passed_sites_map,
allowed_variant_types=set([]), allowed_genes=set([]),
lower_threshold=config.consensus_lower_threshold,
upper_threshold=config.consensus_upper_threshold):

                total_genes = set(passed_sites_map.keys())

                if len(allowed_genes)==0:
                                allowed_genes = set(passed_sites_map.keys())

                allowed_genes = (allowed_genes & total_genes)

                if len(allowed_variant_types)==0:
                                allowed_variant_types = set(['1D','2D','3D','4D'])

                doubleton_matrix =
numpy.zeros_like( passed_sites_map.values()[0].values()[0]['sites'] )*1.0
                singleton_matrix = numpy.zeros_like(doubleton_matrix)

                difference_matrix = numpy.zeros_like(doubleton_matrix)

                opportunity_matrix = numpy.zeros_like(doubleton_matrix)


                for gene_name in allowed_genes:

                                for variant_type in passed_sites_map[gene_name].keys():

                                        if variant_type not in allowed_variant_types:
                                                continue

                                        passed_sites =
passed_sites_map[gene_name][variant_type]['sites']

                                        allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

                                        if len(allele_counts)==0:
                                                continue
```

```
                                        depths = allele_counts.sum(axis=2)
                                        freqs = allele_counts[:,:,0]*1.0/(depths+(depths==0))

                                        derived_sites = (freqs>=upper_threshold)
                                        ancestral_sites = (freqs<=lower_threshold)

                                        # Sites where the major allele is at sufficiently high
frequency
                                        high_freq_sites = numpy.logical_or(ancestral_sites,
derived_sites)
                                        # Those where it is not
                                        intermediate_freq_sites =
numpy.logical_not(high_freq_sites)

                                        # site*sample*sample matrix of sites with sufficient
coverage in both samples
                                        passed_depths =
(depths>0)[:,:,None]*(depths>0)[:,None,:]

                                        # site*sample*sample matrix of sites where we can look
for differences
                                        confident_sites =
numpy.logical_and(high_freq_sites[:,:,None], high_freq_sites[:,None,:])*passed_depths

                                        site_difference_matrix =
numpy.logical_or(derived_sites[:,:,None]*ancestral_sites[:,None,:],
       ancestral_sites[:,:,None]*derived_sites[:,None,:] )

                                        # this is really the only place you have to switch
                                        # to within-between hosts

                                        site_sample_size_matrix = confident_sites.sum(axis=2)
                                        # Want at least a sample size of 4
                                        site_sufficient_sample_size_matrix =
(site_sample_size_matrix>3.5)
                                        #
                                        site_total_difference_matrix =
(site_difference_matrix*confident_sites).sum(axis=2)

                                        potential_singletons = ((site_sample_size_matrix-
site_total_difference_matrix)==1)

                                        potential_doubletons = ((site_sample_size_matrix-
site_total_difference_matrix)==2)


                                        # Confident sites is now asymmetric
                                        confident_sites *=
site_sufficient_sample_size_matrix[:,:,None]

                                        # Sites that we can't count (but we would have counted
in passed_sites)
                                        non_confident_sites =
numpy.logical_not(confident_sites)*passed_depths


                                        # total number of differences between i and j
                                        # (regardless of singleton/doubleton status)
                                        differences =
(site_difference_matrix*confident_sites).sum(axis=0)
```

```
                                        singletons =
(confident_sites*site_difference_matrix*potential_singletons[:,:,None]).sum(axis=0)

                                        doubletons = (confident_sites *
numpy.logical_not(site_difference_matrix) * potential_doubletons[:,:,None]).sum(axis=0)

                                        opportunities = passed_sites -
(numpy.logical_not(confident_sites)*passed_depths).sum(axis=0)

                                        singleton_matrix += singletons
                                        doubleton_matrix += doubletons
                                        difference_matrix += differences
                                        opportunity_matrix += opportunities


            return doubleton_matrix, singleton_matrix, difference_matrix, opportunity_matrix

'''
# ====================================================================
# Given allele counts for each "passed" site in each sample,
# get the locations of just SNP opportunities
# ====================================================================

def calculate_opportunity_locations(allele_counts_map, alt_passed_sites_map,
allowed_variant_types=set(['1D','2D','3D','4D']), allowed_genes=set([]),
lower_threshold=config.consensus_lower_threshold,
upper_threshold=config.consensus_upper_threshold, min_change=config.fixation_min_change):

            # Which genes to consider
            total_genes = set(passed_sites_map.keys())
            allowed_genes = total_genes if len(allowed_genes) == 0 else (allowed_genes &
total_genes)

            # Result: gene -> variant_type -> location tuple (contig, position) -> NxN matrix
of whether a sample pair has opportunity at that site
            opportunity_location_map = {gene: {vt: {} for vt in allowed_variant_types} for gene
in allowed_genes}

            # Recall: allele_counts_map is
            # gene -> variant_type -> ['alleles', 'locations']
            # 'alleles': list, across sites, of array, across samples, of (A,D) tuples
            # 'locations': list, across sites, of location tuples (contig, position)

            # Recall: passed_sites_map is
            # gene -> variant_type -> ['locations', 'sites']
            # 'locations': list, across sites, of location tuples (contig, position)
            # 'sites': sample x sample matrix showing number of sites that "pass" between both
samples

            arbdict = passed_sites_map.values()[0]
            arbmatrix = arbdict['1D']['sites']

            for gene_name in allowed_genes:
                  for variant_type in passed_sites_map[gene_name]:

                        if variant_type not in allowed_variant_types:
                                    continue

                        passed_sites = passed_sites_map[gene_name][variant_type]['sites']
                        allele_counts = allele_counts_map[gene_name][variant_type]['alleles']
                        if len(allele_counts)==0:
```

```
                                        continue

                            # Set up list of location tuples lists
                            opportunity_location_map[gene_name][variant_type] = [[] for _ in
range(arbmatrix.shape[0])]

                            # matrices: S x N where S is # sites, N is # samples
                            depths = allele_counts.sum(axis=2)
                            freqs = allele_counts[:,:,0]*1.0/(depths+(depths==0))

                            derived_sites = (freqs>=upper_threshold)
                            ancestral_sites = (freqs<=lower_threshold)

                            # Sites where the major allele is at sufficiently high frequency
                            high_freq_sites = numpy.logical_or(ancestral_sites, derived_sites)
                            # Those where it is not
                            intermediate_freq_sites = numpy.logical_not(high_freq_sites)

                            # site*sample*sample matrix of sites with sufficient coverage in both
samples
                            passed_depths = (depths>0)[:,:,None]*(depths>0)[:,None,:]

                            # site*sample*sample matrix of sites where we can look for
differences
                            confident_sites = numpy.logical_and(high_freq_sites[:,:,None],
high_freq_sites[:,None,:])*passed_depths

                            # site*sample*sample matrix of sites that are missing data
                            # based on allele freqs, but which had sufficient coverage
                            # (we need to remove these from opportunities below)
                            missing_data_sites =
numpy.logical_or(intermediate_freq_sites[:,:,None],intermediate_freq_sites[:,None,:])*passed_dep
ths

                            # sites where you could have had a reversion
                            reversion_opportunities = derived_sites[:,:,None]*confident_sites
                            mutation_opportunities = (passed_sites - missing_data_sites.sum -
reversion_opportunities)

                            ac_locations = allele_counts_map[gene_name][variant_type]
                            ps_locations = passed_sites_map[

                            for rev_opp_matrix, location in zip(reversion_opportunities,
ac_locations):
                                  opportunity_location_map[gene_name][variant_type][location] =
numpy.zeros_like(arbmatrix)
                                  for

                            for mut_opp_matrix, location in zip(mutation_opportunities,

             return mut_fixation_matrix, rev_fixation_matrix, mut_opportunity_matrix,
rev_opportunity_matrix
'''
# ============================================================================
# Given allele counts for each "passed" site in each sample
# ============================================================================

def calculate_mutation_reversion_matrix(allele_counts_map, passed_sites_map,
allowed_variant_types=set([]), allowed_genes=set([]),
lower_threshold=config.consensus_lower_threshold,
upper_threshold=config.consensus_upper_threshold, min_change=config.fixation_min_change):
```

```
	# Which genes to consider
	total_genes = set(passed_sites_map.keys())

	if len(allowed_genes) == 0:
		allowed_genes = set(passed_sites_map.keys())

	allowed_genes = (allowed_genes & total_genes)

	if len(allowed_variant_types)==0:
		allowed_variant_types = set(['1D','2D','3D','4D'])

	arbdict = passed_sites_map.values()[0]
	arbmatrix = arbdict['1D']['sites']

	mut_fixation_matrix = numpy.zeros_like(arbmatrix).astype('float64')
	rev_fixation_matrix = numpy.zeros_like(mut_fixation_matrix)
	mut_opportunity_matrix = numpy.zeros_like(mut_fixation_matrix)
	rev_opportunity_matrix = numpy.zeros_like(mut_fixation_matrix)

	for gene_name in allowed_genes:
		for variant_type in passed_sites_map[gene_name]:

			if variant_type not in allowed_variant_types:
				continue

			passed_sites = passed_sites_map[gene_name][variant_type]['sites']
			allele_counts = allele_counts_map[gene_name][variant_type]['alleles']

			if len(allele_counts)==0:
				continue

			depths = allele_counts.sum(axis=2)
			freqs = allele_counts[:,:,0]*1.0/(depths+(depths==0))

			derived_sites = (freqs>=upper_threshold)
			ancestral_sites = (freqs<=lower_threshold)

			# Sites where the major allele is at sufficiently high frequency
			high_freq_sites = numpy.logical_or(ancestral_sites, derived_sites)
			# Those where it is not
			intermediate_freq_sites = numpy.logical_not(high_freq_sites)

			# site*sample*sample matrix of sites with sufficient coverage in both
samples
			passed_depths = (depths>0)[:,:,None]*(depths>0)[:,None,:]

			# site*sample*sample matrix of sites where we can look for
differences
			confident_sites = numpy.logical_and(high_freq_sites[:,:,None],
high_freq_sites[:,None,:])*passed_depths


			# site*sample*sample matrix of sites that are missing data
			# based on allele freqs, but which had sufficient coverage
			# (we need to remove these from opportunities below)
			missing_data_sites =
numpy.logical_or(intermediate_freq_sites[:,:,None],intermediate_freq_sites[:,None,:])*passed_dep
ths

			# Calculate mutations and reversions
```

```
                        mutations =
(ancestral_sites[:,:,None])*(derived_sites[:,None,:])*confident_sites

                        reversions =
(derived_sites[:,:,None])*(ancestral_sites[:,None,:])*confident_sites

                        # sites were you could have had a reversion
                        reversion_opportunities = derived_sites[:,:,None]*confident_sites

                        mut_fixation_matrix += (mutations).sum(axis=0)
                        rev_fixation_matrix += (reversions).sum(axis=0)

                        rev_opportunity_matrix += (reversion_opportunities).sum(axis=0)
                        mut_opportunity_matrix += (passed_sites -
missing_data_sites.sum(axis=0) - reversion_opportunities.sum(axis=0) )

            return mut_fixation_matrix, rev_fixation_matrix, mut_opportunity_matrix,
rev_opportunity_matrix


def calculate_fixation_matrix(allele_counts_map, passed_sites_map,
allowed_variant_types=set([]), allowed_genes=set([]),
lower_threshold=config.consensus_lower_threshold,
upper_threshold=config.consensus_upper_threshold, min_change=config.fixation_min_change):

            mut_fixation_matrix, rev_fixation_matrix, mut_opportunity_matrix,
rev_opportunity_matrix = calculate_mutation_reversion_matrix(allele_counts_map,
passed_sites_map, allowed_variant_types, allowed_genes, lower_threshold,
upper_threshold, min_change)

            fixation_matrix = mut_fixation_matrix + rev_fixation_matrix
            opportunity_matrix = mut_opportunity_matrix + rev_opportunity_matrix

            return fixation_matrix, opportunity_matrix


# same as above, but returns two matrices with counts of
# mutations (i->j away from consensus allele) and
# reversion (i->j toward consensus allele)
def calculate_fixation_matrix_mutation_reversion(allele_counts_map, passed_sites_map,
allowed_variant_types=set([]), allowed_genes=set([]),
lower_threshold=config.consensus_lower_threshold,
upper_threshold=config.consensus_upper_threshold, min_change=config.fixation_min_change):

            total_genes = set(passed_sites_map.keys())

            if len(allowed_genes)==0:
                        allowed_genes = set(passed_sites_map.keys())

            allowed_genes = (allowed_genes & total_genes)

            if len(allowed_variant_types)==0:
                        allowed_variant_types = set(['1D','2D','3D','4D'])

            fixation_matrix_mutation =
numpy.zeros_like(passed_sites_map.values()[0].values()[0]['sites'])*1.0
            fixation_matrix_reversion = numpy.zeros_like(fixation_matrix_mutation)*1.0

            passed_sites = numpy.zeros_like(fixation_matrix_mutation)*1.0

            for gene_name in allowed_genes:
```

```
                        for variant_type in passed_sites_map[gene_name].keys():

                                    if variant_type not in allowed_variant_types:
                                                continue

                                    passed_sites +=
passed_sites_map[gene_name][variant_type]['sites']

                                    allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

                                    if len(allele_counts)==0:
                                                continue

                                    depths = allele_counts.sum(axis=2)
                                    freqs = allele_counts[:,:,0]*1.0/(depths+(depths==0))

                                    intermediate_freq_sites =
(freqs>lower_threshold)*(freqs<upper_threshold)

                                    passed_depths =
(depths>0)[:,:,None]*(depths>0)[:,None,:]

                                    bad_sites =
numpy.logical_or(intermediate_freq_sites[:,:,None],intermediate_freq_sites[:,None,:])*passed_dep
ths

                                    delta_freqs = (freqs[:,:,None]-
freqs[:,None,:])*passed_depths

                                    mutations = (delta_freqs>=min_change)
                                    reversions = (delta_freqs<=(-1*min_change))

                                    fixation_matrix_mutation += mutations.sum(axis=0) # sum
over sites
                                    fixation_matrix_reversion += reversions.sum(axis=0) #
sum over sites

                                    passed_sites -= bad_sites.sum(axis=0)

            return fixation_matrix_mutation, fixation_matrix_reversion, passed_sites

# same as above, but returns two matrices with counts of
# mutations (i->j away from consensus allele) and
# reversion (i->j toward consensus allele)
def calculate_preexisting_snps(allele_counts_map, passed_sites_map,
allowed_variant_types=set([]), allowed_genes=set([]),min_freq=0.1):

            total_genes = set(passed_sites_map.keys())

            if len(allowed_genes)==0:
                        allowed_genes = set(passed_sites_map.keys())

            allowed_genes = (allowed_genes & total_genes)

            if len(allowed_variant_types)==0:
                        allowed_variant_types = set(['1D','2D','3D','4D'])

            snp_locations = []
```

```
	for gene_name in allowed_genes:

		for variant_type in passed_sites_map[gene_name].keys():

			if variant_type not in allowed_variant_types:
				continue

			allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

			if len(allele_counts)==0:
				continue

			depths = allele_counts.sum(axis=2)
			freqs = allele_counts[:,:,0]*1.0/(depths+(depths==0))

			site_raw_prevalence = (depths>0).sum(axis=1)
			snp_raw_prevalence = (freqs>min_freq).sum(axis=1)

			snp_prevalence =
snp_raw_prevalence*1.0/(site_raw_prevalence+(site_raw_prevalence==0))


			polymorphic_sites = (snp_raw_prevalence>0)
			if polymorphic_sites.sum()==0:
				continue

			# get locations
			polymorphic_site_idxs =
numpy.nonzero(polymorphic_sites)[0]
			for idx in polymorphic_site_idxs:

	snp_locations.append( (allele_counts_map[gene_name][variant_type]['locations'][idx][0],
allele_counts_map[gene_name][variant_type]['locations'][idx][1], snp_prevalence[idx] ) )

	return snp_locations

####
#
# Calculates the number of within-patient polymorphism differences between
# two samples. (e.g. something that is fixed in one timepoint and polymorphic
# in another.
#
####
def calculate_new_snp_matrix(allele_counts_map, passed_sites_map, allowed_variant_types=set([]),
allowed_genes=set([]), min_freq=0.05, max_freq=0.2):

	total_genes = set(passed_sites_map.keys())

	if len(allowed_genes)==0:
		allowed_genes = set(passed_sites_map.keys())

	allowed_genes = (allowed_genes & total_genes)

	if len(allowed_variant_types)==0:
		allowed_variant_types = set(['1D','2D','3D','4D'])

	new_snp_matrix =
numpy.zeros_like(passed_sites_map.values()[0].values()[0]['sites'])*1.0
	passed_sites = numpy.zeros_like(new_snp_matrix)*1.0
```

```
        for gene_name in allowed_genes:

                for variant_type in passed_sites_map[gene_name].keys():

                        if variant_type not in allowed_variant_types:
                                continue

                        passed_sites +=
passed_sites_map[gene_name][variant_type]['sites']

                        allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

                        if len(allele_counts)==0:
                                continue


                        depths = allele_counts.sum(axis=2)
                        freqs = allele_counts[:,:,0]/(depths+(depths==0))
                        # turn into minor allele frequencies
                        mafs = numpy.fmin(freqs,1-freqs)

                        # Turn

                        new_snps_1 =
(mafs[:,:,None]<min_freq)*(mafs[:,None,:]>max_freq)
                        new_snps_2 =
(mafs[:,:,None]>max_freq)*(mafs[:,None,:]<min_freq)
                        total_new_snps = new_snps_1+new_snps_2

                        passed_depths =
(depths>0)[:,:,None]*(depths>0)[:,None,:]

                        total_new_snps[passed_depths==0] = 0

                        new_snp_matrix += total_new_snps.sum(axis=0)

        return new_snp_matrix, passed_sites



def calculate_pi_matrix(allele_counts_map, passed_sites_map, variant_type='4D',
allowed_genes=None):

        if allowed_genes == None:
                allowed_genes = set(passed_sites_map.keys())

        pi_matrix =
numpy.zeros_like(passed_sites_map[passed_sites_map.keys()[0]][variant_type]['sites'])*1.0
        avg_pi_matrix = numpy.zeros_like(pi_matrix)
        passed_sites = numpy.zeros_like(pi_matrix)

        for gene_name in allowed_genes:

                if gene_name in passed_sites_map.keys():
                        #print passed_sites_map[gene_name][variant_type].shape,
passed_sites.shape
                        #print gene_name, variant_type

                        passed_sites +=
passed_sites_map[gene_name][variant_type]['sites']
```

```
                                        allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

                                        if len(allele_counts)==0:
                                                continue


                                        depths = allele_counts.sum(axis=2)
                                        freqs = allele_counts/(depths+(depths<0.1))[:,:,None]
                                        self_freqs = (allele_counts-1)/(depths-
1+2*(depths<1.1))[:,:,None]
                                        self_pis = ((depths>0)-(freqs*self_freqs).sum(axis=2))

                                        I,J = depths.shape

                                        # pi between sample j and sample l
                                        gene_pi_matrix =
numpy.einsum('ij,il',(depths>0)*1.0,(depths>0)*1.0)-numpy.einsum('ijk,ilk',freqs,freqs)

                                        # average of pi within sample j and within sample i
                                        gene_avg_pi_matrix =
(numpy.einsum('ij,il',self_pis,(depths>0)*1.0)+numpy.einsum('ij,il',(depths>0)*1.0,self_pis))/2

                                        diagonal_idxs = numpy.diag_indices(J)
                                        gene_pi_matrix[diagonal_idxs] =
gene_avg_pi_matrix[diagonal_idxs]

                                        pi_matrix += gene_pi_matrix
                                        avg_pi_matrix += gene_avg_pi_matrix

                # We used to normalize here
                #pi_matrix = pi_matrix /(passed_sites+(passed_sites==0))
                #avg_pi_matrix = avg_pi_matrix/(passed_sites+(passed_sites==0))
                # Now we return passed sites

                return pi_matrix, avg_pi_matrix, passed_sites




def phylip_distance_matrix_str(matrix, samples):

                lines = [str(len(samples))]
                for i in xrange(0,len(samples)):
                                lines.append( "\t".join([samples[i]]+["%g" % x for x in
matrix[i,:]]))

                return "\n".join(lines)

import numpy
from scipy.special import gammaln as loggamma

def fold_sfs(fs):
                n = len(fs)+1
                folded_fs = (fs + fs[::-1])[0:(n-1)/2]
                if (n-1) % 2 != 0:
                                folded_fs[-1] *= 0.5
                return folded_fs
```

```
def estimate_sfs_naive_binning(allele_counts, target_depth=10):

		depths = allele_counts.sum(axis=1)

		allele_counts = allele_counts[depths>0]
		depths = depths[depths>0]

		freqs = allele_counts[:,0]/depths

		bins = (numpy.arange(0,target_depth+2)-0.5)/target_depth

		counts,dummy = numpy.histogram(freqs,bins)

		return counts

def estimate_sfs_downsampling(allele_counts, target_depth=10):

		depths = allele_counts.sum(axis=1)

		allele_counts = allele_counts[depths>0]
		depths = depths[depths>0]

		Dmin = min([depths.min(),target_depth]) # this is what we have to downsample to
		# if you don't like it, send us an allele_counts matrix
		# that has been thresholded to a higher min value

		count_density = numpy.zeros(Dmin+1)*1.0


		A = numpy.outer(allele_counts[:,0], numpy.ones(Dmin+1))
		D = numpy.outer(depths, numpy.ones(Dmin+1))
		ks = numpy.outer(numpy.ones_like(depths), numpy.arange(0,Dmin+1))

		count_density = numpy.exp(loggamma(A+1)-loggamma(A-ks+1)-loggamma(ks+1) +
loggamma(D-A+1)-loggamma(D-A-(Dmin-ks)+1)-loggamma(Dmin-ks+1) + loggamma(D-Dmin+1) +
loggamma(Dmin+1) - loggamma(D+1)).sum(axis=0)

		return count_density



# Calculate polarized SNP changes from i to j that exceed threshold
# Returns list of differences, number of comparisons. Each difference is a tuple of form
#
# (gene_name, (contig, location), (alt_i, depth_i), (alt_j, depth_j))
#
def calculate_snp_differences_between(i,j,allele_counts_map, passed_sites_map, avg_depth_i,
avg_depth_j, min_coverage=config.min_median_coverage, allowed_variant_types=set([]),
allowed_genes=set([]), lower_threshold=config.temporal_lower_threshold,
upper_threshold=config.temporal_upper_threshold,
log10_depth_ratio_threshold=config.fixation_log10_depth_ratio_threshold):

		if len(allowed_genes)==0:
				allowed_genes = set(passed_sites_map.keys())

		if len(allowed_variant_types)==0:
				allowed_variant_types = set(['1D','2D','3D','4D'])

		snp_changes = []
		for gene_name in allowed_genes:
```

```
                        if gene_name not in allele_counts_map.keys():
                                continue

                        for variant_type in allele_counts_map[gene_name].keys():

                                if variant_type not in allowed_variant_types:
                                        continue

                                allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

                                if len(allele_counts)==0:
                                        continue

                                allele_counts = allele_counts[:,[i,j],:]
                                depths = allele_counts.sum(axis=2)
                                alt_freqs = allele_counts[:,:,0]/(depths+(depths==0))

                                safe_depths = depths+(depths==0)

                                log10_depth_ratios =
numpy.fabs(numpy.log10((safe_depths[:,0]/avg_depth_i)/(safe_depths[:,1]/avg_depth_j)))

                                passed_depths =
(depths>min_coverage)[:,0]*(depths>min_coverage)[:,1]*(log10_depth_ratios<log10_depth_ratio_thre
shold)

                                mutations =
(alt_freqs[:,0]<=lower_threshold)*(alt_freqs[:,1]>=upper_threshold)*passed_depths
                                reversions =
(alt_freqs[:,0]>=upper_threshold)*(alt_freqs[:,1]<=lower_threshold)*passed_depths

                                changed_sites =
numpy.nonzero( numpy.logical_or(mutations, reversions) )[0]

                                if len(changed_sites)>0:
                                        # some fixations!

                                        for idx in changed_sites:
                                                snp_changes.append((gene_name,
allele_counts_map[gene_name][variant_type]['locations'][idx], variant_type,
(allele_counts[idx,0,0], depths[idx,0]), (allele_counts[idx,1,0],depths[idx,1]) ))

        return snp_changes

# Calculate polarized SNP changes from i to j that exceed threshold
# Returns list of differences, number of comparisons. Each difference is a tuple of form
#
# (gene_name, (contig, location), (alt_i, depth_i), (alt_j, depth_j))
#
def calculate_tracked_private_snvs(i,j,allele_counts_map, passed_sites_map, avg_depth_i,
avg_depth_j, private_snv_map, allowed_variant_types=set([]), allowed_genes=set([]),
lower_threshold=config.consensus_lower_threshold,
upper_threshold=config.consensus_upper_threshold,
log10_depth_ratio_threshold=config.fixation_log10_depth_ratio_threshold):

        if len(allowed_genes)==0:
                allowed_genes = set(passed_sites_map.keys())

        if len(allowed_variant_types)==0:
                allowed_variant_types = set(['1D','2D','3D','4D'])
```

```
            tracked_private_snps = []
            for gene_name in allowed_genes:

                        if gene_name not in allele_counts_map.keys():
                                    continue

                        for variant_type in allele_counts_map[gene_name].keys():

                                    if variant_type not in allowed_variant_types:
                                                continue

                                    allele_counts =
allele_counts_map[gene_name][variant_type]['alleles']

                                    if len(allele_counts)==0:
                                                continue

                                    allele_counts = allele_counts[:,[i,j],:]
                                    depths = allele_counts.sum(axis=2)
                                    alt_freqs = allele_counts[:,:,0]/(depths+(depths==0))

                                    safe_depths = depths+(depths==0)

                                    log10_depth_ratios =
numpy.fabs(numpy.log10((safe_depths[:,0]/avg_depth_i)/(safe_depths[:,1]/avg_depth_j)))

                                    passed_depths =
(depths>0)[:,0]*(depths>0)[:,1]*(log10_depth_ratios<log10_depth_ratio_threshold)

                                    initial_high_freqs = alt_freqs[:,0]>=upper_threshold
                                    final_high_freqs = alt_freqs[:,1]>=upper_threshold
                                    final_low_freqs = alt_freqs[:,1]<=lower_threshold

                                    potential_private_snps =
numpy.nonzero( initial_high_freqs*numpy.logical_or(final_high_freqs, final_low_freqs) )[0]

                                    if len(potential_private_snps)>0:
                                                # some candidates for private SNVs
                                                for idx in potential_private_snps:
                                                            # check to see if it is indeed
a private SNV
                                                            location_tuple =
allele_counts_map[gene_name][variant_type]['locations'][idx]

                                                            if location_tuple in
private_snv_map:
                                                                        # it is indeed
private!

      tracked_private_snps.append((gene_name, location_tuple, variant_type,
(allele_counts[idx,0,0], depths[idx,0]), (allele_counts[idx,1,0],depths[idx,1]) ))

            return tracked_private_snps



########################################

def calculate_mean_pi_matrix_per_pathway(pi_per_gene, avg_pi_per_gene,
passed_sites_per_gene,num_people_with_data, kegg_ids,min_passed_sites_per_person=100):
```

```
            pi_per_pathway={}
            avg_pi_per_pathway={}
            passed_sites_per_pathway={}
            num_genes_per_pathway={}
            num_people_with_data_pathway={}
            gene_name=avg_pi_per_gene.keys()[0]

            pi_per_pathway['Annotated pathways'] = numpy.zeros_like(pi_per_gene[gene_name])
            avg_pi_per_pathway['Annotated
pathways']=numpy.zeros_like(avg_pi_per_gene[gene_name])
            passed_sites_per_pathway['Annotated
pathways']=numpy.zeros_like(passed_sites_per_gene[gene_name])
            num_genes_per_pathway['Annotated pathways']=0
            num_people_with_data_pathway['Annotated pathways']=0

            for gene_name in avg_pi_per_gene.keys():
                        pathway=kegg_ids[gene_name][0][1]
                        if pathway not in avg_pi_per_pathway.keys():
                                    pi_per_pathway[pathway]=pi_per_gene[gene_name]
                                    avg_pi_per_pathway[pathway]=avg_pi_per_gene[gene_name]

      passed_sites_per_pathway[pathway]=passed_sites_per_gene[gene_name]
                                    num_genes_per_pathway[pathway]=1

      num_people_with_data_pathway[pathway]=num_people_with_data[gene_name]
                        else:
                                    pi_per_pathway[pathway]+=pi_per_gene[gene_name]
                                    avg_pi_per_pathway[pathway]+=avg_pi_per_gene[gene_name]

      passed_sites_per_pathway[pathway]+=passed_sites_per_gene[gene_name]
                                    num_genes_per_pathway[pathway]+=1

      num_people_with_data_pathway[pathway]+=num_people_with_data[gene_name]
                        if pathway !='':
                                    pi_per_pathway['Annotated pathways'] +=
pi_per_gene[gene_name]
                                    avg_pi_per_pathway['Annotated pathways']
+=avg_pi_per_gene[gene_name]
                                    passed_sites_per_pathway['Annotated pathways']
+=passed_sites_per_gene[gene_name]
                                    num_genes_per_pathway['Annotated pathways']+=1
                                    num_people_with_data_pathway['Annotated
pathways']+=num_people_with_data[gene_name]

            for pathway_name in avg_pi_per_pathway.keys():
                        # we want to identify people that have few passed sites even after
aggregating the data accross genes. Then set the values in these cells to zero because these
data points are too noisy

      low_passed_sites_idxs=passed_sites_per_pathway[pathway_name]<min_passed_sites_per_person
                        passed_sites_per_pathway[pathway_name][low_passed_sites_idxs]=0
                        avg_pi_per_pathway[pathway_name][low_passed_sites_idxs]=0
                        pi_per_pathway[pathway_name][low_passed_sites_idxs]=0
                        # now compute pi/pathway.
                        avg_pi_per_pathway[pathway_name] =
avg_pi_per_pathway[pathway_name]/(passed_sites_per_pathway[pathway_name]+(passed_sites_per_pathw
ay[pathway_name]==0))
                        pi_per_pathway[pathway_name] =
pi_per_pathway[pathway_name]/(passed_sites_per_pathway[pathway_name]+(passed_sites_per_pathway[p
athway_name]==0))
```

```
      #num_people_with_data_pathway[pathway_name]=sum(numpy.diagonal(passed_sites_per_pathway[p
athway_name])>=min_passed_sites_per_person)
                        num_people_with_data_pathway[pathway_name]=
num_people_with_data_pathway[pathway_name]/num_genes_per_pathway[pathway_name]
            return
pi_per_pathway,avg_pi_per_pathway,passed_sites_per_pathway,num_people_with_data_pathway,
num_genes_per_pathway

################################


def calculate_mean_fixation_matrix_per_pathway(fixation_per_gene,
passed_sites_per_gene,num_people_with_data, kegg_ids, min_passed_sites_per_person=100):

            fixation_per_pathway={}
            passed_sites_per_pathway={}
            num_genes_per_pathway={}
            num_people_with_data_pathway={}

            gene_name=fixation_per_gene.keys()[0]
            fixation_per_pathway['Annotated pathways'] =
numpy.zeros_like(fixation_per_gene[gene_name])
            passed_sites_per_pathway['Annotated
pathways']=numpy.zeros_like(passed_sites_per_gene[gene_name])
            num_genes_per_pathway['Annotated pathways']=0
            num_people_with_data_pathway['Annotated pathways']=0


            for gene_name in fixation_per_gene.keys():
                        pathway=kegg_ids[gene_name][0][1]
                        if pathway not in fixation_per_pathway.keys():

      fixation_per_pathway[pathway]=fixation_per_gene[gene_name]

      passed_sites_per_pathway[pathway]=passed_sites_per_gene[gene_name]
                                    num_genes_per_pathway[pathway]=1

      num_people_with_data_pathway[pathway]=num_people_with_data[gene_name]
                        else:

      fixation_per_pathway[pathway]+=fixation_per_gene[gene_name]

      passed_sites_per_pathway[pathway]+=passed_sites_per_gene[gene_name]
                                    num_genes_per_pathway[pathway]+=1

      num_people_with_data_pathway[pathway]+=num_people_with_data[gene_name]
                        if pathway !='':
                                    fixation_per_pathway['Annotated pathways'] +=
fixation_per_gene[gene_name]
                                    passed_sites_per_pathway['Annotated pathways']
+=passed_sites_per_gene[gene_name]
                                    num_genes_per_pathway['Annotated pathways']+=1
                                    num_people_with_data_pathway['Annotated
pathways']+=num_people_with_data[gene_name]


            for pathway_name in fixation_per_pathway.keys():
                  # we want to identify people that have few passed sites even after
aggregating the data accross genes. Then set the values in these cells to zero because these
data points are too noisy
```

```
	low_passed_sites_idxs=passed_sites_per_pathway[pathway_name]<min_passed_sites_per_person
                        passed_sites_per_pathway[pathway_name][low_passed_sites_idxs]=0
                        fixation_per_pathway[pathway_name][low_passed_sites_idxs]=0
                        #now compute fixation/pathway
                        fixation_per_pathway[pathway_name] =
fixation_per_pathway[pathway_name]/(passed_sites_per_pathway[pathway_name]+(passed_sites_per_pat
hway[pathway_name]==0))

	num_people_with_data_pathway[pathway_name]=num_people_with_data_pathway[pathway_name]/flo
at(num_genes_per_pathway[pathway_name])
            return fixation_per_pathway, passed_sites_per_pathway,
num_people_with_data_pathway, num_genes_per_pathway

######################
#
# Calculate pi from SFS map
#
######################
def calculate_pi_from_sfs_map(sfs_map):

            alts = []
            refs = []
            depths = []
            counts = []
            for key in sfs_map.keys():
                        D,A = key
                        n = sfs_map[key][0]

                        alts.append(A)
                        refs.append(D-A)
                        depths.append(D)
                        counts.append(n)

            alts = numpy.array(alts)
            refs = numpy.array(refs)
            depths = numpy.array(depths)
            counts = numpy.array(counts)

            alt_lower_threshold = numpy.ceil(depths*0.05)+0.5 #at least one read above 5%.
            alts[alts<alt_lower_threshold] = 0
            alt_upper_threshold = numpy.floor(depths*0.95)-0.5 #at least one read below 95%
            alts[alts>alt_upper_threshold] = depths[alts>alt_upper_threshold]

            total_pi = ((2*alts*(depths-alts)*1.0/(depths*(depths-
1)+(depths<1.1)))*(counts)).sum()
            num_opportunities = counts.sum()

            return total_pi/num_opportunities

def calculate_polymorphism_rates_from_sfs_map(sfs_map,lower_threshold=0.2,upper_threshold=0.8):

            total_sites = 0
            within_sites = 0
            between_sites = 0
            for key in sfs_map.keys():
                        D,A = key
                        n = sfs_map[key][0]
                        reverse_n = sfs_map[key][1]

                        f = A*1.0/D
```

```
                        total_sites += n

                        if ((f>lower_threshold) and (f<upper_threshold)):
                                # an intermediate frequency site
                                within_sites += n
                        else:
                                if f>0.5:
                                        between_sites += (n-reverse_n)
                                else:
                                        between_sites += reverse_n


        between_polymorphism_rate = between_sites*1.0/total_sites
        within_polymorphism_rate = within_sites*1.0/total_sites

        return within_polymorphism_rate, between_polymorphism_rate

#######################
#
# Estimate smoothed within-person SFS with EM algorithm
#
#######################
def calculate_smoothed_sfs(sfs_map, num_iterations=100, perr=0.01,
lower_threshold=config.consensus_lower_threshold,
upper_threshold=config.consensus_upper_threshold):

        alts = []
        refs = []
        depths = []
        counts = []
        for key in sfs_map.keys():
                D,A = key
                n = sfs_map[key][0]

                alts.append(A)
                refs.append(D-A)
                depths.append(D)
                counts.append(n)

        alts = numpy.array(alts)
        refs = numpy.array(refs)
        depths = numpy.array(depths)
        counts = numpy.array(counts)
        weights = counts*1.0/counts.sum()

        # calculate median depth (or rough approximation)
        sorted_depths, sorted_counts = (numpy.array(x) for x in zip(*sorted(zip(depths,
counts))))
        CDF = numpy.cumsum(sorted_counts)*1.0/sorted_counts.sum()
        Dbar = sorted_depths[CDF>0.5][0]
        #Dbar = min([Dbar,100])

        Abars = numpy.arange(0,Dbar+1)
        Rbars = Dbar-Abars
        fs = Abars*1.0/Dbar
        df = fs[1]-fs[0]
        flowers=     fs-df/2
        flowers[0] = 0-1e-10
        fuppers = fs+df/2
        fuppers[-1] = 1+1e-10
```

```
            pfs = numpy.zeros_like(fs)


            # first infer rate of polymorphisms (p_poly) using EM

            # Initial guess
            p_poly = 1e-04

            # calculate probability of data, conditioned on it not being polymorphic
            # (i.e., alt reads are sequencing errors)
            # (this doesn't depend on p_poly)
            pdata_errs = (betainc(alts+1,refs+1,perr)+betainc(refs+1,alts+1,perr))/(2*perr)
            pdata_intermediates = 1-(betainc(alts+1,refs+1,
lower_threshold)+betainc(refs+1,alts+1,1-upper_threshold))
            # EM loop
            for iteration in xrange(0,num_iterations):
                        posterior_polys = 1.0/(1.0+(1-p_poly)/(p_poly)*pdata_errs)
                        p_poly = (posterior_polys*weights).sum()



            # Calculate avg posterior probability of freq being between lower and upper
threshold
            p_intermediate = (posterior_polys*pdata_intermediates*weights).sum()

            # Now Calculate smoothed SFS estimate

            # Posterior method
            #posterior_frequencies = (betainc(alts[:,None]+1,refs[:,None]+1, fuppers[None,:])-
betainc(alts[:,None]+1,refs[:,None]+1,flowers[None,:]))
            # The reason why we don't use this one is that it assumes a higher variance than
our internal model. In reality, we believe that there are a few fixed frequencies, not that
every one is independent. (Really we'd want to do some sort of EM, but it's slowly converging)



            # Bin overlap method

            freqs = alts*1.0/depths
            freqs_plushalf = numpy.clip((alts+0.5)*1.0/depths,0,1)
            freqs_minushalf = numpy.clip((alts-0.5)*1.0/depths,0,1)

            a = numpy.fmax(flowers[None,:],freqs_minushalf[:,None])
            b = numpy.fmin(fuppers[None,:],freqs_plushalf[:,None])

            posterior_frequencies = (b-a)*(b>a)/(freqs_plushalf-freqs_minushalf)[:,None]

            # Delta function method
            #posterior_frequencies =
(freqs[:,None]>flowers[None,:])*(freqs[:,None]<=fuppers[None,:])
            # the reason why we don't use this one is that it suffers from binning artefacts
            # though not *so* bad

            #pfs = ((posterior_frequencies)*((posterior_polys*weights)[:,None])).sum(axis=0)
            pfs = ((posterior_frequencies)*((weights)[:,None])).sum(axis=0)

            pfs /= pfs.sum()


            # Re-sampling method (too smooth)
```

```
            #prefactors =
numpy.exp( loggamma(Abars[None,:]+alts[:,None]+1)+loggamma(Rbars[None,:]+refs[:,None]+1)+loggamm
a(Dbar+1)+loggamma(depths+1)[:,None]-loggamma(Dbar+depths+2)[:,None]-loggamma(Abars+1)[None,:]-
loggamma(Rbars+1)[None,:]-loggamma(alts+1)[:,None]-loggamma(refs+1)[:,None])
            #pfs = ((prefactors*(p_poly+(1-p_poly)*(betainc(Abars[None,:]+alts[:,None]+1,
Rbars[None,:]+refs[:,None]+1, perr)+betainc(Rbars[None,:]+refs[:,None]+1,
Abars[None,:]+alts[:,None]+1, perr))/(2*perr)))*weights[:,None]).sum(axis=0)

            print p_poly, p_intermediate, Dbar

            return fs, pfs, p_intermediate, p_poly



######################
#
# Estimate smoothed within-person SFS with EM algorithm
#
######################
def calculate_smoothed_sfs_continuous_EM(sfs_map,fs=[],num_iterations=100):

            alts = []
            refs = []
            depths = []
            counts = []
            for key in sfs_map.keys():
                        D,A = key
                        n = sfs_map[key][0]

                        alts.append(A)
                        refs.append(D-A)
                        depths.append(D)
                        counts.append(n)

            alts = numpy.array(alts)
            refs = numpy.array(refs)
            depths = numpy.array(depths)
            counts = numpy.array(counts)


            weights = counts*1.0/counts.sum()

            if len(fs)==0:
                        fs = numpy.linspace(0,1,101)[1:-1]

            dfs = fs[1]-fs[0]

            logfs = numpy.log(fs)
            log1minusfs = numpy.log(1-fs)

            # initial guess for pfs
            pfs = numpy.zeros_like(fs)
            pfs[fs>=0.99] = 1e-02/(fs>=0.99).sum()
            pfs[(fs<0.99)*(fs>0.01)] = 1e-04/((fs<0.99)*(fs>0.01)).sum()
            pfs[fs<=0.01] = (1-1e-02-1e-04)/(fs<=0.01).sum()
            #print pfs.sum()
            pfs /= pfs.sum()

            # EM loop
            for iteration in xrange(0,num_iterations):
                        log_pfs = numpy.log(pfs)
```

```
                        log_posteriors =
alts[:,None]*logfs[None,:]+refs[:,None]*log1minusfs[None,:]+numpy.log(pfs)[None,:]

                        log_posteriors -= log_posteriors.max(axis=1)[:,None]

                        posteriors = numpy.exp(log_posteriors)
                        posteriors /= posteriors.sum(axis=1)[:,None]

                        pfs = (posteriors*weights[:,None]).sum(axis=0)
                        pfs = numpy.clip(pfs, 1e-100, 1e100)

                        #print pfs.sum()

                        # normalize
                        pfs /= pfs.sum()

            return fs, pfs

def get_truong_pvalue(A,D):
            A = min([A,D-A])
            perr = 1e-02

            return scipy.stats.binom.sf(A,D,perr)+scipy.stats.binom.pmf(A,D,perr)


# definition of a polymorphic site according to Truong et al
def is_polymorphic_truong(A,D):

            alpha = get_truong_pvalue(A,D)

            return alpha<0.05

def calculate_highcoverage_samples(species_name, min_coverage=config.min_median_coverage):

            # Load genomic coverage distributions
            sample_coverage_histograms, samples =
parse_midas_data.parse_coverage_distribution(species_name, prev_cohort='any') # note prev cohort
'any'
            median_coverages =
numpy.array([stats_utils.calculate_nonzero_median_from_histogram(sample_coverage_histogram) for
sample_coverage_histogram in sample_coverage_histograms])
            sample_coverage_map = {samples[i]: median_coverages[i] for i in
xrange(0,len(samples))}
            samples = numpy.array(samples)

            median_coverages = numpy.array([sample_coverage_map[samples[i]] for i in
xrange(0,len(samples))])

            # Only plot samples above a certain depth threshold
            desired_samples = samples[(median_coverages>=min_coverage)]

            # Only include samples that are in 'all samples' (HACKY: TODO?)
            all_samples = sample_utils.get_sample_names('all')
            final_desired_samples = []
            for sample in desired_samples:
                  if sample in all_samples:
                        final_desired_samples.append(sample)

            return final_desired_samples
```

```
def calculate_haploid_samples(species_name, min_coverage=config.min_median_coverage,
threshold_pi=config.threshold_pi,
threshold_within_between_fraction=config.threshold_within_between_fraction, prev_cohort='all',
debug=False):

            desired_samples = calculate_highcoverage_samples(species_name, min_coverage)

            if len(desired_samples)==0:
                        return numpy.array([])

            # Old way, calculate pi_s
            # Load pi information for species_name
            # Load core gene set
            #sys.stderr.write("Loading core genes...\n")
            #core_genes = parse_midas_data.load_core_genes(species_name)
            #sys.stderr.write("Done! Core genome consists of %d genes\n" % len(core_genes))


            #sys.stderr.write("Loading within-sample diversity for %s...\n" % species_name)
            #samples, total_pis, total_pi_opportunities =
      parse_midas_data.parse_within_sample_pi(species_name, allowed_genes=core_genes,
debug=debug)
            #sys.stderr.write("Done!\n")
            #pis = total_pis/total_pi_opportunities

            #median_coverages = numpy.array([sample_coverage_map[samples[i]] for i in
xrange(0,len(samples))])

            # Only plot samples above a certain depth threshold that are "haploids"
            #haploid_samples = samples[(median_coverages>=min_coverage)*(pis<=threshold_pi)]

            #return haploid_samples

            # New way with pre-computed SFS
            # Load SFS information for species_name
            import sfs_utils
            samples, sfs_map = parse_midas_data.parse_within_sample_sfs(species_name,
            allowed_variant_types=set(['4D']), prev_cohort=prev_cohort)

            haploid_samples = []
            for sample in desired_samples:
                        within_sites, between_sites, total_sites =
sfs_utils.calculate_polymorphism_rates_from_sfs_map(sfs_map[sample])

                        if within_sites <= threshold_within_between_fraction*between_sites:
                                    haploid_samples.append(sample)

            return numpy.array(haploid_samples)

# Returns all high coverage samples that are involved in a temporal pair
# Considers mothers and infants the same subject
def calculate_temporal_samples(species_name, min_coverage=config.min_median_coverage):

            highcoverage_samples = calculate_highcoverage_samples(species_name, min_coverage)

            sample_order_map = sample_utils.parse_sample_order_map()

            # Calculate which pairs of idxs belong to the same sample, which to the same
subject
```

```
            # and which to different subjects
            same_sample_idxs, same_subject_idxs, diff_subject_idxs =
sample_utils.calculate_ordered_subject_pairs(sample_order_map, highcoverage_samples,
within_host_type = 'nonconsecutive')

            temporal_samples = set()
            for sample_pair_idx in xrange(0,len(same_subject_idxs[0])):
                        i = same_subject_idxs[0][sample_pair_idx]
                        j = same_subject_idxs[1][sample_pair_idx]
                        temporal_samples.add(highcoverage_samples[i])
                        temporal_samples.add(highcoverage_samples[j])

            sample_subject_map = sample_utils.parse_sample_subject_map()
            mother_samples = sample_utils.get_sample_names('mother','all')
            infant_samples = sample_utils.get_sample_names('infant','all')

            for sample_pair_idx in xrange(0,len(diff_subject_idxs[0])):
                        i = diff_subject_idxs[0][sample_pair_idx]
                        j = diff_subject_idxs[1][sample_pair_idx]
                        sample_i = highcoverage_samples[i]
                        sample_j = highcoverage_samples[j]
                        if sample_utils.is_mi_pair(sample_i, sample_j, mother_samples,
infant_samples) and sample_utils.is_same_mi_subject(sample_i, sample_j, sample_subject_map):
                                    temporal_samples.add(sample_i)
                                    temporal_samples.add(sample_j)

            desired_samples = []
            for sample in highcoverage_samples:
                        if sample in temporal_samples:
                                    desired_samples.append(sample)

            return numpy.array(desired_samples)


def calculate_fixation_error_rate(sfs_map, sample_i, sample_j,dfs=[0.6, 0.3], frequency_bins =
numpy.linspace(0,1,21)):


            dfs = numpy.array(dfs)

            dummy_fs, pfs_i =
sfs_utils.calculate_binned_sfs_from_sfs_map(sfs_map[sample_i],bins=frequency_bins)
            dummy_fs, pfs_j =
sfs_utils.calculate_binned_sfs_from_sfs_map(sfs_map[sample_j],bins=frequency_bins)

            fs = frequency_bins[1:]-(frequency_bins[1]-frequency_bins[0])/2.0

            pfs = (pfs_i+pfs_j)/2.0
            # fold
            pfs = (pfs+pfs[::-1])/2

            # Calculate depth distributions
            dummy, D1s, pD1s =
sfs_utils.calculate_binned_depth_distribution_from_sfs_map(sfs_map[sample_i])
            dummy, D2s, pD2s =
sfs_utils.calculate_binned_depth_distribution_from_sfs_map(sfs_map[sample_j])

            fs = fs[pfs>0]
            pfs = pfs[pfs>0]

            D1s = D1s[pD1s>0]
```

```
            pD1s = pD1s[pD1s>0]
            D2s = D2s[pD2s>0]
            pD2s = pD2s[pD2s>0]


            perrs = {df:0 for df in dfs}
            for D1,pD1 in zip(D1s,pD1s):
                        for D2,pD2 in zip(D2s,pD2s):
                                    for f,pf in zip(fs,pfs): # True f
                                                for df in dfs: # df = 0.6              #
(1-0.6)/2 = 0.2
                                                            # P(true frequency is f | alt
allele frequency f* <= 0.2) at timepoint 1
                                                            # P(true frequency is 1-f |
alt allele frequency f* >= 0.8) at timepoint 2
                                                            # pD1: proportion of sites
that have this depth
                                                            # pD2: proportion of sites
that have this depth
                                                            # freq * depth 1 * depth 2
                                                            perrs[df] +=
2*binom.cdf(D1*(1-df)/2, D1, f)*binom.cdf(D2*(1-df)/2, D2, 1-f)*pD1*pD2*pf

            perrs = numpy.array([perrs[df] for df in dfs])
            return perrs




def find_snps_in_gene_pair(gene1_fasta, gene2_fasta):
            alignment={}
            # key=bp
            # value=[B. vul, B. dorei]

            if len(gene1_fasta) == len(gene2_fasta):
                        for bp in range(0, len(gene1_fasta)):
                                    if gene1_fasta[bp] != gene2_fasta[bp]:

      alignment[bp]=[gene1_fasta[bp],gene2_fasta[bp]]

            return alignment
```

gene_diversity_utils.py

```
import numpy
import sys
from scipy.stats import poisson
from math import fabs
import config

# For each gene in gene_depth_matrix, calculates # of samples
# in which it is "present". Returns vector of prevalences
def calculate_gene_prevalences(gene_depth_matrix, marker_coverages, min_copynum=0.3):

    gene_copynum_matrix = gene_depth_matrix * 1.0 / numpy.clip(marker_coverages,1,1e09)

    return (gene_copynum_matrix>=min_copynum).sum(axis=1)


def calculate_fractional_gene_prevalences(gene_depth_matrix, marker_coverages, min_copynum=0.3):
```

```
    return calculate_gene_prevalences(gene_depth_matrix, marker_coverages,
min_copynum)*1.0/len(marker_coverages)


# For each sample in gene_depth_matrix, calculates # of genes
# that are "present". Returns vector of gene numbers
def calculate_gene_numbers(gene_depth_matrix, marker_coverages, min_copynum=0.5):

    gene_copynum_matrix = gene_depth_matrix * 1.0 / numpy.clip(marker_coverages,1,1e09)

    return (gene_copynum_matrix>=min_copynum).sum(axis=0)

# Calculates the number of gene differences between pairs of samples
# Returns: matrix of # of gene differences
#          matrix of # of opportunities
#          --> we should chat about what the denominator should be!
def calculate_coverage_based_gene_hamming_matrix(gene_reads_matrix, gene_depth_matrix,
marker_coverages, absent_threshold=config.gainloss_max_absent_copynum,
present_lower_threshold=config.gainloss_min_normal_copynum, present_upper_threshold=
config.gainloss_max_normal_copynum):

    gene_hamming_matrix_gain, gene_hamming_matrix_loss, num_opportunities =
calculate_coverage_based_gene_hamming_matrix_gain_loss(gene_reads_matrix, gene_depth_matrix,
marker_coverages, absent_threshold=absent_threshold,
present_lower_threshold=present_lower_threshold,
present_upper_threshold=present_upper_threshold)

    gene_hamming_matrix = gene_hamming_matrix_gain + gene_hamming_matrix_loss
    return gene_hamming_matrix, num_opportunities


def calculate_coverage_based_gene_hamming_matrix_gain_loss(gene_reads_matrix, gene_depth_matrix,
marker_coverages, absent_threshold=config.gainloss_max_absent_copynum,
present_lower_threshold=config.gainloss_min_normal_copynum, present_upper_threshold=
config.gainloss_max_normal_copynum):
    # in this definition, we keep track of whether there was a gene 'gain' or 'loss' by removing
numpy.fabs. This info will be used to plot the gain/loss events between two successive time
pints.

    gene_copynum_matrix = gene_depth_matrix*1.0/marker_coverages[None,:]
    #gene_copynum_matrix = numpy.clip(gene_copynum_matrix, 1e-09,1e09)

    # copynum is between 0.5 and 2
    is_present_copynum =
(gene_copynum_matrix>present_lower_threshold)*(gene_copynum_matrix<present_upper_threshold)
    is_absent_copynum = (gene_copynum_matrix<=absent_threshold)
    is_low_copynum = (gene_copynum_matrix<present_upper_threshold)



    # now size is about to get a lot bigger
    num_genes = gene_copynum_matrix.shape[0]
    num_samples = gene_copynum_matrix.shape[1]

    gene_hamming_matrix_gain = numpy.zeros((num_samples, num_samples))
    gene_hamming_matrix_loss = numpy.zeros((num_samples, num_samples))
    num_opportunities = numpy.zeros((num_samples, num_samples))

    # chunk it up into groups of 1000 genes
    chunk_size = 1000
```

```
    num_chunks = long(num_genes/chunk_size)+1
    for i in xrange(0, num_chunks):

        lower_gene_idx = i*chunk_size
        upper_gene_idx = min([(i+1)*chunk_size, num_genes])

        sub_gene_copynum_matrix = gene_copynum_matrix[lower_gene_idx:upper_gene_idx,:]
        sub_is_present_copynum = is_present_copynum[lower_gene_idx:upper_gene_idx,:]
        sub_is_absent_copynum = is_absent_copynum[lower_gene_idx:upper_gene_idx,:]
        sub_is_low_copynum = is_low_copynum[lower_gene_idx:upper_gene_idx,:]

        present_present = numpy.logical_and(sub_is_present_copynum[:,:,None],
sub_is_present_copynum[:,None,:])

        present_absent = numpy.logical_and(sub_is_present_copynum[:,:,None],
sub_is_absent_copynum[:,None,:])

        absent_present = numpy.logical_and(sub_is_absent_copynum[:,:,None],
sub_is_present_copynum[:,None,:])

        gene_hamming_matrix_gain += (absent_present).sum(axis=0)
        gene_hamming_matrix_loss += (present_absent).sum(axis=0)
        num_opportunities += (present_present+absent_present+present_absent).sum(axis=0)

    return gene_hamming_matrix_gain, gene_hamming_matrix_loss, num_opportunities


# Calculate polarized gene copynum changes from i to j that exceed threshold
# Returns list of differences. Each difference is a tuple of form
#
# (gene_name, (cov_i, marker_cov_i), (cov_j, marker_cov_j))
#
def calculate_gene_differences_between(i, j, gene_reads_matrix, gene_depth_matrix,
marker_coverages, absent_threshold=config.gainloss_max_absent_copynum,
present_lower_threshold=config.gainloss_min_normal_copynum, present_upper_threshold=
config.gainloss_max_normal_copynum):

    changed_genes = calculate_gene_differences_between_idxs(i, j, gene_reads_matrix,
gene_depth_matrix, marker_coverages, absent_threshold=absent_threshold,
present_lower_threshold=present_lower_threshold,
present_upper_threshold=present_upper_threshold)

    gene_differences = []
    if len(changed_genes>0):

        # Look at these two samples
        gene_depth_matrix = gene_depth_matrix[:,[i,j]]
        marker_coverages = marker_coverages[[i,j]]

        for gene_idx in changed_genes:

            gene_differences.append( (gene_idx, (gene_depth_matrix[gene_idx,0],
marker_coverages[0]), (gene_depth_matrix[gene_idx,1], marker_coverages[1])) )

    return gene_differences

def calculate_gene_differences_between_idxs(i, j, gene_reads_matrix, gene_depth_matrix,
marker_coverages, absent_threshold=config.gainloss_max_absent_copynum,
present_lower_threshold=config.gainloss_min_normal_copynum, present_upper_threshold=
config.gainloss_max_normal_copynum):
```

```
    # Look at these two samples
    gene_depth_matrix = gene_depth_matrix[:,[i,j]]
    marker_coverages = marker_coverages[[i,j]]

    #marker_coverages = numpy.clip(marker_coverages,1,1e09)
    #gene_copynum_matrix = numpy.clip(gene_depth_matrix,1,1e09)*1.0/marker_coverages[None,:]

    gene_copynum_matrix = gene_depth_matrix*1.0/marker_coverages[None,:]
    gene_differences = []

    # copynum is between 0.5 and 2
    is_present_copynum =
(gene_copynum_matrix>present_lower_threshold)*(gene_copynum_matrix<present_upper_threshold)
    is_absent_copynum = (gene_copynum_matrix<=absent_threshold)
    is_low_copynum = (gene_copynum_matrix<present_upper_threshold)

    present_present = numpy.logical_and(is_present_copynum[:,0], is_present_copynum[:,1])

    present_absent = numpy.logical_and(is_present_copynum[:,0], is_absent_copynum[:,1])

    absent_present = numpy.logical_and(is_absent_copynum[:,0], is_present_copynum[:,1])

    changed_genes = numpy.nonzero(numpy.logical_or(present_absent, absent_present))[0]

    return changed_genes



def calculate_triplet_gene_copynums(gene_depth_matrix, marker_coverages, i, j, k,
absent_threshold=config.gainloss_max_absent_copynum,
present_lower_threshold=config.gainloss_min_normal_copynum, present_upper_threshold=
config.gainloss_max_normal_copynum):

    desired_samples = numpy.array([i, j, k])

    # Look at these three samples
    gene_depth_matrix = gene_depth_matrix[:,desired_samples]
    marker_coverages = marker_coverages[desired_samples]

    gene_copynum_matrix = gene_depth_matrix*1.0/marker_coverages[None,:]

    # copynum is between 0.5 and 2
    is_present_copynum =
(gene_copynum_matrix>present_lower_threshold)*(gene_copynum_matrix<present_upper_threshold)
    is_absent_copynum = (gene_copynum_matrix<=absent_threshold)
    is_low_copynum = (gene_copynum_matrix<present_upper_threshold)

    changed_idxs =
(is_low_copynum.all(axis=1))*(is_present_copynum.any(axis=1))*(is_absent_copynum.any(axis=1))*nu
mpy.logical_or(is_present_copynum[:,0], is_absent_copynum[:,0])

    copynum_trajectories = []
    if changed_idxs.sum() > 0:

        gene_copynum_matrix = gene_copynum_matrix[changed_idxs]

        for gene_idx in xrange(0,gene_copynum_matrix.shape[0]):
            copynum_trajectories.append(gene_copynum_matrix[gene_idx,:])

    return copynum_trajectories
```

```
##############################################################################
#
#  return a complete list of prevalences including all genes in the pangenome
#
##############################################################################
def gene_prevalences_whole_pangenome(gene_names, gene_names_subset, prevalences):

    # first make a dictionary of gene names and prevalences:
    prevalence_dict={}
    for i in range(0, len(gene_names_subset)):
        gene=gene_names_subset[i]
        prevalence_dict[gene]=prevalences[i]

    gene_prevalences=[]
    for gene in gene_names:
        if gene in prevalence_dict.keys():
            gene_prevalences.append(prevalence_dict[gene])
        else:
            gene_prevalences.append(0)

    return numpy.asarray(gene_prevalences)


##########################################
#                                        #
#  return a histogram  of kegg pathway IDs #
#                                        #
##########################################

def kegg_pathways_histogram(kegg_ids, gene_names, gene_samples,gene_prevalences=[],
spgenes=False):

    import pandas

    # if no gene_prevalences are provided, assume that every gene is in every person.
    if len(gene_prevalences)==0:

gene_prevalences=numpy.repeat(len(gene_samples),len(gene_names))/float(len(gene_samples))
    else:
        gene_prevalences=gene_prevalences/float(len(gene_samples))


    pathway_histogram={}
    pathway_description={}
    gene_idx=0
    for gene in gene_names:
        if gene in kegg_ids.keys():
            prevalence=gene_prevalences[gene_idx]
            pathways=kegg_ids[gene]
            for i in range(0, len(pathways)):
                pathway=pathways[i][0]
                description=pathways[i][1]
                if pathway not in pathway_histogram.keys():
                    pathway_histogram[pathway]=[prevalence]
                else:
                    pathway_histogram[pathway].append(prevalence)
                if spgenes==True:
                    pathway_description[pathway]=pathway
                else:
                    pathway_description[pathway]=description
```

```
            gene_idx +=1

    # create different prevalence bins for stacked histograms [100%, <0x<100, 0%]
    bins = numpy.asarray([0,0.1,0.5,0.9,1.0])
    pathway_counts_list={}
    for val in range(0, len(bins)): # the last value will be the total
        pathway_counts_list[val]=[]
    pathway_description_list=[]

    for pathway in pathway_histogram.keys():
        counts,dummy=numpy.histogram(pathway_histogram[pathway],bins)
        if pathway != '':
            pathway_description_list.append(pathway_description[pathway])
            for val in range(0, len(bins)-1):
                pathway_counts_list[val].append(counts[val])
            pathway_counts_list[len(bins)-1].append(sum(counts))

    # convert to dataframe:
    kegg_df={'total':pathway_counts_list[len(bins)-1],'names':pathway_description_list}
    for val in range(0, len(bins)-1):
        kegg_df[bins[val]]=pathway_counts_list[val]

    kegg_df=pandas.DataFrame(kegg_df)

#    sorted_kegg_df=pandas.DataFrame.sort(kegg_df, columns='total')
#    pathway_counts_list=sorted_kegg_df['counts'].tolist()
#    pathway_description_list=sorted_kegg_df['names'].tolist()

#    return pathway_counts_list, pathway_description_list

    return kegg_df, pathway_description_list


def calculate_gene_error_rate(i, j, gene_reads_matrix, gene_depth_matrix, marker_coverages,
absent_thresholds=[config.gainloss_max_absent_copynum],
present_lower_threshold=config.gainloss_min_normal_copynum, present_upper_threshold=
config.gainloss_max_normal_copynum):

    # Get reads, depths, and marker coverages in sample 1
    N1s = gene_reads_matrix[:,i]
    D1s = gene_depth_matrix[:,i]
    Dm1 = marker_coverages[i]

    # Get reads, depths, and marker coverages in sample 2
    N2s = gene_reads_matrix[:,j]
    D2s = gene_depth_matrix[:,j]
    Dm2 = marker_coverages[j]

    # Calculate gene copy numbers in both samples
    C1s = D1s*1.0/Dm1
    C2s = D2s*1.0/Dm2

    # Get list of genes that are in "normal" range in both samples
    good_idxs_1 = (C1s>=present_lower_threshold)*(C1s<=present_upper_threshold)
    good_idxs_2 = (C2s>=present_lower_threshold)*(C2s<=present_upper_threshold)


    # Calculate empirical prior distribution p(l) from SI 3.5.
    # For convenience, length_factor = 1/l
    length_factors = []
    if good_idxs_1.sum() > 0:
```

```
        length_factors.extend( D1s[good_idxs_1]*1.0/N1s[good_idxs_1] )
    if good_idxs_2.sum() > 0:
        length_factors.extend( D2s[good_idxs_2]*1.0/N2s[good_idxs_2] )
    length_factors = numpy.array(length_factors)
    # Will use whole array as prior distribution

    # Calculate empirical prior distribution p(c) from SI 3.5
    copynum_bins = numpy.linspace(0,2,21)
    Cs = copynum_bins[1:]-(copynum_bins[1]-copynum_bins[0])/2
    copynum_bins[0] = -1 # Just to make sure we include things with zero copynum
    copynum_bins[-1] = 1e09 # Assume things with c>2 have copynum 2 (conservative for detecting
changes to lower values)

    # Prior distribution p(c)
    pCs = numpy.histogram(numpy.hstack([C1s,C2s]), bins=copynum_bins)[0]
    pCs = pCs*1.0/pCs.sum()

    # Vectors for matrix calculation
    C_lowers = numpy.ones_like(Cs)*present_lower_threshold
    C_uppers = numpy.ones_like(Cs)*present_upper_threshold

    # The parameter of the poisson distribution of reads in sample 1
    Navg1s = Cs[:,None]*Dm1/length_factors[None,:]
    # Lower limit of number of reads for normal range
    Nlower1s = C_lowers[:,None]*Dm1/length_factors[None,:]
    # Upper limit of number of reads for normal range
    Nupper1s = C_uppers[:,None]*Dm1/length_factors[None,:]

    # Same for sample 2
    Navg2s = Cs[:,None]*Dm2/length_factors[None,:]
    Nlower2s = C_lowers[:,None]*Dm2/length_factors[None,:]
    Nupper2s = C_uppers[:,None]*Dm2/length_factors[None,:]

    perrs = []
    for absent_threshold in absent_thresholds:

        C_absents = numpy.ones_like(Cs)*absent_threshold

        # Upper limit of number of reads for "absent" range
        Nabsent1s = C_absents[:,None]*Dm1/length_factors[None,:]
        Nabsent2s = C_absents[:,None]*Dm2/length_factors[None,:]

        perr = 0
        perr += (poisson.cdf(Nabsent1s, Navg1s)*(poisson.cdf(Nupper2s,Navg2s)-
poisson.cdf(Nlower2s,Navg2s))*pCs[:,None]).sum()
        perr += (poisson.cdf(Nabsent2s, Navg2s)*(poisson.cdf(Nupper1s,Navg1s)-
poisson.cdf(Nlower1s,Navg1s))*pCs[:,None]).sum()
        # To emulate the integral over p(l)
        perr = perr/len(length_factors)

        # BG: 5/23/18. This is now done outside this function.
        # To add up all the genes
        #perr = perr*len(C1s)

        perrs.append(perr)

    return numpy.array(perrs)


# Fuzzy matching of nearby genes
def is_nearby(gene_change_1, gene_change_2):
```

```
    gene_name_1 = gene_change_1[0]
    gene_name_2 = gene_change_2[0]

    gene_items_1 = gene_name_1.split(".")
    gene_items_2 = gene_name_2.split(".")

    genome_1 = ".".join([gene_items_1[0],gene_items_1[1]])
    gene_number_1 = long(gene_items_1[-1])
    genome_2 = ".".join([gene_items_2[0],gene_items_2[1]])
    gene_number_2 = long(gene_items_2[-1])

    if genome_1==genome_2:
        if fabs(gene_number_1-gene_number_2) < 6:
            return True
    else:
        return False

def get_nearby_gene_idxs(gene_names, gene_idx, spacing=1, skip_target_gene=True):

    gene_name = gene_names[gene_idx]

    gene_items = gene_name.split(".")
    gene_id = long(gene_items[-1])


    idxs = []

    for i in xrange(-spacing,spacing+1):

        if skip_target_gene==True and i==0:
            continue

        gene_name = ".".join(gene_items[:-1]+[str(gene_id+i)])

        if gene_name in gene_names:
            idxs.append(gene_names.index(gene_name))

    return idxs

# Tries to merge nearby gene differences into blocks
def merge_nearby_gene_differences(gene_differences):

    blocks = []

    for new_difference in gene_differences:
        print new_difference[0]
        matched=False
        for block_idx in xrange(0,len(blocks)):
            for old_difference in blocks[block_idx]:
                if is_nearby(new_difference, old_difference):
                    matched=True
                    break
            if matched:
                blocks[block_idx].append(new_difference)
                break

        if not matched:
            blocks.append([new_difference])

    print len(gene_differences), len(blocks)
```

```
    return blocks
```

midas_db_utils.py

```
import config
import gzip, os

midas_dir = config.midas_directory

def get_ref_genome_ids(desired_species_name):
       genome_ids=[]
       genome_info = open("%sgenome_info.txt" % midas_dir)
       genome_info.readline() #header
       for line in genome_info:
                    items = line.split("\t")
                    genome_id = items[0].strip()
                    species_id=items[5].strip()
                    if desired_species_name == species_id:
                                 genome_ids.append(genome_id)
       return genome_ids

# ========================================================================
# Loads set of genes in the MIDAS reference genome for a given species
# ========================================================================

def load_reference_genes(species_name):

              rep_genome_filename = "%s/rep_genomes/%s/genome.features.gz" %
(midas_dir, species_name)
              file = gzip.open(rep_genome_filename, 'r')
              file.readline() # header

              reference_genes = []

              for line in file:
                           items = line.split()
                           gene_name = items[0].strip()
                           reference_genes.append(gene_name)

              file.close()

              return set(reference_genes)

# ========================================================================
# Returns number of non-redundant genes in MIDAS pangenome for given species
# ========================================================================

def get_reference_genome_size(species_name):
       return len(load_reference_genes(species_name))

# ========================================================================
# Loads MIDAS pangenome map (genome -> gene -> centroids) for a given species
# ========================================================================

def get_pangenome_map(species_name):

              gene_info_filename = '%s/pan_genomes/%s/gene_info.txt.gz' %
(midas_dir, species_name)
              file = gzip.open(gene_info_filename, 'r')
```

```
		file.readline() # header

		pangenome_map = {}

		for line in file:
				items = line.split("\t")
				gene_id = items[0].strip()
				genome_id = items[1].strip()
				centroid_99 = items[2].strip()
				centroid_95 = items[3].strip()

				if genome_id not in pangenome_map:
						pangenome_map[genome_id] = {}

				pangenome_map[genome_id][gene_id] = (centroid_99,
centroid_95)

		file.close()

		return pangenome_map

# ============================================================================
# Loads set of non-redundant genes in MIDAS pangenome for given species
# where genes in same 99% identity cluster are considered redundant
# ============================================================================

def load_pangenome_genes(species_name):

	pangenome_map = get_pangenome_map(species_name)

	non_redundant_genes = set()

	for genome in pangenome_map:
		for gene in pangenome_map[genome]:
			centroid_99 = pangenome_map[genome][gene][0]
			non_redundant_genes.add(centroid_99)

	return non_redundant_genes

# ============================================================================
# Returns number of non-redundant genes in MIDAS pangenome for given species
# where genes in same 99% identity cluster are considered redundant
# ============================================================================

def get_pangenome_size(species_name):
	return len(load_pangenome_genes(species_name))

# ============================================================================
# Returns number of genomes in MIDAS pangenome for a given species
# ============================================================================

def get_number_of_genomes(species_name):
		return len(get_pangenome_map(species_name))

# ============================================================================
# Returns list of MIDAS species (5926 species for midas_db_v1.2)
# ============================================================================

def parse_species_list():

		species_directories = os.listdir("%s/pan_genomes" % midas_dir)
```

```
            species_names = []
            for potential_species_name in species_directories:
                        if not potential_species_name.startswith('.'):

      species_names.append(potential_species_name)

            return species_names

# ============================================================================
# The gene_ids in the pangenome list are the centroids of gene clusters.
# Sometimes the gene in the reference genome is not chosen as the centroid.
# This function creates a map between pangenome_centroids and genes in
# reference genome (if it exists -- otherwise map to itself)
# ============================================================================

def load_centroid_gene_map(species_name = None):

            if species_name == None:
                        import parse_midas_data
                        desired_species =
parse_midas_data.parse_good_species_list()
            else:
                        desired_species = [species_name]

            for species_name in desired_species:

                        # First load reference genes
                        reference_genes = load_reference_genes(species_name)

                        # Next load pangenome centroids
                        gene_info_filename =
'%s/pan_genomes/%s/gene_info.txt.gz' % (midas_dir, species_name)
                        gene_info_file = gzip.open(gene_info_filename, 'r')
                        gene_info_file.readline() # header

                        centroid_gene_map = {}

                        for line in gene_info_file:

                                    items = line.split("\t")
                                    gene_id = items[0].strip()
                                    centroid_id = items[3].strip() # 95%
centroid

                                    if centroid_id not in centroid_gene_map:
                                                centroid_gene_map[centroid_id]
= centroid_id

                                    if (gene_id in reference_genes) and
(centroid_id not in reference_genes):
                                                centroid_gene_map[centroid_id]
= gene_id

                        gene_info_file.close()

            return centroid_gene_map

# ============================================================================
# Returns set of genes in the MIDAS pangenome of a given species
# which are shared with other species (plus other shared genes which
```

```
# the given species may not have.)
#
# The purpose is to ignore genes which have >= 95% sequence identity with
# at least one other gene in a different species' pangenome.
# ============================================================================

def parse_midas_shared_genes(desired_species):

            midas_shared_genes = set()

            # Get mapping of pangenome centroids to reference genes
            centroid_gene_map = load_centroid_gene_map(desired_species)

            midas_db_shared_gene_filename = "%s/cross_species_centroids.txt.gz" %
midas_dir
            file = gzip.open(midas_db_shared_gene_filename, "r")
            for line in file:
                        items = line.split()

                        # One name for a shared gene
                        big_centroid = items[0].strip()
                        midas_shared_genes.add(big_centroid)

                        # Check if this species has any gene which is shared
                        # If so, add it to the list of shared genes
                        other_centroids = items[1].strip().split(",")
                        for centroid in other_centroids:
                                    if centroid in centroid_gene_map:
                                                # Specifically, add the
reference gene if possible

      midas_shared_genes.add(centroid_gene_map[centroid])

            return midas_shared_genes

# ============================================================================
# Returns map: genus / family / order -> phylum
# ============================================================================

def load_gfo_phylum_map():

      genus_phylum_map = {}

      taxonomy_file = open("%s/genome_taxonomy.txt" % (midas_dir), 'r')
      taxonomy_file.readline() # remove header

      for line in taxonomy_file:
            genome_id, genome_name, taxon_id, kingdom, phylum, class_, order,
family, genus, species, _, _ = line.strip('\n').split('\t')
            # Manually correct for Bilophila
            if genus == 'Bilophila':
                  genus_phylum_map[genus] = 'Proteobacteria'
            else:
                  genus_phylum_map[genus] = phylum
            genus_phylum_map[family] = phylum
            genus_phylum_map[order] = phylum

      taxonomy_file.close()
      return genus_phylum_map

# ============================================================================
```

```
# Outputs information on number of genomes per species, why not?
# ============================================================================

if __name__=='__main__':

            import parse_midas_data
            good_species_list = parse_midas_data.parse_good_species_list()
            for species_name in good_species_list:
                        num_genomes = get_number_of_genomes(species_name)
                        print("%s: %i" % (species_name, num_genomes))
```

parse_midas_data.py

```
import config, sample_utils as su, stats_utils, gene_diversity_utils,
midas_db_utils
import sys, bz2, gzip, os.path
import numpy
from math import floor, ceil
from collections import defaultdict

# ============================================================================
# Set up default source and output directories, shorthand
# ============================================================================

data_dir = config.data_directory
midas_dir = config.midas_directory
metadata_dir = config.metadata_directory

# We use this one to debug because it was the first one we looked at
debug_species_name = config.debug_species_name

# ============================================================================
# METHODS FOR PARSING SPECIES METADATA
#            - parse_species_list
#            - parse_depth_sorted_species_list
#            - parse_good_species_list
#            - load_pickled_good_species_list
# ============================================================================

# ============================================================================
# Returns a list of all species that MIDAS called SNPS for
# ============================================================================

def parse_species_list():
      with open("%s/snps/species_snps.txt" % data_dir, 'r') as file:
            species_names = [line.strip() for line in file]
      return species_names

# ============================================================================
# Returns a list of all species that MIDAS called SNPS for, sorted in order
# of decreasing total sequencing depth [based on marker genes]
# ============================================================================

def parse_depth_sorted_species_list():
      return parse_global_marker_gene_coverages()[2]

# ============================================================================
# Returns a list of all species that MIDAS called SNPS for
# that passed a certain depth / prevalence requirement,
# again sorted in order of decreasing total sequencing depth
# ============================================================================
```

```
def parse_good_species_list(min_marker_coverage = config.good_species_min_coverage,

              min_prevalence = config.good_species_min_prevalence):
       good_species_list = []

       species_coverage_matrix, samples, species =
parse_global_marker_gene_coverages()

       for i in range(len(species)):
              species_coverages = species_coverage_matrix[i,:]
              # Number of samples whose marker gene coverage exceeds a threshold
              # must exceed a prevalence threshold
              # Here: at least 10 samples have marker gene coverage >= 10
              if (species_coverages>=min_marker_coverage).sum() >= min_prevalence:
                     good_species_list.append(species[i])

       return good_species_list

# ====================================================================
# Returns good_species_list as defined above, but from pickle file
# ====================================================================

def load_pickled_good_species_list():

       import pickle
       pickle_path = '%s/pickles/good_species_list.pkl' % data_dir

       if os.path.isfile(pickle_path):
              return pickle.load(open(pickle_path, 'rb'))
       else:
              good_species_list = parse_good_species_list()
              pickle.dump(good_species_list, open(pickle_path, 'wb'))
              return good_species_list

# ====================================================================
# METHODS FOR PARSING COVERAGE OF DIFFERENT SPECIES ACROSS SAMPLES
#             - parse_global_marker_gene_coverages()
#             - parse_species_marker_gene_coverages(species)
#             - parse_marker_gene_coverage_distribution(species)
#             - parse_gene_coverages(species)
#             - parse_coverage_distribution(species)
#             - parse_sample_coverage_map(species)
# ====================================================================

# ====================================================================
# Loads marker gene coverages produced by MIDAS
# for all species in which SNPs were called
#
# Returns: species-by-sample matrix of marker gene coverages,
#                                  with species sorted in descending order of total
coverage;
#                                  ordered list of sample ids; ordered list of
species names
# ====================================================================

def parse_global_marker_gene_coverages():

       desired_species_names = set(parse_species_list())

       # coverage.txt: average read-depth of 15 marker genes per species
```

```
	# (total bp of mapped reads/total bp of 15 marker-genes)

	file = bz2.BZ2File("%s/species/coverage.txt.bz2" % data_dir, 'r')
	samples = file.readline().strip().split()[1:] # header
	samples = su.parse_merged_sample_names(samples) # remove c's

	species = [] # list of species
	species_coverage_matrix = [] # rows - species, cols - samples

	for line in file:
		items = line.strip().split()
		species_name = items[0]
		coverages = numpy.array([float(cov) for cov in items[1:]])

		if species_name in desired_species_names:
			species.append(species_name)
			species_coverage_matrix.append(coverages)

	file.close()

	# Sort by marker gene coverage, summed across all samples, from high to low

	species, species_coverage_matrix = zip(*sorted(zip(species,
species_coverage_matrix), key=lambda pair: pair[1].sum(), reverse=True))

	species_coverage_matrix = numpy.array(species_coverage_matrix)

	return species_coverage_matrix, samples, species

# ================================================================================
# Loads marker gene coverages produced by MIDAS for a particular species
# Returns: list of average marker coverages, list of samples
# ================================================================================

def parse_species_marker_gene_coverages(desired_species_name):

	species_coverage_matrix, samples, species =
parse_global_marker_gene_coverages()

	try:
		species_idx = species.index(desired_species_name)
		return species_coverage_matrix[species_idx,:], samples
	except:
		return None # desired species name not found

# ================================================================================
# Loads site coverages within marker genes for a particular species
# Returns dictionary: sample -> marker gene -> (location, coverage)
#
# Note that marker_coverage_distribution.txt
# is from calculate_marker_coverage_distribution
# ================================================================================

def parse_marker_gene_coverage_distribution(desired_species_name):

	coverage_distribution_file =
bz2.BZ2File("%s/snps/%s/marker_coverage_distribution.txt.bz2" % (data_dir,
desired_species_name))
	coverage_distribution_file.readline() # header

	# Dictionary: sample -> marker gene -> (location, coverage) list
```

```
	marker_cov_dict = defaultdict(dict)

	for line in coverage_distribution_file:
		items = line.strip().split("\t")
		sample, gene_name = items[0].split(",")
		sample = sample[:-1] if (sample[-1] == 'c') else sample # remove c

		locations, coverages = [], []

		for item in items[1:]:
			loc, cov = item.split(",")
			locations.append(long(loc))
			coverages.append(float(cov))

		marker_cov_dict[sample][gene_name] = (numpy.array(locations),
numpy.array(coverages))

	return marker_cov_dict

# ============================================================================
# Loads gene coverages for a particular species (gene_name -> avg coverage)
#
# Recall how gene_coverage.txt was generated:
# Average depth, i.e. sum of depths across all sites with known variant
# type divided by the number of such sites, is reported for each gene
# in each sample (so long some sample-gene has at least one such site).
# ============================================================================

def parse_gene_coverages(desired_species_name):

	gene_cov_file = bz2.BZ2File("%s/snps/%s/gene_coverage.txt.bz2" % (data_dir,
desired_species_name))

	samples = gene_cov_file.readline().strip().split()[1:] # header
	samples = su.parse_merged_sample_names(samples) # remove c's

	# Dictionary: pangenome gene -> average depth in each sample
	gene_coverages = {}

	for line in gene_cov_file:
		items = line.split()
		gene_name = items[0]
		depths = numpy.array([float(item) for item in items[1:]])
		gene_coverages[gene_name] = depths

	return gene_coverages, samples

# ============================================================================
# Recall how the site coverage distribution files were generated:
#
#		full_coverage_distribution.txt: for each sample, consider all sites
#		with known variant type (1D, 2D, 3D, or 4D) and output number of sites
#		which have each depth genome-wide
#
#		coverage_distribution.txt: consider only sites with known variant type
#		and that have coverage >= 3 in at least 95% of all samples
#		cohort options: infant, adult, hmp, any
#
# This function loads coverage histogram (dictionary: depth -> site count)
# for each sample for a given species.
# ============================================================================
```

```
def parse_coverage_distribution(desired_species_name, prevalence_filter = True,
remove_c = True, prev_cohort='any'):

	if prev_cohort in ['infant', 'adult', 'hmp']:
		prev_cohort_str = 'prev_' + prev_cohort
	else:
		prev_cohort_str = prev_cohort

	full_str = "" if prevalence_filter else "full_"
	cov_dist_file = bz2.BZ2File("%s/snps/%s/%scoverage_distribution_%s.txt.bz2" %
(data_dir, desired_species_name, full_str, prev_cohort_str))

	cov_dist_file.readline() # remove header
	samples, sample_coverage_histograms = [], []
	for line in cov_dist_file:
		items = line.strip().split()
		sample_coverage_histogram = {}
		for item in items[1:]:
			subitems = item.split(",")
			sample_coverage_histogram[float(subitems[0])] =
float(subitems[1])
		sample_coverage_histograms.append(sample_coverage_histogram)
		samples.append(items[0])

	if remove_c == True:
		samples = su.parse_merged_sample_names(samples)

	return sample_coverage_histograms, samples

# ============================================================================
# Returns median site coverage (i.e. half of all sites have same or lower)
# for each sample in a dictionary (sample -> median_coverage)
# ============================================================================

def parse_sample_coverage_map(species, prev_cohort='any'):

	sample_coverage_histograms, samples = parse_coverage_distribution(species,
prev_cohort=prev_cohort)

	sample_coverage_map = {}

	for hist, sample in zip(sample_coverage_histograms, samples):
		median_coverage =
stats_utils.calculate_nonzero_median_from_histogram(hist)
		sample_coverage_map[sample] = median_coverage

	return sample_coverage_map

# ============================================================================
# METHODS FOR LOADING SETS OF GENES FOR A SPECIES
# (technically, these should be in midas_db_utils because they do not
# depend on a particular dataset. But I'll keep these here anyways...)
#
#		- load_pangenome_genes
#		- load_reference_genes
#		- load_marker_genes
#		- load_core_genes
# ============================================================================

# ============================================================================
```

```
# Loads MIDAS pangenome (after clustering at 95% identity) for given species
# Returns a complete set of genes (irrespective of prevalence)
# ============================================================================

def load_pangenome_genes(species_name):

	# Open MIDAS output: presence absence calls for all pangenome centroids
	gene_presabs_file =  bz2.BZ2File("%s/genes/%s/genes_presabs.txt.bz2" %
(data_dir, species_name), 'r')
	gene_presabs_file.readline() # remove header
	genes = [line.split()[0] for line in gene_presabs_file]

	# Maps original centroid to reference-corrected centroid
	centroid_gene_map = midas_db_utils.load_centroid_gene_map(species_name)
	ref_corrected_genes = [centroid_gene_map[c] for c in genes]

	return set(genes), set(ref_corrected_genes)

# ============================================================================
# Loads list of genes in reference genome used by MIDAS for a given species
# ============================================================================

def load_reference_genes(desired_species_name):

	features_file = gzip.open("%s/rep_genomes/%s/genome.features.gz" %
(midas_dir, desired_species_name), 'r')
	features_file.readline() # remove header
	reference_genes = [line.split()[0] for line in features_file]
	return set(reference_genes)

# ============================================================================
# Loads list of MIDAS marker genes for a given species
# (by default, load only those 15 IDs that are in the reference genome)
# ============================================================================

def load_marker_genes(desired_species_name, require_in_reference_genome=True):

	# Chosen markers (From table S7 of Nayfach et al (Genome Res 2016)
	marker_ids = set(['B000032', 'B000039', 'B000041', 'B000062', 'B000063',
'B000065', 'B000071', 'B000079', 'B000080', 'B000081', 'B000082', 'B000086',
'B000096', 'B000103', 'B000114'])

	# Get list of reference genome genes
	reference_genes = set(load_reference_genes(desired_species_name))

	# Maps marker ID to gene name
	marker_gene_map = {marker_id : [] for marker_id in marker_ids}
	marker_genes = []

	# MIDAS marker gene database: gene ID, rep(?) genome ID, marker ID
	# Note that multiple gene IDs may correspond to the same marker gene
	marker_gene_file = open("%s/marker_genes/phyeco.map" % (midas_dir), 'r')
	marker_gene_file.readline() # header

	for line in marker_gene_file:
		gene_name, gene_len, genome_id, species_name, marker_id =
line.strip().split("\t")

		# If you only want marker genes in the reference genome,
		# the gene name must be in reference genes
```

```
            if species_name == desired_species_name and marker_id in marker_ids
and ((not require_in_reference_genome) or (gene_name in reference_genes)):
                  marker_genes.append(gene_name)
                  marker_gene_map[marker_id].append(gene_name)

      return set(marker_genes)

# ====================================================================
# Loads a subset of "core" genes
# prev_cohort: adult, hmp, infant, all, any
# *Deprecated: Use core_gene_utils.parse_core_genes instead
# ====================================================================

def load_core_genes(desired_species_name, prev_cohort='all'):
      import core_gene_utils
      return core_gene_utils.parse_core_genes(desired_species_name, prev_cohort)

# ====================================================================
# METHODS THAT USE MIDAS INTERMEDIATE FILES
# ====================================================================

# ====================================================================
# Returns read counts of allowed genes for each genome
# in each sample-species pair; uses MIDAS intermediate files
# ====================================================================

def parse_99_percent_genes(species, samples, allowed_genes=[]):

      # Dictionary: sample -> gene -> read count
      data = defaultdict(dict)

      # Assume samples are unmerged
      for sample in samples:
            int_genes_output_dir = "%s/%s/genes/output" %
(config.int_data_directory, sample)
            try:
                  file = gzip.open("%s/%s.genes.gz" % (int_genes_output_dir,
species), 'r')
                  file.readline() # remove header
                  for line in file:
                        gene, count_reads, coverage, copynum =
line.strip().split('\t')
                        if gene in allowed_genes:
                              data[sample][gene] = int(count_reads)
            except:
                  print("%s has no genes output for sample %s" % (species,
sample))

      # Dictionary: gene -> list of read counts, ordered by sample
      data_numpy_array_dict = defaultdict(list)

      for gene in allowed_genes:
            for sample in samples:
                  read_count = data[sample][gene] if (gene in data[sample]) else
0
                  data_numpy_array_dict[gene].append(read_count)
            data_numpy_array_dict[gene] =
numpy.asarray(data_numpy_array_dict[gene])

      # Dictionary: ref_genome -> list of read counts, ordered by gene, sample
      ref_genome_dict = defaultdict(list)
```

```
	for gene in allowed_genes:
		ref_genome='.'.join(gene.split('.')[0:2])
		ref_genome_dict[ref_genome] += data_numpy_array_dict[gene]

	return ref_genome_dict

# ========================================================================
# Parse MIDAS intermediate species file to get a list of species
# at least 3x coverage
# ========================================================================

def parse_intermediate_species_file(sample_id, inFN):
	inFile = open(inFN,'r')
	inFile.readline() # remove header

	species_list=[]

	for line in inFile:
		items=line.strip().split('\t')
		species_id=items[0]
		coverage=float(items[2])
		if coverage >=3.0:
			species_list.append(species_id)

	return set(species_list)

# ========================================================================
# COMPLICATED POSTPROCESSING METHODS...
# ========================================================================

def calculate_relative_depth_threshold_map(sample_coverage_histograms, samples,
min_nonzero_median_coverage=5, lower_factor=0.5, upper_factor=2):

	# returns map of sample name: coverage threshold
	# essentially filtering out samples whose marker depth coverage
	# does not exceed the average coverage threshold

	depth_threshold_map = {}
	for i in xrange(0,len(samples)):

		# Check if coverage distribution meets certain requirements
		is_bad_coverage_distribution = False

		# First check if passes median coverage requirement
		nonzero_median_coverage =
stats_utils.calculate_nonzero_median_from_histogram(sample_coverage_histograms[i])
		if round(nonzero_median_coverage) < min_nonzero_median_coverage:
				is_bad_coverage_distribution=True

		# Passed median coverage requirement
		# Now check whether a significant number of sites fall between lower
and upper factor.
		lower_depth_threshold = floor(nonzero_median_coverage*lower_factor)-
0.5 # why is 0.5 being added/subtracted? NRG
		upper_depth_threshold = ceil(nonzero_median_coverage*upper_factor)+0.5

		depths, depth_CDF =
stats_utils.calculate_CDF_from_histogram(sample_coverage_histograms[i])
		# remove zeros
		if depths[0]<0.5:
				depth_CDF -= depth_CDF[0]
```

```
                        depth_CDF /= depth_CDF[-1]

            fraction_in_good_range =
depth_CDF[(depths>lower_depth_threshold)*(depths<upper_depth_threshold)].sum()

            if fraction_in_good_range < 0.6: #where does 0.6 come from? NRG
                        is_bad_coverage_distribution=True

            if is_bad_coverage_distribution:
                        lower_depth_threshold = 1000000001
                        upper_depth_threshold = 1000000001

            depth_threshold_map[samples[i]] = (lower_depth_threshold,
upper_depth_threshold)

      return depth_threshold_map

# ====================================================================
# Reads midas output and prints to stdout in a format
# suitable for further downstream processing
#
# In the process, filters sites that fail to meet the depth requirements
# ====================================================================

def pipe_snps(species_name,
min_nonzero_median_coverage=config.pipe_snps_min_nonzero_median_coverage,
lower_factor=config.pipe_snps_lower_depth_factor,
upper_factor=config.pipe_snps_upper_depth_factor,
min_samples=config.pipe_snps_min_samples, debug=False):

      # lower_factor = 0.3 is the default to be consistent with MIDAS gene presence
criterion
      # upper factor = 3 is the default for (logarithmic) symmetry
      # min_samples=4 is the default because then the site is guaranteed to be
present in
      # at least 2 independent people.
      # NRG: Why is a site guaranteed to be in at least 2 independent people?
      # BG: In our cohort, the maximum number of samples per person is 3.
      #                   If there are 4 samples, then they must be spread across
at least 2 people.

      # Load genomic coverage distributions
      sample_coverage_histograms, sample_list =
parse_coverage_distribution(species_name, remove_c=False)

      # depth threshold map returns the lower and upper depth values that are
0.3*median and 3*median depth in the data.
      depth_threshold_map =
calculate_relative_depth_threshold_map(sample_coverage_histograms, sample_list,
min_nonzero_median_coverage, lower_factor, upper_factor)



      # Open MIDAS output files
      ref_freq_file = bz2.BZ2File("%s/snps/%s/snps_ref_freq.txt.bz2" % (data_dir,
species_name),"r")
      depth_file = bz2.BZ2File("%s/snps/%s/snps_depth.txt.bz2" % (data_dir,
species_name),"r")
      alt_allele_file = bz2.BZ2File("%s/snps/%s/snps_alt_allele.txt.bz2" %
(data_dir, species_name),"r")
      info_file = bz2.BZ2File("%s/snps/%s/snps_info.txt.bz2" % (data_dir,
species_name),"r")
```

```
	marker_file = bz2.BZ2File("%s/snps/%s/marker_coverage.txt.bz2" % (data_dir,
species_name))

	# get header lines from each file
	depth_line = depth_file.readline()
	ref_freq_line = ref_freq_file.readline()
	alt_line = alt_allele_file.readline()
	info_line = info_file.readline()
	marker_line = marker_file.readline()

	# get list of samples
	depth_items = depth_line.split()
	samples = numpy.array(depth_items[1:])

	# BHG (06/24/17) removed this so that all "raw" data have "c"s.
	# All functions that write something keep them.
	# All loading functions strip them.
	#samples= parse_merged_sample_names(samples) # NRG (06/06/17): I added this
so that the keys in dictionary are compatible.

	# samples
	prevalence_threshold = min([min_samples*1.0/len(samples), 0.5])

	# create depth threshold vector from depth threshold map
	lower_depth_threshold_vector = []
	upper_depth_threshold_vector = []
	for sample in samples:

	lower_depth_threshold_vector.append(depth_threshold_map[sample][0])

	upper_depth_threshold_vector.append(depth_threshold_map[sample][1])

	lower_depth_threshold_vector = numpy.array(lower_depth_threshold_vector)
	upper_depth_threshold_vector = numpy.array(upper_depth_threshold_vector)

	# Figure out which samples passed our avg_depth_threshold
	passed_samples = (lower_depth_threshold_vector<1e09) #1e09 comes from the
calculate_relative_depth_threshold_map definition above, which is a code for a bad
sample. A bad sample has median depth less than 5 or greater than 0.6 fraction of
the genome is outside the acceptable range of good depths.
	total_passed_samples = passed_samples.sum()

	# Let's focus on those from now on
	samples = list(samples[passed_samples])
	lower_depth_threshold_vector = lower_depth_threshold_vector[passed_samples]
	upper_depth_threshold_vector = upper_depth_threshold_vector[passed_samples]

	#print lower_depth_threshold_vector

	# print header
	print_str = "\t".join(["site_id"]+samples)
	print print_str

	# Only going to look at 1D, 2D, 3D, and 4D sites
	# (we will restrict to 1D and 4D downstream)
	allowed_variant_types = set(['1D','2D','3D','4D'])

	allele_counts_syn = [] # alt and reference allele counts at 4D synonymous
sites with snps
	locations_syn = [] # genomic location of 4D synonymous sites with snps
	genes_syn = [] # gene name of 4D synonymous sites with snps
```

```
	passed_sites_syn = numpy.zeros(len(samples))*1.0

	allele_counts_non = [] # alt and reference allele counts at 1D nonsynonymous
sites with snps
	locations_non = [] # genomic location of 1D nonsynonymous sites
	genes_non = [] # gene name of 1D nonsynonymous sites with snps
	passed_sites_non = numpy.zeros_like(passed_sites_syn)

	num_sites_processed = 0
	while True:

			# load next lines
			depth_line = depth_file.readline()
			ref_freq_line = ref_freq_file.readline()
			alt_line = alt_allele_file.readline()
			info_line = info_file.readline()

			# quit if file has ended
			if depth_line=="":
				break

			# parse site info
			info_items = info_line.split("\t")
			variant_type = info_items[5]

			# make sure it is either a 1D or 4D site
			if not variant_type in allowed_variant_types:
				continue

			# continue parsing site info
			gene_name = info_items[6]
			site_id_items = info_items[0].split("|")
			# NRG: added this if condition to deal with extra 'accn' in db
swap.
			if site_id_items[0]=='accn':
				contig = site_id_items[1]
				location = site_id_items[2]
			else:
				contig = site_id_items[0]
				location = site_id_items[1]


			# now parse allele count info
			depths = numpy.array([float(item) for item in
depth_line.split()[1:]])
			ref_freqs = numpy.array([float(item) for item in
ref_freq_line.split()[1:]])

			depths = depths[passed_samples]
			ref_freqs = ref_freqs[passed_samples]

			#if (ref_freqs==1.0).all():
			#	sys.stderr.write("Non-polymorphic site!\n")

			refs = numpy.round(ref_freqs*depths)
			alts = depths-refs

			passed_sites = (depths>=lower_depth_threshold_vector)*1.0
			passed_sites *= (depths<=upper_depth_threshold_vector)
```

```
                    #print passed_sites.sum(), total_passed_samples,
passed_sites.sum()/total_passed_samples

                    # make sure the site is prevalent in enough samples to count as
"core"
                    if (passed_sites).sum()*1.0/total_passed_samples <
prevalence_threshold:
                                continue
                                #passed_sites *= 0

                    refs = refs*passed_sites
                    alts = alts*passed_sites
                    depths = depths*passed_sites

                    total_alts = alts.sum()
                    total_refs = depths.sum()
                    total_depths = total_alts+total_refs

                    # BG: 05/18: moving the polarization part to another part of
the pipeline
                    # so that we can use HMP polarization with other datasets.
                    # at the moment, still saving polarization state.

                    polarization = "R"
                    new_site_id_str = "|".join([contig, location, gene_name,
variant_type, polarization])


                    # print string
                    read_strs = ["%g,%g" % (A,A+R) for A,R in zip(alts, refs)]
                    print_str = "\t".join([new_site_id_str]+read_strs)

                    print print_str
                    #print total_alts

                    num_sites_processed+=1
                    if num_sites_processed%10000==0:
                                #sys.stderr.write("%dk sites processed...\n" %
(num_sites_processed/1000))
                                if debug:
                                            break

      ref_freq_file.close()
      depth_file.close()
      alt_allele_file.close()
      info_file.close()

      # returns nothing

# ============================================================================
# Loads list of SNPs and counts of target sites from annotated SNPs file
# for a particular species, from allowed samples at allowed sites
#
# returns:   desired_samples - samples considered (default all)
#                                    allele_counts_map - (gene -> vartype ->
locations, allele counts)
#                                    passed_sites_map - (gene -> vartype ->
location -> pass matrix)
#                                    final_line_number - last line in
annotated_snps.txt processed
# ============================================================================
```

```
def parse_snps(species_name, debug=False, allowed_samples=[], allowed_genes=[],
allowed_variant_types=['1D','2D','3D','4D'], initial_line_number=0,
chunk_size=1000000000, alt_passed_sites_map=False, prev_cohort='all'):

            from utils import snps_utils

            # Load population freqs (for polarization purposes)
            population_freqs = snps_utils.parse_population_freqs(prev_cohort,
species_name, polarize_by_consensus=False)

            # Open post-processed MIDAS output
            snps_dir = "%s/snps/%s/" % (data_dir, species_name)
            snp_file = bz2.BZ2File("%s/annotated_snps.txt.bz2" % snps_dir, 'r')

            # Get lists of desired samples, genes, and variant types
            items = snp_file.readline().strip().split()[1:]
            samples = su.parse_merged_sample_names(items)
            samples_list = list(samples)

            if len(allowed_samples) == 0:
                  allowed_sample_set = set(samples)
            else:
                  allowed_sample_set = (set(allowed_samples) & set(samples))

            desired_sample_idxs = numpy.array(sorted([samples_list.index(sample)
for sample in allowed_sample_set]))
            desired_samples =    samples[desired_sample_idxs]

            allowed_genes = set(allowed_genes)
            allowed_variant_types = set(allowed_variant_types)

            # Map: gene name -> variant type -> (list of locations, matrix of
allele counts)
            allele_counts_map = {}

            # Map: gene name -> variant type -> location -> sample-sample matrix
of whether both samples can be called at that site
            passed_sites_map = {}

            num_sites_processed = 0
            line_number = -1
            final_line_number = -1
            previous_gene_name = ""
            gene_name = ""

            for line in snp_file:

                        line_number += 1 # Start at line 0 (really, second line
of file)
                        previous_gene_name = gene_name

                        if line_number < initial_line_number:
                                    continue # Skip until desired initial line
number is reached

                        items = line.split()

                        # Load information about site
                        info_items = items[0].split("|")
                        chromosome = info_items[0]
```

```
                        location = long(info_items[1])
                        gene_name = info_items[2]
                        variant_type = info_items[3]

                        if len(info_items) > 5: # for backwards compatability
                                polarization = info_items[4]
                                pvalue = float(info_items[5])
                        else:
                                polarization="R" # not correct, but avoids a
crash (TODO: ??)
                                pvalue = float(info_items[4])

                        if num_sites_processed >= chunk_size and gene_name !=
previous_gene_name:
                                # Chunk processed, we are done for now!
                                final_line_number = line_number
                                break

                        # Ignore site if variant type or gene is not allowed

                        if not variant_type in allowed_variant_types:
                                continue

                        if len(allowed_genes)>0 and (not gene_name in
allowed_genes):
                                continue

                        # Load alt and depth counts for all desired samples
                        alts, depths = [], []

                        for idx in desired_sample_idxs:
                               item = items[1+idx]
                               subitems = item.split(",")
                               alts.append(float(subitems[0]))
                               depths.append(float(subitems[1]))

                        alts = numpy.array(alts)
                        depths = numpy.array(depths)

                        # Obtain population frequency of alt allele
                        if (chromosome, location) in population_freqs:
                               population_freq = population_freqs[(chromosome,
location)]
                        else: # alt population prevalence is (probably? TODO) 0
                               population_freq = 0

                        # Polarize SFS according to population freq
                        if population_freq > 0.5: # This means alt allele is the
major allele
                                alts = depths - alts
                                polarization = 'A'


                        # 0/1 array in which sample has 1 if site has nonzero
depth, else 0
                        passed_sites = (depths>0) * 1.0

                        # Set up passed_sites_map, allele_counts_map for new
genes
                        if gene_name not in passed_sites_map:
                               if alt_passed_sites_map == True:
```

```
                                        passed_sites_map[gene_name] = {v:
{'locations':[], 'sites':[]} for v in allowed_variant_types}
                                else:
                                        passed_sites_map[gene_name] = {v:
{'locations':[], 'sites':numpy.zeros((len(desired_samples), len(desired_samples)))}
for v in allowed_variant_types}
                                allele_counts_map[gene_name] = {v:
{'locations':[], 'alleles':[]} for v in allowed_variant_types}

                            # Store passed sites information
                            full_location = (chromosome,location)
                            passed_site_matrix =
passed_sites[:,None]*passed_sites[None,:]


      passed_sites_map[gene_name][variant_type]['locations'].append(full_location)
                            if alt_passed_sites_map == True:

      passed_sites_map[gene_name][variant_type]['sites'].append(passed_site_matrix)
                            else:
                                        passed_sites_map[gene_name][variant_type]['sites']
+= passed_site_matrix

                            # Zero out non-passed sites (TODO: pointless?)
                            alts = alts*passed_sites
                            depths = depths*passed_sites

                            # Calculate whether SNP has passed, i.e. number of
alternate alleles
                            # make up more than 5% of all alleles in at least one
sample
                            # and pvalue < 0.05 (where pvalue is from
annotated_snps.txt)

                            alt_threshold =
numpy.ceil(depths*config.parse_snps_min_freq) + 0.5
                            snp_passed = ((alts > alt_threshold).sum()>0) and
(pvalue<0.05)

                            # Criteria used in Schloissnig et al (Nature, 2013)
                            #total_alts = alts.sum()
                            #total_depths = depths.sum()
                            #pooled_freq =
total_alts/((total_depths+(total_depths==0))
                            #snp_passed = (freq>0.01) and (total_alts>=4) and
((total_depths-total_alts)>=4)

                            # Store allele counts only if the site is interesting
                            if snp_passed:
                                        allele_counts =
numpy.transpose(numpy.array([alts,depths-alts]))

      allele_counts_map[gene_name][variant_type]['locations'].append(full_location)

      allele_counts_map[gene_name][variant_type]['alleles'].append(allele_counts)

                                        num_sites_processed += 1

                                        if num_sites_processed>0 and
num_sites_processed%1000==0:
```

```
                                                sys.stderr.write("%dk sites
processed...\n" % (num_sites_processed/1000))
                                                if debug:
                                                            break

            snp_file.close()

            for gene_name in passed_sites_map.keys():
                  for variant_type in passed_sites_map[gene_name].keys():
                        allele_counts_map[gene_name][variant_type]['alleles'] =
numpy.array(allele_counts_map[gene_name][variant_type]['alleles'])

            return desired_samples, allele_counts_map, passed_sites_map,
final_line_number

# ====================================================================
# Calculates within-sample synonymous pi directly from annotated_snps.txt.bz2
# Ugly hack (since it does not encourage code re-use and puts pop-gen logic
# in the basic parsing scripts) but we need it so that we can call parse_snps
# on subsets of samples later on to improve performance
#
# Returns: samples, vector pi_s (raw counts), vector of opportunities
# ====================================================================

def parse_within_sample_pi_new(species_name, allowed_genes=set([]),
allowed_variant_types=set(['4D']), debug=False):

      samples, sfs_map = parse_within_sample_sfs(species_name,
allowed_variant_types)

      total_pi = []
      total_opportunities = []

      for sample in samples:
                  p,n =
diversity_utils.calculate_pi_from_sfs_map(sfs_map[sample])
                  total_pi.append(p)
                  total_opportunities.append(n)

      total_pi = numpy.array(total_pi)*1.0
      total_opportunities = numpy.array(total_opportunities)*1.0

      return samples, total_pi, total_opportunities

# ====================================================================
# Calculates within-sample sfs directly from annotated_snps.txt.bz2.
# Ugly hack (since it does not encourage code re-use and puts pop-gen logic
# in the basic parsing scripts) but we need it so that we can call parse_snps
# on subsets of haploid samples later on to improve performance
#
# prev cohorts: all, hmp, infant
# where core genes are still based on all but prev cohort is for snp prevalence
#
# returns vector of samples, vector of sfs maps
# ====================================================================

def parse_within_sample_sfs(species_name,
allowed_variant_types=set(['1D','2D','3D','4D']), prev_cohort='all'):

      # First read (filtered) genome-wide coverage distribution
```

```
	sfs_file = bz2.BZ2File("%s/snps/%s/within_sample_sfs_prev_%s.txt.bz2" %
(data_dir, species_name, prev_cohort), 'r')
	sfs_file.readline() # header

	sfs_map = defaultdict(dict)
	samples = []

	for line in sfs_file:
		items = line.strip().split("\t")
		sample, variant_type, sfs_items = items[0], items[1], items[2:]

		if variant_type not in allowed_variant_types:
			continue

		sample = sample[:-1] if sample[-1] == 'c' else sample
		if sample not in sfs_map:
			samples.append(sample)

		for sfs_item in sfs_items:
			subitems = sfs_item.split(",")
			D = long(subitems[0]) # Total depth
			A = long(subitems[1]) # Alt allele depth
			n = long(subitems[2]) # Number of sites with this D,A
			reverse_n = float(subitems[3]) # Reverse count (?)

			if D<0.5: # Ignore 0,0
				continue

			if (A,D) not in sfs_map[sample]: # why is this A,D instead of
D,A?
				sfs_map[sample][(D,A)] = [0,0.0]

			sfs_map[sample][(D,A)][0] += n
			sfs_map[sample][(D,A)][1] += reverse_n

	return numpy.array(samples), sfs_map

# ============================================================================
# Loads MIDAS's pangenome coverage data for a given species
# returns (lots of things, see below)
# ============================================================================

def parse_pangenome_data(species_name, allowed_samples = None, allowed_genes=[],
convert_centroid_names = True, disallowed_genes= []):

		gene_reads_path =   "%s/genes/%s/genes_reads.txt.bz2" % (data_dir,
species_name)

		if not os.path.isfile(gene_reads_path):
				return [], [], [], [], [], []

		# Open post-processed MIDAS output

		# genes_reads.txt: number of reads mapped to each gene per sample
		# TODO: is a single base overlap sufficient?
		gene_reads_file =    bz2.BZ2File("%s/genes/%s/genes_reads.txt.bz2" %
(data_dir, species_name),"r")

		# genes_depth.txt: average read depth of each gene per sample
		# TODO: averaged over all bases?
```

```
            gene_depth_file =    bz2.BZ2File("%s/genes/%s/genes_depth.txt.bz2" %
(data_dir, species_name),"r")

            # genes_presabs.txt: presence (1)/absence (0) of each gene per sample
            # where presabs is based on copy number threshold
            gene_presabs_file =  bz2.BZ2File("%s/genes/%s/genes_presabs.txt.bz2" %
(data_dir, species_name),"r")

            # Get marker gene coverages
            # marker_covs, mcov_samples =
parse_species_marker_gene_coverages(species_name)
            # # Convert to dictionary form
            # marker_coverage_map = {}
            # for mcov, sample in zip(marker_covs, mcov_samples):
            #             marker_coverage_map[sample] = mcov
            # Old way of getting marker gene coverages from genes_summary
            gene_summary_file = file("%s/genes/%s/genes_summary.txt" % (data_dir,
species_name),"r")
            marker_coverage_map = {}
            gene_summary_file.readline() # header
            marker_coverage_samples = []
            marker_coverages = []
            for summary_line in gene_summary_file:
                  items = summary_line.split()
                  sample = items[0].strip()
                  marker_coverage = float(items[5])
                  marker_coverage_samples.append(sample)
                  marker_coverages.append(marker_coverage)

            gene_summary_file.close()
            marker_coverage_samples =
su.parse_merged_sample_names(marker_coverage_samples)
            marker_coverage_map = {sample: marker_coverage for
sample,marker_coverage in zip(marker_coverage_samples, marker_coverages)}

            # Get rid of headers + get list of samples
            reads_line = gene_reads_file.readline() # header
            depth_line = gene_depth_file.readline() # header
            presabs_line = gene_presabs_file.readline() # header
            items = reads_line.split() # gene_id, samples...
            samples = su.parse_merged_sample_names(items[1:])

            # Restrict to intersection of allowed_samples with samples
            allowed_samples = set(samples) if (allowed_samples is None) else
(set(allowed_samples) & set(samples))

            # Boolean array: whether each of allowed_samples is in samples
            desired_sample_idxs = numpy.array([(s in allowed_samples) for s in
samples])

            # Marker coverages in same order as samples
            marker_coverages = numpy.array([marker_coverage_map[s] for s in
samples])

            # Final version of samples and marker coverages
            desired_samples = samples[desired_sample_idxs]
            marker_coverages = marker_coverages[desired_sample_idxs]

            gene_presence_matrix = []
            gene_reads_matrix = []
            gene_depth_matrix = []
```

```
		gene_names = []

		num_genes_processed = 0

		# Read first lines
		reads_line = gene_reads_file.readline()
		depth_line = gene_depth_file.readline()
		presabs_line = gene_presabs_file.readline()

		while reads_line != "":

				# First check for gene presence/absence
				items = presabs_line.split()
				gene_name = items[0]
				gene_presences = numpy.array([float(item) for item in
items[1:]])[desired_sample_idxs]

				if True: # gene_presences.sum() > 0.5:

					gene_reads = numpy.array([float(item) for
item in reads_line.split()[1:]])[desired_sample_idxs]
					gene_depths = numpy.array([float(item) for
item in depth_line.split()[1:]])[desired_sample_idxs]

					# Note to self: not uniform across samples!
					#gene_lengths =
gene_reads/(gene_depths+(gene_reads<0.5))
					#print gene_lengths

					# gene is present in at least one
individual!
					gene_presence_matrix.append(gene_presences)
					gene_depth_matrix.append(gene_depths)
					gene_reads_matrix.append(gene_reads)
					gene_names.append(gene_name)

				num_genes_processed+=1

				reads_line = gene_reads_file.readline() # header
				depth_line = gene_depth_file.readline() # header
				presabs_line = gene_presabs_file.readline() # header

		gene_reads_file.close()
		gene_depth_file.close()
		gene_presabs_file.close()
		gene_presence_matrix = numpy.array(gene_presence_matrix)
		gene_depth_matrix = numpy.array(gene_depth_matrix)
		gene_reads_matrix = numpy.array(gene_reads_matrix)

		if convert_centroid_names:
				new_gene_names = []
				centroid_gene_map =
midas_db_utils.load_centroid_gene_map(species_name)
				for gene_name in gene_names:

	new_gene_names.append(centroid_gene_map[gene_name])
		else:
				new_gene_names=gene_names

		new_gene_names = numpy.array(new_gene_names)
```

```
            # Now weed out disallowed genes if provided
            disallowed_genes=set(disallowed_genes)
            allowed_gene_idxs = []
            for gene_idx in xrange(0,len(new_gene_names)):

                        if new_gene_names[gene_idx] in disallowed_genes:
                                    # don't include
                                    pass
                        else:
                                    allowed_gene_idxs.append(gene_idx)
            allowed_gene_idxs = numpy.array(allowed_gene_idxs)

            new_gene_names = new_gene_names[allowed_gene_idxs]
            gene_presence_matrix = gene_presence_matrix[allowed_gene_idxs,:]
            gene_depth_matrix = gene_depth_matrix[allowed_gene_idxs,:]
            gene_reads_matrix = gene_reads_matrix[allowed_gene_idxs,:]

            return desired_samples, new_gene_names, gene_presence_matrix,
gene_depth_matrix, marker_coverages, gene_reads_matrix
```

parse_patric.py

```
import numpy
import sys
import bz2
import gzip
import os.path
import stats_utils
from math import floor, ceil
import gene_diversity_utils
import parse_midas_data
import config

################################################################################################
#
# Read in the Kegg info for a given speceis
# What is returned are all pathways for a given species.
#
################################################################################################

def load_kegg_annotations(genome_ids):

            # dictionary to store the kegg ids (gene_id -> [[kegg_id, description]])
            kegg_ids={}


            genomes_visited=[] #check if I have already loaded the genome for this gene
            for genome_id in genome_ids:
                        file = open("%skegg/%s.kegg.txt" % (config.patric_directory,
genome_id),"r")
      # file= bz2.BZ2File("%skegg/%s.kegg.txt.bz2" % (config.patric_directory, genome_id),"r")
                        file.readline() #header
                        for line in file:
                                    if line.split('\t')[0]!='':
                                                if line.strip() != "":
                                                            items = line.split("\t")

      gene_name=items[0].strip().split('|')[1]
                                                            kegg_ids[gene_name]=[]
```

```
	kegg_pathway_tmp=items[1].strip().split(';')
                                                                    if len(kegg_pathway_tmp)>0 and
kegg_pathway_tmp[0] !='':
                                                                        for i in range(0,
len(kegg_pathway_tmp)):

	kegg_ids[gene_name].append(kegg_pathway_tmp[i].split('|'))
                                                                    elif kegg_pathway_tmp[0] =='':

	kegg_ids[gene_name].append(['',''])
		return kegg_ids

########################################
def load_spgenes_annotations(gene_names):

		# dictionary to store the special gene ids (gene_id -> [property,product])
		spgenes_ids={}
		genomes_visited=[]
		for gene_name in gene_names:
				genome_id='.'.join(gene_name.split('.')[0:2])
				if genome_id not in genomes_visited:
						genomes_visited.append(genome_id)
						file=
gzip.open("%spatric_spgene/%s.PATRIC.spgene.tab.gz" % (config.patric_directory, genome_id),"r")
                                        file.readline() #header
                                        for line in file:
                                                if line.strip() != "":
                                                        items = line.split("\t")

	gene_name=items[2].strip().split('|')[1]
                                                        product=items[6]
                                                        property=items[7]

	spgenes_ids[gene_name]=[[property,product]]

		return spgenes_ids

def load_antibiotic_resistance_genes(species_name):

		# get pangenome genome for species_name

		pangenome_genes = parse_midas_data.load_pangenome_genes(species_name)
		spgenes_ids = load_spgenes_annotations(pangenome_genes)

		antibiotic_resistance_genes = set([])

		for gene_name in spgenes_ids.keys():

				if spgenes_ids[gene_name][0][0] == 'Antibiotic Resistance':
						antibiotic_resistance_genes.add(gene_name)

		return antibiotic_resistance_genes

def load_virulence_factors(species_name):

		# get pangenome genome for species_name

		pangenome_genes = parse_midas_data.load_pangenome_genes(species_name)
		spgenes_ids = load_spgenes_annotations(pangenome_genes)
```

```
	virulence_genes = set([])

	for gene_name in spgenes_ids.keys():

			if spgenes_ids[gene_name][0][0] == 'Virulence Factor':
				virulence_genes.add(gene_name)

	return virulence_genes



#################################################################
#
# Load individual gene names from patric
# This returns a dictionary with all gene names for the genomes included in the genome_ids
object.
# In the main code, I will pull out the actual gene names.
#
#################################################################

def load_patric_gene_descriptions(genome_ids,  allowed_genes=[]):

	allowed_genes = set(allowed_genes)
	# dictionary to store all gene names (gene_id -> )
	gene_descriptions={}

	for genome_id in genome_ids:
			# file = open('%s/genomes/%s/%s.PATRIC.features.tab' %
(config.patric_directory, genome_id, genome_id), 'r')
			try:
				file=gzip.open('%s/features/%s.PATRIC.features.tab.gz' %
(config.patric_directory, genome_id), 'r')
			except:
				print("PATRIC file for " + genome_id + " not found!")
				continue
			file.readline() #header
			for line in file:
				items = line.strip().split("\t")
				if items[0] !='':
					if items[5] !='' and len(items)>14: #
sometimes entries are blank
						gene_id =
items[5].split('|')[1] # id of gene
						if gene_id in allowed_genes:
							gene_description
= items[14] # what the gene does

		gene_descriptions[gene_id] = gene_description # load into the dictionary

	return gene_descriptions

#################################################################
#
# Load gene name from patric for a specific gene
#
#################################################################

def load_patric_gene_name(gene_id):

	# group genes by
	genome_id = gene_id.split('.')[0] + '.' + gene_id.split('.')[1]
```

```
            file=gzip.open('%s/features/%s.PATRIC.features.tab.gz' % (config.patric_directory,
genome_id), 'r')
            file.readline() #header
            for line in file:
                        items = line.strip().split("\t")
                        if items[0] !='':
                                    if items[5] !='' and len(items)>14: # sometimes entries
are blank
                                                if gene_id == items[5].split('|')[1]:
                                                            return items[14] # what the
gene does

############################################################
#
# Categorize PATRIC gene descriptions by      regular expression
#
############################################################
import operator
def cluster_patric_gene_descriptions(gene_descriptions):


            #iterate through and alphabetically categorize genes based on their string
identity. If there are at most 2 string mismatches with the previous string, then clump it with
that string.

            gene_categories={}  # key=gene_name, value=number of genes in this category
            gene_category_map={} # key=gene_id, value=category
            # I'm making this map so that we can later look up which category a patric id
belongs in.

            prev_gene='' # I will do a regexp with this as I iterate and keep updating
            gene_categories[prev_gene]=0 #initialize
            #iterate through alphabetically (faster)
            for item in sorted(gene_descriptions.items(), key=operator.itemgetter(1)):
                        gene_id=item[0]
                        gene=item[1]
                        hamming_distance= hamming(gene, prev_gene)
                        if hamming_distance<=2:
                                    gene_categories[prev_gene]+=1
                                    gene_category_map[gene_id]=prev_gene
                        else:
                                    # sometimes the alpha sort doesn't take care of corner
cases. Iterate through the whole list again to find an existing match if possible.
                                    found_category=False
                                    for existing_gene in gene_categories.keys():
                                                hamming_distance= hamming(gene,
existing_gene)
                                                if hamming_distance <=2 and
found_category==False:
                                                            gene_categories[existing_gene]
+=1

      gene_category_map[gene_id]=existing_gene
                                                            found_category=True
                                    # if no match is found, create a new category
                                    if found_category==False:
                                                gene_categories[gene]=1
                                                gene_category_map[gene_id]=gene
                                                prev_gene=gene
```

```
		return gene_categories, gene_category_map

#########
# hamming distance between two strings:

import itertools

def hamming(str1, str2):
	diff=sum(itertools.imap(str.__ne__, str1, str2))
	diff += abs(len(str1) - len(str2)) # above doesn't take into account difference in string
length
	return diff

##############################################################################################
#
# Create a new genome.features.gz file for running MIDAS on a different representative genome
for SNP calling.
#
##############################################################################################
def new_genome_features_file(genome_id, outFN):

		pollard_patric_dir='/pollard/shattuck0/snayfach/databases/PATRIC/genomes'


	#outFile=gzip.open('/pollard/home/ngarud/BenNanditaProject/MIDAS_ref_genome_test/genome_f
eatures_files/%s_features.gz' % genome_id,"w")

		outFile=gzip.open(outFN, "w")
		outFile.write("gene_id\tscaffold_id\tstart\tend\tstrand\tgene_type\tfunctions\n")

		for genome_part in ['cds','rna']:
			file= gzip.open("%s/%s/%s.PATRIC.%s.tab.gz" %
(pollard_patric_dir,genome_id, genome_id, genome_part),"r")
			file.readline() #header

			for line in file:
				items = line.split("\t")
				gene_id=items[5].strip().split('|')[1]
				scaffold_id=items[2].strip()
				start=items[9].strip()
				end=items[10].strip()
				strand=items[11].strip()
				gene_type=items[4].strip()
				if len(items)>15:
					functions=items[15].strip()
				else:
					functions=''
				# NRG: added 'accn|' to match the headers in the fasta
file (09/06/17)
				outFile.write(gene_id +'\t' +'accn|'+scaffold_id
+'\t'+start +'\t' + end +'\t' +strand +'\t' +gene_type +'\t' +functions +'\n')

##############################################################################################
#
# Read in the genome_metadata file. This tells you where genomes in PATRIC came from
# for example, if we want genomes that are part of the HMP reference panel, this is the file to
look into
#
##############################################################################################
def get_HMP_reference_genomes():
```

```
        genome_metadata =
open("/pollard/shattuck0/snayfach/databases/PATRIC/metadata/genome_metadata")
        HMP_genomes={}
        for line in genome_metadata:
                items=line.strip().split('\t')
                if len(items) == 65: # sometimes a line is shorter, this creates
problems
                        genome_id=items[0]
                        species=items[1]
                        annotation=items[64]
                        contigs=items[27]
                        genome_length=items[29]
                        body_part = items[35]
                        host = items[45]
                        # This annotation is one of a few! Missed a few genomes
(NRG, 09/06/17)
                        if 'Reference genome for the Human Microbiome Project'
in annotation:
                                HMP_genomes[genome_id] = [int(contigs),
int(genome_length), body_part, host]

        return HMP_genomes
```

plot_utils.py

```
import matplotlib
# matplotlib.use('Agg')
import matplotlib.pyplot as plt
import parse_midas_data as pmd
import midas_db_utils
from collections import defaultdict

def get_species_color_map(all_species = pmd.parse_species_list()):

    gfo_phylum_map = midas_db_utils.load_gfo_phylum_map()

    all_phyla = set()
    phylum_species_map = defaultdict(list)

    for species in all_species:
        gfo = species.split('_')[0] # Either genus, family or order
        try:
            phylum = gfo_phylum_map[gfo]
            all_phyla.add(phylum)
            phylum_species_map[phylum].append(species)
        except:
            print(species) # Guyana massiliensis is unclassified, skip for
now

    # There are the following 7 phyla in these gut microbiota
    species_color_map = {}
    ordered_species_list = []

    # Firmicutes (81)
    species_list = sorted(phylum_species_map['Firmicutes'], key=lambda x: x[1])
    ordered_species_list += species_list
    m = get_cm_ScalerMappable(matplotlib.cm.PuRd, len(species_list), 7, 3)
    for i in range(len(species_list)):
        species_color_map[species_list[i]] = m.to_rgba(i)
```

```
	# Bacteroidetes (54)
	species_list = sorted(phylum_species_map['Bacteroidetes'], key=lambda x:
x[1])
	ordered_species_list += species_list
	m = get_cm_ScalerMappable(matplotlib.cm.YlGnBu, len(species_list), 5, 2)
	for i in range(len(species_list)):
		species_color_map[species_list[i]] = m.to_rgba(i)

	# Proteobacteria (10)
	species_list = sorted(phylum_species_map['Proteobacteria'], key=lambda x:
x[1])
	ordered_species_list += species_list
	m = get_cm_ScalerMappable(matplotlib.cm.Oranges, len(species_list), 2, 3)
	for i in range(len(species_list)):
		species_color_map[species_list[i]] = m.to_rgba(i)

	# Actinobacteria (9)
	species_list = sorted(phylum_species_map['Actinobacteria'], key=lambda x:
x[1])
	ordered_species_list += species_list
	m = get_cm_ScalerMappable(matplotlib.cm.Wistia, len(species_list), 2, 4)
	for i in range(len(species_list)):
		species_color_map[species_list[i]] = m.to_rgba(i)

	# Fusobacteria (1), Spirochaetes (1), Verrucomicrobia (1)
	species_list = phylum_species_map['Fusobacteria'] +
phylum_species_map['Spirochaetes'] + phylum_species_map['Verrucomicrobia']
	ordered_species_list += species_list
	m = get_cm_ScalerMappable(matplotlib.cm.Greens, len(species_list), 2, 1)
	for i in range(len(species_list)):
		species_color_map[species_list[i]] = m.to_rgba(i)

	# Special: set '-' to gray
	species_color_map['-'] = (0.6, 0.6, 0.6, 1.0)

	return species_color_map, ordered_species_list

def get_cm_ScalerMappable(cmap, num_colors, offset1 = 0, offset2 = 0):
	norm = matplotlib.colors.Normalize(vmin = 0 - offset1, vmax = num_colors- 1
+ offset2)
	return matplotlib.cm.ScalarMappable(norm=norm, cmap=cmap)

def list_to_colors(input_list):
	cmap = matplotlib.cm.hsv

	input_list_dict = {}
	i = 0
	for elem in set(input_list):
		input_list_dict[elem] = i
		i += 1

	norm = matplotlib.colors.Normalize(vmin=0, vmax=i-1)
	m = matplotlib.cm.ScalarMappable(norm=norm, cmap=cmap)

	color_list = []
	for elem in input_list:
		color_i = input_list_dict[elem]
		color_list.append(m.to_rgba(color_i))

	return color_list
```

```
def colors_to_legend_elements(colors, labels):
	legend_elements = {}
	for color, label in zip(colors, labels):
		legend_elements[color] = matplotlib.patches.Patch(facecolor=color,
label=label)
	return [legend_elements[c] for c in remove_list_duplicates(colors)]

def remove_list_duplicates(mylist):
	# Returns list with only unique elements (like set)
	# but ordered according to first appearnce in original list
	result = []
	for elem in mylist:
		if elem not in result:
			result.append(elem)
	return result


def get_pretty_species_name(species_name, include_number=False):
	items = species_name.split("_")
	pretty_name = "%s %s" % (items[0], items[1])
	if include_number:
			pretty_name += (" (%s)" % (items[2]))
	return pretty_name

def get_abbreviated_species_name(species_name):
	items = species_name.split("_")
	pretty_name = "%s. %s" % (items[0][0], items[1])
	return pretty_name
```

sample_utils.py

```
import numpy as np
import config
import parse_midas_data
from collections import defaultdict

# ============================================================================
# This module (parse_metadata) contains the following utilities:
#
#	sample_to_tp, sample_pair_to_tp_pair
#
#	is_same_mi_subject, is_mi_pair
#
#	get_sample_names, get_ferretti_sample_names_by_site, calculate_qp_samples
#	calculate_ordered_same_subject_pairs, calculate_ordered_diff_subject_pairs
#	calculate_ordered_same_sample_pairs, calculate_ordered_subject_pairs
#	calculate_mi_ordered_same_subject_pairs, calculate_sample_idx_map
#	apply_sample_index_map_to_indices, calculate_samples_in_different_subjects
#	calculate_subject_pairs, calculate_sample_subject_matrix
#
#	parse_sample_metadata_map, filter_sample_metadata_map,
extract_sample_metadata_map
#	parse_sample_subject_map, parse_sample_country_map,
parse_sample_continent_map
#	parse_subject_sample_map, parse_sample_order_map
#
#	flatten_samples, flatten_subjects, parse_isolate_metadata_map
#	list_of_isolates_and_mixtures, parse_merged_sample_names
#	calculate_unique_samples
# ============================================================================
```

```
# Generic helper functions

flatten = lambda l: [item for sublist in l for item in sublist]

def subject_and_tp_to_sample(subject, tp, subject_sample_map, sample_order_map):
	for sample in subject_sample_map[subject].keys():
		subject, cur_order = sample_order_map[sample]
		if float(tp[1:]) == cur_order:
			return sample
	print("Sample not found")
	return None

# ========================================================================
# FUNCTIONS FOR REFORMATTING METADATA
# ========================================================================

# ========================================================================
# sample_pair_to_tp: converts sample to generic timepoint descriptor
# A for adult, M for mother, I for infant
# ========================================================================

def sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples):
	order_str = str(sample_order_map[sample][1])
	if sample in hmp_samples:
		return 'A' + order_str
	else: # If not HMP, assume mother or infant
		return ('M' if sample in mother_samples else 'I') + order_str

# ========================================================================
# sample_pair_to_tp_pair: converts sample pair to general timepoint pair form
# example: frozenset('I1', 'I3')
# A for adult, M for mother, I for infant
# ========================================================================

def sample_pair_to_tp_pair(sample_i, sample_j, sample_order_map, hmp_samples,
mother_samples):
	if sample_i in hmp_samples:
		tp_i = 'A' + str(sample_order_map[sample_i][1])
	else: # If not HMP, assume mother or infant
		tp_i = ('M' if sample_i in mother_samples else 'I') +
str(sample_order_map[sample_i][1])
	if sample_j in hmp_samples:
		tp_j = 'A' + str(sample_order_map[sample_j][1])
	else: # If not HMP, assume mother or infant
		tp_j = ('M' if sample_j in mother_samples else 'I') +
str(sample_order_map[sample_j][1])
	tp_pair = frozenset((tp_i, tp_j))
	return tp_pair

# ========================================================================
# FUNCTIONS FOR DEALING WITH SAME SUBJECT MOTHER-INFANT
# ========================================================================

# ========================================================================
# get_same_mi_pair_dict: returns mapping of mother/child subject to
# child/mother subject (only valid for mi cohorts excluding olm)
# ========================================================================

def get_same_mi_pair_dict(subject_sample_map):
	same_mi_pair_dict = {}
	for subject1 in subject_sample_map:
```

```
            for subject2 in subject_sample_map:
                  if subject2 != subject1 and subject2[:-2] == subject1[:-2]:
                        same_mi_pair_dict[subject1] = subject2
      return same_mi_pair_dict

# ============================================================================
# is_same_mi_subject: returns true if two samples are from same mother-infant pair
# (Includes two infant or two mother samples at different timepoints)
# ============================================================================

def is_same_mi_subject(sample1, sample2, sample_subject_map):

      # Note: this only works on mother/infant datasets
      # Observe the following subject name format conventions
      # Backhed: 64-I/-M
      # Ferretti: CA_C10006IS
      # Yassour: M0333-M
      # Shao: B02752-I
      # Olm: S2_017

      subject1 = sample_subject_map[sample1]
      subject2 = sample_subject_map[sample2]

      return (subject1[:-2] == subject2[:-2])

# ============================================================================
# is_mi_pair: returns true if one sample is mother and other is infant
# ============================================================================

def is_mi_pair(sample1, sample2, mother_samples, infant_samples):

      return ((sample1 in mother_samples and sample2 in infant_samples) or
      (sample1 in infant_samples and sample2 in mother_samples))

# ============================================================================
# FUNCTIONS FOR DEALING WITH SAMPLE LISTS
# ============================================================================

# ============================================================================
# get_sample_names: returns list of samples in specified cohort/timepoint
#
# cohort: one of 'HMP', 'Backhed', 'Ferretti', 'Yassour', Shao', 'Olm'
# (case insensitive),     'mother', 'infant', 'adult', 'all', or 'all-dict'
# note that 'dict' is (cohort->tp->samples)
#
# timepoint:
#     HMP - 1, 2, 3
#           Backhed - B, 4M, 12M, M
#           Ferretti - M0, I1, I2, I3, I4, I5
#           Yassour - MGest, MBirth, M3, CBirth, C14, C1, C2, C3
#     Shao - Mother, Neonatal (4-21 days), Infancy
#     Olm - NIH1, NIH2, NIH3, NIH4, Sloan2
#           or 'all' for every sample in a cohort
# ============================================================================

def get_sample_names(cohort, timepoint = 'all', remove_c=True):

      sample_dict = {'hmp': defaultdict(set), 'backhed': defaultdict(set), \
                                                      'ferretti': defaultdict(set),
'yassour': defaultdict(set), \
```

```
                                                    'shao': defaultdict(set),
'olm': defaultdict(set)}

      # Note: we need the sample lists because not all samples in the metadata
      # files should be returned. The final ID is based on metadata file, however
      # (so -c suffixes for combined samples will be omitted.)

      metadata_dir = config.metadata_directory
      samples_dir = "%s/final_sample_lists" % metadata_dir

      # HMP - expect 469 samples
      samples_fpath = "%s/HMP1-2_samples.txt" % samples_dir
      hmp_samples = [line.strip() for line in open(samples_fpath, 'r')]
      hmp_samples = parse_merged_sample_names(hmp_samples) if remove_c else
hmp_samples # Remove c's

      metadata_fpath = "%s/HMP1-2_metadata.txt" % metadata_dir
      with open(metadata_fpath, 'r') as metadata_file:
            metadata = [row.strip().split('\t') for row in metadata_file]

      for sample in metadata[1:]:
            _, sample_id, _, _, _, tp = sample
            if sample_id in hmp_samples:
                  sample_dict['hmp'][tp].add(sample_id)
            if (sample_id+'c') in hmp_samples:
                  sample_dict['hmp'][tp].add((sample_id+'c'))

      # Backhed - expect 391 samples
      samples_fpath = "%s/Backhed_samples.txt" % samples_dir
      backhed_samples = [line.strip() for line in open(samples_fpath, 'r')]

      metadata_fpath = "%s/Backhed_metadata.txt" % metadata_dir
      with open(metadata_fpath, 'r') as metadata_file:
            metadata = [row.strip().split('\t') for row in metadata_file]

      for sample in metadata[1:]:
            sample_id, _, tp = sample
            if sample_id in backhed_samples:
                  sample_dict['backhed'][tp].add(sample_id)

      # Ferretti - expect 119 samples
      samples_fpath = "%s/Ferretti_samples.txt" % samples_dir
      ferretti_samples = [line.strip() for line in open(samples_fpath, 'r')]

      metadata_fpath = "%s/Ferretti_metadata.txt" % metadata_dir
      with open(metadata_fpath, 'r') as metadata_file:
            metadata = [row.strip().split('\t') for row in metadata_file]

      for sample in metadata[1:]:
            sample_id = sample[4]
            tp = sample[6][9] + sample[6][19]
            # Restrict to fecal samples
            body_site = sample[6][15:17]
            if body_site == 'FE' and sample_id in ferretti_samples:
                  sample_dict['ferretti'][tp].add(sample_id)

      # Yassour - expect 286 samples
      samples_fpath = "%s/Yassour_samples.txt" % samples_dir
      yassour_samples = [line.strip() for line in open(samples_fpath, 'r')]

      metadata_fpath = "%s/Yassour_metadata.txt" % metadata_dir
```

```
	with open(metadata_fpath, 'r') as metadata_file:
		metadata = [row.strip().split('\t') for row in metadata_file]

	for sample in metadata[1:]:
		sample_id = sample[4]
		tp_raw = sample[7][6:].split(':')
		tp = tp_raw[0][0] + tp_raw[1].split()[0]
		if sample_id in yassour_samples:
			sample_dict['yassour'][tp].add(sample_id)

	# Shao - expect 1679 samples
	samples_fpath = "%s/Shao_samples.txt" % samples_dir
	shao_samples = [line.strip() for line in open(samples_fpath, 'r')]

	metadata_fpath = "%s/Shao_metadata.txt" % metadata_dir
	with open(metadata_fpath, 'r') as metadata_file:
		metadata = [row.strip().split('\t') for row in metadata_file]

	for sample in metadata[1:]:
		sample_id, _, _, neonatal_tp, tp_cat = sample
		if tp_cat == 'NA':
			continue # Ignore if order info not known
		tp_cat = 'Infancy' if neonatal_tp == 'Infancy' else tp_cat
		if sample_id in shao_samples:
			sample_dict['shao'][tp_cat].add(sample_id)

	# Olm - expect 898 samples
	olm_samples = []
	for campaign in ['NIH1', 'NIH2', 'NIH3', 'NIH4', 'Sloan2']:
		samples_fpath = "%s/Olm_%s_samples.txt" % (samples_dir, campaign)
		olm_sub_samples = [line.strip() for line in open(samples_fpath, 'r')]
		if remove_c:
			olm_samples += list(parse_merged_sample_names(olm_sub_samples))
# Remove c's
		else:
			olm_samples += list(olm_sub_samples)

	metadata_fpath = "%s/Olm_metadata.txt" % metadata_dir
	with open(metadata_fpath, 'r') as metadata_file:
		metadata = [row.strip().split('\t') for row in metadata_file]

	for sample in metadata[1:]:
		try:
			sample_id = sample[9]
			campaign = sample[3]
			if sample_id in olm_samples:
				sample_dict['olm'][campaign].add(sample_id)
			if (sample_id+'c') in olm_samples:
				sample_dict['olm'][campaign].add((sample_id+'c'))
		except:
			continue

	# Aggregate + convert sets to lists
	all_samples = []
	for lcohort in sample_dict:
		for ltp in sample_dict[lcohort]:
			sample_dict[lcohort][ltp] = list(sample_dict[lcohort][ltp])
			all_samples += sample_dict[lcohort][ltp]

	# All HMP samples
```

```
	hmp_samples = flatten([sample_dict['hmp'][htp] for htp in
sample_dict['hmp']])

	# All mother and infant samples
	mother_samples = []
	mother_samples += sample_dict['backhed']['M']
	mother_samples += sample_dict['ferretti']['M0']
	mother_samples += sample_dict['shao']['Mother']
	mother_samples += flatten([sample_dict['yassour'][ytp] for ytp in ['MGest',
'MBirth', 'M3']])
	infant_samples = [s for s in all_samples if (s not in mother_samples and s
not in hmp_samples)]

	general_cohort_dict = {'all': all_samples, 'mother': mother_samples,
'infant': infant_samples, 'adult': mother_samples + hmp_samples}

	cohort = cohort.lower() # for case insensitivity

	if cohort in general_cohort_dict: # all, mother, infant, adult
		return general_cohort_dict[cohort]
	elif cohort == 'all-dict': # all-dict
		return sample_dict
	elif timepoint == 'all': # specific cohort, all
		all_cohort_samples = []
		for ltp in sample_dict[cohort]:
			all_cohort_samples += sample_dict[cohort][ltp]
		return all_cohort_samples
	else: # specific cohort, specific timepoint
		return sample_dict[cohort][timepoint]

# ============================================================================
# get_mi_tp_sample_dict: returns dictionary that maps each each timepoint to
# list of samples across ALL mother-infant cohorts
#
# The following conventions are used to unify timepoints across cohorts:
#	- Assume each "month" is 30.5 days long, so month 1 = day 30.5
#		- Assume each "week" is 7 days long, so week 2 = day 14
#	- Time values are expressed in days relative to birth/delivery
#	- Time values are rounded to the nearest day
#
# If binned = True, use each day as timepoint. Otherwise, use custom bins
# of timepoints (days 0-6, weeks 1-3, months 1-11, year 1 for infants)
#
# Also, uses finer resolution timepoints for Backhed (not just order)
#
# ============================================================================

def get_mi_tp_sample_dict(exclude_cohorts = [], binned = False):

	sample_cohort_dict = get_sample_names('all-dict')
	sample_order_map = parse_sample_order_map()
	subject_sample_map = parse_subject_sample_map()
	tp_sample_dict = {'mother': defaultdict(list), 'infant': defaultdict(list)}

	# Backhed timepoints:
	# M: Mother, delivery / 0-5 days after birth (median 2)
	# B: Infant, birth / 2-5 days after birth (median 3)
	# 12M: Infant, 4 months / 119-125 days after birth (median 122)
	# 4M: Infant, 12 months / 363-372 days after birth (median 366)

	# Load info from metadata file
```

```
	backhed_default_tps = [3, 2, 122, 366]
	with open('%s/phenotypes/Backhed_phenotype_metadata.csv' %
config.metadata_directory, 'r') as file:
		header = file.readline()
		for line in file:
			items = line.strip().split(',')
			study_id = items[0]

			B_M_4M_12M_days = []
			for i in range(4):
				try:
					B_M_4M_12M_days.append(int(items[3:7][i]))
				except:
					B_M_4M_12M_days.append(backhed_default_tps[i])
			# sample_days_B, sample_days_M, sample_days_4M, sample_days_12M
= [int(val) for val in ]
			for sample in subject_sample_map[study_id+'-I']:
				_, order = sample_order_map[sample]
				if order == 1: # Birth

	tp_sample_dict['infant'][B_M_4M_12M_days[0]].append(sample)
				elif order == 2: # 4M

	tp_sample_dict['infant'][B_M_4M_12M_days[2]].append(sample)
				elif order == 3: # 12M

	tp_sample_dict['infant'][B_M_4M_12M_days[3]].append(sample)
			for sample in subject_sample_map[study_id+'-M']:

	tp_sample_dict['mother'][B_M_4M_12M_days[1]].append(sample)

	'''
	if 'backhed' not in exclude_cohorts:
		backhed_tp_map = {'M': 2, 'B': 3, '4M': 122, '12M': 366}
		for btp in backhed_tp_map:
			stp = backhed_tp_map[btp]
			mother_or_infant = 'mother' if (btp == 'M') else 'infant'
			tp_sample_dict[mother_or_infant][stp] +=
sample_cohort_dict['backhed'][btp]
	'''

	# Ferretti timepoints:
	# M0: Mother, delivery
	# I1: Infant, 1 day
	# I2: Infant, 3 days
	# I3: Infant, 1 week
	# I4: Infant, 1 month
	# I5: Infant, 4 months

	if 'ferretti' not in exclude_cohorts:
		ferretti_tp_map = {'M0': 0, 'I1': 1, 'I2': 3, 'I3': 7, 'I4': 30, 'I5':
122}
		for ftp in ferretti_tp_map:
			stp = ferretti_tp_map[ftp]
			mother_or_infant = 'mother' if (ftp == 'M0') else 'infant'
			tp_sample_dict[mother_or_infant][stp] +=
sample_cohort_dict['ferretti'][ftp]

	# Yassour timepoints:
	# MGest: Mother, gestational week 27
	# MBirth: Mother, delivery
```

```
	# M3: Mother, 3 months post-delivery
	# CBirth: Infant, birth / meconium
	# C14: Infant, 2 weeks
	# C1: Infant, 1 week
	# C2: Infant, 2 months
	# C3: Infant, 3 months

	if 'yassour' not in exclude_cohorts:
		yassour_tp_map = {'MGest': -92, 'MBirth': 0, 'M3': 92, 'CBirth': 0,
'C14': 14, 'C1': 30, 'C2': 61, 'C3': 92}
		for ytp in yassour_tp_map:
			stp = yassour_tp_map[ytp]
			mother_or_infant = 'mother' if (ytp[0] == 'M') else 'infant'
			tp_sample_dict[mother_or_infant][stp] +=
sample_cohort_dict['yassour'][ytp]

	# Shao timepoints:
	# Mother: one timepoint at delivery
	# Neonatal: 4-21 days (various)
	# Infancy: 4-15 months (various)

	if 'shao' not in exclude_cohorts:
		shao_samples = get_sample_names('Shao')
		for sample in shao_samples:
			subject, order = sample_order_map[sample]
			order = int(round(order))
			mother_or_infant = 'mother' if (subject[-1] == 'M') else
'infant'
			tp_sample_dict[mother_or_infant][order].append(sample)

	'''
	# Olm timepoints:
	# 5-86 days

	if 'olm' not in exclude_cohorts:
		olm_samples = get_sample_names('Olm')
		for sample in olm_samples:
			subject, order = sample_order_map[sample]
			order = int(round(order))
			tp_sample_dict['infant'][order].append(sample)
	'''

	if binned == False:
		return tp_sample_dict

	# Custom timepoint binning (label -> lower bound of bin, inclusive)
	custom_infant_bins = {'birth': 0, '1 day': 1, '2 day': 2, '3 day': 3, '4
day': 4, '5 day': 5, '6 day': 6, \
	'1 wk': 7, '2 wk': 14, '3 wk': 21, '1 mon': 30, '2 mon': 61, '3 mon': 91, '4
mon': 122, '5 mon': 152, '6 mon': 183, \
	'7 mon': 213, '8 mon': 244, '9 mon': 274, '10 mon': 305, '11 mon': 335, '1
yr': 365}
	custom_infant_bins_inverted = {v: k for k, v in custom_infant_bins.items()}
	custom_infant_bin_bounds = sorted(custom_infant_bins.values(), reverse=True)


	# Like tp_sample_dict but uses new bins instead of days
	binned_infant_tp_sample_dict = defaultdict(list)

	for day in tp_sample_dict['infant']:
		# Find the right bin for this day
```

```
            for bound in custom_infant_bin_bounds:
                  if day >= bound:
                        bin_label = custom_infant_bins_inverted[bound]
                        binned_infant_tp_sample_dict[bin_label] +=
tp_sample_dict['infant'][day]
                        break

      custom_infant_bin_labels = []
      for bound in sorted(custom_infant_bins.values()):
            bin_label = custom_infant_bins_inverted[bound]
            if bin_label in binned_infant_tp_sample_dict:
                  custom_infant_bin_labels.append(bin_label)

      return {'mother': tp_sample_dict['mother'], 'infant':
binned_infant_tp_sample_dict}, custom_infant_bin_labels

def get_mi_sample_day_dict(exclude_cohorts = [], binned = False):

      mi_tp_sample_dict = get_mi_tp_sample_dict(exclude_cohorts, binned)
      mi_sample_day_dict = {}

      for cat in ['infant', 'mother']:
            for day in mi_tp_sample_dict[cat]:
                  for sample in mi_tp_sample_dict[cat][day]:
                        mi_sample_day_dict[sample] = day

      return mi_sample_day_dict

# ================================================================================
# get_ferretti_sample_names_by_site: get dictionary of samples by body site
# Site names: FE (stool), SA (oral cavity), SK (skin), VA (vagina)
# ================================================================================

def get_ferretti_sample_names_by_site(body_site_code):

      metadata_fpath = config.metadata_directory + "PRJNA352475.txt" # Ferretti
      with open(metadata_fpath, 'r') as metadata_file:
            metadata = [row.strip().split('\t') for row in
metadata_file.readlines()]

      site_dict = defaultdict(list)

      for sample in metadata[1:]:
            sample_name = sample[4]
            body_site = sample[6][-8:-6]
            site_dict[body_site].append(sample_name)

      return site_dict[body_site_code]

# ================================================================================
# Output information on the timepoint pairs present in a sample set
# ================================================================================

# TODO

# ================================================================================
# calculate_qp_samples: returns QP status dictionary given samples, species
# 'qp', 'non-qp' [high coverage], 'low-coverage'
# ================================================================================

def calculate_qp_samples(all_samples, species_name, prev_cohort='all'):
```

```
	import diversity_utils

	# list of samples that meet coverage criteria for this species
	highcoverage_samples =
set(diversity_utils.calculate_highcoverage_samples(species_name))

	# list of samples that meet QP criteria for this species
	haploid_samples = set(diversity_utils.calculate_haploid_samples(species_name,
prev_cohort=prev_cohort))

	qp_statuses = ['qp','non-qp','low-coverage']
	qp_sample_sets = {status: set() for status in qp_statuses}

	for sample in all_samples:
		if sample in haploid_samples:    # QP (must be high coverage)
			qp_sample_sets['qp'].add(sample)
		elif sample in highcoverage_samples:   # Non-QP (high coverage)
			qp_sample_sets['non-qp'].add(sample)
		else:  # Low coverage
			qp_sample_sets['low-coverage'].add(sample)

	return qp_sample_sets

# ========================================================================
# load_qp_samples: returns QP status dictionary given samples, species
# 'qp', 'non-qp' [high coverage], 'low-coverage' (uses pickle)
# ========================================================================

def load_qp_samples(desired_samples, species_name, prev_cohort='all', force_repickle
= False):

	import pickle, os.path
	pickle_fn = "%s/pickles/qp_samples/%s_qp_sample_dict.pkl" %
(config.data_directory, species_name)

	if force_repickle or not os.path.isfile(pickle_fn):
		all_samples = get_sample_names('all')
		qp_sample_sets = calculate_qp_samples(all_samples, species_name,
prev_cohort=prev_cohort)
		pickle.dump(qp_sample_sets, open(pickle_fn, 'wb'))
		return qp_sample_sets
	else:
		qp_sample_sets = pickle.load(open(pickle_fn, 'rb'))
		for cat in qp_sample_sets: # qp, non-qp, low-coverage
			old_sample_list = list(qp_sample_sets[cat])
			for sample in old_sample_list:
				if sample not in desired_samples:
					qp_sample_sets[cat].remove(sample)
		return qp_sample_sets

# ========================================================================
# FUNCTIONS FOR DEALING WITH SAMPLE PAIRS
# ========================================================================

# ========================================================================
# calculate_ordered_same_subject_pairs: computes same-subject sample pairs;
# specify if timepoints should be nonconsecutive, consecutive, or longest
#
# Considers mothers and infants different subjects
#
```

```
# Returns same_subject_idxs, tuple with idx1 (lower order) and idx2 (higher)
# ============================================================================

def calculate_ordered_same_subject_pairs(sample_order_map, sample_list=[],
within_host_type='consecutive'):
	idx_lower, idx_upper = [], []

	# reconstruct "timeseries" for each subject
	subject_order_idx_map = defaultdict(dict)
	for i in xrange(0,len(sample_list)):
		subject, order = sample_order_map[sample_list[i]]
		subject_order_idx_map[subject][order] = i

	# create index pairs within subjects
	for subject in subject_order_idx_map:
		sorted_orders = list(sorted(subject_order_idx_map[subject].keys()))

		if len(sorted_orders) < 2:# no pairs can be formed
			continue

		if within_host_type=='longest':

	idx_lower.append(subject_order_idx_map[subject][sorted_orders[0]])
			idx_upper.append(subject_order_idx_map[subject][sorted_orders[-
1]])
		elif within_host_type=='consecutive':
			for order_idx in xrange(1,len(sorted_orders)):

	idx_lower.append(subject_order_idx_map[subject][sorted_orders[order_idx-1]])

	idx_upper.append(subject_order_idx_map[subject][sorted_orders[order_idx]])
		elif within_host_type=='nonconsecutive':
			for order_idx_i in xrange(0,len(sorted_orders)):
				for order_idx_j in
xrange(order_idx_i+1,len(sorted_orders)):

	idx_lower.append(subject_order_idx_map[subject][sorted_orders[order_idx_i]])

	idx_upper.append(subject_order_idx_map[subject][sorted_orders[order_idx_j]])

	same_subject_idxs = (np.array(idx_lower,dtype=np.int32),
np.array(idx_upper,dtype=np.int32))
	return same_subject_idxs

# ============================================================================
# calculate_ordered_diff_subject_pairs: computes diff-subject sample pairs;
# only one sample considered for each subject; specify whether timepoint
# should be first or last # TODO?
#
# Returns diff_subject_idxs, tuple with idx1 and idx2
# ============================================================================

def calculate_ordered_diff_subject_pairs(sample_order_map, sample_list=[],
diff_host_type='first'):

	# reconstruct "timeseries" for each subject
	subject_order_idx_map = defaultdict(dict)
	for i in xrange(0,len(sample_list)):
		subject, order = sample_order_map[sample_list[i]]
		subject_order_idx_map[subject][order] = i
```

```
	sorted_subjects = sorted(subject_order_idx_map.keys()) # all subjects
	op = max if diff_host_type == 'last' else min
	idx_lower, idx_upper = [], []

	for subject_i_idx in xrange(0,len(sorted_subjects)):
		subject_i = sorted_subjects[subject_i_idx]
		earliest_order_i = op(subject_order_idx_map[subject_i].keys())
		i = subject_order_idx_map[subject_i][earliest_order_i]

		for subject_j_idx in xrange(subject_i_idx+1,len(sorted_subjects)):
			subject_j = sorted_subjects[subject_j_idx]
			earliest_order_j = op(subject_order_idx_map[subject_j].keys())
			j = subject_order_idx_map[subject_j][earliest_order_j]

			idx_lower.append(i)
			idx_upper.append(j)

	diff_subject_idxs = (np.array(idx_lower,dtype=np.int32),
np.array(idx_upper,dtype=np.int32))
	return diff_subject_idxs

# ============================================================================
# calculate_ordered_same_sample_pairs: computes same sample "pair" indices
# Assumes no duplicate samples in sample_list
# ============================================================================

def calculate_ordered_same_sample_pairs(sample_order_map, sample_list=[]):
	idxs = np.arange(0,len(sample_list))
	return (np.array(idxs,dtype=np.int32), np.array(idxs,dtype=np.int32))

# ============================================================================
# calculate_ordered_subject_pairs: wrapper function for combining
# calculate_ordered_same_sample_pairs, calculate_ordered_same_subject_pairs
# and calculate_ordered_diff_subject_pairs
# ============================================================================

def calculate_ordered_subject_pairs(sample_order_map, sample_list=[],
within_host_type='consecutive', diff_host_type='first'):

	same_sample_idxs = calculate_ordered_same_sample_pairs(sample_order_map,
sample_list)
	same_subject_idxs = calculate_ordered_same_subject_pairs(sample_order_map,
sample_list, within_host_type)
	diff_subject_idxs = calculate_ordered_diff_subject_pairs(sample_order_map,
sample_list, diff_host_type)
	return same_sample_idxs, same_subject_idxs, diff_subject_idxs

# ============================================================================
# calculate_mi_ordered_same_subject_pairs: computes same-subject sample pairs
# considering each mother-infant pair as the SAME subject. Parameters:
#	within_host_type: nonconsecutive, consecutive, longest
#	one_per_mi_pair: True, False
#	infant_timepoint_pref: first, random, last
#
# Returns same_subject_idxs, tuple with idx1 (lower order / mother) and
# idx2 (higher order / infant)
# ============================================================================

def calculate_mi_ordered_same_subject_pairs(sample_order_map, sample_list=[],
within_host_type='consecutive', one_per_mi_pair=False,
infant_timepoint_pref='first'):
```

```
            same_subject_idxs =
calculate_ordered_same_subject_pairs(sample_order_map, sample_list,
within_host_type)

            mother_samples = get_sample_names('mother','all')
            infant_samples = get_sample_names('infant','all')
            sample_subject_map = parse_sample_subject_map()

            # Include same mother and infant comparisons (one per subject)
            more_same_subject_idxs_i = []
            more_same_subject_idxs_j = []


            if one_per_mi_pair == True: # Choose one timepoint pair for each
mother-infant pair

                  existing_subjects = {}

                  for i in range(len(sample_list)):
                        for j in range(i+1, len(sample_list)):
                              sample_i, sample_j = sample_list[i], sample_list[j]
                              # Check if one is mother and other is her infant
                              if is_mi_pair(sample_i, sample_j, mother_samples,
infant_samples) and is_same_mi_subject(sample_i, sample_j, sample_subject_map):

                                    m_index, i_index = (i, j) if
sample_order_map[sample_i][0][-1] == 'M' else (j, i)
                                    subject = sample_order_map[sample_i][0][:-2]
# Excludes -I/-M
                                    # If random, any mother-infant timepoint
pair works
                                    if subject not in existing_subjects:
                                          existing_subjects[subject] = (m_index,
i_index)
                                    elif infant_timepoint_pref != 'random':
                                          prev_i_index =
existing_subjects[subject][1]
                                          prev_infant_tp =
sample_order_map[sample_list[prev_i_index]][1]
                                          cur_infant_tp =
sample_order_map[sample_list[i_index]][1]
                                          if infant_timepoint_pref == 'first':
                                                if cur_infant_tp <
prev_infant_tp:

      existing_subjects[subject] = (m_index, i_index)
                                          elif infant_timepoint_pref == 'last':
                                                if cur_infant_tp >
prev_infant_tp:

      existing_subjects[subject] = (m_index, i_index)

                  for index_pair in existing_subjects.values():
                        more_same_subject_idxs_i.append(index_pair[0])
                        more_same_subject_idxs_j.append(index_pair[1])

            else: # Choose all timepoint pairs for each mother-infant pair

                  for i in range(len(sample_list)):
                        for j in range(i+1, len(sample_list)):
```

```
                                    sample_i, sample_j = sample_list[i], sample_list[j]
                                    if is_mi_pair(sample_i, sample_j, mother_samples,
infant_samples) and is_same_mi_subject(sample_i, sample_j, sample_subject_map):
                                          m_index, i_index = (i, j) if
sample_order_map[sample_i][0][-1] == 'M' else (j, i)
                                          more_same_subject_idxs_i.append(m_index)
                                          more_same_subject_idxs_j.append(i_index)

            idxs1 = np.array(np.append(same_subject_idxs[0],
more_same_subject_idxs_i), dtype=np.int32)
            idxs2 = np.array(np.append(same_subject_idxs[1],
more_same_subject_idxs_j), dtype=np.int32)
            return (idxs1, idxs2)

# ================================================================================
# calculate_sample_idx_map: creates a map of indexes from one list of samples
# (sample_list_from) to another list of samples (sample_list_to).
# The from list must be a strict subset of the to list.
# ================================================================================

def calculate_sample_idx_map(sample_list_from, sample_list_to):

      sample_list_to = list(sample_list_to)
      sample_map = {}
      for i in xrange(0,len(sample_list_from)):
            sample_map[i] = sample_list_to.index(sample_list_from[i])

      return sample_map

def apply_sample_index_map_to_indices(sample_idx_map, idxs):
      new_idxs = (np.array([sample_idx_map[i] for i in idxs[0]]),
np.array([sample_idx_map[i] for i in idxs[1]]))
      return new_idxs

# ================================================================================
# calculate_samples_in_different_subjects: returns boolean array indicating
# whether each sample in sample_list has same subject as focal_sample
# ================================================================================

def calculate_samples_in_different_subjects(sample_subject_map, sample_list,
focal_sample):
      focal_subject = sample_subject_map[focal_sample]
      subjects = np.array([sample_subject_map[s] for s in sample_list])
      return (subjects != focal_subject)

# ================================================================================
# calculate_subject_pairs: calculates which samples belong to different
# subjects, which belong to different timepoints in same subject, and
# which are the same timepoint.
#
# Considers mothers and infants different subjects
# This is a naive way to generate sample pairs
#
# Returns same_sample_idxs, same_subject_idxs, diff_subject_idxs,
# each of which is a tuple with lists idx1 and idx2.
# All pairs are included only once (order doesn't matter).
# ================================================================================

def calculate_subject_pairs(sample_subject_map, sample_list = None):

      if sample_list is None:
```

```
              sample_list = sample_subject_map.keys()

       same_sample_idx_lower, same_sample_idx_upper = [], []
       same_subject_idx_lower, same_subject_idx_upper = [], []
       diff_subject_idx_lower, diff_subject_idx_upper = [], []

       for i in xrange(0,len(sample_list)):
              sample = sample_list[i]
              same_sample_idx_lower.append(i)
              same_sample_idx_upper.append(i)
              for j in xrange(0,i):
                     if sample_subject_map[sample] == sample_subject_map[sample]:
                            same_subject_idx_lower.append(i)
                            same_subject_idx_upper.append(j)
                     else:
                            diff_subject_idx_lower.append(i)
                            diff_subject_idx_upper.append(j)

       same_sample_idxs = (np.array(same_sample_idx_lower,dtype=np.int32),
np.array(same_sample_idx_upper,dtype=np.int32))
       same_subject_idxs = (np.array(same_subject_idx_lower,dtype=np.int32),
np.array(same_subject_idx_upper,dtype=np.int32))
       diff_subject_idxs = (np.array(diff_subject_idx_lower,dtype=np.int32),
np.array(diff_subject_idx_upper,dtype=np.int32))

       return same_sample_idxs, same_subject_idxs, diff_subject_idxs

# ============================================================================
# calculate_sample_subject_matrix: matrix, rows are subjects, columns are hosts
# A_ih = 1 if sample i is in host h
# ============================================================================

def calculate_sample_subject_matrix(samples):

       sample_idx_map = {samples[i]:i for i in xrange(0,len(samples))}

       subject_sample_map = parse_subject_sample_map()
       subjects = subject_sample_map.keys()

       sample_subject_matrix = np.zeros((len(samples),len(subjects)),dtype=np.bool)

       for subject_idx in xrange(0,len(subjects)):
              for sample in subject_sample_map[subjects[subject_idx]]:
                     if sample in sample_idx_map:
                            sample_subject_matrix[sample_idx_map[sample],
subject_idx] = True

       return sample_subject_matrix, subjects

# ============================================================================
# FUNCTIONS FOR SAMPLE-METADATA MAPS
# ============================================================================

#
# ============================================================================
# parse_subject_delivery_mode_map
#
# Returns map: subject (infants only) -> delivery mode
#
# ============================================================================
```

```
def parse_subject_delivery_mode_map():

	from config import metadata_directory
	subject_devmode_map = {} # 'Vaginal' or 'C-section'

	# Backhed
	# Confirmed that all 98 infant subjects are included
	with open('%s/phenotypes/Backhed_phenotype_metadata.csv' %
metadata_directory, 'r') as file:
		header = file.readline()
		for line in file:
			items = line.strip().split(',')
			subject = items[0] + '-I'
			delivery_mode = 'Vaginal' if items[2] == 'vaginal' else 'C-
section'
			subject_devmode_map[subject] = delivery_mode

	# Ferretti
	# Confirmed that all 25 infant subjects are included
	with open('%s/phenotypes/Ferretti_phenotype_metadata.csv' %
metadata_directory, 'r') as file:
		header = file.readline()
		for line in file:
			items = line.strip().split(',')
			subject = "CA_C" + items[0] + "IS"
			delivery_mode = "Vaginal" # All were vaginal
			subject_devmode_map[subject] = delivery_mode

	# Yassour
	# Confirmed that all 43 infant subjects are included
	with open('%s/phenotypes/Yassour_phenotype_metadata.csv' %
metadata_directory, 'r') as file:
		header = file.readline()
		for line in file:
			items = line.strip().split(',')
			subject = items[0] + '-C'
			delivery_mode = items[1]
			subject_devmode_map[subject] = delivery_mode

	# Shao
	# Confirmed that all 596 infant subjects are included
	with open('%s/phenotypes/Shao_phenotype_metadata.csv' % metadata_directory,
'r') as file:
		header = file.readline()
		for line in file:
			items = line.strip().split(',')
			subject = items[1] + '-I'
			delivery_mode = 'C-section' if items[3] == 'Caesarean' else
items[3]
			subject_devmode_map[subject] = delivery_mode

	return subject_devmode_map

#
================================================================================
# parse_feeding_mode_map
#
# Returns map: subject (infants only) -> feeding mode (in first 4 months)
#
================================================================================
```

```
def parse_subject_feeding_mode_map():

	from config import metadata_directory
	subject_feeding_map = {} # 'breast', 'mixed' or 'formula'

	# Backhed
	with open('%s/phenotypes/Backhed_phenotype_metadata.csv' %
metadata_directory, 'r') as file:
		header = file.readline()
		for line in file:
			items = line.strip().split(',')
			subject = items[0] + '-I'
			feeding_orig = items[7] # Week 1
			if feeding_orig == 'mixed feeding':
				feeding_mode = 'mixed'
			elif feeding_orig == 'exclusively breastfeeding':
				feeding_mode = 'breast'
			elif 'formula' in feeding_orig.lower():
				feeding_mode = 'formula'
			subject_feeding_map[subject] = feeding_mode

	# Ferretti
	with open('%s/phenotypes/Ferretti_phenotype_metadata.csv' %
metadata_directory, 'r') as file:
		header = file.readline()
		for line in file:
			items = line.strip().split(',')
			subject = "CA_C" + items[0] + "IS"
			breastfed_1day = items[8]
			if breastfed_1day == 'YES':
				feeding_mode = 'breast'
			elif breastfed_1day == 'NO':
				print(subject) # Doesn't exist in this case
			subject_feeding_map[subject] = feeding_mode

	# Yassour
	with open('%s/phenotypes/Yassour_phenotype_metadata.csv' %
metadata_directory, 'r') as file:
		header = file.readline()
		for line in file:
			items = line.strip().split(',')
			subject = items[0] + '-C'
			feeding_orig = items[11] # Week 2
			if feeding_orig == 'Exclusive breastmilk':
				feeding_mode = 'breast'
			elif feeding_orig == 'Breastmilk & formula':
				feeding_mode = 'mixed'
			elif feeding_orig == 'NA':
				continue
			subject_feeding_map[subject] = feeding_mode

	# Shao
	# Confirmed that all 596 infant subjects are included
	with open('%s/phenotypes/Shao_phenotype_metadata.csv' % metadata_directory,
'r') as file:
		header = file.readline()
		for line in file:
			items = line.strip().split(',')
			subject = items[1] + '-I'
			feeding_orig = items[12]
			if feeding_orig == 'BF':
```

```
                        feeding_mode = 'breast'
                elif feeding_orig == 'Mixed':
                        feeding_mode = 'mixed'
                elif feeding_orig == 'NoBF':
                        feeding_mode = 'formula'
                elif feeding_orig == 'NA':
                        continue
                subject_feeding_map[subject] = feeding_mode

        return subject_feeding_map

#
================================================================================
# parse_sample_read_count_map
#
# Loads HMP read counts (# lines in fastq files / 4)
# Returns map: sample -> read_count
#
================================================================================

def parse_sample_read_count_map():

        from config import metadata_directory
        sample_rc_map = {}

        # First load HMP metadata
        with open("%s/read_counts/HMP1-2_read_counts.txt" % (metadata_directory),
'r') as file:
                for line in file:
                        sample, read_count = line.strip().split('\t')
                        if sample.endswith('c'):
                                sample = sample[:-1]
                        sample_rc_map[sample] = int(read_count)

        # Next load Backhed, Ferretti, Yassour, Qin (same file format)
        for cohort in ['Backhed', 'Ferretti', 'Yassour', 'Shao', 'Qin']:
                with open("%s/read_counts/%s_read_counts.txt" % (metadata_directory,
cohort), 'r') as file:
                        header = file.readline()
                        for line in file:
                                _, _, _, run_accession, _, read_count, base_count =
line.strip().split('\t')
                                sample_rc_map[run_accession] = int(read_count)

        return sample_rc_map

#
================================================================================
# parse_sample_metadata_map
#
# Loads metadata for HMP, Yassour, Backhed, Ferretti, Shao, Olm samples
# Returns map:
# sample -> (subject_id, sample_id, accession_id, country, continent,
temporal_order)
#
# By default (at least for postprocessing steps), include all samples. But can
# restrict to fecal samples only, and good timepoints only (some Shao have 'NA' tp)
#
================================================================================

def parse_sample_metadata_map(fecal_only = False, good_tp_only = False):
```

```
	metadata_dir = config.metadata_directory
	samples_dir = "%s/final_sample_lists" % metadata_dir

	sample_metadata_map = {} # What is returned!

	# First load HMP metadata (469 samples)

	samples_fpath = "%s/HMP1-2_samples.txt" % samples_dir
	hmp_samples = [line.strip() for line in open(samples_fpath, 'r')]
	hmp_samples = parse_merged_sample_names(hmp_samples) # Remove c's

	with open("%s/HMP1-2_metadata.txt" % metadata_dir, 'r') as metadata_file:
		metadata_file.readline() # header
		for line in metadata_file:
			subject_id, sample_id, accession_id, country, continent, order =
line.strip().split('\t')
			order = int(order)
			if sample_id in hmp_samples:
				sample_metadata_map[sample_id] = (subject_id, sample_id,
accession_id, country, continent, order)

	# Then load Backhed data (391 samples)

	timept_order_map_mother = {'M':1}
	timept_order_map_infant = {'B':1,'4M':2,'12M':3}

	with open("%s/Backhed_metadata.txt" % metadata_dir, 'r') as metadata_file:
		metadata_file.readline() # header
		for line in metadata_file:
			accession_id, subject_id, timept = line.strip().split('\t')
			# Using the family/study_id as subject id, and specify mother (-
M) vs infant (-I)
			subject_id = subject_id + ('-M' if timept == 'M' else '-I')
			sample_id = accession_id # Sample ID same as run accession
			order = timept_order_map_mother[timept] if timept == 'M' else
timept_order_map_infant[timept]
			sample_metadata_map[sample_id] = (subject_id, sample_id,
accession_id, 'Sweden', 'Europe', order)

	# Then load Ferretti data (119 fecal samples)

	timept_order_map_mother = {'t0':1}
	timept_order_map_infant = {'t1':1,'t2':2,'t3':3,'t4':4,'t5':5}

	with open("%s/Ferretti_metadata.txt" % metadata_dir, 'r') as metadata_file:
		metadata_file.readline() # header
		for line in metadata_file:
			items = line.strip().split("\t")
			accession_id = items[4]
			subject_id = items[6][:11] # Using first 11 characters of
experiment_alias as subject id (e.g. CA_C10055IS)
			sample_id = accession_id # Sample ID same as run accession
			timept = items[5][-2:] # Using last two characters of
experiment_alias to identify timepoint
			order = timept_order_map_mother[timept] if subject_id[-2:] ==
'MS' else timept_order_map_infant[timept]
			if fecal_only:
				if items[6][15:17] == 'FE': # Restrict to fecal samples
					sample_metadata_map[sample_id] = (subject_id,
sample_id, accession_id, 'Italy', 'Europe', order)
```

```
                        else:
                                sample_metadata_map[sample_id] = (subject_id, sample_id,
accession_id, 'Italy', 'Europe', order)

        # Then load Yassour data (286 samples)

        timept_order_map_mother = {'Mother:Gest':1, 'Mother:Birth':2,'Mother:3
months':3}
        timept_order_map_infant = {'Child:Birth':1,'Child:14 days':2,'Child:1
month':3,'Child:2 months':4,'Child:3 months':5}

        with open("%s/Yassour_metadata.txt" % metadata_dir, 'r') as metadata_file:
                metadata_file.readline() # header
                for line in metadata_file:
                        items = line.strip().split("\t")
                        accession_id = items[4]
                        subject_id = items[7][:7] # Using first 7 characters of
sample_title as subject id (e.g. M0059-M)
                        sample_id = accession_id # Sample ID same as run accession
                        timept = items[7][6:] # Using characters after 6th of
sample_title to identify timepoint
                        order = timept_order_map_mother[timept] if subject_id[-1] == 'M'
else timept_order_map_infant[timept]
                        sample_metadata_map[sample_id] = (subject_id, sample_id,
accession_id, 'Finland', 'Europe', order)

        # Then load Shao data (1679 samples - 1676 excluding NA timepoints)

        with open("%s/Shao_metadata.txt" % metadata_dir, 'r') as metadata_file:
                metadata_file.readline() # header
                for line in metadata_file:
                        accession_id, _, subject_id, timept, infancy_months =
line.strip().split("\t")
                        sample_id = accession_id # Sample ID same as run accession

                        # Adjust subject ID by appending -M or -I
                        if timept == 'Mother' and infancy_months == 'Mother':
                                subject_id += '-M'
                                order = 1 # Only one mother timepoint
                        elif timept == 'Infancy':
                                subject_id += '-I'

                                if infancy_months == 'NA':
                                        if good_tp_only == True:
                                                continue # Skip if infancy months field is
NA
                                        else:
                                                infancy_months = -999 # Bogus negative
number

                                order = float(infancy_months) * 30.5 # Convert months to
approx. days
                        elif infancy_months == 'Neonatal':
                                subject_id += '-I'
                                order = int(timept) # In days since birth

                        sample_metadata_map[sample_id] = (subject_id, sample_id,
accession_id, 'United Kingdom', 'Europe', order)

        # Then load Olm data (898 samples)
```

```
	olm_samples = []
	for campaign in ['NIH1', 'NIH2', 'NIH3', 'NIH4', 'Sloan2']:
		samples_fpath = "%s/Olm_%s_samples.txt" % (samples_dir, campaign)
		olm_sub_samples = [line.strip() for line in open(samples_fpath, 'r')]
		olm_samples += list(parse_merged_sample_names(olm_sub_samples)) #
Remove c's

	with open("%s/Olm_metadata.txt" % metadata_dir, 'r') as metadata_file:
		metadata_file.readline() # header
		for line in metadata_file:
			items = line.strip().split("\t")
			if len(items) == 10: # Must have available accession
				subject_id = items[1]
				timept = items[2]
				accession_id = items[9]
				sample_id = accession_id # Sample ID same as run
accession
				order = int(timept) # In days since birth
				if accession_id in olm_samples: # Restrict to considered
samples
					sample_metadata_map[sample_id] = (subject_id,
sample_id, accession_id, 'United States', 'North America', order)

	return sample_metadata_map

#
================================================================================
# filter_sample_metadata_map
#
# Using passed in sample-metadata map, filters only sample entries corresponding to
a
# certain subject_id, country, continent or order
#
================================================================================

def filter_sample_metadata_map(sample_metadata_map, field, field_value):

	field_dict = {"subject_id": 0, "country": 3, "continent": 4, "order": 5}
	if field in field_dict:
		field_idx = field_dict[field]
	else:
		return sample_metadata_map

	filtered_sample_metadata_map = {}
	for sample in sample_metadata_map:
		if sample_metadata_map[sample][field_idx] == field_value:
			filtered_sample_metadata_map[sample] =
sample_metadata_map[sample]

	return filtered_sample_metadata_map

#
================================================================================
# extract_sample_metadata_map
#
# Using passed in sample-metadata map, extracts information for only one column
# (options: subject_id, country, continent or order) and returns new map
# Loads the default full sample-metadata map if nothing passed in
#
================================================================================
```

```
def extract_sample_metadata_map(field, sample_metadata_map = None):

	field_dict = {"subject_id": 0, "country": 3, "continent": 4, "order": 5}
	field_idx = field_dict[field] if (field in field_dict) else 0 # Defaults to
subject_id

	if sample_metadata_map is None:
		sample_metadata_map = parse_sample_metadata_map() # Load it

	extracted_sample_metadata_map = {}
	for sample in sample_metadata_map:
		extracted_sample_metadata_map[sample] =
sample_metadata_map[sample][field_idx]

	return extracted_sample_metadata_map

#
================================================================================
# parse_sample_subject_map, parse_sample_country_map, parse_sample_continent_map
#
# Convenience functions for extract_sample_metadata_map
#
================================================================================

def parse_sample_subject_map(sample_metadata_map = None):
	return extract_sample_metadata_map("subject_id", sample_metadata_map)

def parse_sample_country_map(sample_metadata_map = None):
	return extract_sample_metadata_map("country", sample_metadata_map)

def parse_sample_continent_map(sample_metadata_map = None):
	return extract_sample_metadata_map("continent", sample_metadata_map)

#
================================================================================
# parse_subject_sample_map
#
# Returns map: subject -> map: samples -> set of accession IDs
#
================================================================================

def parse_subject_sample_map(sample_metadata_map = None):

	if sample_metadata_map is None:
		sample_metadata_map = parse_sample_metadata_map() # Load it

	subject_sample_map = {}
	for sample in sample_metadata_map:
		subject_id, _, accession_id, country, continent, order =
sample_metadata_map[sample]
		if subject_id not in subject_sample_map:
			subject_sample_map[subject_id] = {}
		if sample not in subject_sample_map[subject_id]:
			subject_sample_map[subject_id][sample] = set()
		subject_sample_map[subject_id][sample].add(accession_id)

	return subject_sample_map

#
================================================================================
# parse_sample_order_map
```

```
#
# Returns map from sample -> (subject_id, temporal_order)
#
================================================================================

def parse_sample_order_map(sample_metadata_map = None):

	if sample_metadata_map is None:
		sample_metadata_map = parse_sample_metadata_map() # Load it

	sample_order_map = {}
	for sample in sample_metadata_map:
			subject_id, _, _, _, _, order = sample_metadata_map[sample]
			sample_order_map[sample] = (subject_id, order)

	return sample_order_map

#
================================================================================
# parse_sample_cohort_map
#
================================================================================

def parse_sample_cohort_map():

	sample_cohort_map = {}
	sample_dict = get_sample_names('all-dict')

	for cohort in sample_dict:
		for sample_list in sample_dict[cohort].values():
			for sample in sample_list:
				sample_cohort_map[sample] = cohort

	return sample_cohort_map

#
================================================================================
# FUNCTIONS FOR DEALING WITH REPLICATES
#
================================================================================

#
================================================================================
# Returns a flat map of all the replicate sets for
# the samples in subject_sample_map, indexed by sample key
#
================================================================================

def flatten_samples(subject_sample_map):

	grouping_replicate_map = {}
	for subject in sorted(subject_sample_map.keys()):
		for sample in sorted(subject_sample_map[subject].keys()):
			grouping_replicate_map[sample] =
subject_sample_map[subject][sample]

	return grouping_replicate_map

#
================================================================================
# Returns a flat map of the merged replicate sets for each subject,
```

```
# indexed by subject key
#
================================================================================

def flatten_subjects(subject_sample_map):

	grouping_replicate_map = {}
	for subject in sorted(subject_sample_map.keys()):
		merged_replicates = set()
		for sample in subject_sample_map[subject].keys():
			merged_replicates.update(subject_sample_map[subject][sample])
		grouping_replicate_map[subject] = merged_replicates

	return grouping_replicate_map

#
================================================================================
# groupings = ordered list of nonoverlapping sets of sample names
# samples = ordered list of samples
#
# returns: list whose i-th element contains a np array of idxs
#                                of the items in samples that are present in the
ith grouping
#
================================================================================

def calculate_grouping_idxs(groupings, samples):

		grouping_idxs = []
		for i in xrange(0,len(groupings)):

				idxs = []
				for j in xrange(0,len(samples)):
						if samples[j] in groupings[i]:
								idxs.append(j)
				idxs = np.array(idxs,dtype=np.int32)
				#print idxs
				grouping_idxs.append(idxs)

		return grouping_idxs

# ============================================================================
# Isolate and mixture metadata parsing
# ============================================================================

def parse_isolate_metadata_map():

		isolate_metadata_map = {}

		# load simulations
		file =
open(parse_midas_data.scripts_directory+"isolates_genome_list.txt","r")
		file.readline() #
		for line in file:
				items = line.strip().split("\t")
				subject_id = items[0]
				sample_id = subject_id
				accession_id=subject_id
				country = "isolate"
				continent = "isolate"
				order = 1
```

```
                        isolate_metadata_map[sample_id] = (subject_id, sample_id,
accession_id, country, continent, order)

            file =
open(parse_midas_data.scripts_directory+"mixture_labels.txt","r")
            file.readline() # header
            for line in file:
                        items = line.strip().split("\t")
                        subject_id = items[0] # this is one of two of the 90/10
mixtures
                        sample_id = items[1] # This is the exact simulation
                        accession_id=sample_id # same as sample
                        country = "mixture"
                        continent = "mixture"
                        order = 1
                        isolate_metadata_map[sample_id] = (subject_id, sample_id,
accession_id, country, continent, order)

            return isolate_metadata_map

def list_of_isolates_and_mixtures():

            isolate_metadata_map = parse_isolate_metadata_map()

            isolates=[]
            mixtures=[]

            for sample_id in isolate_metadata_map:
                        subject_id, dummy, accession_id, country, continent,
order = sample_metadata_map[sample_id]
                        if country=='isolate':
                                    isolates.append(sample_id)
                        elif country=='mixture':
                                    mixtures.append(sample_id)

            return isolates, mixtures

# ================================================================================
# Simply removes 'c' suffix for any merged samples
# ================================================================================

def parse_merged_sample_names(items):
      samples = []
      for item in items:
            sample = item.strip()
            if sample.endswith('c'):
                  sample = sample[:-1]
            samples.append(sample)

      samples = np.array(samples)
      return samples

# ================================================================================
# Prunes sample list to remove multiple timepoints from same subject
# Considers mothers and infants as different subjects
# Returns len(sample_list) boolean array with element=False if sample was pruned

# ================================================================================

def calculate_unique_samples(subject_sample_map, sample_list=None):
```

```
            if sample_list is None:
                        sample_list =
list(sorted(flatten_samples(subject_sample_map).keys()))

            # invert subject sample map
            sample_subject_map = parse_sample_subject_map()

            subject_idx_map = {}

            for i in xrange(0,len(sample_list)):
                        sample = sample_list[i]
                        if sample.endswith('c'):
                                    sample = sample[:-1]
                        subject = sample_subject_map[sample]
                        if not subject in subject_idx_map:
                                    subject_idx_map[subject] = i

            unique_idxs = np.zeros(len(sample_list),dtype=np.bool_)
            for i in subject_idx_map.values():
                        unique_idxs[i]=True

            return unique_idxs
```

sfs_utils.py

```
################################################################################
#
# Utility methods for working with (primarily within-sample) SFSs
#
################################################################################
import numpy
import config
from math import log10
from utils.classes import Interval

def calculate_binned_sfs_from_sfs_map(sfs_map, bins=[], folding='minor'):

    alts = []
    depths = []
    counts = []
    for key in sfs_map.keys():
        D,A = key
        n = sfs_map[key][0]

        alts.append(A)
        depths.append(D)
        counts.append(n)

    alts = numpy.array(alts)
    depths = numpy.array(depths)
    counts = numpy.array(counts)
    weights = counts*1.0/counts.sum()

    freqs = alts*1.0/depths
    minor_freqs = numpy.fmin(freqs,1-freqs)

    # calculate median depth (or rough approximation)
    sorted_depths, sorted_weights = (numpy.array(x) for x in zip(*sorted(zip(depths,
weights))))
    CDF = numpy.cumsum(sorted_weights)
```

```
    Dbar = sorted_depths[CDF>0.5][0]

    if len(bins)==0:
        # use this to set up bins
        bins = (numpy.arange(0,Dbar+2)-0.5)/Dbar
        fs = numpy.arange(0,Dbar+1)*1.0/Dbar

    else:
        bins = numpy.array(bins)
        fs = bins[1:]

    pfs = numpy.zeros_like(fs)
    bin_idxs = numpy.digitize(minor_freqs, bins=bins)

    for bin_idx, weight in zip(bin_idxs, weights):
        pfs[bin_idx-1] += weight

    # should already be normalized, but just to make sure...
    pfs /= pfs.sum()

    if folding=='major':
        pfs = pfs[::-1]
        fs = (1.0-fs)[::-1]

    return fs,pfs

def calculate_binned_depth_distribution_from_sfs_map(sfs_map,bins=[],num_bins=30):

    alts = []
    depths = []
    counts = []
    for key in sfs_map.keys():
        D,A = key
        n = sfs_map[key][0]

        alts.append(A)
        depths.append(D)
        counts.append(n)

    alts = numpy.array(alts)
    depths = numpy.array(depths)
    counts = numpy.array(counts) # read counts for all sites with (alt, depth)
    weights = counts*1.0/counts.sum() # read counts for each frequency bin normalized by
total read counts

    # calculate median depth (or rough approximation)

              # sort weights by depth
    sorted_depths, sorted_weights = (numpy.array(x) for x in zip(*sorted(zip(depths,
weights))))

              # get median depth
    CDF = numpy.cumsum(sorted_weights)
    Dbar = sorted_depths[CDF>0.5][0]

    if len(bins)==0:
        # use this to set up bins
                          # range from median depth/8 to depth*8, log space
        bins = numpy.logspace(log10(Dbar/8),log10(Dbar*8),num_bins)
                          # Ds are depth bins
        Ds = bins[0:-1]
```

```
    else:
        bins = numpy.array(bins, copy=True)
        Ds = bins[0:-1]

    bins[0] = 0
    bins[-1] = 1e09

    pDs = numpy.zeros_like(Ds)
             # For each depth, assign which bin it falls in (via bins index)
    bin_idxs = numpy.digitize(depths, bins=bins)

    for bin_idx, weight in zip(bin_idxs, weights):
                           # pDs are weights for depth bins
        pDs[bin_idx-1] += weight

    # should already be normalized, but just to make sure...
    pDs /= pDs.sum()

    return bins, Ds, pDs

def calculate_depth_distribution_from_sfs_map(sfs_map):

    depth_map = {}
    for key in sfs_map.keys():
        D,A = key
        count = sfs_map[key][0]

        if D not in depth_map:
            depth_map[D] = 0

        depth_map[D] += count

    depths = numpy.array(sorted(depth_map.keys()))
    counts = numpy.array([depth_map[d] for d in depths])

    return depths, counts

def
calculate_polymorphism_rates_from_sfs_map(sfs_map,lower_threshold=0.2,upper_threshold=0.8):

    total_sites = 0 # Total number of sites
    within_sites = 0 # Number of sites with frequency 0.2-0.8 (not inclusive)
    between_sites = 0 # Number of sites with frequency <= 0.2, >= 0.8
    for key in sfs_map.keys():
        D,A = key
        n = sfs_map[key][0]
        reverse_n = sfs_map[key][1]

        f = A*1.0/D

        total_sites += n

        if ((f>lower_threshold) and (f<upper_threshold)):
            # an intermediate frequency site
            within_sites += n
        else:
            if f>0.5:
                between_sites += (n-reverse_n) #NRG what does this do?
            else:
                between_sites += reverse_n
```

```
    return within_sites, between_sites, total_sites

# ============================================================
# number of polymorphisms among synonymous sites
# (number sites with minor allele frequency >0.05) / (total number synonymous sites)
# Rare SFS: (0, 0.05], [0.95, 1)
# Common SFS: [0.2, 0.8]
# ============================================================

# ============================================================
# Computes number of sites, in given SFS, that have frequency
# within given list of frequency intervals/ranges
#
# Note that Interval class is used to express ranges
# ============================================================

def calculate_sites_within_freq_range_from_sfs_map(sfs_map, freq_ranges = [Interval('[0.2,
0.8]')], min_alt = 2):

	total_sites = 0
	within_sites = 0

	for D, A in sfs_map:
		n, reverse_n = sfs_map[(D, A)]
		if A < min_alt:
			f = 0 # If not enough alt reads, assume frequency 0
		else:
			f = A*1.0/D # Frequency of alternate allele

		total_sites += n

		for range in freq_ranges:
			# As long as one of the ranges contains the frequency
			if range.contains(f):
				within_sites += n
				continue

	return total_sites, within_sites

def calculate_singleton_rates_from_sfs_map(sfs_map, lower_threshold=0,
upper_threshold=0.2):

	total_sites = 0 # Total number of sites in SFS
	within_sites = 0 # Frequency < lower_threshold, > upper_threshold
	between_sites = 0 # Frequency within lower_threhsold-upper_threshold

	for D, A in sfs_map.keys():
		n, reverse_n = sfs_map[(D, A)]
		f = A*1.0/D # Frequency of alternate allele

		total_sites += n

		if ((f>0 and f<lower_threshold) or (f>upper_threshold and f<1)):
			# an intermediate frequency site
			within_sites += n
		else:
			if f>0.5:
					between_sites += (n-reverse_n) #NRG what does this do?
			else:
```

```
                                between_sites += reverse_n

        return within_sites, between_sites, total_sites
```

snps_utils.py

```
import config
import numpy
import gzip
import os

def parse_preexisting_snps(species_name):

        intermediate_filename = '%s/preexisting_snps.txt.gz' %
(config.data_directory)

        preexisting_snps = {}
        file = gzip.GzipFile(intermediate_filename,"r")
        for line in file:
                if line.startswith(species_name):
                                contig_items = line.split(";")[1:]
                                for contig_item in contig_items:
                                                contig_item = contig_item.strip()
                                                if contig_item=="":
                                                                continue
                                                contig_subitems =
contig_item.split(":")
                                                contig = contig_subitems[0].strip()
                                                snp_items =
contig_subitems[1].split()
                                                for snp_item in snp_items:
                                                                snp_subitems =
snp_item.split(",")
                                                                location =
long(snp_subitems[0])
                                                                prevalence =
float(snp_subitems[1])


        preexisting_snps[(contig,location)] = prevalence

        file.close()

        return preexisting_snps

def load_private_snv_map(species_name, prev_cohort='all'):

        private_snv_directory = '%s/private_snvs/%s/' % (config.data_directory,
prev_cohort)
        intermediate_filename_template = '%s/%s.txt.gz'
        intermediate_filename = intermediate_filename_template %
(private_snv_directory, species_name)

        private_snv_map = {}

        if not os.path.isfile(intermediate_filename):
                return private_snv_map

        file = gzip.open(intermediate_filename,"r")
        file.readline() # header
        for line in file:
```

```
                    items = line.split(",")

                    contig = items[0].strip()
                    location = long(items[1])
                    gene_name = items[2].strip()
                    variant_type = items[3].strip()
                    host = items[4].strip()

                    private_snv_map[(contig, location)] = (gene_name, variant_type,
host)

       return private_snv_map

def load_snv_distance_map(species_name):

             private_snv_directory = '%s/snv_distances/' % (config.data_directory)
             intermediate_filename_template = '%s/%s.txt.gz'
             intermediate_filename = intermediate_filename_template %
(private_snv_directory, species_name)

             snv_distance_map = {}

             file = gzip.open(intermediate_filename,"r")
             file.readline() # header
             for line in file:

                          items = line.split(",")

                          contig = items[0].strip()
                          location = long(items[1])
                          variant_type = items[2].strip()
                          derived_allele_count = long(items[3])
                          ancestral_allele_count = long(items[4])
                          min_between_d = float(items[5])
                          max_within_d1 = float(items[6])
                          max_within_d2 = float(items[7])

                          snv_distance_map[(contig, location)] = (variant_type,
derived_allele_count, ancestral_allele_count, min_between_d, max_within_d1,
max_within_d2)

             return snv_distance_map

# Loading file
def parse_snp_prevalences(cohort, desired_species_name):

             intermediate_filename_template =
config.data_directory+"snp_prevalences_%s/%s.txt.gz"
             intermediate_filename_alt_template =
config.data_directory+"snp_prevalences_alt_%s/%s.txt.gz"

             if cohort == 'all':
                    intermediate_filename = intermediate_filename_template %
(cohort, desired_species_name)
             else:
                    intermediate_filename = intermediate_filename_alt_template %
(cohort, desired_species_name)

             snp_prevalences = {}
```

```
            if not os.path.isfile(intermediate_filename):
                        return snp_prevalences

            file = gzip.GzipFile(intermediate_filename,"r")
            file.readline()
            for line in file:
                        items = line.split(",")
                        contig = items[0]
                        location = long(items[1])
                        population_freq = float(items[2])
                        snp_freq = float(items[3])

                        snp_prevalences[(contig,location)] = snp_freq


            file.close()

            return snp_prevalences

# originally from calculate_snp_prevalences
def parse_population_freqs(cohort, desired_species_name,
polarize_by_consensus=False):

            intermediate_filename_template =
config.data_directory+"snp_prevalences/%s/%s.txt.gz"
            intermediate_filename_alt_template =
config.data_directory+"snp_prevalences_alt/%s/%s.txt.gz"

            if cohort == 'all':
                  intermediate_filename = intermediate_filename_template %
(cohort, desired_species_name)
            else:
                  intermediate_filename = intermediate_filename_template %
(cohort, desired_species_name)

            population_freqs = {}

            if not os.path.isfile(intermediate_filename):
                        return population_freqs

            file = gzip.GzipFile(intermediate_filename,"r")
            file.readline()
            for line in file:
                        items = line.split(",")
                        contig = items[0]
                        location = long(items[1])
                        population_freq = float(items[2])
                        snp_freq = float(items[3])

                        if polarize_by_consensus:
                                    if population_freq > 0.5:
                                                population_freq = 1-
population_freq

                        if population_freq==0:
                                    pass
                        else:
                                    population_freqs[(contig,location)] =
population_freq

            file.close()
```

```
		return population_freqs

def load_singleton_rate_map(species_name):
# This definition is called whenever another script downstream uses the output of
this data.

		singleton_directory = '%s/singleton_rates/' % (config.data_directory)
		intermediate_filename_template = '%s/%s.txt.gz'
		intermediate_filename = intermediate_filename_template %
(singleton_directory, species_name)

		singleton_rate_map = {}

		if not os.path.isfile(intermediate_filename):
				return singleton_rate_map

		file = gzip.open(intermediate_filename,"r")
		file.readline() # header
		for line in file:
				items = line.split(",")
				if items[0].strip()!=species_name:
						continue

				sample_i = items[1].strip()
				sample_j = items[2].strip()
				type = items[3].strip()
				num_singletons = float(items[4])
				num_doubletons = float(items[5])
				num_differences = float(items[6])
				num_opportunities = float(items[7])

				if type not in singleton_rate_map:
						singleton_rate_map[type] = {}

				if sample_i==sample_j:
						num_singletons = 0
						num_doubletons = 0
						num_differences = 0

				singleton_rate_map[type][sample_i, sample_j] =
(num_singletons, num_doubletons, num_differences, num_opportunities)

		return singleton_rate_map

def calculate_matrices_from_singleton_rate_map(singleton_rate_map, type,
allowed_samples=[]):
# once the map is loaded, then we can compute rate matrices in this definition (so,
it relies on the previous def)

		sample_set = set([])
		for sample in singleton_rate_map[type].keys():
				sample_set.add(sample)

		if len(allowed_samples)>0:
				allowed_sample_set = set(allowed_samples)
		else:
				allowed_sample_set = sample_set

		sample_set = set()
		for sample_i, sample_j in singleton_rate_map[type]:
```

```
                        sample_set.add(sample_i)
                        sample_set.add(sample_j)

            if len(allowed_samples)==0:
                        allowed_samples = list(sorted(allowed_sample_set))


            samples = []
            # preserve same order as allowed samples
            for sample in allowed_samples:
                        if sample in sample_set:
                                    samples.append(sample)

            singleton_matrix = numpy.zeros((len(samples),len(samples)))*1.0
            doubleton_matrix = numpy.zeros_like(singleton_matrix)
            difference_matrix = numpy.zeros_like(singleton_matrix)
            opportunity_matrix = numpy.zeros_like(singleton_matrix)

            for i in xrange(0,len(samples)):
                        for j in xrange(0,len(samples)):

                                    num_singletons, num_doubletons,
num_differences, num_opportunities = singleton_rate_map[type][(samples[i],
samples[j])]

                                    if i==j:
                                                num_doubletons = 0

                                    singleton_matrix[i,j] = num_singletons
                                    doubleton_matrix[i,j] = num_doubletons
                                    difference_matrix[i,j] = num_differences
                                    opportunity_matrix[i,j] = num_opportunities

            #print singleton_matrix, opportunity_matrix
            return samples, singleton_matrix, doubleton_matrix,
difference_matrix, opportunity_matrix


def calculate_matrices_from_substitution_rate_map(substitution_rate_map, type,
allowed_samples=[]):
# once the map is loaded, then we can compute rate matrices in this definition (so,
it relies on the previous def)

            samples, mut_difference_matrix, rev_difference_matrix,
mut_opportunity_matrix, rev_opportunity_matrix =
calculate_mutrev_matrices_from_substitution_rate_map( substitution_rate_map, type,
allowed_samples)

            difference_matrix = mut_difference_matrix+rev_difference_matrix
            opportunity_matrix = mut_opportunity_matrix+rev_opportunity_matrix

            return samples, difference_matrix, opportunity_matrix
```

species_phylogeny_utils.py

```
import parse_midas_data

def get_genus_name(species_name):
    return species_name.split("_")[0]

def get_taxonomy_map():
```

```
    species_genome_map = {}
    genome_species_map = {}

    file = open("%sspecies_info.txt" % parse_midas_data.midas_directory,"r")
    file.readline() # header
    for line in file:
        items = line.split()
        species_name = items[0].strip()
        genome = items[1].strip()
        species_genome_map[species_name] = genome
        genome_species_map[genome] = species_name
    file.close()

    species_taxonomy_map = {}

    file = open("%sgenome_taxonomy.txt" % parse_midas_data.midas_directory,"r")
    file.readline() # header
    for line in file:
        items =  line.split("\t")

        genome_id = items[0].strip()

        kingdom = items[3].strip()
        phylum = items[4].strip()
        class_name = items[5].strip()
        order = items[6].strip()
        family = items[7].strip()
        genus = items[8].strip()

        if genome_id in genome_species_map:

            species_name = genome_species_map[genome_id]
            species_taxonomy_map[species_name] =
(kingdom,phylum,class_name,order,family,genus)

    return species_taxonomy_map


def sort_phylogenetically(species_list, first_entry="",
second_sorting_attribute=[]):

    species_taxonomy_map = get_taxonomy_map()

    kingdom_order_map = {}
    order_kingdom_map = []

    phylum_order_map = {}
    order_phylum_map = []

    class_order_map = {}
    order_class_map = []

    order_name_order_map = {}
    order_order_name_map = []

    family_order_map = {}
    order_family_map = []

    genus_order_map = {}
    order_genus_map = []
```

```
    if first_entry!="":

        kingdom,phylum,class_name,order,family,genus =
species_taxonomy_map[first_entry]

        kingdom_order_map[kingdom] = 0
        order_kingdom_map.append(kingdom)

        phylum_order_map[phylum] = 0
        order_phylum_map.append(phylum)

        class_order_map[class_name] = 0
        order_class_map.append(class_name)

        order_name_order_map[order] = 0
        order_order_name_map.append(order)

        family_order_map[family] = 0
        order_family_map.append(family)

        genus_order_map[genus] = 0
        order_genus_map.append(genus)

    # Now walk through the species
    # to get sorting order
    for species_name in species_list:

        kingdom,phylum,class_name,order,family,genus =
species_taxonomy_map[species_name]

        if kingdom not in kingdom_order_map:
            kingdom_order_map[kingdom] = len(order_kingdom_map)
            order_kingdom_map.append(kingdom)

        if phylum not in phylum_order_map:
            phylum_order_map[phylum] = len(order_phylum_map)
            order_phylum_map.append(phylum)

        if class_name not in class_order_map:
            class_order_map[class_name] = len(order_class_map)
            order_class_map.append(class_name)

        if order not in order_name_order_map:
            order_name_order_map[order] = len(order_order_name_map)
            order_order_name_map.append(order)

        if family not in family_order_map:
            family_order_map[family] = len(order_family_map)
            order_family_map.append(family)

        if genus not in genus_order_map:
            genus_order_map[genus] = len(order_genus_map)
            order_genus_map.append(genus)


    # Now actually do the sort
    order_list = []
    for species_name in species_list:
```

```
        kingdom,phylum,class_name,order,family,genus =
species_taxonomy_map[species_name]
        order_list.append( (kingdom_order_map[kingdom], phylum_order_map[phylum],
class_order_map[class_name], order_name_order_map[order], family_order_map[family],
genus_order_map[genus]) )

    if len(second_sorting_attribute)==0:
        second_sorting_attribute = list(species_list)

    first_entry_list = [1-(species==first_entry) for species in species_list]

    # sort in descending order of sample size
    # Sort by num haploids
    sorted_order_list, sorted_first_entry_list, sorted_second_sorting_attribute,
sorted_species_list = zip(*sorted(zip(order_list, first_entry_list,
second_sorting_attribute, species_list)))

    return sorted_species_list


# Returns a copy of the list of species sorted phylogenetically
def old_sort_phylogenetically(species):


    # Load the species tree from the midas directory
    from Bio import Phylo
    newick_filename = parse_midas_data.midas_directory+"/species_tree.newick"
    tree = Phylo.read(newick_filename, 'newick')
    ordered_species_ids = [term.name for term in tree.get_terminals()]

    # the ordered species IDs are just the numbers at the end of the full species
name
    # so we need to convert the allowed list
    id_name_map = {}
    for species_name in species:
        items = species_name.split("_")
        id = items[-1]
        id_name_map[id] = species_name

    # walk through the sorted species_id list and output the ones that we want
    ordered_species = []
    for species_id in ordered_species_ids:
        if species_id in id_name_map:
            ordered_species.append(id_name_map[species_id])

    return ordered_species


if __name__=='__main__':
    # Test that it works
    good_species = parse_midas_data.parse_good_species_list()
    ordered_good_species = sort_phylogenetically(good_species)

    good_species_set = set(good_species)
    ordered_good_species_set = set(ordered_good_species)

    print ordered_good_species
    print good_species_set - ordered_good_species_set
    print ordered_good_species_set - good_species_set
```

stats_utils.py

```
import numpy
from math import log
from scipy.stats import gamma

####
#
# Calculates median from histogram
# histogram is map of value: counts
#
####
def calculate_median_from_histogram(histogram):

    xs, CDF = calculate_CDF_from_histogram(histogram)
    median_idx = numpy.nonzero(CDF>=0.5)[0][0]
    return xs[median_idx]

####
#
# Calculates median from histogram
# histogram is map of value: counts
#
####
def calculate_nonzero_median_from_histogram(histogram):



    xs, CDF = calculate_CDF_from_histogram(histogram)

    if len(xs)<2:
        return xs[0]

    #if CDF[-1]==0:
    #    return xs[0]


    if xs[0]<0.5:

        if CDF[0]>0.8:
            return xs[0]

        CDF -= CDF[0]
        CDF /= CDF[-1]

    median_idx = numpy.nonzero(CDF>=0.5)[0][0]
    return xs[median_idx]

####
#
# Calculates median from histogram
# histogram is map of value: counts
#
####
def calculate_thresholded_median_from_histogram(histogram,xmin=0):


    xs, CDF = calculate_CDF_from_histogram(histogram)
```

```
    # Get last index below xmin
    idx = numpy.nonzero(xs>xmin+0.5)[0][0]-1

    CDF -= CDF[idx]
    CDF /= CDF[-1]
    CDF = numpy.clip(CDF,0,1e09)

    median_idx = numpy.nonzero(CDF>=0.5)[0][0]
    return xs[median_idx]



####
#
# Calculates CDF from histogram
# histogram is map of value: counts
#
####
def calculate_unnormalized_CDF_from_histogram(histogram):

    xs = sorted(histogram.keys())
    ns = numpy.array([histogram[x] for x in xs])*1.0
    xs = numpy.array(xs)*1.0

    CDF = ns.cumsum()
    return xs, CDF

####
#
# Calculates CDF from histogram
# histogram is map of value: counts
#
####
def calculate_CDF_from_histogram(histogram):

    xs = sorted(histogram.keys())
    ns = numpy.array([histogram[x] for x in xs])*1.0
    xs = numpy.array(xs)*1.0

    CDF = ns.cumsum()/ns.sum()
    return xs, CDF


def calculate_total_from_histogram(histogram):
    xs = sorted(histogram.keys())
    ns = numpy.array([histogram[x] for x in xs])*1.0
    return ns.sum()

####
#
# Calculates unnormalized survival functions (i.e. # of observations >= x)
# from numpy vector of observations
#
####
def calculate_unnormalized_survival_from_vector(xs, min_x=None, max_x=None,
min_p=1e-10, eps=1e-10):

    xs = numpy.array(xs)

    if min_x==None:
```

```
        min_x = xs.min()-1

    if max_x==None:
        max_x = xs.max()+1

    unique_xs = set(xs)
    unique_xs.add(min_x)
    unique_xs.add(max_x)

    xvalues = []
    num_observations = []

    for x in sorted(unique_xs):
        xvalues.append(x)
        num_observations.append( (xs>=x).sum() )

    # So that we can plot CDF, SF on log scale
    num_observations[0] -= min_p
    num_observations[1] -= min_p
    num_observations[-1] += min_p

    return numpy.array(xvalues), numpy.array(num_observations)

def calculate_IQR_from_distribution(xs, ns):

    weights = ns*1.0/ns.sum()
    CDF = numpy.cumsum(weights)
    upper_idx = numpy.nonzero(CDF>=0.75)[0][0]
    lower_idx = numpy.nonzero(CDF>=0.25)[0][0]

    return xs[lower_idx], xs[upper_idx]

def calculate_median_from_distribution(xs, ns):
    weights = ns*1.0/ns.sum()
    CDF = numpy.cumsum(weights)
    mid_idx = numpy.nonzero(CDF>=0.5)[0][0]
    return xs[mid_idx]

####
#
# Calculates IQR (75-25 percentile) from histogram
# histogram is map of value: counts
#
####
def calculate_IQR_from_histogram(histogram):

    xs, CDF = calculate_CDF_from_histogram(histogram)

    upper_idx = numpy.nonzero(CDF>=0.75)[0][0]
    lower_idx = numpy.nonzero(CDF>=0.25)[0][0]

    return xs[upper_idx]-xs[lower_idx]



####
#
# Calculates "confidence intervals" on rate from Poisson distribution
# based on n>=0 counts at L>=0 sites.
#
####
```

```
def calculate_poisson_rate_interval(n,L,alpha=0.5): # by default use a 50%
confidence interval

    if n<0.5:
        # No counts. Have some info on upper bound, but none on lower bound.
        plower = 0
        pupper = log(2/alpha)/L

    else:
        # Posterior distribution is Gamma with shape n-1 and scale 1/L
        # Get confidence intervals from tail probabilities
        plower = gamma.ppf(alpha/2, n)/L
        pupper = gamma.ppf(1-alpha/2,n)/L

    return plower,pupper
```

substitution_rates_utils.py

```
import numpy
import gzip
import os
import config

substitution_rate_directory = '%s/substitution_rates/' % (config.data_directory)
intermediate_filename_template = '%s/%s/%s.txt.gz'

def load_substitution_rate_map(species_name, prev_cohort='all'):

    intermediate_filename = intermediate_filename_template %
(substitution_rate_directory, prev_cohort, species_name)

    substitution_rate_map = {}

    if not os.path.isfile(intermediate_filename):
        return substitution_rate_map

    file = gzip.open(intermediate_filename,"r")
    file.readline() # header
    for line in file:
        items = line.split(",")
        if items[0].strip()!=species_name:
            continue

        record_strs = [", ".join(['Species', 'Sample1', 'Sample2', 'Type',
'Num_muts', 'Num_revs', 'Num_mut_opportunities', 'Num_rev_opportunities'])]

        sample_1 = items[1].strip()
        sample_2 = items[2].strip()
        type = items[3].strip()
        num_muts = float(items[4])
        num_revs = float(items[5])
        num_mut_opportunities = float(items[6])
        num_rev_opportunities = float(items[7])

        num_changes = num_muts+num_revs
        num_opportunities = num_mut_opportunities+num_rev_opportunities

        sample_pair = (sample_1, sample_2)

        if type not in substitution_rate_map:
```

```
            substitution_rate_map[type] = {}

        substitution_rate_map[type][sample_pair] = (num_muts, num_revs,
num_mut_opportunities, num_rev_opportunities)

    return substitution_rate_map

def calculate_mutrev_matrices_from_substitution_rate_map(substitution_rate_map,
type, allowed_samples=[]):
    # Rewritten to preserve order of allowed samples
    # If allowed samples contains things that are not in DB, it returns zero
opportunities

    total_sample_set = set([])
    for sample_1, sample_2 in substitution_rate_map[type].keys():
        total_sample_set.add(sample_1)
        total_sample_set.add(sample_2)

    if len(allowed_samples)==0:
        allowed_samples = list(sorted(total_sample_set))

    # allows us to go from sample name to idx in allowed samples (to preserve
order)
    sample_idx_map = {allowed_samples[i]:i for i in xrange(0,len(allowed_samples))}

    mut_difference_matrix = numpy.zeros((len(allowed_samples),
len(allowed_samples)))*1.0
    rev_difference_matrix = numpy.zeros_like(mut_difference_matrix)

    mut_opportunity_matrix = numpy.zeros_like(mut_difference_matrix)
    rev_opportunity_matrix = numpy.zeros_like(mut_difference_matrix)

    for sample_pair in substitution_rate_map[type].keys():

        sample_i = sample_pair[0]
        sample_j = sample_pair[1]

        if not ((sample_i in sample_idx_map) and (sample_j in sample_idx_map)):
            continue

        i = sample_idx_map[sample_i]
        j = sample_idx_map[sample_j]

        num_muts, num_revs, num_mut_opportunities, num_rev_opportunities =
substitution_rate_map[type][sample_pair]

        mut_difference_matrix[i,j] = num_muts
        rev_difference_matrix[i,j] = num_revs

        mut_opportunity_matrix[i,j] = num_mut_opportunities
        rev_opportunity_matrix[i,j] = num_rev_opportunities

    return allowed_samples, mut_difference_matrix, rev_difference_matrix,
mut_opportunity_matrix, rev_opportunity_matrix


def calculate_matrices_from_substitution_rate_map(substitution_rate_map, type,
allowed_samples=[]):
# once the map is loaded, then we can compute rate matrices in this definition (so,
it relies on the previous def)
```

```
    samples, mut_difference_matrix, rev_difference_matrix, mut_opportunity_matrix,
rev_opportunity_matrix =
calculate_mutrev_matrices_from_substitution_rate_map( substitution_rate_map, type,
allowed_samples)

    difference_matrix = mut_difference_matrix+rev_difference_matrix
    opportunity_matrix = mut_opportunity_matrix+rev_opportunity_matrix

    return samples, difference_matrix, opportunity_matrix
```

temporal_changes_utils.py

```
import sample_utils, config, parse_midas_data, sfs_utils, diversity_utils,
gene_diversity_utils, core_gene_utils
import os, os.path, sys, gzip
import numpy

temporal_change_directory = '%s/temporal_changes/' % (config.data_directory)
intermediate_filename_template = '%s/%s.txt.gz'

min_coverage = config.min_median_coverage
min_sample_size = 2

def load_temporal_change_map(species_name, prev_cohort='all', min_coverage = 20):

            dir = "%s/cov%i_prev_%s" % (temporal_change_directory, min_coverage,
prev_cohort)
            intermediate_filename = intermediate_filename_template % (dir,
species_name)

            temporal_change_map = {}


            if not os.path.isfile(intermediate_filename):
                        return temporal_change_map

            file = gzip.open(intermediate_filename,"r")
            file.readline() # header
            for line in file:
                        items = line.split(",")
                        if items[0].strip()!=species_name:
                                    continue

                        sample_1 = items[1].strip()
                        sample_2 = items[2].strip()
                        type = items[3].strip()
                        num_opportunities = float(items[4])
                        perr = float(items[5])
                        sample_pair = (sample_1, sample_2)
                        if sample_pair not in temporal_change_map:
                                    temporal_change_map[sample_pair] = {}

                        changes = []
                        if len(items)<7:
                                    pass
                        else:
                                    change_strs = items[6:]
                                    for change_str in change_strs:

                                                subitems =
change_str.split(";")
```

```
                                                    # switch on type of change
                                                    if type=='snps':
                                                                gene_name =
subitems[0].strip()
                                                                contig =
subitems[1].strip()
                                                                position =
long(subitems[2])
                                                                variant_type =
subitems[3].strip()
                                                                A1 =
float(subitems[4])
                                                                D1 =
float(subitems[5])
                                                                A2 =
float(subitems[6])
                                                                D2 =
float(subitems[7])

      changes.append( (gene_name, contig, position, variant_type, A1, D1, A2,
D2) )


                                                    elif type=='genes':
                                                                gene_name =
subitems[0].strip()
                                                                D1 =
float(subitems[1])
                                                                Dm1 =
float(subitems[2])
                                                                D2 =
float(subitems[3])
                                                                Dm2 =
float(subitems[4])

      changes.append( (gene_name, D1, Dm1, D2, Dm2) )

                                                    elif type=='private_snps':

                                                                gene_name =
subitems[0].strip()
                                                                contig =
subitems[1].strip()
                                                                position =
long(subitems[2])
                                                                variant_type =
subitems[3].strip()
                                                                A1 =
float(subitems[4])
                                                                D1 =
float(subitems[5])
                                                                A2 =
float(subitems[6])
                                                                D2 =
float(subitems[7])

      changes.append( (gene_name, contig, position, variant_type, A1, D1, A2,
D2) )
```

```
                        temporal_change_map[sample_pair][type] =
num_opportunities, perr, changes

            return temporal_change_map

def calculate_private_reversions_from_temporal_change_map(temporal_change_map,
sample_1, sample_2, lower_threshold=config.consensus_lower_threshold,
upper_threshold=config.consensus_upper_threshold):

            sample_pair = sample_1, sample_2
            if sample_pair not in temporal_change_map:
                        return -1, None, None

            if 'private_snps' not in temporal_change_map[sample_pair]:
                        return -1, None, None

            # otherwise, some hope!

            private_snp_opportunities, private_snp_perr, private_snps =
temporal_change_map[sample_pair]['private_snps']

            mutations = []
            private_snp_reversions = []
            for snp_change in private_snps:

                        a,b,c,d,A1,D1,A2,D2 = snp_change

                        if D1==0 or D2==0:
                                    private_snp_opportunities-=1
                                    continue

                        f1 = A1*1.0/D1
                        f2 = A2*1.0/D2

                        if f1>=upper_threshold and f2<=lower_threshold:
                                    private_snp_reversions.append(snp_change)
                        if f1<=upper_threshold and f2>=upper_threshold:
                                    mutations.append(snp_change)


            return private_snp_opportunities, private_snp_perr,
private_snp_reversions


def calculate_mutations_reversions_from_temporal_change_map(temporal_change_map,
sample_1, sample_2, lower_threshold=config.consensus_lower_threshold,
upper_threshold=config.consensus_upper_threshold):

            sample_pair = sample_1, sample_2
            if sample_pair not in temporal_change_map:
                        return -1, -1, [], []

            if 'snps' not in temporal_change_map[sample_pair]:
                        return -1, -1, [], []

            # otherwise, some hope!
            snp_opportunities, snp_perr, snp_changes =
temporal_change_map[sample_pair]['snps']

            mutations = []
            reversions = []
```

```
        for snp_change in snp_changes:

                a,b,c,d,A1,D1,A2,D2 = snp_change

                f1 = A1*1.0/D1
                f2 = A2*1.0/D2

                if (f1<=lower_threshold) and (f2>=upper_threshold):
                        mutations.append(snp_change)
                elif (f1>=upper_threshold) and (f2<=lower_threshold):
                        reversions.append(snp_change)


        return snp_opportunities, snp_perr, mutations, reversions


def calculate_gains_losses_from_temporal_change_map(temporal_change_map, sample_1,
sample_2, max_absent_copynum=config.gainloss_max_absent_copynum,
min_normal_copynum=config.gainloss_min_normal_copynum,
max_normal_copynum=config.gainloss_max_normal_copynum):


        sample_pair = sample_1, sample_2
        if sample_pair not in temporal_change_map:
                return -1, -1, [], []

        if 'genes' not in temporal_change_map[sample_pair]:
                return -1, -1, [], []

        # otherwise, some hope!
        gene_opportunities, gene_perr, gene_changes =
temporal_change_map[sample_pair]['genes']

        gains = []
        losses = []
        for gene_change in gene_changes:

                gene_name, D1, Dm1, D2, Dm2 = gene_change

                copynum_1 = D1/Dm1
                copynum_2 = D2/Dm2

                if (copynum_1<=max_absent_copynum) and
(copynum_2>=min_normal_copynum) and (copynum_2<=max_normal_copynum):
                        gains.append(gene_change)
                elif (copynum_2<=max_absent_copynum) and
(copynum_1>=min_normal_copynum) and (copynum_1<=max_normal_copynum):
                        losses.append(gene_change)

        return gene_opportunities, gene_perr, gains, losses
```

Scripts to store data for plotting

pickle_core_genes_sharing.py

```
from utils import parse_midas_data as pmd, core_gene_utils as cgu
import config

good_species_list = pmd.parse_good_species_list()

f = open('%s/core_genes_sharing.csv' % config.analysis_directory, 'w')
```

```
f.write((',').join(['species', 'num_total', 'num_hmp', 'num_hmp_infant_shared',
'num_infant', 'num_adult_infant_shared', 'num_adult']))
f.write('\n')

for species in good_species_list:
        print('Working on %s...' % species)
        core_genes_all = cgu.parse_core_genes(species, prev_cohort='all')
        core_genes_hmp = cgu.parse_core_genes(species, prev_cohort='hmp')
        core_genes_infant = cgu.parse_core_genes(species, prev_cohort='infant')
        core_genes_adult = cgu.parse_core_genes(species, prev_cohort='adult')

        core_genes_hmp_infant_shared =
core_genes_hmp.intersection(core_genes_infant)
        core_genes_adult_infant_shared =
core_genes_adult.intersection(core_genes_infant)

        data_items = [species, len(core_genes_all), len(core_genes_hmp),
len(core_genes_hmp_infant_shared), len(core_genes_infant),
len(core_genes_adult_infant_shared), len(core_genes_adult)]

        row_str = (',').join([str(item) for item in data_items])
        f.write(row_str + '\n')

f.close()
```

pickle_details.py

```
# Script to pickle very specific information (allele identity and counts)
# for small set of given QP pairs at given sites

from utils import sample_utils as su, parse_midas_data, substitution_rates_utils, config,
temporal_changes_utils, snps_utils, core_gene_utils, gene_diversity_utils
import numpy as np
from numpy.random import choice, random as np_random, randint
import random
from collections import defaultdict
import pickle
import bz2
import numpy
import os, sys

# Desired samples and sites
# sites are given as gene_id, contig, location tuples
# inconsistent but bleh

desired_host_species_sites = [(['M0806-C'], \
  'Bacteroides_vulgatus_57955', \
  [('435590.9.peg.1499', 'NC_009614', 1915720L), \
   ('435590.9.peg.1499', 'NC_009614', 1915720L)]), \
 (['3-I'], \
  'Bifidobacterium_catenulatum_58257', \
  [('566552.4.peg.96', 'NZ_ABXY01000001', 124565L), \
   ('566552.4.peg.96', 'NZ_ABXY01000001', 124565L)]), \
 (['67-I'], \
  'Bacteroides_fragilis_54507', \
  [('1339327.3.peg.4421', 'JGDJ01000264', 9480L), \
   ('1339327.3.peg.4421', 'JGDJ01000264', 9480L)]), \
 (['M0901-C'], \
  'Blautia_wexlerae_56130', \
```

```
  [('1121115.4.peg.32', 'AXVN01000001', 35625L), \
   ('1121115.4.peg.32', 'AXVN01000001', 35625L)]), \
 (['1-I'], \
  'Bacteroides_fragilis_54507', \
  [('1339327.3.peg.2283', 'JGDJ01000171', 5687L), \
   ('1339327.3.peg.2283', 'JGDJ01000171', 5687L)])]

# desired_host_species_sites = [(['C02143-I', 'C02143-M'], 'Bifidobacterium_bifidum_55065',
[('500634.3.peg.1861','AWSW01000054',37945), ('500634.3.peg.945','AWSW01000030',35925),
('500634.3.peg.1636','AWSW01000046',21960), ('500634.3.peg.875','AWSW01000027',7916),
('500634.3.peg.952','AWSW01000030',45351), ('500634.3.peg.1619','AWSW01000046',4226)]), \
# (['59-I', '59-M'], 'Bifidobacterium_adolescentis_56815',
[('592977.3.peg.642','JGZQ01000005',14361),('592977.3.peg.860','JGZQ01000006',69020),('592977.3.
peg.1216','JGZQ01000008',284119),('592977.3.peg.1129','JGZQ01000008',186577),
('592977.3.peg.39','JGZQ01000001',53358), ('592977.3.peg.1732','JGZQ01000009',58203),
('592977.3.peg.1705','JGZQ01000009',29026)])]

# desired_samples = ['ERR3405741', 'ERR3405661', 'ERR3406235']

# ========================================================================
# Loads allele counts for specific samples at specific sites
# where sites are provided as (contig, location, gene_id) tuples
# TODO: move to parse_midas_data later
# ========================================================================

def parse_snps_specify_sites_details(species_name, desired_samples=[], desired_sites=[],
prev_cohort='all'):

        # Alternate version without gene names
        desired_sites_no_gene = [(contig, location) for contig, location, gene_id in
desired_sites]

        # SNPs directory
        snps_dir = "%s/snps/%s/" % (config.data_directory, species_name)

        # Load population freqs (for polarization purposes)
        population_freqs = snps_utils.parse_population_freqs(prev_cohort, species_name,
polarize_by_consensus=False)

        # Open post-processed MIDAS output
        snp_file = bz2.BZ2File("%s/annotated_snps.txt.bz2" % snps_dir, 'r')

        # ========================================================================
        # Process allele information
        # ========================================================================

        # sample -> site -> (ref allele, alt allele)
        sample_site_allele_dict = defaultdict(dict)

        # Load snps_alt_allele.txt
        snp_alleles_file = bz2.BZ2File("%s/snps_alt_allele.txt.bz2" % snps_dir, 'r')
        items = snp_alleles_file.readline().strip().split()[1:]
        samples = list(su.parse_merged_sample_names(items))

        desired_sample_idxs = []
        for sample in desired_samples:
                if sample in samples:
                        desired_sample_idxs.append(samples.index(sample))

        desired_sample_idxs = numpy.array(sorted(desired_sample_idxs))
```

```
        for line in snp_alleles_file:
                items = line.split()

                # Load information about site
                info_items = items[0].split("|")
                contig = info_items[0]
                location = long(info_items[1])
                ref_allele = info_items[2]

                if (contig, location) not in desired_sites_no_gene:
                        continue

                for idx in desired_sample_idxs:
                        alt_allele = items[1+idx]
                        sample = samples[idx]
                        sample_site_allele_dict[sample][(contig, location)] = (ref_allele,
alt_allele)

        snp_alleles_file.close()

        # ====================================================================
        # Process annotated_snps information
        # ====================================================================

        # Open post-processed MIDAS output
        snp_file = bz2.BZ2File("%s/annotated_snps.txt.bz2" % snps_dir, 'r')

        # Get lists of desired sample idxs
        items = snp_file.readline().strip().split()[1:]
        samples = list(su.parse_merged_sample_names(items))

        desired_sample_idxs = []
        for sample in desired_samples:
                if sample in samples:
                        desired_sample_idxs.append(samples.index(sample))

        desired_sample_idxs = numpy.array(sorted(desired_sample_idxs))

        # Map: sample -> site (contig, location, gene_id) -> allele count
        allele_counts_map = defaultdict(dict)

        # Map: site (contig, location, gene_id) -> variant type
        variant_type_map = {}

        num_sites_processed = 0

        # Loop over sites in annotated_snps.txt file
        for line in snp_file:
                if num_sites_processed>0 and num_sites_processed%10000==0:
                        sys.stderr.write("%d0k sites processed...\n" %
(num_sites_processed/10000))
                num_sites_processed += 1

                items = line.split()

                # Load information about site
                info_items = items[0].split("|")
                contig = info_items[0]
                location = long(info_items[1])
                gene_name = info_items[2]
                variant_type = info_items[3]
```

```
                polarization = 'R' # note R was assigned indiscriminately
                pvalue = float(info_items[5])

                # Only look at sites of interest
                if (contig, location, gene_name) not in desired_sites:
                        continue

                # Store variant type
                variant_type_map[(contig, location, gene_name)] = variant_type

                # Store alt and depth counts at this site for all desired samples
                for idx in desired_sample_idxs:
                        alt, depth = [float(num) for num in items[1+idx].split(",")]
                        sample = samples[idx]
                        allele_counts_map[sample][(contig, location, gene_name)] = (alt, depth)

        snp_file.close()

        return allele_counts_map, sample_site_allele_dict, variant_type_map

# Load a few things
subject_sample_map = su.parse_subject_sample_map()

# Set up pickle directory

ddir = config.data_directory
pdir = "%s/pickles/reversion_examples/" % (ddir)
os.system('mkdir -p %s' % pdir)

# Store these two dicts for each host-species pair
for subjects, species, sites in desired_host_species_sites:

        # Get all samples within the host
        desired_samples = []
        for subject in subjects:
                desired_samples += subject_sample_map[subject].keys()

        # Reformat sites
        desired_sites = []
        for gene_id, contig, location in sites:
                desired_sites.append((contig, location, gene_id))

        # Obtain desired dicts
        allele_counts_map, sample_site_allele_dict, variant_type_map =
parse_snps_specify_sites_details(species, desired_samples, desired_sites=desired_sites)

        # Pickle dicts
        sys.stderr.write("Pickling...\n")
        pickle.dump((allele_counts_map, sample_site_allele_dict, variant_type_map),
open('%s/allele_info_%s_%s.pkl' % (pdir, ('_').join(subjects), species), 'wb'))

sys.stderr.write("Done!\n")
```

pickle_everything.py

```
from utils import sample_utils as su, parse_midas_data, substitution_rates_utils, config,
temporal_changes_utils, snps_utils, core_gene_utils, gene_diversity_utils
import numpy as np
from numpy.random import choice, random as np_random, randint
```

```
import random
from collections import defaultdict
import pickle
import os, sys

# ======================================================
# Examines all consecutive timepoint pairs within hosts
# across all cohorts, and pickles SNP/gene change info
# ======================================================

# Parameters
sweep_type = 'full' # assume full for now
pp_prev_cohort = 'all'
min_coverage = 0
thresholds = {'full': (0.2, 0.8), 'partial': (0.35, 0.65)}
lower_threshold, upper_threshold = thresholds[sweep_type]

clade_divergence_threshold = 3e-02 # TODO: change to top level clade definition later

min_sample_size = 3
variant_types = ['1D','4D']
within_host_type = 'nonconsecutive' # consecutive timepoints
min_snp_change_sample_size = 5

# For partitioning SNVs according to prevalence
derived_freq_bins = np.array([-1,0,0.01,0.1,0.5,0.9,0.99,1,2])
derived_virtual_freqs = np.arange(0,len(derived_freq_bins)-1)
derived_virtual_xticks = list(derived_virtual_freqs[:-1]+0.5)
derived_virtual_xticklabels = ['0','.01','.1','.5','.9','.99','1']

# For partitioning genes into different prevalence classes
gene_freq_bins = np.array([-1,0.1,0.5,0.9,2])
gene_freq_xticks                  = [-4, -3,       -2,              -1,              0,       1,
                  2,              3, 4]
gene_freq_xticklabels = ['0','0.1','0.5', '0.9','1','0.9','0.5', '0.1','0']
gene_gain_virtual_freqs = np.array([3.5,2.5,1.5,0.5])
gene_loss_virtual_freqs = np.array([-3.5,-2.5,-1.5,-0.5])

# Sample-subject-order maps
sys.stderr.write("Loading sample metadata...\n")
subject_sample_map = su.parse_subject_sample_map()
sample_order_map = su.parse_sample_order_map()
sample_subject_map = su.parse_sample_subject_map()
sys.stderr.write("Done!\n")

# Timepoint pair types
tp_pair_names = ['MM', 'MI', 'II', 'AA']

# Cohorts
cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
mi_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

# Prevalence cohorts
prev_cohorts = ['all', 'hmp', 'infant', 'nonpremie', 'mother']

# Samples for each cohort
samples = {cohort: su.get_sample_names(cohort) for cohort in cohorts}
hmp_samples = su.get_sample_names('hmp')
mother_samples = su.get_sample_names('mother')
infant_samples = su.get_sample_names('infant')
```

```
# Species list
good_species_list = parse_midas_data.load_pickled_good_species_list()

def get_sweep_prevalence(snp_change, snv_freq_map, private_snv_map):

                gene_name, contig, position, variant_type, A1, D1, A2, D2 = snp_change

                f1 = A1*1.0/D2
                f2 = A2*1.0/D2

                is_reversion = (f1>f2)

                location_tuple = (contig, position)

                is_private_snv = (location_tuple in private_snv_map)


                # Now calculate frequency-stratified version

                if location_tuple in snv_freq_map:
                                f = snv_freq_map[location_tuple]
                else:
                                sys.stderr.write("SNP not in map. Shouldn't happen!\n")
                                f = -0.5

                # Let's impose that private snvs have zero freq (specifically, lim 0^-)

                if is_private_snv:
                                f = -0.5

                # Change f so that it represents
                # frequency of allele at second timepoint
                if is_reversion:
                                f = 1-f

                return f

# ====================================================================
# Species SNP/gene change distributions

# species -> (sample1, sample2) -> [list of SNP change tuples]
# OR # SNP changes if that number exceeds 20 (replacement)
# where each SNP change tuple consists of
# (gene_name, contig, position, variant_type, A1, D1, A2, D2)
snp_changes = {species: {} for species in good_species_list}

# species -> (sample1, sample2) -> [nonsyn_diffs, nonsyn_opps, syn_diffs, syn_opps]
dnds_info = {species: {} for species in good_species_list}

# species -> (sample1, sample2) -> (gene gain tuples, gene loss tuples)
# OR (# gene gains, # gene losses) if replacement event
# where each gene change tuple consists of...
gene_changes = {species: {} for species in good_species_list}

# species -> (sample1, sample2) -> [list of (variant_type, prevalence, opps) tuples]
# only for modification events
snp_change_freqs = {species: defaultdict(list) for species in good_species_list}
# species -> (sample1, sample2) -> prev_cohort -> [list of (prevalence, weight) tuples]
snp_change_null_freqs = {species: defaultdict(dict) for species in good_species_list}

# species -> (sample1, sample2) -> [list of prevalences]
```

```
# only for modification events
gene_gain_freqs = {species: defaultdict(list) for species in good_species_list}
gene_loss_freqs = {species: defaultdict(list) for species in good_species_list}
gene_loss_null_freqs = {species: defaultdict(list) for species in good_species_list}

# species -> (sample1, sample2) -> random number of SNP/gene differences between sample1 and
unrelated host
# note that sample2 doesn't really mean anything here
between_snp_change_counts = {species: {} for species in good_species_list}
between_gene_change_counts = {species: {} for species in good_species_list}

# 3 null distributions of: for each SNP change...
# present gene: assign random gene that is present at either timepoint
# between host: assign random gene that changes between hosts
# pangenome: assign random gene from pangenome
# ...
# species -> (sample1, sample2) -> gene ID -> # occurrence (normalized by num_trials)
snp_change_present_gene_null = {species: defaultdict(dict) for species in good_species_list}
snp_change_between_host_null = {species: defaultdict(dict) for species in good_species_list}
snp_change_pangenome_null = {species: defaultdict(dict) for species in good_species_list}

# ====================================================================

for species_name in good_species_list[::-1]:

        sys.stderr.write("\nProcessing %s...\n" % species_name)

        # Grab QP samples for this species
        qp_sample_lists = {}
        for cohort in cohorts:
                qp_sample_lists[cohort] = sorted(su.load_qp_samples(samples[cohort],
species_name, prev_cohort=pp_prev_cohort)['qp'])

        combined_qp_samples = sorted(su.flatten([qp_sample_lists[cohort] for cohort in
cohorts]))
        combined_sample_name_idx_map = {combined_qp_samples[i] : i for i in
range(len(combined_qp_samples))}

        # Using all QP samples to threshold on sample size
        if len(combined_qp_samples) < min_sample_size:
                sys.stderr.write("Not enough haploid samples!\n")
                continue

        # Load substitution rates for all QP samples
        sys.stderr.write("Loading pre-computed substitution rates for %s...\n" % species_name)
        substitution_rate_map =
substitution_rates_utils.load_substitution_rate_map(species_name, prev_cohort=pp_prev_cohort)

        if substitution_rate_map == {}: # Not enough haploid samples
                sys.stderr.write("Not enough haploid samples!\n")
                continue

        sys.stderr.write("Calculating SNV matrix...\n")
        dummy_samples, snp_mut_difference_matrix, snp_rev_difference_matrix,
snp_mut_opportunity_matrix, snp_rev_opportunity_matrix =
substitution_rates_utils.calculate_mutrev_matrices_from_substitution_rate_map(substitution_rate_
map, 'all', allowed_samples=combined_qp_samples)

        snp_difference_matrix = snp_mut_difference_matrix + snp_rev_difference_matrix
        snp_opportunity_matrix = snp_mut_opportunity_matrix+snp_rev_opportunity_matrix
```

```
        snp_substitution_rate =
snp_difference_matrix*1.0/(snp_opportunity_matrix+(snp_opportunity_matrix==0))
        sys.stderr.write("Done!\n")

        sys.stderr.write("Loading gene matrix...\n")
        gene_samples, gene_loss_difference_matrix, gene_gain_difference_matrix,
gene_loss_opportunity_matrix, gene_gain_opportunity_matrix =
substitution_rates_utils.calculate_mutrev_matrices_from_substitution_rate_map(substitution_rate_
map, 'genes', allowed_samples=combined_qp_samples)
        gene_difference_matrix = gene_gain_difference_matrix + gene_loss_difference_matrix
        gene_opportunity_matrix = gene_loss_opportunity_matrix
        gene_difference_matrices = {'gains': gene_gain_difference_matrix, 'losses':
gene_loss_difference_matrix}
        sys.stderr.write("Done!\n")

        sys.stderr.write("Loading 1D & 4D opportunity matrices...\n")
        difference_matrices, opportunity_matrices = {}, {}
        for var_type in variant_types:
                matrix_samples, difference_matrix, opportunity_matrix =
substitution_rates_utils.calculate_matrices_from_substitution_rate_map(substitution_rate_map,
var_type, allowed_samples=combined_qp_samples)
                difference_matrices[var_type] = difference_matrix
                opportunity_matrices[var_type] = opportunity_matrix

        sys.stderr.write("Done!\n")

        # Load temporal change map
        sys.stderr.write("Loading pre-computed temporal changes...\n")
        temporal_change_map = temporal_changes_utils.load_temporal_change_map(species_name,
prev_cohort=pp_prev_cohort, min_coverage=min_coverage) # Default min coverage 20
        sys.stderr.write("Done!\n")

        # Load private SNV map
        private_snv_map = snps_utils.load_private_snv_map(species_name,
prev_cohort=pp_prev_cohort)

        # Load prevalences
        snv_freq_map = {prev_cohort: snps_utils.parse_population_freqs(prev_cohort,
species_name, polarize_by_consensus=True) for prev_cohort in prev_cohorts}

        # Load gene frequencies
        gene_freq_map = {}
        for prev_cohort in prev_cohorts:
                gene_freqs = core_gene_utils.parse_gene_freqs(species_name,
prev_cohort=prev_cohort)
                if len(gene_freqs) != 0:
                        gene_freq_map[prev_cohort] = gene_freqs
        gene_freq_values = {prev_cohort: np.array(gene_freq_map[prev_cohort].values()) for
prev_cohort in gene_freq_map}
        gene_freq_weights = {prev_cohort:
(gene_freq_values[prev_cohort]*1.0/gene_freq_values[prev_cohort].sum()) for prev_cohort in
gene_freq_map}

        # Load info for pangenome null
        pangenome_gene_names, pangenome_new_species_names =
parse_midas_data.load_pangenome_genes(species_name)

        # Load data for between-host changes null
        gene_samples, gene_names, gene_presence_matrix, gene_depth_matrix, marker_coverages,
gene_reads_matrix =
parse_midas_data.parse_pangenome_data(species_name,allowed_samples=combined_qp_samples)
```

```
        gene_copynum_matrix = gene_depth_matrix*1.0/(marker_coverages+(marker_coverages==0))

        # Loop over different cohorts
        for cohort in cohorts:
                desired_samples = qp_sample_lists[cohort]

                # These indices are w.r.t. desired_samples
                same_subject_idxs =
su.calculate_mi_ordered_same_subject_pairs(sample_order_map, desired_samples,
within_host_type=within_host_type, one_per_mi_pair=False)
                diff_subject_idxs = su.calculate_ordered_diff_subject_pairs(sample_order_map,
desired_samples)

                # These indices are w.r.t. combined_qp_samples, so that we can
                # properly index into opportunity matrices
                combined_sample_idx_map = su.calculate_sample_idx_map(desired_samples,
combined_qp_samples)
                combined_same_subject_idxs =
su.apply_sample_index_map_to_indices(combined_sample_idx_map, same_subject_idxs)
                combined_diff_subject_idxs =
su.apply_sample_index_map_to_indices(combined_sample_idx_map, diff_subject_idxs)

                # These indices are w.r.t. gene_samples, so that we can
                # properly index into pangenome data
                gene_sample_idx_map = su.calculate_sample_idx_map(desired_samples,
gene_samples)
                gene_same_subject_idxs =
su.apply_sample_index_map_to_indices(gene_sample_idx_map, same_subject_idxs)
                gene_diff_subject_idxs =
su.apply_sample_index_map_to_indices(gene_sample_idx_map, diff_subject_idxs)

                # This specifically gets all genes that differ between arbitrary hosts
                # within this cohort, as long as they each have a QP sample with sufficient
                # marker coverage and low enough SNP substitution rate
                between_host_gene_idxs = []
                for idx in range(len(combined_diff_subject_idxs[0])):
                        combined_i = combined_diff_subject_idxs[0][idx]
                        combined_j = combined_diff_subject_idxs[1][idx]
                        gene_i = gene_diff_subject_idxs[0][idx]
                        gene_j = gene_diff_subject_idxs[1][idx]
                        if (marker_coverages[gene_i]>min_coverage) and
(marker_coverages[gene_j]>min_coverage):
                                if snp_substitution_rate[combined_i, combined_j] <
clade_divergence_threshold:
                                        gene_idxs =
gene_diversity_utils.calculate_gene_differences_between_idxs(combined_i,combined_j,
gene_reads_matrix, gene_depth_matrix, marker_coverages)
                                        between_host_gene_idxs.extend(gene_idxs)

                # Loop over different pairs of within-host samples
                for sample_pair_idx in range(len(same_subject_idxs[0])):

                                sample_i =
desired_samples[same_subject_idxs[0][sample_pair_idx]]
                                sample_j =
desired_samples[same_subject_idxs[1][sample_pair_idx]]

                                sample_i_combined_idx =
combined_same_subject_idxs[0][sample_pair_idx]
                                sample_j_combined_idx =
combined_same_subject_idxs[1][sample_pair_idx]
```

```
                                sample_i_gene_idx = gene_same_subject_idxs[0][sample_pair_idx]
                                sample_j_gene_idx = gene_same_subject_idxs[1][sample_pair_idx]

                                subject = (sample_subject_map[sample_i],
sample_subject_map[sample_j])
                                tp_pair = su.sample_pair_to_tp_pair(sample_i, sample_j,
sample_order_map, hmp_samples, mother_samples)

                                i = combined_sample_name_idx_map[sample_i]
                                j = combined_sample_name_idx_map[sample_j]

                                matrix_idx_i = matrix_samples.index(sample_i)
                                matrix_idx_j = matrix_samples.index(sample_j)

                                # Set up data for present gene null
                                # Obtain the indices of all genes which have copynum 0.5-2.0
                                # at either timepoint (duplicate if show up in both samples)
                                present_gene_idxs = []

        present_gene_idxs.extend(np.nonzero((gene_copynum_matrix[:,sample_i_gene_idx]>0.5)*(gen
e_copynum_matrix[:,sample_i_gene_idx]<2))[0])

        present_gene_idxs.extend(np.nonzero((gene_copynum_matrix[:,sample_j_gene_idx]>0.5)*(gen
e_copynum_matrix[:,sample_j_gene_idx]<2))[0])

                                # Checks if among those samples from different hosts,
                                # at least one of them has nonzero SNP and gene opportunities
                                good_idxs =
su.calculate_samples_in_different_subjects(sample_subject_map, combined_qp_samples, sample_i)
                                good_idxs *= ((snp_opportunity_matrix[i,:]>0.5) *
(gene_opportunity_matrix[i,:]>0.5))

                                # FIRST FILTER
                                if good_idxs.sum() < 1:
                                        sys.stderr.write("Not enough other-host samples!\n")
                                        continue

                                matrix_idx_i = matrix_samples.index(sample_i)
                                matrix_idx_j = matrix_samples.index(sample_j)

                                # Numbers of site differences and opportunities between the
timepoints
                                nonsyn_diffs =
difference_matrices['1D'][matrix_idx_i][matrix_idx_j]
                                nonsyn_opps =
opportunity_matrices['1D'][matrix_idx_i][matrix_idx_j]
                                syn_diffs =
difference_matrices['4D'][matrix_idx_i][matrix_idx_j]
                                syn_opps =
opportunity_matrices['4D'][matrix_idx_i][matrix_idx_j]

                                # SNP temporal changes
                                L, perr, mutations, reversions =
temporal_changes_utils.calculate_mutations_reversions_from_temporal_change_map(temporal_change_m
ap, sample_i, sample_j, lower_threshold=lower_threshold, upper_threshold=upper_threshold)

                                # SECOND FILTER
                                if L<config.min_opportunities:
                                        sys.stderr.write("Not enough SNP opportunities (should
be >=100,000)!\n")
```

```
                                        continue

                                nerr = L*perr

                                num_mutations = len(mutations)
                                num_reversions = len(reversions)
                                num_snp_changes = num_mutations + num_reversions

                                # Gene temporal changes
                                gene_L, gene_perr, gains, losses =
temporal_changes_utils.calculate_gains_losses_from_temporal_change_map(temporal_change_map,
sample_i, sample_j) #, min_normal_copynum = 0.6, max_normal_copynum = 1.2)

                                gene_nerr = gene_L*gene_perr
                                num_gains = len(gains)
                                num_losses = len(losses)
                                num_gene_changes = num_gains+num_losses

                                # THIRD FILTER
                                if (perr<-0.5) or (gene_perr < -0.5):
                                        sys.stderr.write("Perr too high!\n")
                                        continue

                                # FOURTH FILTER
                                if (nerr > max([0.5, 0.1*num_snp_changes])) or (gene_nerr >
max([0.5, 0.1*num_gene_changes])):
                                        sys.stderr.write("Nerr too high!\n")
                                        continue # Only take things with low-ish FPR

                                #
================================================================
                                # Store information
                                #
================================================================
                                '''
                                # dNdS information
                                dnds_info[species_name][(sample_i, sample_j)] = [nonsyn_diffs,
nonsyn_opps, syn_diffs, syn_opps]

                                # Nulls
                                num_trials = 100
                                normalized_count = (1/float(num_trials))
                                snp_change_present_gene_null[species_name][(sample_i,
sample_j)] = defaultdict(int)
                                snp_change_between_host_null[species_name][(sample_i,
sample_j)] = defaultdict(int)
                                snp_change_pangenome_null[species_name][(sample_i, sample_j)]
= defaultdict(int)

                                # If no genes available for a null, skip this QP pair
                                if len(between_host_gene_idxs) > 0 and
len(present_gene_idxs) > 0:

                                        for trial in range(0,num_trials):
                                                # In each trial, get a random list of genes
that change according to respective null
                                                present_gene_null_names =
gene_names[choice(present_gene_idxs, num_snp_changes)]
                                                between_gene_null_names =
gene_names[choice(between_host_gene_idxs, num_snp_changes)]
```

```
                                                pangenome_gene_null_names =
choice(list(pangenome_gene_names), num_snp_changes)

                                                # Then add counts normalized by total number
of trails
                                                for gene in present_gene_null_names:

        snp_change_present_gene_null[species_name][(sample_i, sample_j)][gene] +=
normalized_count

                                                for gene in between_gene_null_names:

        snp_change_between_host_null[species_name][(sample_i, sample_j)][gene] +=
normalized_count

                                                for gene in pangenome_gene_null_names:

        snp_change_pangenome_null[species_name][(sample_i, sample_j)][gene] += normalized_count

                                    # Number of SNP differences with random unrelated host
                                    between_snp_change_counts[species_name][(sample_i, sample_j)]
= choice(snp_difference_matrix[i, good_idxs])
                                    between_gene_change_counts[species_name][(sample_i, sample_j)]
= choice(gene_difference_matrix[i, good_idxs])
                                    '''
                                    if num_snp_changes <= 20: # Modification event

                                            snp_changes[species_name][(sample_i, sample_j)] =
(mutations + reversions) #!
                                            gene_changes[species_name][(sample_i, sample_j)] =
(gains, losses) #!


                                            # Delve into prevalence of modified SNPs and genes
                                            null_freq_dict = defaultdict(list)

                                            for snp_change in (mutations + reversions):

                                                    variant_type = snp_change[3]

                                                    # Construct freq_dict (prevalence of sweeping
allele)
                                                    freq_dict = {}
                                                    for prev_cohort in prev_cohorts:
                                                            f = get_sweep_prevalence(snp_change,
snv_freq_map[prev_cohort], private_snv_map)
                                                            freq_dict[prev_cohort] = f


                                                    # Construct opp_dict (number of 1D/4D
opportunities for the QP pair)
                                                    opp_dict = {'1D':
opportunity_matrices['1D'][i,j], '4D': opportunity_matrices['4D'][i,j]}

                                                    # Store prevalence and opportunity info
                                                    snp_change_freqs[species_name][(sample_i,
sample_j)].append((variant_type, freq_dict, opp_dict)) #!

                                                    # Draw null prevalence from genome
                                                    for prev_cohort in prev_cohorts:


                                                            snv_freq_keys =
snv_freq_map[prev_cohort].keys()
```

```
                                                        L = snp_opportunity_matrix[i,j]
                                                        L_snv =
len(snv_freq_map[prev_cohort]) # A slight overestimate
                                                        snv_fraction = L_snv*1.0/L
                                                        num_bootstraps = 10

                                                        for _ in range(num_bootstraps):

                                                                if np_random() <
snv_fraction: # Polymorphic site

                                                                        random_snv_location =
snv_freq_keys[randint(0, len(snv_freq_keys))]
                                                                        f = 0 if
random_snv_location in private_snv_map else snv_freq_map[prev_cohort][random_snv_location]
                                                                        rev_f = 1-f

                                                                        # Now add in
probability weight

        null_freq_dict[prev_cohort].append((f, (1-f)*1.0/num_bootstraps))

        null_freq_dict[prev_cohort].append((1-f, f*1.0/num_bootstraps))

                                                                else: # A truly invariant
site


        null_freq_dict[prev_cohort].append((0, 1.0/num_bootstraps))

                                        # Store the dictionary of null prevalences by
prevalence cohort
                                        snp_change_null_freqs[species_name][(sample_i,
sample_j)] = null_freq_dict

                                        for gene_change in gains:

                                                gene_name = gene_change[0]

                                                freq_dict = {}
                                                for prev_cohort in gene_freq_map:
                                                        f =
gene_freq_map[prev_cohort][gene_name] if gene_name in gene_freq_map[prev_cohort] else 0
                                                        freq_dict[prev_cohort] = f

                                                gene_gain_freqs[species_name][(sample_i,
sample_j)].append(freq_dict) #!

                                        for gene_change in losses:

                                                gene_name = gene_change[0]

                                                freq_dict = {}
                                                null_freq_dict = {}
                                                num_bootstraps = 10

                                                for prev_cohort in gene_freq_map:
                                                        f =
gene_freq_map[prev_cohort][gene_name] if gene_name in gene_freq_map[prev_cohort] else 0
                                                        freq_dict[prev_cohort] = f
```

```
                                                        fs =
choice(gene_freq_values[prev_cohort], size=num_bootstraps, p=gene_freq_weights[prev_cohort])
                                                        null_freq_dict[prev_cohort] = fs



                                                gene_loss_freqs[species_name][(sample_i,
sample_j)].append(freq_dict) #!

                                                gene_loss_null_freqs[species_name][(sample_i,
sample_j)].append(null_freq_dict)

                                    else: # Likely replacement and too many SNPs to store info for
                                            snp_changes[species_name][(sample_i, sample_j)] =
num_snp_changes
                                            gene_changes[species_name][(sample_i, sample_j)] =
(num_gains, num_losses)


# Pickle time
sys.stderr.write("Pickling...\n")

ddir = config.data_directory
pdir = "%s/pickles/cov%i_prev_%s/nonconsecutive/" % (ddir, min_coverage, pp_prev_cohort)
os.system('mkdir -p %s' % pdir)

'''
pickle.dump(snp_changes, open('%s/big_snp_changes_%s.pkl' % (pdir, sweep_type), 'wb'))
pickle.dump(gene_changes, open('%s/big_gene_changes_%s.pkl' % (pdir, sweep_type), 'wb'))
'''
pickle.dump(snp_change_freqs, open('%s/snp_change_freqs_with_opps_%s.pkl' % (pdir, sweep_type),
'wb'))
pickle.dump(snp_change_null_freqs, open('%s/snp_change_null_freqs_%s.pkl' % (pdir, sweep_type),
'wb'))
pickle.dump(gene_gain_freqs, open('%s/gene_gain_freqs_%s.pkl' % (pdir, sweep_type), 'wb'))
pickle.dump(gene_loss_freqs, open('%s/gene_loss_freqs_%s.pkl' % (pdir, sweep_type), 'wb'))
pickle.dump(gene_loss_null_freqs, open('%s/gene_loss_null_freqs_%s.pkl' % (pdir, sweep_type),
'wb'))
'''
pickle.dump(between_snp_change_counts, open('%s/between_snp_change_counts_%s.pkl' % (pdir,
sweep_type), 'wb'))
pickle.dump(between_gene_change_counts, open('%s/between_gene_change_counts_%s.pkl' % (pdir,
sweep_type), 'wb'))

pickle.dump(snp_change_present_gene_null, open('%s/snp_change_present_gene_null.pkl' % pdir,
'wb'))
pickle.dump(snp_change_between_host_null, open('%s/snp_change_between_host_null.pkl' % pdir,
'wb'))
pickle.dump(snp_change_pangenome_null, open('%s/snp_change_pangenome_null.pkl' % pdir, 'wb'))
pickle.dump(dnds_info, open('%s/dnds_info.pkl' % pdir, 'wb'))
'''

sys.stderr.write("Done!\n")
```

pickle_ferretti_snp_changes.py

```
from utils import sample_utils as su, parse_midas_data, substitution_rates_utils, config,
temporal_changes_utils, snps_utils, core_gene_utils, gene_diversity_utils
import numpy as np
```

```
from numpy.random import choice, random as np_random, randint
import random
from collections import defaultdict
import pickle
import os, sys

# ======================================================
# Examines all consecutive timepoint pairs within hosts
# across all cohorts, and pickles SNP/gene change info
# ======================================================

# Parameters
sweep_type = 'full' # assume full for now
pp_prev_cohort = 'all'
min_coverage = 0
thresholds = {'full': (0.2, 0.8), 'partial': (0.35, 0.65)}
lower_threshold, upper_threshold = thresholds[sweep_type]

clade_divergence_threshold = 3e-02 # TODO: change to top level clade definition later

min_sample_size = 3
variant_types = ['1D','4D']
within_host_type = 'nonconsecutive' # consecutive timepoints
min_snp_change_sample_size = 5

# For partitioning SNVs according to prevalence
derived_freq_bins = np.array([-1,0,0.01,0.1,0.5,0.9,0.99,1,2])
derived_virtual_freqs = np.arange(0,len(derived_freq_bins)-1)
derived_virtual_xticks = list(derived_virtual_freqs[:-1]+0.5)
derived_virtual_xticklabels = ['0','.01','.1','.5','.9','.99','1']

# For partitioning genes into different prevalence classes
gene_freq_bins = np.array([-1,0.1,0.5,0.9,2])
gene_freq_xticks                  = [-4, -3,       -2,              -1,              0,       1,
                  2,              3, 4]
gene_freq_xticklabels = ['0','0.1','0.5', '0.9','1','0.9','0.5', '0.1','0']
gene_gain_virtual_freqs = np.array([3.5,2.5,1.5,0.5])
gene_loss_virtual_freqs = np.array([-3.5,-2.5,-1.5,-0.5])

# Sample-subject-order maps
sys.stderr.write("Loading sample metadata...\n")
subject_sample_map = su.parse_subject_sample_map()
sample_order_map = su.parse_sample_order_map()
sample_subject_map = su.parse_sample_subject_map()
sys.stderr.write("Done!\n")

# Timepoint pair types
tp_pair_names = ['MM', 'MI', 'II', 'AA']

# Cohorts
cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
mi_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

# Prevalence cohorts
prev_cohorts = ['all', 'hmp', 'infant', 'nonpremie', 'mother']

# Samples for each cohort
samples = {cohort: su.get_sample_names(cohort) for cohort in cohorts}
hmp_samples = su.get_sample_names('hmp')
mother_samples = su.get_sample_names('mother')
infant_samples = su.get_sample_names('infant')
```

```
# Species list
good_species_list = parse_midas_data.load_pickled_good_species_list()

# ================================================================
# Species SNP/gene change distributions

# species -> (sample1, sample2) -> # SNP differences
snp_changes = {species: {} for species in good_species_list}

# ================================================================

for species_name in good_species_list[::-1]:

        sys.stderr.write("\nProcessing %s...\n" % species_name)

        # Do not restrict to QP samples!

        # Load temporal change map
        sys.stderr.write("Loading pre-computed temporal changes...\n")
        temporal_change_map = temporal_changes_utils.load_temporal_change_map(species_name,
prev_cohort=pp_prev_cohort, min_coverage=min_coverage) # Default min coverage 20
        sys.stderr.write("Done!\n")

        # Load private SNV map
        private_snv_map = snps_utils.load_private_snv_map(species_name,
prev_cohort=pp_prev_cohort)

        # Get all mother-infant comparisons
        cohort = 'ferretti'

        desired_samples = su.get_sample_names(cohort)

        # These indices are w.r.t. desired_samples
        same_subject_idxs = su.calculate_mi_ordered_same_subject_pairs(sample_order_map,
desired_samples, within_host_type='nonconsecutive', one_per_mi_pair=False)
        diff_subject_idxs = su.calculate_ordered_diff_subject_pairs(sample_order_map,
desired_samples)

        # Loop over different pairs of within-host samples
        for sample_pair_idx in range(len(same_subject_idxs[0])):

                        sample_i = desired_samples[same_subject_idxs[0][sample_pair_idx]]
                        sample_j = desired_samples[same_subject_idxs[1][sample_pair_idx]]

                        '''
                        # Checks if among those samples from different hosts,
                        # at least one of them has nonzero SNP and gene opportunities
                        good_idxs =
su.calculate_samples_in_different_subjects(sample_subject_map, combined_qp_samples, sample_i)
                        good_idxs *= ((snp_opportunity_matrix[i,:]>0.5) *
(gene_opportunity_matrix[i,:]>0.5))

                        # FIRST FILTER
                        if good_idxs.sum() < 1:
                                sys.stderr.write("Not enough other-host samples!\n")
                                continue
                        '''

                        # SNP temporal changes
```

```
                        L, perr, mutations, reversions =
temporal_changes_utils.calculate_mutations_reversions_from_temporal_change_map(temporal_change_m
ap, sample_i, sample_j, lower_threshold=lower_threshold, upper_threshold=upper_threshold)
                        '''
                        # SECOND FILTER
                        if L<config.min_opportunities:
                                sys.stderr.write("Not enough SNP opportunities (should
be >=100,000)!\n")
                                continue
                        '''
                        nerr = L*perr

                        num_mutations = len(mutations)
                        num_reversions = len(reversions)
                        num_snp_changes = num_mutations + num_reversions

                        # Gene temporal changes
                        gene_L, gene_perr, gains, losses =
temporal_changes_utils.calculate_gains_losses_from_temporal_change_map(temporal_change_map,
sample_i, sample_j) #, min_normal_copynum = 0.6, max_normal_copynum = 1.2)

                        gene_nerr = gene_L*gene_perr
                        num_gains = len(gains)
                        num_losses = len(losses)
                        num_gene_changes = num_gains+num_losses
                        '''
                        # THIRD FILTER
                        if (perr<-0.5) or (gene_perr < -0.5):
                                sys.stderr.write("Perr too high!\n")
                                continue

                        # FOURTH FILTER
                        if (nerr > max([0.5, 0.1*num_snp_changes])) or (gene_nerr > max([0.5,
0.1*num_gene_changes])):
                                sys.stderr.write("Nerr too high!\n")
                                continue # Only take things with low-ish FPR
                        '''
                        # ================================================================
                        # Store information
                        # ================================================================
                        snp_changes[species_name][(sample_i, sample_j)] = num_snp_changes


# Pickle time
sys.stderr.write("Pickling...\n")

ddir = config.data_directory
pdir = "%s/pickles/cov%i_prev_%s/nonconsecutive/" % (ddir, min_coverage, pp_prev_cohort)
os.system('mkdir -p %s' % pdir)

pickle.dump(snp_changes, open('%s/ferretti_all_snp_changes_%s.pkl' % (pdir, sweep_type), 'wb'))

sys.stderr.write("Done!\n")
```

pickle_mom_seeding.py

```
# Question: are sweeping alleles in infants present in mom?
# Idea: store data on allele frequency in mom for each sweeping allele in infant
```

```
from utils import sample_utils as su, parse_midas_data, substitution_rates_utils,
config, temporal_changes_utils, snps_utils, core_gene_utils, gene_diversity_utils
import numpy as np
from numpy.random import choice, random as np_random, randint
import random
from collections import defaultdict
import pickle
import bz2
import numpy
import os, sys

# ====================================================================
# Loads allele counts for specific samples at specific sites
# where sites are provided as (contig, location, gene_id) tuples
# TODO: move to parse_midas_data later
# ====================================================================

def parse_snps_specify_sites(species_name, desired_samples=[], desired_sites=[],
prev_cohort='all'):

        # Load population freqs (for polarization purposes)
        population_freqs = snps_utils.parse_population_freqs(prev_cohort,
species_name, polarize_by_consensus=False)

        # Open post-processed MIDAS output
        snps_dir = "%s/snps/%s/" % (config.data_directory, species_name)
        snp_file = bz2.BZ2File("%s/annotated_snps.txt.bz2" % snps_dir, 'r')

        # Get lists of desired sample idxs
        items = snp_file.readline().strip().split()[1:]
        samples = list(su.parse_merged_sample_names(items))

        desired_sample_idxs = []
        for sample in desired_samples:
                if sample in samples:
                        desired_sample_idxs.append(samples.index(sample))

        desired_sample_idxs = numpy.array(sorted(desired_sample_idxs))

        # Map: sample -> site (contig, location, gene_id) -> allele count
        allele_counts_map = defaultdict(dict)

        num_sites_processed = 0

        # Loop over sites in annotated_snps.txt file
        for line in snp_file:
                if num_sites_processed>0 and num_sites_processed%10000==0:
                        sys.stderr.write("%d0k sites processed...\n" %
(num_sites_processed/10000))
                num_sites_processed += 1

                items = line.split()

                # Load information about site
                info_items = items[0].split("|")
                contig = info_items[0]
                location = long(info_items[1])
                gene_name = info_items[2]
                polarization = 'R' # note R was assigned indiscriminately
                pvalue = float(info_items[5])
```

```
                # Only look at sites of interest
                if (contig, location, gene_name) not in desired_sites:
                        continue

                # Load alt and depth counts at this site for all desired samples
                alts, depths = [], []

                for idx in desired_sample_idxs:
                        alt, depth = [float(num) for num in
items[1+idx].split(",")]
                        alts.append(alt); depths.append(depth)

                alts = numpy.array(alts)
                depths = numpy.array(depths)

                # Obtain population frequency of alt allele
                # Recall: this is average proportion of majority-alt samples
across subjects
                if (contig, location) in population_freqs:
                        population_freq = population_freqs[(contig, location)]
                else: # alt population prevalence is (probably? TODO) 0
                        population_freq = 0

                # Polarize SFS according to population freq
                if population_freq > 0.5: # This means alt allele is the major
allele
                        alts = depths - alts
                        polarization = 'A'

                # For sites from temporal changes, note that we should assume
                # site is "passed" i.e. alt aleles make up >5% of all alleles
                # in at least one sample and pvalue < 0.05

                # Store allele counts only if the site is interesting
                for alt, depth, sample in zip(alts, depths, desired_samples):
                        allele_counts_map[sample][(contig, location, gene_name)] =
(alt, depth)

        snp_file.close()

        return allele_counts_map

# Parameters
species = sys.argv[1]
sweep_type = 'full' # assume full for now
pp_prev_cohort = 'all'
min_coverage = 0
thresholds = {'full': (0.2, 0.8), 'partial': (0.35, 0.65)}
lower_threshold, upper_threshold = thresholds[sweep_type]

# Sample-subject-order maps
sys.stderr.write("Loading sample metadata...\n")
subject_sample_map = su.parse_subject_sample_map()
sample_order_map = su.parse_sample_order_map()
sample_subject_map = su.parse_sample_subject_map()
sample_cohort_map = su.parse_sample_cohort_map()
same_mi_pair_dict = su.get_same_mi_pair_dict(subject_sample_map)
sys.stderr.write("Done!\n")

# Use output from pickle_everything.py to get list of sweeping alleles in infants
ddir = config.data_directory
```

```
pdir = "%s/pickles/cov%i_prev_%s/nonconsecutive/" % (ddir, min_coverage,
pp_prev_cohort)
snp_changes = pickle.load(open('%s/big_snp_changes_%s.pkl' % (pdir, sweep_type),
'rb'))
'''
# Loop over all modification SNP changes
for species in snp_changes:

        # Form list of sites and samples of interest
        desired_sites = set()
        desired_samples = set()

        for s1, s2 in snp_changes[species]:

                val = snp_changes[species][(s1, s2)]

                # Interested in all samples within the mother-infant pair
                # Note that mothers and infants differ by 2-letter suffix of
subject
                subject = sample_subject_map[s1]
                for sample in subject_sample_map[subject]:
                        desired_samples.add(sample)

                if subject in same_mi_pair_dict:
                        other_subject = same_mi_pair_dict[subject]
                        for sample in subject_sample_map[other_subject]:
                                desired_samples.add(sample)

                if type(val) == type([]): # Modification event
                        for snp_change in val:
                                gene_name, contig, position, variant_type, A1,
D1, A2, D2 = snp_change
                                desired_sites.add((contig, position, gene_name))

        # Load allele_counts_map
        allele_counts_map = parse_snps_specify_sites(species, desired_samples,
desired_sites=list(desired_sites))
'''
# Form list of sites and samples of interest
desired_sites = set()
desired_samples = set()

for s1, s2 in snp_changes[species]:

        val = snp_changes[species][(s1, s2)]

        # Only look at mothers and infants excluding olm
        cohort = sample_cohort_map[s1]
        if cohort not in ['backhed', 'ferretti', 'yassour', 'shao']:
                continue

        # Interested in all samples within the mother-infant pair
        # Note that mothers and infants differ by 2-letter suffix of subject
        subject = sample_subject_map[s1]
        for sample in subject_sample_map[subject]:
                desired_samples.add(sample)

        if subject in same_mi_pair_dict:
                other_subject = same_mi_pair_dict[subject]
                for sample in subject_sample_map[other_subject]:
                        desired_samples.add(sample)
```

```
        if type(val) == type([]): # Modification event
                for snp_change in val:
                        gene_name, contig, position, variant_type, A1, D1, A2, D2
= snp_change
                        desired_sites.add((contig, position, gene_name))

# Load allele_counts_map
allele_counts_map = parse_snps_specify_sites(species, desired_samples,
desired_sites=list(desired_sites))

# Pickle time
sys.stderr.write("Pickling...\n")

ddir = config.data_directory
pdir = "%s/pickles/seeding/" % (ddir)
os.system('mkdir -p %s' % pdir)

pickle.dump(allele_counts_map, open('%s/allele_counts_map_%s.pkl' % (pdir,
species), 'wb'))

sys.stderr.write("Done!\n")
```

pickle_pnps.py

```
import matplotlib
matplotlib.use('Agg')
from utils import parse_midas_data, sample_utils,  config, sfs_utils,
diversity_utils, stats_utils
import sys, os.path, numpy
from math import log10,ceil
import matplotlib.pyplot as plt
from collections import defaultdict
from utils.classes import Interval

type = sys.argv[1] # common, rare, etc.

# Parameters
min_coverage = 20 # orig 20
common_freqrange = [Interval('[0.2, 0.8]')]
rare_freqrange = [Interval('(0, 0.1]'), Interval('[0.9, 1)')]
seg_freqrange = [Interval('(0, 1)')]

if type == 'common':
        freqrange = common_freqrange
elif type == 'rare':
        freqrange = rare_freqrange
elif type == 'seg':
        freqrange = seg_freqrange

# Good species list
good_species_list = parse_midas_data.load_pickled_good_species_list()

# Dictionary: sample -> species -> (within_1D, total_1D, within_4D, total_4D)
within_total_sites_QP = defaultdict(dict)
within_total_sites_nonQP = defaultdict(dict)

# Dictionary: sample -> species -> pN/pS
pNpS_QP = defaultdict(dict)
```

```
pNpS_nonQP = defaultdict(dict)

for species in good_species_list:

        # Load SNP information for this species
        sys.stderr.write("Loading SFSs for %s...\t" % species)
        samples_4D, sfs_map_4D = parse_midas_data.parse_within_sample_sfs(species,
allowed_variant_types=set(['4D'])) # synonymous
        samples_1D, sfs_map_1D = parse_midas_data.parse_within_sample_sfs(species,
allowed_variant_types=set(['1D'])) # nonsynonymous
        sys.stderr.write("Done!\n")

        # Load genomic coverage distributions
        sample_coverage_histograms, samples =
parse_midas_data.parse_coverage_distribution(species)
        samples = numpy.array(samples)
        median_coverages =
numpy.array([stats_utils.calculate_nonzero_median_from_histogram(hist) for hist in
sample_coverage_histograms])
        sample_median_coverage_map = {samples[i]: median_coverages[i] for i in
range(len(samples))}

        # Get QP samples (note: low coverage samples are excluded)
        qp_sample_dict = sample_utils.calculate_qp_samples(samples, species)
        samples_QP = qp_sample_dict['qp']
        samples_nonQP = qp_sample_dict['non-qp']

        # Only plot samples above a certain median coverage threshold (100)
        desired_samples = samples[(median_coverages >= min_coverage)]
        desired_median_coverages = numpy.array([sample_median_coverage_map[sample]
for sample in desired_samples])

        if len(desired_samples) <= 0:
                continue

        # ====================================
        # Calculate within polymorphism rates
        # ====================================

        # Final list of samples used (filtered for nonzero total site counts)
        sample_names = []

        # Sites with freq 0-0.05 (rare alleles)
        between_rates_1D = []
        between_rates_4D = []

        # Sites with freq = 0, freq > 0.05
        within_rates_1D = []
        within_rates_4D = []

        within_sites_1D_array=[]
        total_sites_1D_array=[]
        within_sites_4D_array=[]
        total_sites_4D_array=[]

        for sample in desired_samples:

                total_sites_1D, within_sites_1D =
sfs_utils.calculate_sites_within_freq_range_from_sfs_map(sfs_map_1D[sample],
freqrange)
```

```
                total_sites_4D, within_sites_4D =
sfs_utils.calculate_sites_within_freq_range_from_sfs_map(sfs_map_4D[sample],
freqrange)

                # Skip if zero of either syn. or nonsyn. total sites
                if total_sites_1D <= 0 or total_sites_4D <= 0:
                        continue

                # Fraction of all nonsynonymous sites with minor allele
frequency > 0.05
                pN = (within_sites_1D*1.0 + 1.0)/(total_sites_1D + 1.0)

                # Fraction of all synonymous sites with minor allele frequency >
0.05
                pS = (within_sites_4D*1.0 + 1.0)/(total_sites_4D + 1.0)

                # Store within and total sites, pN/pS for each sample-species pair
                if sample in samples_QP:
                        within_total_sites_QP[sample][species] = (within_sites_1D,
total_sites_1D, within_sites_4D, total_sites_4D)
                        pNpS_QP[sample][species] = pN/pS

                elif sample in samples_nonQP:
                        within_total_sites_nonQP[sample][species] =
(within_sites_1D, total_sites_1D, within_sites_4D, total_sites_4D)
                        pNpS_nonQP[sample][species] = pN/pS

# Pickle!!
import pickle

pdir = "%s/pickles" % config.data_directory

pickle.dump(within_total_sites_nonQP,
open("%s/within_total_sites_%s_nonQP_cov20_rare10pct.pkl" % (pdir, type), 'wb'))
pickle.dump(within_total_sites_QP,
open("%s/within_total_sites_%s_QP_cov20_rare10pct.pkl" % (pdir, type), 'wb'))
pickle.dump(pNpS_nonQP, open("%s/pNpS_%s_nonQP.pkl" % (pdir, type), 'wb'))
pickle.dump(pNpS_QP, open("%s/pNpS_%s_QP.pkl" % (pdir, type), 'wb'))
```

pickle_qp.py

```
from utils import parse_midas_data, sample_utils as su, config
from collections import defaultdict

# "Categories" (may include Olm, HMP later)
cats = ['mother', 'infant', 'hmp']

# Load species and samples list
good_species_list = parse_midas_data.load_pickled_good_species_list()
samples = {cat: su.get_sample_names(cat) for cat in cats}

# ============================================================================
# Consolidate QP information
# ============================================================================

# Map: mother / infant -> samples
sample_species_qp_dict = {cat: defaultdict(dict) for cat in cats}

for cat in cats:
```

```
        for species in good_species_list:
                print("Working on species %s..." % species)
                qp_sample_sets = su.load_qp_samples(samples[cat], species)
                for qp_status in ['qp', 'non-qp', 'low-coverage']:
                        for sample in qp_sample_sets[qp_status]:
                                sample_species_qp_dict[cat][sample][species] =
qp_status

# ====================================================================
# Pickle
# ====================================================================

import pickle
pickle.dump(sample_species_qp_dict, open("%s/pickles/sample_species_qp_dict.pkl" %
config.data_directory, 'wb'))
```

pickle_transmission.py

```
from utils import config, parse_midas_data, sample_utils as su, temporal_changes_utils,
stats_utils, midas_db_utils, parse_patric, snps_utils
from collections import defaultdict
import math, random, numpy as np
import pickle, sys, bz2, os
import matplotlib.pyplot as plt

# ====================================================
# Examines all consecutive timepoint pairs within hosts
# across all cohorts, and pickles SNP/gene change info
# ====================================================

# Parameters
sweep_type = 'full' # assume full for now
pp_prev_cohort = 'all'
min_coverage = 0
thresholds = {'full': (0.2, 0.8), 'partial': (0.35, 0.65)}
lower_threshold, upper_threshold = thresholds[sweep_type]

clade_divergence_threshold = 3e-02 # TODO: change to top level clade definition later

min_sample_size = 3
variant_types = ['1D','4D']
within_host_type = 'nonconsecutive' # consecutive timepoints
min_snp_change_sample_size = 5

# Sample-subject-order maps
sys.stderr.write("Loading sample metadata...\n")
subject_sample_map = su.parse_subject_sample_map()
sample_order_map = su.parse_sample_order_map()
sample_subject_map = su.parse_sample_subject_map()
sys.stderr.write("Done!\n")

# Cohorts
cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
mi_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

# Samples for each cohort
samples = {cohort: su.get_sample_names(cohort) for cohort in cohorts}
hmp_samples = su.get_sample_names('hmp')
mother_samples = su.get_sample_names('mother')
```

```
infant_samples = su.get_sample_names('infant')
olm_samples = su.get_sample_names('olm')
infant_samples_no_olm = [sample for sample in infant_samples if sample not in olm_samples]

# Sample-timepoint map
mi_sample_day_dict = su.get_mi_sample_day_dict(exclude_cohorts=['olm'])

# Consider all mother/infant samples here
desired_samples = mother_samples + infant_samples
# These indices are w.r.t. desired_samples
same_subject_idxs = su.calculate_mi_ordered_same_subject_pairs(sample_order_map,
desired_samples, within_host_type='consecutive', one_per_mi_pair=False,
infant_timepoint_pref='first')
diff_subject_idxs = su.calculate_ordered_diff_subject_pairs(sample_order_map, desired_samples)

# Species list
good_species_list = parse_midas_data.load_pickled_good_species_list()

# Relative abundance file
relab_fpath = "%s/species/relative_abundance.txt.bz2" % (config.data_directory)
relab_file = open(relab_fpath, 'r')
decompressor = bz2.BZ2Decompressor()
raw = decompressor.decompress(relab_file.read())
data = [row.split('\t') for row in raw.split('\n')]
data.pop() # Get rid of extra element due to terminal newline
header = su.parse_merged_sample_names(data[0]) # species_id, samples...

# Load species presence/absence information
sample_species_dict = defaultdict(set)

for row in data[1:]:
    species = row[0]
    for relab_str, sample in zip(row[1:], header[1:]):
        relab = float(relab_str)
        if relab > 0:
            sample_species_dict[sample].add(species)

# Custom mother-infant test
is_mi = lambda sample_i, sample_j: ((sample_i in mother_samples and sample_j in
infant_samples_no_olm) and mi_sample_day_dict[sample_i] >= 0 and mi_sample_day_dict[sample_j] <=
7)

# ================================================================
# (sample1, sample2) -> species -> (shared/not_shared, # SNP differences)
# if not_shared, # SNP differences should be -1
transmission_info = defaultdict(dict)
# ================================================================

for species_name in good_species_list[::-1]:

        sys.stderr.write("\nProcessing %s...\n" % species_name)

        # Do not restrict to QP samples! However, keep high coverage filter later

        # Load temporal change map
        sys.stderr.write("Loading pre-computed temporal changes...\n")
        temporal_change_map = temporal_changes_utils.load_temporal_change_map(species_name,
prev_cohort=pp_prev_cohort, min_coverage=min_coverage) # Default min coverage 20
        sys.stderr.write("Done!\n")

        # Loop over different pairs of within-host samples
```

```
        for sample_pair_idx in range(len(same_subject_idxs[0])):

                        sample_i = desired_samples[same_subject_idxs[0][sample_pair_idx]]
                        sample_j = desired_samples[same_subject_idxs[1][sample_pair_idx]]

                        if not is_mi(sample_i, sample_j):
                                continue

                        # Shared species
                        shared_species =
sample_species_dict[sample_i].intersection(sample_species_dict[sample_j])

                        # SNP temporal changes
                        L, perr, mutations, reversions =
temporal_changes_utils.calculate_mutations_reversions_from_temporal_change_map(temporal_change_m
ap, sample_i, sample_j, lower_threshold=lower_threshold, upper_threshold=upper_threshold)

                        # FILTER
                        if L<config.min_opportunities:
                                sys.stderr.write("Not enough SNP opportunities (should
be >=100,000)!\n")
                                continue

                        nerr = L*perr
                        num_mutations = len(mutations)
                        num_reversions = len(reversions)
                        num_snp_changes = num_mutations + num_reversions

                        # ================================================================
                        # Store information
                        # ================================================================
                        shared_status = 'shared' if species_name in shared_species else
'not_shared'
                        transmission_info[(sample_i, sample_j)][species_name] = (shared_status,
num_snp_changes)


# Pickle time
sys.stderr.write("Pickling...\n")

ddir = config.data_directory
pdir = "%s/pickles/cov%i_prev_%s/" % (ddir, min_coverage, pp_prev_cohort)
os.system('mkdir -p %s' % pdir)

pickle.dump(transmission_info, open('%s/transmission_info.pkl' % (pdir), 'wb'))

sys.stderr.write("Done!\n")

# Get dem numbers

num_transmissions = 0
num_total = 0
num_shared_species_all_vals = []
num_shared_species_highcov_vals = []
shared_species_highcov_agg = []
half_prop_transmissions = []

for sample_i, sample_j in transmission_info:

        # Shared species
```

```
        shared_species =
sample_species_dict[sample_i].intersection(sample_species_dict[sample_j])

        num_shared_species_all = len(shared_species)
        num_shared_species_highcov = 0
        num_transmission = 0
        num_replacement = 0
        shared_species_highcov = set()

        for species in transmission_info[(sample_i, sample_j)]:

                shared_status, num_snp_changes = transmission_info[(sample_i,
sample_j)][species]

                if shared_status == 'shared':
                        num_shared_species_highcov += 1

                shared_species_highcov.add(species)
                num_total += 1

                if num_snp_changes <= 20: # Transmission
                        num_transmission += 1
                        num_transmissions += 1
                elif num_snp_changes > 20: # Replacement
                        num_replacement += 1

        shared_species_highcov_agg += list(shared_species_highcov)
        prop_transmission = float(num_transmission)/float(num_shared_species_highcov)
        half_prop_transmission = float(num_transmission)/float(2*num_shared_species_highcov)
        half_prop_transmissions.append(half_prop_transmission)
        num_shared_species_all_vals.append(num_shared_species_all)
        num_shared_species_highcov_vals.append(num_shared_species_highcov)

shared_species_highcov_count = defaultdict(int)

for species in shared_species_highcov_agg:
        shared_species_highcov_count[species] += 1
```

Plotting scripts

plot_figure_1.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[15]:


from matplotlib import pyplot as plt
import config, sample_utils as su, parse_midas_data as pmd, plot_utils
from collections import defaultdict
import bz2, sys
import numpy as np
import pickle
import math
import scipy.stats
import copy


# In[2]:
```

```
# Pickle directory
pickle_dir = "%s/pickles" % config.data_directory

# Plot directory
plot_dir = "%s/" % (config.analysis_directory)


# In[3]:


# Load pickles
sample_species_polymorphism_dict = pickle.load(open("%s/sample_species_polymorphism_dict.pkl" %
(pickle_dir), 'rb'),
                                                encoding='latin1')

# Load Qin polymorphism data and add to dictionary

f = open('%s/Qin_polymorphism.csv' % config.data_directory)
samples = f.readline().strip().split(',')[1:]
for line in f:
    items = line.strip().split(',')
    species = items[0]
    for sample, val in zip(samples, items[1:]):
        if val != '':
            sample_species_polymorphism_dict[sample][species] = float(val)
f.close()


# In[4]:


# Qin metadata
qin_sample_subject_map = {}
qin_subject_sample_map = {}
qin_samples = []

qin_ids_fpath = "%s/qin_ids.txt" % (config.metadata_directory)
with open(qin_ids_fpath, 'r') as qin_file:
    header = qin_file.readline()
    for line in qin_file:
        subject, _, sample, _, _, _ = line.strip().split('\t')
        qin_sample_subject_map[sample] = subject
        qin_subject_sample_map[subject] = sample
        qin_samples.append(sample)


# In[5]:


# Load subject and sample metadata
sample_order_map = su.parse_sample_order_map()
subject_sample_map = su.parse_subject_sample_map()
sample_subject_map = su.parse_sample_subject_map()
sample_cohort_map = su.parse_sample_cohort_map()

all_samples = su.get_sample_names('all') # Note: c's removed
hmp_samples = su.get_sample_names('hmp')
olm_samples = su.get_sample_names('olm')
mother_samples_orig = su.get_sample_names('mother')
```

```
infant_samples = [sample for sample in su.get_sample_names('infant') if sample not in
olm_samples]

mi_tp_sample_dict, infant_tps_ordered = su.get_mi_tp_sample_dict(exclude_cohorts = ['olm'],
binned = True)
mother_tps_ordered = sorted(mi_tp_sample_dict['mother'].keys())
tps_ordered_dict = {'mother': mother_tps_ordered, 'infant': infant_tps_ordered}
mi_sample_day_dict = su.get_mi_sample_day_dict(exclude_cohorts=['olm'])

# Remove -92 and 92
mother_tps_ordered.remove(-92)
mother_tps_ordered.remove(92)
mother_samples = []
for sample in mother_samples_orig:
    if mi_sample_day_dict[sample] == -92 or mi_sample_day_dict[sample] == 92:
        continue
    mother_samples.append(sample)

# Bootleg load HMP female subjects
f = open('%s/HMP1-2_female_subjects.txt' % config.metadata_directory, 'r')
hmp_female_subjects = [line.strip() for line in f]

# Bootleg load Qin gender
f = open('%s/Qin_metadata.txt' % config.metadata_directory)
header = f.readline()
qin_sample_gender_dict = {}
for line in f:
    subject, gender, age = line.strip().split('\t')
    subject = subject.replace(' ', '-')
    age = int(age)
    if subject in qin_subject_sample_map:
        sample = qin_subject_sample_map[subject]
        qin_sample_gender_dict[sample] = gender

# Species list
species_list = pmd.parse_species_list()
good_species_list = pmd.load_pickled_good_species_list()

# Utility functions
def round_down(num, divisor):
    return num - (num % divisor)

# function to calculate Cohen's d for independent samples
def cohenD(d1, d2):
    n1, n2 = len(d1), len(d2)
    s1, s2 = np.var(d1, ddof=1), np.var(d2, ddof=1)
    s = np.sqrt(((n1 - 1) * s1 + (n2 - 1) * s2) / (n1 + n2 - 2))
    u1, u2 = np.mean(d1), np.mean(d2)
    return (u1 - u2) / s


# In[6]:


# Plot function
def plot_interval(y, xstart, xstop, color='b', tickh=0.1):
    """Plot interval at y from xstart to xstop with given color."""
    plt.hlines(y, xstart, xstop, color, lw=1)
    plt.vlines(xstart, y+tickh, y-tickh, color, lw=1)
    plt.vlines(xstop, y+tickh, y-tickh, color, lw=1)
```

```
# Plot function on specific ax
def plot_interval_on_ax(ax, y, xstart, xstop, color='b', tickh=0.1):
    """Plot interval at y from xstart to xstop with given color."""
    ax.hlines(y, xstart, xstop, color, lw=1)
    ax.vlines(xstart, y+tickh, y-tickh, color, lw=1)
    ax.vlines(xstop, y+tickh, y-tickh, color, lw=1)


# In[7]:


# Alpha diversity, richness at different timepoints

# Relative abundance file
relab_fpath = "%s/species/relative_abundance.txt.bz2" % (config.data_directory)
relab_file = bz2.open(relab_fpath, 'rt')
data = [line.split('\t') for line in relab_file]
header = su.parse_merged_sample_names(data[0])

# Generate alpha diversity dictionary
alpha_div_dict = {}
richness_dict = {}
for i in range(1, len(header)):
    acc = 0
    richness = 0
    for row in data[1:]:
        rel_ab = float(row[i])
        if rel_ab != 0:
            acc += (rel_ab * math.log(rel_ab))
            richness += 1
    alpha_div_dict[header[i]] = (acc*-1)
    richness_dict[header[i]] = richness


# In[8]:


# Store relative abundances

# Relative abundance file
relab_fpath = "%s/species/relative_abundance.txt.bz2" % (config.data_directory)
relab_file = bz2.open(relab_fpath, 'rt')
data = [line.split('\t') for line in relab_file]
header = su.parse_merged_sample_names(data[0])

relab_dict = defaultdict(dict) # sample -> species -> relab

for i in range(1, len(header)):
    sample = header[i]
    for row in data[1:]:
        species = row[0]
        rel_ab = float(row[i])
        if rel_ab > 0:
            relab_dict[sample][species] = rel_ab


# In[9]:


# Poyet, Korpela, Qin adult data too
```

```
korpela_data_dir = '/u/home/d/daisyche/dbd/data_korpela/'
relab_file = bz2.open(relab_fpath, 'rt')
data = [line.split('\t') for line in relab_file]
header = su.parse_merged_sample_names(data[0])

# Generate alpha diversity dictionary
korpela_alpha_div_dict = {}
korpela_richness_dict = {}
for i in range(1, len(header)):
    acc = 0
    richness = 0
    for row in data[1:]:
        rel_ab = float(row[i])
        if rel_ab != 0:
            acc += (rel_ab * math.log(rel_ab))
            richness += 1
    korpela_alpha_div_dict[header[i]] = (acc*-1)
    korpela_richness_dict[header[i]] = richness

poyet_data_dir = '/u/home/d/daisyche/dbd/data_poyet/'
relab_fpath = "%s/species/relative_abundance.txt.bz2" % (poyet_data_dir)
relab_file = bz2.open(relab_fpath, 'rt')
data = [line.split('\t') for line in relab_file]
header = su.parse_merged_sample_names(data[0])

# Generate alpha diversity dictionary
poyet_alpha_div_dict = {}
poyet_richness_dict = {}
for i in range(1, len(header)):
    acc = 0
    richness = 0
    for row in data[1:]:
        rel_ab = float(row[i])
        if rel_ab != 0:
            acc += (rel_ab * math.log(rel_ab))
            richness += 1
    poyet_alpha_div_dict[header[i]] = (acc*-1)
    poyet_richness_dict[header[i]] = richness

qin_data_dir = '/u/home/d/daisyche/mother_infant/Qin_species/'
relab_fpath = "%s/relative_abundance.txt.bz2" % (qin_data_dir)
relab_file = bz2.open(relab_fpath, 'rt')
data = [line.split('\t') for line in relab_file]
header = su.parse_merged_sample_names(data[0])

# Generate alpha diversity dictionary
qin_alpha_div_dict = {}
qin_richness_dict = {}
for i in range(1, len(header)):
    sample = header[i]
    if sample not in qin_sample_subject_map:
        continue
    acc = 0
    richness = 0
    for row in data[1:]:
        rel_ab = float(row[i])
        if rel_ab != 0:
            acc += (rel_ab * math.log(rel_ab))
            richness += 1
    qin_alpha_div_dict[sample] = (acc*-1)
    qin_richness_dict[sample] = richness
```

```
# In[13]:


# Make CSV to make linear model
# alpha diversity ~ pregnancy (mother vs. adult) + cohort
# ONLY INCLUDE FEMALE THIS TIME
output_file = open('%s/alpha_div_preg_status_female_only.csv' % config.analysis_directory, 'w')
output_file.write(','.join(['sample', 'subject', 'alpha_div', 'preg_status', 'cohort']) + '\n')

for sample in alpha_div_dict:
    subject = sample_subject_map[sample]

    # Skip infants and non-female adults
    if sample in hmp_samples and subject not in hmp_female_subjects:
        continue
    if sample in qin_samples and qin_sample_gender_dict[sample] != 'female':
        continue
    if sample in infant_samples:
        continue

    alpha_div = alpha_div_dict[sample]

    if sample in hmp_samples:
        preg_status = 0
    elif sample in mother_samples:
        preg_status = 1
    else:
        continue

    cohort = sample_cohort_map[sample]
    output_file.write(','.join([str(v) for v in [sample, subject, alpha_div, preg_status,
cohort]]) + '\n')

for sample in qin_alpha_div_dict:
    subject = qin_sample_subject_map[sample]
    alpha_div = qin_alpha_div_dict[sample]
    preg_status = 0
    cohort = 'qin'
    output_file.write(','.join([str(v) for v in [sample, subject, alpha_div, preg_status,
cohort]]) + '\n')

output_file.close()


# In[11]:


alpha_divs = [] # list of sample values for each tp
labels = []
sample_sizes = []

for tp in infant_tps_ordered:
    # num_samples = len(mi_tp_sample_dict['infant'][tp])
    alpha_divs_tp = []
    subjects_tp = set()
    for sample in mi_tp_sample_dict['infant'][tp]:
        subject = sample_subject_map[sample]
        if subject in subjects_tp:
            continue
```

```
        alpha_divs_tp.append(alpha_div_dict[sample])
        subjects_tp.add(subject)
    num_samples = len(subjects_tp)
    if num_samples < 10:
        continue # Skip timepoints with not enough data
    labels.append(tp + "\n" + ("n=%i" % num_samples))
    alpha_divs.append(alpha_divs_tp)
    sample_sizes.append(num_samples)

alpha_divs_mother_combined = []
for i in range(len(mother_tps_ordered)):
    tp = mother_tps_ordered[i]
    num_samples = len(mi_tp_sample_dict['mother'][tp])
    alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['mother'][tp]]
    if tp == -92 or tp == 92: # Skip 3month/-3month
        continue # Because of this, only one sample per subject
    alpha_divs_mother_combined += alpha_divs_tp

alpha_divs.append(alpha_divs_mother_combined)
labels.append('Mother'+ "\n" + ("n=%i" % len(alpha_divs_mother_combined)))
sample_sizes.append(len(alpha_divs_mother_combined))

alpha_divs_hmp = []; subjects_tp = set()
for sample in hmp_samples:
    subject = sample_subject_map[sample]
    if subject in subjects_tp:
        continue
    alpha_divs_hmp.append(alpha_div_dict[sample])
    subjects_tp.add(subject)
alpha_divs.append(alpha_divs_hmp)
labels.append('Adult\n(HMP)' + "\n" + ("n=%i" % len(alpha_divs_hmp)))
sample_sizes.append(len(alpha_divs_hmp))

# Note that Qin has only one timepoint per subject
alpha_divs_qin = list(qin_alpha_div_dict.values())
alpha_divs.append(alpha_divs_qin)
labels.append('Adult\n(Qin)' + "\n" + ("n=%i" % len(alpha_divs_qin)))
sample_sizes.append(len(alpha_divs_qin))

fig, ax = plt.subplots(figsize=(15, 4))

boxprops = dict(color='black')
medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(alpha_divs, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

boxplots['boxes'][-2].set_facecolor('#396651')
boxplots['boxes'][-1].set_facecolor('#396651')
boxplots['boxes'][-3].set_facecolor('#7bb551')

ax.set_ylabel("Shannon alpha diversity\nper sample")
ax.set_title("Alpha diversity by timepoint (infants exclude Olm)")
ax.axvline(19.5, color='gray', linestyle='-')
ax.set_xticklabels(labels)

plt.show()
```

```
# In[14]:


import scipy.stats

# Get statistics for paper
alpha_divs_birth = alpha_divs[0]
print("Median alpha diversity, birth: %.02f" % np.median(alpha_divs_birth))
alpha_divs_1day = alpha_divs[1]
print("Median alpha diversity, 1 day: %.02f" % np.median(alpha_divs_1day))
alpha_divs_3day = alpha_divs[2]
print("Median alpha diversity, 3 days: %.02f" % np.median(alpha_divs_3day))
alpha_divs_mother = alpha_divs[-2]
print("Median alpha diversity, mother: %.02f" % np.median(alpha_divs_mother))
alpha_divs_adult = alpha_divs[-1]
print("Median alpha diversity, adult: %.02f" % np.median(alpha_divs_adult))

t, p = scipy.stats.ttest_ind(alpha_divs_birth, alpha_divs_1day)
print("T-test p-value for alpha div increase, birth->1 day: %f" % p)
print("Cohen's D: %s" % str(cohenD(alpha_divs_birth, alpha_divs_1day)))
t, p = scipy.stats.ttest_ind(alpha_divs_1day, alpha_divs_3day)
print("T-test p-value for alpha div decrease, 1 day->3 days: %f" % p)
print("Cohen's D: %s" % str(cohenD(alpha_divs_1day, alpha_divs_3day)))
t, p = scipy.stats.ttest_ind(alpha_divs_mother, alpha_divs_adult)
print("T-test p-value for alpha div mother vs. adult: %s" % str(p))
print("Cohen's D: %s" % str(cohenD(alpha_divs_mother, alpha_divs_adult)))

print("Sample sizes range from %i to %i with median of %.02f." % (min(sample_sizes),
max(sample_sizes), np.median(sample_sizes)))


# In[12]:


# Mom from each cohort, HMP1-2, Poyet, Korpela

alpha_divs_mother_by_cohort = defaultdict(list)

sample_cohort_map = su.parse_sample_cohort_map()
mother_samples = su.get_sample_names('mother')
hmp_samples = su.get_sample_names('hmp')
mother_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

for sample in mother_samples:
    cohort = sample_cohort_map[sample]
    alpha_div = alpha_div_dict[sample]
    alpha_divs_mother_by_cohort[cohort].append(alpha_div)

alpha_divs = []
labels = []
for cohort in mother_cohorts:
    alpha_divs.append(alpha_divs_mother_by_cohort[cohort])
    labels.append('Mother\n%s\nn=%i' % (cohort, len(alpha_divs_mother_by_cohort[cohort])))

hmp_alpha_divs = [alpha_div_dict[sample] for sample in hmp_samples]
alpha_divs.append(hmp_alpha_divs)
labels.append('Adult\nHMP1-2\nn=%i' % (len(hmp_alpha_divs)))
alpha_divs.append(poyet_alpha_div_dict.values())
labels.append('Adult\nPoyet\nn=%i' % (len(poyet_alpha_div_dict)))
alpha_divs.append(korpela_alpha_div_dict.values())
```

```
labels.append('Adult\nKorpela\nn=%i' % (len(korpela_alpha_div_dict)))
alpha_divs.append(qin_alpha_div_dict.values())
labels.append('Adult\nQin\nn=%i' % (len(qin_alpha_div_dict)))

fig, ax = plt.subplots(figsize=(8, 4))

medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(alpha_divs, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops)
for patch in boxplots['boxes'][:4]:
    patch.set_facecolor('#7bb551')
for patch in boxplots['boxes'][-4:]:
    patch.set_facecolor('#396651')

ax.set_ylabel("Shannon alpha diversity")
ax.set_title("Alpha diversity of adult samples by cohort")
ax.set_xticklabels(labels)

plt.show()
fig.savefig('%s/alpha_div_adults_by_cohort.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[32]:


# Mom from each cohort, HMP1-2, Qin, pick one sample per host

alpha_divs_mother_by_subject = {}

for sample in mother_samples:
    subject = sample_subject_map[sample]
    alpha_div = alpha_div_dict[sample]
    alpha_divs_mother_by_subject[subject] = (alpha_div, sample)

alpha_divs_mother_by_cohort = defaultdict(list)

sample_cohort_map = su.parse_sample_cohort_map()
mother_samples = su.get_sample_names('mother')
hmp_samples = su.get_sample_names('hmp')
mother_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

for subject in alpha_divs_mother_by_subject:
    alpha_div, rep_sample = alpha_divs_mother_by_subject[subject]
    cohort = sample_cohort_map[rep_sample]
    alpha_divs_mother_by_cohort[cohort].append(alpha_div)

alpha_divs = []
labels = []
for cohort in mother_cohorts:
    alpha_divs.append(alpha_divs_mother_by_cohort[cohort])
    labels.append('Mother\n%s\nn=%i' % (cohort.capitalize(),
len(alpha_divs_mother_by_cohort[cohort])))

alpha_divs_hmp_by_subject = {}
for sample in hmp_samples:
    subject = sample_subject_map[sample]
    alpha_div = alpha_div_dict[sample]
    alpha_divs_hmp_by_subject[subject] = alpha_div
```

```
alpha_divs.append(list(alpha_divs_hmp_by_subject.values()))
labels.append('Adult\nHMP\nn=%i' % (len(alpha_divs_hmp_by_subject)))

alpha_divs_qin_by_subject = {}
for sample in qin_alpha_div_dict:
    subject = qin_sample_subject_map[sample]
    alpha_div = qin_alpha_div_dict[sample]
    alpha_divs_qin_by_subject[subject] = alpha_div

alpha_divs.append(list(alpha_divs_qin_by_subject.values()))
labels.append('Adult\nQin\nn=%i' % (len(alpha_divs_qin_by_subject)))

fig, ax = plt.subplots(figsize=(8, 6))

medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(alpha_divs, patch_artist=True,
medianprops=medianprops,flierprops=flierprops)
for patch in boxplots['boxes'][:4]:
    patch.set_facecolor('#7bb551')
for patch in boxplots['boxes'][-2:]:
    patch.set_facecolor('#396651')

# Perform pairwise t-test
j = 0
for i in [0, 1, 2, 3]:
    a1 = alpha_divs[i]
    a2 = alpha_divs[4] # Compare to HMP first
    t, p = scipy.stats.ttest_ind(a1, a2)
    color = 'red' if p < 0.05 else 'black'
    plot_interval(5.3 - j, i+1, 5, color=color, tickh=0.05)
    plt.text(0.8+i+((4-i)/2.0), 5.3 - j+0.02, '{:.2e}'.format(p), color=color)
    j += 0.3

j = 0
for i in [0, 1, 2, 3]:
    a1 = alpha_divs[i]
    a2 = alpha_divs[5] # Compare to Qin
    t, p = scipy.stats.ttest_ind(a1, a2)
    print(p)
    color = 'red' if p < 0.05 else 'black'
    plot_interval(-0.5+j, i+1, 6, color=color, tickh=0.05)
    plt.text(0.8+i+((4-i)/2.0), -0.5+j+0.02, '{:.2e}'.format(p), color=color)
    j += 0.3

ax.set_ylabel("Shannon alpha diversity\nper host")
ax.set_title("Alpha diversity of adults by cohort")
ax.set_xticklabels(labels)

plt.show()
fig.savefig('%s/alpha_div_adults_by_cohort.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[14]:


# Save labels for combined plot
alpha_div_adults_by_cohort_labels = labels
```

```
# In[15]:


# Mom from each cohort, HMP1-2, Poyet, Korpela

richnesses_mother_by_cohort = defaultdict(list)

sample_cohort_map = su.parse_sample_cohort_map()
mother_samples = su.get_sample_names('mother')
hmp_samples = su.get_sample_names('hmp')
mother_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

for sample in mother_samples:
    cohort = sample_cohort_map[sample]
    richness = richness_dict[sample]
    richnesses_mother_by_cohort[cohort].append(richness)

richnesses = []
labels = []
for cohort in mother_cohorts:
    richnesses.append(richnesses_mother_by_cohort[cohort])
    labels.append('Mother\n%s\nn=%i' % (cohort, len(richnesses_mother_by_cohort[cohort])))

hmp_richnesses = [richness_dict[sample] for sample in hmp_samples]
richnesses.append(hmp_richnesses)
labels.append('Adult\nHMP1-2\nn=%i' % (len(hmp_richnesses)))
richnesses.append(poyet_richness_dict.values())
labels.append('Adult\nPoyet\nn=%i' % (len(poyet_richness_dict)))
richnesses.append(korpela_richness_dict.values())
labels.append('Adult\nKorpela\nn=%i' % (len(korpela_richness_dict)))
richnesses.append(qin_richness_dict.values())
labels.append('Adult\nQin\nn=%i' % (len(qin_richness_dict)))

fig, ax = plt.subplots(figsize=(8, 4))

medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(richnesses, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops)
for patch in boxplots['boxes'][:4]:
    patch.set_facecolor('#7bb551')
for patch in boxplots['boxes'][-4:]:
    patch.set_facecolor('#396651')

ax.set_ylabel("Species richness")
ax.set_title("Species richness of adult samples by cohort")
ax.set_xticklabels(labels)

plt.show()
fig.savefig('%s/richness_adults_by_cohort.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[16]:


# Mom from each cohort, HMP1-2, Qin, pick one sample per host
```

```
richnesses_mother_by_subject = {}

for sample in mother_samples:
    subject = sample_subject_map[sample]
    richness = richness_dict[sample]
    richnesses_mother_by_subject[subject] = (richness, sample)

richnesses_mother_by_cohort = defaultdict(list)

sample_cohort_map = su.parse_sample_cohort_map()
mother_samples = su.get_sample_names('mother')
hmp_samples = su.get_sample_names('hmp')
mother_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

for subject in richnesses_mother_by_subject:
    richness, rep_sample = richnesses_mother_by_subject[subject]
    cohort = sample_cohort_map[rep_sample]
    richnesses_mother_by_cohort[cohort].append(richness)

richnesses = []
labels = []
for cohort in mother_cohorts:
    richnesses.append(richnesses_mother_by_cohort[cohort])
    labels.append('Mother\n%s\nn=%i' % (cohort.capitalize(),
len(richnesses_mother_by_cohort[cohort])))

richnesses_hmp_by_subject = {}
for sample in hmp_samples:
    subject = sample_subject_map[sample]
    richness = richness_dict[sample]
    richnesses_hmp_by_subject[subject] = richness

richnesses.append(richnesses_hmp_by_subject.values())
labels.append('Adult\nHMP\nn=%i' % (len(richnesses_hmp_by_subject)))

richnesses_qin_by_subject = {}
for sample in qin_richness_dict:
    subject = qin_sample_subject_map[sample]
    richness = qin_richness_dict[sample]
    richnesses_qin_by_subject[subject] = richness

richnesses.append(richnesses_qin_by_subject.values())
labels.append('Adult\nQin\nn=%i' % (len(richnesses_qin_by_subject)))

fig, ax = plt.subplots(figsize=(8, 6))

medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(richnesses, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops)
for patch in boxplots['boxes'][:4]:
    patch.set_facecolor('#7bb551')
for patch in boxplots['boxes'][-2:]:
    patch.set_facecolor('#396651')

# Perform pairwise t-test
j = 0
for i in [0, 1, 2, 3]:
    a1 = richnesses[i]
    a2 = richnesses[4] # Compare to HMP first
```

```
    t, p = scipy.stats.ttest_ind(a1, a2)
    color = 'red' if p < 0.05 else 'black'
    plot_interval(400 - (60*j), i+1, 5, color=color, tickh=5)
    plt.text(0.8+i+((4-i)/2.0), 400 - (60*j) + 1, '{:.2e}'.format(p), color=color)
    j += 0.3

j = 0
for i in [0, 1, 2, 3]:
    a1 = richnesses[i]
    a2 = richnesses[5] # Compare to Qin
    t, p = scipy.stats.ttest_ind(a1, a2)
    print(p)
    color = 'red' if p < 0.05 else 'black'
    plot_interval(-60+(60*j), i+1, 6, color=color, tickh=5)
    plt.text(0.8+i+((4-i)/2.0), -60 + (60*j)+1, '{:.2e}'.format(p), color=color)
    j += 0.3

ax.set_ylabel("Species richness\nper host")
ax.set_title("Species richness of adults by cohort")
ax.set_xticklabels(labels)

plt.show()
fig.savefig('%s/richness_adults_by_cohort.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[17]:


# Save labels for combined plot
richness_adults_by_cohort_labels = labels


# In[18]:


# Statistical significance to asterisk representation mapping
def get_sig_str(pval):
    if pval <= 0.001:
        return '***'
    elif pval <= 0.01:
        return '**'
    elif pval <= 0.05:
        return '*'
    elif pval > 0.05:
        return 'ns'


# In[18]:


'''
ns: P > 0.05

*: P ≤ 0.05

**: P ≤ 0.01

***: P ≤ 0.001

****: P ≤ 0.0001
```

```
'''


# In[19]:


# Combine alpha diversity and richness plots for
# Mom from each cohort, HMP1-2, Qin, pick one sample per host

# Assume individual plots' code has been run

fig, ax = plt.subplots(1, 2, figsize=(18, 6))
medianprops = dict(color='black')
flierprops = dict(marker='.')

# ============================================================
# Alpha diversity
# ============================================================
boxplots = ax[0].boxplot(alpha_divs, patch_artist=True,
                         medianprops=medianprops,
                         flierprops=flierprops)
for patch in boxplots['boxes'][:4]:
    patch.set_facecolor('#7bb551')
for patch in boxplots['boxes'][-2:]:
    patch.set_facecolor('#396651')

# Perform pairwise t-test
j = 0
for i in [0, 1, 2, 3]:
    a1 = alpha_divs[i]
    a2 = alpha_divs[4] # Compare to HMP first
    t, p = scipy.stats.ttest_ind(a1, a2)
    color = 'red' if t < 0 else 'blue'
    if p > 0.05:
        color = 'gray'
    plot_interval_on_ax(ax[0], 5.3 - (0.8*j), i+1, 5, color=color, tickh=0.05)
    ax[0].text(1+i+((4-i)/2.0), 5.3 - (0.8*j) +0.01, get_sig_str(p), color=color)
    # ax[0].text(0.8+i+((4-i)/2.0), 5.3 - j+0.02, '{:.2e}'.format(p), color=color)
    j += 0.3

j = 0
for i in [0, 1, 2, 3]:
    a1 = alpha_divs[i]
    a2 = alpha_divs[5] # Compare to Qin
    t, p = scipy.stats.ttest_ind(a1, a2)
    print(p)
    color = 'red' if t < 0 else 'blue'
    if p > 0.05:
        color = 'gray'
    plot_interval_on_ax(ax[0], -0.5 + (0.8*j), i+1, 6, color=color, tickh=0.05)
    ax[0].text(1+i+((5-i)/2.0), -0.5 + (0.8*j) + 0.01, get_sig_str(p), color=color)
    # ax[0].text(0.8+i+((4-i)/2.0), -0.5+j+0.02, '{:.2e}'.format(p), color=color)
    j += 0.3

ax[0].set_ylabel("Shannon alpha diversity per host")
ax[0].set_title("Alpha diversity of adults by cohort")
ax[0].set_xticklabels(alpha_div_adults_by_cohort_labels)
ax[0].text(-0.08, 1, 'A', size=20, transform=ax[0].transAxes, weight='bold')


# ============================================================
```

```
# Richness
# ========================================================
boxplots = ax[1].boxplot(richnesses, patch_artist=True,
                      medianprops=medianprops,
                      flierprops=flierprops)
for patch in boxplots['boxes'][:4]:
    patch.set_facecolor('#7bb551')
for patch in boxplots['boxes'][-2:]:
    patch.set_facecolor('#396651')

# Perform pairwise t-test
j = 0
for i in [0, 1, 2, 3]:
    a1 = richnesses[i]
    a2 = richnesses[4] # Compare to HMP first
    t, p = scipy.stats.ttest_ind(a1, a2)
    color = 'red' if t < 0 else 'blue'
    if p > 0.05:
        color = 'gray'
    plot_interval_on_ax(ax[1], 400 - (60*j), i+1, 5, color=color, tickh=5)
    ax[1].text(1+i+((4-i)/2.0), 400 - (60*j) + 1, get_sig_str(p), color=color)
    # ax[1].text(0.8+i+((4-i)/2.0), 400 - (60*j) + 1, '{:.2e}'.format(p), color=color)
    j += 0.3

j = 0
for i in [0, 1, 2, 3]:
    a1 = richnesses[i]
    a2 = richnesses[5] # Compare to Qin
    t, p = scipy.stats.ttest_ind(a1, a2)
    print(p)
    color = 'red' if t < 0 else 'blue' # red means negative t
    if p > 0.05:
        color = 'gray'
    plot_interval_on_ax(ax[1], -60+(60*j), i+1, 6, color=color, tickh=5)
    ax[1].text(1+i+((5-i)/2.0), -60 + (60*j)+1, get_sig_str(p), color=color)
    # ax[1].text(0.8+i+((4-i)/2.0), -60 + (60*j)+1, '{:.2e}'.format(p), color=color)
    j += 0.3

ax[1].set_ylabel("Species richness per host")
ax[1].set_title("Species richness of adults by cohort")
ax[1].set_xticklabels(richness_adults_by_cohort_labels)
ax[1].text(-0.08, 1, 'B', size=20, transform=ax[1].transAxes, weight='bold')

plt.show()
plt.subplots_adjust(wspace=0)
fig.savefig('%s/alpha_div_and_richness_adults_by_cohort.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[25]:


# Why trends in alpha diversity and richness different?
# Ideas: (1) Plot aplpha diversity vs. richness
# (2) Look at individual samples


# In[52]:


alpha_divs_mother = []
```

```
richnesses_mother = []
samples_mother = []

for tp in mother_tps_ordered:
    if tp == -92 or tp == 92: # Skip 3month/-3month
        continue

    samples_tp = list(mi_tp_sample_dict['mother'][tp])
    alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['mother'][tp]]
    richnesses_tp = [richness_dict[sample] for sample in mi_tp_sample_dict['mother'][tp]]

    samples_mother += samples_tp
    alpha_divs_mother += alpha_divs_tp
    richnesses_mother += richnesses_tp

samples_hmp = hmp_samples # kek
alpha_divs_hmp = [alpha_div_dict[sample] for sample in samples_hmp]
richnesses_hmp = [richness_dict[sample] for sample in samples_hmp]

samples_qin = list(qin_sample_subject_map.keys())
alpha_divs_qin = [qin_alpha_div_dict[sample] for sample in samples_qin]
richnesses_qin = [qin_richness_dict[sample] for sample in samples_qin]

fig, ax = plt.subplots()

ax.plot(alpha_divs_mother, richnesses_mother, 'r.', label="Mother (delivery)")
ax.plot(alpha_divs_hmp, richnesses_hmp, 'g.', label="HMP Adult")
ax.plot(alpha_divs_qin, richnesses_qin, 'b.', label="Qin Adult")
ax.plot(np.median(alpha_divs_mother), np.median(richnesses_mother), '.', color='orange',
markersize=16)
ax.plot(np.median(alpha_divs_hmp), np.median(richnesses_hmp), 'y.', markersize=16)
ax.plot(np.median(alpha_divs_qin), np.median(richnesses_qin), '.', color='cyan', markersize=16)
ax.set_ylabel("Richness")
ax.set_xlabel("Shannon alpha diversity")
ax.legend()

# So yes, mother samples do seem to have higher median alpha diversity but comparable median
richness

plt.show()


# In[25]:


# Pick out example
mother_examples = []
for alpha_div, richness, sample in zip(alpha_divs_mother, richnesses_mother, samples_mother):
    if alpha_div > 3.5 and richness < 120:
        mother_examples.append(sample)

hmp_examples = []
for alpha_div, richness, sample in zip(alpha_divs_hmp, richnesses_hmp, samples_hmp):
    if alpha_div < 2.5 and richness == 117:
        hmp_examples.append(sample)


# In[57]:


# Overlap in species when richness is the same?
```

```
for adult_sample in hmp_examples:
    mother_species = set(relab_dict['ERR3405313'].keys())
    adult_species = set(relab_dict[adult_sample].keys())
    intersection = mother_species.intersection(adult_species)
    print("%i species are shared out of %i" % (len(intersection), len(mother_species)))


# In[48]:


fig, ax = plt.subplots()

labels = []

i = 0
for sample in ['700015857', '700164870', '700105312', 'ERR3405313']:
    relab_vals = []
    for species, relab in sorted(relab_dict[sample].items(), key=lambda x: x[1], reverse=True):
        relab_vals.append(relab)

    acc = 0
    for relab_val in relab_vals:
        ax.bar(i, relab_val, bottom=acc)
        acc += relab_val

    htype = 'Mother' if sample in mother_samples else 'Adult'
    label = "%s\na=%.02f\nr=%i" % (htype, alpha_div_dict[sample], richness_dict[sample])
    labels.append(label)
    i += 1

ax.set_title("Sample breakdown by species relative abundance\n(Note that colors are meaningless
here)")
ax.set_xticks(np.arange(i))
ax.set_xticklabels(labels)

plt.show()


# In[94]:


# Mom from each cohort, HMP1-2, Qin, pick one sample per host

richnesses_mother_by_subject = {}

for sample in mother_samples:
    subject = sample_subject_map[sample]
    richness = richness_dict[sample]
    richnesses_mother_by_subject[subject] = richness

richnesses = []
labels = []

richnesses.append(richnesses_mother_by_subject.values())
labels.append('Mother\nn=%i' % (len(richnesses_mother_by_subject)))

richnesses_hmp_by_subject = {}
for sample in hmp_samples:
    subject = sample_subject_map[sample]
    richness = richness_dict[sample]
    richnesses_hmp_by_subject[subject] = richness
```

```
richnesses_qin_by_subject = {}
for sample in qin_richness_dict:
    subject = qin_sample_subject_map[sample]
    richness = qin_richness_dict[sample]
    richnesses_qin_by_subject[subject] = richness

richnesses_adults = richnesses_hmp_by_subject.values() + richnesses_qin_by_subject.values()
richnesses.append(richnesses_adults)
labels.append('Adult\nn=%i' % (len(richnesses_adults)))

fig, ax = plt.subplots(figsize=(4, 4))

medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(richnesses, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops)
boxplots['boxes'][0].set_facecolor('#7bb551')
boxplots['boxes'][1].set_facecolor('#396651')

ax.set_ylabel("Species richness per host")
ax.set_title("Species richness of adults by cohort")
ax.set_xticklabels(labels)

plt.show()
fig.savefig('%s/richness_mothers_vs_adults.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[95]:


scipy.stats.ttest_ind(richnesses[0], richnesses[1])


# In[33]:


fig, ax = plt.subplots(2, 1, figsize=(12,6))

species_list = good_species_list[0:2]

all_tps = tps_ordered_dict['infant'] + ['Mother', 'Adult']

for i in range(len(species_list)):
    desired_species = species_list[i]
    data = []
    labels = []

    polymorphism_by_tp_dict = defaultdict(list)

    for sample in mother_samples:
        for species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][species]
            if species == desired_species:
                polymorphism_by_tp_dict['Mother'].append(polymorphism)

    for sample in hmp_samples:
        for species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][species]
```

```
            if species == desired_species:
                polymorphism_by_tp_dict['Adult'].append(polymorphism)

    for tp in mi_tp_sample_dict['infant']:
        for sample in mi_tp_sample_dict['infant'][tp]:
            for species in sample_species_polymorphism_dict[sample]:
                polymorphism = sample_species_polymorphism_dict[sample][species]
                if species == desired_species:
                    polymorphism_by_tp_dict[tp].append(polymorphism)

    for tp in all_tps:
        polymorphisms = polymorphism_by_tp_dict[tp]
        data.append(polymorphisms)
        labels.append("%s\nn=%i" % (tp, len(polymorphisms)))

    ax[i].set_yscale('log')
    boxprops = dict(color='#77acff')
    medianprops = dict(color='black')
    flierprops = dict(marker='.')
    boxplots = ax[i].boxplot(data, patch_artist=True,
                             boxprops=boxprops,
                             medianprops=medianprops,
                             flierprops=flierprops)
    for patch in boxplots['boxes'][:-2]:
        patch.set_facecolor('#77acff')
    boxplots['boxes'][-1].set_facecolor('#396651')
    boxplots['boxes'][-2].set_facecolor('#7bb551')
    ax[i].set_title('%s' % desired_species)
    ax[i].axvline(x=20.5, color='gray')
    ax[i].set_ylabel("pS")
    ax[i].set_xticklabels(labels)

plt.tight_layout()
plt.show()


# In[51]:


# Split E. coli polymorphism by dataset

fig, ax = plt.subplots(1, 4, figsize=(18,4), sharey=True, constrained_layout=True,
                       gridspec_kw={'width_ratios': [2, 3, 3, 3]})
desired_species = 'Escherichia_coli_58110'

# HMP1-2
hmp_samples = su.get_sample_names('HMP')
order_polymorphism_dict = defaultdict(list)
for sample in hmp_samples:
    subject, order = sample_order_map[sample]
    if desired_species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][desired_species]
        order_polymorphism_dict[order].append(polymorphism)

order_list = [1, 2, 3]
order_labels = ['1', '2', '3']
plot_data = [order_polymorphism_dict[o] for o in order_list]

ax[0].set_yscale('log')
ax[0].boxplot(plot_data)
ax[0].set_xticklabels(order_labels)
```

```
ax[0].set_ylabel("Polymorphism")
ax[0].set_title("HMP1-2")

# Backhed
backhed_samples = su.get_sample_names('Backhed')
order_polymorphism_dict = defaultdict(list)
for sample in backhed_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples)
    if desired_species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][desired_species]
        order_polymorphism_dict[tp].append(polymorphism)

order_list = ['M1', 'I1', 'I2', 'I3']
order_labels = ['Mother', 'Birth (2-5 days)', '4 Months', '12 Months']
plot_data = [order_polymorphism_dict[o] for o in order_list]

ax[1].set_yscale('log')
ax[1].boxplot(plot_data)
ax[1].set_xticklabels(order_labels)
ax[1].set_title("Backhed")

# Ferretti
ferretti_samples = su.get_sample_names('Ferretti')
order_polymorphism_dict = defaultdict(list)
for sample in ferretti_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples)
    if desired_species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][desired_species]
        order_polymorphism_dict[tp].append(polymorphism)

order_list = ['M1', 'I1', 'I2', 'I3', 'I4', 'I5']
order_labels = ['Mother', '1 day', '3 days', '1 wk', '1 mon', '4 mon']
plot_data = [order_polymorphism_dict[o] for o in order_list]

ax[2].set_yscale('log')
ax[2].boxplot(plot_data)
ax[2].set_xticklabels(order_labels)
ax[2].set_title("Ferretti")

# Yassour
yassour_samples = su.get_sample_names('Yassour')
order_polymorphism_dict = defaultdict(list)
for sample in yassour_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples)
    if desired_species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][desired_species]
        order_polymorphism_dict[tp].append(polymorphism)

order_list = ['M1', 'M2', 'M3', 'I1', 'I2', 'I3', 'I4', 'I5']
order_labels = ['Mother:\nGest', 'Mother:\nDelivery','Mother:\n3 mon', 'Birth', '1 wk', '2 wk',
'1 mon', '2 mon', '3 mon']
plot_data = [order_polymorphism_dict[o] for o in order_list]

ax[3].set_yscale('log')
ax[3].boxplot(plot_data)
ax[3].set_xticklabels(order_labels)
ax[3].set_title("Yassour")

plt.show()
```

```
# In[48]:


fig, ax = plt.subplots(figsize=(12, 3.6), sharey=True, constrained_layout=True)

# Shao
month_bins = np.arange(4, 15) * 30.5
order_bins = [0, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 21] + list(month_bins)

shao_samples = su.get_sample_names('Shao')
order_polymorphism_dict = defaultdict(list)
for sample in shao_samples:
    subject, order = sample_order_map[sample]
    if desired_species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][desired_species]
    else:
        continue
    if subject[-1] == 'M': # Mother
        order_polymorphism_dict[0].append(polymorphism)
    elif order > 21: # Convert to month bins
        month_bin = round_down(order, 30.5)
        order_polymorphism_dict[month_bin].append(polymorphism)
    else:
        order_polymorphism_dict[order].append(polymorphism)

plot_data = [order_polymorphism_dict[o] for o in order_bins]
order_labels = ['Mother', '4d', '6d', '7d', '8d', '9d', '10d', '11d', '12d', '13d', '14d',
'17d', '18d', '21d', '4m', '5m', '6m', '7m', '8m', '9m', '10m', '11m', '12m', '13m', '14m']

ax.set_yscale('log')
ax.boxplot(plot_data)
ax.set_xticklabels(order_labels)
ax.set_ylabel("Polymorphism")
ax.set_title("Shao")


# In[22]:


good_species_list[0:30]


# In[15]:


# Find most prevalent species among infants

species_nonzero_count = defaultdict(int)

for sample in infant_samples:
    for species in relab_dict[sample]:
        if relab_dict[sample][species] > 0:
            species_nonzero_count[species] += 1

species_infant_prev_ordered = []
for species, count in sorted(species_nonzero_count.items(), key=lambda x: x[1], reverse=True):
    species_infant_prev_ordered.append(species)


# In[16]:
```

```
species_infant_prev_ordered


# In[25]:


fig, ax = plt.subplots(6, 2, figsize=(25,16.5))# , sharex=True)

species_list = good_species_list[1:13] # species_infant_prev_ordered[:3] +
species_infant_prev_ordered[4:13]

infant_tps = copy.copy(tps_ordered_dict['infant'])
infant_tps.remove('birth')
infant_tps.remove('5 day')
infant_tps.remove('6 day')
all_tps = infant_tps + ['Mother', 'HMP'] # Skip birth (combine with 1 day)
all_tps_abbreviated = [(label.split(' ')[0] + label.split(' ')[1][0].lower()) for label in
infant_tps] + ['M', 'HMP']

i = 0
j = 0
for desired_species in species_list:

    data = []
    labels = []

    polymorphism_by_tp_dict = defaultdict(list)

    for sample in mother_samples:
        for species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][species]
            if species == desired_species:
                polymorphism_by_tp_dict['Mother'].append(polymorphism)

    for sample in hmp_samples:
        for species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][species]
            if species == desired_species:
                polymorphism_by_tp_dict['HMP'].append(polymorphism)

    for tp in mi_tp_sample_dict['infant']:
        for sample in mi_tp_sample_dict['infant'][tp]:
            for species in sample_species_polymorphism_dict[sample]:
                polymorphism = sample_species_polymorphism_dict[sample][species]
                if species == desired_species:
                    polymorphism_by_tp_dict[tp].append(polymorphism)

    for tp in all_tps:
        polymorphisms = polymorphism_by_tp_dict[tp]
        if tp == '1 day':
            polymorphisms += polymorphism_by_tp_dict['birth']
        if len(polymorphisms) < 3: # Don't show if < 3 samples
            data.append([])
        else:
            data.append(polymorphisms)
        labels.append("n=%i" % (len(polymorphisms)))

    ax[i][j].set_yscale('log')
    boxprops = dict(color='#77acff')
    medianprops = dict(color='black')
```

```
    flierprops = dict(marker='.')
    boxplots = ax[i][j].boxplot(data, patch_artist=True,
                                medianprops=medianprops,
                                flierprops=flierprops)
    for patch in boxplots['boxes'][:-2]:
        patch.set_facecolor('#77acff')
    boxplots['boxes'][-1].set_facecolor('#396651')
    boxplots['boxes'][-2].set_facecolor('#7bb551')
    ax[i][j].set_title('%s' % plot_utils.get_pretty_species_name(desired_species), fontsize=14)
    ax[i][j].axvline(x=19.5, color='gray')
    ax[i][j].set_ylabel("pS", fontsize=14)

    if i == 5:
        ax[i][j].set_xticklabels(["%s\n%s" % (tp, label) for tp, label in
zip(all_tps_abbreviated, labels)], fontsize=12)
    else:
        ax[i][j].set_xticklabels(labels, fontsize=12)

    if i < 6:
        i += 1

    if i == 6:
        i = 0
        j = 1

plt.tight_layout()
plt.show()

fig.savefig('%s/polymorphism_top_12_prevalence_species.pdf' % (config.analysis_directory),
bbox_inches='tight')
fig.savefig('%s/polymorphism_top_12_prevalence_species.png' % (config.analysis_directory),
bbox_inches='tight', dpi=250)


# In[12]:


# Final: literally everything

fig, ax = plt.subplots(30, 2, figsize=(19,60), sharex=True)

species_list = good_species_list[60:120]

all_tps = tps_ordered_dict['infant'] + ['Mother', 'HMP']
all_tps_abbreviated = ['B'] + [(label.split(' ')[1][0].upper() + label.split(' ')[0]) for label
in tps_ordered_dict['infant'][1:]] + ['M', 'HMP']

i = 0
j = 0
for desired_species in species_list:

    data = []
    labels = []

    polymorphism_by_tp_dict = defaultdict(list)

    for sample in mother_samples:
        for species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][species]
            if species == desired_species:
                polymorphism_by_tp_dict['Mother'].append(polymorphism)
```

```
    for sample in hmp_samples:
        for species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][species]
            if species == desired_species:
                polymorphism_by_tp_dict['HMP'].append(polymorphism)

    for tp in mi_tp_sample_dict['infant']:
        for sample in mi_tp_sample_dict['infant'][tp]:
            for species in sample_species_polymorphism_dict[sample]:
                polymorphism = sample_species_polymorphism_dict[sample][species]
                if species == desired_species:
                    polymorphism_by_tp_dict[tp].append(polymorphism)

    for tp in all_tps:
        polymorphisms = polymorphism_by_tp_dict[tp]
        if len(polymorphisms) < 3: # Don't show if < 3 samples
            data.append([])
        else:
            data.append(polymorphisms)
        labels.append("%s\nn=%i" % (tp, len(polymorphisms)))

    ax[i][j].set_yscale('log')
    boxprops = dict(color='#77acff')
    medianprops = dict(color='black')
    flierprops = dict(marker='.')
    boxplots = ax[i][j].boxplot(data, patch_artist=True,
                                medianprops=medianprops,
                                flierprops=flierprops)
    for patch in boxplots['boxes'][:-2]:
        patch.set_facecolor('#77acff')
    boxplots['boxes'][-1].set_facecolor('#396651')
    boxplots['boxes'][-2].set_facecolor('#7bb551')
    ax[i][j].set_title('%s' % plot_utils.get_pretty_species_name(desired_species), fontsize=14)
    ax[i][j].axvline(x=20.5, color='gray')
    ax[i][j].set_ylabel("pS", fontsize=14)
    ax[4][j].set_xticklabels(all_tps_abbreviated, fontsize=14)

    if i < 30:
        i += 1

    if i == 30:
        i = 0
        j = 1

plt.tight_layout()
plt.show()

# fig.savefig('%s/polymorphism_top_12_prevalence_species.pdf' % (config.analysis_directory),
bbox_inches='tight')
# fig.savefig('%s/polymorphism_top_12_prevalence_species.png' % (config.analysis_directory),
bbox_inches='tight', dpi=500)


# We excluded some species because they didn’t have enough data. What was the criteria for that?
#
# Also, TODO: make pretty with same colors from Fig 1. Increase the font size for the axes
labels. Remove the redundant labels for the time to de-clutter, but, keep sample size. Remove
boxes with <3 samples. In legend say that each box plot is composed of >n samples
#
# Make pretty the species name (remove the number)
```

```
#
# Change ‘adult’ label to be HMP since technically mothers are adults.
#
# Are the species ordered in some manner?
#

# In[59]:


# Investigate specific species
desired_species = 'Bifidobacterium_breve_57133'
# desired_species = 'Enterococcus_faecalis_56297'
# desired_species = 'Bifidobacterium_bifidum_55065'

for desired_species in ['Bifidobacterium_breve_57133', 'Enterococcus_faecalis_56297',
'Bifidobacterium_bifidum_55065']:
    mother_days = []
    for sample in sample_species_polymorphism_dict:
        if sample not in mother_samples:
                continue

        if desired_species in sample_species_polymorphism_dict[sample]:
            mother_days.append(mi_sample_day_dict[sample])

            subject = sample_subject_map[sample]
            host_mother_samples = subject_sample_map[subject].keys()
            if len(host_mother_samples) > 1:
                print(sample_cohort_map[sample])
                print("Is the species also present postpartum?")
                for sample2 in host_mother_samples:
                    if sample2 not in mi_sample_day_dict: # happens if non-fecal?
                        continue
                    if desired_species in sample_species_polymorphism_dict[sample2]:
                        print("%s present at day %i" % (desired_species,
mi_sample_day_dict[sample2]))
                    else:
                        print("%s absent at day %i" % (desired_species,
mi_sample_day_dict[sample2]))

    print("Days of mother samples (n=%i) having %s in SFS map/polymorphism info:" %
(len(mother_days), desired_species))
    print(mother_days)


# In[47]:


# How does number of mothers having polymorphism info for these species
# compare to number of mothers that have that species present at all?
# Subject to some presence threshold

threshold = 0

for desired_species in ['Bifidobacterium_breve_57133', 'Enterococcus_faecalis_56297',
'Bifidobacterium_bifidum_55065']:
    mother_samples_pres = set()
    mother_subjects_pres = set()
    for sample in mother_samples:
        if desired_species not in relab_dict[sample]:
            relab = 0
        else:
```

```
            relab = relab_dict[sample][desired_species]
        if relab > threshold:
            mother_samples_pres.add(sample)
            mother_subjects_pres.add(sample_subject_map[sample])
    print("%i mother samples / %i subjects have %s present" % (len(mother_samples_pres),
                                                               len(mother_subjects_pres),
desired_species))


# In[10]:


# THIS IS FIGURE 1
# Remove 'Birth" and include Qin (male and female)

fig, ax = plt.subplots(2, 1, figsize=(13,7))# , sharex=True)

species_list = good_species_list[0:2]

infant_tps = [tp for tp in tps_ordered_dict['infant']]
infant_tps.remove('5 day')
infant_tps.remove('6 day')
all_tps = infant_tps + ['Mother', 'Adult']

# Alpha diversity ========================================
alpha_divs = [] # list of sample values for each tp
labels = []

for tp in infant_tps:
    num_samples = len(mi_tp_sample_dict['infant'][tp])
    labels.append(tp + "\n" + ("n=%i" % num_samples))
    alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['infant'][tp]]
    alpha_divs.append(alpha_divs_tp)

alpha_divs_mother_combined = []
for i in range(len(mother_tps_ordered)):
    tp = mother_tps_ordered[i]
    num_samples = len(mi_tp_sample_dict['mother'][tp])
    alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['mother'][tp]]
    alpha_divs_mother_combined += alpha_divs_tp

alpha_divs.append(alpha_divs_mother_combined)
labels.append('Mother'+ "\n" + ("n=%i" % len(alpha_divs_mother_combined)))

alpha_divs_hmp = [alpha_div_dict[sample] for sample in hmp_samples]
alpha_divs_qin = list(qin_alpha_div_dict.values())
alpha_divs.append(alpha_divs_hmp)
labels.append('HMP' + "\n" + ("n=%i" % len(alpha_divs_hmp)))
alpha_divs.append(alpha_divs_qin)
labels.append('Qin' + "\n" + ("n=%i" % len(alpha_divs_qin)))

boxprops = dict(color='#77acff')
medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax[0].boxplot(alpha_divs, patch_artist=True,
                         medianprops=medianprops,
                         flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
```

```
boxplots['boxes'][-1].set_facecolor('#396651')
boxplots['boxes'][-2].set_facecolor('#396651')
boxplots['boxes'][-3].set_facecolor('#7bb551')

ax[0].set_ylabel("Shannon\nalpha diversity", fontsize=14)
# ax[0].set_title("Alpha diversity by timepoint (infants exclude Olm)")
ax[0].axvline(19.5, color='gray', linestyle='-')
ax[0].set_xticklabels(labels)
ax[0].text(-0.08, 0.92, 'A', size=20, transform=ax[0].transAxes, weight='bold')
# ========================================================

# Polymorphism for E. coli ================================
all_tps = infant_tps + ['HMP', 'Qin']
desired_species = 'Escherichia_coli_58110'
data = []
labels = []
sample_sizes = []

polymorphism_by_tp_dict = defaultdict(list)
for sample in mother_samples:
    if desired_species in sample_species_polymorphism_dict[sample]:

polymorphism_by_tp_dict['Mother'].append(sample_species_polymorphism_dict[sample][desired_specie
s])

for sample in hmp_samples:
    if desired_species in sample_species_polymorphism_dict[sample]:

polymorphism_by_tp_dict['HMP'].append(sample_species_polymorphism_dict[sample][desired_species])

for sample in qin_samples:
    if desired_species in sample_species_polymorphism_dict[sample]:

polymorphism_by_tp_dict['Qin'].append(sample_species_polymorphism_dict[sample][desired_species])

for tp in mi_tp_sample_dict['infant']:
    for sample in mi_tp_sample_dict['infant'][tp]:
        if desired_species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][desired_species]
            polymorphism_by_tp_dict[tp].append(polymorphism)

for tp in all_tps:
    polymorphisms = polymorphism_by_tp_dict[tp]
    data.append(polymorphisms)
    labels.append("%s\nn=%i" % (tp, len(polymorphisms)))
    sample_sizes.append(len(polymorphisms))

# Report sample size statistics
print("Sample sizes range from %i to %i with median of %.02f." % (min(sample_sizes),
max(sample_sizes), np.median(sample_sizes)))

ax[1].set_yscale('log')
boxprops = dict(color='#77acff')
medianprops = dict(color='black')
flierprops = dict(marker='.')
boxplots = ax[1].boxplot(data, patch_artist=True,
                         medianprops=medianprops,
                         flierprops=flierprops)
for patch in boxplots['boxes'][:-2]:
    patch.set_facecolor('#77acff')
```

```
boxplots['boxes'][-1].set_facecolor('#396651')
boxplots['boxes'][-2].set_facecolor('#396651')
boxplots['boxes'][-3].set_facecolor('#7bb551')

# ax[1].set_title('$\\it{Escherichia}$ $\\it{coli}$', fontsize=14)# '$\it{%s}$' %
plot_utils.get_pretty_species_name(desired_species), fontsize=14)
ax[1].axvline(x=18.5, color='gray')
ax[1].set_ylabel("Within-sample\npolymorphism", fontsize=14)
ax[1].set_xticklabels(labels)
ax[1].text(-0.08, 0.92, 'B', size=20, transform=ax[1].transAxes, weight='bold')
# ========================================================

plt.tight_layout()
plt.show()
fig.savefig('%s/alpha_div_and_polymorphism_over_time.pdf' % (config.analysis_directory),
bbox_inches='tight', dpi=600)


# In[10]:


# THIS IS FIGURE 1 (new: only polymorphism, also combine birth and one day into 1 day)

fig, ax = plt.subplots(1, figsize=(13,3.4))

species_list = good_species_list[0:2]

infant_tps = [tp for tp in tps_ordered_dict['infant']]
infant_tps.remove('5 day')
infant_tps.remove('6 day')
infant_tps.remove('birth')

# Polymorphism for E. coli ================================
all_tps = infant_tps + ['Mother', 'HMP', 'Qin']
desired_species = 'Escherichia_coli_58110'
data = []
labels = []
sample_sizes = []

polymorphism_by_tp_dict = defaultdict(list)
for sample in mother_samples:
    if desired_species in sample_species_polymorphism_dict[sample]:

polymorphism_by_tp_dict['Mother'].append(sample_species_polymorphism_dict[sample][desired_specie
s])

for sample in hmp_samples:
    if desired_species in sample_species_polymorphism_dict[sample]:

polymorphism_by_tp_dict['HMP'].append(sample_species_polymorphism_dict[sample][desired_species])

for sample in qin_samples:
    if desired_species in sample_species_polymorphism_dict[sample]:

polymorphism_by_tp_dict['Qin'].append(sample_species_polymorphism_dict[sample][desired_species])

for tp in mi_tp_sample_dict['infant']:
    for sample in mi_tp_sample_dict['infant'][tp]:
        if desired_species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][desired_species]
            polymorphism_by_tp_dict[tp].append(polymorphism)
```

```
for tp in all_tps:
    polymorphisms = polymorphism_by_tp_dict[tp]
    if tp == '1 day': # Special case: combine birth with 1 day
        polymorphisms += polymorphism_by_tp_dict['birth']
    data.append(polymorphisms)
    labels.append("%s\nn=%i" % (tp, len(polymorphisms)))
    sample_sizes.append(len(polymorphisms))

# Report sample size statistics
print("Sample sizes range from %i to %i with median of %.02f." % (min(sample_sizes),
max(sample_sizes), np.median(sample_sizes)))

ax.set_yscale('log')
boxprops = dict(color='black'); medianprops = dict(color='black'); flierprops = dict(marker='.')
boxplots = ax.boxplot(data, patch_artist=True, medianprops=medianprops, flierprops=flierprops)
for patch in boxplots['boxes'][:-2]:
    patch.set_facecolor('#77acff')

boxplots['boxes'][-1].set_facecolor('#396651')
boxplots['boxes'][-2].set_facecolor('#396651')
boxplots['boxes'][-3].set_facecolor('#7bb551')

# ax[1].set_title('$\\it{Escherichia}$ $\\it{coli}$', fontsize=14)# '$\it{%s}$' %
plot_utils.get_pretty_species_name(desired_species), fontsize=14)
ax.axvline(x=19.5, color='gray')
ax.set_ylabel("Within-sample\npolymorphism", fontsize=14)
ax.set_xticklabels(labels)
# ========================================================

plt.tight_layout()
plt.show()
fig.savefig('%s/polymorphism_over_time.pdf' % (config.analysis_directory), bbox_inches='tight',
dpi=600)


# In[22]:


'$\it{%s}$' % plot_utils.get_pretty_species_name(desired_species)


# In[10]:


# Get range of all polymorphism
all_polymorphism = []
for sample in sample_species_polymorphism_dict:
    for species in sample_species_polymorphism_dict[sample]:
        all_polymorphism.append(sample_species_polymorphism_dict[sample][species])


# In[15]:


plt.boxplot(all_polymorphism)
plt.yscale('log')
plt.show()


# In[21]:
```

```
np.quantile(all_polymorphism, 0.75)


# In[28]:


# Follow relative abundance of species and its polymorphism



# Relative abundance file
relab_fpath = "%s/species/relative_abundance.txt.bz2" % (config.data_directory)
relab_file = open(relab_fpath, 'r')
decompressor = bz2.BZ2Decompressor()
raw = decompressor.decompress(relab_file.read())
data = [row.split('\t') for row in raw.split('\n')]
data.pop() # Get rid of extra element due to terminal newline
header = su.parse_merged_sample_names(data[0])

# Generate alpha diversity dictionary
relab_dict = {sample: {} for sample in header}
for i in range(1, len(header)):
    sample = header[i]
    for row in data[1:]:
        species = row[0]
        relab_dict[sample][species] = float(row[i])


# In[30]:


for sample in relab_dict:
    if sample not in hmp_samples:
        print(relab_dict[sample]['Bifidobacterium_breve_57133'])
```

plot_figure_2.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


# Rates plot


# In[2]:


import config, parse_midas_data, sample_utils as su, temporal_changes_utils, stats_utils,
midas_db_utils, parse_patric, plot_utils
from collections import defaultdict
import numpy as np
from numpy.random import binomial as sample_binomial
import math, pickle, sys, random
from math import log10,ceil,log,exp
import scipy
```

```
import scipy.stats as stats
import matplotlib.pyplot as plt
import matplotlib.cm as cmx
import matplotlib.ticker
from matplotlib.patches import Patch
from matplotlib import rcParams
from matplotlib import gridspec
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Arial']

# Plot directory
plot_dir = "%s/" % (config.analysis_directory)

# Species list
good_species_list = parse_midas_data.load_pickled_good_species_list()

# Sample-subject-order-cohort maps
sys.stderr.write("Loading sample metadata...\n")
subject_sample_map = su.parse_subject_sample_map()
sample_order_map = su.parse_sample_order_map()
sample_subject_map = su.parse_sample_subject_map()
sample_cohort_map = su.parse_sample_cohort_map()
same_mi_pair_dict = su.get_same_mi_pair_dict(subject_sample_map)
subject_delivery_mode_map = su.parse_subject_delivery_mode_map()
subject_feeding_mode_map = su.parse_subject_feeding_mode_map()
sys.stderr.write("Done!\n")

# Cohorts
cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
mi_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

# Samples for each cohort
samples = {cohort: su.get_sample_names(cohort) for cohort in cohorts}
hmp_samples = su.get_sample_names('hmp')
mother_samples = su.get_sample_names('mother')
infant_samples = su.get_sample_names('infant')
olm_samples = su.get_sample_names('olm')
infant_samples = [sample for sample in infant_samples if sample not in olm_samples] # Exclude
Olm
mother_samples = [sample for sample in mother_samples if sample not in olm_samples] # Exclude
Olm
mi_samples = [sample for sample in (mother_samples + infant_samples) if sample not in
olm_samples] # Exclude Olm

# Sample-timepoint map
mi_sample_day_dict = su.get_mi_sample_day_dict(exclude_cohorts=['olm'])
mi_tp_sample_dict = su.get_mi_tp_sample_dict(exclude_cohorts=['olm']) # no binning
mi_tp_sample_dict_binned, mi_tp_binned_labels =
su.get_mi_tp_sample_dict(exclude_cohorts=['olm'], binned=True)

# Narrow down samples to no Olm, and mother only at delivery
infant_samples = [sample for sample in infant_samples if sample not in olm_samples]
mother_samples = [sample for sample in mother_samples if (mi_sample_day_dict[sample] < 7 and
mi_sample_day_dict[sample] > -7)]
mi_samples = [sample for sample in (mother_samples + infant_samples) if sample not in
olm_samples]

# Sample identity wrapper functions
is_infant = lambda sample: sample in infant_samples
is_mother = lambda sample: sample in mother_samples
is_adult = lambda sample: sample in hmp_samples
```

```
# In[3]:


# ======================================================================
# Load pickled data
# ======================================================================

# Parameters
sweep_type = 'full'
pp_prev_cohort = 'all'
min_coverage = 0

ddir = config.data_directory
pdir = "%s/pickles/cov%i_prev_%s/" % (ddir, min_coverage, pp_prev_cohort)

snp_changes = pickle.load(open('%s/big_snp_changes_%s.pkl' % (pdir, sweep_type), 'rb'),
encoding='latin1')
gene_changes = pickle.load(open('%s/big_gene_changes_%s.pkl' % (pdir, sweep_type), 'rb'),
encoding='latin1')
snp_change_freqs = pickle.load(open('%s/snp_change_freqs_%s.pkl' % (pdir, sweep_type), 'rb'),
encoding='latin1')
snp_change_null_freqs = pickle.load(open('%s/snp_change_null_freqs_%s.pkl' % (pdir, sweep_type),
'rb'), encoding='latin1')
gene_gain_freqs = pickle.load(open('%s/gene_gain_freqs_%s.pkl' % (pdir, sweep_type), 'rb'),
encoding='latin1')
gene_loss_freqs = pickle.load(open('%s/gene_loss_freqs_%s.pkl' % (pdir, sweep_type), 'rb'),
encoding='latin1')
gene_loss_null_freqs = pickle.load(open('%s/gene_loss_null_freqs_%s.pkl' % (pdir, sweep_type),
'rb'), encoding='latin1')
between_snp_change_counts = pickle.load(open('%s/between_snp_change_counts_%s.pkl' % (pdir,
sweep_type), 'rb'), encoding='latin1')
between_gene_change_counts = pickle.load(open('%s/between_gene_change_counts_%s.pkl' % (pdir,
sweep_type), 'rb'), encoding='latin1')

# nonconsecutive
pdir = "%s/pickles/cov%i_prev_%s/nonconsecutive/" % (ddir, min_coverage, pp_prev_cohort)

dnds_info = pickle.load(open('%s/dnds_info.pkl' % (pdir), 'rb'), encoding='latin1')
snp_change_freqs_with_opps = pickle.load(open('%s/snp_change_freqs_with_opps_full.pkl' % (pdir),
'rb'), encoding='latin1')


# In[4]:


# Calculate number of days for a sample pair (mother-infant)
def mi_sample_pair_to_days(sample1, sample2):
    days = mi_sample_day_dict[sample2] - mi_sample_day_dict[sample1]
    return days

# Rough approximation of HMP time intervals
def adult_sample_pair_to_days(sample1, sample2):
    order1 = sample_order_map[sample1][1]
    order2 = sample_order_map[sample2][1]
    return (order2 - order1)*183

# Unify days calculation (assume if not AA, then must be mother-infant)
def sample_pair_to_days(sample1, sample2):
    if is_adult(sample1) and is_adult(sample2):
```

```
        return adult_sample_pair_to_days(sample1, sample2)
    else:
        return mi_sample_pair_to_days(sample1, sample2)


# In[5]:


# Define function for bootstrapping

def bootstrapped_agg_from_list_dict(listdict, aggregator_fn, num_bootstraps=20, n=50):
    bootstrapped_agg_dict = defaultdict(list)
    for key in listdict:
        all_items = listdict[key]
        if len(all_items) < n:
            print("Skipping %s" % key)
            continue
        for _ in np.arange(num_bootstraps):
            bootstrap = [random.choice(all_items) for i in np.arange(n)]
            bootstrapped_agg_dict[key].append(aggregator_fn(bootstrap))
    return bootstrapped_agg_dict

# Plot function
def plot_interval(y, xstart, xstop, color='b', tickh=0.1):
    """Plot interval at y from xstart to xstop with given color."""
    plt.hlines(y, xstart, xstop, color, lw=1)
    plt.vlines(xstart, y+tickh, y-tickh, color, lw=1)
    plt.vlines(xstop, y+tickh, y-tickh, color, lw=1)

# Plot function on specific ax
def plot_interval_on_ax(ax, y, xstart, xstop, color='b', tickh=0.1):
    """Plot interval at y from xstart to xstop with given color."""
    ax.hlines(y, xstart, xstop, color, lw=1)
    ax.vlines(xstart, y+tickh, y-tickh, color, lw=1)
    ax.vlines(xstop, y+tickh, y-tickh, color, lw=1)

# Significance tests
# function to calculate Cohen's d for independent samples
def cohenD(d1, d2):
    n1, n2 = len(d1), len(d2)
    s1, s2 = np.var(d1, ddof=1), np.var(d2, ddof=1)
    s = np.sqrt(((n1 - 1) * s1 + (n2 - 1) * s2) / (n1 + n2 - 2))
    u1, u2 = np.mean(d1), np.mean(d2)
    return (u1 - u2) / s

def summarize_ttest(a, b, simple=False):
    t, p = scipy.stats.ttest_ind(a, b)
    D = cohenD(a, b)
    if simple:
        print("t=%.04f" % t)
        print("P=" + str(p))
        print("d=" + str(D))
    else:
        print("Group 1 size: %i | Group 2 size: %i" % (len(a), len(b)))
        print("T-statistic: %.04f" % t)
        print("P-value: " + str(p))
        print("Cohen's D: " + str(cohenD(a, b)))
    return t, p, D

def MWU_effect_size(p, n1, n2):
    return stats.norm.isf(p/2.0)/np.sqrt(n1 + n2)
```

```
def summarize_utest(a, b, simple=False):
    U, p = scipy.stats.mannwhitneyu(a, b)
    es = MWU_effect_size(p, len(a), len(b))
    if simple:
        print("U=%.04f" % U)
        print("P=" + str(p))
        print("Effect size: " + str(es))
    else:
        print("Group 1 size: %i | Group 2 size: %i" % (len(a), len(b)))
        print("U-statistic: %.04f" % U)
        print("P-value: " + str(p))
        print("Effect size: " + str(MWU_effect_size(p, len(a), len(b))))
    return U, p, es

# Statistical significance to asterisk representation mapping
def get_sig_str(pval):
    if pval <= 0.001:
        return '***'
    elif pval <= 0.01:
        return '**'
    elif pval <= 0.05:
        return '*'
    elif pval > 0.05:
        return 'ns'


# In[6]:


def permutation_test_p(nulls, true):
    # Firstly check if true is above or below median
    median = np.median(nulls); n = len(nulls)
    if true < median: # Below median
        sorted_nulls = sorted(nulls)
        for rank, null in zip(np.arange(0, n), sorted_nulls):
            if true < null:
                return rank/float(n)
    else: # Above median
        sorted_nulls = sorted(nulls, reverse=True)
        for rank, null in zip(np.arange(n, 0, step=-1), sorted_nulls):
            if true > null:
                return 1-(rank/float(n))


# In[7]:


# Define function needed for survival curve plots

def calculate_unnormalized_survival_from_vector(counts):
    counts = sorted(counts)
    xs = [0]
    ns = [len(counts)]
    ns_cur = len(counts)
    min_count = -1
    for count in counts:
        if count > min_count:
            ns.append(ns_cur) # Number of elements greater or equal
            xs.append(count)
            min_count = count
```

```
        ns_cur -= 1
    xs.append(xs[len(xs)-1]+1)
    ns.append(0)
    return xs, np.array(ns)

# Calculating rate from tuples
def get_rate_from_tuples(sweep_day_tuple_list):
    total_sweeps = 0
    total_days = 0
    for num_sweeps, days in sweep_day_tuple_list:
        total_sweeps += num_sweeps
        total_days += days
    rate = float(total_sweeps)/total_days
    return rate

# Calculating replacement rate from all tuples
def get_replacement_rate_from_tuples(snp_diffs_day_tuple_list):
    total_replacements = 0
    total_days = 0
    for num_snp_changes, days in snp_diffs_day_tuple_list:
        if num_snp_changes >= 500:
            total_replacements += 1
        total_days += days
    rate = (float(total_replacements)/total_days)
    return rate


# In[8]:


# Custom sample pair cohorts [not just sample!]
# Alternate version where a sample pair may be assigned multiple cohorts

# custom_cohort_tests: dictionary where keys are cohort names and values are
# boolean functions taking in sample_i, sample_j as arguments

def custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j, species='dummy'):
    my_cohorts = set()
    for cohort in custom_cohort_tests:
        if custom_cohort_tests[cohort](sample_i, sample_j, species):
            my_cohorts.add(cohort)
    return my_cohorts

# Establish custom cohorts to be used for all three rates plots

custom_cohort_tests = {}

custom_cohort_tests['Mother-Infant(all)']        = lambda s1, s2, sp: (is_mother(s1) and
is_infant(s2))

# Within the first week (BOTH mother and infant timepoints!)
custom_cohort_tests['Mother-Infant']             = lambda s1, s2, sp: ((is_mother(s1) and
is_infant(s2))
                                                                       and mi_sample_day_dict[s1] >= 0 and
mi_sample_day_dict[s1] <= 7
                                                                       and mi_sample_day_dict[s2] <= 7)

custom_cohort_tests['Infant-Infant']        = lambda s1, s2, sp: (is_infant(s1) and
is_infant(s2))

custom_cohort_tests['Day 0-Week 1']   = lambda s1, s2, sp: ((is_infant(s1) and is_infant(s2))
```

```
                                                    and mi_sample_day_dict[s1] >= 0
                                                    and mi_sample_day_dict[s2] <= 7)

custom_cohort_tests['Week 1-Month 1'] = lambda s1, s2, sp: ((is_infant(s1) and is_infant(s2))
                                                    and mi_sample_day_dict[s1] >= 7
                                                    and mi_sample_day_dict[s2] <= 31)

custom_cohort_tests['Month 1-Year 1'] = lambda s1, s2, sp: ((is_infant(s1) and is_infant(s2))
                                                    and mi_sample_day_dict[s1] >= 31
                                                    and mi_sample_day_dict[s2] <= 367)

custom_cohort_tests['Adult-Adult']             = lambda s1, s2, sp: (is_adult(s1) and
is_adult(s2))

# 1mon and 3mon are alternatives
custom_cohort_tests['II-1mon'] = lambda s1, s2, sp: ((is_infant(s1) and is_infant(s2))
                                                     and (sample_pair_to_days(s1, s2) < 32))

custom_cohort_tests['II-3mon'] = lambda s1, s2, sp: ((is_infant(s1) and is_infant(s2))
                                                    and (sample_pair_to_days(s1, s2) <= 90))

custom_cohort_tests['II-1yr'] = lambda s1, s2, sp: ((is_infant(s1) and is_infant(s2))
                                                    and (sample_pair_to_days(s1, s2) > 90))

custom_cohort_tests['6mon duration infant'] = lambda s1, s2, sp: ((is_infant(s1) and
is_infant(s2))
                                                                  and (sample_pair_to_days(s1,
s2) >= 175)
                                                                  and (sample_pair_to_days(s1, s2)
<= 191))

custom_cohort_tests['6mon duration adult'] = lambda s1, s2, sp: ((is_adult(s1) and is_adult(s2))
                                                                  and (sample_pair_to_days(s1,
s2) >= 175)
                                                                  and (sample_pair_to_days(s1, s2)
<= 191))

# 6 months +/- 8 days: if days >= 175 and days <= 191

custom_cohort_tests['4-8mon duration infant'] = lambda s1, s2, sp: ((is_infant(s1) and
is_infant(s2))
                                                                  and (sample_pair_to_days(s1,
s2) >= (4*30.5))
                                                                  and (sample_pair_to_days(s1, s2)
<= (8*30.5)))

custom_cohort_tests['4-8mon duration adult'] = lambda s1, s2, sp: ((is_adult(s1) and
is_adult(s2))
                                                                  and (sample_pair_to_days(s1,
s2) >= (4*30.5))
                                                                  and (sample_pair_to_days(s1, s2)
<= (8*30.5)))


# In[9]:


# species -> mother_sample -> infant_sample -> # days of infant_sample
# mother timepoint at delivery (-1 to 7 days)
mi_dict = {species: defaultdict(dict) for species in gene_gain_freqs}
```

```
for species in snp_changes:
    for sample_i, sample_j in snp_changes[species]:

        # MI: restrict to earliest mother-infant pair per host
        if 'Mother-Infant(all)' in custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i,
sample_j):
            mother_days = mi_sample_day_dict[sample_i]
            if mother_days >= -1 and mother_days <= 7:
                mi_dict[species][sample_i][sample_j] = mi_sample_day_dict[sample_j]

# species -> mother_sample -> earliest infant sample
mi_earliest_infant_sample_dict = defaultdict(dict)

# Distribution of which day the earliest infant tp occurs at
# Hopefully nothing too late
earliest_infant_days = []

for species in mi_dict:
    for mother_sample in mi_dict[species]:
        ordered_infant_days = sorted(mi_dict[species][mother_sample].items(), key=lambda x:
x[1])
        infant_sample, days = ordered_infant_days[0]
        mi_earliest_infant_sample_dict[species][mother_sample] = infant_sample
        earliest_infant_days.append(days)


# In[10]:


print(sorted(earliest_infant_days))


# In[11]:


# Add one more category
custom_cohort_tests['Mother-Infant(earliest)']  = lambda s1, s2, sp: ((is_mother(s1) and
is_infant(s2)
                                                                      and (s1 in
mi_earliest_infant_sample_dict[sp])
                                                                      and (s2 ==
mi_earliest_infant_sample_dict[sp][s1])))


# In[12]:


# General permutation test setup
# Have two versions: permute labels by subject vs. by sample

# Returns null_differences and true_difference
# Assumes only permuting two things, and difference will be for label_set[0]-label_set[1]
def permutation_test(orig_label_data_dict, label_set, aggregator_fn, num_bootstraps=1000):

    orig_data = []; labels = []
    for label in label_set:
        for orig_datum in orig_label_data_dict[label]:
            orig_data.append(orig_datum)
            labels.append(label)

    # First compute true rate
```

```
    true_value_by_label = {label: aggregator_fn(orig_label_data_dict[label]) for label in
orig_label_data_dict}

    # Next compute null rates
    permuted_values_by_label = {label: [] for label in label_set}

    for _ in range(num_bootstraps):
        random.shuffle(labels) # Randomly permute labels
        label_data_dict = defaultdict(list)
        for datum, label in zip(orig_data, labels):
            label_data_dict[label].append(datum)

        for label in label_set:
            permuted_values_by_label[label].append(aggregator_fn(label_data_dict[label]))

    # Compute differences between the labels
    label1, label2 = label_set[0], label_set[1]

    null_differences = []
    for val1, val2 in zip(permuted_values_by_label[label1], permuted_values_by_label[label2]):
        null_differences.append(val1-val2)

    true_difference = true_value_by_label[label1]-true_value_by_label[label2]

    return (null_differences, true_difference)


# In[13]:


# This time orig_label_subject_data_dict values must be of the form (subject, datum)
def permutation_test_subject(orig_label_subject_data_dict, label_set, aggregator_fn,
num_bootstraps=1000):

    subjects = []; labels = []

    for label in label_set:
        for subject, orig_datum in orig_label_subject_data_dict[label]:
            if subject not in subjects: # First appearance
                labels.append(label)
                subjects.append(subject)

    # First compute true rate
    true_value_by_label = {}
    for label in label_set:
        orig_data_list = [orig_datum for subject, orig_datum in
orig_label_subject_data_dict[label]]
        true_value_by_label[label] = aggregator_fn(orig_data_list)

    # Next compute null rates
    permuted_values_by_label = {label: [] for label in label_set}

    for _ in range(num_bootstraps):
        random.shuffle(labels) # Randomly permute labels according to subject
        subject_label_dict = {subject: label for subject, label in zip(subjects, labels)}

        label_data_dict = defaultdict(list)
        for label in label_set: # Basically ignore old label
            for subject, datum in orig_label_subject_data_dict[label]:
                new_label = subject_label_dict[subject]
                label_data_dict[new_label].append(datum)
```

```
        for label in label_set:
            permuted_values_by_label[label].append(aggregator_fn(label_data_dict[label]))

    # Compute differences between the labels
    label1, label2 = label_set[0], label_set[1]

    null_differences = []
    for val1, val2 in zip(permuted_values_by_label[label1], permuted_values_by_label[label2]):
        null_differences.append(val1-val2)

    true_difference = true_value_by_label[label1]-true_value_by_label[label2]

    return (null_differences, true_difference)


# In[14]:


feeding_modes = ['breast', 'mixed', 'formula']
delivery_modes = ['Vaginal', 'C-section']

dmode_subjects = list(subject_delivery_mode_map.keys())
fmode_subjects = list(subject_feeding_mode_map.keys())
comb_subjects = list(set(dmode_subjects).intersection(set(fmode_subjects)))
print("%i subjects with both delivery and feeding mode information" % len(comb_subjects))

dmode_fmode_dict = {dmode: {fmode: 0 for fmode in feeding_modes} for dmode in delivery_modes}

for subject in comb_subjects:
    dmode = subject_delivery_mode_map[subject]
    fmode = subject_feeding_mode_map[subject]
    dmode_fmode_dict[dmode][fmode] += 1

for dmode in dmode_fmode_dict:
    for fmode in dmode_fmode_dict[dmode]:
        print("%s\t%s: %i" % (dmode, fmode, dmode_fmode_dict[dmode][fmode]))

# Try to get numbers for QP pairs


# In[16]:


# Do four-way comparison
'''
Vaginal vs. c-section, breast only
Vaginal vs. c-section, formula only
Breast vs. formula, Vaginal only
Breast vs. formula, C-section only
'''



# # =================
# # SNV change rates
# # =================

# In[16]:


# MUST RUN THIS FOR SNV CHANGE AND REPLACEMENT RATES
```

```
# Store SNP change information in CSV

'''
f = open('%s/snp_change_rate.csv' % config.analysis_directory, 'w')
f.write(','.join(['sample_1', 'sample_2', 'species', 'subject', 'cohort', 'day_1', 'day_2',
'days', 'day_mid', 'tp_cat',
                  'snp_changes', 'snp_change_rate', 'breast', 'formula', 'vaginal', 'csection'])
+ '\n')
'''
tp_types_all = list(custom_cohort_tests.keys())
tp_types_subset = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1']

# Store SNP change information

replacement_time_tups_by_tp_type = defaultdict(list)
all_time_tups_by_tp_type = defaultdict(list)
subject_all_time_tups_by_tp_type = defaultdict(list)
count_time_tups_by_tp_type = defaultdict(list)
subject_count_time_tups_by_tp_type = defaultdict(list)

for species in snp_changes:
    for s1, s2 in snp_changes[species]:

        subject = sample_subject_map[s2]
        cohort = sample_cohort_map[s2]

        val = snp_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)


        # SPECIAL: try recoding mother-infant duration as sum
        if 'Mother-Infant' in custom_cohorts:
            day1 = mi_sample_day_dict[s1]
            day2 = mi_sample_day_dict[s2]
            days = day1 + day2
            if days == 0:
                days = 1

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue

        '''
        # Arbitrarily set same mother and infant (in particular, 0-0 or 1-1) as 1 day
        if is_mother(s1) and is_infant(s2) and mi_sample_day_dict[s2] == mi_sample_day_dict[s1]:
            days = 1
        '''
        if isinstance(val, int): # Replacement
            for custom_cohort in custom_cohorts:

                replacement_time_tups_by_tp_type[custom_cohort].append((1, days))

                all_time_tups_by_tp_type[custom_cohort].append((val, days))
                subject_all_time_tups_by_tp_type[custom_cohort].append((subject, val, days))

        else: # Not replacement (modifications/no change)
            for custom_cohort in custom_cohorts:

                count_time_tups_by_tp_type[custom_cohort].append((len(val), days))
```

```
                subject_count_time_tups_by_tp_type[custom_cohort].append((subject, len(val),
days))

                all_time_tups_by_tp_type[custom_cohort].append((len(val), days))
                subject_all_time_tups_by_tp_type[custom_cohort].append((subject, len(val),
days))

                rate = len(val)/float(days)

                if custom_cohort in tp_types_subset: # Should be mother-infant or infant-infant
in unique tp bins
                    day1 = mi_sample_day_dict[s1]; day2 = mi_sample_day_dict[s2]
                    day_mid = day1 + (days/2.0)
                    feeding_mode = subject_feeding_mode_map[subject] if subject in
subject_feeding_mode_map else 'NA'
                    if feeding_mode == 'breast':
                        breast = 1; formula = 0
                    elif feeding_mode == 'mixed':
                        breast = 1; formula = 1
                    elif feeding_mode == 'formula':
                        breast = 0; formula = 1
                    delivery_mode = subject_delivery_mode_map[subject] if subject in
subject_delivery_mode_map else 'NA'
                    if delivery_mode == 'Vaginal':
                        vaginal = 1; csection = 0
                    elif delivery_mode == 'C-section':
                        vaginal = 0; csection = 1

                    '''
                    f.write(','.join([str(x) for x in [s1, s2, species, subject, cohort, day1,
day2, days, day_mid,
                                                       custom_cohort, len(val), rate, breast,
formula, vaginal, csection]]) + '\n')
                    '''

# f.close()


# In[17]:


# Investigate the Mother-Infant category
# Namely, how much does Backhed higher resolution timepoint assignment change results?
# Are rates being inflated due to a smaller denominator due to mother timepoints
# being mistakenly considered 0 when they are actually not 0?
total_days = 0
n = 0
day_tups = []

for species in snp_changes:
    for s1, s2 in snp_changes[species]:

        subject = sample_subject_map[s2]
        cohort = sample_cohort_map[s2]

        val = snp_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        if 'Mother-Infant' in custom_cohorts:
            day1 = mi_sample_day_dict[s1]
```

```
            day2 = mi_sample_day_dict[s2]
            # print('Mother %i\tInfant %i' % (day1, day2))
            day_tups.append((day1, day2))
            if days >= 0:
                total_days += days
                n += 1

fig, ax = plt.subplots()
idx = 0
for day1, day2 in sorted(day_tups):
    ax.plot([day1, day2], [idx, idx], '.-')
    idx += 1
ax.set_xlabel("Day after birth\nFirst timepoint is mother, second is infant")
plt.show()


# In[18]:


# Subsample QP pairs in a category to n=40 (since MI only has 44) and get #sweeps/day, bootstrap
20 times
bootstrapped_rates_by_tp_type = bootstrapped_agg_from_list_dict(count_time_tups_by_tp_type,
get_rate_from_tuples,
                                                                num_bootstraps=1000, n=40)

# SNP change rates boxplot
fig, ax = plt.subplots(figsize=(10,4))

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']
annotations = [] # These are the true "average" rates per category
tp_type_labels = [] # These are used as x-axis labels

bootstrapped_rates = [bootstrapped_rates_by_tp_type[tp_type] for tp_type in tp_types]
for tp_type in tp_types:
    if len(count_time_tups_by_tp_type[tp_type]) == 0:
        annotations.append(0)
        tp_type_labels.append('%s\nn=%i' % (tp_type, len(count_time_tups_by_tp_type[tp_type])))
        continue
    snp_count, total_days = (0,0)
    for count, days in count_time_tups_by_tp_type[tp_type]:
        snp_count += count
        total_days += days
    annotations.append(float(snp_count)/total_days)
    tp_type_labels.append('%s\nn=%i' % (tp_type, len(count_time_tups_by_tp_type[tp_type])))

ax.plot(np.arange(1, 1+len(annotations)), annotations, marker=(5, 2), linestyle='None',
markersize=12)
ax.boxplot(bootstrapped_rates)
# ax.set_ylim(0, 0.14)
ax.set_yscale('log')
ax.set_xticklabels(tp_type_labels, fontsize=12)
ax.set_ylabel("SNP changes per QP pair per day", fontsize=12)
ax.set_title("Bootstrapped (subsample QP pairs 50 times), include zeros")
for tp_type, annotation in zip(tp_types, annotations):
    print("%s: %s" % (tp_type, annotation))

plt.show()


# In[19]:
```

```
# Perform PERMUTATION tests between all pairs

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

snp_change_rate_tp_pair_ptests = {} # (tp1, tp2) -> (n1, n2, p)

print("SNP CHANGE RATE TIMEPOINT PAIRWISE COMPARISONS")

for i in range(len(tp_types)):
    tp_type1 = tp_types[i]
    for j in range(i+1, len(tp_types)):
        tp_type2 = tp_types[j]
        # Permutation test for tp_type 1 vs._tp type2
        label_set = [tp_type1, tp_type2]
        orig_label_data_dict = {label: count_time_tups_by_tp_type[label] for label in label_set}
        null_differences, true_difference = permutation_test(orig_label_data_dict, label_set,
                                                             aggregator_fn=get_rate_from_tuples,
                                                            num_bootstraps=10000)
        p = permutation_test_p(null_differences, true_difference)
        n1 = len(orig_label_data_dict[tp_type1])
        n2 = len(orig_label_data_dict[tp_type2])
        snp_change_rate_tp_pair_ptests[(tp_type1, tp_type2)] = (n1, n2, p)
        print("==============================")
        print("%s vs. %s" % (tp_type1, tp_type2))
        print(str(p) + '\t' + get_sig_str(p))


# In[20]:


# Store for later use
snp_change_rates = bootstrapped_rates
snp_change_rate_annotations = annotations
snp_change_rate_tp_type_labels = tp_type_labels

# Get statistics for paper
print(snp_change_rate_annotations)


# In[46]:


# Matched
# SNP change rates boxplot
# Rerun above block for combined infant vs. adult figure

# Uses bootstrapped_rates_by_tp_type from above with num_bootstraps=1000, n=40

fig, ax = plt.subplots(figsize=(4,4))

tp_types = ['4-8mon duration infant', '4-8mon duration adult']
labels = ['4-8mon duration\nInfant', '4-8mon duration\nAdult']
annotations = [] # These are the true "average" rates per category
tp_type_labels = [] # These are used as x-axis labels

bootstrapped_rates = [bootstrapped_rates_by_tp_type[tp_type] for tp_type in tp_types]
for tp_type, label in zip(tp_types, labels):
    snp_count, total_days = (0,0)
    for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
        snp_count += count
```

```
        total_days += days
    annotations.append(float(snp_count)/total_days)
    tp_type_labels.append('%s\nn=%i' % (label, len(count_time_tups_by_tp_type[tp_type])))

ax.plot(np.arange(1, 1+len(annotations)), annotations, marker=(5, 2), linestyle='None',
markersize=12)
ax.boxplot(bootstrapped_rates)
# ax.set_ylim(0, 0.14)
ax.set_xticklabels(tp_type_labels, fontsize=12)
ax.set_ylabel("SNP changes per day", fontsize=12)
# ax.set_title("Bootstrapped (subsample QP pairs 50 times), include zeros")

t, p = stats.ttest_ind(bootstrapped_rates[0], bootstrapped_rates[1])
print("T: %.02f\tp: %.3E" % (t, p))
plt.show()


# In[135]:


# Old permutation test code...
'''
num_bootstraps = 1000

orig_data = []; labels = []

for label in label_set:
    for count, days in count_time_tups_by_tp_type[label]:
        orig_data.append((count, days))
        labels.append(label)

label_data_dict = defaultdict(list)
for datum, label in zip(orig_data, labels):
    label_data_dict[label].append(datum)

# First compute true rate
true_value_by_label = {}
for label in label_data_dict:
    snp_count, total_days = (0,0)
    for count, days in label_data_dict[label]:
        snp_count += count
        total_days += days
    true_value_by_label[label] = (float(snp_count)/total_days)

permuted_values_by_label = {label: [] for label in label_set}

i = 0
while i < num_bootstraps:

    random.shuffle(labels) # Randomly permute labels
    label_data_dict = defaultdict(list)
    for datum, label in zip(orig_data, labels):
        label_data_dict[label].append(datum)


    for label in label_set:
        snp_count, total_days = (0,0)
        for count, days in label_data_dict[label]:
            snp_count += count
            total_days += days
        permuted_values_by_label[label].append(float(snp_count)/total_days)
```

```
    i += 1
'''
print()


# In[32]:


# Permutation test for infant vs. adult

label_set = ['4-8mon duration infant', '4-8mon duration adult']
orig_label_data_dict = {label: count_time_tups_by_tp_type[label] for label in label_set}
null_differences, true_difference = permutation_test(orig_label_data_dict, label_set,
aggregator_fn=get_rate_from_tuples,
                                                     num_bootstraps=10000)

# Plot permutation test results
fig, ax = plt.subplots(figsize=(2.3, 4))
p = permutation_test_p(null_differences, true_difference)
color = 'red' if p < 0.05 else 'gray'
ax.violinplot(null_differences)
ax.plot([1], [true_difference], marker=(5, 2), linestyle='None', markersize=12, color=color,
zorder=9)
ax.set_xticks([1]); ax.set_xticklabels([]); ax.set_xlim((0.6, 1.4))
ax.set_ylabel("Difference (Infant-Adult)", fontsize=12)
ax.set_xlabel(r'$\it{p}$' + ("<0.0001" if p == 0 else "=%.03f" % p), fontsize=12)
plt.tight_layout()
fig.savefig('%s/infant_vs_adult_comparison_snv_change_rate.pdf' % config.analysis_directory)


# In[61]:


# Perform tests between all pairs

snp_change_rate_tp_pair_tests = {} # (tp1, tp2) -> (t, p, es) Store for later table

print("SNP CHANGE RATE TIMEPOINT PAIRWISE COMPARISONS")
print("Note: all sample sizes are equal to number of bootstraps")

for i in range(len(tp_types)):
    ind_rates1 = bootstrapped_rates[i]; tp_type1 = tp_types[i]
    for j in range(i+1, len(tp_types)):
        ind_rates2 = bootstrapped_rates[j]; tp_type2 = tp_types[j]
        print("============================")
        print("%s vs. %s" % (tp_type1, tp_type2))
        stat, p, es = summarize_utest(ind_rates1, ind_rates2, simple=True)
        snp_change_rate_tp_pair_tests[(tp_type1, tp_type2)] = (stat, p, es)
        print(get_sig_str(p))


# In[30]:


# Store SNP change information, separate by cohort

cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']

replacement_time_tups_by_cohort_tp_type = {ds: defaultdict(list) for ds in cohorts}
all_time_tups_by_cohort_tp_type = {ds: defaultdict(list) for ds in cohorts}
```

```
count_time_tups_by_cohort_tp_type = {ds: defaultdict(list) for ds in cohorts}

for species in snp_changes:
    for s1, s2 in snp_changes[species]:

        dataset = sample_cohort_map[s1]
        val = snp_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue

        # Arbitrarily set mother 0 - infant 0 as 1 day
        if is_mother(s1) and mi_sample_day_dict[s1] == 0 and is_infant(s2) and
mi_sample_day_dict[s2] == 0:
            days = 1

        if isinstance(val, int): # Replacement
            for custom_cohort in custom_cohorts:
                replacement_time_tups_by_cohort_tp_type[dataset][custom_cohort].append((1,
days))
                all_time_tups_by_cohort_tp_type[dataset][custom_cohort].append((val, days))
        else: # Not replacement (modifications/no change)
            for custom_cohort in custom_cohorts:
                count_time_tups_by_cohort_tp_type[dataset][custom_cohort].append((len(val),
days))
                all_time_tups_by_cohort_tp_type[dataset][custom_cohort].append((len(val), days))


# In[31]:


# Matched timepoint comparison

n = 10
num_bootstraps = 1000

bootstrapped_rates_by_cohort_tp_type = defaultdict(dict)

for tp_type in tp_types:
    for dataset in ['backhed', 'ferretti', 'yassour', 'shao']:
        tups = count_time_tups_by_cohort_tp_type[dataset][tp_type]
        if len(tups) < n:
            print("Not enough data")
            continue
        average = get_rate_from_tuples(tups)
        bootstrapped_rates = []
        for _ in np.arange(num_bootstraps):
            bootstrap = [random.choice(tups) for i in np.arange(n)]
            bootstrapped_rates.append(get_rate_from_tuples(bootstrap))

        bootstrapped_rates_by_cohort_tp_type[dataset][tp_type] = bootstrapped_rates


# In[32]:


for tp_type in tp_types:
```

```
    print("==================================")
    print(tp_type)
    datasets_with_data = []
    for dataset in ['backhed', 'ferretti', 'yassour', 'shao']:
        if tp_type in bootstrapped_rates_by_cohort_tp_type[dataset]:
            datasets_with_data.append(dataset)
    for i in range(len(datasets_with_data)):
        d1 = datasets_with_data[i]
        for j in range(i+1, len(datasets_with_data)):
            d2 = datasets_with_data[j]
            t, p = stats.ttest_ind(bootstrapped_rates_by_cohort_tp_type[d1][tp_type],
bootstrapped_rates_by_cohort_tp_type[d2][tp_type])
            print("%s vs %s: t= %.03f, P=%s" % (d1, d2, t, str(p)))


# In[34]:


# SNP change rates boxplot, matched timepoint comparison
fig, ax = plt.subplots(1, 4, figsize=(12, 4), sharey=True)

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

for i in range(4):

    dataset = cohorts[i]
    # Subsample QP pairs in a category to n=50 and get #sweeps/day, bootstrap 1000 times
    bootstrapped_rates_by_tp_type =
bootstrapped_agg_from_list_dict(count_time_tups_by_cohort_tp_type[dataset],
get_rate_from_tuples,
                                                                    num_bootstraps=1000, n=20)

    annotations = [] # These are the true "average" rates per category
    tp_type_labels = [] # These are used as x-axis labels
    bootstrapped_rates = [] # Data

    for tp_type in tp_types:
        if len(bootstrapped_rates_by_tp_type[tp_type]) < 10:
            continue
        snp_count, total_days = (0,0)
        for count, days in count_time_tups_by_cohort_tp_type[dataset][tp_type]:
            snp_count += count
            total_days += days
        bootstrapped_rates.append(bootstrapped_rates_by_tp_type[tp_type])
        annotations.append(float(snp_count)/total_days)
        tp_type_labels.append('%s\nn=%i' % (tp_type,
len(count_time_tups_by_cohort_tp_type[dataset][tp_type])))

    ax[i].plot(np.arange(1, 1+len(annotations)), annotations, marker=(5, 2), linestyle='None',
markersize=12)
    ax[i].boxplot(bootstrapped_rates)
    # ax.set_ylim(0, 0.14)
    ax[i].set_yscale('log')
    ax[i].set_xticklabels(tp_type_labels, fontsize=12)
    ax[i].set_title(dataset)
    # ax[i].set_title("Bootstrapped (subsample QP pairs 50 times), include zeros")

ax[0].set_ylabel("SNP changes per QP pair\nper day", fontsize=12)
plt.tight_layout()
plt.show()
```

```
# In[165]:


# Compare delivery modes

cats = ['Vaginal', 'C-section']
replacement_time_tups_by_cat_tp_type = {cat: defaultdict(list) for cat in cats}

all_time_tups_by_cat_tp_type = {cat: defaultdict(list) for cat in cats}
subject_all_time_tups_by_tp_type = defaultdict(list) # For later delivery mode permutation

count_time_tups_by_cat_tp_type = {cat: defaultdict(list) for cat in cats}
subject_count_time_tups_by_tp_type = defaultdict(list) # For later delivery mode permutation

for species in snp_changes:
    for s1, s2 in snp_changes[species]:

        subject = sample_subject_map[s2]
        if subject not in subject_delivery_mode_map: # Skip non-infant second sample
            continue

        delivery_mode = subject_delivery_mode_map[subject]; cat = delivery_mode
        val = snp_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue

        # Arbitrarily set mother 0 - infant 0 as 1 day
        if is_mother(s1) and mi_sample_day_dict[s1] == 0 and is_infant(s2) and
mi_sample_day_dict[s2] == 0:
            days = 1

        if isinstance(val, int): # Replacement
            for custom_cohort in custom_cohorts:
                replacement_time_tups_by_cat_tp_type[cat][custom_cohort].append((1, days))
                all_time_tups_by_cat_tp_type[cat][custom_cohort].append((val, days))
                subject_all_time_tups_by_tp_type[custom_cohort].append((subject, val, days))
        else: # Not replacement (modifications/no change)
            for custom_cohort in custom_cohorts:
                count_time_tups_by_cat_tp_type[cat][custom_cohort].append((len(val), days))
                subject_count_time_tups_by_tp_type[custom_cohort].append((subject, len(val),
days))
                all_time_tups_by_cat_tp_type[cat][custom_cohort].append((len(val), days))
                subject_all_time_tups_by_tp_type[custom_cohort].append((subject, len(val),
days))


# In[166]:


# Permutation test for delivery mode
# 1. permute C and V for all the babies
# 2. compute the SNV change rate for new C and new V babies;
# make sure to take the sum of the changes divided by the total time
# 3. compute difference in rate
# 4. Repeat x1000.
```

```
# 5. Compare true difference between C and V

num_bootstraps = 1000
cats = ['Vaginal', 'C-section']

subjects = list(subject_delivery_mode_map.keys()) # Order matters!!
delivery_modes = [] # True/original
for subject in subjects:
    delivery_modes.append(subject_delivery_mode_map[subject])

true_snv_change_rate_by_cat_tp_type_dict = {cat: {} for cat in cats}

for tp_type in subject_count_time_tups_by_tp_type:
    total_days_by_cat_dict = defaultdict(int)
    total_snv_change_count_by_cat_dict = defaultdict(int)
    for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
        delivery_mode = subject_delivery_mode_map[subject]
        total_days_by_cat_dict[delivery_mode] += days
        total_snv_change_count_by_cat_dict[delivery_mode] += count

    for cat in cats:
        days = float(total_days_by_cat_dict[cat])
        snv_change_rate = total_snv_change_count_by_cat_dict[cat]/days
        true_snv_change_rate_by_cat_tp_type_dict[cat][tp_type] = snv_change_rate

permuted_snv_change_rates_by_cat_tp_type_dict = {cat: defaultdict(list) for cat in cats}

i = 0
while i < num_bootstraps:
    random.shuffle(delivery_modes) # Randomly permute delivery modes
    permuted_subject_delivery_mode_map = {}
    for subject, delivery_mode in zip(subjects, delivery_modes):
        permuted_subject_delivery_mode_map[subject] = delivery_mode

    bad = False # If end up with sample size 0 for any timepoint-cat pair, skip?? Not used for
now

    # Compute SNV change rate from subject_count_time_tups_by_tp_type
    for tp_type in subject_count_time_tups_by_tp_type:
        total_days_by_cat_dict = defaultdict(int)
        total_snv_change_count_by_cat_dict = defaultdict(int)
        for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
            delivery_mode = permuted_subject_delivery_mode_map[subject]
            total_days_by_cat_dict[delivery_mode] += days
            total_snv_change_count_by_cat_dict[delivery_mode] += count

        for cat in cats:
            days = float(total_days_by_cat_dict[cat])
            if days > 0:
                snv_change_rate = total_snv_change_count_by_cat_dict[cat]/days
            else:
                snv_change_rate = 0 # Consider it 0 if there are no samples/babies for this
timepoint-cat
            permuted_snv_change_rates_by_cat_tp_type_dict[cat][tp_type].append(snv_change_rate)

    i += 1

# Just interested in Week 0-Month 1 and Month 1-Year

fig, ax = plt.subplots(figsize=(4, 4))
```

```
tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

null_differences_by_tp_type = []
labels = []
true_differences_by_tp_type = []

for tp_type in tp_types:
    null_differences = []
    for vrate, crate in zip(permuted_snv_change_rates_by_cat_tp_type_dict['Vaginal'][tp_type],
                            permuted_snv_change_rates_by_cat_tp_type_dict['C-
section'][tp_type]):
        null_differences.append(vrate-crate)
    null_differences_by_tp_type.append(null_differences)

    true_vrate = true_snv_change_rate_by_cat_tp_type_dict['Vaginal'][tp_type]
    true_crate = true_snv_change_rate_by_cat_tp_type_dict['C-section'][tp_type]
    true_difference = true_vrate-true_crate
    true_differences_by_tp_type.append(true_difference)

    p = permutation_test_p(null_differences, true_difference)
    labels.append(("%s\n" % tp_type) + r'$\it{p}$' + ("=%.03f" % (p)))

ax.violinplot(null_differences_by_tp_type, positions=[1, 2])
ax.plot(np.arange(1, 1+len(true_differences_by_tp_type)), true_differences_by_tp_type,
        marker=(5, 2), linestyle='None', markersize=12, color='black', zorder=9)
ax.set_xticks([1, 2])
ax.set_xticklabels(labels)
ax.set_ylabel("Difference between vaginal\nand C-section SNV change rate")
plt.show()
fig.savefig('%s/delivery_mode_comparison_snv_change_rate.pdf' % config.analysis_directory)


# In[133]:


# Summarize distribution by delivery, feeding mode
time_interval_category = 'Day 0-Week 1'
feeding_delivery_count_dict = {fmode: defaultdict(int) for fmode in ['breast', 'formula',
'mixed']}
for subject, count, days in subject_count_time_tups_by_tp_type[time_interval_category]:
    if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
        feeding_mode = subject_feeding_mode_map[subject]
        delivery_mode = subject_delivery_mode_map[subject]
        feeding_delivery_count_dict[feeding_mode][delivery_mode] += 1

dmodes = ['Vaginal', 'C-section']
fmodes = ['breast', 'formula', 'mixed']
print("SNV changes | %s" % time_interval_category)
print('----------------------')
print('\t' + '\t'.join(dmodes))
for fmode in fmodes:
    vals = [str(feeding_delivery_count_dict[fmode][dmode]) for dmode in dmodes]
    print("%s\t" % (fmode) + '\t'.join(vals))


# In[187]:


# Reproduce in the more elegant way heh
# Permutation tests for delivery and feeding mode
```

```
fig, ax = plt.subplots(1, 2, figsize=(8, 4))

time_interval_categories = ['Day 0-Week 1', 'Week 1-Month 1']

# Vaginal vs. c-section
label_sets = [['Vaginal', 'C-section'], ['breast', 'formula']]
label_mode_type_list = [0, 1] # 0 for delivery mode, 1 for feeding mode

for i in range(2):
    label_set = label_sets[i] # Which two labels we are comparing
    label_mode_type = label_mode_type_list[i]

    null_differences_list = []
    true_difference_list = []
    labels = []
    for time_interval_category in time_interval_categories:
        orig_label_subject_data_dict = defaultdict(list)
        for label in label_set:
            for subject, count, days in
subject_count_time_tups_by_tp_type[time_interval_category]:
                if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                    feeding_mode = subject_feeding_mode_map[subject]
                    delivery_mode = subject_delivery_mode_map[subject]
                    if label_mode_type == 0:
                        orig_label_subject_data_dict[delivery_mode].append((subject, (count,
days)))
                    if label_mode_type == 1:
                        orig_label_subject_data_dict[feeding_mode].append((subject, (count,
days)))

        null_differences, true_difference =
permutation_test_subject(orig_label_subject_data_dict, label_set,

aggregator_fn=get_rate_from_tuples, num_bootstraps=10000)
        null_differences_list.append(null_differences)
        true_difference_list.append(true_difference)
        p = permutation_test_p(null_differences, true_difference)
        color = 'red' if p < 0.05 else 'black'
        labels.append('%s\n' % time_interval_category + r'$\it{p}$' + "=%.03f" % p)

    ax[i].violinplot(null_differences_list)
    ax[i].plot([1, 2], true_difference_list, marker=(5, 2), linestyle='None', markersize=12,
color=color, zorder=9)
    ax[i].set_xticks([1, 2])
    ax[i].set_xticklabels(labels)
    ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
    # ax[i].set_xlabel(r'$\it{p}$' + "=%.03f" % p, fontsize=12)
    ax[i].set_title("SNV change rate")

plt.subplots_adjust(wspace=0.5)
plt.show()
fig.savefig('%s/mode_comparisons_snv_change_rate.pdf' % config.analysis_directory)
fig.savefig('%s/mode_comparisons_snv_change_rate.png' % config.analysis_directory, dpi=200)


# In[96]:


# Reproduce in the more elegant way heh
# Permutation tests for delivery and feeding mode
```

```
fig, ax = plt.subplots(1, 4, figsize=(8, 4))

time_interval_category = 'Day 0-Week 1'

# Vaginal vs. c-section, breast only
# Vaginal vs. c-section, formula only
# Breast vs. formula, Vaginal only
# Breast vs. formula, C-section only
label_sets = [['Vaginal', 'C-section'], ['Vaginal', 'C-section'], ['breast', 'formula'],
['breast', 'formula']]
label_mode_type_list = [0, 0, 1, 1] # 0 for delivery mode, 1 for feeding mode
restrict_other_mode_list = ['breast', 'formula', 'Vaginal', 'C-section']

for i in range(4):
    label_set = label_sets[i] # Which two labels we are comparing
    label_mode_type = label_mode_type_list[i]
    restrict_other_mode = restrict_other_mode_list[i]

    orig_label_subject_data_dict = defaultdict(list)
    for label in label_set:
        for subject, count, days in subject_count_time_tups_by_tp_type[time_interval_category]:
            if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                feeding_mode = subject_feeding_mode_map[subject]
                delivery_mode = subject_delivery_mode_map[subject]
                if label_mode_type == 0:
                    if feeding_mode == restrict_other_mode:
                        orig_label_subject_data_dict[delivery_mode].append((subject, (count,
days)))
                if label_mode_type == 1:
                    if delivery_mode == restrict_other_mode:
                        orig_label_subject_data_dict[feeding_mode].append((subject, (count,
days)))

    null_differences, true_difference = permutation_test_subject(orig_label_subject_data_dict,
label_set, aggregator_fn=get_rate_from_tuples)

    p = permutation_test_p(null_differences, true_difference)
    color = 'red' if p < 0.05 else 'black'
    ax[i].violinplot(null_differences)
    ax[i].plot([1], [true_difference], marker=(5, 2), linestyle='None', markersize=12,
color=color, zorder=9)
    ax[i].set_xticks([1]); ax[i].set_xticklabels([]); ax[i].set_xlim((0.6, 1.4))
    ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
    ax[i].set_xlabel(r'$\it{p}$' + "=%.03f" % p, fontsize=12)
    if i == 0:
        ax[i].set_title("%s\n%s only" % (time_interval_category, restrict_other_mode),
loc='left')
    else:
        ax[i].set_title("%s only" % restrict_other_mode)

plt.subplots_adjust(wspace=2)
plt.show()


# In[12]:


# Subsample QP pairs in a category to n=50 and get #sweeps/day, bootstrap 20 times
bootstrapped_rates_by_cat_tp_type = {cat:
bootstrapped_agg_from_list_dict(count_time_tups_by_cat_tp_type[cat],
```

```
                                                                                get_rate_from_tuples,
num_bootstraps=1000, n=100) for cat in cats}

# SNP change rates boxplot
fig, ax = plt.subplots(figsize=(14,4))

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1']
annotations = [] # These are the true "average" rates per category
tp_type_labels = [] # These are used as x-axis labels

bootstrapped_rates = []
for tp_type in tp_types:
    for cat in cats:
        bootstrapped_rates.append(bootstrapped_rates_by_cat_tp_type[cat][tp_type])

for tp_type in tp_types:
    for cat in cats:
        snp_count, total_days = (0,0)
        for count, days in count_time_tups_by_cat_tp_type[cat][tp_type]:
            snp_count += count
            total_days += days
        annotations.append(float(snp_count)/total_days)
        tp_type_labels.append('%s\nn=%i' % (tp_type,
len(count_time_tups_by_cat_tp_type[cat][tp_type])))

boxprops = dict(color='#77acff'); medianprops = dict(color='black'); flierprops =
dict(marker='.')
boxplots = ax.boxplot(bootstrapped_rates, patch_artist=True, medianprops=medianprops,
flierprops=flierprops)

for i in np.arange(len(bootstrapped_rates), step=2):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Vaginal
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # C-section

ax.plot(np.arange(1, 1+len(annotations)), annotations, marker=(5, 2), linestyle='None',
markersize=12, color='black', zorder=9)

# ax.set_ylim(0, 0.14)
ax.set_yscale('log')
ax.set_xticklabels(tp_type_labels, fontsize=12)
ax.set_ylabel("SNP changes per QP pair per day", fontsize=12)
ax.set_title("Bootstrapped (subsample QP pairs 50 times), include zeros")

legend_elements = [Patch(facecolor=plot_utils.col_blue, label='Vaginal'),
                   Patch(facecolor=plot_utils.col_orange, label='C-section')]
ax.legend(handles=legend_elements, loc='upper right', frameon=False)

plt.show()


# In[27]:


# Store for later use
snp_change_rates_devmode = bootstrapped_rates[2:6]
snp_change_rate_annotations_devmode = annotations[2:6]
snp_change_rate_tp_type_labels_devmode = tp_type_labels[2:6]


# In[140]:
```

```
# Compare feeding modes

cats = ['breast', 'mixed', 'formula']

replacement_time_tups_by_cat_tp_type = {cat: defaultdict(list) for cat in cats}
all_time_tups_by_cat_tp_type = {cat: defaultdict(list) for cat in cats}
subject_all_time_tups_by_tp_type = defaultdict(list) # For later permutation tests
count_time_tups_by_cat_tp_type = {cat: defaultdict(list) for cat in cats}
subject_count_time_tups_by_tp_type = defaultdict(list) # For later permutation tests

for species in snp_changes:
    for s1, s2 in snp_changes[species]:

        subject = sample_subject_map[s2]
        if subject not in subject_feeding_mode_map: # Skip non-infant second sample
            continue

        feeding_mode = subject_feeding_mode_map[subject]; cat = feeding_mode
        val = snp_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue

        # Arbitrarily set mother 0 - infant 0 as 1 day
        if is_mother(s1) and mi_sample_day_dict[s1] == 0 and is_infant(s2) and
mi_sample_day_dict[s2] == 0:
            days = 1

        if isinstance(val, int): # Replacement
            for custom_cohort in custom_cohorts:
                replacement_time_tups_by_cat_tp_type[cat][custom_cohort].append((1, days))

                all_time_tups_by_cat_tp_type[cat][custom_cohort].append((val, days))
                subject_all_time_tups_by_tp_type[custom_cohort].append((subject, val, days))

        else: # Not replacement (modifications/no change)
            for custom_cohort in custom_cohorts:

                count_time_tups_by_cat_tp_type[cat][custom_cohort].append((len(val), days))
                subject_count_time_tups_by_tp_type[custom_cohort].append((subject, len(val),
days))

                all_time_tups_by_cat_tp_type[cat][custom_cohort].append((len(val), days))
                subject_all_time_tups_by_tp_type[custom_cohort].append((subject, len(val),
days))

# Subsample QP pairs in a category to n=50 and get #sweeps/day, bootstrap 20 times
bootstrapped_rates_by_cat_tp_type = {cat:
bootstrapped_agg_from_list_dict(count_time_tups_by_cat_tp_type[cat], get_rate_from_tuples) for
cat in cats}

# SNP change rates boxplot
fig, ax = plt.subplots(figsize=(20,4))

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1']
annotations = [] # These are the true "average" rates per category
```

```
tp_type_labels = [] # These are used as x-axis labels

bootstrapped_rates = []
for tp_type in tp_types:
    for cat in cats:
        bootstrapped_rates.append(bootstrapped_rates_by_cat_tp_type[cat][tp_type])

for tp_type in tp_types:
    for cat in cats:
        snp_count, total_days = (0,0)
        for count, days in count_time_tups_by_cat_tp_type[cat][tp_type]:
            snp_count += count
            total_days += days
        annotations.append(float(snp_count)/total_days)
        tp_type_labels.append('%s\nn=%i' % (tp_type,
len(count_time_tups_by_cat_tp_type[cat][tp_type])))

ax.plot(np.arange(1, 1+len(annotations)), annotations, marker=(5, 2), linestyle='None',
markersize=12)

boxprops = dict(color='#77acff'); medianprops = dict(color='black'); flierprops =
dict(marker='.')
boxplots = ax.boxplot(bootstrapped_rates, patch_artist=True, medianprops=medianprops,
flierprops=flierprops)

for i in np.arange(len(bootstrapped_rates), step=3):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Breast
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # Mixed
    boxplots['boxes'][i+2].set_facecolor(plot_utils.col_darkgreen) # Formula

# ax.set_ylim(0, 0.14)
ax.set_yscale('log')
ax.set_xticklabels(tp_type_labels, fontsize=12)
ax.set_ylabel("SNP changes per QP pair per day", fontsize=12)
ax.set_title("Bootstrapped (subsample QP pairs 50 times), include zeros")

legend_elements = [Patch(facecolor=plot_utils.col_blue, label='Breast'),
                   Patch(facecolor=plot_utils.col_orange, label='Mixed'),
                   Patch(facecolor=plot_utils.col_darkgreen, label='Formula')]
ax.legend(handles=legend_elements, loc='upper right', frameon=False)

plt.show()


# In[97]:


# Permutation test for feeding mode
# 1. permute feeding mode (formula vs. breast, ignore mixed) for all the babies
# 2. compute the SNV change rate for new B and new F babies;
# make sure to take the sum of the changes divided by the total time
# 3. compute difference in rate
# 4. Repeat x1000.
# 5. Compare true difference between feeding modes

num_bootstraps = 1000
cats = ['formula', 'breast']
tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

subjects = list(subject_feeding_mode_map.keys()) # Order matters!!
feeding_modes = [] # True/original
```

```
for subject in subjects:
    feeding_modes.append(subject_feeding_mode_map[subject])

true_snv_change_rate_by_cat_tp_type_dict = {cat: {} for cat in cats}

for tp_type in tp_types:
    total_days_by_cat_dict = defaultdict(int)
    total_snv_change_count_by_cat_dict = defaultdict(int)
    for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
        feeding_mode = subject_feeding_mode_map[subject]
        total_days_by_cat_dict[feeding_mode] += days
        total_snv_change_count_by_cat_dict[feeding_mode] += count

    for cat in cats:
        days = float(total_days_by_cat_dict[cat])
        if days == 0:
            print("Can't use %s" % tp_type)
            continue
        snv_change_rate = total_snv_change_count_by_cat_dict[cat]/days
        true_snv_change_rate_by_cat_tp_type_dict[cat][tp_type] = snv_change_rate

permuted_snv_change_rates_by_cat_tp_type_dict = {cat: defaultdict(list) for cat in cats}

i = 0
while i < num_bootstraps:
    random.shuffle(feeding_modes) # Randomly permute feeding modes
    permuted_subject_feeding_mode_map = {}
    for subject, feeding_mode in zip(subjects, feeding_modes):
        permuted_subject_feeding_mode_map[subject] = feeding_mode

    bad = False # If end up with sample size 0 for any timepoint-cat pair, skip?? Not used for
now

    # Compute SNV change rate from subject_count_time_tups_by_tp_type
    for tp_type in subject_count_time_tups_by_tp_type:
        total_days_by_cat_dict = defaultdict(int)
        total_snv_change_count_by_cat_dict = defaultdict(int)
        for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
            feeding_mode = permuted_subject_feeding_mode_map[subject]
            total_days_by_cat_dict[feeding_mode] += days
            total_snv_change_count_by_cat_dict[feeding_mode] += count

        for cat in cats:
            days = float(total_days_by_cat_dict[cat])
            if days > 0:
                snv_change_rate = total_snv_change_count_by_cat_dict[cat]/days
            else:
                snv_change_rate = 0 # Consider it 0 if there are no samples/babies for this
timepoint-cat
            permuted_snv_change_rates_by_cat_tp_type_dict[cat][tp_type].append(snv_change_rate)

    i += 1

# Just interested in Week 0-Month 1 and Month 1-Year

fig, ax = plt.subplots(figsize=(4,4))

tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

null_differences_by_tp_type = []
labels = []
```

```
true_differences_by_tp_type = []

for tp_type in tp_types:
    null_differences = []
    for rate1, rate2 in zip(permuted_snv_change_rates_by_cat_tp_type_dict['breast'][tp_type],
                            permuted_snv_change_rates_by_cat_tp_type_dict['formula'][tp_type]):
        null_differences.append(rate1-rate2)
    null_differences_by_tp_type.append(null_differences)
    true_rate1 = true_snv_change_rate_by_cat_tp_type_dict['breast'][tp_type]
    true_rate2 = true_snv_change_rate_by_cat_tp_type_dict['formula'][tp_type]
    true_difference = true_rate1-true_rate2
    true_differences_by_tp_type.append(true_difference)

    p = permutation_test_p(null_differences, true_difference)
    labels.append(("%s\n" % tp_type) + r'$\it{p}$' + ("=%.03f" % (p)))

ax.violinplot(null_differences_by_tp_type, positions=[1, 2])
ax.plot(np.arange(1, 1+len(true_differences_by_tp_type)), true_differences_by_tp_type,
        marker=(5, 2), linestyle='None', markersize=12, color='black', zorder=9)
ax.set_xticks([1, 2])
ax.set_xticklabels(labels)
ax.set_ylabel("Difference between breast\nand formula feeding SNV change rate")
plt.show()
fig.savefig('%s/feeding_mode_comparison_snv_change_rate.pdf' % config.analysis_directory)


# In[156]:


# Permutation tests for delivery and feeding mode

fig, ax = plt.subplots(1, 2, figsize=(8, 4))

# ======================================================================
# Delivery mode
time_interval_categories = ['Day 0-Week 1', 'Week 1-Month 1']
mult_null_differences = []
mult_true_differences = []
plot_labels = []

label_set = ['Vaginal', 'C-section']

for time_interval_category in time_interval_categories:
    orig_label_subject_data_dict = defaultdict(list)
    for label in label_set:
        for subject, count, days in subject_count_time_tups_by_tp_type[time_interval_category]:
            if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                feeding_mode = subject_feeding_mode_map[subject]
                delivery_mode = subject_delivery_mode_map[subject]
                orig_label_subject_data_dict[delivery_mode].append((subject, (count, days)))

    null_differences, true_difference = permutation_test_subject(orig_label_subject_data_dict,
label_set,

aggregator_fn=get_rate_from_tuples)
    p = permutation_test_p(null_differences, true_difference)

    mult_null_differences.append(null_differences)
    mult_true_differences.append(true_difference)
    plot_labels.append(("%s\n" % time_interval_category) + r'$\it{p}$' + ("=%.03f" % (p)))
```

```
i = 0

xs = np.arange(1, 1+len(mult_true_differences))
ax[i].violinplot(mult_null_differences)
ax[i].plot(xs, mult_true_differences, marker=(5, 2),
           linestyle='None', markersize=12, color=color, zorder=9)

ax[i].set_xticks(xs)
ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
ax[i].set_xticklabels(plot_labels, fontsize=12)
# ========================================================================

# ========================================================================
# Feeding mode
time_interval_categories = ['Day 0-Week 1', 'Week 1-Month 1']
mult_null_differences = []
mult_true_differences = []
plot_labels = []

label_set = ['breast', 'formula']

for time_interval_category in time_interval_categories:
    orig_label_subject_data_dict = defaultdict(list)
    for label in label_set:
        for subject, count, days in subject_count_time_tups_by_tp_type[time_interval_category]:
            if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                feeding_mode = subject_feeding_mode_map[subject]
                delivery_mode = subject_delivery_mode_map[subject]
                orig_label_subject_data_dict[feeding_mode].append((subject, (count, days)))

    null_differences, true_difference = permutation_test_subject(orig_label_subject_data_dict,
label_set,

aggregator_fn=get_rate_from_tuples)
    p = permutation_test_p(null_differences, true_difference)

    mult_null_differences.append(null_differences)
    mult_true_differences.append(true_difference)
    plot_labels.append(("%s\n" % time_interval_category) + r'$\it{p}$' + ("=%.03f" % (p)))

i += 1

xs = np.arange(1, 1+len(mult_true_differences))
ax[i].violinplot(mult_null_differences)
ax[i].plot(xs, mult_true_differences, marker=(5, 2),
           linestyle='None', markersize=12, color=color, zorder=9)

ax[i].set_xticks(xs)
ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
ax[i].set_xticklabels(plot_labels, fontsize=12)

plt.subplots_adjust(wspace=0.5)
plt.show()


# # ==================
# # Replacement rates
# # ==================

# In[28]:
```

```
# Subsample QP pairs in a category to n=40 and get #sweeps/day, bootstrap x times
bootstrapped_rates_by_tp_type = bootstrapped_agg_from_list_dict(all_time_tups_by_tp_type,
get_replacement_rate_from_tuples,
                                                                num_bootstraps=1000, n=40)

# Replacement rates boxplot
fig, ax = plt.subplots(figsize=(10,4))

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']
annotations = [] # These are the true "average" rates per category
tp_type_labels = [] # These are used as x-axis labels

bootstrapped_rates = [bootstrapped_rates_by_tp_type[tp_type] for tp_type in tp_types]
for tp_type in tp_types:
    replacement_count, total_days = (0,0)
    for num_snp_changes, days in all_time_tups_by_tp_type[tp_type]:
        if num_snp_changes >= 500:
            replacement_count += 1
        total_days += days
    annotations.append(float(replacement_count)/total_days)
    tp_type_labels.append('%s\nn=%i' % (tp_type, len(all_time_tups_by_tp_type[tp_type])))

ax.set_yscale('log')
ax.plot(np.arange(1, 1+len(annotations)), annotations, marker=(5, 2), linestyle='None',
markersize=12)
ax.boxplot(bootstrapped_rates)
ax.set_xticklabels(tp_type_labels, fontsize=12)
ax.set_ylabel("Replacements per QP pair per day", fontsize=12)
ax.set_title("Bootstrapped (subsample QP pairs 50 times)")
for tp_type, annotation in zip(tp_types, annotations):
    print("%s: %s" % (tp_type, annotation))

plt.show()

# fig.savefig('%s/replacement_rates_by_tp_type_overall_with_zeros_labelled.pdf' % plot_dir,
bbox_inches='tight')


# In[29]:


# store for later use
replacement_rates = bootstrapped_rates
replacement_rate_annotations = annotations
replacement_rate_tp_type_labels = tp_type_labels

# Get statistics for paper
print(replacement_rate_annotations)


# In[49]:


# Matched
# replacement rates boxplot
# Uses bootstrapped_rates_by_tp_type from above with n=40, num_bootstraps=1000

fig, ax = plt.subplots(figsize=(4,4))

tp_types = ['4-8mon duration infant', '4-8mon duration adult']
```

```
labels = ['4-8mon duration\nInfant', '4-8mon duration\nAdult']
annotations = [] # These are the true "average" rates per category
tp_type_labels = [] # These are used as x-axis labels

bootstrapped_rates = [bootstrapped_rates_by_tp_type[tp_type] for tp_type in tp_types]
for tp_type, label in zip(tp_types, labels):
    replacement_count, total_days = (0,0)
    for num_snp_changes, days in all_time_tups_by_tp_type[tp_type]:
        if num_snp_changes >= 500:
            replacement_count += 1
        total_days += days
    annotations.append(float(replacement_count)/total_days)
    tp_type_labels.append('%s\nn=%i' % (label, len(all_time_tups_by_tp_type[tp_type])))

ax.plot(np.arange(1, 1+len(annotations)), annotations, marker=(5, 2), linestyle='None',
markersize=12)
ax.boxplot(bootstrapped_rates)
# ax.set_ylim(0, 0.14)
ax.set_xticklabels(tp_type_labels, fontsize=12)
ax.set_ylabel("Replacements per day", fontsize=12)
# ax.set_title("Bootstrapped (subsample QP pairs 50 times), include zeros")

t, p = stats.ttest_ind(bootstrapped_rates[0], bootstrapped_rates[1])
print("T: %.02f\tp: %.3E" % (t, p))
plt.show()


# In[36]:


# Permutation test for infant vs. adult
label_set = ['4-8mon duration infant', '4-8mon duration adult']
orig_label_data_dict = {label: all_time_tups_by_tp_type[label] for label in label_set}
null_differences, true_difference = permutation_test(orig_label_data_dict, label_set,

aggregator_fn=get_replacement_rate_from_tuples,
                                                     num_bootstraps=10000)

# Plot permutation test results
fig, ax = plt.subplots(figsize=(2.3, 4))
p = permutation_test_p(null_differences, true_difference)
color = 'red' if p < 0.05 else 'gray'
ax.violinplot(null_differences)
ax.plot([1], [true_difference], marker=(5, 2), linestyle='None', markersize=12, color=color,
zorder=9)
ax.set_xticks([1]); ax.set_xticklabels([]); ax.set_xlim((0.6, 1.4))
ax.set_ylabel("Difference (Infant-Adult)", fontsize=12)
ax.set_xlabel(r'$\it{p}$' + ("<0.0001" if p == 0 else "=%.03f" % p), fontsize=12)
plt.tight_layout()
fig.savefig('%s/infant_vs_adult_comparison_replacement_rate.pdf' % config.analysis_directory)


# In[155]:


# Perform PERMUTATION tests between all pairs

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

replacement_rate_tp_pair_ptests = {} # (tp1, tp2) -> n1, n2, p
```

```
print("REPLACEMENT RATE TIMEPOINT PAIRWISE COMPARISONS")

for i in range(len(tp_types)):
    tp_type1 = tp_types[i]
    for j in range(i+1, len(tp_types)):
        tp_type2 = tp_types[j]
        # Permutation test for tp_type 1 vs._tp type2
        label_set = [tp_type1, tp_type2]
        orig_label_data_dict = {label: all_time_tups_by_tp_type[label] for label in label_set}
        null_differences, true_difference = permutation_test(orig_label_data_dict, label_set,

aggregator_fn=get_replacement_rate_from_tuples,
                                                             num_bootstraps=10000)
        p = permutation_test_p(null_differences, true_difference)
        n1 = len(orig_label_data_dict[tp_type1])
        n2 = len(orig_label_data_dict[tp_type2])
        replacement_rate_tp_pair_ptests[(tp_type1, tp_type2)] = (n1, n2, p)
        print("==============================")
        print("%s vs. %s" % (tp_type1, tp_type2))
        print(str(p) + '\t' + get_sig_str(p))


# In[162]:


# Perform tests between all pairs

replacement_rate_tp_pair_tests = {} # (tp1, tp2) -> (t, p, es) Store for later table

print("REPLACEMENT RATE TIMEPOINT PAIRWISE COMPARISONS")
print("Note: all sample sizes are the same")

for i in range(len(tp_types)):
    ind_rates1 = bootstrapped_rates[i]; tp_type1 = tp_types[i]
    for j in range(i+1, len(tp_types)):
        ind_rates2 = bootstrapped_rates[j]; tp_type2 = tp_types[j]
        print("==============================")
        print("%s vs. %s" % (tp_type1, tp_type2))
        stat, p, es = summarize_utest(ind_rates1, ind_rates2, simple=True)
        replacement_rate_tp_pair_tests[(tp_type1, tp_type2)] = (stat, p, es)
        print(get_sig_str(p))


# In[81]:


# Replacement rates by cohort boxplot
fig, ax = plt.subplots(4, 1, figsize=(6, 12))

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

for i in range(4):

    dataset = cohorts[i]
    # Subsample QP pairs in a category to n=50 and get #sweeps/day, bootstrap 20 times
    bootstrapped_rates_by_tp_type =
bootstrapped_agg_from_list_dict(all_time_tups_by_cohort_tp_type[dataset],
get_replacement_rate_from_tuples, n=20)

    annotations = [] # These are the true "average" rates per category
    tp_type_labels = [] # These are used as x-axis labels
```

```
    bootstrapped_rates = [] # Data

    for tp_type in tp_types:
        if len(bootstrapped_rates_by_tp_type[tp_type]) < 10:
            continue
        replacement_count, total_days = (0,0)
        for num_snp_changes, days in all_time_tups_by_cohort_tp_type[dataset][tp_type]:
            if num_snp_changes >= 500:
                replacement_count += 1
            total_days += days
        bootstrapped_rates.append(bootstrapped_rates_by_tp_type[tp_type])
        annotations.append(float(replacement_count)/total_days)
        tp_type_labels.append('%s\nn=%i' % (tp_type,
len(all_time_tups_by_cohort_tp_type[dataset][tp_type])))

    ax[i].plot(np.arange(1, 1+len(annotations)), annotations, marker=(5, 2), linestyle='None',
markersize=12)
    ax[i].boxplot(bootstrapped_rates)
    # ax.set_ylim(0, 0.14)
    ax[i].set_yscale('log')
    ax[i].set_xticklabels(tp_type_labels, fontsize=12)
    ax[i].set_ylabel("Replacements per QP pair\nper day", fontsize=12)
    ax[i].set_title(dataset)
    # ax[i].set_title("Bootstrapped (subsample QP pairs 50 times), include zeros")

plt.tight_layout()
plt.show()


# In[137]:


all_time_tups_by_devmode_tp_type = all_time_tups_by_cat_tp_type
devmodes = ['Vaginal', 'C-section']

# Subsample QP pairs in a category and get #sweeps/day, bootstrap 1000 times
bootstrapped_rates_by_devmode_tp_type = {dm:
bootstrapped_agg_from_list_dict(all_time_tups_by_devmode_tp_type[dm],

get_replacement_rate_from_tuples,
                                                                        num_bootstraps=1000,
n=100) for dm in devmodes}

# Replacement rates boxplot
fig, ax = plt.subplots(figsize=(14,4))

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1']
annotations = [] # These are the true "average" rates per category
tp_type_labels = [] # These are used as x-axis labels

bootstrapped_rates = []
for tp_type in tp_types:
    for dm in devmodes:
        bootstrapped_rates.append(bootstrapped_rates_by_devmode_tp_type[dm][tp_type])

for tp_type in tp_types:
    for dm in devmodes:
        replacement_count, total_days = (0,0)
        for num_snp_changes, days in all_time_tups_by_devmode_tp_type[dm][tp_type]:
            if num_snp_changes >= 500:
                replacement_count += 1
```

```
            total_days += days
        annotations.append(float(replacement_count)/total_days)
        tp_type_labels.append('%s\nn=%i' % (tp_type,
len(all_time_tups_by_devmode_tp_type[dm][tp_type])))

ax.plot(np.arange(1, 1+len(annotations)), annotations, marker=(5, 2), linestyle='None',
markersize=12)
ax.boxplot(bootstrapped_rates)
ax.set_xticklabels(tp_type_labels, fontsize=12)
ax.set_yscale('log')
ax.set_ylabel("Replacements per QP pair per day", fontsize=12)
ax.set_title("Bootstrapped (subsample QP pairs 50 times)")

plt.show()


# In[32]:


# Store for later use
replacement_rates_devmode = bootstrapped_rates[2:6]
replacement_rate_annotations_devmode = annotations[2:6]
replacement_rate_tp_type_labels_devmode = tp_type_labels[2:6]


# In[135]:


# Summarize distribution by delivery, feeding mode
time_interval_category = 'Week 1-Month 1'
feeding_delivery_count_dict = {fmode: defaultdict(int) for fmode in ['breast', 'formula',
'mixed']}
for subject, count, days in subject_all_time_tups_by_tp_type[time_interval_category]:
    if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
        feeding_mode = subject_feeding_mode_map[subject]
        delivery_mode = subject_delivery_mode_map[subject]
        feeding_delivery_count_dict[feeding_mode][delivery_mode] += 1

dmodes = ['Vaginal', 'C-section']
fmodes = ['breast', 'formula', 'mixed']
print("SNV changes | %s" % time_interval_category)
print('----------------------')
print('\t' + '\t'.join(dmodes))
for fmode in fmodes:
    vals = [str(feeding_delivery_count_dict[fmode][dmode]) for dmode in dmodes]
    print("%s\t" % (fmode) + '\t'.join(vals))


# In[188]:


# Reproduce in the more elegant way heh
# Permutation tests for delivery and feeding mode

fig, ax = plt.subplots(1, 2, figsize=(8, 4))

time_interval_categories = ['Day 0-Week 1', 'Week 1-Month 1']

# Vaginal vs. c-section
label_sets = [['Vaginal', 'C-section'], ['breast', 'formula']]
label_mode_type_list = [0, 1] # 0 for delivery mode, 1 for feeding mode
```

```
for i in range(2):
    label_set = label_sets[i] # Which two labels we are comparing
    label_mode_type = label_mode_type_list[i]

    null_differences_list = []
    true_difference_list = []
    labels = []
    for time_interval_category in time_interval_categories:
        orig_label_subject_data_dict = defaultdict(list)
        for label in label_set:
            for subject, count, days in
subject_all_time_tups_by_tp_type[time_interval_category]:
                if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                    feeding_mode = subject_feeding_mode_map[subject]
                    delivery_mode = subject_delivery_mode_map[subject]
                    if label_mode_type == 0:
                        orig_label_subject_data_dict[delivery_mode].append((subject, (count,
days)))
                    if label_mode_type == 1:
                        orig_label_subject_data_dict[feeding_mode].append((subject, (count,
days)))

        null_differences, true_difference =
permutation_test_subject(orig_label_subject_data_dict, label_set,

aggregator_fn=get_replacement_rate_from_tuples, num_bootstraps=10000)
        null_differences_list.append(null_differences)
        true_difference_list.append(true_difference)
        p = permutation_test_p(null_differences, true_difference)
        color = 'red' if p < 0.05 else 'black'
        labels.append('%s\n' % time_interval_category + r'$\it{p}$' + "=%.03f" % p)

    ax[i].violinplot(null_differences_list)
    ax[i].plot([1, 2], true_difference_list, marker=(5, 2), linestyle='None', markersize=12,
color=color, zorder=9)
    ax[i].set_xticks([1, 2])
    ax[i].set_xticklabels(labels)
    ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
    # ax[i].set_xlabel(r'$\it{p}$' + "=%.03f" % p, fontsize=12)
    ax[i].set_title("Replacement rate")

plt.subplots_adjust(wspace=0.5)
plt.show()
fig.savefig('%s/mode_comparisons_replacement_rate.pdf' % config.analysis_directory)
fig.savefig('%s/mode_comparisons_replacement_rate.png' % config.analysis_directory, dpi=200)


# In[98]:


# Permutation tests for delivery and feeding mode

fig, ax = plt.subplots(1, 4, figsize=(8, 4))

time_interval_category = 'Week 1-Month 1'

label_sets = [['Vaginal', 'C-section'], ['Vaginal', 'C-section'], ['breast', 'formula'],
['breast', 'formula']]
label_mode_type_list = [0, 0, 1, 1] # 0 for delivery mode, 1 for feeding mode
restrict_other_mode_list = ['breast', 'formula', 'Vaginal', 'C-section']
```

```
for i in range(4):
    label_set = label_sets[i] # Which two labels we are comparing
    label_mode_type = label_mode_type_list[i]
    restrict_other_mode = restrict_other_mode_list[i]

    orig_label_subject_data_dict = defaultdict(list)
    for label in label_set:
        for subject, count, days in subject_all_time_tups_by_tp_type[time_interval_category]:
            if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                feeding_mode = subject_feeding_mode_map[subject]
                delivery_mode = subject_delivery_mode_map[subject]
                if label_mode_type == 0:
                    if feeding_mode == restrict_other_mode:
                        orig_label_subject_data_dict[delivery_mode].append((subject, (count,
days)))
                if label_mode_type == 1:
                    if delivery_mode == restrict_other_mode:
                        orig_label_subject_data_dict[feeding_mode].append((subject, (count,
days)))

    null_differences, true_difference = permutation_test_subject(orig_label_subject_data_dict,
label_set,

aggregator_fn=get_replacement_rate_from_tuples)

    p = permutation_test_p(null_differences, true_difference)
    color = 'red' if p < 0.05 else 'black'
    ax[i].violinplot(null_differences)
    ax[i].plot([1], [true_difference], marker=(5, 2), linestyle='None', markersize=12,
color=color, zorder=9)
    ax[i].set_xticks([1]); ax[i].set_xticklabels([]); ax[i].set_xlim((0.6, 1.4))
    ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
    ax[i].set_xlabel(r'$\it{p}$' + ("<0.0001" if p == 0 else "=%.03f" % p), fontsize=12)
    if i == 0:
        ax[i].set_title("%s\n%s only" % (time_interval_category, restrict_other_mode),
loc='left')
    else:
        ax[i].set_title("%s only" % restrict_other_mode)

plt.subplots_adjust(wspace=2)
plt.show()


# In[141]:


# Compare feeding modes

cats = ['breast', 'mixed', 'formula']

# Subsample QP pairs in a category to n=50 and get #sweeps/day, bootstrap 20 times
bootstrapped_rates_by_cat_tp_type = {cat:
bootstrapped_agg_from_list_dict(all_time_tups_by_cat_tp_type[cat],
get_replacement_rate_from_tuples) for cat in cats}

# Replacement rates boxplot
fig, ax = plt.subplots(figsize=(20,4))

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1']
annotations = [] # These are the true "average" rates per category
```

```
tp_type_labels = [] # These are used as x-axis labels

bootstrapped_rates = []
for tp_type in tp_types:
    for cat in cats:
        bootstrapped_rates.append(bootstrapped_rates_by_cat_tp_type[cat][tp_type])

for tp_type in tp_types:
    for cat in cats:
        replacement_count, total_days = (0,0)
        for num_snp_changes, days in all_time_tups_by_cat_tp_type[cat][tp_type]:
            if num_snp_changes >= 500:
                replacement_count += 1
            total_days += days
        annotations.append(float(replacement_count)/total_days)
        tp_type_labels.append('%s\nn=%i' % (tp_type,
len(all_time_tups_by_cat_tp_type[cat][tp_type])))

ax.plot(np.arange(1, 1+len(annotations)), annotations, marker=(5, 2), linestyle='None',
markersize=12)

boxprops = dict(color='#77acff'); medianprops = dict(color='black'); flierprops =
dict(marker='.')
boxplots = ax.boxplot(bootstrapped_rates, patch_artist=True, medianprops=medianprops,
flierprops=flierprops)

for i in np.arange(len(bootstrapped_rates), step=3):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Breast
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # Mixed
    boxplots['boxes'][i+2].set_facecolor(plot_utils.col_darkgreen) # Formula

# ax.set_ylim(0, 0.14)
ax.set_yscale('log')
ax.set_xticklabels(tp_type_labels, fontsize=12)
ax.set_ylabel("Replacements per QP pair per day", fontsize=12)
ax.set_title("Bootstrapped (subsample QP pairs 50 times), include zeros")

legend_elements = [Patch(facecolor=plot_utils.col_blue, label='Breast'),
                   Patch(facecolor=plot_utils.col_orange, label='Mixed'),
                   Patch(facecolor=plot_utils.col_darkgreen, label='Formula')]
ax.legend(handles=legend_elements, loc='upper right', frameon=False)

plt.show()


# In[142]:


# Permutation test for feeding mode
# 1. permute feeding mode (formula vs. breast, ignore mixed) for all the babies
# 2. compute the SNV change rate for new B and new F babies;
# make sure to take the sum of the changes divided by the total time
# 3. compute difference in rate
# 4. Repeat x1000.
# 5. Compare true difference between feeding modes

num_bootstraps = 1000
cats = ['formula', 'breast']
tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

subjects = list(subject_feeding_mode_map.keys()) # Order matters!!
```

```
feeding_modes = [] # True/original
for subject in subjects:
    feeding_modes.append(subject_feeding_mode_map[subject])

true_replacement_rate_by_cat_tp_type_dict = {cat: {} for cat in cats}

for tp_type in tp_types:
    total_days_by_cat_dict = defaultdict(int)
    total_replacement_count_by_cat_dict = defaultdict(int)
    for subject, count, days in subject_all_time_tups_by_tp_type[tp_type]:
        feeding_mode = subject_feeding_mode_map[subject]
        total_days_by_cat_dict[feeding_mode] += days
        if count >= 500:
            total_replacement_count_by_cat_dict[feeding_mode] += 1

    for cat in cats:
        days = float(total_days_by_cat_dict[cat])
        replacement_rate = total_replacement_count_by_cat_dict[cat]/days
        true_replacement_rate_by_cat_tp_type_dict[cat][tp_type] = replacement_rate

permuted_replacement_rates_by_cat_tp_type_dict = {cat: defaultdict(list) for cat in cats}

i = 0
while i < num_bootstraps:
    random.shuffle(feeding_modes) # Randomly permute feeding modes
    permuted_subject_feeding_mode_map = {}
    for subject, feeding_mode in zip(subjects, feeding_modes):
        permuted_subject_feeding_mode_map[subject] = feeding_mode

    bad = False # If end up with sample size 0 for any timepoint-cat pair, skip?? Not used for
now

    # Compute SNV change rate from subject_count_time_tups_by_tp_type
    for tp_type in subject_count_time_tups_by_tp_type:
        total_days_by_cat_dict = defaultdict(int)
        total_replacement_count_by_cat_dict = defaultdict(int)
        for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
            feeding_mode = permuted_subject_feeding_mode_map[subject]
            total_days_by_cat_dict[feeding_mode] += days
            total_replacement_count_by_cat_dict[feeding_mode] += count

        for cat in cats:
            days = float(total_days_by_cat_dict[cat])
            if days > 0:
                replacement_rate = total_replacement_count_by_cat_dict[cat]/days
            else:
                replacement_rate = 0 # Consider it 0 if there are no samples/babies for this
timepoint-cat

permuted_replacement_rates_by_cat_tp_type_dict[cat][tp_type].append(replacement_rate)

    i += 1

# Just interested in Week 0-Month 1 and Month 1-Year

fig, ax = plt.subplots(figsize=(4,4))

tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

null_differences_by_tp_type = []
labels = []
```

```
true_differences_by_tp_type = []

for tp_type in tp_types:
    null_differences = []
    for rate1, rate2 in zip(permuted_replacement_rates_by_cat_tp_type_dict['breast'][tp_type],
                            permuted_replacement_rates_by_cat_tp_type_dict['formula'][tp_type]):
        null_differences.append(rate1-rate2)
    null_differences_by_tp_type.append(null_differences)
    true_rate1 = true_replacement_rate_by_cat_tp_type_dict['breast'][tp_type]
    true_rate2 = true_replacement_rate_by_cat_tp_type_dict['formula'][tp_type]
    true_difference = true_rate1-true_rate2
    true_differences_by_tp_type.append(true_difference)

    p = permutation_test_p(null_differences, true_difference)
    labels.append(("%s\n" % tp_type) + r'$\it{p}$' + ("=%.03f" % (p)))

ax.violinplot(null_differences_by_tp_type, positions=[1, 2])
ax.plot(np.arange(1, 1+len(true_differences_by_tp_type)), true_differences_by_tp_type,
        marker=(5, 2), linestyle='None', markersize=12, color='black', zorder=9)
ax.set_xticks([1, 2])
ax.set_xticklabels(labels)
ax.set_ylabel("Difference between breast\nand formula feeding replacement rate")
plt.show()
fig.savefig('%s/feeding_mode_comparison_replacement_rate.pdf' % config.analysis_directory)


# In[154]:


# Permutation tests for delivery and feeding mode

fig, ax = plt.subplots(1, 2, figsize=(8, 4))

# ========================================================================
# Delivery mode
time_interval_categories = ['Day 0-Week 1', 'Week 1-Month 1']
mult_null_differences = []
mult_true_differences = []
plot_labels = []

label_set = ['Vaginal', 'C-section']

for time_interval_category in time_interval_categories:
    orig_label_subject_data_dict = defaultdict(list)
    for label in label_set:
        for subject, count, days in subject_all_time_tups_by_tp_type[time_interval_category]:
            if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                feeding_mode = subject_feeding_mode_map[subject]
                delivery_mode = subject_delivery_mode_map[subject]
                orig_label_subject_data_dict[delivery_mode].append((subject, (count, days)))

    null_differences, true_difference = permutation_test_subject(orig_label_subject_data_dict,
label_set,

aggregator_fn=get_replacement_rate_from_tuples)
    p = permutation_test_p(null_differences, true_difference)

    mult_null_differences.append(null_differences)
    mult_true_differences.append(true_difference)
    plot_labels.append(("%s\n" % time_interval_category) + r'$\it{p}$' + ("=%.03f" % (p)))
```

```
i = 0

xs = np.arange(1, 1+len(mult_true_differences))
ax[i].violinplot(mult_null_differences)
ax[i].plot(xs, mult_true_differences, marker=(5, 2),
           linestyle='None', markersize=12, color=color, zorder=9)

ax[i].set_xticks(xs)
ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
ax[i].set_xticklabels(plot_labels, fontsize=12)
# ========================================================================

# ========================================================================
# Feeding mode
time_interval_categories = ['Day 0-Week 1', 'Week 1-Month 1']
mult_null_differences = []
mult_true_differences = []
plot_labels = []

label_set = ['breast', 'formula']

for time_interval_category in time_interval_categories:
    orig_label_subject_data_dict = defaultdict(list)
    for label in label_set:
        for subject, count, days in subject_all_time_tups_by_tp_type[time_interval_category]:
            if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                feeding_mode = subject_feeding_mode_map[subject]
                delivery_mode = subject_delivery_mode_map[subject]
                orig_label_subject_data_dict[feeding_mode].append((subject, (count, days)))

    null_differences, true_difference = permutation_test_subject(orig_label_subject_data_dict,
label_set,

aggregator_fn=get_replacement_rate_from_tuples)
    p = permutation_test_p(null_differences, true_difference)

    mult_null_differences.append(null_differences)
    mult_true_differences.append(true_difference)
    plot_labels.append(("%s\n" % time_interval_category) + r'$\it{p}$' + ("=%.03f" % (p)))

i += 1

xs = np.arange(1, 1+len(mult_true_differences))
ax[i].violinplot(mult_null_differences)
ax[i].plot(xs, mult_true_differences, marker=(5, 2),
           linestyle='None', markersize=12, color=color, zorder=9)

ax[i].set_xticks(xs)
ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
ax[i].set_xticklabels(plot_labels, fontsize=12)

plt.subplots_adjust(wspace=0.5)
plt.show()


# In[ ]:


# Change to violin plot
# Add p value in the xlabel
# Combine feeding and delivery mode into one huge plot
```

```
# In[ ]:


# Get rank of given value in given list
# where rank ranges from 0 (<= min(list)) to len()


# In[65]:


# Closer look at replacement rates data

num_replacement = 0
total = 0

for num_snp_diffs, days in all_time_tups_by_devmode_tp_type['Vaginal']['Month 1-Year 1']:
    total += 1
    if num_snp_diffs > 20:
        num_replacement += 1

num_replacement_csection = 0
total_csection = 0

for num_snp_diffs, days in all_time_tups_by_devmode_tp_type['C-section']['Month 1-Year 1']:
    total_csection += 1
    if num_snp_diffs > 20:
        num_replacement_csection += 1

print("%i/%i vaginal QP pairs are replacements" % (num_replacement, total))
print("%i/%i C-section QP pairs are replacements" % (num_replacement_csection, total_csection))
print()
for num_snp_diffs, days, subject in all_time_tups_with_subject_by_devmode_tp_type['C-
section']['Month 1-Year 1']:
    if num_snp_diffs > 20:
        print((num_snp_diffs, days, subject))
print('======')
for num_snp_diffs, days, subject in all_time_tups_with_subject_by_devmode_tp_type['C-
section']['Month 1-Year 1']:
    if num_snp_diffs <= 20:
        print((num_snp_diffs, days, subject))


# In[53]:


all_time_tups_with_subject_by_devmode_tp_type = {dm: defaultdict(list) for dm in devmodes}

for species in snp_changes:
    for s1, s2 in snp_changes[species]:

        subject = sample_subject_map[s2]
        if subject not in subject_delivery_mode_map: # Skip non-infant second sample
            continue

        delivery_mode = subject_delivery_mode_map[subject]
        val = snp_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)
```

```
        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue

        # Arbitrarily set mother 0 - infant 0 as 1 day
        if is_mother(s1) and mi_sample_day_dict[s1] == 0 and is_infant(s2) and
mi_sample_day_dict[s2] == 0:
            days = 1

        if isinstance(val, int): # Replacement
            for custom_cohort in custom_cohorts:

all_time_tups_with_subject_by_devmode_tp_type[delivery_mode][custom_cohort].append((val, days,
subject))

        else: # Not replacement (modifications/no change)
            for custom_cohort in custom_cohorts:

all_time_tups_with_subject_by_devmode_tp_type[delivery_mode][custom_cohort].append((len(val),
days, subject))


# In[52]:


print(num_replacement)
print(total)
print(num_replacement_csection)
print(total_csection)


# In[90]:


# Permutation test for delivery mode
# 1. permute C and V for all the babies
# 2. compute the SNV change rate for new C and new V babies;
# make sure to take the sum of the changes divided by the total time
# 3. compute difference in rate
# 4. Repeat x1000.
# 5. Compare true difference between C and V

num_bootstraps = 1000
cats = ['Vaginal', 'C-section']

subjects = list(subject_delivery_mode_map.keys()) # Order matters!!
delivery_modes = [] # True/original
for subject in subjects:
    delivery_modes.append(subject_delivery_mode_map[subject])

true_replacement_rate_by_cat_tp_type_dict = {cat: {} for cat in cats}

for tp_type in subject_all_time_tups_by_tp_type:
    total_days_by_cat_dict = defaultdict(int)
    total_replacement_count_by_cat_dict = defaultdict(int)
    for subject, count, days in subject_all_time_tups_by_tp_type[tp_type]:
        delivery_mode = subject_delivery_mode_map[subject]
        total_days_by_cat_dict[delivery_mode] += days
        if count >= 500:
            total_replacement_count_by_cat_dict[delivery_mode] += 1
```

```
    for cat in cats:
        days = float(total_days_by_cat_dict[cat])
        replacement_rate = total_replacement_count_by_cat_dict[cat]/days
        true_replacement_rate_by_cat_tp_type_dict[cat][tp_type] = replacement_rate

permuted_replacement_rates_by_cat_tp_type_dict = {cat: defaultdict(list) for cat in cats}

i = 0
while i < num_bootstraps:
    random.shuffle(delivery_modes) # Randomly permute delivery modes
    permuted_subject_delivery_mode_map = {}
    for subject, delivery_mode in zip(subjects, delivery_modes):
        permuted_subject_delivery_mode_map[subject] = delivery_mode

    bad = False # If end up with sample size 0 for any timepoint-cat pair, skip?? Not used for
now

    # Compute SNV change rate from subject_count_time_tups_by_tp_type
    for tp_type in subject_count_time_tups_by_tp_type:
        total_days_by_cat_dict = defaultdict(int)
        total_replacement_count_by_cat_dict = defaultdict(int)
        for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
            delivery_mode = permuted_subject_delivery_mode_map[subject]
            total_days_by_cat_dict[delivery_mode] += days
            total_replacement_count_by_cat_dict[delivery_mode] += count

        for cat in cats:
            days = float(total_days_by_cat_dict[cat])
            if days > 0:
                replacement_rate = total_replacement_count_by_cat_dict[cat]/days
            else:
                replacement_rate = 0 # Consider it 0 if there are no samples/babies for this
timepoint-cat

permuted_replacement_rates_by_cat_tp_type_dict[cat][tp_type].append(replacement_rate)

    i += 1

# Just interested in Week 0-Month 1 and Month 1-Year

fig, ax = plt.subplots(figsize=(4, 4))

tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

null_differences_by_tp_type = []
labels = []
true_differences_by_tp_type = []

for tp_type in tp_types:
    null_differences = []
    for vrate, crate in zip(permuted_replacement_rates_by_cat_tp_type_dict['Vaginal'][tp_type],
                            permuted_replacement_rates_by_cat_tp_type_dict['C-
section'][tp_type]):
        null_differences.append(vrate-crate)
    null_differences_by_tp_type.append(null_differences)

    true_vrate = true_replacement_rate_by_cat_tp_type_dict['Vaginal'][tp_type]
    true_crate = true_replacement_rate_by_cat_tp_type_dict['C-section'][tp_type]
    true_difference = true_vrate-true_crate
    true_differences_by_tp_type.append(true_difference)
```

```
    p = permutation_test_p(null_differences, true_difference)
    labels.append(("%s\n" % tp_type) + r'$\it{p}$' + ("=%.03f" % (p)))

ax.violinplot(null_differences_by_tp_type, positions=[1, 2])
ax.plot(np.arange(1, 1+len(true_differences_by_tp_type)), true_differences_by_tp_type,
        marker=(5, 2), linestyle='None', markersize=12, color='black', zorder=9)
ax.set_xticks([1, 2])
ax.set_xticklabels(labels)
ax.set_ylabel("Difference between vaginal\nand C-section replacement rate")
plt.show()
fig.savefig('%s/delivery_mode_comparison_replacement_rate.pdf' % config.analysis_directory)


# # ================
# # Gene change rates
# # ================

# In[21]:


# Store gene change information

gain_tups_by_tp_type = defaultdict(list)
loss_tups_by_tp_type = defaultdict(list)
genes_gained_by_tp_type = defaultdict(set)

subject_gain_tups_by_tp_type = defaultdict(list)
subject_loss_tups_by_tp_type = defaultdict(list)

# infant subject -> (species, gene) genes gained
genes_gained_m1y1 = defaultdict(list)

bad = []

for species in gene_changes:
    for s1, s2 in gene_changes[species]:

        subject = sample_subject_map[s2]
        gains, losses = gene_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        # SPECIAL: try recoding mother-infant duration as sum
        if 'Mother-Infant' in custom_cohorts:
            day1 = mi_sample_day_dict[s1]
            day2 = mi_sample_day_dict[s2]
            days = day1 + day2
            if days == 0:
                days = 1

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue
        '''
        # Arbitrarily set mother 0 - infant 0 as 1 day
        if is_mother(s1) and mi_sample_day_dict[s1] == 0 and is_infant(s2) and
mi_sample_day_dict[s2] == 0:
            days = 1
        '''
```

```
        if isinstance(snp_changes[species][(s1, s2)], int): # Replacement (in terms of SNP
changes)
            continue# num_gains, num_losses = gene_changes[species][(s1, s2)]
        else:
            num_gains = len(gene_changes[species][(s1, s2)][0])
            num_losses = len(gene_changes[species][(s1, s2)][1])

        # Try excluding the Mother-Infant(wk1) QP pair with unsually high number of losses
        if (num_losses >= 13) and ('Mother-Infant' in custom_cohorts):
            print("Sample pair %s, %s | %s" % (s1, s2, species))
            print("Num losses: %i | Num gains: %i" % (num_losses, num_gains))
            continue

        for custom_cohort in custom_cohorts:

            gain_tups_by_tp_type[custom_cohort].append((num_gains, days))
            subject_gain_tups_by_tp_type[custom_cohort].append((subject, num_gains, days))

            loss_tups_by_tp_type[custom_cohort].append((num_losses, days))
            subject_loss_tups_by_tp_type[custom_cohort].append((subject, num_losses, days))

            for gene, _, _, _, _ in gains:
                genes_gained_by_tp_type[custom_cohort].add((species, gene))

        if 'Month 1-Year 1' in custom_cohorts:
            if sample_subject_map[s1] != sample_subject_map[s2]:
                print("What??")
            infant_subject = sample_subject_map[s1]
            for gene, _, _, _, _ in gains:
                genes_gained_m1y1[infant_subject].append((species, gene))


# In[73]:


# Investigate the Mother-Infant category
# Namely, how much does Backhed higher resolution timepoint assignment change results?
# Are rates being inflated due to a smaller denominator due to mother timepoints
# being mistakenly considered 0 when they are actually not 0?
total_days = 0
n = 0
day_tups = []

for species in gene_changes:
    for s1, s2 in gene_changes[species]:

        subject = sample_subject_map[s2]
        gains, losses = gene_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        if 'Mother-Infant' in custom_cohorts:
            day1 = mi_sample_day_dict[s1]
            day2 = mi_sample_day_dict[s2]
            # print('Mother %i\tInfant %i' % (day1, day2))
            day_tups.append((day1, day2))
            if days >= 0:
                total_days += days
                n += 1

fig, ax = plt.subplots()
```

```
idx = 0
for day1, day2 in sorted(day_tups):
    ax.plot([day1, day2], [idx, idx], '.-')
    idx += 1
ax.set_xlabel("Day after birth\nFirst timepoint is mother, second is infant")
plt.show()


# In[97]:


genes_gained_by_tp_type.keys()


# In[69]:


# Check right away: Are the gene gains observed between 1 month and 1 year in infant observed in
the mother?

tp_gene_presabs_dict = defaultdict(dict) # tp -> gene -> array of 0s and 1s

import bz2
for tp in genes_gained_by_tp_type:
    print(tp)
    all_species = set([species for species, gene in genes_gained_by_tp_type[tp]])
    for species in all_species:
        f = bz2.open('%s/genes/%s/genes_presabs.txt.bz2' % (config.data_directory, species),
'rt')
        header = f.readline().strip().split('\t') # Includes gene_id first column and with-c
sample names
        mother_sample_idx_name_tups = [(header.index(s), s) for s in mother_samples if (s in
header or (s+'c') in header)]
        # print(len(mother_sample_idx_name_tups))
        for line in f:
            items = line.strip().split('\t')
            gene = items[0]
            if (species, gene) in genes_gained_by_tp_type[tp]:
                tp_gene_presabs_dict[tp][gene] = [int(items[i]) for i, sample in
mother_sample_idx_name_tups]


# In[ ]:


num_total = len(mother_samples)
num_gene_pres_list = []

for gene in mother_gene_presabs_dict:
    num_species = len(mother_gene_presabs_dict[gene])
    num_gene_pres = sum(mother_gene_presabs_dict[gene]);
num_gene_pres_list.append(num_gene_pres)
    print('%s:\tpresent in %i/%i mothers with species' % (gene, num_gene_pres, num_species))


# In[ ]:


fig, ax = plt.subplots(5, 1, figsize=(20, 12), sharex=True)
tps = ['Mother-Infant(earliest)', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-
Adult']
```

```
i=0
for tp in tps:
    num_gene_pres_list = [sum(tp_gene_presabs_dict[tp][gene]) for gene in
tp_gene_presabs_dict[tp]]
    xs = np.arange(len(num_gene_pres_list))
    ax[i].set_title(tp)
    ax[i].bar(xs, height=num_gene_pres_list)
    ax[i].set_ylabel("Number of genes")
    i += 1
ax[0].set_title("Presence of genes gained in infant among %i mothers\n%s" %
(len(mother_samples), tps[0]))
ax[4].set_xlabel("Number of mothers")


# In[42]:


# Ok change to: Are the gene gains observed between 1 month and 1 year in infant observed in the
same infant's mother?

subject_gene_presabs_dict = defaultdict(dict) # infant subject -> gene -> True/False

import bz2
for infant_subject in genes_gained_m1y1:
    genes_gained = genes_gained_m1y1[infant_subject]
    mother_subject = same_mi_pair_dict[infant_subject]
    my_mother_samples = list(subject_sample_map[mother_subject].keys())
    print(infant_subject)

    all_species = set([species for species, gene in genes_gained])
    for species in all_species:
        f = bz2.open('%s/genes/%s/genes_presabs.txt.bz2' % (config.data_directory, species),
'rt')
        header = f.readline().strip().split('\t') # Includes gene_id first column and with-c
sample names
        mother_sample_idx_name_tups = [(header.index(s), s) for s in my_mother_samples if s in
header]
        # print(len(mother_sample_idx_name_tups))
        for line in f:
            items = line.strip().split('\t')
            gene = items[0]
            if (species, gene) in genes_gained:
                subject_gene_presabs_dict[infant_subject][gene] = (1 in [int(items[i]) for i,
sample in mother_sample_idx_name_tups])


# In[44]:


subject_num_genes_pres_dict = defaultdict(int)

for infant_subject in subject_gene_presabs_dict:
    for gene in subject_gene_presabs_dict[infant_subject]:
        is_pres = subject_gene_presabs_dict[infant_subject][gene]
        subject_num_genes_pres_dict[infant_subject] += int(is_pres)


# In[46]:


for infant_subject in subject_num_genes_pres_dict:
```

```
    print("Infant %s:\t%i of %i genes gained (M1-Y1) are present in mother" % (infant_subject,

subject_num_genes_pres_dict[infant_subject],

len(subject_gene_presabs_dict[infant_subject])))


# In[25]:


for num_losses, days in loss_tups_by_tp_type['Mother-Infant']:
    if num_losses > 0:
        print((num_losses, days))


# In[26]:


for num_gains, days in gain_tups_by_tp_type['Mother-Infant']:
    if num_gains > 0:
        print((num_gains, days))


# In[27]:


# Subsample QP pairs in a category to n and get #sweeps/day, bootstrap x times
bootstrapped_gain_rates_by_tp_type = bootstrapped_agg_from_list_dict(gain_tups_by_tp_type,
get_rate_from_tuples,
                                                                     num_bootstraps=1000, n=40)
bootstrapped_loss_rates_by_tp_type = bootstrapped_agg_from_list_dict(loss_tups_by_tp_type,
get_rate_from_tuples,
                                                                     num_bootstraps=1000, n=40)

# Gene gain/loss rates boxplot
fig, ax = plt.subplots(1, 2, figsize=(16,4), sharey=True)

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

# These are the true "average" rates per category
gain_annotations = []; loss_annotations = []

# These are used as x-axis labels
gain_tp_type_labels = []; loss_tp_type_labels = []

bootstrapped_gain_rates = [bootstrapped_gain_rates_by_tp_type[tt] for tt in tp_types]
bootstrapped_loss_rates = [bootstrapped_loss_rates_by_tp_type[tt] for tt in tp_types]

for tp_type in tp_types:

    # Store gain info
    gain_count, total_days = (0,0)
    for num_gains, days in gain_tups_by_tp_type[tp_type]:
        gain_count += num_gains; total_days += days
    gain_annotations.append(float(gain_count)/total_days)
    gain_tp_type_labels.append('%s\nn=%i' % (tp_type, len(gain_tups_by_tp_type[tp_type])))

    # Store loss info
    loss_count, total_days = (0,0)
    for num_losss, days in loss_tups_by_tp_type[tp_type]:
        loss_count += num_losss; total_days += days
```

```
    loss_annotations.append(float(loss_count)/total_days)
    loss_tp_type_labels.append('%s\nn=%i' % (tp_type, len(loss_tups_by_tp_type[tp_type])))

ax[0].set_yscale('log')
ax[0].boxplot(bootstrapped_gain_rates)
ax[0].plot(np.arange(1, 1+len(gain_annotations)), gain_annotations, marker=(5, 2),
linestyle='None', markersize=12)
ax[0].set_xticklabels(gain_tp_type_labels, fontsize=12)
ax[0].set_ylabel("Gene changes per non-replacement QP pair per day\nBootstrapped (subsample QP
pairs 50 times)", fontsize=12)
ax[0].set_title("Gene gains")

ax[1].boxplot(bootstrapped_loss_rates)
ax[1].plot(np.arange(1, 1+len(loss_annotations)), loss_annotations, marker=(5, 2),
linestyle='None', markersize=12)
ax[1].set_xticklabels(loss_tp_type_labels, fontsize=12)
ax[1].set_title("Gene losses")

for tp_type, annotation in zip(tp_types, gain_annotations):
    print("%s: %s" % (tp_type, annotation))

for tp_type, annotation in zip(tp_types, loss_annotations):
    print("%s: %s" % (tp_type, annotation))

plt.subplots_adjust(wspace=0, hspace=0)
plt.show()

# fig.savefig('%s/gene_gain_and_loss_rates_by_tp_type_overall_with_zeros_labelled.png' %
plot_dir, bbox_inches='tight', dpi=600)


# In[53]:


# Matched
# gene gain/loss rate boxplot
fig, ax = plt.subplots(figsize=(4,4))

tp_types = ['4-8mon duration infant', '4-8mon duration adult']
labels = ['4-8mon duration\nInfant', '4-8mon duration\nAdult']
gain_annotations = [] # These are the true "average" rates per category
gain_tp_type_labels = [] # These are used as x-axis labels

bootstrapped_gain_rates = [bootstrapped_gain_rates_by_tp_type[tp_type] for tp_type in tp_types]
for tp_type, label in zip(tp_types, labels):
    gain_count, total_days = (0,0)
    for num_gains, days in gain_tups_by_tp_type[tp_type]:
        gain_count += num_gains; total_days += days
    gain_annotations.append(float(gain_count)/total_days)
    gain_tp_type_labels.append('%s\nn=%i' % (label, len(gain_tups_by_tp_type[tp_type])))

ax.plot(np.arange(1, 1+len(gain_annotations)), gain_annotations, marker=(5, 2),
linestyle='None', markersize=12)
ax.boxplot(bootstrapped_gain_rates)
# ax.set_ylim(0, 0.14)
ax.set_xticklabels(gain_tp_type_labels, fontsize=12)
ax.set_ylabel("Gene gains per day", fontsize=12)
# ax.set_title("Bootstrapped (subsample QP pairs 50 times), include zeros")

t, p = stats.ttest_ind(bootstrapped_gain_rates[0], bootstrapped_gain_rates[1])
print("T: %.02f\tp: %.3E" % (t, p))
```

```
plt.show()


# In[54]:


# Matched
# gene gain/loss rate boxplot
fig, ax = plt.subplots(figsize=(4,4))

tp_types = ['4-8mon duration infant', '4-8mon duration adult']
labels = ['4-8mon duration\nInfant', '4-8mon duration\nAdult']
loss_annotations = [] # These are the true "average" rates per category
loss_tp_type_labels = [] # These are used as x-axis labels

bootstrapped_loss_rates = [bootstrapped_loss_rates_by_tp_type[tp_type] for tp_type in tp_types]
for tp_type, label in zip(tp_types, labels):
    loss_count, total_days = (0,0)
    for num_losss, days in loss_tups_by_tp_type[tp_type]:
        loss_count += num_losss; total_days += days
    loss_annotations.append(float(loss_count)/total_days)
    loss_tp_type_labels.append('%s\nn=%i' % (label, len(loss_tups_by_tp_type[tp_type])))

ax.plot(np.arange(1, 1+len(loss_annotations)), loss_annotations, marker=(5, 2),
linestyle='None', markersize=12)
ax.boxplot(bootstrapped_loss_rates)
# ax.set_ylim(0, 0.14)
ax.set_xticklabels(loss_tp_type_labels, fontsize=12)
ax.set_ylabel("Gene losses per day", fontsize=12)
# ax.set_title("Bootstrapped (subsample QP pairs 50 times), include zeros")

t, p = stats.ttest_ind(bootstrapped_loss_rates[0], bootstrapped_loss_rates[1])
print("T: %.02f\tp: %.3E" % (t, p))
plt.show()


# In[42]:


# Permutation test for infant vs. adult
label_set = ['4-8mon duration infant', '4-8mon duration adult']
orig_label_data_dict = {label: gain_tups_by_tp_type[label] for label in label_set}
null_differences, true_difference = permutation_test(orig_label_data_dict, label_set,
                                                     aggregator_fn=get_rate_from_tuples,
                                                    num_bootstraps=10000)

# Plot permutation test results
fig, ax = plt.subplots(figsize=(2.3, 4))
p = permutation_test_p(null_differences, true_difference)
color = 'red' if p < 0.05 else 'black'
ax.violinplot(null_differences)
ax.plot([1], [true_difference], marker=(5, 2), linestyle='None', markersize=12, color=color,
zorder=9)
ax.set_xticks([1]); ax.set_xticklabels([]); ax.set_xlim((0.6, 1.4))
ax.set_ylabel("Difference, gene gain rate (Infant-Adult)", fontsize=12)
ax.set_xlabel(r'$\it{p}$' + ("<0.0001" if p == 0 else "=%.03f" % p), fontsize=12)
plt.tight_layout()
fig.savefig('%s/infant_vs_adult_comparison_gene_gain_rate.pdf' % config.analysis_directory)


# In[43]:
```

```
# Permutation test for infant vs. adult
label_set = ['4-8mon duration infant', '4-8mon duration adult']
orig_label_data_dict = {label: loss_tups_by_tp_type[label] for label in label_set}
null_differences, true_difference = permutation_test(orig_label_data_dict, label_set,
                                                     aggregator_fn=get_rate_from_tuples,
                                                    num_bootstraps=10000)

# Plot permutation test results
fig, ax = plt.subplots(figsize=(2.3, 4))
p = permutation_test_p(null_differences, true_difference)
color = 'red' if p < 0.05 else 'gray'
ax.violinplot(null_differences)
ax.plot([1], [true_difference], marker=(5, 2), linestyle='None', markersize=12, color=color,
zorder=9)
ax.set_xticks([1]); ax.set_xticklabels([]); ax.set_xlim((0.6, 1.4))
ax.set_ylabel("Difference, gene loss rate (Infant-Adult)", fontsize=12)
ax.set_xlabel(r'$\it{p}$' + ("<0.0001" if p == 0 else "=%.03f" % p), fontsize=12)
plt.tight_layout()
fig.savefig('%s/infant_vs_adult_comparison_gene_loss_rate.pdf' % config.analysis_directory)


# In[49]:


# QQ plots
from scipy import stats

fig, ax = plt.subplots(2, 5, figsize=(16, 8))
xs = np.arange(-3, 3, 0.01)

for i, tp_type in zip(range(len(bootstrapped_gain_rates)), tp_types):
    vals = stats.probplot(bootstrapped_gain_rates[i])
    ax[0][i].set_title(tp_type)
    osm, osr = vals[0]; m, b, r = vals[1]
    ax[0][i].plot(xs, (m*xs)+b, color='gray')
    ax[0][i].plot(osm, osr, '.', color='black', mfc='none')

for i, tp_type in zip(range(len(bootstrapped_loss_rates)), tp_types):
    vals = stats.probplot(bootstrapped_loss_rates[i])
    ax[1][i].set_title(tp_type)
    osm, osr = vals[0]; m, b, r = vals[1]
    ax[1][i].plot(xs, (m*xs)+b, color='gray')
    ax[1][i].plot(osm, osr, '.', color='black', mfc='none')
    ax[1][i].set_xlabel("Theoretical quantile")

ax[0][0].set_ylabel("Observed\ngene gain rate")
ax[1][0].set_ylabel("Observed\ngene loss rate")

plt.tight_layout()


# In[158]:


# Perform PERMUTATION tests between all pairs

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

gene_gain_rate_tp_pair_ptests = {}
```

```
print("GENE GAIN RATE TIMEPOINT PAIRWISE COMPARISONS")

for i in range(len(tp_types)):
    tp_type1 = tp_types[i]
    for j in range(i+1, len(tp_types)):
        tp_type2 = tp_types[j]
        # Permutation test for tp_type 1 vs._tp type2
        label_set = [tp_type1, tp_type2]
        orig_label_data_dict = {label: gain_tups_by_tp_type[label] for label in label_set}
        null_differences, true_difference = permutation_test(orig_label_data_dict, label_set,
                                                             aggregator_fn=get_rate_from_tuples,
                                                             num_bootstraps=10000)
        p = permutation_test_p(null_differences, true_difference)
        n1 = len(orig_label_data_dict[tp_type1])
        n2 = len(orig_label_data_dict[tp_type2])
        gene_gain_rate_tp_pair_ptests[(tp_type1, tp_type2)] = (n1, n2, p)
        print("==============================")
        print("%s vs. %s" % (tp_type1, tp_type2))
        print(str(p) + '\t' + get_sig_str(p))


# In[67]:


# Perform tests between all pairs

gene_gain_rate_tp_pair_tests = {} # (tp1, tp2) -> (t, p, es) Store for later table

print("GENE GAIN RATE TIMEPOINT PAIRWISE COMPARISONS")
print("Note: all sample sizes are 50")

for i in range(len(tp_types)):
    ind_rates1 = bootstrapped_gain_rates[i]; tp_type1 = tp_types[i]
    for j in range(i+1, len(tp_types)):
        ind_rates2 = bootstrapped_gain_rates[j]; tp_type2 = tp_types[j]
        print("==============================")
        print("%s vs. %s" % (tp_type1, tp_type2))
        stat, p, es = summarize_utest(ind_rates1, ind_rates2, simple=True)
        gene_gain_rate_tp_pair_tests[(tp_type1, tp_type2)] = (stat, p, es)
        print(get_sig_str(p))


# In[159]:


# Perform PERMUTATION tests between all pairs

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

gene_loss_rate_tp_pair_ptests = {}

print("GENE LOSS RATE TIMEPOINT PAIRWISE COMPARISONS")

for i in range(len(tp_types)):
    tp_type1 = tp_types[i]
    for j in range(i+1, len(tp_types)):
        tp_type2 = tp_types[j]
        # Permutation test for tp_type 1 vs._tp type2
        label_set = [tp_type1, tp_type2]
        orig_label_data_dict = {label: loss_tups_by_tp_type[label] for label in label_set}
```

```
        null_differences, true_difference = permutation_test(orig_label_data_dict, label_set,
                                                             aggregator_fn=get_rate_from_tuples,
                                                             num_bootstraps=10000)
        p = permutation_test_p(null_differences, true_difference)
        n1 = len(orig_label_data_dict[tp_type1])
        n2 = len(orig_label_data_dict[tp_type2])
        gene_loss_rate_tp_pair_ptests[(tp_type1, tp_type2)] = (n1, n2, p)
        print("==============================")
        print("%s vs. %s" % (tp_type1, tp_type2))
        print(str(p) + '\t' + get_sig_str(p))


# In[68]:


print("GENE LOSS RATE TIMEPOINT PAIRWISE COMPARISONS")
print("Note: all sample sizes are 50")

gene_loss_rate_tp_pair_tests = {} # (tp1, tp2) -> (t, p, es) Store for later table

for i in range(len(tp_types)):
    ind_rates1 = bootstrapped_loss_rates[i]; tp_type1 = tp_types[i]
    for j in range(i+1, len(tp_types)):
        ind_rates2 = bootstrapped_loss_rates[j]; tp_type2 = tp_types[j]
        print("==============================")
        print("%s vs. %s" % (tp_type1, tp_type2))
        stat, p, es = summarize_utest(ind_rates1, ind_rates2, simple=True)
        gene_loss_rate_tp_pair_tests[(tp_type1, tp_type2)] = (stat, p, es)
        print(get_sig_str(p))


# In[160]:


# Perform PERMUTATION tests between all pairs

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

gene_gain_vs_loss_rate_ptests = {}

print("GENE GAIN vs. LOSS RATE COMPARISONS")

for tp_type in tp_types:
    # Permutation test for gainvs .loss
    label_set = ['gain', 'loss']
    orig_label_data_dict = {'gain': gain_tups_by_tp_type[tp_type],
                            'loss': loss_tups_by_tp_type[tp_type]}
    null_differences, true_difference = permutation_test(orig_label_data_dict, label_set,
                                                         aggregator_fn=get_rate_from_tuples,
                                                         num_bootstraps=10000)
    p = permutation_test_p(null_differences, true_difference)
    n1 = len(orig_label_data_dict['gain'])
    n2 = len(orig_label_data_dict['loss'])
    gene_gain_vs_loss_rate_ptests[tp_type] = (n1, n2, p)
    print("==============================")
    print("%s: %s vs. %s" % (tp_type, label_set[0], label_set[1]))
    print(str(p) + '\t' + get_sig_str(p))


# In[69]:
```

```
print("GENE GAIN VS. LOSS RATE COMPARISONS")

gene_gain_vs_loss_rate_tests = {} # tp -> (t, p, es) Store for later table

for ind_rates1, ind_rates2, tp_type in zip(bootstrapped_gain_rates, bootstrapped_loss_rates,
tp_types):
    print("============================")
    print("%s: Gain vs. loss rate" % (tp_type))
    stat, p, es = summarize_utest(ind_rates1, ind_rates2, simple=True)
    gene_gain_vs_loss_rate_tests[tp_type] = (stat, p, es)
    print(get_sig_str(p))


# In[57]:


# Store all test results in table

f = open('%s/temporal_change_rate_tests.csv' % (config.analysis_directory), 'w')
f.write(','.join(['type', 'tp_type_1', 'tp_type_2', 't', 'P', "Cohen's D"]) + '\n')

for change_rate_type, stored_dict in zip(['SNV change rate', 'replacement rate', 'gene gain',
'gene loss'],
                                         [snp_change_rate_tp_pair_tests,
replacement_rate_tp_pair_tests,
                                          gene_gain_rate_tp_pair_tests,
gene_loss_rate_tp_pair_tests]):
    for tp1, tp2 in stored_dict:
        t, p, es = stored_dict[(tp1, tp2)]
        f.write(','.join([str(val) for val in [change_rate_type, tp1, tp2, t, p, es]]) + '\n')

for tp in gene_gain_vs_loss_rate_tests:
    t, p, es = gene_gain_vs_loss_rate_tests[tp]
    f.write(','.join([str(val) for val in ['gene gain vs. loss', tp, 'NA', t, p, es]]) + '\n')

f.close()


# In[161]:


# Store all permutation test results in table

f = open('%s/temporal_change_rate_permutation_tests_final.csv' % (config.analysis_directory),
'w')
f.write(','.join(['type', 'tp_type_1', 'tp_type_2', 'n1', 'n2', 'p']) + '\n')

for change_rate_type, stored_dict in zip(['SNV change rate', 'replacement rate', 'gene gain',
'gene loss'],
                                         [snp_change_rate_tp_pair_ptests,
replacement_rate_tp_pair_ptests,
                                          gene_gain_rate_tp_pair_ptests,
gene_loss_rate_tp_pair_ptests]):
    for tp1, tp2 in stored_dict:
        n1, n2, p = stored_dict[(tp1, tp2)]
        f.write(','.join([str(val) for val in [change_rate_type, tp1, tp2, n1, n2, p]]) + '\n')

for tp in gene_gain_vs_loss_rate_ptests:
    n1, n2, p = gene_gain_vs_loss_rate_ptests[tp]
    f.write(','.join([str(val) for val in ['gene gain vs. loss', tp, 'NA', n1, n2, p]]) + '\n')
```

```
f.close()


# In[98]:


# Store gene change information

cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
gain_tups_by_cohort_tp_type = {ds: defaultdict(list) for ds in cohorts}
loss_tups_by_cohort_tp_type = {ds: defaultdict(list) for ds in cohorts}
genes_gained_by_cohort_tp_type = {ds: defaultdict(set) for ds in cohorts}

# infant subject -> (species, gene) genes gained
genes_gained_m1y1 = defaultdict(list)

for species in gene_changes:
    for s1, s2 in gene_changes[species]:

        dataset = sample_cohort_map[s1]
        gains, losses = gene_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue

        # Arbitrarily set mother 0 - infant 0 as 1 day
        if is_mother(s1) and mi_sample_day_dict[s1] == 0 and is_infant(s2) and
mi_sample_day_dict[s2] == 0:
            days = 1

        if isinstance(snp_changes[species][(s1, s2)], int): # Replacement (in terms of SNP
changes)
            continue# num_gains, num_losses = gene_changes[species][(s1, s2)]
        else:
            num_gains = len(gene_changes[species][(s1, s2)][0])
            num_losses = len(gene_changes[species][(s1, s2)][1])

        # Try excluding the Mother-Infant(wk1) QP pair with unsually high number of losses
        if (num_losses >= 13) and ('Mother-Infant' in custom_cohorts):
            print("Sample pair %s, %s | %s" % (s1, s2, species))
            print("Num losses: %i | Num gains: %i" % (num_losses, num_gains))
            continue

        for custom_cohort in custom_cohorts:
            gain_tups_by_cohort_tp_type[dataset][custom_cohort].append((num_gains, days))
            loss_tups_by_cohort_tp_type[dataset][custom_cohort].append((num_losses, days))
            for gene, _, _, _, _ in gains:
                genes_gained_by_cohort_tp_type[dataset][custom_cohort].add((species, gene))

        if 'Month 1-Year 1' in custom_cohorts:
            if sample_subject_map[s1] != sample_subject_map[s2]:
                print("What??")
            infant_subject = sample_subject_map[s1]
            for gene, _, _, _, _ in gains:
                genes_gained_m1y1[infant_subject].append((species, gene))
```

```
# Gene change rates by cohort boxplot
fig, ax = plt.subplots(4, 2, figsize=(10, 12), sharey=True)

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

for i in range(4):

    dataset = cohorts[i]
    # Subsample QP pairs in a category to n=50 and get #sweeps/day, bootstrap 20 times
    bootstrapped_gain_rates_by_tp_type =
bootstrapped_agg_from_list_dict(gain_tups_by_cohort_tp_type[dataset], get_rate_from_tuples,
n=10)
    bootstrapped_loss_rates_by_tp_type =
bootstrapped_agg_from_list_dict(loss_tups_by_cohort_tp_type[dataset], get_rate_from_tuples,
n=10)

    gain_annotations = []; loss_annotations = [] # These are the true "average" rates per
category
    gain_tp_type_labels = []; loss_tp_type_labels = [] # These are used as x-axis labels
    bootstrapped_gain_rates = []; bootstrapped_loss_rates = [] # Data

    for tp_type in tp_types:

        if len(bootstrapped_gain_rates_by_tp_type[tp_type]) > 0:
            # Store gain info
            gain_count, total_days = (0,0)
            for num_gains, days in gain_tups_by_cohort_tp_type[dataset][tp_type]:
                gain_count += num_gains; total_days += days
            bootstrapped_gain_rates.append(bootstrapped_gain_rates_by_tp_type[tp_type])
            gain_annotations.append(float(gain_count)/total_days)
            gain_tp_type_labels.append('%s\nn=%i' % (tp_type,
len(gain_tups_by_cohort_tp_type[dataset][tp_type])))

        if len(bootstrapped_loss_rates_by_tp_type[tp_type]) > 0:
            # Store loss info
            loss_count, total_days = (0,0)
            for num_losss, days in loss_tups_by_cohort_tp_type[dataset][tp_type]:
                loss_count += num_losss; total_days += days
            bootstrapped_loss_rates.append(bootstrapped_loss_rates_by_tp_type[tp_type])
            loss_annotations.append(float(loss_count)/total_days)
            loss_tp_type_labels.append('%s\nn=%i' % (tp_type,
len(loss_tups_by_cohort_tp_type[dataset][tp_type])))

    ax[i][0].boxplot(bootstrapped_gain_rates)
    ax[i][0].plot(np.arange(1, 1+len(gain_annotations)), gain_annotations, marker=(5, 2),
linestyle='None', markersize=12)
    ax[i][0].set_xticklabels(gain_tp_type_labels, fontsize=12)
    ax[i][0].set_ylabel("Gene changes per\nnon-replacement\nQP pair per day", fontsize=12)
    # ax[i][0].set_yscale('log')
    ax[i][0].set_title("Gene gains")

    ax[i][1].boxplot(bootstrapped_loss_rates)
    ax[i][1].plot(np.arange(1, 1+len(loss_annotations)), loss_annotations, marker=(5, 2),
linestyle='None', markersize=12)
    ax[i][1].set_xticklabels(loss_tp_type_labels, fontsize=12)
    ax[i][1].set_title("Gene losses")

plt.subplots_adjust(hspace=0)
plt.tight_layout()
plt.show()
```

```
# In[81]:


# Store gene change information
# Split by delivery mode

devmodes = ['Vaginal', 'C-section']

gain_tups_by_devmode_tp_type = {dm: defaultdict(list) for dm in devmodes}
subject_gain_tups_by_tp_type = defaultdict(list)
loss_tups_by_devmode_tp_type = {dm: defaultdict(list) for dm in devmodes}
subject_loss_tups_by_tp_type = defaultdict(list)
genes_gained_by_devmode_tp_type = {dm: defaultdict(set) for dm in devmodes}

restrict_cohort = [] # disabled for now

for species in gene_changes:
    for s1, s2 in gene_changes[species]:

        cohort = sample_cohort_map[s1]

        subject = sample_subject_map[s2]
        if subject not in subject_delivery_mode_map: # Skip non-infant second sample
            continue

        delivery_mode = subject_delivery_mode_map[subject]

        subject = sample_subject_map[s2]
        delivery_mode = subject_delivery_mode_map[subject]
        gains, losses = gene_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue

        # Arbitrarily set mother 0 - infant 0 as 1 day
        if is_mother(s1) and mi_sample_day_dict[s1] == 0 and is_infant(s2) and
mi_sample_day_dict[s2] == 0:
            days = 1

        if isinstance(snp_changes[species][(s1, s2)], int): # Replacement (in terms of SNP
changes)
            continue# num_gains, num_losses = gene_changes[species][(s1, s2)]
        else:
            num_gains = len(gene_changes[species][(s1, s2)][0])
            num_losses = len(gene_changes[species][(s1, s2)][1])

        # Try excluding the Mother-Infant(wk1) QP pair with unsually high number of losses
        if (num_losses >= 13) and ('Mother-Infant' in custom_cohorts):
            print("Sample pair %s, %s | %s" % (s1, s2, species))
            print("Num losses: %i | Num gains: %i" % (num_losses, num_gains))
            continue

        for custom_cohort in custom_cohorts:
            gain_tups_by_devmode_tp_type[delivery_mode][custom_cohort].append((num_gains, days))
            subject_gain_tups_by_tp_type[custom_cohort].append((subject, num_gains, days))
```

```
            loss_tups_by_devmode_tp_type[delivery_mode][custom_cohort].append((num_losses,
days))
            subject_loss_tups_by_tp_type[custom_cohort].append((subject, num_losses, days))
            for gene, _, _, _, _ in gains:
                genes_gained_by_devmode_tp_type[delivery_mode][custom_cohort].add((species,
gene))


# In[100]:


for dm in gain_tups_by_devmode_tp_type:
    for tp in gain_tups_by_devmode_tp_type[dm]:
        print("%i %s %s" % (len(gain_tups_by_devmode_tp_type[dm][tp]), dm, tp))


# In[58]:


tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']
for tp_type in tp_types:
    print(tp_type)
    print(len(gain_tups_by_devmode_tp_type['C-section'][tp_type]))


# In[125]:


# Summarize distribution by delivery, feeding mode for gene gain rates
time_interval_category = 'Day 0-Week 1'
feeding_delivery_count_dict = {fmode: defaultdict(int) for fmode in ['breast', 'formula',
'mixed']}
for subject, count, days in subject_gain_tups_by_tp_type[time_interval_category]:
    if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
        feeding_mode = subject_feeding_mode_map[subject]
        delivery_mode = subject_delivery_mode_map[subject]
        feeding_delivery_count_dict[feeding_mode][delivery_mode] += 1

dmodes = ['Vaginal', 'C-section']
fmodes = ['breast', 'formula', 'mixed']
print("Gene gains | %s" % time_interval_category)
print('----------------------')
print('\t' + '\t'.join(dmodes))
for fmode in fmodes:
    vals = [str(feeding_delivery_count_dict[fmode][dmode]) for dmode in dmodes]
    print("%s\t" % (fmode) + '\t'.join(vals))


# In[189]:


# Reproduce in the more elegant way heh
# Permutation tests for delivery and feeding mode

fig, ax = plt.subplots(1, 2, figsize=(8, 4))

time_interval_categories = ['Day 0-Week 1', 'Week 1-Month 1']

# Vaginal vs. c-section
label_sets = [['Vaginal', 'C-section'], ['breast', 'formula']]
label_mode_type_list = [0, 1] # 0 for delivery mode, 1 for feeding mode
```

```
for i in range(2):
    label_set = label_sets[i] # Which two labels we are comparing
    label_mode_type = label_mode_type_list[i]

    null_differences_list = []
    true_difference_list = []
    labels = []
    color_list = []

    for time_interval_category in time_interval_categories:
        orig_label_subject_data_dict = defaultdict(list)
        for label in label_set:
            for subject, count, days in subject_gain_tups_by_tp_type[time_interval_category]:
                if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                    feeding_mode = subject_feeding_mode_map[subject]
                    delivery_mode = subject_delivery_mode_map[subject]
                    if label_mode_type == 0:
                        orig_label_subject_data_dict[delivery_mode].append((subject, (count,
days)))
                    if label_mode_type == 1:
                        orig_label_subject_data_dict[feeding_mode].append((subject, (count,
days)))

        null_differences, true_difference =
permutation_test_subject(orig_label_subject_data_dict, label_set,

aggregator_fn=get_rate_from_tuples, num_bootstraps=10000)
        null_differences_list.append(null_differences)
        true_difference_list.append(true_difference)
        p = permutation_test_p(null_differences, true_difference)
        color = 'red' if p < 0.05 else 'black'
        color_list.append(color)
        labels.append('%s\n' % time_interval_category + r'$\it{p}$' + ("<0.0001" if p == 0 else
"=%.03f" % p))

    ax[i].violinplot(null_differences_list)
    for j in range(len(true_difference_list)):
        ax[i].plot([j+1], [true_difference_list[j]], marker=(5, 2), linestyle='None',
markersize=12, color=color_list[j], zorder=9)
    ax[i].set_xticks([1, 2])
    ax[i].set_xticklabels(labels)
    ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
    # ax[i].set_xlabel(r'$\it{p}$' + "=%.04f" % p, fontsize=12)
    ax[i].set_title("Gene gain rate")

plt.subplots_adjust(wspace=0.5)
plt.show()
fig.savefig('%s/mode_comparisons_gene_gain_rate.pdf' % config.analysis_directory)
fig.savefig('%s/mode_comparisons_gene_gain_rate.png' % config.analysis_directory, dpi=200)


# In[116]:


# Reproduce in the more elegant way heh
# Permutation tests for delivery and feeding mode

fig, ax = plt.subplots(1, 4, figsize=(8, 4))

time_interval_category = 'Week 1-Month 1'
```

```
label_sets = [['Vaginal', 'C-section'], ['Vaginal', 'C-section'], ['breast', 'formula'],
['breast', 'formula']]
label_mode_type_list = [0, 0, 1, 1] # 0 for delivery mode, 1 for feeding mode
restrict_other_mode_list = ['breast', 'formula', 'Vaginal', 'C-section']

for i in range(4):
    label_set = label_sets[i] # Which two labels we are comparing
    label_mode_type = label_mode_type_list[i]
    restrict_other_mode = restrict_other_mode_list[i]

    orig_label_subject_data_dict = defaultdict(list)
    for label in label_set:
        for subject, count, days in subject_gain_tups_by_tp_type[time_interval_category]:
            if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                feeding_mode = subject_feeding_mode_map[subject]
                delivery_mode = subject_delivery_mode_map[subject]
                if label_mode_type == 0:
                    if feeding_mode == restrict_other_mode:
                        orig_label_subject_data_dict[delivery_mode].append((subject, (count,
days)))
                if label_mode_type == 1:
                    if delivery_mode == restrict_other_mode:
                        orig_label_subject_data_dict[feeding_mode].append((subject, (count,
days)))
    null_differences, true_difference = permutation_test_subject(orig_label_subject_data_dict,
label_set, aggregator_fn=get_rate_from_tuples)

    p = permutation_test_p(null_differences, true_difference)
    color = 'red' if p < 0.05 else 'black'
    ax[i].violinplot(null_differences)
    ax[i].plot([1], [true_difference], marker=(5, 2), linestyle='None', markersize=12,
color=color, zorder=9)
    ax[i].set_xticks([1]); ax[i].set_xticklabels([]); ax[i].set_xlim((0.6, 1.4))
    ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
    ax[i].set_xlabel(r'$\it{p}$' + "=%.03f" % p, fontsize=12)
    if i == 0:
        ax[i].set_title("%s\n%s only" % (time_interval_category, restrict_other_mode),
loc='left')
    else:
        ax[i].set_title("%s only" % restrict_other_mode)

plt.subplots_adjust(wspace=2)
plt.show()


# In[128]:


# Summarize distribution by delivery, feeding mode for gene loss rates
# Should be same as for gains
time_interval_category = 'Week 1-Month 1'
feeding_delivery_count_dict = {fmode: defaultdict(int) for fmode in ['breast', 'formula',
'mixed']}
for subject, count, days in subject_loss_tups_by_tp_type[time_interval_category]:
    if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
        feeding_mode = subject_feeding_mode_map[subject]
        delivery_mode = subject_delivery_mode_map[subject]
        feeding_delivery_count_dict[feeding_mode][delivery_mode] += 1

dmodes = ['Vaginal', 'C-section']
```

```
fmodes = ['breast', 'formula', 'mixed']
print("Gene loss | %s" % time_interval_category)
print('----------------------')
print('\t' + '\t'.join(dmodes))
for fmode in fmodes:
    vals = [str(feeding_delivery_count_dict[fmode][dmode]) for dmode in dmodes]
    print("%s\t" % (fmode) + '\t'.join(vals))


# In[190]:


# Reproduce in the more elegant way heh
# Permutation tests for delivery and feeding mode

fig, ax = plt.subplots(1, 2, figsize=(8, 4))

time_interval_categories = ['Day 0-Week 1', 'Week 1-Month 1']

# Vaginal vs. c-section
label_sets = [['Vaginal', 'C-section'], ['breast', 'formula']]
label_mode_type_list = [0, 1] # 0 for delivery mode, 1 for feeding mode

for i in range(2):
    label_set = label_sets[i] # Which two labels we are comparing
    label_mode_type = label_mode_type_list[i]

    null_differences_list = []
    true_difference_list = []
    labels = []
    color_list = []

    for time_interval_category in time_interval_categories:
        orig_label_subject_data_dict = defaultdict(list)
        for label in label_set:
            for subject, count, days in subject_loss_tups_by_tp_type[time_interval_category]:
                if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                    feeding_mode = subject_feeding_mode_map[subject]
                    delivery_mode = subject_delivery_mode_map[subject]
                    if label_mode_type == 0:
                        orig_label_subject_data_dict[delivery_mode].append((subject, (count,
days)))
                    if label_mode_type == 1:
                        orig_label_subject_data_dict[feeding_mode].append((subject, (count,
days)))

        null_differences, true_difference =
permutation_test_subject(orig_label_subject_data_dict, label_set,

aggregator_fn=get_rate_from_tuples, num_bootstraps=10000)
        null_differences_list.append(null_differences)
        true_difference_list.append(true_difference)
        p = permutation_test_p(null_differences, true_difference)
        color = 'red' if p < (0.05) else 'black'
        color_list.append(color)
        labels.append('%s\n' % time_interval_category + r'$\it{p}$' + "=%.03f" % p)

    ax[i].violinplot(null_differences_list)
    for j in range(len(true_difference_list)):
        ax[i].plot([j+1], [true_difference_list[j]], marker=(5, 2), linestyle='None',
markersize=12, color=color_list[j], zorder=9)
```

```
    ax[i].set_xticks([1, 2])
    ax[i].set_xticklabels(labels)
    ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
    # ax[i].set_xlabel(r'$\it{p}$' + "=%.03f" % p, fontsize=12)
    ax[i].set_title("Gene loss rate")

plt.subplots_adjust(wspace=0.5)
plt.show()
fig.savefig('%s/mode_comparisons_gene_loss_rate.pdf' % config.analysis_directory)
fig.savefig('%s/mode_comparisons_gene_loss_rate.png' % config.analysis_directory, dpi=200)


# In[130]:


# Permutation tests for delivery and feeding mode

fig, ax = plt.subplots(1, 4, figsize=(8, 4))

time_interval_category = 'Day 0-Week 1'

label_sets = [['Vaginal', 'C-section'], ['Vaginal', 'C-section'], ['breast', 'formula'],
['breast', 'formula']]
label_mode_type_list = [0, 0, 1, 1] # 0 for delivery mode, 1 for feeding mode
restrict_other_mode_list = ['breast', 'formula', 'Vaginal', 'C-section']

for i in range(4):
    label_set = label_sets[i] # Which two labels we are comparing
    label_mode_type = label_mode_type_list[i]
    restrict_other_mode = restrict_other_mode_list[i]

    orig_label_subject_data_dict = defaultdict(list)
    for label in label_set:
        for subject, count, days in subject_loss_tups_by_tp_type[time_interval_category]:
            if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                feeding_mode = subject_feeding_mode_map[subject]
                delivery_mode = subject_delivery_mode_map[subject]
                if label_mode_type == 0:
                    if feeding_mode == restrict_other_mode:
                        orig_label_subject_data_dict[delivery_mode].append((subject, (count,
days)))
                if label_mode_type == 1:
                    if delivery_mode == restrict_other_mode:
                        orig_label_subject_data_dict[feeding_mode].append((subject, (count,
days)))
    null_differences, true_difference = permutation_test_subject(orig_label_subject_data_dict,
label_set, aggregator_fn=get_rate_from_tuples)

    p = permutation_test_p(null_differences, true_difference)
    color = 'red' if p < 0.05 else 'black'
    ax[i].violinplot(null_differences)
    ax[i].plot([1], [true_difference], marker=(5, 2), linestyle='None', markersize=12,
color=color, zorder=9)
    ax[i].set_xticks([1]); ax[i].set_xticklabels([]); ax[i].set_xlim((0.6, 1.4))
    ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
    ax[i].set_xlabel(r'$\it{p}$' + "=%.03f" % p, fontsize=12)
    if i == 0:
        ax[i].set_title("%s\n%s only" % (time_interval_category, restrict_other_mode),
loc='left')
    else:
        ax[i].set_title("%s only" % restrict_other_mode)
```

```
plt.subplots_adjust(wspace=2)
plt.show()


# In[105]:


i = 0
label_set = label_sets[i] # Which two labels we are comparing
label_mode_type = label_mode_type_list[i]
restrict_other_mode = restrict_other_mode_list[i]

orig_label_subject_data_dict = defaultdict(list)
for label in label_set:
    for subject, count, days in subject_gain_tups_by_tp_type[time_interval_category]:
        if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
            feeding_mode = subject_feeding_mode_map[subject]
            delivery_mode = subject_delivery_mode_map[subject]
            if label_mode_type == 0:
                if feeding_mode == restrict_other_mode:
                    orig_label_subject_data_dict[delivery_mode].append((subject, (count, days)))
            if label_mode_type == 1:
                if delivery_mode == restrict_other_mode:
                    orig_label_subject_data_dict[feeding_mode].append((subject, (count, days)))


# In[158]:


# Permutation tests for delivery and feeding mode

fig, ax = plt.subplots(1, 2, figsize=(8, 4))

# ========================================================================
# Delivery mode
time_interval_categories = ['Day 0-Week 1', 'Week 1-Month 1']
mult_null_differences = []
mult_true_differences = []
plot_labels = []

label_set = ['Vaginal', 'C-section']

for time_interval_category in time_interval_categories:
    orig_label_subject_data_dict = defaultdict(list)
    for label in label_set:
        for subject, count, days in subject_loss_tups_by_tp_type[time_interval_category]:
            if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                feeding_mode = subject_feeding_mode_map[subject]
                delivery_mode = subject_delivery_mode_map[subject]
                orig_label_subject_data_dict[delivery_mode].append((subject, (count, days)))

    null_differences, true_difference = permutation_test_subject(orig_label_subject_data_dict,
label_set,

aggregator_fn=get_rate_from_tuples)
    p = permutation_test_p(null_differences, true_difference)

    mult_null_differences.append(null_differences)
    mult_true_differences.append(true_difference)
    plot_labels.append(("%s\n" % time_interval_category) + r'$\it{p}$' + ("=%.03f" % (p)))
```

```
i = 0

xs = np.arange(1, 1+len(mult_true_differences))
ax[i].violinplot(mult_null_differences)
ax[i].plot(xs, mult_true_differences, marker=(5, 2),
           linestyle='None', markersize=12, color=color, zorder=9)

ax[i].set_xticks(xs)
ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
ax[i].set_xticklabels(plot_labels, fontsize=12)
# ========================================================================

# ========================================================================
# Feeding mode
time_interval_categories = ['Day 0-Week 1', 'Week 1-Month 1']
mult_null_differences = []
mult_true_differences = []
plot_labels = []

label_set = ['breast', 'formula']

for time_interval_category in time_interval_categories:
    orig_label_subject_data_dict = defaultdict(list)
    for label in label_set:
        for subject, count, days in subject_loss_tups_by_tp_type[time_interval_category]:
            if subject in subject_delivery_mode_map and subject in subject_feeding_mode_map:
                feeding_mode = subject_feeding_mode_map[subject]
                delivery_mode = subject_delivery_mode_map[subject]
                orig_label_subject_data_dict[feeding_mode].append((subject, (count, days)))

    null_differences, true_difference = permutation_test_subject(orig_label_subject_data_dict,
label_set,

aggregator_fn=get_rate_from_tuples)
    p = permutation_test_p(null_differences, true_difference)

    mult_null_differences.append(null_differences)
    mult_true_differences.append(true_difference)
    plot_labels.append(("%s\n" % time_interval_category) + r'$\it{p}$' + ("=%.03f" % (p)))

i += 1

xs = np.arange(1, 1+len(mult_true_differences))
ax[i].violinplot(mult_null_differences)
ax[i].plot(xs, mult_true_differences, marker=(5, 2),
           linestyle='None', markersize=12, color=color, zorder=9)

ax[i].set_xticks(xs)
ax[i].set_ylabel("Difference (%s-%s)" % (label_set[0], label_set[1]), fontsize=12)
ax[i].set_xticklabels(plot_labels, fontsize=12)

plt.subplots_adjust(wspace=0.5)
plt.show()


# In[88]:


# Permutation test for delivery mode
# 1. permute C and V for all the babies
```

```
# 2. compute the SNV change rate for new C and new V babies;
# make sure to take the sum of the changes divided by the total time
# 3. compute difference in rate
# 4. Repeat x1000.
# 5. Compare true difference between C and V

num_bootstraps = 1000
cats = ['Vaginal', 'C-section']

subjects = list(subject_delivery_mode_map.keys()) # Order matters!!
delivery_modes = [] # True/original
for subject in subjects:
    delivery_modes.append(subject_delivery_mode_map[subject])

true_gain_rate_by_cat_tp_type_dict = {cat: {} for cat in cats}

for tp_type in subject_gain_tups_by_tp_type:
    total_days_by_cat_dict = defaultdict(int)
    total_gain_count_by_cat_dict = defaultdict(int)
    for subject, count, days in subject_gain_tups_by_tp_type[tp_type]:
        delivery_mode = subject_delivery_mode_map[subject]
        total_days_by_cat_dict[delivery_mode] += days
        total_gain_count_by_cat_dict[delivery_mode] += count

    for cat in cats:
        days = float(total_days_by_cat_dict[cat])
        gain_rate = total_gain_count_by_cat_dict[cat]/days
        true_gain_rate_by_cat_tp_type_dict[cat][tp_type] = gain_rate

permuted_gain_rates_by_cat_tp_type_dict = {cat: defaultdict(list) for cat in cats}

i = 0
while i < num_bootstraps:
    random.shuffle(delivery_modes) # Randomly permute delivery modes
    permuted_subject_delivery_mode_map = {}
    for subject, delivery_mode in zip(subjects, delivery_modes):
        permuted_subject_delivery_mode_map[subject] = delivery_mode

    bad = False # If end up with sample size 0 for any timepoint-cat pair, skip?? Not used for
now

    # Compute SNV change rate from subject_count_time_tups_by_tp_type
    for tp_type in subject_count_time_tups_by_tp_type:
        total_days_by_cat_dict = defaultdict(int)
        total_gain_count_by_cat_dict = defaultdict(int)
        for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
            delivery_mode = permuted_subject_delivery_mode_map[subject]
            total_days_by_cat_dict[delivery_mode] += days
            total_gain_count_by_cat_dict[delivery_mode] += count

        for cat in cats:
            days = float(total_days_by_cat_dict[cat])
            if days > 0:
                gain_rate = total_gain_count_by_cat_dict[cat]/days
            else:
                gain_rate = 0 # Consider it 0 if there are no samples/babies for this timepoint-
cat
            permuted_gain_rates_by_cat_tp_type_dict[cat][tp_type].append(gain_rate)

    i += 1
```

```
# Just interested in Week 0-Month 1 and Month 1-Year

fig, ax = plt.subplots(figsize=(4,4))

tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

null_differences_by_tp_type = []
labels = []
true_differences_by_tp_type = []

for tp_type in tp_types:
    null_differences = []
    for vrate, crate in zip(permuted_gain_rates_by_cat_tp_type_dict['Vaginal'][tp_type],
                            permuted_gain_rates_by_cat_tp_type_dict['C-section'][tp_type]):
        null_differences.append(vrate-crate)
    null_differences_by_tp_type.append(null_differences)
    true_vrate = true_gain_rate_by_cat_tp_type_dict['Vaginal'][tp_type]
    true_crate = true_gain_rate_by_cat_tp_type_dict['C-section'][tp_type]
    true_difference = true_vrate-true_crate
    true_differences_by_tp_type.append(true_difference)

    p = permutation_test_p(null_differences, true_difference)
    labels.append(("%s\n" % tp_type) + r'$\it{p}$' + ("=%.03f" % (p)))

ax.violinplot(null_differences_by_tp_type, positions=[1, 2])
ax.plot(np.arange(1, 1+len(true_differences_by_tp_type)), true_differences_by_tp_type,
        marker=(5, 2), linestyle='None', markersize=12, color='black', zorder=9)
ax.set_xticks([1, 2])
ax.set_xticklabels(labels)
ax.set_ylabel("Difference between vaginal\nand C-section gene gain rate")
plt.show()
fig.savefig('%s/delivery_mode_comparison_gene_gain_rate.pdf' % config.analysis_directory)


# In[100]:


# Permutation test for feeding mode
# 1. permute C and V for all the babies
# 2. compute the SNV change rate for new C and new V babies;
# make sure to take the sum of the changes divided by the total time
# 3. compute difference in rate
# 4. Repeat x1000.
# 5. Compare true difference between C and V

num_bootstraps = 1000
cats = ['breast', 'formula']
tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

subjects = list(subject_feeding_mode_map.keys()) # Order matters!!
feeding_modes = [] # True/original
for subject in subjects:
    feeding_modes.append(subject_feeding_mode_map[subject])

true_gain_rate_by_cat_tp_type_dict = {cat: {} for cat in cats}

for tp_type in tp_types:
    total_days_by_cat_dict = defaultdict(int)
    total_gain_count_by_cat_dict = defaultdict(int)
    for subject, count, days in subject_gain_tups_by_tp_type[tp_type]:
        try:
```

```
            feeding_mode = subject_feeding_mode_map[subject]
        except:
            continue # Lots of Shao samples, for example, have NA for feeding mode
        total_days_by_cat_dict[feeding_mode] += days
        total_gain_count_by_cat_dict[feeding_mode] += count

    for cat in cats:
        days = float(total_days_by_cat_dict[cat])
        gain_rate = total_gain_count_by_cat_dict[cat]/days
        true_gain_rate_by_cat_tp_type_dict[cat][tp_type] = gain_rate

permuted_gain_rates_by_cat_tp_type_dict = {cat: defaultdict(list) for cat in cats}

i = 0
while i < num_bootstraps:
    random.shuffle(feeding_modes) # Randomly permute feeding modes
    permuted_subject_feeding_mode_map = {}
    for subject, feeding_mode in zip(subjects, feeding_modes):
        permuted_subject_feeding_mode_map[subject] = feeding_mode

    bad = False # If end up with sample size 0 for any timepoint-cat pair, skip?? Not used for
now

    # Compute SNV change rate from subject_count_time_tups_by_tp_type
    for tp_type in subject_count_time_tups_by_tp_type:
        total_days_by_cat_dict = defaultdict(int)
        total_gain_count_by_cat_dict = defaultdict(int)
        for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
            feeding_mode = permuted_subject_feeding_mode_map[subject]
            total_days_by_cat_dict[feeding_mode] += days
            total_gain_count_by_cat_dict[feeding_mode] += count

        for cat in cats:
            days = float(total_days_by_cat_dict[cat])
            if days > 0:
                gain_rate = total_gain_count_by_cat_dict[cat]/days
            else:
                gain_rate = 0 # Consider it 0 if there are no samples/babies for this timepoint-
cat
            permuted_gain_rates_by_cat_tp_type_dict[cat][tp_type].append(gain_rate)

    i += 1

# Just interested in Week 0-Month 1 and Month 1-Year

fig, ax = plt.subplots(figsize=(4,4))

tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

null_differences_by_tp_type = []
labels = []
true_differences_by_tp_type = []

for tp_type in tp_types:
    null_differences = []
    for rate1, rate2 in zip(permuted_gain_rates_by_cat_tp_type_dict['breast'][tp_type],
                            permuted_gain_rates_by_cat_tp_type_dict['formula'][tp_type]):
        null_differences.append(rate1-rate2)
    null_differences_by_tp_type.append(null_differences)
    true_rate1 = true_gain_rate_by_cat_tp_type_dict['breast'][tp_type]
    true_rate2 = true_gain_rate_by_cat_tp_type_dict['formula'][tp_type]
```

```
    true_difference = true_rate1-true_rate2
    true_differences_by_tp_type.append(true_difference)

    p = permutation_test_p(null_differences, true_difference)
    labels.append(("%s\n" % tp_type) + r'$\it{p}$' + ("=%.03f" % (p)))

ax.violinplot(null_differences_by_tp_type, positions=[1, 2])
ax.plot(np.arange(1, 1+len(true_differences_by_tp_type)), true_differences_by_tp_type,
        marker=(5, 2), linestyle='None', markersize=12, color='black', zorder=9)
ax.set_xticks([1, 2])
ax.set_xticklabels(labels)
ax.set_ylabel("Difference between breast\nand formula feeding gene gain rate")
plt.show()
fig.savefig('%s/feeding_mode_comparison_gene_gain_rate.pdf' % config.analysis_directory)


# In[87]:


# Loss next
# Permutation test for delivery mode
# 1. permute C and V for all the babies
# 2. compute the SNV change rate for new C and new V babies;
# make sure to take the sum of the changes divided by the total time
# 3. compute difference in rate
# 4. Repeat x1000.
# 5. Compare true difference between C and V

num_bootstraps = 1000
cats = ['Vaginal', 'C-section']

subjects = list(subject_delivery_mode_map.keys()) # Order matters!!
delivery_modes = [] # True/original
for subject in subjects:
    delivery_modes.append(subject_delivery_mode_map[subject])

true_loss_rate_by_cat_tp_type_dict = {cat: {} for cat in cats}

for tp_type in subject_loss_tups_by_tp_type:
    total_days_by_cat_dict = defaultdict(int)
    total_loss_count_by_cat_dict = defaultdict(int)
    for subject, count, days in subject_loss_tups_by_tp_type[tp_type]:
        delivery_mode = subject_delivery_mode_map[subject]
        total_days_by_cat_dict[delivery_mode] += days
        total_loss_count_by_cat_dict[delivery_mode] += count

    for cat in cats:
        days = float(total_days_by_cat_dict[cat])
        loss_rate = total_loss_count_by_cat_dict[cat]/days
        true_loss_rate_by_cat_tp_type_dict[cat][tp_type] = loss_rate

permuted_loss_rates_by_cat_tp_type_dict = {cat: defaultdict(list) for cat in cats}

i = 0
while i < num_bootstraps:
    random.shuffle(delivery_modes) # Randomly permute delivery modes
    permuted_subject_delivery_mode_map = {}
    for subject, delivery_mode in zip(subjects, delivery_modes):
        permuted_subject_delivery_mode_map[subject] = delivery_mode
```

```
    bad = False # If end up with sample size 0 for any timepoint-cat pair, skip?? Not used for
now

    # Compute SNV change rate from subject_count_time_tups_by_tp_type
    for tp_type in subject_count_time_tups_by_tp_type:
        total_days_by_cat_dict = defaultdict(int)
        total_loss_count_by_cat_dict = defaultdict(int)
        for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
            delivery_mode = permuted_subject_delivery_mode_map[subject]
            total_days_by_cat_dict[delivery_mode] += days
            total_loss_count_by_cat_dict[delivery_mode] += count

        for cat in cats:
            days = float(total_days_by_cat_dict[cat])
            if days > 0:
                loss_rate = total_loss_count_by_cat_dict[cat]/days
            else:
                loss_rate = 0 # Consider it 0 if there are no samples/babies for this timepoint-
cat
            permuted_loss_rates_by_cat_tp_type_dict[cat][tp_type].append(loss_rate)

    i += 1

# Just interested in Week 0-Month 1 and Month 1-Year

fig, ax = plt.subplots(figsize=(4, 4))

tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

null_differences_by_tp_type = []
labels = []
true_differences_by_tp_type = []

for tp_type in tp_types:
    null_differences = []
    for vrate, crate in zip(permuted_loss_rates_by_cat_tp_type_dict['Vaginal'][tp_type],
                            permuted_loss_rates_by_cat_tp_type_dict['C-section'][tp_type]):
        null_differences.append(vrate-crate)
    null_differences_by_tp_type.append(null_differences)
    true_vrate = true_loss_rate_by_cat_tp_type_dict['Vaginal'][tp_type]
    true_crate = true_loss_rate_by_cat_tp_type_dict['C-section'][tp_type]
    true_difference = true_vrate-true_crate
    true_differences_by_tp_type.append(true_difference)

    p = permutation_test_p(null_differences, true_difference)
    labels.append(("%s\n" % tp_type) + r'$\it{p}$' + ("=%.03f" % (p)))

ax.violinplot(null_differences_by_tp_type, positions=[1, 2])
ax.plot(np.arange(1, 1+len(true_differences_by_tp_type)), true_differences_by_tp_type,
        marker=(5, 2), linestyle='None', markersize=12, color='black', zorder=9)
ax.set_xticks([1, 2])
ax.set_xticklabels(labels)
ax.set_ylabel("Difference between vaginal\nand C-section gene loss rate")
plt.show()
fig.savefig('%s/delivery_mode_comparison_gene_loss_rate.pdf' % config.analysis_directory)


# In[101]:


# Permutation test for feeding mode
```

```
# 1. permute C and V for all the babies
# 2. compute the SNV change rate for new C and new V babies;
# make sure to take the sum of the changes divided by the total time
# 3. compute difference in rate
# 4. Repeat x1000.
# 5. Compare true difference between C and V

num_bootstraps = 1000
cats = ['breast', 'formula']
tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

subjects = list(subject_feeding_mode_map.keys()) # Order matters!!
feeding_modes = [] # True/original
for subject in subjects:
    feeding_modes.append(subject_feeding_mode_map[subject])

true_loss_rate_by_cat_tp_type_dict = {cat: {} for cat in cats}

for tp_type in tp_types:
    total_days_by_cat_dict = defaultdict(int)
    total_loss_count_by_cat_dict = defaultdict(int)
    for subject, count, days in subject_loss_tups_by_tp_type[tp_type]:
        try:
            feeding_mode = subject_feeding_mode_map[subject]
        except:
            continue # Lots of Shao samples, for example, have NA for feeding mode
        total_days_by_cat_dict[feeding_mode] += days
        total_loss_count_by_cat_dict[feeding_mode] += count

    for cat in cats:
        days = float(total_days_by_cat_dict[cat])
        loss_rate = total_loss_count_by_cat_dict[cat]/days
        true_loss_rate_by_cat_tp_type_dict[cat][tp_type] = loss_rate

permuted_loss_rates_by_cat_tp_type_dict = {cat: defaultdict(list) for cat in cats}

i = 0
while i < num_bootstraps:
    random.shuffle(feeding_modes) # Randomly permute feeding modes
    permuted_subject_feeding_mode_map = {}
    for subject, feeding_mode in zip(subjects, feeding_modes):
        permuted_subject_feeding_mode_map[subject] = feeding_mode

    bad = False # If end up with sample size 0 for any timepoint-cat pair, skip?? Not used for
now

    # Compute SNV change rate from subject_count_time_tups_by_tp_type
    for tp_type in subject_count_time_tups_by_tp_type:
        total_days_by_cat_dict = defaultdict(int)
        total_loss_count_by_cat_dict = defaultdict(int)
        for subject, count, days in subject_count_time_tups_by_tp_type[tp_type]:
            feeding_mode = permuted_subject_feeding_mode_map[subject]
            total_days_by_cat_dict[feeding_mode] += days
            total_loss_count_by_cat_dict[feeding_mode] += count

        for cat in cats:
            days = float(total_days_by_cat_dict[cat])
            if days > 0:
                loss_rate = total_loss_count_by_cat_dict[cat]/days
            else:
```

```
                loss_rate = 0 # Consider it 0 if there are no samples/babies for this timepoint-
cat
            permuted_loss_rates_by_cat_tp_type_dict[cat][tp_type].append(loss_rate)

    i += 1

# Just interested in Week 0-Month 1 and Month 1-Year

fig, ax = plt.subplots(figsize=(4,4))

tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

null_differences_by_tp_type = []
labels = []
true_differences_by_tp_type = []

for tp_type in tp_types:
    null_differences = []
    for rate1, rate2 in zip(permuted_loss_rates_by_cat_tp_type_dict['breast'][tp_type],
                            permuted_loss_rates_by_cat_tp_type_dict['formula'][tp_type]):
        null_differences.append(rate1-rate2)
    null_differences_by_tp_type.append(null_differences)
    true_rate1 = true_loss_rate_by_cat_tp_type_dict['breast'][tp_type]
    true_rate2 = true_loss_rate_by_cat_tp_type_dict['formula'][tp_type]
    true_difference = true_rate1-true_rate2
    true_differences_by_tp_type.append(true_difference)

    p = permutation_test_p(null_differences, true_difference)
    labels.append(("%s\n" % tp_type) + r'$\it{p}$' + ("=%.03f" % (p)))

ax.violinplot(null_differences_by_tp_type, positions=[1, 2])
ax.plot(np.arange(1, 1+len(true_differences_by_tp_type)), true_differences_by_tp_type,
        marker=(5, 2), linestyle='None', markersize=12, color='black', zorder=9)
ax.set_xticks([1, 2])
ax.set_xticklabels(labels)
ax.set_ylabel("Difference between breast\nand formula feeding gene loss rate")
plt.show()
fig.savefig('%s/feeding_mode_comparison_gene_loss_rate.pdf' % config.analysis_directory)


# In[59]:


# Vaginal vs. C-section, gene change rates
# Uses previous x_tups_by_devmode_tp_type
# All based on 1000 bootstraps of size 100 subsample

devmodes = ['Vaginal', 'C-section']

# Subsample QP pairs in a category and bootstrap
bootstrapped_gain_rates_by_devmode_tp_type = {dm:
bootstrapped_agg_from_list_dict(gain_tups_by_devmode_tp_type[dm], get_rate_from_tuples,

num_bootstraps=1000, n=100) for dm in devmodes}
bootstrapped_loss_rates_by_devmode_tp_type = {dm:
bootstrapped_agg_from_list_dict(loss_tups_by_devmode_tp_type[dm], get_rate_from_tuples,

num_bootstraps=1000, n=100) for dm in devmodes}

# Gene gain/loss rates boxplot
fig, ax = plt.subplots(2, 1, figsize=(8, 8))
```

```
tp_types = ['Day 0-Week 1', 'Week 1-Month 1']

bootstrapped_gain_rates = []
for tp_type in tp_types:
    for dm in devmodes:
        bootstrapped_gain_rates.append(bootstrapped_gain_rates_by_devmode_tp_type[dm][tp_type])

bootstrapped_loss_rates = []
for tp_type in tp_types:
    for dm in devmodes:
        bootstrapped_loss_rates.append(bootstrapped_loss_rates_by_devmode_tp_type[dm][tp_type])

# These are the true "average" rates per category
gain_annotations = []; loss_annotations = []
# These are used as x-axis labels
gain_tp_type_labels = []; loss_tp_type_labels = []

for tp_type in tp_types:
    for dm in devmodes:
        # Store gain info
        gain_count, total_days = (0,0)
        for num_gains, days in gain_tups_by_devmode_tp_type[dm][tp_type]:
            gain_count += num_gains; total_days += days
        gain_annotations.append(float(gain_count)/total_days)
        gain_tp_type_labels.append('%s\nn=%i' % (tp_type,
len(gain_tups_by_devmode_tp_type[dm][tp_type])))

        # Store loss info
        loss_count, total_days = (0,0)
        for num_losss, days in loss_tups_by_devmode_tp_type[dm][tp_type]:
            loss_count += num_losss; total_days += days
        loss_annotations.append(float(loss_count)/total_days)
        loss_tp_type_labels.append('%s\nn=%i' % (tp_type,
len(loss_tups_by_devmode_tp_type[dm][tp_type])))

boxprops = dict(color='#77acff'); medianprops = dict(color='black'); flierprops =
dict(marker='.')
boxplots = ax[0].boxplot(bootstrapped_gain_rates, patch_artist=True, medianprops=medianprops,
flierprops=flierprops)
for i in np.arange(len(bootstrapped_gain_rates), step=2):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Vaginal
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # C-section

ax[0].plot(np.arange(1, 1+len(gain_annotations)), gain_annotations, marker=(5, 2),
linestyle='None', markersize=12, color='black', zorder=9)
ax[0].set_xticklabels(gain_tp_type_labels, fontsize=12)
ax[0].set_ylabel("Gene gains per day", fontsize=12)
ax[0].set_title("Shao\nGene changes per non-replacement QP pair per day\nBootstrapped (subsample
100 QP pairs 50 times)")
ax[0].set_yscale('log')

boxplots = ax[1].boxplot(bootstrapped_loss_rates, patch_artist=True, medianprops=medianprops,
flierprops=flierprops)
for i in np.arange(len(bootstrapped_loss_rates), step=2):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Vaginal
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # C-section

ax[1].plot(np.arange(1, 1+len(loss_annotations)), loss_annotations, marker=(5, 2),
linestyle='None', markersize=12, color='black', zorder=9)
ax[1].set_xticklabels(loss_tp_type_labels, fontsize=12)
```

```
ax[1].set_ylabel("Gene losses per day", fontsize=12)
ax[1].set_yscale('log')

legend_elements = [Patch(facecolor=plot_utils.col_blue, label='Vaginal'),
                   Patch(facecolor=plot_utils.col_orange, label='C-section')]
ax[0].legend(handles=legend_elements, loc='upper right', frameon=False)

plt.show()


# In[60]:


# Store for combined plot
bootstrapped_gain_rates_devmode = bootstrapped_gain_rates
gain_annotations_devmode = gain_annotations
gain_tp_type_labels_devmode = gain_tp_type_labels

bootstrapped_loss_rates_devmode = bootstrapped_loss_rates
loss_annotations_devmode = loss_annotations
loss_tp_type_labels_devmode = loss_tp_type_labels


# In[87]:


# POTENTIAL MAIN FIGURE
# C-section vs. vaginal for two main timepoint categories with enough data (mostly Shao)

# Vaginal vs. C-section, gene change rates
# Uses previous x_tups_by_devmode_tp_type
# All based on 1000 bootstraps of size 100 subsample

fig, ax = plt.subplots(2, 2, figsize=(14, 8))

devmodes = ['Vaginal', 'C-section']
tp_types = ['Day 0-Week 1', 'Week 1-Month 1']
boxprops = dict(color='#77acff'); medianprops = dict(color='black'); flierprops =
dict(marker='.')

i = 0; j = 0
boxplots = ax[i][j].boxplot(snp_change_rates_devmode, patch_artist=True,
medianprops=medianprops, flierprops=flierprops)
for k in np.arange(len(bootstrapped_loss_rates_devmode), step=2):
    boxplots['boxes'][k].set_facecolor(plot_utils.col_blue) # Vaginal
    boxplots['boxes'][k+1].set_facecolor(plot_utils.col_orange) # C-section

ax[i][j].plot(np.arange(1, 1+len(snp_change_rate_annotations_devmode)),
              snp_change_rate_annotations_devmode, marker=(5, 2), linestyle='None',
markersize=12, color='black', zorder=9)
ax[i][j].set_xticklabels(snp_change_rate_tp_type_labels_devmode, fontsize=12)
ax[i][j].set_ylabel("SNV changes per day", fontsize=12)
ax[i][j].set_yscale('log')

i = 1; j = 0
boxplots = ax[i][j].boxplot(replacement_rates_devmode, patch_artist=True,
medianprops=medianprops, flierprops=flierprops)
for k in np.arange(len(bootstrapped_loss_rates_devmode), step=2):
    boxplots['boxes'][k].set_facecolor(plot_utils.col_blue) # Vaginal
    boxplots['boxes'][k+1].set_facecolor(plot_utils.col_orange) # C-section
```

```
ax[i][j].plot(np.arange(1, 1+len(replacement_rate_annotations_devmode)),
              replacement_rate_annotations_devmode, marker=(5, 2), linestyle='None',
markersize=12, color='black', zorder=9)
ax[i][j].set_xticklabels(replacement_rate_tp_type_labels_devmode, fontsize=12)
ax[i][j].set_ylabel("Replacements per day", fontsize=12)
ax[i][j].set_yscale('log')

i = 0; j = 1
boxplots = ax[i][j].boxplot(bootstrapped_gain_rates_devmode, patch_artist=True,
medianprops=medianprops, flierprops=flierprops)
for k in np.arange(len(bootstrapped_loss_rates_devmode), step=2):
    boxplots['boxes'][k].set_facecolor(plot_utils.col_blue) # Vaginal
    boxplots['boxes'][k+1].set_facecolor(plot_utils.col_orange) # C-section

ax[i][j].plot(np.arange(1, 1+len(gain_annotations_devmode)), gain_annotations_devmode,
marker=(5, 2), linestyle='None', markersize=12, color='black', zorder=9)
ax[i][j].set_xticklabels(gain_tp_type_labels_devmode, fontsize=12)
ax[i][j].set_ylabel("Gene gains per day", fontsize=12)
ax[i][j].set_yscale('log')

i = 1; j = 1
boxplots = ax[i][j].boxplot(bootstrapped_loss_rates_devmode, patch_artist=True,
medianprops=medianprops, flierprops=flierprops)
for k in np.arange(len(bootstrapped_loss_rates_devmode), step=2):
    boxplots['boxes'][k].set_facecolor(plot_utils.col_blue) # Vaginal
    boxplots['boxes'][k+1].set_facecolor(plot_utils.col_orange) # C-section

ax[i][j].plot(np.arange(1, 1+len(loss_annotations_devmode)), loss_annotations_devmode,
marker=(5, 2), linestyle='None', markersize=12, color='black', zorder=9)
ax[i][j].set_xticklabels(loss_tp_type_labels_devmode, fontsize=12)
ax[i][j].set_ylabel("Gene losses per day", fontsize=12)
ax[i][j].set_yscale('log')

legend_elements = [Patch(facecolor=plot_utils.col_blue, label='Vaginal'),
                   Patch(facecolor=plot_utils.col_orange, label='C-section')]
ax[0][0].legend(handles=legend_elements, loc='upper right', frameon=False)

plt.tight_layout()
plt.show()

fig.savefig('%s/SX_temporal_change_rates_vaginal_vs_csection.pdf' % plot_dir,
bbox_inches='tight')


# In[73]:


def get_confidence_interval(a, percent):
    lower = np.percentile(a, ((100-percent)/2.0))
    upper = np.percentile(a, (100-((100-percent)/2.0)))
    return (lower, upper)


# In[86]:


def get_confidence_interval(a, r):
    half = r*scipy.stats.sem(a)
    avg = np.mean(a)
    lower = avg-half
    upper = avg+half
```

```
    return (lower, upper)


# In[85]:


fig, ax = plt.subplots()

i = 0
for a in snp_change_rates_devmode:
    ax.plot(get_confidence_interval(a, 1.96), [i, i], '-')
    print(get_confidence_interval(a, 1.96))
    i += 1

for a in replacement_rates_devmode:
    ax.plot(get_confidence_interval(a, 1.96), [i, i], '-')
    i += 1

for a in bootstrapped_gain_rates_devmode:
    ax.plot(get_confidence_interval(a, 1.96), [i, i], '-')
    i += 1

for a in bootstrapped_loss_rates_devmode:
    ax.plot(get_confidence_interval(a, 1.96), [i, i], '-')
    i += 1

ax.set_xscale('log')
plt.show()


# In[70]:


print("\nSNV change rate")
summarize_ttest(snp_change_rates_devmode[0], snp_change_rates_devmode[1])
print()
summarize_ttest(snp_change_rates_devmode[2], snp_change_rates_devmode[3])
print("\nReplacement rate")
summarize_ttest(replacement_rates_devmode[0], replacement_rates_devmode[1])
print()
summarize_ttest(replacement_rates_devmode[2], replacement_rates_devmode[3])
print("\nGene gain rate")
summarize_ttest(bootstrapped_gain_rates_devmode[0], bootstrapped_gain_rates_devmode[1])
print()
summarize_ttest(bootstrapped_gain_rates_devmode[2], bootstrapped_gain_rates_devmode[3])
print("\nGene loss rate")
summarize_ttest(bootstrapped_loss_rates_devmode[0], bootstrapped_loss_rates_devmode[1])
print()
summarize_ttest(bootstrapped_loss_rates_devmode[2], bootstrapped_loss_rates_devmode[3])


# In[25]:


# Store gene change information
# Split by feeding mode

gain_tups_by_cat_tp_type = {dm: defaultdict(list) for dm in cats}
loss_tups_by_cat_tp_type = {dm: defaultdict(list) for dm in cats}
genes_gained_by_cat_tp_type = {dm: defaultdict(set) for dm in cats}
```

```
for species in gene_changes:
    for s1, s2 in gene_changes[species]:

        subject = sample_subject_map[s2]
        if subject not in subject_feeding_mode_map: # Skip non-infant second sample
            continue

        feeding_mode = subject_feeding_mode_map[subject]

        subject = sample_subject_map[s2]
        feeding_mode = subject_feeding_mode_map[subject]
        gains, losses = gene_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue

        # Arbitrarily set mother 0 - infant 0 as 1 day
        if is_mother(s1) and mi_sample_day_dict[s1] == 0 and is_infant(s2) and
mi_sample_day_dict[s2] == 0:
            days = 1

        if isinstance(snp_changes[species][(s1, s2)], int): # Replacement (in terms of SNP
changes)
            continue# num_gains, num_losses = gene_changes[species][(s1, s2)]
        else:
            num_gains = len(gene_changes[species][(s1, s2)][0])
            num_losses = len(gene_changes[species][(s1, s2)][1])

        # Try excluding the Mother-Infant(wk1) QP pair with unsually high number of losses
        if (num_losses >= 13) and ('Mother-Infant' in custom_cohorts):
            print("Sample pair %s, %s | %s" % (s1, s2, species))
            print("Num losses: %i | Num gains: %i" % (num_losses, num_gains))
            continue

        for custom_cohort in custom_cohorts:
            gain_tups_by_cat_tp_type[feeding_mode][custom_cohort].append((num_gains, days))
            loss_tups_by_cat_tp_type[feeding_mode][custom_cohort].append((num_losses, days))
            for gene, _, _, _, _ in gains:
                genes_gained_by_cat_tp_type[feeding_mode][custom_cohort].add((species, gene))


# In[30]:


cats = ['breast', 'mixed', 'formula']

# Subsample QP pairs in a category to n=50 and get #sweeps/day, bootstrap 20 times
bootstrapped_gain_rates_by_cat_tp_type = {cat:
bootstrapped_agg_from_list_dict(gain_tups_by_cat_tp_type[cat], get_rate_from_tuples) for cat in
cats}
bootstrapped_loss_rates_by_cat_tp_type = {cat:
bootstrapped_agg_from_list_dict(loss_tups_by_cat_tp_type[cat], get_rate_from_tuples) for cat in
cats}

# Gene gain/loss rates boxplot
fig, ax = plt.subplots(2, 1, figsize=(20, 8))
```

```
tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1']

bootstrapped_gain_rates = []
for tp_type in tp_types:
    for cat in cats:
        bootstrapped_gain_rates.append(bootstrapped_gain_rates_by_cat_tp_type[cat][tp_type])

bootstrapped_loss_rates = []
for tp_type in tp_types:
    for cat in cats:
        bootstrapped_loss_rates.append(bootstrapped_loss_rates_by_cat_tp_type[cat][tp_type])

# These are the true "average" rates per category
gain_annotations = []; loss_annotations = []
# These are used as x-axis labels
gain_tp_type_labels = []; loss_tp_type_labels = []

for tp_type in tp_types:
    for cat in cats:
        # Store gain info
        gain_count, total_days = (0,0)
        for num_gains, days in gain_tups_by_cat_tp_type[cat][tp_type]:
            gain_count += num_gains; total_days += days
        gain_annotations.append(float(gain_count)/total_days)
        gain_tp_type_labels.append('%s\nn=%i' % (tp_type,
len(gain_tups_by_cat_tp_type[cat][tp_type])))

        # Store loss info
        loss_count, total_days = (0,0)
        for num_losss, days in loss_tups_by_cat_tp_type[cat][tp_type]:
            loss_count += num_losss; total_days += days
        loss_annotations.append(float(loss_count)/total_days)
        loss_tp_type_labels.append('%s\nn=%i' % (tp_type,
len(loss_tups_by_cat_tp_type[cat][tp_type])))

boxprops = dict(color='#77acff'); medianprops = dict(color='black'); flierprops =
dict(marker='.')

boxplots = ax[0].boxplot(bootstrapped_gain_rates, patch_artist=True,
                         medianprops=medianprops, flierprops=flierprops)

for i in np.arange(len(gain_annotations), step=3):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Breast
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # Mixed
    boxplots['boxes'][i+2].set_facecolor(plot_utils.col_darkgreen) # Formula

ax[0].plot(np.arange(1, 1+len(gain_annotations)), gain_annotations, marker=(5, 2),
linestyle='None', markersize=12)
ax[0].set_xticklabels(gain_tp_type_labels, fontsize=12)
ax[0].set_ylabel("Gene gains per day", fontsize=12)
ax[0].set_title("Gene changes per non-replacement QP pair per day\nBootstrapped (subsample QP
pairs 50 times)")
ax[0].set_yscale('log')

boxplots = ax[1].boxplot(bootstrapped_loss_rates, patch_artist=True,
                         medianprops=medianprops, flierprops=flierprops)

for i in np.arange(len(loss_annotations), step=3):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Breast
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # Mixed
    boxplots['boxes'][i+2].set_facecolor(plot_utils.col_darkgreen) # Formula
```

```
ax[1].plot(np.arange(1, 1+len(loss_annotations)), loss_annotations, marker=(5, 2),
linestyle='None', markersize=12)
ax[1].set_xticklabels(loss_tp_type_labels, fontsize=12)
ax[1].set_ylabel("Gene losses per day", fontsize=12)
ax[1].set_yscale('log')

legend_elements = [Patch(facecolor=plot_utils.col_blue, label='Breast'),
                   Patch(facecolor=plot_utils.col_orange, label='Mixed'),
                   Patch(facecolor=plot_utils.col_darkgreen, label='Formula')]
ax[1].legend(handles=legend_elements, loc='upper right', frameon=False)

plt.show()


# In[40]:


# quick overview of delivery mode info

cohort_devmode_dict = {cohort: {dm: 0 for dm in devmodes} for cohort in cohorts}

for sample in infant_samples:
    cohort = sample_cohort_map[sample]
    devmode = subject_delivery_mode_map[sample_subject_map[sample]]
    cohort_devmode_dict[cohort][devmode] += 1

cohort_devmode_dict


# In[42]:


# Combined matched plot
# SUPPLEMENTAL FIGURE
# OLD version with 50 bootstraps

fig, ax = plt.subplots(1, 3, figsize=(12, 4))

tp_types = ['4-8mon duration infant', '4-8mon duration adult']
labels = ['4-8mon\nInfant', '4-8mon\nAdult']

# ====================
# SNP CHANGES
# ====================

boxplots_0 = ax[0].boxplot(snp_change_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))

for i in np.arange(len(snp_change_rates)):
    boxplots_0['boxes'][i].set_facecolor('#77acff')

ax[0].plot(np.arange(1, 1+len(snp_change_rate_annotations)), snp_change_rate_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=12)
ax[0].set_xticklabels(['\n'.join(label.split('\n')[1:]) for label in
snp_change_rate_tp_type_labels], fontsize=12)
ax[0].set_ylabel("SNV changes per day", fontsize=12)
ax[0].text(-0.35, 0.93, 'A', size=20, transform=ax[0].transAxes, weight='bold')

print("SNV change rate, infant vs adult")
```

```
summarize_utest(snp_change_rates[0], snp_change_rates[1]); print()
summarize_ttest(snp_change_rates[0], snp_change_rates[1]); print()

# ====================
# Replacements
# ====================

boxplots_1 = ax[1].boxplot(replacement_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))

for i in np.arange(len(replacement_rates)):
    boxplots_1['boxes'][i].set_facecolor('#77acff')

ax[1].plot(np.arange(1, 1+len(replacement_rate_annotations)), replacement_rate_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=12)
ax[1].set_xticklabels(['\n'.join(label.split('\n')[1:]) for label in
replacement_rate_tp_type_labels], fontsize=12)
ax[1].set_ylabel("Replacements per day", fontsize=12)
ax[1].text(-0.35, 0.93, 'B', size=20, transform=ax[1].transAxes, weight='bold')

print("Replacement rate, infant vs adult")
summarize_utest(replacement_rates[0], replacement_rates[1]); print()
summarize_ttest(replacement_rates[0], replacement_rates[1]); print()

# ====================
# Gene gain/loss
# ====================
bootstrapped_all_gene_change_rates = []
for gain_rates, loss_rates in zip(bootstrapped_gain_rates, bootstrapped_loss_rates):
    bootstrapped_all_gene_change_rates.append(gain_rates)
    bootstrapped_all_gene_change_rates.append(loss_rates)

all_gene_change_annotations = []
for gain_annotation, loss_annotation in zip(gain_annotations, loss_annotations):
    all_gene_change_annotations.append(gain_annotation)
    all_gene_change_annotations.append(loss_annotation)

boxplots_2 = ax[2].boxplot(bootstrapped_all_gene_change_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))

for i in np.arange(len(bootstrapped_all_gene_change_rates), step=2):
    boxplots_2['boxes'][i].set_facecolor('#b3de69') # gain
    boxplots_2['boxes'][i+1].set_facecolor('#ff7f00') # loss

print("Gene gain rate, infant vs adult")
summarize_utest(bootstrapped_all_gene_change_rates[0], bootstrapped_all_gene_change_rates[2]);
print()
summarize_ttest(bootstrapped_all_gene_change_rates[0], bootstrapped_all_gene_change_rates[2]);
print()

print("Gene loss rate, infant vs adult")
summarize_utest(bootstrapped_all_gene_change_rates[1], bootstrapped_all_gene_change_rates[3]);
print()
summarize_ttest(bootstrapped_all_gene_change_rates[1], bootstrapped_all_gene_change_rates[3]);
print()

ax[2].plot(np.arange(1, 1+len(all_gene_change_annotations)), all_gene_change_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=12)
ax[2].set_xticks(np.arange(2*len(gain_tp_type_labels), step=2) + 1.5)
```

```
ax[2].set_xticklabels(['\n'.join(label.split('\n')[1:]) for label in gain_tp_type_labels],
fontsize=12)
ax[2].set_ylabel("Gene changes per day", fontsize=12)
ax[2].set_yscale('log')
ax[2].text(-0.35, 0.93, 'C', size=20, transform=ax[2].transAxes, weight='bold')

legend_elements = [Patch(facecolor='#b3de69', label='Gain'), Patch(facecolor='#ff7f00',
label='Loss')]

ax[2].legend(handles=legend_elements, loc='upper right', frameon=False)

plt.subplots_adjust(wspace=0.42)
plt.show()

fig.savefig('%s/S8_snv_gene_change_replacement_matched_infant_adult_compare.pdf' % plot_dir,
bbox_inches='tight')


# In[55]:


# Combined matched plot
# SUPPLEMENTAL FIGURE
# Updated VERSION with 1000 bootstraps
# Also add tests (MWU) ok scratched

fig, ax = plt.subplots(1, 3, figsize=(14, 4), gridspec_kw={'width_ratios': [1, 1, 2]})

tp_types = ['4-8mon duration infant', '4-8mon duration adult']
labels = ['4-8mon\nInfant', '4-8mon\nAdult']

# ====================
# SNP CHANGES
# ====================

boxplots_0 = ax[0].boxplot(snp_change_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'), widths=0.5)

for i in np.arange(len(snp_change_rates)):
    boxplots_0['boxes'][i].set_facecolor('#77acff')

ax[0].plot(np.arange(1, 1+len(snp_change_rate_annotations)), snp_change_rate_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=12)
ax[0].set_xticklabels(['\n'.join(label.split('\n')[1:]) for label in
snp_change_rate_tp_type_labels], fontsize=12)
ax[0].set_ylabel("SNV changes per day", fontsize=12)
# ax[0].set_yscale('log')
ax[0].text(-0.53, 0.93, 'A', size=20, transform=ax[0].transAxes, weight='bold')
'''
print("SNV change rate, infant vs adult")
U, p, es = summarize_utest(snp_change_rates[0], snp_change_rates[1]); print()
summarize_ttest(snp_change_rates[0], snp_change_rates[1]); print()

i1, i2 = (0, 1); j = 0.02
color = 'red' if p < 0.05 else 'gray'; offset = 0.36 if p < 0.01 else 0.23
plot_interval_on_ax(ax[0], j, i1+1, i2+1, color=color, tickh=0.0008); label = '{:.1e}'.format(p)
if p < 0.01 else ('%.03f' % p)
ax[0].text(i1+1+((i2-i1)/2)-offset, j+0.0005, label, color=color)
'''
# ====================
```

```
# Replacements
# ====================

boxplots_1 = ax[1].boxplot(replacement_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'), widths=0.5)

for i in np.arange(len(replacement_rates)):
    boxplots_1['boxes'][i].set_facecolor('#77acff')

ax[1].plot(np.arange(1, 1+len(replacement_rate_annotations)), replacement_rate_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=12)
ax[1].set_xticklabels(['\n'.join(label.split('\n')[1:]) for label in
replacement_rate_tp_type_labels], fontsize=12)
ax[1].set_ylabel("Replacements per day", fontsize=12)
# ax[1].set_yscale('log')
ax[1].text(-0.6, 0.93, 'B', size=20, transform=ax[1].transAxes, weight='bold')
'''
print("Replacement rate, infant vs adult")
U, p, es = summarize_utest(replacement_rates[0], replacement_rates[1]); print()
summarize_ttest(replacement_rates[0], replacement_rates[1]); print()

i1, i2 = (0, 1); j = 0.0025
color = 'red' if p < 0.05 else 'gray'; offset = 0.36 if p < 0.01 else 0.23
plot_interval_on_ax(ax[1], j, i1+1, i2+1, color=color, tickh=0.0001); label = '{:.1e}'.format(p)
if p < 0.01 else ('%.03f' % p)
ax[1].text(i1+1+((i2-i1)/2)-offset, j+0.0001, label, color=color)
'''
# ====================
# Gene gain/loss
# ====================
bootstrapped_all_gene_change_rates = []
for gain_rates, loss_rates in zip(bootstrapped_gain_rates, bootstrapped_loss_rates):
    bootstrapped_all_gene_change_rates.append(gain_rates)
    bootstrapped_all_gene_change_rates.append(loss_rates)

all_gene_change_annotations = []
for gain_annotation, loss_annotation in zip(gain_annotations, loss_annotations):
    all_gene_change_annotations.append(gain_annotation)
    all_gene_change_annotations.append(loss_annotation)

boxplots_2 = ax[2].boxplot(bootstrapped_all_gene_change_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'), widths=0.5)

for i in np.arange(len(bootstrapped_all_gene_change_rates), step=2):
    boxplots_2['boxes'][i].set_facecolor('#b3de69') # gain
    boxplots_2['boxes'][i+1].set_facecolor('#ff7f00') # loss
'''
print("Gene gain rate, infant vs adult")
U, p, es = summarize_utest(bootstrapped_all_gene_change_rates[0],
bootstrapped_all_gene_change_rates[2]); print()
summarize_ttest(bootstrapped_all_gene_change_rates[0], bootstrapped_all_gene_change_rates[2]);
print()

i1, i2 = (0, 1); j = 0.2
color = 'red' if p < 0.05 else 'gray'; offset = 0.36 if p < 0.01 else 0.23
plot_interval_on_ax(ax[2], j, i1+1, i2+1, color=color, tickh=0.02); label = '{:.1e}'.format(p)
if p < 0.01 else ('%.03f' % p)
ax[2].text(i1+1+((i2-i1)/2)-offset, j+0.04, label, color=color)
```

```
print("Gene loss rate, infant vs adult")
U, p, es = summarize_utest(bootstrapped_all_gene_change_rates[1],
bootstrapped_all_gene_change_rates[3]); print()
summarize_ttest(bootstrapped_all_gene_change_rates[1], bootstrapped_all_gene_change_rates[3]);
print()

i1, i2 = (2, 3); j = 0.02
color = 'red' if p < 0.05 else 'gray'; offset = 0.36 if p < 0.01 else 0.23
plot_interval_on_ax(ax[2], j, i1+1, i2+1, color=color, tickh=0.001); label = '{:.1e}'.format(p)
if p < 0.01 else ('%.03f' % p)
ax[2].text(i1+1+((i2-i1)/2)-offset, j+0.005, label, color=color)
'''
ax[2].plot(np.arange(1, 1+len(all_gene_change_annotations)), all_gene_change_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=12)
ax[2].set_xticks(np.arange(2*len(gain_tp_type_labels), step=2) + 1.5)
ax[2].set_xticklabels(['\n'.join(label.split('\n')[1:]) for label in gain_tp_type_labels],
fontsize=12)
ax[2].set_ylabel("Gene changes per day", fontsize=12)
ax[2].set_yscale('log')
ax[2].text(-0.28, 0.93, 'C', size=20, transform=ax[2].transAxes, weight='bold')

legend_elements = [Patch(facecolor='#b3de69', label='Gain'), Patch(facecolor='#ff7f00',
label='Loss')]

ax[2].legend(handles=legend_elements, loc='upper right', frameon=False)

plt.subplots_adjust(wspace=1) # Will do final formatting later
plt.show()

fig.savefig('%s/S8_snv_gene_change_replacement_matched_infant_adult_compare.pdf' % plot_dir,
bbox_inches='tight')


# In[41]:


# QQ plots
from scipy import stats

fig, ax = plt.subplots(2, 4, figsize=(14, 8))
xs = np.arange(-3, 3, 0.01)

vals = stats.probplot(snp_change_rates[0]); ax[0][0].set_title('SNV change rate\nInfant 4-8mon
duration')
osm, osr = vals[0]; m, b, r = vals[1]
ax[0][0].plot(xs, (m*xs)+b, color='gray'); ax[0][0].plot(osm, osr, '.', color='black',
mfc='none')

vals = stats.probplot(snp_change_rates[1]); ax[0][1].set_title('SNV change rate\nAdult 4-8mon
duration')
osm, osr = vals[0]; m, b, r = vals[1]
ax[0][1].plot(xs, (m*xs)+b, color='gray'); ax[0][1].plot(osm, osr, '.', color='black',
mfc='none')

vals = stats.probplot(replacement_rates[0]); ax[0][2].set_title('Replacement rate\nInfant 4-8mon
duration')
osm, osr = vals[0]; m, b, r = vals[1]
ax[0][2].plot(xs, (m*xs)+b, color='gray'); ax[0][2].plot(osm, osr, '.', color='black',
mfc='none')
```

```
vals = stats.probplot(replacement_rates[1]); ax[0][3].set_title('Replacement rate\nAdult 4-8mon
duration')
osm, osr = vals[0]; m, b, r = vals[1]
ax[0][3].plot(xs, (m*xs)+b, color='gray'); ax[0][3].plot(osm, osr, '.', color='black',
mfc='none')

labels = ['Gene gain rate\nInfant 4-8mon duration', 'Gene loss rate\nInfant 4-8mon duration',
          'Gene gain rate\nAdult 4-8mon duration', 'Gene loss rate\nAdult 4-8mon duration']
for i, label in zip(range(len(bootstrapped_all_gene_change_rates)), labels):
    vals = stats.probplot(bootstrapped_all_gene_change_rates[i])
    ax[1][i].set_title(label)
    osm, osr = vals[0]; m, b, r = vals[1]
    ax[1][i].plot(xs, (m*xs)+b, color='gray')
    ax[1][i].plot(osm, osr, '.', color='black', mfc='none')
    ax[1][i].set_xlabel("Theoretical quantile")

ax[0][0].set_ylabel("Observed")
ax[1][0].set_ylabel("Observed")

plt.tight_layout()
fig.savefig('%s/SX_rates_4-8mon_qq_plots.pdf' % plot_dir, bbox_inches='tight')


# In[27]:


# Get t test statistics
t, p = stats.ttest_ind(snp_change_rates[0], snp_change_rates[1])
print("T: %.02f\tp: %.3E" % (t, p))

t, p = stats.ttest_ind(replacement_rates[0], replacement_rates[1])
print("T: %.02f\tp: %.3E" % (t, p))

t, p = stats.ttest_ind(bootstrapped_gain_rates[0], bootstrapped_gain_rates[1])
print("T: %.02f\tp: %.3E" % (t, p))

t, p = stats.ttest_ind(bootstrapped_loss_rates[0], bootstrapped_loss_rates[1])
print("T: %.02f\tp: %.3E" % (t, p))


# In[28]:


# Get Wilcoxon rank sum test statistics (two-sided is default)
U, p = stats.mannwhitneyu(snp_change_rates[0], snp_change_rates[1])
print("U: %.02f\tp: %.3E" % (U, p))

U, p = stats.mannwhitneyu(replacement_rates[0], replacement_rates[1])
print("U: %.02f\tp: %.3E" % (U, p))

U, p = stats.mannwhitneyu(bootstrapped_gain_rates[0], bootstrapped_gain_rates[1])
print("U: %.02f\tp: %.3E" % (U, p))

U, p = stats.mannwhitneyu(bootstrapped_loss_rates[0], bootstrapped_loss_rates[1])
print("U: %.02f\tp: %.3E" % (U, p))


# In[29]:


# Get statistics for paper
```

```
for tp_type, gain_val, loss_val in zip(tp_types, gain_annotations, loss_annotations):
    print(tp_type)
    print('Avg gain rate: %.3E \t Avg loss rate: %.3E' % (gain_val, loss_val))


# In[26]:


# Combined SNP change, replacement, gene gain and loss rate plot: final version??

fig, ax = plt.subplots(3, 1, figsize=(8, 10))

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

boxplots_0 = ax[0].boxplot(snp_change_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))

for i in np.arange(len(snp_change_rates)):
    boxplots_0['boxes'][i].set_facecolor('#77acff')

ax[0].plot(np.arange(1, 1+len(snp_change_rate_annotations)), snp_change_rate_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=10)
ax[0].set_xticklabels(snp_change_rate_tp_type_labels)
ax[0].set_ylabel("SNP changes\nper day", fontsize=12)
# ax[0].set_title("SNP changes per non-replacement QP pair per day")
ax[0].set_yscale('log')
ax[0].text(-0.12, 0.95, 'A', size=20, transform=ax[0].transAxes, weight='bold')

boxplots_1 = ax[1].boxplot(replacement_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))

for i in np.arange(len(replacement_rates)):
    boxplots_1['boxes'][i].set_facecolor('#77acff')

ax[1].plot(np.arange(1, 1+len(replacement_rate_annotations)), replacement_rate_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=10)
ax[1].set_xticklabels(replacement_rate_tp_type_labels)
ax[1].set_ylabel("Replacements\nper day", fontsize=12)
# ax[1].set_title("Replacements per QP pair per day")
ax[1].set_yscale('log')
ax[1].text(-0.12, 0.95, 'B', size=20, transform=ax[1].transAxes, weight='bold')

bootstrapped_all_gene_change_rates = []
for gain_rates, loss_rates in zip(bootstrapped_gain_rates, bootstrapped_loss_rates):
    bootstrapped_all_gene_change_rates.append(gain_rates)
    bootstrapped_all_gene_change_rates.append(loss_rates)

all_gene_change_annotations = []
for gain_annotation, loss_annotation in zip(gain_annotations, loss_annotations):
    all_gene_change_annotations.append(gain_annotation)
    all_gene_change_annotations.append(loss_annotation)

boxplots_2 = ax[2].boxplot(bootstrapped_all_gene_change_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))
for i in np.arange(len(bootstrapped_all_gene_change_rates), step=2):
    boxplots_2['boxes'][i].set_facecolor(plot_utils.col_gain) # gain
    boxplots_2['boxes'][i+1].set_facecolor(plot_utils.col_loss) # loss
```

```
ax[2].plot(np.arange(1, 1+len(all_gene_change_annotations)), all_gene_change_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=10)
ax[2].set_xticks(np.arange(2*len(gain_tp_type_labels), step=2) + 1.5)
ax[2].set_xticklabels(gain_tp_type_labels)
ax[2].set_ylabel("Gene changes\nper day", fontsize=12)
# ax[2].set_title("Gene changes per non-replacement QP pair per day")
ax[2].set_yscale('log')
ax[2].text(-0.12, 0.95, 'C', size=20, transform=ax[2].transAxes, weight='bold')

legend_elements = [Patch(facecolor=plot_utils.col_gain, label='Gain'),
Patch(facecolor=plot_utils.col_loss, label='Loss')]

ax[2].legend(handles=legend_elements, loc='upper right', frameon=False)

plt.subplots_adjust(wspace=0, hspace=0.2)
plt.show()

fig.savefig('%s/figure_3_v1.png' % plot_dir, bbox_inches='tight', dpi=600)


# In[29]:


print(snp_change_rates[2])
# print(replacement_rates[1])
# print(bootstrapped_all_gene_change_rates[1])


# In[28]:


# QQ plots
from scipy import stats

fig, ax = plt.subplots(2, 5, figsize=(16, 8))
xs = np.arange(-3, 3, 0.01)

for i, tp_type in zip(range(len(bootstrapped_gain_rates)), tp_types):
    vals = stats.probplot(bootstrapped_gain_rates[i])
    ax[0][i].set_title(tp_type)
    osm, osr = vals[0]; m, b, r = vals[1]
    ax[0][i].plot(xs, (m*xs)+b, color='gray')
    ax[0][i].plot(osm, osr, '.', color='black', mfc='none')

for i, tp_type in zip(range(len(bootstrapped_loss_rates)), tp_types):
    vals = stats.probplot(bootstrapped_loss_rates[i])
    ax[1][i].set_title(tp_type)
    osm, osr = vals[0]; m, b, r = vals[1]
    ax[1][i].plot(xs, (m*xs)+b, color='gray')
    ax[1][i].plot(osm, osr, '.', color='black', mfc='none')
    ax[1][i].set_xlabel("Theoretical quantile")

ax[0][0].set_ylabel("Observed\ngene gain rate")
ax[1][0].set_ylabel("Observed\ngene loss rate")

plt.tight_layout()


# In[37]:
```

```
gain_tp_type_labels


# In[29]:


# Get statistics for paper
for tp_type, gain_val, loss_val in zip(tp_types, gain_annotations, loss_annotations):
    print(tp_type)
    print('Avg gain rate: %.3E \t Avg loss rate: %.3E' % (gain_val, loss_val))

print('')

t, p = stats.ttest_ind(bootstrapped_gain_rates[0], bootstrapped_loss_rates[0])
print("Gain vs loss, mother-infant transition")
summarize_ttest(bootstrapped_gain_rates[0], bootstrapped_loss_rates[0])
# print("T: %.02f\tp: %.3E" % (t, p))

print()

t, p = stats.ttest_ind(bootstrapped_gain_rates[3], bootstrapped_loss_rates[3])
print("Gain vs loss, month 1-year 1")
summarize_ttest(bootstrapped_gain_rates[3], bootstrapped_loss_rates[3])
# print("T: %.02f\tp: %.3E" % (t, p))


# In[31]:


# Combined SNP change, replacement, gene gain and loss rate plot: final version??

fig, ax = plt.subplots(2, 1, figsize=(10, 8))

tp_types = ['MI', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'AA']

bootstrapped_snp_change_and_replacement_rates = []
for sc_rates_cat, r_rates_cat in zip(snp_change_rates, replacement_rates):
    bootstrapped_snp_change_and_replacement_rates.append(sc_rates_cat)
    bootstrapped_snp_change_and_replacement_rates.append(r_rates_cat)

snp_change_and_replacement_annotations = []
for sc_annotation, r_annotation in zip(snp_change_rate_annotations,
replacement_rate_annotations):
    snp_change_and_replacement_annotations.append(sc_annotation)
    snp_change_and_replacement_annotations.append(r_annotation)

boxplots_0 = ax[0].boxplot(bootstrapped_snp_change_and_replacement_rates, patch_artist=True,
zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))
for i in np.arange(len(bootstrapped_snp_change_and_replacement_rates), step=2):
    boxplots_0['boxes'][i].set_facecolor('#77acff') # SNP change
    boxplots_0['boxes'][i+1].set_facecolor('#e7755b') # Replacement

ax[0].plot(np.arange(1, 1+len(snp_change_and_replacement_annotations)),
snp_change_and_replacement_annotations, marker=(5, 2), color='black', linestyle='None',
markersize=10)
ax[0].set_xticks(np.arange(2*len(snp_change_rate_tp_type_labels), step=2) + 1.5)
ax[0].set_xticklabels(snp_change_rate_tp_type_labels)
ax[0].set_ylabel("SNP changes /\nReplacements\nper day", fontsize=12)
ax[0].set_yscale('log')
```

```
ax[0].text(-0.12, 0.95, 'A', size=20, transform=ax[0].transAxes, weight='bold')

legend_elements = [Patch(facecolor='#77acff', label='Modification SNP changes'),
Patch(facecolor='#e7755b', label='Replacements')]

ax[0].legend(handles=legend_elements, loc='upper right', frameon=False)

bootstrapped_all_gene_change_rates = []
for gain_rates, loss_rates in zip(bootstrapped_gain_rates, bootstrapped_loss_rates):
    bootstrapped_all_gene_change_rates.append(gain_rates)
    bootstrapped_all_gene_change_rates.append(loss_rates)

all_gene_change_annotations = []
for gain_annotation, loss_annotation in zip(gain_annotations, loss_annotations):
    all_gene_change_annotations.append(gain_annotation)
    all_gene_change_annotations.append(loss_annotation)

boxplots_1 = ax[1].boxplot(bootstrapped_all_gene_change_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))
for i in np.arange(len(bootstrapped_all_gene_change_rates), step=2):
    boxplots_1['boxes'][i].set_facecolor('#b3de69') # gain
    boxplots_1['boxes'][i+1].set_facecolor('#ff7f00') # loss

ax[1].plot(np.arange(1, 1+len(all_gene_change_annotations)), all_gene_change_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=10)
ax[1].set_xticks(np.arange(2*len(gain_tp_type_labels), step=2) + 1.5)
ax[1].set_xticklabels(gain_tp_type_labels)
ax[1].set_ylabel("Gene changes\nper day", fontsize=12)
ax[1].set_yscale('log')
ax[1].text(-0.12, 0.95, 'B', size=20, transform=ax[1].transAxes, weight='bold')

legend_elements = [Patch(facecolor='#b3de69', label='Gain'), Patch(facecolor='#ff7f00',
label='Loss')]

ax[1].legend(handles=legend_elements, loc='upper right', frameon=False)

plt.subplots_adjust(wspace=0, hspace=0.2)
plt.show()

fig.savefig('%s/figure_3_v2.png' % plot_dir, bbox_inches='tight', dpi=600)


# In[57]:


custom_cohort_tests


# In[23]:


# Plot SNP change distribution with all infant-infant QP pairs combined
'''
def custom_cohort_of_sample_pair(custom_cohort_tests, sample_i, sample_j):
    for cohort in custom_cohort_tests:
        if custom_cohort_tests[cohort](sample_i, sample_j):
            return cohort

custom_cohort_tests = {}
```

```
# custom_cohort_tests['MI'] = lambda sample_i, sample_j: (sample_i in mother_samples and
sample_j in infant_samples)
custom_cohort_tests['MI'] = lambda sample_i, sample_j: ((sample_i in mother_samples and sample_j
in infant_samples) and mi_sample_day_dict[sample_i] >= 0 and mi_sample_day_dict[sample_j] <= 7)
custom_cohort_tests['II-1mon'] = lambda sample_i, sample_j: ((sample_i in infant_samples and
sample_j in infant_samples) and (sample_pair_to_days(sample_i, sample_j) <= 32))
custom_cohort_tests['II-1yr'] = lambda sample_i, sample_j: ((sample_i in infant_samples and
sample_j in infant_samples) and (sample_pair_to_days(sample_i, sample_j) > 90))
custom_cohort_tests['MM'] = lambda sample_i, sample_j: (sample_i in mother_samples and sample_j
in mother_samples)
custom_cohort_tests['AA'] = lambda sample_i, sample_j: (sample_i in hmp_samples and sample_j in
hmp_samples)
'''
pooled_snp_change_distribution = defaultdict(list)
pooled_between_snp_change_distribution = defaultdict(list)

time_length_dist = []

for species in snp_changes:
    for sample_i, sample_j in snp_changes[species]:
        val = snp_changes[species][(sample_i, sample_j)]
        num_snp_changes = val if (type(val) == type(1)) else len(val)
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j,
species)
        if 'II-1yr' in custom_cohorts:
            time_length_dist.append(sample_pair_to_days(sample_i, sample_j))
        for custom_cohort in custom_cohorts:
            pooled_snp_change_distribution[custom_cohort].append(num_snp_changes)

        num_snp_changes_between = between_snp_change_counts[species][(sample_i, sample_j)]
        if sample_i in hmp_samples and sample_j in hmp_samples:
            pooled_between_snp_change_distribution['AA'].append(num_snp_changes_between)
        else:
            pooled_between_snp_change_distribution['MI'].append(num_snp_changes_between)


# In[24]:


pooled_snp_change_distribution.keys()


# In[49]:


# Old and scrapped custom cohort code
'''
custom_cohort_tests = {}
custom_cohort_tests['MI'] = lambda sample_i, sample_j: ((sample_i in mother_samples and sample_j
in infant_samples) and mi_sample_day_dict[sample_i] >= 0 and mi_sample_day_dict[sample_j] <= 7)
custom_cohort_tests['MI-other'] = lambda sample_i, sample_j: ((sample_i in mother_samples and
sample_j in infant_samples))
custom_cohort_tests['II'] = lambda sample_i, sample_j: (sample_i in infant_samples and sample_j
in infant_samples)
custom_cohort_tests['MM'] = lambda sample_i, sample_j: (sample_i in mother_samples and sample_j
in mother_samples)
custom_cohort_tests['AA'] = lambda sample_i, sample_j: (sample_i in hmp_samples and sample_j in
hmp_samples)
'''
```

```
# In[25]:


# Get statistics for paper

pooled_snp_change_distribution = defaultdict(list)
pooled_between_snp_change_distribution = defaultdict(list)

time_length_dist = []

for species in snp_changes:
    for sample_i, sample_j in snp_changes[species]:

        val = snp_changes[species][(sample_i, sample_j)]
        num_snp_changes = val if (type(val) == type(1)) else len(val)
        num_snp_changes_between = between_snp_change_counts[species][(sample_i, sample_j)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j,
species)
        '''
        if custom_cohort == None:
            print(sample_order_map[sample_i])
            print(sample_order_map[sample_j])
        '''
        time_length_dist.append(sample_pair_to_days(sample_i, sample_j))

        for custom_cohort in custom_cohorts:
            pooled_snp_change_distribution[custom_cohort].append(num_snp_changes)

pooled_between_snp_change_distribution[custom_cohort].append(num_snp_changes_between)

'''
Most QP sample pairs experience zero SNV changes over timescales of less than a week,
but a small percentage undergo a small number of SNV changes (< 20).
An even smaller percentage of hosts harbor ~104 SNV differences,
which is on the same order of magnitude of the number of SNV differences between unrelated hosts
'''

print("\nWITHIN\n")
for tp_type in pooled_snp_change_distribution:
    print(tp_type)
    num_total = len(pooled_snp_change_distribution[tp_type])
    num_zero = sum(np.array(pooled_snp_change_distribution[tp_type]) == 0)
    num_small = sum(np.array(pooled_snp_change_distribution[tp_type]) < 20)
    num_mod = num_small - num_zero
    num_large = sum(np.array(pooled_snp_change_distribution[tp_type]) > 500)
    print("%i out of %i (%.03f) QP pairs have 0 SNP diffs" % (num_zero, num_total,
float(num_zero)/num_total))
    print("%i out of %i (%.03f) QP pairs have less than 20 SNP diffs" % (num_small, num_total,
float(num_small)/num_total))
    print("%i out of %i (%.03f) QP pairs have 1-20 SNP diffs" % (num_mod, num_total,
float(num_mod)/num_total))
    print("%i out of %i (%.03f) QP pairs have greater than 500 SNP diffs" % (num_large,
num_total, float(num_large)/num_total))
    print('')

print("\nBETWEEN\n")
for tp_type in pooled_between_snp_change_distribution:
    print(tp_type)
    num_total = len(pooled_between_snp_change_distribution[tp_type])
    num_zero = sum(np.array(pooled_between_snp_change_distribution[tp_type]) == 0)
    num_small = sum(np.array(pooled_between_snp_change_distribution[tp_type]) < 20)
```

```
    num_mod = num_small - num_zero
    num_large = sum(np.array(pooled_between_snp_change_distribution[tp_type]) > 500)
    print("%i out of %i (%.03f) QP pairs have 0 SNP diffs" % (num_zero, num_total,
float(num_zero)/num_total))
    print("%i out of %i (%.03f) QP pairs have less than 20 SNP diffs" % (num_small, num_total,
float(num_small)/num_total))
    print("%i out of %i (%.03f) QP pairs have 1-20 SNP diffs" % (num_mod, num_total,
float(num_mod)/num_total))
    print("%i out of %i (%.03f) QP pairs have greater than 500 SNP diffs" % (num_large,
num_total, float(num_large)/num_total))
    print('')


# In[35]:


# Understand durations and initial timepoints of II-1mon and II-1yr categories

custom_cohort_durations = defaultdict(list)
custom_cohort_day1s = defaultdict(list)

for species in snp_changes:
    for sample_i, sample_j in snp_changes[species]:

        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j,
species)
        days = sample_pair_to_days(sample_i, sample_j)
        day1 = mi_sample_day_dict[sample_i] if sample_i in mi_sample_day_dict else -1

        for custom_cohort in custom_cohorts:
            custom_cohort_durations[custom_cohort].append(days)
            custom_cohort_day1s[custom_cohort].append(day1)


# In[51]:


# Print histogram for list of integers
def print_histogram(vals):
    for val in sorted(list(set(vals))):
        print("%i\t%i" % (val, vals.count(val)))
    print('')

print("============================================")
print("II-1mon category")
print("============================================")
print("Day 1\tCount")
print_histogram(custom_cohort_day1s['II-1mon'])

plt.hist(custom_cohort_durations['II-1mon'], bins=20)
plt.xlabel("Duration")
plt.show()


# In[54]:


print("============================================")
print("II-3mon category")
print("============================================")
print("Day 1\tCount")
```

```
print_histogram(custom_cohort_day1s['II-3mon'])

plt.hist(custom_cohort_durations['II-3mon'], bins=20)
plt.xlabel("Duration")
plt.show()


# In[53]:


print("============================================")
print("II-1yr category")
print("============================================")

print("Day 1\tCount")
print_histogram(custom_cohort_day1s['II-1yr'])

plt.hist(custom_cohort_durations['II-1yr'], bins=20)
plt.xlabel("Duration")
plt.show()


# In[26]:


# Plot SNP change distribution survival curve plot

fig_snp, ax_snp = plt.subplots(figsize=(5,3))

num_colors = 8
colormap = cmx.get_cmap('viridis', num_colors)
colors = [colormap(x) for x in np.array([x for x in range(0,num_colors)])/float(num_colors)]
colors = ['gray', 'blue', '#00f0b6', '#7bb551', '#0e750e']

modification_difference_threshold = config.modification_difference_threshold
replacement_difference_threshold = config.replacement_difference_threshold

ax_snp.set_xscale('log')
ax_snp.set_yscale('log')
ax_snp.set_ylabel('Fraction comparisons $\geq n$', fontsize=11)
ax_snp.set_xlabel('# SNP changes', fontsize=11)

ax_snp.spines['top'].set_visible(False)
ax_snp.spines['right'].set_visible(False)
ax_snp.get_xaxis().tick_bottom()
ax_snp.get_yaxis().tick_left()

color_i = 0

ymin, ymax = 0.0001, 1.3
ax_snp.set_ylim([ymin,ymax])
xmin, xmax = 1e-01, 2e05
ax_snp.set_xlim([xmin, xmax])

# Now fill in the graphics

ax_snp.fill_between([xmin,1], [ymin,ymin],[ymax,ymax],color='0.8',zorder=1)
ax_snp.fill_between([1,modification_difference_threshold],[ymin,ymin],[ymax,ymax],color='#deebf7
',zorder=1)
ax_snp.fill_between([replacement_difference_threshold,xmax],[ymin,ymin],[ymax,ymax],color='#fee0
d2',zorder=1)
```

```
ax_snp.text( exp((log(1e05)+log(replacement_difference_threshold))/2), ymax*1.2,
'putative\nreplacement',fontsize=12,fontstyle='italic',ha='center',color='#fc9272',zorder=1)
ax_snp.text( exp((log(1)+log(modification_difference_threshold))/2), ymax*1.2,
'putative\nmodification',fontsize=12,fontstyle='italic',ha='center',color='#9ecae1',zorder=1)

# Unrelated hosts (adults OR mother/infant)
counts = []
for cat in ['Adult-Adult', 'Mother-Infant', 'II-1mon', 'II-1yr']:
    counts += pooled_between_snp_change_distribution[cat]
# counts = pooled_between_snp_change_distribution['Adult-Adult'] +
pooled_between_snp_change_distribution['Mother-Infant'] +
pooled_between_snp_change_distribution['Infant-Infant']
xs, ns = calculate_unnormalized_survival_from_vector(counts)
ax_snp.step(xs,ns/float(ns[0]),'-',color=colors[color_i],linewidth=1.4, label="Unrelated hosts",
where='pre',zorder=4)
color_i += 1

# Within-host, adult
counts = pooled_snp_change_distribution['Adult-Adult']
xs, ns = calculate_unnormalized_survival_from_vector(counts)
mlabel = 'Adult-adult, ~6 months' + ('\n(n=%d)' % ns[0])
ax_snp.step(xs,ns/float(ns[0]),'-',color=colors[color_i],linewidth=1.4, label=mlabel,
where='pre',zorder=4)
color_i += 1

# Within-host, mother-infant
counts = pooled_snp_change_distribution['Mother-Infant']
xs, ns = calculate_unnormalized_survival_from_vector(counts)
mlabel = "Mother-infant, <1 week" + ('\n(n=%d)' % ns[0])
ax_snp.step(xs,ns/float(ns[0]),'-',color=colors[color_i],linewidth=1.4, label=mlabel,
where='pre',zorder=4)
color_i += 1

# Within-host, infant-infant
for infant_custom_cohort, infant_custom_label in zip(['II-3mon', 'II-1yr'], ['Infant-infant,
sampled <3 months apart', 'Infant-infant, sampled >3 months apart']):
    counts = pooled_snp_change_distribution[infant_custom_cohort]
    xs, ns = calculate_unnormalized_survival_from_vector(counts)
    mlabel = ('%s\n(n=%d)' % (infant_custom_label, ns[0]))
    ax_snp.step(xs,ns/float(ns[0]),'-',color=colors[color_i],linewidth=1.4, label=mlabel,
where='pre',zorder=4)
    color_i += 1

# Save
ax_snp.legend(loc='best', bbox_to_anchor=(1.05, 1), fontsize=10, numpoints=1, ncol=1,
handlelength=1, frameon=False)
fig_snp.savefig('%s/temporal_snp_changes_%s_pooled_infant_combined_v2.pdf' %
(config.analysis_directory, sweep_type),bbox_inches='tight',dpi=600)
plt.show()


# In[57]:


# Compare between with within, combined across Adult-Adult, Mother-Infant, II-1mon and II-1yr
# cats = ['Adult-Adult', 'Mother-Infant', 'II-1mon', 'II-1yr']
cats = ['Mother-Infant']
between_counts = []
within_counts = []
for cat in cats:
```

```
    between_counts += pooled_between_snp_change_distribution[cat]
    within_counts += pooled_snp_change_distribution[cat]

for counts in [between_counts, within_counts]:
    num_zero = 0
    num_mod = 0
    num_replace = 0
    for count in counts:
        if count == 0:
            num_zero += 1
        elif count <= 20:
            num_mod += 1
        elif count > 500:
            num_replace += 1

    print("Proportion no change: %.04f" % (float(num_zero)/len(counts)))
    print("Proportion <= 20: %.04f" % (float(num_mod)/len(counts)))
    print("Proportion > 500: %.04f" % (float(num_replace)/len(counts)))
    print('')


# In[38]:


# THIS IS FIGURE 2
# Combined SNP change, replacement, gene gain and loss rate plot, AND SNP changes plot

# fig, ax = plt.subplots(4, 1, figsize=(8, 16), gridspec_kw={'height_ratios': [1, 1, 1, 1]})
fig = plt.figure(figsize=(8, 12))
gs = gridspec.GridSpec(2, 2, width_ratios=[3, 1], height_ratios=[3, 6], hspace=0.15)

ax = []
# ============================================================================================
# First SNP diff survival curve plot
# ============================================================================================

ax.append(fig.add_subplot(gs[0, 0]))

num_colors = 8
colormap = cmx.get_cmap('viridis', num_colors)
colors = [colormap(x) for x in np.array([x for x in range(0,num_colors)])/float(num_colors)]
colors = ['gray', 'blue', '#00f0b6', '#7bb551', '#0e750e']

modification_difference_threshold = config.modification_difference_threshold
replacement_difference_threshold = config.replacement_difference_threshold

ax[0].set_xscale('log'); ax[0].set_yscale('log')
ax[0].set_ylabel('Fraction comparisons $\geq n$', fontsize=11)
ax[0].set_xlabel('# SNV changes', fontsize=11)
ax[0].spines['top'].set_visible(False); ax[0].spines['right'].set_visible(False)
ax[0].get_xaxis().tick_bottom(); ax[0].get_yaxis().tick_left()

color_i = 0

ymin, ymax = 0.0001, 1.3; ax[0].set_ylim([ymin,ymax])
xmin, xmax = 1e-01, 2e05; ax[0].set_xlim([xmin, xmax])

# Now fill in the graphics
ax[0].fill_between([xmin,1], [ymin,ymin],[ymax,ymax],color='0.8',zorder=1)
ax[0].fill_between([1,modification_difference_threshold],[ymin,ymin],[ymax,ymax],color='#deebf7'
,zorder=1)
```

```
ax[0].fill_between([replacement_difference_threshold,xmax],[ymin,ymin],[ymax,ymax],color='#fee0d
2',zorder=1)

ax[0].text( exp((log(xmax)+log(replacement_difference_threshold))/2), ymax*1.2,
'strain\nreplacement',
           fontsize=12,fontstyle='italic',ha='center',color='#ca7258',zorder=1)
ax[0].text( exp((log(1)+log(modification_difference_threshold))/2), ymax*1.2,
'evolutionary\nmodification',
           fontsize=12,fontstyle='italic',ha='center',color='#789eb1',zorder=1)

# Unrelated hosts (adults OR mother/infant)
counts = []
for cat in ['Adult-Adult', 'Mother-Infant', 'II-3mon', 'II-1yr']:
    counts += pooled_between_snp_change_distribution[cat]
xs, ns = calculate_unnormalized_survival_from_vector(counts)
ax[0].step(xs,ns/float(ns[0]),'-',color=colors[color_i],linewidth=1.4, label="Unrelated hosts",
where='pre',zorder=4)
color_i += 1

# Within-host, adult
counts = pooled_snp_change_distribution['Adult-Adult']
xs, ns = calculate_unnormalized_survival_from_vector(counts)
mlabel = 'Adult-adult, sampled ~6 months apart' + ('\n(n=%d)' % ns[0])
ax[0].step(xs,ns/float(ns[0]),'-',color=colors[color_i],linewidth=1.4, label=mlabel,
where='pre',zorder=4)
color_i += 1

# Within-host, mother-infant
counts = pooled_snp_change_distribution['Mother-Infant']
xs, ns = calculate_unnormalized_survival_from_vector(counts)
mlabel = "Mother-infant, sampled <1 week apart" + ('\n(n=%d)' % ns[0])
ax[0].step(xs,ns/float(ns[0]),'-',color=colors[color_i],linewidth=1.4, label=mlabel,
where='pre',zorder=4)
color_i += 1

# Within-host, infant-infant
for infant_custom_cohort, infant_custom_label, color in zip(['II-3mon', 'II-1yr'],
                                                            ['Infant-infant, sampled <3 months
apart', 'Infant-infant, sampled >3 months apart'],
                                                            ['#e02a72', '#7b3294']):
    counts = pooled_snp_change_distribution[infant_custom_cohort]
    xs, ns = calculate_unnormalized_survival_from_vector(counts)
    mlabel = ('%s\n(n=%d)' % (infant_custom_label, ns[0]))
    ax[0].step(xs,ns/float(ns[0]),'-',color=color,linewidth=1.4, label=mlabel,
where='pre',zorder=4)
    color_i += 1

ax[0].legend(loc='best', bbox_to_anchor=(1.002, 0.7), fontsize=10, numpoints=1, ncol=1,
handlelength=1, frameon=False)
ax[0].text(-0.2, 0.95, 'A', size=20, transform=ax[0].transAxes, weight='bold')

# ============================================================================================

ylim_lower = 5e-5

ylabel_size=11
tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

gs1 = gridspec.GridSpecFromSubplotSpec(3, 1, subplot_spec=gs[1, :], hspace=0)
ax_snps = fig.add_subplot(gs1[0])
ax.append(ax_snps)
```

```
boxplots_0 = ax[1].boxplot(snp_change_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))

for i in np.arange(len(snp_change_rates)):
    boxplots_0['boxes'][i].set_facecolor('#77acff')

ax[1].plot(np.arange(1, 1+len(snp_change_rate_annotations)), snp_change_rate_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=10)
# ax[1].set_xticklabels(snp_change_rate_tp_type_labels)
ax[1].get_xaxis().set_visible(False)
ax[1].set_ylabel("SNV changes\nper day", fontsize=ylabel_size)
# ax[0].set_title("SNP changes per non-replacement QP pair per day")
ax[1].set_yscale('log')
ax[1].set_ylim((ylim_lower,0.8))
ax[1].text(-0.15, 0.95, 'B', size=20, transform=ax[1].transAxes, weight='bold')

ax.append(fig.add_subplot(gs1[1]))
boxplots_1 = ax[2].boxplot(replacement_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))

for i in np.arange(len(replacement_rates)):
    boxplots_1['boxes'][i].set_facecolor('#77acff')

ax[2].plot(np.arange(1, 1+len(replacement_rate_annotations)), replacement_rate_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=10)
# ax[2].set_xticklabels(replacement_rate_tp_type_labels)
ax[2].get_xaxis().set_visible(False)
ax[2].set_ylabel("Replacements\nper day", fontsize=ylabel_size)
# ax[1].set_title("Replacements per QP pair per day")
ax[2].set_yscale('log')
ax[2].set_ylim((ylim_lower,0.8))
ax[2].text(-0.15, 0.95, 'C', size=20, transform=ax[2].transAxes, weight='bold')

bootstrapped_all_gene_change_rates = []
for gain_rates, loss_rates in zip(bootstrapped_gain_rates, bootstrapped_loss_rates):
    bootstrapped_all_gene_change_rates.append(gain_rates)
    bootstrapped_all_gene_change_rates.append(loss_rates)

all_gene_change_annotations = []
for gain_annotation, loss_annotation in zip(gain_annotations, loss_annotations):
    all_gene_change_annotations.append(gain_annotation)
    all_gene_change_annotations.append(loss_annotation)

ax.append(fig.add_subplot(gs1[2]))
boxplots_2 = ax[3].boxplot(bootstrapped_all_gene_change_rates, patch_artist=True, zorder=-1,
                           medianprops=dict(color='black'),
                           flierprops=dict(marker='.'))
for i in np.arange(len(bootstrapped_all_gene_change_rates), step=2):
    boxplots_2['boxes'][i].set_facecolor(plot_utils.col_gain) # gain
    boxplots_2['boxes'][i+1].set_facecolor(plot_utils.col_loss) # loss

ax[3].plot(np.arange(1, 1+len(all_gene_change_annotations)), all_gene_change_annotations,
marker=(5, 2), color='black', linestyle='None', markersize=10)
ax[3].set_xticks(np.arange(2*len(gain_tp_type_labels), step=2) + 1.5)
ax[3].set_xticklabels(tp_types) # ax[3].set_xticklabels(gain_tp_type_labels)
ax[3].set_ylabel("Gene changes\nper day", fontsize=ylabel_size)
# ax[2].set_title("Gene changes per non-replacement QP pair per day")
ax[3].set_yscale('log')
ax[3].set_ylim((ylim_lower,0.8))
```

```
ax[3].text(-0.15, 0.95, 'D', size=20, transform=ax[3].transAxes, weight='bold')

legend_elements = [Patch(facecolor=plot_utils.col_gain, label='Gain'),
Patch(facecolor=plot_utils.col_loss, label='Loss')]

ax[3].legend(handles=legend_elements, loc='upper right', frameon=False)

# plt.subplots_adjust(wspace=0, hspace=0.3)
plt.show()

fig.savefig('%s/figure_3_v8.png' % plot_dir, bbox_inches='tight', dpi=600)
fig.savefig('%s/figure_3_v8.pdf' % plot_dir, bbox_inches='tight')


# In[145]:


# Linear regression
# log(SNV rates) vs. log(infant timepoint)

adult_interval_days = 183
adult_initial_days = 40*365

# Store SNP change information: including starting day

replacement_time_tups_by_tp_type = defaultdict(list)
all_time_tups_by_tp_type = defaultdict(list)

for species in snp_changes:
    for s1, s2 in snp_changes[species]:

        val = snp_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        # SPECIAL: try recoding mother-infant duration as sum
        if 'Mother-Infant' in custom_cohorts:
            day1 = mi_sample_day_dict[s1]
            day2 = mi_sample_day_dict[s2]
            days = day1 + day2
            if days == 0:
                days = 1

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue
        '''
        # Arbitrarily set mother 0 - infant 0 as 1 day
        if is_mother(s1) and mi_sample_day_dict[s1] == 0 and is_infant(s2) and
mi_sample_day_dict[s2] == 0:
            days = 1
        '''
        if 'Adult-Adult' in custom_cohorts:
            day1 = adult_initial_days + ((sample_order_map[s1][1]-1)*adult_interval_days)
        else:
            day1 = mi_sample_day_dict[s1]

        if isinstance(val, int): # Replacement
            for custom_cohort in custom_cohorts:
                replacement_time_tups_by_tp_type[custom_cohort].append((1, days, day1))
```

```
                all_time_tups_by_tp_type[custom_cohort].append((val, days, day1))
        else: # Not replacement (modifications/no change)
            for custom_cohort in custom_cohorts:
                count_time_tups_by_tp_type[custom_cohort].append((len(val), days, day1))
                all_time_tups_by_tp_type[custom_cohort].append((len(val), days, day1))

# Store gene change information: include starting day

gain_tups_by_tp_type = defaultdict(list)
loss_tups_by_tp_type = defaultdict(list)

for species in gene_changes:
    for s1, s2 in gene_changes[species]:

        gains, losses = gene_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        # SPECIAL: try recoding mother-infant duration as sum
        if 'Mother-Infant' in custom_cohorts:
            day1 = mi_sample_day_dict[s1]
            day2 = mi_sample_day_dict[s2]
            days = day1 + day2
            if days == 0:
                days = 1

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue
        '''
        # Arbitrarily set mother 0 - infant 0 as 1 day
        if is_mother(s1) and mi_sample_day_dict[s1] == 0 and is_infant(s2) and
mi_sample_day_dict[s2] == 0:
            days = 1
        '''
        if isinstance(snp_changes[species][(s1, s2)], int): # Replacement (in terms of SNP
changes)
            continue# num_gains, num_losses = gene_changes[species][(s1, s2)]
        else:
            num_gains = len(gene_changes[species][(s1, s2)][0])
            num_losses = len(gene_changes[species][(s1, s2)][1])

        # Try excluding the Mother-Infant(wk1) QP pair with unsually high number of losses
        if (num_losses >= 13) and ('Mother-Infant' in custom_cohorts):
            print("Sample pair %s, %s | %s" % (s1, s2, species))
            print("Num losses: %i | Num gains: %i" % (num_losses, num_gains))
            continue

        if 'Adult-Adult' in custom_cohorts:
            day1 = adult_initial_days + ((sample_order_map[s1][1]-1)*adult_interval_days)
        else:
            day1 = mi_sample_day_dict[s1]

        for custom_cohort in custom_cohorts:
            gain_tups_by_tp_type[custom_cohort].append((num_gains, days, day1))
            loss_tups_by_tp_type[custom_cohort].append((num_losses, days, day1))


# In[146]:
```

```
tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

gain_rates = []
gain_day1s = []
gain_means = []

loss_rates = []
loss_day1s = []
loss_means = []

rates = [] # Modification SNP change rate
day1s = []
means = []

gain_middle_days_lists = []
loss_middle_days_lists = []
middle_days_lists = []

for tp_type in tp_types:

    # SNP change rates ========================================================
    total_count = 0; total_days = 0
    middle_days = []

    for count, days, day1 in all_time_tups_by_tp_type[tp_type]:
        if count <= 20: # Restrict to modification/no change
            rate = count/float(days)

            if day1 == 0:
                day1 = 0.1 # Arbitrary number << 1 for visualization purposes
            if count == 0:
                rate = 0 # Arbitrary number << 0.002 for visualization purposes

            rates.append(rate)
            day1s.append(day1)

            total_count += count
            total_days += days
            middle_days.append((day1) + (days/2.0))

    middle_days_lists.append(middle_days)
    means.append(total_count/float(total_days))
    # ============================================================================

    # Gene gain rates =========================================================
    total_count = 0; total_days = 0
    middle_days = []

    for count, days, day1 in gain_tups_by_tp_type[tp_type]:
        # Already excludes replacements
        rate = count/float(days)

        if day1 == 0:
            day1 = 0.1 # Arbitrary number << 1 for visualization purposes

        gain_rates.append(rate)
        gain_day1s.append(day1)

        total_count += count
        total_days += days
```

```
        middle_days.append((day1) + (days/2.0))

    gain_middle_days_lists.append(middle_days)
    gain_means.append(total_count/float(total_days))
    # ========================================================================

    # Gene loss rates ========================================================
    total_count = 0; total_days = 0
    middle_days = []

    for count, days, day1 in loss_tups_by_tp_type[tp_type]:
        # Already excludes replacements
        rate = count/float(days)

        if day1 == 0:
            day1 = 0.1 # Arbitrary number << 1 for visualization purposes

        loss_rates.append(rate)
        loss_day1s.append(day1)

        total_count += count
        total_days += days
        middle_days.append((day1) + (days/2.0))

    loss_middle_days_lists.append(middle_days)
    loss_means.append(total_count/float(total_days))
    # ========================================================================


# In[147]:


fig, ax = plt.subplots(4, 1, figsize=(8, 8), sharex=True) # , sharey=True)

xs = np.array([0.1, 3, 14, 60, 14600])
ax[0].plot([0.1, 3, 14, 60, 14600], means, marker=(5, 2), color='black', linestyle='None',
markersize=10)
ax[0].plot(day1s, rates, '.')

m, b = np.polyfit(np.log(xs), np.log(means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(means))
ax[0].plot(xs, np.exp(m*np.log(xs) + b), color='0.7', zorder=-1)
ax[0].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[0].transAxes)

ax[0].set_yscale('log')
ax[0].set_xscale('log')

ax[0].set_ylabel("SNP changes\nper day")

# ========================================================================

ax[1].plot([0.1, 3, 14, 60, 14600], gain_means, marker=(5, 2), color='black', linestyle='None',
markersize=10)
ax[1].plot(gain_day1s, gain_rates, '.')

m, b = np.polyfit(np.log(xs), np.log(gain_means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(gain_means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(gain_means))
ax[1].plot(xs, np.exp(m*np.log(xs) + b), color='red', alpha=0.44, zorder=-1)
```

```
ax[1].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[1].transAxes,
          color='red')

ax[1].set_yscale('log')
ax[1].set_xscale('log')

ax[1].set_ylabel("Gene gains\nper day")

# ============================================================================

ax[2].plot([0.1, 3, 14, 60, 14600], loss_means, marker=(5, 2), color='black', linestyle='None',
markersize=10)
ax[2].plot(loss_day1s, loss_rates, '.')

m, b = np.polyfit(np.log(xs), np.log(loss_means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(loss_means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(loss_means))
ax[2].plot(xs, np.exp(m*np.log(xs) + b), color='0.7', zorder=-1)
ax[2].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[2].transAxes)

ax[2].set_yscale('log')
ax[2].set_xscale('log')

ax[2].set_ylabel("Gene losses\nper day")

# ============================================================================
replace_means = replacement_rate_annotations

ax[3].plot([0.1, 3, 14, 60, 14600], replace_means, marker=(5, 2), color='black',
linestyle='None', markersize=10)

m, b = np.polyfit(np.log(xs), np.log(replace_means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(replace_means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(replace_means))
ax[3].plot(xs, np.exp(m*np.log(xs) + b), color='0.7', zorder=-1)
ax[3].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[3].transAxes)

ax[3].set_yscale('log')

ax[3].set_xlabel("Days after birth")
ax[3].set_ylabel("Replacements\nper day")

ax[3].set_xticks([1e-1, 1, 10, 100, 1000])
ax[3].get_xaxis().set_major_formatter(matplotlib.ticker.ScalarFormatter())

plt.subplots_adjust(hspace=0)
plt.show()


# In[151]:


# Second version: put gains and losses on same subplot

gain_color = 'green'
loss_color = '#ff7f00'

fig, ax = plt.subplots(3, 1, figsize=(8, 7), sharex=True)
```

```
xs = np.array([0.1, 3, 14, 60, 14600])
xs = [np.mean(days) for days in middle_days_lists]
ax[0].plot(xs, means, marker=(5, 2), color='black', linestyle='None', markersize=10)
# ax[0].scatter(xs, means, marker='D', color='black', linestyle='None', s=16)
# ax[0].scatter(day1s, rates, alpha=0.6)

m, b = np.polyfit(np.log(xs), np.log(means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(means))
ax[0].plot(xs, np.exp(m*np.log(xs) + b), color='0.7', zorder=-1)
ax[0].text(0.03, 0.1, r'$r^2$=%.02f  $\it{p}$=%.02f' % (r2, p_value), transform=ax[0].transAxes)
# ax[0].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[0].transAxes)
ax[0].text(-0.15, 0.92, 'A', size=20, transform=ax[0].transAxes, weight='bold')

ax[0].set_yscale('log')
ax[0].set_xscale('log')

ax[0].set_ylabel("SNV changes\nper day")

# ============================================================================

xs = [np.mean(days) for days in gain_middle_days_lists]

legend_elements = [Patch(facecolor='#b3de69', label='Gain'), Patch(facecolor='#ff7f00',
label='Loss')]

ax[1].plot(xs, gain_means, marker=(5, 2), color=gain_color, linestyle='None', markersize=10,
zorder=5, label='Gain')
# gainr_all = ax[1].scatter(xs, gain_means, marker='^', color='green', linestyle='None', s=16,
zorder=5, label='Gain')
# gainr_means = ax[1].scatter(gain_day1s, gain_rates, color='#b3de69', alpha=0.6, label='Gain')

m, b = np.polyfit(np.log(xs), np.log(gain_means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(gain_means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(gain_means))
ax[1].plot(xs, np.exp(m*np.log(xs) + b), color=gain_color, alpha=0.44, zorder=-2)
ax[1].text(0.03, 0.2, r'$r^2$=%.02f  $\it{p}$=%.02f' % (r2, p_value), transform=ax[1].transAxes,
color=gain_color)
# ax[1].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[1].transAxes, color='red')

xs = [np.mean(days) for days in loss_middle_days_lists]

ax[1].plot(xs, loss_means, marker=(5, 2), color=loss_color, linestyle='None', markersize=10,
zorder=6, label='Loss')
# lossr_all = ax[1].scatter(xs, loss_means, marker='v', color='orange', linestyle='None', s=16,
zorder=6, label='Loss')
# lossr_means = ax[1].scatter(loss_day1s, loss_rates, color='#ff7f00', alpha=0.6, zorder=5,
label='Loss')

m, b = np.polyfit(np.log(xs), np.log(loss_means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(loss_means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(loss_means))
ax[1].plot(xs, np.exp(m*np.log(xs) + b), color=loss_color, alpha=0.44, zorder=-1)
ax[1].text(0.03, 0.1, r'$r^2$=%.02f  $\it{p}$=%.02f' % (r2, p_value), transform=ax[1].transAxes,
color=loss_color)
# ax[1].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[1].transAxes)
ax[1].text(-0.15, 0.92, 'B', size=20, transform=ax[1].transAxes, weight='bold')
```

```
ax[1].set_yscale('log')
ax[1].set_xscale('log')

ax[1].legend(frameon=False, ncol=2)
# ax[1].legend([(gainr_means, gainr_all), (lossr_means, lossr_all)], ["Gain", "Loss"],
frameon=False, ncol=2)

ax[1].set_ylabel("Gene changes\nper day")

# ============================================================================
replace_means = replacement_rate_annotations

ax[2].plot(xs, replace_means, marker=(5, 2), color='black', linestyle='None', markersize=10)
# ax[2].scatter(xs, replace_means, marker='D', color='black', linestyle='None', s=16)

m, b = np.polyfit(np.log(xs), np.log(replace_means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(replace_means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(replace_means))
ax[2].plot(xs, np.exp(m*np.log(xs) + b), color='0.7', zorder=-1)
ax[2].text(0.03, 0.1, r'$r^2$=%.02f  $\it{p}$=%.02f' % (r2, p_value), transform=ax[2].transAxes)
# ax[2].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[2].transAxes)
ax[2].text(-0.15, 0.92, 'C', size=20, transform=ax[2].transAxes, weight='bold')

ax[2].set_yscale('log')

ax[2].set_xlabel("Days after birth")
ax[2].set_ylabel("Replacements\nper day")

ax[2].set_xlim((1, 20000))
ax[2].set_xticks([1, 10, 100, 1000])
ax[2].get_xaxis().set_major_formatter(matplotlib.ticker.ScalarFormatter())

plt.subplots_adjust(hspace=0.13)
plt.show()
fig.savefig('%s/S7_rates_vs_median_time.pdf' % plot_dir, bbox_inches='tight')


# In[ ]:


# Second version: put gains and losses on same subplot

fig, ax = plt.subplots(3, 1, figsize=(8, 6), sharex=True)

xs = np.array([0.1, 5.4, 14.5, 171, 14750])

ax[0].plot(xs, means, marker=(5, 2), color='black', linestyle='None', markersize=10)
# ax[0].scatter(xs, means, marker='D', color='black', linestyle='None', s=16)
# ax[0].scatter(day1s, rates, alpha=0.6)

m, b = np.polyfit(np.log(xs), np.log(means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(means))
ax[0].plot(xs, np.exp(m*np.log(xs) + b), color='0.7', zorder=-1)
ax[0].text(0.03, 0.1, r'$r^2$=%.02f  $\it{p}$=%.02f' % (r2, p_value), transform=ax[0].transAxes)
# ax[0].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[0].transAxes)

ax[0].set_yscale('log')
```

```
ax[0].set_xscale('log')

ax[0].set_ylabel("SNP changes\nper day")

# ================================================================================

legend_elements = [Patch(facecolor='#b3de69', label='Gain'), Patch(facecolor='#ff7f00',
label='Loss')]

ax[1].plot(xs, gain_means, marker=(5, 2), color='green', linestyle='None', markersize=10,
zorder=5, label='Gain')
# gainr_all = ax[1].scatter(xs, gain_means, marker='^', color='green', linestyle='None', s=16,
zorder=5, label='Gain')
# gainr_means = ax[1].scatter(gain_day1s, gain_rates, color='#b3de69', alpha=0.6, label='Gain')

m, b = np.polyfit(np.log(xs), np.log(gain_means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(gain_means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(gain_means))
ax[1].plot(xs, np.exp(m*np.log(xs) + b), color='red', alpha=0.44, zorder=-2)
ax[1].text(0.03, 0.2, r'$r^2$=%.02f  $\it{p}$=%.02f' % (r2, p_value), transform=ax[1].transAxes,
color='red')
# ax[1].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[1].transAxes, color='red')

ax[1].plot(xs, loss_means, marker=(5, 2), color='orange', linestyle='None', markersize=10,
zorder=6, label='Loss')
# lossr_all = ax[1].scatter(xs, loss_means, marker='v', color='orange', linestyle='None', s=16,
zorder=6, label='Loss')
# lossr_means = ax[1].scatter(loss_day1s, loss_rates, color='#ff7f00', alpha=0.6, zorder=5,
label='Loss')

m, b = np.polyfit(np.log(xs), np.log(loss_means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(loss_means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(loss_means))
ax[1].plot(xs, np.exp(m*np.log(xs) + b), color='0.7', zorder=-1)
ax[1].text(0.03, 0.1, r'$r^2$=%.02f  $\it{p}$=%.02f' % (r2, p_value), transform=ax[1].transAxes)
# ax[1].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[1].transAxes)

ax[1].set_yscale('log')
ax[1].set_xscale('log')

ax[1].legend(frameon=False, ncol=2)
# ax[1].legend([(gainr_means, gainr_all), (lossr_means, lossr_all)], ["Gain", "Loss"],
frameon=False, ncol=2)

ax[1].set_ylabel("Gene changes\nper day")

# ================================================================================
replace_means = replacement_rate_annotations

ax[2].plot(xs, replace_means, marker=(5, 2), color='black', linestyle='None', markersize=10)
# ax[2].scatter(xs, replace_means, marker='D', color='black', linestyle='None', s=16)

m, b = np.polyfit(np.log(xs), np.log(replace_means), deg=1)
r2 = np.corrcoef(np.log(xs), np.log(replace_means))[0, 1]**2
slope, intercept, r_value, p_value, std_err = stats.linregress(np.log(xs),np.log(replace_means))
ax[2].plot(xs, np.exp(m*np.log(xs) + b), color='0.7', zorder=-1)
ax[2].text(0.03, 0.1, r'$r^2$=%.02f  $\it{p}$=%.02f' % (r2, p_value), transform=ax[2].transAxes)
# ax[2].text(0.8, 0.55, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f\np: %.02f" % (m, b, r2,
p_value), transform=ax[2].transAxes)
```

```
ax[2].set_yscale('log')

ax[2].set_xlabel("Days after birth")
ax[2].set_ylabel("Replacements\nper day")

ax[2].set_xlim((0.05, 20000))
ax[2].set_xticks([0.1, 1, 10, 100, 1000])
ax[2].get_xaxis().set_major_formatter(matplotlib.ticker.ScalarFormatter())

plt.subplots_adjust(hspace=0)
plt.show()


# In[ ]:


# Linear regression

# Combined SNP change, replacement, gene gain and loss rate plot: final version??

fig, ax = plt.subplots(4, 1, figsize=(8, 10), sharex=True)

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

xs = np.arange(1, 1+len(tp_types))

ax[0].plot(xs, snp_change_rate_annotations, '.b', linestyle='None', markersize=10)
ax[0].set_yscale('log')
m, b = np.polyfit(xs, np.log(snp_change_rate_annotations), deg=1)
r2 = np.corrcoef(xs, np.log(snp_change_rate_annotations))[0, 1]**2
ax[0].plot(xs, np.exp(m*xs + b))
ax[0].text(0.8, 0.6, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f" % (m, b, r2),
transform=ax[0].transAxes)
ax[0].set_title("SNP changes per day (per non-replacement QP pair)")

ax[1].plot(xs, replacement_rate_annotations, '.b', linestyle='None', markersize=10)
ax[1].set_yscale('log')
m, b = np.polyfit(xs, np.log(replacement_rate_annotations), deg=1)
r2 = np.corrcoef(xs, np.log(replacement_rate_annotations))[0, 1]**2
ax[1].plot(xs, np.exp(m*xs + b))
ax[1].text(0.8, 0.6, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f" % (m, b, r2),
transform=ax[1].transAxes)
ax[1].set_title("Replacements per day (per QP pair)")

ax[2].plot(xs, gain_annotations, '.b', linestyle='None', markersize=10)
ax[2].set_yscale('log')
m, b = np.polyfit(xs, np.log(gain_annotations), deg=1)
r2 = np.corrcoef(xs, np.log(gain_annotations))[0, 1]**2
ax[2].plot(xs, np.exp(m*xs + b))
ax[2].text(0.8, 0.6, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f" % (m, b, r2),
transform=ax[2].transAxes)
ax[2].set_title("Gene gains per day (per non-replacement QP pair)")

ax[3].plot(xs, loss_annotations, '.b', linestyle='None', markersize=10)
ax[3].set_yscale('log')
m, b = np.polyfit(xs, np.log(loss_annotations), deg=1)
r2 = np.corrcoef(xs, np.log(loss_annotations))[0, 1]**2
ax[3].plot(xs, np.exp(m*xs + b))
ax[3].text(0.8, 0.6, "Slope: %.02f\nIntercept: %.02f\nR2: %.02f" % (m, b, r2),
transform=ax[3].transAxes)
```

```
ax[3].set_title("Gene losses per day (per non-replacement QP pair)")
ax[3].set_xticks(xs)
ax[3].set_xticklabels(tp_types)

plt.show()


# In[3]:


# Statistics for plot
for tp_type, rate in zip(tp_types, replacement_rate_annotations):
    print("Rate for %s: %.2E" % (tp_type, rate))


# In[ ]:


# Store gene change information

tp_types = ['Mother-Infant', 'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Adult-Adult']

num_gain_by_species_tp_type = {tt: defaultdict(int) for tt in tp_types}
num_loss_by_species_tp_type = {tt: defaultdict(int) for tt in tp_types}

for species in gene_changes:
    for s1, s2 in gene_changes[species]:

        gains, losses = gene_changes[species][(s1, s2)]
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, s1, s2, species)
        days = sample_pair_to_days(s1, s2)

        # Skip negative days for now (seems to be either Yassour 3rd mom tp, Ferretti reverse
IM)
        if days < 0:
            continue

        if isinstance(snp_changes[species][(s1, s2)], int): # Replacement (in terms of SNP
changes)
            pass
            # num_gains, num_losses = gene_changes[species][(s1, s2)]
        else:
            num_gains = len(gene_changes[species][(s1, s2)][0])
            num_losses = len(gene_changes[species][(s1, s2)][1])

        for custom_cohort in custom_cohorts:
            num_gain_by_species_tp_type[custom_cohort][species] += num_gains
            num_loss_by_species_tp_type[custom_cohort][species] += num_losses


# In[ ]:


'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1'


# In[ ]:


custom_cohort = 'Mother-Infant'
fig, ax = plt.subplots(figsize=(6, 12))
```

```
species_count_dict = num_gain_by_species_tp_type[custom_cohort]
species_count_tups_sorted = sorted(species_count_dict.items(), key=lambda x: x[1])
counts = [x[1] for x in species_count_tups_sorted]
species = [x[0] for x in species_count_tups_sorted]
ys = np.arange(len(species))
ax.barh(species, counts)
ax.set_title(custom_cohort + ": gene gains")
plt.show()


# In[ ]:


custom_cohort = 'Mother-Infant'
fig, ax = plt.subplots(figsize=(6, 12))
species_count_dict = num_loss_by_species_tp_type[custom_cohort]
species_count_tups_sorted = sorted(species_count_dict.items(), key=lambda x: x[1])
counts = [x[1] for x in species_count_tups_sorted]
species = [x[0] for x in species_count_tups_sorted]
ys = np.arange(len(species))
ax.barh(species, counts)
ax.set_title(custom_cohort + ": gene losses")
plt.show()


# In[ ]:


sample_species_qp_dict = pickle.load(open("%s/pickles/sample_species_qp_dict.pkl" %
config.data_directory, 'rb'))


# In[ ]:


total_infant_lowcov_samples = 0
total_infant_highcov_samples = 0
total_infant_qp_samples = 0

for sample in sample_species_qp_dict['infant']:

    # Skip  Olm
    if sample not in infant_samples:
        continue

    for species in sample_species_qp_dict['infant'][sample]:
        status = sample_species_qp_dict['infant'][sample][species]
        if status == 'low-coverage':
            total_infant_lowcov_samples += 1
        else:
            total_infant_highcov_samples += 1
        if status == 'qp':
            total_infant_qp_samples += 1


# In[ ]:


print(total_infant_qp_samples)
```

```
# In[ ]:


total_mother_lowcov_samples = 0
total_mother_highcov_samples = 0
total_mother_qp_samples = 0

for sample in sample_species_qp_dict['mother']:

    # Skip  Olm
    if sample not in mother_samples:
        continue

    for species in sample_species_qp_dict['mother'][sample]:
        status = sample_species_qp_dict['mother'][sample][species]
        if status == 'low-coverage':
            total_mother_lowcov_samples += 1
        else:
            total_mother_highcov_samples += 1
        if status == 'qp':
            total_mother_qp_samples += 1


# In[ ]:


print(total_mother_highcov_samples)


# In[ ]:


sample_species_qp_dict['hmp']
```

plot_figure_3.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import config, parse_midas_data, sample_utils as su, temporal_changes_utils,
stats_utils, midas_db_utils, parse_patric
from collections import defaultdict
import numpy as np
from numpy.random import binomial as sample_binomial
import math
import pickle
import sys
import random
from math import log10,ceil,log,exp

import matplotlib.cm as cmx
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import matplotlib.colors as mcolors
import matplotlib.patheffects as pe
from matplotlib.patches import Patch
```

```
# Cohort list
cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'olm', 'hmp']

# Plot directory
plot_dir = "%s/" % (config.analysis_directory)

# Species list
good_species_list = parse_midas_data.load_pickled_good_species_list()

# Sample-subject-order maps
sys.stderr.write("Loading sample metadata...\n")
subject_sample_map = su.parse_subject_sample_map()
sample_order_map = su.parse_sample_order_map()
sample_subject_map = su.parse_sample_subject_map()
same_mi_pair_dict = su.get_same_mi_pair_dict(subject_sample_map)
sys.stderr.write("Done!\n")

# Timepoint pair types
tp_pair_names = ['MM', 'MI', 'II', 'AA']

# Cohorts
cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
mi_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

# Samples for each cohort
samples = {cohort: su.get_sample_names(cohort) for cohort in cohorts}
hmp_samples = su.get_sample_names('hmp')
mother_samples = su.get_sample_names('mother')
infant_samples = su.get_sample_names('infant')
olm_samples = su.get_sample_names('olm')
infant_samples_no_olm = [sample for sample in infant_samples if sample not in
olm_samples]
mi_samples_no_olm = [sample for sample in (mother_samples + infant_samples) if
sample not in olm_samples]

# Sample-cohort map
sample_cohort_map = su.parse_sample_cohort_map()

# Sample-timepoint map
mi_sample_day_dict = su.get_mi_sample_day_dict(exclude_cohorts=['olm'])
mi_tp_sample_dict = su.get_mi_tp_sample_dict(exclude_cohorts=['olm']) # no binning
mi_tp_sample_dict_binned, mi_tp_binned_labels =
su.get_mi_tp_sample_dict(exclude_cohorts=['olm'], binned=True)


# In[2]:


# ================================================================
# Load pickled data
# ================================================================

# Parameters
sweep_type = 'full' # assume full for now
pp_prev_cohort = 'all'
min_coverage = 0

ddir = config.data_directory
pdir = "%s/pickles/cov%i_prev_%s/" % (ddir, min_coverage, pp_prev_cohort)
```

```
snp_changes = pickle.load(open('%s/big_snp_changes_%s.pkl' % (pdir, sweep_type),
'rb'), encoding='latin1')
gene_changes = pickle.load(open('%s/big_gene_changes_%s.pkl' % (pdir, sweep_type),
'rb'), encoding='latin1')
snp_change_freqs = pickle.load(open('%s/snp_change_freqs_%s.pkl' % (pdir,
sweep_type), 'rb'), encoding='latin1')
snp_change_null_freqs = pickle.load(open('%s/snp_change_null_freqs_%s.pkl' % (pdir,
sweep_type), 'rb'), encoding='latin1')
gene_gain_freqs = pickle.load(open('%s/gene_gain_freqs_%s.pkl' % (pdir,
sweep_type), 'rb'), encoding='latin1')
gene_loss_freqs = pickle.load(open('%s/gene_loss_freqs_%s.pkl' % (pdir,
sweep_type), 'rb'), encoding='latin1')
gene_loss_null_freqs = pickle.load(open('%s/gene_loss_null_freqs_%s.pkl' % (pdir,
sweep_type), 'rb'), encoding='latin1')
between_snp_change_counts =
pickle.load(open('%s/between_snp_change_counts_%s.pkl' % (pdir, sweep_type), 'rb'),
encoding='latin1')
between_gene_change_counts =
pickle.load(open('%s/between_gene_change_counts_%s.pkl' % (pdir, sweep_type),
'rb'), encoding='latin1')


# In[3]:


# nonconsecutive
pdir = "%s/pickles/cov%i_prev_%s/nonconsecutive/" % (ddir, min_coverage,
pp_prev_cohort)

dnds_info = pickle.load(open('%s/dnds_info.pkl' % (pdir), 'rb'), encoding='latin1')
snp_change_null_freqs = pickle.load(open('%s/snp_change_null_freqs_%s.pkl' % (pdir,
sweep_type), 'rb'), encoding='latin1')
snp_change_freqs_with_opps =
pickle.load(open('%s/snp_change_freqs_with_opps_full.pkl' % (pdir), 'rb'),
encoding='latin1')


# In[4]:


# ======================================================================
# Load pickled data
# ======================================================================

# Parameters
sweep_type = 'full' # assume full for now
pp_prev_cohort = 'hmp'
min_coverage = 0

ddir = config.data_directory
pdir = "%s/pickles/cov%i_prev_%s/" % (ddir, min_coverage, pp_prev_cohort)

gene_gain_freqs_prev_hmp = pickle.load(open('%s/gene_gain_freqs_%s.pkl' % (pdir,
sweep_type), 'rb'), encoding='latin1')
gene_loss_freqs_prev_hmp = pickle.load(open('%s/gene_loss_freqs_%s.pkl' % (pdir,
sweep_type), 'rb'), encoding='latin1')
gene_loss_null_freqs_prev_hmp = pickle.load(open('%s/gene_loss_null_freqs_%s.pkl' %
(pdir, sweep_type), 'rb'), encoding='latin1')


# In[5]:
```

```
# Calculate number of days for a timpeoint pair

mi_sample_day_dict = su.get_mi_sample_day_dict()

def sample_pair_to_days(sample1, sample2):
    days = mi_sample_day_dict[sample2] - mi_sample_day_dict[sample1]
    if days < 0:
        return np.abs(days)
    return days

def tp_pair_to_days(tp_pair):
    tpa, tpb = tp_pair
    o1 = float(tpa[1:])
    o2 = float(tpb[1:])
    return np.abs(o1-o2)

# Rough approximation of HMP time intervals
def adult_tp_pair_to_days(tp_pair):
    tpa, tpb = tp_pair
    return np.abs(int(tpa[-1:]) - int(tpb[-1:]))*183

def tp_pair_to_tp_type(tp_pair):
    tpa, tpb = tp_pair
    tp_type = tpa[0] + tpb[0]
    if tp_type == 'IM':
        tp_type = 'MI'
    return tp_type


# In[6]:


# Settings for prevalence plots

modification_difference_threshold = config.modification_difference_threshold
replacement_difference_threshold = config.replacement_difference_threshold
default_num_bootstraps = 10000
min_sample_size = 3
min_haploid_sample_size = 10

variant_types = ['1D','4D']
within_host_type = 'consecutive' # consecutive timepoints (vs. longest)

num_bootstraps = 10 # for gene change prevalence null

# For partitioning SNVs according to prevalence
derived_freq_bins = np.array([-1,0,0.01,0.1,0.5,0.9,0.99,1,2])
derived_virtual_freqs = np.arange(0,len(derived_freq_bins)-1)
derived_virtual_xticks = list(derived_virtual_freqs[:-1]+0.5)
derived_virtual_xticklabels = ['0','.01','.1','.5','.9','.99','1']

# For partitioning genes into different prevalence classes
gene_freq_bins = np.array([-1,0.1,0.5,0.9,2])
gene_freq_xticks = [-4, -3, -2, -1, 0, 1, 2, 3, 4]
gene_freq_xticklabels = ['0','0.1','0.5', '0.9','1','0.9','0.5', '0.1','0']
gene_gain_virtual_freqs = np.array([3.5,2.5,1.5,0.5])
gene_loss_virtual_freqs = np.array([-3.5,-2.5,-1.5,-0.5])

# Function
```

```
def get_f_idx(f):
    return ((f>derived_freq_bins[:-1])*
            (f<=derived_freq_bins[1:])).argmax()


# In[7]:


get_f_idx(0.0001)


# In[8]:


# species -> mother_sample -> infant_sample -> # days of infant_sample
# mother timepoint at delivery (-1 to 7 days)
mi_dict = {species: defaultdict(dict) for species in gene_gain_freqs}

for species in snp_changes:
    for sample_i, sample_j in snp_changes[species]:

        tp_pair = su.sample_pair_to_tp_pair(sample_i, sample_j, sample_order_map,
hmp_samples, mother_samples)
        tp_type = tp_pair_to_tp_type(tp_pair)

        # MI: restrict to earliest mother-infant pair per host
        if tp_type == 'MI':
            mother_days = mi_sample_day_dict[sample_i]
            if mother_days >= -1 and mother_days <= 7:
                mi_dict[species][sample_i][sample_j] = mi_sample_day_dict[sample_j]

# species -> mother_sample -> earliest infant sample
mi_earliest_infant_sample_dict = defaultdict(dict)

for species in mi_dict:
    for mother_sample in mi_dict[species]:
        ordered_infant_days = sorted(mi_dict[species][mother_sample].items(),
key=lambda x: x[1])
        infant_sample, days = ordered_infant_days[0]
        mi_earliest_infant_sample_dict[species][mother_sample] = infant_sample


# In[9]:


# Custom sample pair cohorts [not just sample!]

def custom_cohort_of_sample_pair(custom_cohort_tests, sample_i, sample_j):
    for cohort in custom_cohort_tests:
        if custom_cohort_tests[cohort](sample_i, sample_j):
            return cohort

custom_cohort_tests = {}
# custom_cohort_tests['MI'] = lambda sample_i, sample_j: (sample_i in
mother_samples and sample_j in infant_samples_no_olm)
custom_cohort_tests['MI'] = lambda sample_i, sample_j: ((sample_i in mother_samples
and sample_j in infant_samples_no_olm) and mi_sample_day_dict[sample_i] >= 0 and
mi_sample_day_dict[sample_j] <= 7)
custom_cohort_tests['II'] = lambda sample_i, sample_j: ((sample_i in
infant_samples_no_olm and sample_j in infant_samples_no_olm))
```

```
# custom_cohort_tests['Day 0-Week 1'] = lambda sample_i, sample_j: ((sample_i in
infant_samples_no_olm and sample_j in infant_samples_no_olm) and
(mi_sample_day_dict[sample_i] >= 0 and mi_sample_day_dict[sample_j] <= 7))
# custom_cohort_tests['Week 1-Month 1'] = lambda sample_i, sample_j: ((sample_i in
infant_samples_no_olm and sample_j in infant_samples_no_olm) and
(mi_sample_day_dict[sample_i] >= 7 and mi_sample_day_dict[sample_j] <= 31))
# custom_cohort_tests['Month 1-Year 1'] = lambda sample_i, sample_j: ((sample_i in
infant_samples_no_olm and sample_j in infant_samples_no_olm) and
(mi_sample_day_dict[sample_i] >= 31 and mi_sample_day_dict[sample_j] <= 400))
custom_cohort_tests['MM'] = lambda sample_i, sample_j: (sample_i in mother_samples
and sample_j in mother_samples)
custom_cohort_tests['AA'] = lambda sample_i, sample_j: (sample_i in hmp_samples and
sample_j in hmp_samples)


# # SNP change prevalences

# In[10]:


# Make pooled snp change distributions from snp_changes

pooled_snp_change_distribution = defaultdict(list)
pooled_between_snp_change_distribution = defaultdict(list)

for species in snp_changes:
    for sample_i, sample_j in snp_changes[species]:
        # if sample_i in mother_samples and sample_j in infant_samples:
            # print("%i > %i" % (mi_sample_day_dict[sample_i],
mi_sample_day_dict[sample_j]))
        # if sample_j in mother_samples and sample_i in infant_samples:
            # print("%i < %i" % (mi_sample_day_dict[sample_i],
mi_sample_day_dict[sample_j]))

        val = snp_changes[species][(sample_i, sample_j)]
        num_snp_changes = val if (type(val) == type(1)) else len(val)
        custom_cohort = custom_cohort_of_sample_pair(custom_cohort_tests, sample_i,
sample_j)
        '''
        # MI: restrict to earliest mother-infant pair per host
        if custom_cohort == 'MI':
            if sample_i not in mi_earliest_infant_sample_dict[species]:
                continue
            if sample_j != mi_earliest_infant_sample_dict[species][sample_i]:
                continue
        '''
        pooled_snp_change_distribution[custom_cohort].append(num_snp_changes)

        num_snp_changes_between = between_snp_change_counts[species][(sample_i,
sample_j)]
        if sample_i in hmp_samples and sample_j in hmp_samples:

pooled_between_snp_change_distribution['AA'].append(num_snp_changes_between)
        else:

pooled_between_snp_change_distribution['MI'].append(num_snp_changes_between)


# In[11]:
```

```
# Figure 3: SNP prevalences

desired_event_type = 'modification'
tp_types = ['II', 'MI', 'AA']
prev_cohorts = ['nonpremie', 'hmp', 'mother']
variant_types = ['4D', '1D', '2D', '3D']

num_mod_events_by_tp_type = {tp_type: 0 for tp_type in tp_types}
num_snp_changes_by_tp_type = {tp_type: 0 for tp_type in tp_types}

prev_distribution = {prev_cohort: {tp_type: {variant_type:
np.zeros(len(derived_virtual_freqs)) for variant_type in variant_types} for tp_type
in tp_types}
                     for prev_cohort in prev_cohorts}
qp_pairs_by_freq = {prev_cohort: {tp_type: [set()] * len(derived_virtual_freqs) for
tp_type in tp_types}
                         for prev_cohort in prev_cohorts}
null_prev_distribution = {prev_cohort: {tp_type:
np.zeros(len(derived_virtual_freqs)) for tp_type in tp_types}
                          for prev_cohort in prev_cohorts}
nonsyn_opps_distribution = {prev_cohort: {tp_type: {variant_type:
np.zeros(len(derived_virtual_freqs)) for variant_type in variant_types} for tp_type
in tp_types}
                     for prev_cohort in prev_cohorts}
syn_opps_distribution = {prev_cohort: {tp_type: {variant_type:
np.zeros(len(derived_virtual_freqs)) for variant_type in variant_types} for tp_type
in tp_types}
                     for prev_cohort in prev_cohorts}

for species in snp_change_freqs:
    for sample_i, sample_j in snp_change_freqs[species]:

        tp_type = custom_cohort_of_sample_pair(custom_cohort_tests, sample_i,
sample_j)
        if tp_type not in tp_types:
            continue
        '''
        # MI: restrict to earliest mother-infant pair per host
        if tp_type == 'MI':
            if sample_i not in mi_earliest_infant_sample_dict[species]:
                continue
            if sample_j != mi_earliest_infant_sample_dict[species][sample_i]:
                continue
        '''
        pdicts = snp_change_freqs_with_opps[species][(sample_i, sample_j)] # list
of (vartype, freq_dict, opp_dict tuples)
        npdict = snp_change_null_freqs[species][(sample_i, sample_j)] # dict:
prev_cohort > list of (freq, weight) tuples

        snp_change_val = snp_changes[species][(sample_i, sample_j)]

        # How to assign dN/dS to different prevalence bins?
        # For each QP pair, there are a number of 4D and 1D opportunities,
        # and a number of SNPs that actually change each with different derived
allele prevlences
        # For each SNP change, we add its nonsyn_opps and syn_opps to running total
for
        # the prevalence bin, then divide actual numbers
        nonsyn_diffs, nonsyn_opps, syn_diffs, syn_opps =
dnds_info[species][(sample_i, sample_j)]
```

```
        if type(snp_change_val) == type(1):
            event_type = 'replacement'
        elif len(snp_change_val) > 0 and len(snp_change_val) <=
modification_difference_threshold:
            event_type = 'modification'
        else:
            continue

        if event_type != desired_event_type:
            continue

        for prev_cohort in prev_cohorts:
            for f, weight in npdict[prev_cohort]:
                f_idx = get_f_idx(f)
                null_prev_distribution[prev_cohort][tp_type][f_idx] += weight

        if len(pdicts) > 0: # Extra check that this is modification
            num_mod_events_by_tp_type[tp_type] += 1
            num_snp_changes_by_tp_type[tp_type] += len(pdicts)

        for vartype, fdict, opp_dict in pdicts:
            for prev_cohort in prev_cohorts:

                f = fdict[prev_cohort]
                f_idx = get_f_idx(f)

                prev_distribution[prev_cohort][tp_type][vartype][f_idx] += 1
                nonsyn_opps_distribution[prev_cohort][tp_type][vartype][f_idx] +=
nonsyn_opps
                syn_opps_distribution[prev_cohort][tp_type][vartype][f_idx] +=
syn_opps

                qp_pairs_by_freq[prev_cohort][tp_type][f_idx].add((sample_i,
sample_j, species))


# In[23]:


# Number of rare infant sweeping alleles that are found in mother

num_rare_infant_sweeping_alleles = 0

for species in snp_change_freqs:
    for sample_i, sample_j in snp_change_freqs[species]:
        # This is a sample pair.
        tp_type = custom_cohort_of_sample_pair(custom_cohort_tests, sample_i,
sample_j)
        if tp_type != 'II':
            continue

        pdicts = snp_change_freqs_with_opps[species][(sample_i, sample_j)] # list
of (vartype, freq_dict, opp_dict tuples)
        npdict = snp_change_null_freqs[species][(sample_i, sample_j)] # dict:
prev_cohort > list of (freq, weight) tuples

        if len(snp_change_val) > 0 and len(snp_change_val) <=
modification_difference_threshold and len(pdicts) > 0: # modification
            snp_changes = snp_changes[species][(sample_i, sample_j)]
            for snp_change in snp_changes:
            for vartype, fdict, opp_dict in pdicts:
```

```
                f = fdict['nonpremie']
                f_idx = get_f_idx(f)
                if f_idx == 0: # Rare sweeping allele
                    num_rare_infant_sweeping_alleles += 1


# In[24]:


pdicts


# In[12]:


# Info on number of SNP changes falling into each tp type category
print(num_mod_events_by_tp_type)
print(num_snp_changes_by_tp_type)


# In[13]:


# Opportunities per prevalence bin summed over QP pairs instead of SNP changes
nonsyn_opps_distribution_alt = {prev_cohort: {tp_type:
np.zeros(len(derived_virtual_freqs)) for tp_type in tp_types}
                        for prev_cohort in prev_cohorts}
syn_opps_distribution_alt = {prev_cohort: {tp_type:
np.zeros(len(derived_virtual_freqs)) for tp_type in tp_types}
                        for prev_cohort in prev_cohorts}

for prev_cohort in prev_cohorts:
    for tp_type in tp_types:
        for i in range(len(derived_virtual_freqs)):
            for qp_pair in qp_pairs_by_freq[prev_cohort][tp_type][i]:
                sample_i, sample_j, species = qp_pair
                nonsyn_diffs, nonsyn_opps, syn_diffs, syn_opps =
dnds_info[species][(sample_i, sample_j)]
                nonsyn_opps_distribution_alt[prev_cohort][tp_type][i] +=
nonsyn_opps
                syn_opps_distribution_alt[prev_cohort][tp_type][i] += syn_opps


# In[14]:


# the whole shebang (format 2)
# binned histogram of prevalences of sites
# that undergo modification
# in infants, adults, mothers with respect to themselves

# from matplotlib import rcParams
# rcParams['font.family'] = 'sans-serif'

fig, ax = plt.subplots(3, 3, figsize=(16, 10)) # sharey='row' # originally (12, 10)
for 3 x 2
colors = {'1D': 'orange', '4D': 'green', '2D': 'brown', '3D': 'maroon'}
variant_type_pretty_dict = {'1D': '1D (nonsyn)', '2D': '2D', '3D': '3D', '4D': '4D
(syn)'}

i=0
```

```
event_type = 'modification'

tp_types = ['II', 'MI', 'AA']
prev_cohorts = ['nonpremie', 'hmp', 'mother']

num_bootstraps = 10000

for tp_type in tp_types:
    j = 0
    for prev_cohort in prev_cohorts:

        # Get and plot actual counts
        cum_bin_counts = np.zeros(len(derived_virtual_freqs))
        for variant_type in ['4D', '1D', '2D', '3D']:
            bin_counts = prev_distribution[prev_cohort][tp_type][variant_type]
            ax[i][j].bar(derived_virtual_freqs, bin_counts, bottom=cum_bin_counts,
width=0.3, align='center', color=colors[variant_type])
            cum_bin_counts += bin_counts

        # Get and plot null expectation
        null_bin_counts = null_prev_distribution[prev_cohort][tp_type]
        ax[i][j].bar(derived_virtual_freqs-0.3, null_bin_counts, width=0.3,
align='center', color='0.7')

        # Get dN/dS values
        dNs = []
        for nonsyn_opps, nonsyn_count in
zip(nonsyn_opps_distribution_alt[prev_cohort][tp_type],
prev_distribution[prev_cohort][tp_type]['1D']):
            if nonsyn_opps <= 0:
                dNs.append('N/A')
                continue
            dNs.append(float(nonsyn_count)/float(nonsyn_opps))

        dSs = []
        for syn_opps, syn_count in
zip(syn_opps_distribution_alt[prev_cohort][tp_type],
prev_distribution[prev_cohort][tp_type]['4D']):
            if syn_opps <= 0:
                dSs.append('N/A')
                continue
            dSs.append(float(syn_count)/float(syn_opps))

        dNdSs = []
        for dN, dS in zip(dNs, dSs):
            try:
                dNdSs.append('%.02f' % (dN/float(dS)))
            except:
                dNdSs.append('N/A') # Dummy number

        # Get bootstrapped dN/dS values
        non_ns = prev_distribution[prev_cohort][tp_type]['1D']
        syn_ns = prev_distribution[prev_cohort][tp_type]['4D']
        non_opportunities = nonsyn_opps_distribution_alt[prev_cohort][tp_type]
        syn_opportunities = syn_opps_distribution_alt[prev_cohort][tp_type]

        total_ns = (non_ns + syn_ns).astype('int64')
        ps = non_ns*1.0/total_ns

        bootstrapped_dNdSs = []
```

```
        for bootstrap_idx in range(0,num_bootstraps):
            bootstrapped_non_ns = sample_binomial(total_ns, ps)
            bootstrapped_syn_ns = total_ns - bootstrapped_non_ns
            bootstrapped_dNdSs.append( bootstrapped_non_ns*1.0/bootstrapped_syn_ns
/ (non_opportunities*1.0/syn_opportunities) )

        bootstrapped_dNdSs = np.sort( bootstrapped_dNdSs, axis=0 )
        bootstrapped_dNdSs_lower = bootstrapped_dNdSs[int(0.025*num_bootstraps),:]
        bootstrapped_dNdSs_upper = bootstrapped_dNdSs[int(0.975*num_bootstraps),:]

        # Show dN/dS values
        for k in range(len(dNdSs)):
            dNdS = dNdSs[k]
            dNdS_lower = bootstrapped_dNdSs_lower[k]
            dNdS_upper = bootstrapped_dNdSs_upper[k]
            dNdS_CI_str = '%.02f\n%.02f\n\n' % (dNdS_lower, dNdS_upper)
            dNdS_with_CI_str = '%.02f\n%.02f\n\n%s' % (dNdS_lower, dNdS_upper,
dNdS)
            # ax[i][j].text(k-0.1, cum_bin_counts[k] + (max(cum_bin_counts)*0.04),
dNdS_with_CI_str, color='gray')
            ax[i][j].text(k-0.16, cum_bin_counts[k] + (max(cum_bin_counts)*0.04),
dNdS, fontsize=12, color='blue')
            ax[i][j].text(k-0.16, cum_bin_counts[k] + (max(cum_bin_counts)*0.04),
dNdS_CI_str, fontsize=12, color='gray')
        # xtick_labels_with_dNdS = ['%s\ndN/dS:\n%s' % (freq, dNdS) for freq, dNdS
in zip(derived_virtual_xticklabels, dNdSs)]

        ax[i][j].tick_params(axis='both', which='major', labelsize=13)
        ax[i][j].set_xticks(derived_virtual_freqs + 0.25)
        ax[i][j].set_xticklabels(derived_virtual_xticklabels, fontsize=13)
        ax[i][j].set_ylabel("SNV count", fontsize=14)
        ax[i][j].set_xlabel("Derived allele prevalence", fontsize=14)
        ax[i][j].set_title("n=%i" % sum(cum_bin_counts), fontsize=14)
        if j == 0:
            tp_type_label_dict = {'II': 'Infant-Infant', 'MI': 'Mother-Infant',
'AA': 'Adult-Adult'}
            ax[i][j].text(-0.2, 0.5, tp_type_label_dict[tp_type], fontsize=18,
transform=ax[i][j].transAxes, ha='right')
        if i == 0:
            prev_cohort_label_dict = {'nonpremie': 'Prev. cohort: infants',
                                      'hmp': 'Prev. cohort: HMP',
                                      'mother': 'Prev. cohort: mothers'}
            ax[i][j].text(0.5, 1.2, prev_cohort_label_dict[prev_cohort],
ha='center', fontsize=18, transform=ax[i][j].transAxes)

        j += 1
    i += 1

# ax[0][0].text(1.0, 1.4, "%s sweeps" % sweep_type, ha='center', fontsize=20,
transform=ax[0][0].transAxes)
for u in range(len(prev_cohorts)):
    ax[0][u].set_ylim((0,420))
    ax[1][u].set_ylim((0,40))
    ax[2][u].set_ylim((0,260))

# Legend
legend_elements = [Patch(facecolor=colors[vt], label=variant_type_pretty_dict[vt])
for vt in ['4D', '1D', '2D', '3D']] + [Patch(facecolor='0.7', label='Total null
exp.')]
ax[0][0].legend(handles=legend_elements, loc='upper center', frameon=False)
```

```
plt.subplots_adjust(hspace=0.6)
plt.tight_layout()
plt.show()
fig.savefig('%s/modification_snp_prevs_%s.pdf' % (config.analysis_directory,
sweep_type), bbox_inches='tight')


# In[15]:


vartypes = ['1D', '2D', '3D', '4D']
print(sum([prev_distribution['nonpremie']['II'][vartype][0] for vartype in
vartypes]))
print(sum([prev_distribution['hmp']['II'][vartype][0] for vartype in vartypes]))
print(sum([prev_distribution['mother']['II'][vartype][0] for vartype in vartypes]))


# In[38]:


total = 0
for vartype in prev_distribution['mother']['II']:
    total += sum(prev_distribution['mother']['II'][vartype])
print(total)


# In[15]:


prev_distribution['hmp']['II']


# # Gene change prevalences

# In[15]:


# V1

prev_cohorts = ['nonpremie', 'hmp']
tp_types = ['II', 'MI', 'AA']

gene_gain_prev_distribution = {prev_cohort: {tp_type:
np.zeros(len(gene_gain_virtual_freqs)) for tp_type in tp_types}
                     for prev_cohort in prev_cohorts}
gene_loss_prev_distribution = {prev_cohort: {tp_type:
np.zeros(len(gene_loss_virtual_freqs)) for tp_type in tp_types}
                     for prev_cohort in prev_cohorts}
gene_loss_null_prev_distribution = {prev_cohort: {tp_type:
np.zeros(len(gene_loss_virtual_freqs)) for tp_type in tp_types}
                          for prev_cohort in prev_cohorts}

for prev_cohort in prev_cohorts:
    for desired_tp_type in tp_types:

        total_freq_gains = np.zeros(len(gene_freq_bins)-1)*1.0
        total_freq_losses = np.zeros(len(gene_freq_bins)-1)*1.0
        total_null_freq_losses = np.zeros(len(gene_freq_bins)-1)*1.0

        for species in gene_gain_freqs:
            for sample_i, sample_j in gene_gain_freqs[species]:
```

```
                tp_pair = su.sample_pair_to_tp_pair(sample_i, sample_j,
sample_order_map, hmp_samples, mother_samples)
                tp_type = tp_pair_to_tp_type(tp_pair)
                if tp_type != desired_tp_type:
                    continue
                for f_dict in gene_gain_freqs[species][(sample_i, sample_j)]:
                    f = f_dict[prev_cohort] if prev_cohort in f_dict else 0
                    f_idx = ((f>gene_freq_bins[:-
1])*(f<=gene_freq_bins[1:])).argmax()
                    total_freq_gains[f_idx] += 1

        for species in gene_loss_freqs:
            for sample_i, sample_j in gene_loss_freqs[species]:
                tp_pair = su.sample_pair_to_tp_pair(sample_i, sample_j,
sample_order_map, hmp_samples, mother_samples)
                tp_type = tp_pair_to_tp_type(tp_pair)
                if tp_type != desired_tp_type:
                    continue
                for f_dict in gene_loss_freqs[species][(sample_i, sample_j)]:
                    f = f_dict[prev_cohort] if prev_cohort in f_dict else 0
                    f_idx = ((f>gene_freq_bins[:-
1])*(f<=gene_freq_bins[1:])).argmax()
                    total_freq_losses[f_idx] += 1

        for species in gene_loss_null_freqs:
            for sample_i, sample_j in gene_loss_null_freqs[species]:
                tp_pair = su.sample_pair_to_tp_pair(sample_i, sample_j,
sample_order_map, hmp_samples, mother_samples)
                tp_type = tp_pair_to_tp_type(tp_pair)
                if tp_type != desired_tp_type:
                    continue
                for f_dict in gene_loss_null_freqs[species][(sample_i, sample_j)]:
                    if prev_cohort not in f_dict:
                        continue # ??? TODO
                    for f in f_dict[prev_cohort]:
                        f_idx = ((f>gene_freq_bins[:-
1])*(f<=gene_freq_bins[1:])).argmax()
                        total_null_freq_losses[f_idx] += (1.0/num_bootstraps)

        gene_gain_prev_distribution[prev_cohort][desired_tp_type] =
total_freq_gains
        gene_loss_prev_distribution[prev_cohort][desired_tp_type] =
total_freq_losses
        gene_loss_null_prev_distribution[prev_cohort][desired_tp_type] =
total_null_freq_losses


# In[16]:


# V2
prev_cohorts = ['infant', 'hmp']
tp_types = ['II', 'MI', 'AA']

gene_gain_prev_distribution = {prev_cohort: {tp_type:
np.zeros(len(gene_gain_virtual_freqs)) for tp_type in tp_types}
                     for prev_cohort in prev_cohorts}
gene_loss_prev_distribution = {prev_cohort: {tp_type:
np.zeros(len(gene_loss_virtual_freqs)) for tp_type in tp_types}
                     for prev_cohort in prev_cohorts}
```

```
gene_loss_null_prev_distribution = {prev_cohort: {tp_type:
np.zeros(len(gene_loss_virtual_freqs)) for tp_type in tp_types}
                          for prev_cohort in prev_cohorts}
gene_loss_null_prev_distribution_alt = {prev_cohort: {tp_type:
np.zeros(len(gene_loss_virtual_freqs)) for tp_type in tp_types}
                          for prev_cohort in prev_cohorts}

gain_qp_pairs = []
loss_qp_pairs = []

# Note: sometimes gene_gain_freqs[species][(sample_i, sample_j)]
# (or gene_loss_freqs) will not have full set of prev cohorts
# because when the data is being pickled,
# it loops over prev_cohort keys in gene_freq_map
# which is generated for each species and will exclude a prev_cohort
# if gene_freqs returned by core_gene_utils.parse_gene_freqs has length 0
# which will happen if the file doesn't exist or it's just empty
# Indeed, number of species with gene_freqs information is halved
# in hmp/prev_hmp compared to infant/prev_infant

for species in gene_gain_freqs:
    for sample_i, sample_j in gene_gain_freqs[species]:

        tp_type = custom_cohort_of_sample_pair(custom_cohort_tests, sample_i,
sample_j)
        if tp_type not in tp_types:
            continue
        '''
        # MI: restrict to earliest mother-infant pair per host
        if tp_type == 'MI':
            if sample_i not in mi_earliest_infant_sample_dict[species]:
                continue
            if sample_j != mi_earliest_infant_sample_dict[species][sample_i]:
                continue
        '''
        gain_qp_pairs.append((species, sample_i, sample_j))

        for gene_gain_freq_dict in gene_gain_freqs[species][(sample_i, sample_j)]:
            for prev_cohort in gene_gain_freq_dict:
                f = gene_gain_freq_dict[prev_cohort]
                f_idx = ((f>gene_freq_bins[:-1])*(f<=gene_freq_bins[1:])).argmax()
                gene_gain_prev_distribution[prev_cohort][tp_type][f_idx] += 1


for species in gene_loss_freqs:
    for sample_i, sample_j in gene_loss_freqs[species]:

        tp_type = custom_cohort_of_sample_pair(custom_cohort_tests, sample_i,
sample_j)
        if tp_type not in tp_types:
            continue

        # MI: restrict to earliest mother-infant pair per host
        if tp_type == 'MI':
            if sample_i not in mi_earliest_infant_sample_dict[species]:
                continue
            if sample_j != mi_earliest_infant_sample_dict[species][sample_i]:
                continue

        loss_qp_pairs.append((species, sample_i, sample_j))
```

```
        for gene_loss_freq_dict in gene_loss_freqs[species][(sample_i, sample_j)]:
            for prev_cohort in gene_loss_freq_dict:
                f = gene_loss_freq_dict[prev_cohort]
                f_idx = ((f>gene_freq_bins[:-1])*(f<=gene_freq_bins[1:])).argmax()
                gene_loss_prev_distribution[prev_cohort][tp_type][f_idx] += 1

        num_bootstraps = 10

        for gene_loss_null_freq_dict in gene_loss_null_freqs[species][(sample_i,
sample_j)]:
            for prev_cohort in gene_loss_null_freq_dict:
                bootstrapped_fs = gene_loss_null_freq_dict[prev_cohort]
                f = (sum(bootstrapped_fs)/float(num_bootstraps))
                f_idx = ((f>gene_freq_bins[:-1])*(f<=gene_freq_bins[1:])).argmax()
                # gene_loss_null_prev_distribution[prev_cohort][tp_type][f_idx] +=
1 # I think this was incorrect

                for f in bootstrapped_fs:
                    f_idx = ((f>gene_freq_bins[:-
1])*(f<=gene_freq_bins[1:])).argmax()
                    gene_loss_null_prev_distribution[prev_cohort][tp_type][f_idx]
+= (1.0/num_bootstraps)


# In[17]:


fig, ax = plt.subplots(3, 2, figsize=(8, 6)) # sharey='row' # originally (12, 10)
for 3 x 2
colors = {'1D': 'orange', '4D': 'green', '2D': 'brown', '3D': 'maroon'}

i=0
event_type = 'modification'

tp_types = ['II', 'MI', 'AA']
prev_cohorts = ['nonpremie', 'hmp']

for tp_type in tp_types:
    j = 0
    for prev_cohort in prev_cohorts:

        ax[i][j].spines['top'].set_visible(False);
ax[i][j].spines['right'].set_visible(False)
        ax[i][j].get_xaxis().tick_bottom(); ax[i][j].get_yaxis().tick_left()
        ax[i][j].set_xlabel('Gene prevalence across hosts')
        ax[i][j].set_ylabel('# gene changes')
        ax[i][j].set_xlim([gene_freq_xticks[0],gene_freq_xticks[-1]])
        ax[i][j].set_xticks(gene_freq_xticks)
        ax[i][j].set_xticklabels(gene_freq_xticklabels)
        ax[i][j].plot([0,0],[100,100],'k-')

        # Really sloppy but need to change nonpremie to infant for gene info
        prev_cohort = 'infant' if prev_cohort == 'nonpremie' else prev_cohort
        ax[i][j].bar(gene_gain_virtual_freqs+0.15,
gene_gain_prev_distribution[prev_cohort][tp_type],
                     width=0.3,linewidth=0,facecolor='#b3de69',label='gain')
        ax[i][j].bar(gene_loss_virtual_freqs+0.15,
gene_loss_prev_distribution[prev_cohort][tp_type],
                     width=0.3,linewidth=0, facecolor='#ff7f00',label='loss')
        ax[i][j].bar(gene_loss_virtual_freqs-0.3+0.15,
gene_loss_null_prev_distribution[prev_cohort][tp_type],
```

```
                    width=0.3,linewidth=0, facecolor='0.7',label='de
novo\nexpectation')

        print(sum(gene_loss_prev_distribution[prev_cohort][tp_type]))
        print(sum(gene_loss_null_prev_distribution[prev_cohort][tp_type]))

        # ax[i][j].set_title("# gains=%i, # losses=%i" %
(sum(gene_gain_prev_distribution[prev_cohort][tp_type]),
        #
sum(gene_loss_prev_distribution[prev_cohort][tp_type])))
        if j == 0:
            tp_type_label_dict = {'II': 'Infant-Infant', 'MI': 'Mother-Infant',
'AA': 'Adult-Adult'}
            ax[i][j].text(-0.26, 0.5, tp_type_label_dict[tp_type], fontsize=14,
transform=ax[i][j].transAxes, ha='right')
        if i == 0:
            prev_cohort_label_dict = {'infant': 'Prev. cohort: infants',
                                      'hmp': 'Prev. cohort: HMP'}
            ax[i][j].text(0.5, 1.2, prev_cohort_label_dict[prev_cohort],
ha='center', fontsize=14, transform=ax[i][j].transAxes)

        ax[1][j].set_ylim((0, 25))

        j += 1
    i += 1

legend_elements = [Patch(facecolor='#b3de69', label='Gain'),
Patch(facecolor='#ff7f00', label='Loss')] + [Patch(facecolor='0.7', label='Loss
null exp.')]
ax[1][0].legend(handles=legend_elements, loc='upper center', frameon=False)

plt.subplots_adjust(hspace=0.6)
plt.tight_layout()
plt.show()
fig.savefig('%s/modification_gene_prevs_%s.pdf' % (config.analysis_directory,
sweep_type), bbox_inches='tight')


# In[18]:


# Get statistics for paper

prev_cohort = 'hmp'
tp_type = 'MI'

gene_losses = gene_loss_prev_distribution[prev_cohort][tp_type]
total = sum(gene_losses)
lowprev = gene_losses[0]

print(lowprev/float(total))


# In[19]:


def remove_top_right_spines(ax):
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()
```

```
# In[20]:


# Final figure

fig, ax = plt.subplots(3, 2, figsize=(8, 8)) # sharey='row' # originally (12, 10)
for 3 x 2
colors = {'1D': 'orange', '4D': 'green', '2D': 'brown', '3D': 'maroon'} # {'1D':
'#ff7f00', '4D': '#b3de69', '2D': '#b15928', '3D': '#7c3e1c'}
variant_type_pretty_dict = {'1D': '1D (nonsyn)', '2D': '2D', '3D': '3D', '4D': '4D
(syn)'}
letters = {0: {0: 'A', 1: 'B'}, 1: {0: 'C', 1: 'D'}, 2: {0: 'E', 1: 'F'}}

event_type = 'modification'

tp_types = ['II', 'MI', 'AA']
tp_types_pretty_dict = {'II': 'Infant-Infant', 'MI': 'Mother-Infant', 'AA': 'HMP
Adults'}
prev_cohorts = ['nonpremie', 'hmp', 'mother']

num_bootstraps = 10000

# First plot prevalence of II, MI, HMP changing SNPs w.r.t. HMP prev cohort

tt_i = 0
for tp_type in tp_types:

    prev_cohort = 'hmp' # Just looking at HMP prev cohort

    # Get and plot actual counts
    cum_bin_counts = np.zeros(len(derived_virtual_freqs))
    for variant_type in ['4D', '1D', '2D', '3D']:
        bin_counts = prev_distribution[prev_cohort][tp_type][variant_type]
        ax[tt_i][1].bar(derived_virtual_freqs, bin_counts, bottom=cum_bin_counts,
width=0.3, align='center', color=colors[variant_type])
        cum_bin_counts += bin_counts

    # Get and plot null expectation
    null_bin_counts = null_prev_distribution[prev_cohort][tp_type]
    ax[tt_i][1].bar(derived_virtual_freqs-0.3, null_bin_counts, width=0.3,
align='center', color='0.7')

    # Labels/formatting
    remove_top_right_spines(ax[tt_i][0])
    ax[tt_i][1].set_xticks(derived_virtual_freqs + 0.25)
    ax[tt_i][1].set_xticklabels(derived_virtual_xticklabels)
    ax[tt_i][1].set_ylabel("SNV count", fontsize=13)
    ax[tt_i][1].set_xlabel("Derived allele prevalence", fontsize=13)
    ax[tt_i][1].text(-0.25, 0.92, letters[tt_i][0], size=20,
transform=ax[tt_i][0].transAxes, weight='bold')

    # Plot gene gains and losses
    ax[tt_i][0].bar(gene_gain_virtual_freqs+0.2,
gene_gain_prev_distribution[prev_cohort][tp_type],
                width=0.3,linewidth=0,facecolor='#b3de69',label='gain')
    ax[tt_i][0].bar(gene_loss_virtual_freqs+0.2,
gene_loss_prev_distribution[prev_cohort][tp_type],
                width=0.3,linewidth=0, facecolor='#ff7f00',label='loss')
```

```
    ax[tt_i][0].bar(gene_loss_virtual_freqs+0.2-0.3,
gene_loss_null_prev_distribution[prev_cohort][tp_type],
                width=0.3,linewidth=0, facecolor='0.7',label='de
novo\nexpectation')

    # Labels/formatting
    remove_top_right_spines(ax[tt_i][1])
    ax[tt_i][0].set_xlabel('Gene prevalence across hosts', fontsize=13)
    ax[tt_i][0].set_ylabel('# gene changes', fontsize=13)
    ax[tt_i][0].set_xlim([gene_freq_xticks[0],gene_freq_xticks[-1]])
    ax[tt_i][0].set_xticks(gene_freq_xticks)
    ax[tt_i][0].set_xticklabels(gene_freq_xticklabels)
    ax[tt_i][0].plot([0,0],[100,100],'k-')
    ax[tt_i][0].text(-0.25, 0.92, letters[tt_i][1], size=20,
transform=ax[tt_i][1].transAxes, weight='bold')

    # TP type label on the side
    ax[tt_i][0].text(-0.25, 0.44, tp_types_pretty_dict[tp_type], fontsize=14,
ha='right', transform=ax[tt_i][0].transAxes)

    tt_i += 1

# Legends
legend_elements = [Patch(facecolor=colors[vt], label=variant_type_pretty_dict[vt])
for vt in ['4D', '1D', '2D', '3D']] + [Patch(facecolor='0.7', label='Total null
exp.')]
ax[0][1].legend(handles=legend_elements, loc='upper center', frameon=False)

legend_elements = [Patch(facecolor='#b3de69', label='Gain'),
Patch(facecolor='#ff7f00', label='Loss')] + [Patch(facecolor='0.7', label='Loss
null exp.')]
ax[0][0].legend(handles=legend_elements, loc='upper center', frameon=False)

ax[1][0].set_ylim((0, 20))

# plt.subplots_adjust(hspace=0.6)
plt.tight_layout()
plt.show()
fig.savefig('%s/figure_3_v2.pdf' % (config.analysis_directory),
bbox_inches='tight')
fig.savefig('%s/figure_3_v2.png' % (config.analysis_directory), dpi=500,
bbox_inches='tight')


# In[ ]:


alt_num_snp_changes_by_tp_type = defaultdict(int)

for tp_type in prev_distribution['hmp']:
    for variant_type in prev_distribution[prev_cohort][tp_type]:
        alt_num_snp_changes_by_tp_type[tp_type] +=
sum(prev_distribution[prev_cohort][tp_type][variant_type])

print(alt_num_snp_changes_by_tp_type)


# In[21]:


# Heatmap of how SNP changes change from adult prev to infant prev
```

```
desired_event_type = 'modification'
tp_types = ['II', 'MI', 'AA']
prev_cohorts = ['nonpremie', 'hmp']
variant_types = ['4D', '1D', '2D', '3D']

# List of (hmp_f_idx, nonpremie_f_idx, weight) tuples for each SNP change
prev_compare_distribution = {tp_type: [] for tp_type in tp_types}
null_prev_compare_distribution = {tp_type: [] for tp_type in tp_types}

for species in snp_change_freqs:
    for sample_i, sample_j in snp_change_freqs[species]:

        tp_type = custom_cohort_of_sample_pair(custom_cohort_tests, sample_i,
sample_j)
        if tp_type not in tp_types:
            continue

        pdicts = snp_change_freqs_with_opps[species][(sample_i, sample_j)] # list
of (vartype, freq_dict, opp_dict tuples)
        npdict = snp_change_null_freqs[species][(sample_i, sample_j)] # dict:
prev_cohort > list of (freq, weight) tuples

        snp_change_val = snp_changes[species][(sample_i, sample_j)]

        if type(snp_change_val) == type(1):
            event_type = 'replacement'
        elif len(snp_change_val) > 0 and len(snp_change_val) <=
modification_difference_threshold:
            event_type = 'modification'
        else:
            continue

        if event_type != desired_event_type:
            continue

        '''

        for tup1, tup2 in zip(npdict['hmp'], npdict['nonpremie']):
            f1, weight1 = tup1; f2, weight2 = tp2
            f1_idx = get_f_idx(f1); f2_idx = get_f_idx(f2)
            null_prev_compare_distribution[tp_type].append((f1_idx, f2_idx,
weight))
        '''

        # Loop over SNP changes
        for vartype, fdict, opp_dict in pdicts:
            prev_compare_distribution[tp_type].append((get_f_idx(fdict['hmp']),
get_f_idx(fdict['nonpremie']), 1))


# In[22]:


prev_compare_matrix = {tp_type: np.zeros((len(derived_virtual_freqs),
len(derived_virtual_freqs))) for tp_type in tp_types}

for tp_type in tp_types:

    for f_idx_hmp, f_idx_nonpremie, weight in prev_compare_distribution[tp_type]:
```

```
        # HMP idx selects the row, nonpremie selects column

        prev_compare_matrix[tp_type][f_idx_hmp][f_idx_nonpremie] += weight


# In[27]:


fig, ax = plt.subplots(1, 3, figsize=(12, 4))

vmin = 0
vmax = 186

im0 = ax[0].imshow(prev_compare_matrix['II'], cmap='pink', vmin=vmin, vmax=vmax)
ax[0].set_title("Infant-infant")
ax[0].set_xlabel("Infant prevalence")
ax[0].set_xticks(derived_virtual_freqs[:-1]+0.5)
ax[0].set_xticklabels(derived_virtual_xticklabels)
ax[0].set_yticks(derived_virtual_freqs[:-1]+0.5)
ax[0].set_yticklabels(derived_virtual_xticklabels)
ax[0].set_ylabel("HMP prevalence")
ax[0].text(-0.14, 1.04, 'A', size=20, transform=ax[0].transAxes, weight='bold')

im1 = ax[1].imshow(prev_compare_matrix['MI'], cmap='pink', vmin=vmin, vmax=vmax)
ax[1].set_title("Mother-infant")
ax[1].set_xlabel("Infant prevalence")
ax[1].set_xticks(derived_virtual_freqs[:-1]+0.5)
ax[1].set_xticklabels(derived_virtual_xticklabels)
ax[1].set_yticks(derived_virtual_freqs[:-1]+0.5)
ax[1].set_yticklabels(derived_virtual_xticklabels)
# ax[1].set_ylabel("HMP prevalence")
ax[1].text(-0.14, 1.04, 'B', size=20, transform=ax[1].transAxes, weight='bold')

im2 = ax[2].imshow(prev_compare_matrix['AA'], cmap='pink', vmin=vmin, vmax=vmax)
ax[2].set_title("Adult-Adult")
ax[2].set_xlabel("Infant prevalence")
ax[2].set_xticks(derived_virtual_freqs[:-1]+0.5)
ax[2].set_xticklabels(derived_virtual_xticklabels)
ax[2].set_yticks(derived_virtual_freqs[:-1]+0.5)
ax[2].set_yticklabels(derived_virtual_xticklabels)
# ax[2].set_ylabel("HMP prevalence")
ax[2].text(-0.14, 1.04, 'C', size=20, transform=ax[2].transAxes, weight='bold')

fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.82, 0.15, 0.02, 0.7])
fig.colorbar(im2, cax=cbar_ax)
ax[2].text(1.4, 0.62, 'SNV count', transform=ax[2].transAxes, rotation=270,
fontsize=12)

plt.show()
fig.savefig('%s/hmp_vs_infant_prev_heatmaps.pdf' % (config.analysis_directory),
bbox_inches='tight')
fig.savefig('%s/hmp_vs_infant_prev_heatmaps.png' % (config.analysis_directory),
dpi=500, bbox_inches='tight')


# In[43]:


derived_virtual_xticklabels
```

```
# In[52]:


derived_virtual_freqs
```

plot_figure_4-5.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import config, parse_midas_data, sample_utils as su, temporal_changes_utils, stats_utils,
midas_db_utils, parse_patric, plot_utils
from collections import defaultdict
import numpy as np
from numpy.random import binomial as sample_binomial
import math, pickle, sys, random
from math import log10,ceil,log,exp

import scipy.stats
import matplotlib.colorbar as colorbar
import matplotlib.colors as mplcol
import matplotlib.pyplot as plt
import matplotlib.cm as cmx
from matplotlib.patches import Patch
from matplotlib import rcParams
from matplotlib import gridspec
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Arial']


# In[ ]:


# Plot directory
plot_dir = "%s/" % (config.analysis_directory)

# Species list
good_species_list = parse_midas_data.load_pickled_good_species_list()

# Sample-subject-order-cohort maps
sys.stderr.write("Loading sample metadata...\n")
subject_sample_map = su.parse_subject_sample_map()
sample_order_map = su.parse_sample_order_map()
sample_subject_map = su.parse_sample_subject_map()
sample_cohort_map = su.parse_sample_cohort_map()
same_mi_pair_dict = su.get_same_mi_pair_dict(subject_sample_map)
sys.stderr.write("Done!\n")

# Cohorts
cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
mi_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

# Samples for each cohort
samples = {cohort: su.get_sample_names(cohort) for cohort in cohorts}
hmp_samples = su.get_sample_names('hmp')
mother_samples = su.get_sample_names('mother')
```

```
infant_samples = su.get_sample_names('infant')
olm_samples = su.get_sample_names('olm')
infant_samples = [sample for sample in infant_samples if sample not in olm_samples] # Exclude
Olm
mother_samples = [sample for sample in mother_samples if sample not in olm_samples] # Exclude
Olm
mi_samples = [sample for sample in (mother_samples + infant_samples) if sample not in
olm_samples] # Exclude Olm

# Sample identity wrapper functions
is_infant = lambda sample: sample in infant_samples
is_mother = lambda sample: sample in mother_samples
is_adult = lambda sample: sample in hmp_samples

# Sample-timepoint map
mi_sample_day_dict = su.get_mi_sample_day_dict(exclude_cohorts=['olm'])
mi_tp_sample_dict = su.get_mi_tp_sample_dict(exclude_cohorts=['olm']) # no binning
mi_tp_sample_dict_binned, mi_tp_binned_labels =
su.get_mi_tp_sample_dict(exclude_cohorts=['olm'], binned=True)


# In[ ]:


# Significance tests
# function to calculate Cohen's d for independent samples
def cohenD(d1, d2):
    n1, n2 = len(d1), len(d2)
    s1, s2 = np.var(d1, ddof=1), np.var(d2, ddof=1)
    s = np.sqrt(((n1 - 1) * s1 + (n2 - 1) * s2) / (n1 + n2 - 2))
    u1, u2 = np.mean(d1), np.mean(d2)
    return (u1 - u2) / s


# In[ ]:


# ======================================================================
# Load pickled data
# ======================================================================

# Parameters
sweep_type = 'full' # assume full for now
pp_prev_cohort = 'all'
min_coverage = 0

ddir = config.data_directory
pdir = "%s/pickles/cov%i_prev_%s/nonconsecutive" % (ddir, min_coverage, pp_prev_cohort)


# In[ ]:


snp_changes = pickle.load(open('%s/big_snp_changes_%s.pkl' % (pdir, sweep_type), 'rb'))
gene_changes = pickle.load(open('%s/big_gene_changes_%s.pkl' % (pdir, sweep_type), 'rb'))
snp_change_freqs = pickle.load(open('%s/snp_change_freqs_%s.pkl' % (pdir, sweep_type), 'rb'))
snp_change_freqs_with_opps = pickle.load(open('%s/snp_change_freqs_with_opps_full.pkl' % (pdir),
'rb'))
snp_change_null_freqs = pickle.load(open('%s/snp_change_null_freqs_%s.pkl' % (pdir, sweep_type),
'rb'))
print("Checkpoint 1")
```

```
gene_gain_freqs = pickle.load(open('%s/gene_gain_freqs_%s.pkl' % (pdir, sweep_type), 'rb'))
gene_loss_freqs = pickle.load(open('%s/gene_loss_freqs_%s.pkl' % (pdir, sweep_type), 'rb'))
gene_loss_null_freqs = pickle.load(open('%s/gene_loss_null_freqs_%s.pkl' % (pdir, sweep_type),
'rb'))
between_snp_change_counts = pickle.load(open('%s/between_snp_change_counts_%s.pkl' % (pdir,
sweep_type), 'rb'))
between_gene_change_counts = pickle.load(open('%s/between_gene_change_counts_%s.pkl' % (pdir,
sweep_type), 'rb'))
print("Checkpoint 2")


# In[ ]:


snp_change_present_gene_null = pickle.load(open('%s/snp_change_present_gene_null' % pdir, 'rb'))
snp_change_between_host_null = pickle.load(open('%s/snp_change_between_host_null' % pdir, 'rb'))
snp_change_pangenome_null = pickle.load(open('%s/snp_change_pangenome_null' % pdir, 'rb'))
print("Checkpoint 3")
dnds_info = pickle.load(open('%s/dnds_info.pkl' % (pdir), 'rb'))
print("Checkpoint 4")


# In[ ]:


# Consecutive data

# Parameters
sweep_type = 'full' # assume full for now
pp_prev_cohort = 'all'
min_coverage = 0

ddir = config.data_directory
pdir = "%s/pickles/cov%i_prev_%s" % (ddir, min_coverage, pp_prev_cohort)

snp_changes_consecutive = pickle.load(open('%s/big_snp_changes_%s.pkl' % (pdir, sweep_type),
'rb'))
gene_changes_consecutive = pickle.load(open('%s/big_gene_changes_%s.pkl' % (pdir, sweep_type),
'rb'))
print("Checkpoint 5")


# In[ ]:


# seeding analysis pickles
pdir = "%s/pickles/seeding" % (ddir)
species_allele_counts_map = {}
for species in snp_changes.keys():
    fpath = "%s/allele_counts_map_%s.pkl" % (pdir, species)
    species_allele_counts_map[species] = pickle.load(open(fpath, 'rb'))
print("Checkpoint 6")


# In[11]:


# Custom sample pair cohorts [not just sample!]
# Alternate version where a sample pair may be assigned multiple cohorts

# custom_cohort_tests: dictionary where keys are cohort names and values are
```

```
# boolean functions taking in sample_i, sample_j as arguments

def custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j, species='dummy'):
    my_cohorts = set()
    for cohort in custom_cohort_tests:
        if custom_cohort_tests[cohort](sample_i, sample_j, species):
            my_cohorts.add(cohort)
    return my_cohorts


# In[12]:


# Establish custom cohorts to be used for all three rates plots

custom_cohort_tests = {}
custom_cohort_tests['Mother-Infant(wk1)']  = lambda s1, s2, sp: ((is_mother(s1) and
is_infant(s2))
                                                                and mi_sample_day_dict[s1] >= 0
                                                                and mi_sample_day_dict[s2] <= 7)

custom_cohort_tests['Mother-Infant(all)']  = lambda s1, s2, sp: ((is_mother(s1) and
is_infant(s2)))

custom_cohort_tests['Day 0-Week 1']   = lambda s1, s2, sp: ((is_infant(s1) and is_infant(s2))
                                                                and mi_sample_day_dict[s1] >= 0
                                                                and mi_sample_day_dict[s2] <= 7)

custom_cohort_tests['Week 1-Month 1'] = lambda s1, s2, sp: ((is_infant(s1) and is_infant(s2))
                                                                and mi_sample_day_dict[s1] >= 7
                                                                and mi_sample_day_dict[s2] <= 31)

custom_cohort_tests['Month 1-Year 1'] = lambda s1, s2, sp: ((is_infant(s1) and is_infant(s2))
                                                                and mi_sample_day_dict[s1] >= 31
                                                                and mi_sample_day_dict[s2] <= 367)

custom_cohort_tests['Adult-Adult']   = lambda s1, s2, sp: (is_adult(s1) and is_adult(s2))

custom_cohort_tests['Infant-Infant'] = lambda s1, s2, sp: (is_infant(s1) and is_infant(s2))


# In[13]:


# species -> mother_sample -> infant_sample -> # days of infant_sample
# mother timepoint at delivery (-1 to 7 days)
mi_dict = {species: defaultdict(dict) for species in gene_gain_freqs}

for species in snp_changes:
    for sample_i, sample_j in snp_changes[species]:

        # MI: restrict to earliest mother-infant pair per host
        if 'Mother-Infant(all)' in custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i,
sample_j):
            mother_days = mi_sample_day_dict[sample_i]
            if mother_days >= -1 and mother_days <= 7:
                mi_dict[species][sample_i][sample_j] = mi_sample_day_dict[sample_j]

# species -> mother_sample -> earliest infant sample
mi_earliest_infant_sample_dict = defaultdict(dict)
```

```
# Distribution of which day the earliest infant tp occurs at
# Hopefully nothing too late
earliest_infant_days = []

for species in mi_dict:
    for mother_sample in mi_dict[species]:
        ordered_infant_days = sorted(mi_dict[species][mother_sample].items(), key=lambda x:
x[1])
        infant_sample, days = ordered_infant_days[0]
        mi_earliest_infant_sample_dict[species][mother_sample] = infant_sample
        earliest_infant_days.append(days)


# In[14]:


# Add one more category
custom_cohort_tests['Mother-Infant(earliest)']  = lambda s1, s2, sp: ((is_mother(s1) and
is_infant(s2)
                                                                      and (s1 in
mi_earliest_infant_sample_dict[sp])
                                                                      and (s2 ==
mi_earliest_infant_sample_dict[sp][s1])))


# In[15]:


def calculate_unnormalized_survival_from_vector(counts):
    counts = sorted(counts)
    xs = []
    ns = []
    ns_cur = len(counts)
    min_count = -1
    for count in counts:
        if count > min_count:
            ns.append(ns_cur) # Number of elements greater or equal
            xs.append(count)
            min_count = count
        ns_cur -= 1
    xs.append(xs[len(xs)-1]+1)
    ns.append(0)
    return xs, np.array(ns)


# # SNP change parallelism

# In[ ]:


# Parallelism (for SNP changes) at gene annotation and gene ID level

custom_cohorts_ordered = list(custom_cohort_tests.keys())
variant_types = ['4D', '1D', '2D', '3D']

# cohort -> gene ID -> variant type -> count of SNP changes
gene_IDs_by_cohort = {cohort: {} for cohort in custom_cohort_tests}
gene_IDs_by_cohort_present_null = {cohort: defaultdict(int) for cohort in
custom_cohorts_ordered}
gene_IDs_by_cohort_between_null = {cohort: defaultdict(int) for cohort in
custom_cohorts_ordered}
```

```
gene_IDs_by_cohort_pangenome_null = {cohort: defaultdict(int) for cohort in
custom_cohorts_ordered}
# cohort -> gene ID -> count of unique hosts
num_host_gene_IDs_by_cohort = {cohort: defaultdict(set) for cohort in custom_cohort_tests}

# To get information about which name each gene ID corresponds to
gene_id_name_map = {}

# cohort -> gene -> variant type -> count of SNP changes
genes_by_cohort = {cohort: {} for cohort in custom_cohort_tests}
genes_by_cohort_present_null = {cohort: defaultdict(int) for cohort in custom_cohorts_ordered}
genes_by_cohort_between_null = {cohort: defaultdict(int) for cohort in custom_cohorts_ordered}
genes_by_cohort_pangenome_null = {cohort: defaultdict(int) for cohort in custom_cohorts_ordered}
# cohort -> gene -> count of unique hosts
num_host_genes_by_cohort = {cohort: defaultdict(set) for cohort in custom_cohort_tests}

for species in snp_changes_consecutive:

    print("Working on %s...." % species)
    genome_ids = midas_db_utils.get_ref_genome_ids(species)
    # load the gene descriptions for all genomes coresponding to this speceis:
    gene_descriptions=parse_patric.load_patric_gene_descriptions(genome_ids)

    for sample_i, sample_j in snp_changes[species]:

        val = snp_changes[species][(sample_i, sample_j)]
        subject_tuple = (sample_subject_map[sample_i], sample_subject_map[sample_j])
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j,
species)

        if type(val) == type([]):
            for custom_cohort in custom_cohorts:

                # Store actual changed genes
                for gene_id, contig, position, variant_type, A1, D1, A2, D2 in val:

                    if gene_id not in gene_IDs_by_cohort[custom_cohort]:
                        gene_IDs_by_cohort[custom_cohort][gene_id] = {vartype: 0 for vartype in
variant_types}
                    gene_IDs_by_cohort[custom_cohort][gene_id][variant_type] += 1
                    num_host_gene_IDs_by_cohort[custom_cohort][gene_id].add(subject_tuple)

                    try:
                        desc = gene_descriptions[gene_id]
                        gene_id_name_map[gene_id] = desc
                        if desc not in genes_by_cohort[custom_cohort]:
                            genes_by_cohort[custom_cohort][desc] = {vartype: 0 for vartype in
variant_types}
                        genes_by_cohort[custom_cohort][desc][variant_type] += 1
                        num_host_genes_by_cohort[custom_cohort][desc].add(subject_tuple)
                    except:
                        print("Weird")
                        continue

                # Store present gene null
                for gene_id in snp_change_present_gene_null[species][(sample_i, sample_j)]:

                    count = snp_change_present_gene_null[species][(sample_i, sample_j)][gene_id]

                    gene_IDs_by_cohort_present_null[custom_cohort][gene_id] += count
```

```
                    try:
                        desc = gene_descriptions[gene_id]
                        genes_by_cohort_present_null[custom_cohort][desc] += count
                    except:
                        print("Weird - present")
                        continue

                # Store between host null
                for gene_id in snp_change_between_host_null[species][(sample_i, sample_j)]:

                    count = snp_change_between_host_null[species][(sample_i, sample_j)][gene_id]

                    gene_IDs_by_cohort_between_null[custom_cohort][gene_id] += count

                    try:
                        desc = gene_descriptions[gene_id]
                        genes_by_cohort_between_null[custom_cohort][desc] += count
                    except:
                        print("Weird - between")
                        continue

                # Store pangenome null
                for gene_id in snp_change_pangenome_null[species][(sample_i, sample_j)]:

                    count = snp_change_pangenome_null[species][(sample_i, sample_j)][gene_id]

                    gene_IDs_by_cohort_pangenome_null[custom_cohort][gene_id] += count

                    try:
                        desc = gene_descriptions[gene_id]
                        genes_by_cohort_pangenome_null[custom_cohort][desc] += count
                    except:
                        print("Weird - pangenome")
                        continue


# In[26]:


same_subject_idxs = su.calculate_mi_ordered_same_subject_pairs(sample_order_map, all_samples,
                                                               within_host_type='consecutive',
one_per_mi_pair=False)


# In[27]:


len(same_subject_idxs[0])


# In[31]:


# Report context to get a sense of how big the parallel mutation counts are

custom_cohorts_ordered = ['Mother-Infant(all)', 'Mother-Infant(wk1)', 'Day 0-Week 1', 'Week 1-
Month 1', 'Month 1-Year 1', 'Infant-Infant', 'Adult-Adult']

num_mod_snp_changes = defaultdict(int)
num_modifications = defaultdict(int)
num_qp_pairs = defaultdict(int)
```

```
qp_pair_unique_hosts = defaultdict(set)

for species in snp_changes_consecutive:
    for sample_i, sample_j in snp_changes[species]:
        val = snp_changes[species][(sample_i, sample_j)]
        subject_tuple = (sample_subject_map[sample_i], sample_subject_map[sample_j])
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j)

        for custom_cohort in custom_cohorts:
            num_qp_pairs[custom_cohort] += 1
            qp_pair_unique_hosts[custom_cohort].add(subject_tuple)

        if type(val) == type([]):
            for custom_cohort in custom_cohorts:
                num_snp_changes = len(val)
                num_mod_snp_changes[custom_cohort] += num_snp_changes
                if num_snp_changes > 0:
                    num_modifications[custom_cohort] += 1

all_samples = hmp_samples + mother_samples + infant_samples
# same_subject_idxs = su.calculate_mi_ordered_same_subject_pairs(sample_order_map, all_samples,
#                                                                within_host_type='consecutive',
one_per_mi_pair=False)

num_cons_pairs = defaultdict(int)
cons_pair_unique_hosts = defaultdict(set)

for i, j in zip(same_subject_idxs[0], same_subject_idxs[1]):
    sample_i = all_samples[i]
    sample_j = all_samples[j]
    subject_tuple = (sample_subject_map[sample_i], sample_subject_map[sample_j])
    custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j)

    for custom_cohort in custom_cohorts:
        num_cons_pairs[custom_cohort] += 1
        cons_pair_unique_hosts[custom_cohort].add(subject_tuple)

print("Total number of consecutive temporal sample pairs across mothers, infants and
adults: %i" % len(same_subject_idxs[0]))

for custom_cohort in custom_cohorts_ordered:
    print('=====================================================================')
    print(custom_cohort)
    print('=====================================================================')
    print("Number of consecutive temporal sample pairs (regardless of QP): %i" %
num_cons_pairs[custom_cohort])
    print("Number of unique hosts of consecutive temporal sample pairs: %i" %
len(cons_pair_unique_hosts[custom_cohort]))
    print("Number of consecutive QP sample-species pairs: %i" % num_qp_pairs[custom_cohort])
    print("Number of unique hosts of consecutive QP sample-species pairs: %i" %
len(qp_pair_unique_hosts[custom_cohort]))
    print("Number of QP sample-species pairs which are modifications: %i" %
num_modifications[custom_cohort])
    print("Number of modification SNP changes considered: %i" %
num_mod_snp_changes[custom_cohort])
    print("")


# In[19]:
```

```
# Pickle everything

gene_parallelism_data = [gene_IDs_by_cohort, gene_IDs_by_cohort_present_null,
gene_IDs_by_cohort_between_null,
                         gene_IDs_by_cohort_pangenome_null, num_host_gene_IDs_by_cohort,
genes_by_cohort,
                         genes_by_cohort_present_null, genes_by_cohort_between_null,
                         genes_by_cohort_pangenome_null, num_host_genes_by_cohort,
gene_id_name_map]

pickle.dump(gene_parallelism_data, open('%s/gene_parallelism_data_v2.pkl' % pdir, 'wb'))


# In[10]:


gene_parallelism_data = pickle.load(open('%s/gene_parallelism_data.pkl' % pdir, 'rb'))

gene_IDs_by_cohort, gene_IDs_by_cohort_present_null, gene_IDs_by_cohort_between_null,
gene_IDs_by_cohort_pangenome_null, num_host_gene_IDs_by_cohort, genes_by_cohort,
genes_by_cohort_present_null, genes_by_cohort_between_null, genes_by_cohort_pangenome_null,
num_host_genes_by_cohort, gene_id_name_map = gene_parallelism_data


# In[19]:


# Statistics for paper
# Outer membrane TonB-dependent transporter, utilization system for glycans and polysaccharides
(PUL), SusC family
# How many times mutated, and in how many hosts, across infant-infant/mother-infant/adult-adult?

gene = 'Outer membrane TonB-dependent transporter, utilization system for glycans and
polysaccharides (PUL), SusC family'
all_cohorts = ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']

count = 0
hosts = set()

for cohort in all_cohorts:
    count += sum(genes_by_cohort[cohort][gene].values())
    for host in num_host_genes_by_cohort[cohort][gene]:
        hosts.add(host)

print("%i SNV changes" % count)
print("%i unique hosts" % len(hosts))


# In[27]:


# Store SNP change annotation info in tabular form

custom_cohorts_ordered = ['Mother-Infant(all)', 'Mother-Infant(wk1)', 'Mother-Infant(earliest)',
'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Infant-Infant', 'Adult-Adult']

f = open('%s/snp_change_gene_parallelism.tsv' % (config.analysis_directory), 'w')
f.write('\t'.join(['tp_type', 'num_unique_hosts', 'num_change', 'num_1d', 'num_2d', 'num_3d',
'num_4d', 'gene_id']) + '\n')

for cat in custom_cohorts_ordered:
```

```
    for gene, count_dict in sorted(genes_by_cohort[cat].items(), key=lambda x: x[1],
reverse=True):

        count_1d = count_dict['1D']
        count_2d = count_dict['2D']
        count_3d = count_dict['3D']
        count_4d = count_dict['4D']
        num_unique_hosts = len(num_host_genes_by_cohort[cat][gene])
        total_count = count_1d + count_2d + count_3d + count_4d

        f.write('\t'.join([str(x) for x in [cat, num_unique_hosts,
                                            total_count, count_1d, count_2d, count_3d, count_4d,
                                            gene]]) + '\n')
f.close()

f = open('%s/snp_change_gene_ID_parallelism.tsv' % (config.analysis_directory), 'w')
f.write('\t'.join(['tp_type', 'num_unique_hosts', 'num_change', 'num_1d', 'num_2d', 'num_3d',
'num_4d', 'gene_id']) + '\n')

for cat in custom_cohorts_ordered:
    for gene, count_dict in sorted(gene_IDs_by_cohort[cat].items(), key=lambda x: x[1],
reverse=True):

        count_1d = count_dict['1D']
        count_2d = count_dict['2D']
        count_3d = count_dict['3D']
        count_4d = count_dict['4D']
        num_unique_hosts = len(num_host_gene_IDs_by_cohort[cat][gene])
        total_count = count_1d + count_2d + count_3d + count_4d

        f.write('\t'.join([str(x) for x in [cat, num_unique_hosts,
                                            total_count, count_1d, count_2d, count_3d, count_4d,
                                            gene]]) + '\n')
f.close()


# In[28]:


# Store SNP change annotation info in tabular form, extended

custom_cohorts_ordered = ['Mother-Infant(all)', 'Mother-Infant(wk1)', 'Mother-Infant(earliest)',
'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Infant-Infant', 'Adult-Adult']

f = open('%s/snp_change_gene_parallelism_extended.tsv' % (config.analysis_directory), 'w')
f.write('\t'.join(['tp_type', 'num_unique_hosts', 'num_change', 'pres_null', 'btwn_null',
'pang_null', 'num_1d', 'num_2d', 'num_3d', 'num_4d', 'gene_id', 'gene_name']) + '\n')

for cat in custom_cohorts_ordered:
    for gene_id, count_dict in sorted(gene_IDs_by_cohort[cat].items(), key=lambda x:
sum(x[1].values()), reverse=True):

        count_1d = count_dict['1D']
        count_2d = count_dict['2D']
        count_3d = count_dict['3D']
        count_4d = count_dict['4D']
        num_unique_hosts = len(num_host_gene_IDs_by_cohort[cat][gene_id])
        pres_null_count = gene_IDs_by_cohort_present_null[cat][gene_id]
        btwn_null_count = gene_IDs_by_cohort_between_null[cat][gene_id]
        pang_null_count = gene_IDs_by_cohort_pangenome_null[cat][gene_id]
        total_count = count_1d + count_2d + count_3d + count_4d
```

```
        gene_name = gene_id_name_map[gene_id]

        f.write('\t'.join([str(x) for x in [cat, num_unique_hosts,
                                            total_count, pres_null_count, btwn_null_count,
pang_null_count,
                                            count_1d, count_2d, count_3d, count_4d,
                                            gene_id, gene_name]]) + '\n')
f.close()


# In[38]:


# Store SNP change annotation in tabular form, final

custom_cohorts_ordered = ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']

f = open('%s/snp_change_gene_parallelism_v3.tsv' % (config.analysis_directory), 'w')
f.write('\t'.join(['MI_num_hosts','II_num_hosts','AA_num_hosts','total_unique_hosts',
                   'total_snp_changes','total_pres_null','total_pang_null',
                   'gene_name','gene_ids']) + '\n')

gene_name_total_hosts_dict = defaultdict(list)
gene_name_gene_ids_dict = defaultdict(set)

for cat in custom_cohorts_ordered:
    for gene_id, count_dict in sorted(gene_IDs_by_cohort[cat].items(), key=lambda x:
sum(x[1].values()), reverse=True):
        hosts = num_host_gene_IDs_by_cohort[cat][gene_id]
        gene_name = gene_id_name_map[gene_id]
        gene_name_total_hosts_dict[gene_name] += hosts
        gene_name_gene_ids_dict[gene_name].add(gene_id)

gene_name_total_hosts_count_dict = {}
for gene_name in gene_name_total_hosts_dict:
    gene_name_total_hosts_dict[gene_name] = len(set(gene_name_total_hosts_dict[gene_name]))

for gene_name, total_hosts in sorted(gene_name_total_hosts_dict.items(), key=lambda x: x[1],
reverse=True):

    gene_ids = gene_name_gene_ids_dict[gene_name]

    mi_hosts = []; ii_hosts = []; aa_hosts = []

    for gene_id in gene_ids:
        mi_hosts += num_host_gene_IDs_by_cohort['Mother-Infant(earliest)'][gene_id]
        ii_hosts += num_host_gene_IDs_by_cohort['Infant-Infant'][gene_id]
        aa_hosts += num_host_gene_IDs_by_cohort['Adult-Adult'][gene_id]

        pres_null_count = gene_IDs_by_cohort_present_null[cat][gene_id]
        pang_null_count = gene_IDs_by_cohort_pangenome_null[cat][gene_id]

    num_snp_changes = 0; pres_null_count = 0; pang_null_count = 0

    for cat in custom_cohorts_ordered:
        for gene_id in gene_ids:
            if gene_id in gene_IDs_by_cohort[cat]:
                num_snp_changes += sum(gene_IDs_by_cohort[cat][gene_id].values())
            pres_null_count += gene_IDs_by_cohort_present_null[cat][gene_id]
            pang_null_count += gene_IDs_by_cohort_pangenome_null[cat][gene_id]
```

```
    f.write('\t'.join([str(x) for x in [len(set(mi_hosts)), len(set(ii_hosts)),
len(set(aa_hosts)),
                                        total_hosts, num_snp_changes, pres_null_count,
pang_null_count,
                                        gene_name, ','.join(list(gene_ids))]]) + '\n')
f.close()


# In[21]:


# Wait, are my nulls ok?

custom_cohorts_ordered = ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']

num_snp_changes = 0; pres_null_count = 0; pang_null_count = 0

for cat in custom_cohorts_ordered:
    combined_genes = set(list(gene_IDs_by_cohort_pangenome_null[cat].keys()) +
list(gene_IDs_by_cohort_present_null[cat].keys()) +
list(gene_IDs_by_cohort[cat].keys()))
    for gene_id in combined_genes:
        if gene_id in gene_IDs_by_cohort[cat]:
            num_snp_changes += sum(gene_IDs_by_cohort[cat][gene_id].values())
        pres_null_count += gene_IDs_by_cohort_present_null[cat][gene_id]
        pang_null_count += gene_IDs_by_cohort_pangenome_null[cat][gene_id]

print(pres_null_count)
print(pang_null_count)
print(num_snp_changes)


# In[23]:


for cat in custom_cohorts_ordered:
    print(cat)
    print(len(gene_IDs_by_cohort_pangenome_null[cat].keys()))
    print(len(gene_IDs_by_cohort_present_null[cat].keys()))
    print(len(gene_IDs_by_cohort[cat].keys()))


# In[58]:


# Parallelism (for SNP changes), assuming gene_id_name_map already done
# Also make a new host-based null

custom_cohorts_ordered = list(custom_cohort_tests.keys())
variant_types = ['4D', '1D', '2D', '3D']

# cohort -> gene ID -> variant type -> count of SNP changes
gene_IDs_by_cohort = {cohort: {} for cohort in custom_cohort_tests}
gene_IDs_by_cohort_present_null = {cohort: defaultdict(int) for cohort in
custom_cohorts_ordered}
gene_IDs_by_cohort_between_null = {cohort: defaultdict(int) for cohort in
custom_cohorts_ordered}
gene_IDs_by_cohort_pangenome_null = {cohort: defaultdict(int) for cohort in
custom_cohorts_ordered}
# cohort -> gene ID -> count of unique hosts
num_host_gene_IDs_by_cohort = {cohort: defaultdict(set) for cohort in custom_cohort_tests}
```

```
# cohort -> gene -> variant type -> count of SNP changes
genes_by_cohort = {cohort: {} for cohort in custom_cohort_tests}

# cohort -> gene -> count of SNP changes under null
genes_by_cohort_present_null = {cohort: defaultdict(int) for cohort in custom_cohorts_ordered}
genes_by_cohort_between_null = {cohort: defaultdict(int) for cohort in custom_cohorts_ordered}
genes_by_cohort_pangenome_null = {cohort: defaultdict(int) for cohort in custom_cohorts_ordered}

# cohort -> gene -> count of unique hosts
num_host_genes_by_cohort = {cohort: defaultdict(set) for cohort in custom_cohort_tests}

# cohort -> gene -> count of unique hosts under null
genes_hosts_by_cohort_present_null = {cohort: defaultdict(int) for cohort in
custom_cohorts_ordered}
genes_hosts_by_cohort_between_null = {cohort: defaultdict(int) for cohort in
custom_cohorts_ordered}
genes_hosts_by_cohort_pangenome_null = {cohort: defaultdict(int) for cohort in
custom_cohorts_ordered}

for species in snp_changes_consecutive:
    for sample_i, sample_j in snp_changes[species]:

        val = snp_changes[species][(sample_i, sample_j)]
        subject_tuple = (sample_subject_map[sample_i], sample_subject_map[sample_j])
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j,
species)

        if type(val) == type([]):
            for custom_cohort in custom_cohorts:

                # Store actual changed genes
                for gene_id, contig, position, variant_type, A1, D1, A2, D2 in val:

                    if gene_id not in gene_IDs_by_cohort[custom_cohort]:
                        gene_IDs_by_cohort[custom_cohort][gene_id] = {vartype: 0 for vartype in
variant_types}
                    gene_IDs_by_cohort[custom_cohort][gene_id][variant_type] += 1
                    num_host_gene_IDs_by_cohort[custom_cohort][gene_id].add(subject_tuple)

                    try:
                        desc = gene_id_name_map[gene_id]
                        gene_id_name_map[gene_id] = desc
                        if desc not in genes_by_cohort[custom_cohort]:
                            genes_by_cohort[custom_cohort][desc] = {vartype: 0 for vartype in
variant_types}
                        genes_by_cohort[custom_cohort][desc][variant_type] += 1
                        num_host_genes_by_cohort[custom_cohort][desc].add(subject_tuple)
                    except:
                        print("Weird")
                        continue

                # Store present gene null
                for gene_id in snp_change_present_gene_null[species][(sample_i, sample_j)]:

                    count = snp_change_present_gene_null[species][(sample_i, sample_j)][gene_id]

                    if count != 0:
                        genes_hosts_by_cohort_between_null[]
                    gene_IDs_by_cohort_present_null[custom_cohort][gene_id] += count
```

```
                    try:
                        desc = gene_id_name_map[gene_id]
                        genes_by_cohort_present_null[custom_cohort][desc] += count
                    except:
                        print("Weird - present")
                        continue

                # Store between host null
                for gene_id in snp_change_between_host_null[species][(sample_i, sample_j)]:

                    count = snp_change_between_host_null[species][(sample_i, sample_j)][gene_id]

                    gene_IDs_by_cohort_between_null[custom_cohort][gene_id] += count

                    try:
                        desc = gene_id_name_map[gene_id]
                        genes_by_cohort_between_null[custom_cohort][desc] += count
                    except:
                        print("Weird - between")
                        continue

                # Store pangenome null
                for gene_id in snp_change_pangenome_null[species][(sample_i, sample_j)]:

                    count = snp_change_pangenome_null[species][(sample_i, sample_j)][gene_id]

                    gene_IDs_by_cohort_pangenome_null[custom_cohort][gene_id] += count

                    try:
                        desc = gene_id_name_map[gene_id]
                        genes_by_cohort_pangenome_null[custom_cohort][desc] += count
                    except:
                        print("Weird - pangenome")
                        continue


# In[21]:


gene_IDs_by_cohort.keys()


# # Gene change parallelism

# In[57]:


# Parallelism (for gene changes)

variant_types = ['4D', '1D', '2D', '3D']

# cohort -> gene ID -> variant type -> count of gene gains/losses
gain_gene_IDs_by_cohort = {cohort: defaultdict(int) for cohort in custom_cohort_tests}
loss_gene_IDs_by_cohort = {cohort: defaultdict(int) for cohort in custom_cohort_tests}
# cohort -> gene ID -> count of unique hosts
gain_num_host_gene_IDs_by_cohort = {cohort: defaultdict(set) for cohort in custom_cohort_tests}
loss_num_host_gene_IDs_by_cohort = {cohort: defaultdict(set) for cohort in custom_cohort_tests}

# To get information about which name each gene ID corresponds to
gene_id_name_map = {}
```

```
# To get information about which species each gene ID corresponds to
gene_id_species_map = {}

# cohort -> gene -> variant type -> count of gene gains/losses
gain_genes_by_cohort = {cohort: defaultdict(int) for cohort in custom_cohort_tests}
loss_genes_by_cohort = {cohort: defaultdict(int) for cohort in custom_cohort_tests}
# cohort -> gene -> count of unique hosts
gain_num_host_genes_by_cohort = {cohort: defaultdict(set) for cohort in custom_cohort_tests}
loss_num_host_genes_by_cohort = {cohort: defaultdict(set) for cohort in custom_cohort_tests}

for species in gene_changes_consecutive:

    print("Working on %s...." % species)
    genome_ids = midas_db_utils.get_ref_genome_ids(species)

    # load the gene descriptions for all genomes coresponding to this speceis:
    gene_descriptions=parse_patric.load_patric_gene_descriptions(genome_ids)

    for sample_i, sample_j in gene_changes_consecutive[species]:

        gains, losses = gene_changes[species][(sample_i, sample_j)]
        subject_tuple = (sample_subject_map[sample_i], sample_subject_map[sample_j])
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j,
species)

        # Also get dN/dS info
        nonsyn_diffs, nonsyn_opps, syn_diffs, syn_opps = dnds_info[species][(sample_i,
sample_j)]

        if type(gains) == type([]): # Modification event

            for custom_cohort in custom_cohorts:
                # Gains
                for gene_id, D1, Dm1, D2, Dm2 in gains:

                    gain_gene_IDs_by_cohort[custom_cohort][gene_id] += 1
                    gain_num_host_gene_IDs_by_cohort[custom_cohort][gene_id].add(subject_tuple)

                    desc = gene_descriptions[gene_id]
                    gene_id_name_map[gene_id] = desc
                    gene_id_species_map[gene_id] = species
                    gain_genes_by_cohort[custom_cohort][desc] += 1
                    gain_num_host_genes_by_cohort[custom_cohort][desc].add(subject_tuple)

                # Losses
                for gene_id, D1, Dm1, D2, Dm2 in losses:

                    loss_gene_IDs_by_cohort[custom_cohort][gene_id] += 1
                    loss_num_host_gene_IDs_by_cohort[custom_cohort][gene_id].add(subject_tuple)

                    desc = gene_descriptions[gene_id]
                    gene_id_name_map[gene_id] = desc
                    gene_id_species_map[gene_id] = species
                    loss_genes_by_cohort[custom_cohort][desc] += 1
                    loss_num_host_genes_by_cohort[custom_cohort][desc].add(subject_tuple)


# In[59]:


# Pickle everything
```

```
gene_change_gene_parallelism_data = [gain_gene_IDs_by_cohort, loss_gene_IDs_by_cohort,
gain_num_host_gene_IDs_by_cohort,
                                     loss_num_host_gene_IDs_by_cohort, gene_id_name_map,
gain_genes_by_cohort,
                                     loss_genes_by_cohort, gain_num_host_genes_by_cohort,
loss_num_host_genes_by_cohort,
                                    gene_id_species_map, gene_id_species_map]

pickle.dump(gene_change_gene_parallelism_data, open('%s/gene_change_gene_parallelism_data.pkl' %
pdir, 'wb'))


# In[60]:


gainloss_genes_by_cohort = {cohort: defaultdict(int) for cohort in custom_cohort_tests}

for cohort in custom_cohort_tests:
    combined_genes = set(gain_genes_by_cohort[cohort].keys() +
loss_genes_by_cohort[cohort].keys())
    for gene in combined_genes:
        if gene not in gain_genes_by_cohort[cohort]:
            gain_genes_by_cohort[cohort][gene] = 0
        if gene not in loss_genes_by_cohort[cohort]:
            loss_genes_by_cohort[cohort][gene] = 0

        combined_gene_change_count = gain_genes_by_cohort[cohort][gene] +
loss_genes_by_cohort[cohort][gene]

        gainloss_genes_by_cohort[cohort][gene] = combined_gene_change_count


# In[61]:


gainloss_gene_IDs_by_cohort = {cohort: defaultdict(int) for cohort in custom_cohort_tests}

for cohort in custom_cohort_tests:
    combined_genes = set(gain_gene_IDs_by_cohort[cohort].keys() +
loss_gene_IDs_by_cohort[cohort].keys())
    for gene in combined_genes:
        if gene not in gain_gene_IDs_by_cohort[cohort]:
            gain_gene_IDs_by_cohort[cohort][gene] = 0
        if gene not in loss_gene_IDs_by_cohort[cohort]:
            loss_gene_IDs_by_cohort[cohort][gene] = 0

        combined_gene_change_count = gain_gene_IDs_by_cohort[cohort][gene] +
loss_gene_IDs_by_cohort[cohort][gene]

        gainloss_gene_IDs_by_cohort[cohort][gene] = combined_gene_change_count


# In[39]:


# Sanity check: are we getting the same number of gene changes as in the prevalence plot?

total_gene_changes = 0
total_gains = 0
total_loss = 0
```

```
for custom_cohort in custom_cohort_tests:
    a = 0
    b = 0
    for gene in gainloss_genes_by_cohort[custom_cohort]:
        a += gainloss_genes_by_cohort[custom_cohort][gene]
    for gene_id in gainloss_gene_IDs_by_cohort[custom_cohort]:
        b += gainloss_gene_IDs_by_cohort[custom_cohort][gene_id]

    gains1 = 0
    gains2 = 0
    for gene in gain_genes_by_cohort[custom_cohort]:
        gains1 += gain_genes_by_cohort[custom_cohort][gene]
    for gene_id in gain_gene_IDs_by_cohort[custom_cohort]:
        gains2 += gain_gene_IDs_by_cohort[custom_cohort][gene_id]

    loss1 = 0
    loss2 = 0
    for gene in loss_genes_by_cohort[custom_cohort]:
        loss1 += loss_genes_by_cohort[custom_cohort][gene]
    for gene_id in loss_gene_IDs_by_cohort[custom_cohort]:
        loss2 += loss_gene_IDs_by_cohort[custom_cohort][gene_id]

    print(custom_cohort)
    print("Number of total gene changes: %i %i" % (a, b))
    print("Number of gains: %i %i" % (gains1, gains2))
    print("Number of losses: %i %i" % (loss1, loss2))
    print('')

    total_gene_changes += a
    total_gains += gains1
    total_loss += loss1

print("Total (duplicates included):")
print(total_gene_changes)
print(total_gains)
print(total_loss)
# 2500 gene changes, 1228 gains, 1272 losses


# In[40]:


# Store gene change annotation info in tabular form

custom_cohorts_ordered = ['Mother-Infant(wk1)', 'Mother-Infant(all)', 'Mother-Infant(earliest)',
'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Infant-Infant', 'Adult-Adult']

f = open('%s/gene_change_gene_parallelism.tsv' % (config.analysis_directory), 'w')
f.write('\t'.join(['tp_type', 'num_change', 'num_gains', 'num_unique_hosts_gains', 'num_loss',
'num_unique_hosts_loss', 'gene']) + '\n')

for cat in custom_cohorts_ordered:
    for gene, total_count in sorted(gainloss_genes_by_cohort[cat].items(), key=lambda x: x[1],
reverse=True):

        num_gains = gain_genes_by_cohort[cat][gene]
        num_unique_hosts_gains = len(gain_num_host_genes_by_cohort[cat][gene])
        num_loss = loss_genes_by_cohort[cat][gene]
        num_unique_hosts_loss = len(loss_num_host_genes_by_cohort[cat][gene])
```

```
        f.write('\t'.join([str(x) for x in [cat, total_count, num_gains, num_unique_hosts_gains,
num_loss, num_unique_hosts_loss, gene]]) + '\n')

f.close()

f = open('%s/gene_change_gene_ID_parallelism.tsv' % (config.analysis_directory), 'w')
f.write('\t'.join(['tp_type', 'num_change', 'num_gains', 'num_unique_hosts_gains', 'num_loss',
'num_unique_hosts_loss', 'gene']) + '\n')

for cat in custom_cohorts_ordered:
    for gene, total_count in sorted(gainloss_gene_IDs_by_cohort[cat].items(), key=lambda x:
x[1], reverse=True):

        num_gains = gain_gene_IDs_by_cohort[cat][gene]
        num_unique_hosts_gains = len(gain_num_host_gene_IDs_by_cohort[cat][gene])
        num_loss = loss_gene_IDs_by_cohort[cat][gene]
        num_unique_hosts_loss = len(loss_num_host_gene_IDs_by_cohort[cat][gene])

        f.write('\t'.join([str(x) for x in [cat, total_count, num_gains, num_unique_hosts_gains,
num_loss, num_unique_hosts_loss, gene]]) + '\n')

f.close()


# In[62]:


# Store gene change annotation info in tabular form, extended

custom_cohorts_ordered = ['Mother-Infant(wk1)', 'Mother-Infant(all)', 'Mother-Infant(earliest)',
'Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Infant-Infant', 'Adult-Adult']

f = open('%s/gene_change_gene_ID_parallelism_extended.tsv' % (config.analysis_directory), 'w')
f.write('\t'.join(['tp_type', 'num_change', 'num_gains', 'num_unique_hosts_gains', 'num_loss',
'num_unique_hosts_loss', 'geneID', 'species', 'gene_name']) + '\n')

for cat in custom_cohorts_ordered:
    for gene_id, total_count in sorted(gainloss_gene_IDs_by_cohort[cat].items(), key=lambda x:
x[1], reverse=True):

        gene_name = gene_id_name_map[gene_id]
        species = gene_id_species_map[gene_id]
        num_gains = gain_gene_IDs_by_cohort[cat][gene_id]
        num_unique_hosts_gains = len(gain_num_host_gene_IDs_by_cohort[cat][gene_id])
        num_loss = loss_gene_IDs_by_cohort[cat][gene_id]
        num_unique_hosts_loss = len(loss_num_host_gene_IDs_by_cohort[cat][gene_id])

        f.write('\t'.join([str(x) for x in [cat, total_count, num_gains, num_unique_hosts_gains,
num_loss, num_unique_hosts_loss, gene_id, species, gene_name]]) + '\n')

f.close()


# # Parallel gene analysis

# In[22]:


# Firstly get list of parallel changing genes
# Restrict to specific categories of interest first
```

```
# Infant-infant:
ii_parallel_genes = ['Periplasmic ligand-binding sensor domain COG3292 / BaeS-type histidine
kinase / OmpR-type DNA-binding response regulator',
                     'Outer membrane TonB-dependent transporter, utilization system for glycans
and polysaccharides (PUL), SusC family',
                     'Cell surface glycan-binding lipoprotein, utilization system for glycans and
polysaccharides (PUL), SusD family',
                     'Two-component system sensor histidine kinase', 'beta-galactosidase (EC
3.2.1.23)',
                     'Transcriptional regulator, AraC family', 'Sporulation transcription
regulator WhiA']

# Mother-infant (old which really includes infant-infant)
mi_parallel_genes = ['Outer membrane TonB-dependent transporter, utilization system for glycans
and polysaccharides (PUL), SusC family',
                      'Periplasmic ligand-binding sensor domain COG3292 / BaeS-type histidine
kinase / OmpR-type DNA-binding response regulator',
                      'Transcriptional regulator, AraC family',
                      'beta-glucosidase (EC 3.2.1.21)',
                      'Two-component system sensor histidine kinase','RND efflux system, membrane
fusion protein',
                      'Cell surface glycan-binding lipoprotein, utilization system for glycans
and polysaccharides (PUL), SusD family']


# In[23]:


# Get list of paralel changing genes: v2
# Any gene NAME that changes in at least 3/4 hosts across infant-infant,
# mother-infant(earliest) or adult-adult categories

host_threshold = 3

gene_hosts_dict = defaultdict(set)

for cat in ['Infant-Infant']: # ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']:
    for gene in num_host_genes_by_cohort[cat]:
        hosts = num_host_genes_by_cohort[cat][gene]
        for host in hosts:
            gene_hosts_dict[gene].add(host)

gene_num_hosts_dict = {gene: len(gene_hosts_dict[gene]) for gene in gene_hosts_dict}

sorted_gene_num_hosts = sorted(gene_num_hosts_dict.items(), key=lambda x: x[1], reverse=True)

ii_parallel_genes = []
for gene, num_hosts in sorted_gene_num_hosts:
    if num_hosts >= host_threshold:
        ii_parallel_genes.append(gene)

ii_parallel_genes.remove('hypothetical protein')
any_parallel_genes = ii_parallel_genes
print("There are %i parallel genes" % len(ii_parallel_genes))


# In[24]:


# Also, plot something like Zhao and Lieberman 4C:
# Histogram of present nulls for parallel genes
```

```
# (Shuffle multiple times---num_trials=100 ok?), demarcate vlines -- see slide 49

parallel_pres_null_counts = []
parallel_btwn_null_counts = []
parallel_pang_null_counts = []

all_pres_null_counts = []
all_btwn_null_counts = []
all_pang_null_counts = []

# cohort -> gene -> variant type -> count of SNP changes
# Originally, only looked at one infant-infant cohort
# cohort = 'Infant-Infant'
cohorts = ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']

all_genes = []
for cohort in cohorts:
    all_genes += genes_by_cohort[cohort].keys()

all_genes = list(set(all_genes))

for gene in all_genes:

    # Skip hypothetical protein
    if gene == 'hypothetical protein':
        continue

    actual_count = 0
    pres_null_count = 0
    btwn_null_count = 0
    pang_null_count = 0

    for cohort in cohorts:

        # Skip if gene not present in cohort
        if gene not in genes_by_cohort[cohort]:
            continue

        actual_count += sum(genes_by_cohort[cohort][gene].values())
        pres_null_count += (genes_by_cohort_present_null[cohort][gene])
        btwn_null_count += (genes_by_cohort_between_null[cohort][gene])
        pang_null_count += (genes_by_cohort_pangenome_null[cohort][gene])

    # Store info
    all_pres_null_counts.append(pres_null_count)
    all_btwn_null_counts.append(btwn_null_count)
    all_pang_null_counts.append(pang_null_count)

    # Store info only if gene is special
    if gene in any_parallel_genes:
        parallel_pres_null_counts.append(pres_null_count)
        parallel_btwn_null_counts.append(btwn_null_count)
        parallel_pang_null_counts.append(pang_null_count)


# In[ ]:


fig, ax = plt.subplots()

cohorts = ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']
```

```
pres_hist = ax.hist(all_pres_null_counts, bins=50, alpha=0.6, label='Present null')
between_hist = ax.hist(all_btwn_null_counts, bins=50, alpha=0.6, label='Between null')
pang_hist = ax.hist(all_pang_null_counts, bins=50, alpha=0.6, label='Pangenome null')

for gene in any_parallel_genes:
    count = 0
    for cohort in cohorts:
        if gene in genes_by_cohort[cohort]:
            count += sum(genes_by_cohort[cohort][gene].values())
    ax.axvline(x=count, color='red', linewidth=0.6)

ax.set_xlabel("Number of gene changes")
ax.set_ylabel("Frequency")
ax.set_xscale('log')
ax.legend()

plt.show()


# In[26]:


# Survival curve version
fig, ax = plt.subplots()

cohorts = ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']

xs, ns = calculate_unnormalized_survival_from_vector(all_pres_null_counts)
ax.step(xs,ns/float(ns[0]),'-',linewidth=1.4, label='Present null', where='pre',zorder=4)
xs, ns = calculate_unnormalized_survival_from_vector(all_btwn_null_counts)
ax.step(xs,ns/float(ns[0]),'-',linewidth=1.4, label='Between null', where='pre',zorder=4)
xs, ns = calculate_unnormalized_survival_from_vector(all_pang_null_counts)
ax.step(xs,ns/float(ns[0]),'-',linewidth=1.4, label='Present null', where='pre',zorder=4)

ax.set_ylim((0, 1))

for gene in any_parallel_genes:
    count = 0
    for cohort in cohorts:
        if gene in genes_by_cohort[cohort]:
            count += sum(genes_by_cohort[cohort][gene].values())
    ax.axvline(x=count, color='red', linewidth=0.6)

# Redundant for label
ax.axvline(x=count, color='red', linewidth=0.6, label="Special gene")

ax.set_xlabel("Number of SNP changes", fontsize=12)
ax.set_ylabel("Fraction of genes >= n", fontsize=12)
ax.set_xscale('log')
ax.legend(frameon=False)
# ax.legend(loc='center right', bbox_to_anchor=(1.397, 0.5), frameon=False, fontsize=11)

fig.savefig('%s/mutations_per_gene_nulls_survival_any.pdf' % plot_dir, bbox_inches='tight')
# fig.savefig('%s/mutations_per_gene_nulls_survival_any.png' % plot_dir, bbox_inches='tight',
dpi=500)
plt.show()


# In[17]:
```

```
# Survival curve version (flattened)
fig, ax = plt.subplots(figsize=(8, 3))

xs, ns = calculate_unnormalized_survival_from_vector(all_pres_null_counts)
ax.step(xs,ns/float(ns[0]),'-',linewidth=1.4, label='Present null', where='pre',zorder=4)
xs, ns = calculate_unnormalized_survival_from_vector(all_btwn_null_counts)
ax.step(xs,ns/float(ns[0]),'-',linewidth=1.4, label='Between null', where='pre',zorder=4)
xs, ns = calculate_unnormalized_survival_from_vector(all_pang_null_counts)
ax.step(xs,ns/float(ns[0]),'-',linewidth=1.4, label='Present null', where='pre',zorder=4)

ax.set_ylim((0, 1))

for gene in any_parallel_genes:
    count = 0
    for cohort in cohorts:
        if gene in genes_by_cohort[cohort]:
            count += sum(genes_by_cohort[cohort][gene].values())
    ax.axvline(x=count, color='red', linewidth=0.6)

# Redundant for label
ax.axvline(x=count, color='red', linewidth=0.6, label="Special gene")

ax.set_xlabel("Number of SNP changes", fontsize=12)
ax.set_ylabel("Fraction of genes >= n", fontsize=12)
ax.set_xscale('log')
ax.legend(loc='upper center', bbox_to_anchor=(0.4, 0.97), frameon=False)
# ax.legend(loc='center right', bbox_to_anchor=(1.397, 0.5), frameon=False, fontsize=11)

# fig.savefig('%s/mutations_per_gene_nulls_survival_any.pdf' % plot_dir, bbox_inches='tight')
# fig.savefig('%s/mutations_per_gene_nulls_survival_any.png' % plot_dir, bbox_inches='tight',
dpi=500)
plt.show()


# In[18]:


ii_parallel_genes


# In[28]:


# Make table
# Number of SNP changes/unique hosts having SNP changes per parallel gene and per timepoint pair
category
parallel_genes = any_parallel_genes
custom_cohorts_ordered = ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']

parallel_genes_by_cohort = {gene: {tt: 0 for tt in custom_cohorts_ordered} for gene in
parallel_genes}
count_by_parallel_gene = defaultdict(int)
host_parallel_genes_by_cohort = {gene: {tt: set() for tt in custom_cohorts_ordered} for gene in
parallel_genes}

for cohort in custom_cohorts_ordered:
    for gene in genes_by_cohort[cohort]:
        if gene in parallel_genes:
            num_snp_changes = sum(genes_by_cohort[cohort][gene].values())
            num_subject_tuples = len(num_host_genes_by_cohort[cohort][gene])
```

```
            parallel_genes_by_cohort[gene][cohort] = num_snp_changes
            count_by_parallel_gene[gene] += num_snp_changes
            for subject_tuple in num_host_genes_by_cohort[cohort][gene]:
                host_parallel_genes_by_cohort[gene][cohort].add(subject_tuple)


# In[29]:


parallel_genes_ordered = []
for gene, count in sorted(count_by_parallel_gene.items(), key=lambda x: x[1], reverse=True):
    parallel_genes_ordered.append(gene)


# In[85]:


# Store table

f = open('%s/parallel_gene_snp_changes_by_tp_type.tsv' % plot_dir, 'w')
f.write('\t'.join(['gene_name'] + custom_cohorts_ordered + ['Total'] + custom_cohorts_ordered +
['Total(unique)']) + '\n')

for gene in parallel_genes_ordered:
    counts = [parallel_genes_by_cohort[gene][cohort] for cohort in custom_cohorts_ordered]
    host_counts = [len(host_parallel_genes_by_cohort[gene][cohort]) for cohort in
custom_cohorts_ordered]

    all_hosts = set()
    for cohort in custom_cohorts_ordered:
        fort host in host_parallel_genes_by_cohort[gene][cohort]:
            all_hosts.add(host)

    data = [gene] + counts + [sum(counts)] + host_counts + [len(all_hosts)]
    f.write('\t'.join([str(datum) for datum in data]) + '\n')

f.close()


# In[98]:


# Bubble plot
fig, ax = plt.subplots(len(parallel_genes_ordered), 1, figsize=(4,8),
gridspec_kw={'wspace':0.025, 'hspace':0})

custom_bins = ['MI', 'II', 'AA']
cmap = cmx.get_cmap('Blues', 13)
colors = [cmap(x) for x in np.array([x for x in range(0,13)])/13.0]

mi_cohort_labels = []

gene_i = 0
for gene in parallel_genes_ordered:

    ax[gene_i].set_xlim((-0.5, len(custom_bins)-0.5))

    col_i = 0
    for tp_type, color in zip(['Mother-Infant(wk1)', 'Infant-Infant', 'Adult-Adult'],
['#7bb551', '#77acff', '#396651']):
        num_snp_changes = parallel_genes_by_cohort[gene][tp_type]
```

```
        num_hosts = len(host_parallel_genes_by_cohort[gene][tp_type])
        print("Gene %s | %s" % (gene, tp_type))
        if gene not in genes_by_cohort[tp_type]:
            print("Nothing")
        else:
            for vartype in genes_by_cohort[tp_type][gene]:
                print("%s: %i" % (vartype, genes_by_cohort[tp_type][gene][vartype]))
        print("# SNP changes: %i" % num_snp_changes)
        print("# hosts: %i" % num_hosts)
        print('')
        ax[gene_i].plot([col_i], [gene_i], 'o', markersize = num_hosts*3.4,
color=colors[num_hosts])
        col_i += 1

    ax[gene_i].set_yticks([gene_i])

    if len(gene) > 50:
        substrs = gene.split(' ')
        half_idx = (len(substrs)/2)
        half1 = ' '.join(substrs[:half_idx])
        half2 = ' '.join(substrs[half_idx:])
        gene_label = half1 + '\n' + half2
    else:
        gene_label = gene

    ax[gene_i].set_yticklabels([gene_label], fontsize=14)
    ax[gene_i].set_xticks([])
    for x in np.arange(0.5, len(custom_bins), step=1):
        ax[gene_i].axvline(x=x)
    gene_i += 1

ax[0].set_xticks(np.arange(len(custom_bins)))
ax[0].set_xticklabels(custom_bins)
ax[0].xaxis.tick_top()

# Add colorbar

# define the bins and normalize
bounds = np.arange(0, 12)
norm = mplcol.BoundaryNorm(bounds, cmap.N)

# create a second axes for the colorbar
ax2 = fig.add_axes([0.95, 0.1, 0.03, 0.8])
cb = colorbar.ColorbarBase(ax2, cmap=cmap, norm=norm,
    spacing='proportional', ticks=bounds, boundaries=bounds, format='%1i')

plt.tick_params(top=False)

fig.savefig('%s/parallel_genes_bubble_plot.pdf' %
(config.analysis_directory),bbox_inches='tight')
fig.savefig('%s/parallel_genes_bubble_plot.png' %
(config.analysis_directory),bbox_inches='tight', dpi=500)
plt.show()


# In[23]:


# Get list of parallel changing genes: final version??
# Any gene NAME that changes in at least 3/4 hosts across infant-infant,
# mother-infant(earliest) or adult-adult categories
```

```
host_threshold = 4

gene_hosts_dict = defaultdict(set)

for cat in ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']:
    for gene in num_host_genes_by_cohort[cat]:
        hosts = num_host_genes_by_cohort[cat][gene]
        for host in hosts:
            gene_hosts_dict[gene].add(host)

gene_num_hosts_dict = {gene: len(gene_hosts_dict[gene]) for gene in gene_hosts_dict}

sorted_gene_num_hosts = sorted(gene_num_hosts_dict.items(), key=lambda x: x[1], reverse=True)

any_parallel_genes = []
for gene, num_hosts in sorted_gene_num_hosts:
    if num_hosts >= host_threshold:
        any_parallel_genes.append(gene)

any_parallel_genes.remove('hypothetical protein')

print("There are %i parallel genes that mutate in parallel across >=%i hosts" %
(len(any_parallel_genes), host_threshold))


# In[27]:


# Survival curve version (flattened)
parallel_actual_counts = []
parallel_pres_null_counts = []
parallel_btwn_null_counts = []
parallel_pang_null_counts = []

all_actual_counts = []
all_pres_null_counts = []
all_btwn_null_counts = []
all_pang_null_counts = []

# cohort -> gene -> variant type -> count of SNP changes
# Originally, only looked at one infant-infant cohort
# cohort = 'Infant-Infant'
cohorts = ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']

all_genes = []
for cohort in cohorts:
    all_genes += genes_by_cohort[cohort].keys()

all_genes = list(set(all_genes))

for gene in all_genes:

    # Skip hypothetical protein
    if gene == 'hypothetical protein':
        continue

    actual_count = 0
    pres_null_count = 0
    btwn_null_count = 0
    pang_null_count = 0
```

```
    for cohort in cohorts:

        # Skip if gene not present in cohort
        if gene not in genes_by_cohort[cohort]:
            continue

        pres_null_count += (genes_by_cohort_present_null[cohort][gene])
        btwn_null_count += (genes_by_cohort_between_null[cohort][gene])
        pang_null_count += (genes_by_cohort_pangenome_null[cohort][gene])
        actual_count += sum(genes_by_cohort[cohort][gene].values())

    # Store info
    all_actual_counts.append(actual_count)
    all_pres_null_counts.append(pres_null_count)
    all_btwn_null_counts.append(btwn_null_count)
    all_pang_null_counts.append(pang_null_count)

    # Store info only if gene is special
    if gene in any_parallel_genes:
        parallel_actual_counts.append(actual_count)
        parallel_pres_null_counts.append(pres_null_count)
        parallel_btwn_null_counts.append(btwn_null_count)
        parallel_pang_null_counts.append(pang_null_count)

fig, ax = plt.subplots(figsize=(5, 3))

# xs, ns = calculate_unnormalized_survival_from_vector(all_actual_counts)
# ax.step(xs,ns/float(ns[0]),'-',linewidth=1.4, label='Observed', where='pre',zorder=4)
# xs, ns = calculate_unnormalized_survival_from_vector(all_btwn_null_counts)
# ax.step(xs,ns/float(ns[0]),'-',linewidth=1.4, label='Between null', where='pre',zorder=4)
xs, ns = calculate_unnormalized_survival_from_vector(all_pang_null_counts)
ax.step(xs,ns/float(ns[0]),'-',linewidth=1.4, label='Pangenome null', where='pre',zorder=4,
color='black', alpha=0.4)
xs, ns = calculate_unnormalized_survival_from_vector(all_pres_null_counts)
ax.step(xs,ns/float(ns[0]),'-',linewidth=1.4, label='Present null', where='pre',zorder=4,
color='black', alpha=0.9)

ax.set_ylim((0, 1))

all_counts = []
for gene in any_parallel_genes:
    count = 0
    for cohort in cohorts:
        if gene in genes_by_cohort[cohort]:
            count += sum(genes_by_cohort[cohort][gene].values())
    # ax.axvline(x=count, color='red', linewidth=0.6)
    all_counts.append(count)

# all_counts_ys = [0]*len(all_counts)
all_counts_ys = []
count_y_sofar_dict = defaultdict(float)
for count in all_counts:
    all_counts_ys.append(count_y_sofar_dict[count])
    count_y_sofar_dict[count] += 0.05

point = ax.plot(all_counts, all_counts_ys, '.', color='#7bb551', alpha=0.8, # #ff7f00
                markersize=10, zorder=99, label=('Gene mutated ' + r'in$\geq$4 hosts'))[0]
point.set_clip_on(False)
# ax.annotate(xy=(count, 0))
```

```
# Redundant for label
# ax.axvline(x=count, color='red', linewidth=0.6, label=('Gene mutated\n' + r'in$\geq$4 hosts'))

ax.set_xlabel("Number of SNV changes")
ax.set_ylabel(r'Fraction of genes $\geq \it{n}$')
ax.set_xscale('log')
ax.legend(loc='upper right', frameon=False) #,bbox_to_anchor=(0.5, 0.97))
ax.text(-0.2, 0.95, 'A', size=20, transform=ax.transAxes, weight='bold')
# ax.legend(loc='center right', bbox_to_anchor=(1.397, 0.5), frameon=False, fontsize=11)

# Circles on the x axis that are translucent
plt.show()

fig.savefig('%s/mutations_per_gene_nulls_survival_any.pdf' % plot_dir, bbox_inches='tight')
# fig.savefig('%s/mutations_per_gene_nulls_survival_any.png' % plot_dir, bbox_inches='tight',
dpi=500)


# In[28]:


# Make table
# Number of SNP changes/unique hosts having SNP changes per parallel gene and per timepoint pair
category
parallel_genes = any_parallel_genes
custom_cohorts_ordered = ['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult']

parallel_genes_by_cohort = {gene: {tt: 0 for tt in custom_cohorts_ordered} for gene in
parallel_genes}
count_by_parallel_gene = defaultdict(int)
host_parallel_genes_by_cohort = {gene: {tt: set() for tt in custom_cohorts_ordered} for gene in
parallel_genes}

for cohort in custom_cohorts_ordered:
    for gene in genes_by_cohort[cohort]:
        if gene in parallel_genes:
            num_snp_changes = sum(genes_by_cohort[cohort][gene].values())
            num_subject_tuples = len(num_host_genes_by_cohort[cohort][gene])
            parallel_genes_by_cohort[gene][cohort] = num_snp_changes
            count_by_parallel_gene[gene] += num_snp_changes
            for subject_tuple in num_host_genes_by_cohort[cohort][gene]:
                host_parallel_genes_by_cohort[gene][cohort].add(subject_tuple)

parallel_genes_ordered = []
for gene, count in sorted(count_by_parallel_gene.items(), key=lambda x: x[1], reverse=True):
    parallel_genes_ordered.append(gene)


# In[30]:


# Heatmap plot
fig, ax = plt.subplots(len(parallel_genes_ordered), 1, figsize=(3,8),
gridspec_kw={'wspace':0.025, 'hspace':0})

custom_bins = ['Mother-\nInfant', 'Infant-\nInfant', 'HMP\nAdults']
cmap = cmx.get_cmap('binary', 14) # Blues
colors = [cmap(x) for x in np.array([x for x in range(0,14)])/14.0]

mi_cohort_labels = []
```

```
gene_i = 0
for gene in parallel_genes_ordered:

    ax[gene_i].set_xlim((-0.5, len(custom_bins)-0.5))
    if len(gene) > 50:
        substrs = gene.split(' ')
        half_idx = (len(substrs)/2)
        half1 = ' '.join(substrs[:half_idx])
        half2 = ' '.join(substrs[half_idx:])
        gene_label = half1 + '\n' + half2
    else:
        gene_label = gene

    ax[gene_i].set_yticks([0.5])
    ax[gene_i].set_yticklabels([gene_label])
    ax[gene_i].set_xticks([])

    col_i = 0
    for tp_type, color in zip(['Mother-Infant(earliest)', 'Infant-Infant', 'Adult-Adult'],
['#7bb551', '#77acff', '#396651']):
        num_snp_changes = parallel_genes_by_cohort[gene][tp_type]
        num_hosts = len(host_parallel_genes_by_cohort[gene][tp_type])

        print("Gene %s | %s" % (gene, tp_type))
        if gene not in genes_by_cohort[tp_type]:
            print("Nothing")
        else:
            for vartype in genes_by_cohort[tp_type][gene]:
                print("%s: %i" % (vartype, genes_by_cohort[tp_type][gene][vartype]))
        print("# SNP changes: %i" % num_snp_changes)
        print("# hosts: %i" % num_hosts)
        print('')

        ax[gene_i].axvspan(col_i-0.5, col_i+0.5, facecolor=colors[num_hosts])
        ax[gene_i].axvline(col_i+0.5, color='black', linewidth=1)
        # ax[gene_i].plot([col_i], [gene_i], 'o', markersize = num_hosts*3.4,
color=colors[num_hosts])
        col_i += 1

    gene_i += 1

ax[0].set_xticks(np.arange(len(custom_bins)))
ax[0].set_xticklabels(custom_bins)
ax[0].xaxis.tick_top()

# Add colorbar

# define the bins and normalize
bounds = np.arange(0, 12)
norm = mplcol.BoundaryNorm(bounds, cmap.N)

# create a second axes for the colorbar
ax2 = fig.add_axes([0.98, 0.16, 0.04, 0.7])
cb = colorbar.ColorbarBase(ax2, cmap=cmap, norm=norm,
    spacing='proportional', ticks=bounds, boundaries=bounds, format='%1i')

ax[0].text(-2.15, 1.25, 'B', size=20, transform=ax[0].transAxes, weight='bold')
plt.tick_params(top=False)

plt.show()
```

```
fig.savefig('%s/parallel_genes_heatmap.pdf' % (config.analysis_directory),bbox_inches='tight')
fig.savefig('%s/parallel_genes_heatmap.png' % (config.analysis_directory),bbox_inches='tight',
dpi=700)


# In[28]:


# Try to make plot like Zhao and Lieberman 4D

host_threshold = 4
gene_hosts_dict = defaultdict(set)

for cat in ['Infant-Infant', 'Adult-Adult', 'Mother-Infant(earliest)']:
    for gene in num_host_genes_by_cohort[cat]:
        hosts = num_host_genes_by_cohort[cat][gene]
        for host in hosts:
            gene_hosts_dict[gene].add(host)

gene_num_hosts_dict = {gene: len(gene_hosts_dict[gene]) for gene in gene_hosts_dict}
sorted_gene_num_hosts = sorted(gene_num_hosts_dict.items(), key=lambda x: x[1], reverse=True)
'''
# Infant-infant:
ii_parallel_genes = ['Periplasmic ligand-binding sensor domain COG3292 / BaeS-type histidine
kinase / OmpR-type DNA-binding response regulator',
                     'Outer membrane TonB-dependent transporter, utilization system for glycans
and polysaccharides (PUL), SusC family',
                     'Cell surface glycan-binding lipoprotein, utilization system for glycans and
polysaccharides (PUL), SusD family',
                     'Two-component system sensor histidine kinase', 'beta-galactosidase (EC
3.2.1.23)',
                     'Transcriptional regulator, AraC family', 'Sporulation transcription
regulator WhiA']

'''
ii_parallel_genes = []
for gene, num_hosts in sorted_gene_num_hosts:
    if num_hosts >= host_threshold:
        ii_parallel_genes.append(gene)

ii_parallel_genes.remove('hypothetical protein')
print("There are %i parallel genes that mutate in parallel across >=%i hosts" %
(len(ii_parallel_genes), host_threshold))

# dN/dS for parallel changing genes
# dN/dS of all SNP changes, excluding those that are involved in parallelism
# dN/dS between hosts: very low

# Report SNPs per base pair, if concerned about longer genes obscuring parallelism signal

variant_types = ['4D', '1D', '2D', '3D']

dnds_replacement = ([], [], [], []) # Replacements
dnds_mod_other = ([], [], [], []) # Tuple of four lists corresponding to N diffs, N opps, S
diffs, S opps
dnds_parallel = ([], [], [], [])
snps_per_bp = ([], [], [], [])

for species in snp_changes_consecutive:

    # print("Working on %s...." % species)
```

```
    genome_ids = midas_db_utils.get_ref_genome_ids(species)
    # load the gene descriptions for all genomes coresponding to this speceis:
    # gene_descriptions=parse_patric.load_patric_gene_descriptions(genome_ids)

    for sample_i, sample_j in snp_changes[species]:

        val = snp_changes[species][(sample_i, sample_j)]
        subject_tuple = (sample_subject_map[sample_i], sample_subject_map[sample_j])
        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j)

        # Originally just looking at II
        # Now look at Mother-Infant(earliest), Infant-Infant, Adult-Adult
        # if not (('Infant-Infant' in custom_cohorts)):
        #     continue
        if not (('Infant-Infant' in custom_cohorts) or ('Mother-Infant(earliest)' in
custom_cohorts) or ('Adult-Adult' in custom_cohorts)):
            continue
        # if not (('Infant-Infant' in custom_cohorts) or ('Mother-Infant(earliest)' in
custom_cohorts)):
        #     continue

        # Get dN/dS info
        nonsyn_diffs, nonsyn_opps, syn_diffs, syn_opps = dnds_info[species][(sample_i,
sample_j)]

        if type(val) == type([]): # Not a replacement

            # Loop over individual SNP changes
            for gene_id, contig, position, variant_type, A1, D1, A2, D2 in val:

                desc = gene_id_name_map[gene_id]
                if desc in ii_parallel_genes: # any_parallel_genes
                    dnds_parallel[0].append(nonsyn_diffs)
                    dnds_parallel[1].append(nonsyn_opps)
                    dnds_parallel[2].append(syn_diffs)
                    dnds_parallel[3].append(syn_opps)
                else:
                    dnds_mod_other[0].append(nonsyn_diffs)
                    dnds_mod_other[1].append(nonsyn_opps)
                    dnds_mod_other[2].append(syn_diffs)
                    dnds_mod_other[3].append(syn_opps)

        else: # Replacement, val is number of SNP differences
            dnds_replacement[0].append(nonsyn_diffs)
            dnds_replacement[1].append(nonsyn_opps)
            dnds_replacement[2].append(syn_diffs)
            dnds_replacement[3].append(syn_opps)


# In[29]:


for x in ii_parallel_genes:
    print(x)


# In[59]:


dnds_mod_other_actual =
(sum(dnds_mod_other[0])/sum(dnds_mod_other[1]))/(sum(dnds_mod_other[2])/sum(dnds_mod_other[3]))
```

```
dnds_mod_other_length = len(dnds_mod_other[2])
dnds_parallel_actual =
(sum(dnds_parallel[0])/sum(dnds_parallel[1]))/(sum(dnds_parallel[2])/sum(dnds_parallel[3]))
dnds_parallel_length = len(dnds_parallel[2])
dnds_replacement_actual =
(sum(dnds_replacement[0])/sum(dnds_replacement[1]))/(sum(dnds_replacement[2])/sum(dnds_replaceme
nt[3]))
dnds_replacement_length = len(dnds_replacement[2])

print(dnds_mod_other_actual)
print(dnds_parallel_actual)
print(dnds_replacement_actual)


# In[60]:


# Bootstrap dN/dS values and perform T-test between special parallel genes and other changing
genes
# Subsample to 50 SNP changes without replacement

num_trials = 1000
lower_ci_idx = int(num_trials*0.025)
upper_ci_idx = int(num_trials*0.975)
subsample_size = 50

parallel_dnds_list = []
mod_other_dnds_list = []
replacement_dnds_list = []

for _ in range(num_trials):
    for dnds_cat, dnds_list in zip([dnds_parallel, dnds_mod_other, dnds_replacement],
                                   [parallel_dnds_list, mod_other_dnds_list,
replacement_dnds_list]):
        nonsyn_diffs = np.random.choice(dnds_cat[0], size=subsample_size, replace=False)
        nonsyn_opps = np.random.choice(dnds_cat[1], size=subsample_size, replace=False)
        syn_diffs = np.random.choice(dnds_cat[2], size=subsample_size, replace=False)
        syn_opps = np.random.choice(dnds_cat[3], size=subsample_size, replace=False)

        dnds = (sum(nonsyn_diffs)/sum(nonsyn_opps))/(sum(syn_diffs)/sum(syn_opps))

        dnds_list.append(dnds)

import scipy.stats as stats
t, p = stats.ttest_ind(parallel_dnds_list, mod_other_dnds_list)
D = cohenD(parallel_dnds_list, mod_other_dnds_list)
print("T: %.02f\tp: %s" % (t, str(p)))
print("D: %s" % str(D))

fig, ax = plt.subplots(figsize=(3, 3))

xs = [2.5, 1.5, 0.5] # These are the bar indices, should be ys if horizontal bars
colors = ['#7bb551', '#9ec9f1', '#fd996a']
# ['#ff7f00', '#9ec9f1', '#fd996a'] # '#deebf7', '#fee0d2'] '#fd996a'
for x, val, color in zip(xs, [dnds_parallel_actual, dnds_mod_other_actual,
dnds_replacement_actual], colors):
    ax.barh([x], val, height=0.5, color=color, alpha=0.8)
ax.hlines(xs[0], sorted(parallel_dnds_list)[lower_ci_idx],
sorted(parallel_dnds_list)[upper_ci_idx])
ax.hlines(xs[1], sorted(mod_other_dnds_list)[lower_ci_idx],
sorted(mod_other_dnds_list)[upper_ci_idx])
```

```
ax.hlines(xs[2], sorted(replacement_dnds_list)[lower_ci_idx],
sorted(replacement_dnds_list)[upper_ci_idx])

ax.axvline(x=1, ls='--', linewidth=0.8, color='black', alpha=0.5, zorder=-99)

ax.set_ylim((0, 3))
ax.set_yticks(xs)
ax.set_yticklabels(['Genes modified\n' + r'$in\geq$4 hosts', 'Other modified\ngenes', 'Genes
that differ\nin replacements'])
ax.text(-0.66, 0.95, 'C', size=20, transform=ax.transAxes, weight='bold')
ax.set_xlabel(r'$d_N/d_S$')
# plt.boxplot([parallel_dnds_list, mod_other_dnds_list, replacement_dnds_list])
plt.show()

fig.savefig('%s/parallel_genes_dnds.pdf' % (config.analysis_directory),bbox_inches='tight')
# fig.savefig('%s/parallel_genes_dnds.png' % (config.analysis_directory),bbox_inches='tight',
dpi=500)


# In[62]:


# T-test on the raw data
nonsyn_diffs, nonsyn_opps, syn_diffs, syn_opps = dnds_parallel
dnds_parallel_actual = []
for i in range(len(nonsyn_diffs)):
    if syn_diffs[i] == 0:
        pass
    else:
        dnds_parallel_actual.append((nonsyn_diffs[i]/nonsyn_opps[i])/(syn_diffs[i]/syn_opps[i]))

nonsyn_diffs, nonsyn_opps, syn_diffs, syn_opps = dnds_mod_other
dnds_mod_other_actual = []
for i in range(len(nonsyn_diffs)):
    if syn_diffs[i] == 0:
        pass
    else:

dnds_mod_other_actual.append((nonsyn_diffs[i]/nonsyn_opps[i])/(syn_diffs[i]/syn_opps[i]))

t, p = stats.ttest_ind(dnds_parallel_actual, dnds_mod_other_actual)
D = cohenD(dnds_parallel_actual, dnds_mod_other_actual)
print("T: %.02f\tp: %s" % (t, str(p)))
print("D: %s" % str(D))

plt.boxplot([dnds_parallel_actual, dnds_mod_other_actual])


# In[63]:


# Permutation test?

def permutation_test(orig_label_data_dict, label_set, aggregator_fn, num_bootstraps=1000):

    orig_data = []; labels = []
    for label in label_set:
        for orig_datum in orig_label_data_dict[label]:
            orig_data.append(orig_datum)
            labels.append(label)
```

```
    # First compute true rate
    true_value_by_label = {label: aggregator_fn(orig_label_data_dict[label]) for label in
orig_label_data_dict}

    # Next compute null rates
    permuted_values_by_label = {label: [] for label in label_set}

    for _ in range(num_bootstraps):
        random.shuffle(labels) # Randomly permute labels
        label_data_dict = defaultdict(list)
        for datum, label in zip(orig_data, labels):
            label_data_dict[label].append(datum)

        for label in label_set:
            permuted_values_by_label[label].append(aggregator_fn(label_data_dict[label]))

    # Compute differences between the labels
    label1, label2 = label_set[0], label_set[1]

    null_differences = []
    for val1, val2 in zip(permuted_values_by_label[label1], permuted_values_by_label[label2]):
        null_differences.append(val1-val2)

    true_difference = true_value_by_label[label1]-true_value_by_label[label2]

    return (null_differences, true_difference)


# In[83]:


def get_overall_dnds(vals):
    total_nonsyn_diff = 0.0
    total_nonsyn_opp = 0.0
    total_syn_diff = 0.0
    total_syn_opp = 0.0
    for nonsyn_diff, nonsyn_opp, syn_diff, syn_opp in vals:
        total_nonsyn_diff += nonsyn_diff
        total_nonsyn_opp += nonsyn_opp
        total_syn_diff += syn_diff
        total_syn_opp += syn_opp
    return ((total_nonsyn_diff/total_nonsyn_opp)/(total_syn_diff/total_syn_opp))

def permutation_test_p(nulls, true):
    # Firstly check if true is above or below median
    median = np.median(nulls); n = len(nulls)
    if true < median: # Below median
        sorted_nulls = sorted(nulls)
        for rank, null in zip(np.arange(0, n), sorted_nulls):
            if true < null:
                return rank/float(n)
    else: # Above median
        sorted_nulls = sorted(nulls, reverse=True)
        for rank, null in zip(np.arange(n, 0, step=-1), sorted_nulls):
            if true > null:
                return 1-(rank/float(n))


# In[87]:
```

```
null_differences, true_difference = permutation_test(orig_label_data_dict, ['parallel',
'mod_other'], get_overall_dnds, num_bootstraps=1000)
print(permutation_test_p(null_differences, true_difference))
plt.violinplot(null_differences)
plt.plot([1], [true_difference], '.')


# In[57]:


t, p = stats.ttest_ind(parallel_dnds_list[:3], mod_other_dnds_list[:3])
print("T: %.02f\tp: %.3E" % (t, p))


# In[38]:


# Distribution of values of number of parallel mutated hosts per gene ID
plt.plot([num for _, num in sorted_gene_num_hosts], '.')

for gene, num in sorted_gene_num_hosts:
    if num > 20 and gene != 'hypothetical protein':
        print("Max number excluding hypothetical protein: %i for %s" % (num, gene))
plt.show()


# In[39]:


fig, ax = plt.subplots()

dnds_lens = [len(dnds_list) for dnds_list in [dnds_parallel[0], dnds_mod_other[0],
dnds_replacement[0]]]

ys = np.arange(3)
ax.barh(ys, [dnds_parallel_actual, dnds_mod_other_actual, dnds_replacement_actual], height=0.6)
ax.set_yticks(ys)

labels = ['SNP changes in genes that mutate in parallel across hosts',
                    'SNP changes in other genes in modification events',
                    'Replacement events']

new_labels = []
for label, dnds_len in zip(labels, dnds_lens):
    new_labels.append(label + ('\nn=%i' % dnds_len))

ax.set_yticklabels(new_labels)
ax.set_xlabel("dN/dS")

plt.show()


# In[143]:


# Now try bootstrapping

subsample_size = 30
num_bootstraps = 100

dnds_parallel_actual = []
```

```
dnds_mod_other_actual = []
dnds_relacement_actual = []

for _ in range(num_bootstraps):

    dnds_mod_other_bootstrapped = [[], [], [], []]

    i = 0
    for elem in dnds_mod_other:
        print(len(elem))
        bootstrap = np.random.choice(elem, size=subsample_size, replace=False)
        dnds_mod_other_bootstrapped[i] = bootstrap
        i += 1

    dnds_parallel_bootstrapped = [[], [], [], []]

    i = 0
    for elem in dnds_parallel:
        bootstrap = np.random.choice(elem, size=subsample_size, replace=False)
        dnds_parallel_bootstrapped[i] = bootstrap
        i += 1

    dnds_replacement_bootstrapped = [[], [], [], []]

    i = 0
    for elem in dnds_replacement:
        bootstrap = np.random.choice(elem, size=subsample_size, replace=False)
        dnds_replacement_bootstrapped[i] = bootstrap
        i += 1


dnds_mod_other_actual.append(sum(dnds_mod_other_bootstrapped[0])/sum(dnds_mod_other_bootstrapped
[1]))/(sum(dnds_mod_other_bootstrapped[2])/sum(dnds_mod_other_bootstrapped[3]))
    dnds_mod_other_length = len(dnds_mod_other_bootstrapped[2])

dnds_parallel_actual.append(sum(dnds_parallel_bootstrapped[0])/sum(dnds_parallel_bootstrapped[1]
))/(sum(dnds_parallel_bootstrapped[2])/sum(dnds_parallel_bootstrapped[3]))
    dnds_parallel_length = len(dnds_parallel_bootstrapped[2])

dnds_replacement_actual.append(sum(dnds_replacement_bootstrapped[0])/sum(dnds_replacement_bootst
rapped[1]))/(sum(dnds_replacement_bootstrapped[2])/sum(dnds_replacement_bootstrapped[3]))
    dnds_replacement_length = len(dnds_replacement_bootstrapped[2])

print(dnds_mod_other_actual)
print(dnds_parallel_actual)
print(dnds_replacement_actual)


# In[145]:


len(dnds_mod_other_bootstrapped)


# In[138]:


fig, ax = plt.subplots()

dnds_lens = [len(dnds_list) for dnds_list in [dnds_parallel[0], dnds_mod_other[0],
dnds_replacement[0]]]
```

```
ys = np.arange(3)
ax.boxplot([dnds_parallel_actual, dnds_mod_other_actual, dnds_replacement_actual])
ax.set_yticks(ys)

labels = ['SNP changes in genes that mutate in parallel across hosts',
                    'SNP changes in other genes in modification events',
                    'Replacement events']

new_labels = []
for label, dnds_len in zip(labels, dnds_lens):
    new_labels.append(label + ('\nn=%i' % dnds_len))

ax.set_yticklabels(new_labels)
ax.set_xlabel("dN/dS")

plt.show()


# In[25]:


# Do suspected de novo mutations in infants revert within hosts?

# subject -> species -> (day1, day2) -> list of SNP change tuples OR 'replacement'
host_snp_changes = defaultdict(dict)
# subject -> species -> (day1, day2) -> list of (vartype, fdict) tuples OR 'replacement
host_snp_changes_freqs = defaultdict(dict)

for species in snp_changes: # snp_changes_consecutive
    for sample_i, sample_j in snp_changes[species]: #snp_changes_consecutive

        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j)
        if not ('Infant-Infant' in custom_cohorts or 'Mother-Infant(earliest)' in
custom_cohorts): # Only look at mother-infant or infant-infant
            continue

        if 'Infant-Infant' in custom_cohorts:
            tp_type = 'II'
        if 'Mother-Infant(earliest)' in custom_cohorts:
            tp_type = 'MI'

        subject = sample_subject_map[sample_j][:-2] # Combine mother and infant subjects!
        day1, day2 = mi_sample_day_dict[sample_i], mi_sample_day_dict[sample_j]

        # Only consider Mother-Infant when mother timepoint is at delivery
        if tp_type == 'MI' and (day1 < 0 or day1 > 5):
            continue

        pdicts = snp_change_freqs_with_opps[species][(sample_i, sample_j)] # list of (vartype,
freq_dict, opp_dict tuples)
        npdict = snp_change_null_freqs[species][(sample_i, sample_j)] # dict: prev_cohort > list
of (freq, weight) tuples

        snp_change_val = snp_changes[species][(sample_i, sample_j)]

        nonsyn_diffs, nonsyn_opps, syn_diffs, syn_opps = dnds_info[species][(sample_i,
sample_j)]

        if type(snp_change_val) == type(1): # Replacement
            # continue # remove this to include replacement
```

```
            try:
                host_snp_changes[subject][species][(day1, day2, tp_type)] = 'replacement'
                host_snp_changes_freqs[subject][species][(day1, day2, tp_type)] = 'replacement'
            except:
                host_snp_changes[subject][species] = {(day1, day2, tp_type): 'replacement'}
                host_snp_changes_freqs[subject][species] = {(day1, day2, tp_type):
'replacement'}
        else: # elif len(snp_change_val) > 0: # formerly else to include no change
            try:
                host_snp_changes[subject][species][(day1, day2, tp_type)] =
snp_changes[species][(sample_i, sample_j)]
                host_snp_changes_freqs[subject][species][(day1, day2, tp_type)] = pdicts
            except:
                host_snp_changes[subject][species] = {(day1, day2, tp_type):
snp_changes[species][(sample_i, sample_j)]}
                host_snp_changes_freqs[subject][species] = {(day1, day2, tp_type): pdicts}


# In[87]:


# Investigate Backhed 59
species = 'Bifidobacterium_adolescentis_56815'
for sample_i, sample_j in snp_changes_consecutive[species]:
    custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j)
    subject = sample_subject_map[sample_j][:-2] # Combine mother and infant subjects!
    day1, day2 = mi_sample_day_dict[sample_i], mi_sample_day_dict[sample_j]
    if subject == '59':
        print((day1, day2, custom_cohorts))
        print(snp_changes[species][(sample_i, sample_j)])
        print('')


# In[64]:


num_host_species_pairs = 0
num_host_species_pairs_modification_twice = 0
desired_host_species_sites = []

for subject in host_snp_changes:
    for species in host_snp_changes[subject]:

        # Want to only keep earliest mother-infant timepoint pair urgh
        mi_min_day2 = 9999
        for day1, day2, tp_type in sorted(host_snp_changes[subject][species].keys(), key=lambda
x: x[0]):
            if tp_type == 'MI' and day2 < mi_min_day2:
                mi_min_day2 = day2

        mi_other_day2 = []
        for day1, day2, tp_type in sorted(host_snp_changes[subject][species].keys(), key=lambda
x: x[0]):
            if tp_type == 'MI' and day2 != mi_min_day2:
                mi_other_day2.append(day2)

        if (len(host_snp_changes[subject][species]) - len(mi_other_day2)) > 1: # More than one
tp pair

            num_host_species_pairs += 1
```

```
            # Check if there are at least two modification events
            num_modifications = 0
            for day1, day2, tp_type in sorted(host_snp_changes[subject][species].keys(),
key=lambda x: x[0]):
                if tp_type == 'MI' and day2 != mi_min_day2:
                    continue
                snp_change_list = host_snp_changes[subject][species][(day1, day2, tp_type)]
                if snp_change_list != 'replacement' and len(snp_change_list) > 0: # If this is a
modification..
                    num_modifications += 1

            if num_modifications >= 2:
                num_host_species_pairs_modification_twice += 1
            else: # Comment out to see everything, keep to restrict to modification twice
                continue

            print("Subject: %s, Species: %s" % (subject, species))
            sites = []

            for day1, day2, tp_type in sorted(host_snp_changes[subject][species].keys(),
key=lambda x: x[0]):

                # Want to only keep earliest mother-infant timepoint pair urgh
                if tp_type == 'MI' and day2 != mi_min_day2:
                    continue

                print('\n%s Day %i to day %i' % (tp_type, day1, day2))
                snp_change_list = host_snp_changes[subject][species][(day1, day2, tp_type)]
                snp_change_freqs_list = host_snp_changes_freqs[subject][species][(day1, day2,
tp_type)]

                if snp_change_list == 'replacement':
                        print("Replacement")
                        continue

                for snp_change, x_snp_change_freqs in zip(snp_change_list,
snp_change_freqs_list):
                    gene_id, contig, location, vartype, a1, d1, a2, d2 = snp_change

                    try:
                        gene_name = gene_id_name_map[gene_id]
                    except:
                        gene_name = 'Lack gene name info :('

                    vartype2, freq_dict, opp_dict = x_snp_change_freqs
                    if vartype != vartype2:
                        print("Weird")
                    sites.append((gene_id, contig, location))
                    print("Site: %s,%s,%s\t| Vartype: %s | %s" % (gene_id, contig, location,
vartype, gene_name))
                    print("\t%i %i %i %i | %.02f -> %.02f" % (a1, d1, a2, d2, float(a1)/d1,
float(a2)/d2))
                    print("\tInfant prev: %.02f | HMP prev: %.02f | Mother prev: %.02f" %
(freq_dict['nonpremie'], freq_dict['hmp'], freq_dict['mother']))
            desired_host_species_sites.append(([subject], species, sites))

print("\n================================================================================")


# In[89]:
```

```
num_host_species_pairs_modification_twice


# In[90]:


num_host_species_pairs


# In[27]:


# Alternative version, trying to see across replacements too
num_host_species_pairs = 0
num_host_species_pairs_modification_twice = 0
desired_host_species_sites = []

for subject in host_snp_changes:
    for species in host_snp_changes[subject]:

        if len(host_snp_changes[subject][species]) > 1: # More than one tp pair

            print(sorted(host_snp_changes[subject][species].keys(), key=lambda x: x[0]))

            num_host_species_pairs += 1

            # Check if there are at least two non-overlapping modification events
            num_modifications = 0
            for day1, day2, tp_type in sorted(host_snp_changes[subject][species].keys(),
key=lambda x: x[0]):
                snp_change_list = host_snp_changes[subject][species][(day1, day2, tp_type)]
                if snp_change_list != 'replacement' and len(snp_change_list) > 0: # If this is a
modification..
                    num_modifications += 1

            if num_modifications >= 2:
                num_host_species_pairs_modification_twice += 1
            else: # Comment out to see everything, keep to restrict to modification twice
                continue

            print("Subject: %s, Species: %s" % (subject, species))
            sites = []

            for day1, day2, tp_type in sorted(host_snp_changes[subject][species].keys(),
key=lambda x: x[0]):

                print('\n%s Day %i to day %i' % (tp_type, day1, day2))
                snp_change_list = host_snp_changes[subject][species][(day1, day2, tp_type)]
                snp_change_freqs_list = host_snp_changes_freqs[subject][species][(day1, day2,
tp_type)]

                if snp_change_list == 'replacement':
                        print("Replacement")
                        continue

                for snp_change, x_snp_change_freqs in zip(snp_change_list,
snp_change_freqs_list):
                    gene_id, contig, location, vartype, a1, d1, a2, d2 = snp_change

                    try:
```

```
                        gene_name = gene_id_name_map[gene_id]
                    except:
                        gene_name = 'Lack gene name info :('

                    vartype2, freq_dict, opp_dict = x_snp_change_freqs
                    if vartype != vartype2:
                        print("Weird")
                    sites.append((gene_id, contig, location))
                    print("Site: %s,%s,%s\t| Vartype: %s | %s" % (gene_id, contig, location,
vartype, gene_name))
                    print("\t%i %i %i %i | %.02f -> %.02f" % (a1, d1, a2, d2, float(a1)/d1,
float(a2)/d2))
                    print("\tInfant prev: %.02f | HMP prev: %.02f | Mother prev: %.02f" %
(freq_dict['nonpremie'], freq_dict['hmp'], freq_dict['mother']))
            desired_host_species_sites.append(([subject], species, sites))

print("\n================================================================================")


# # One more try at reversion enumeration

# In[35]:


# Do suspected de novo mutations in infants revert within hosts?

# subject -> species -> (day1, day2) -> list of SNP change tuples OR 'replacement'
host_snp_changes = defaultdict(dict)
# subject -> species -> (day1, day2) -> list of (vartype, fdict) tuples OR 'replacement
host_snp_changes_freqs = defaultdict(dict)

for species in snp_changes: # snp_changes_consecutive
    for sample_i, sample_j in snp_changes[species]: #snp_changes_consecutive

        custom_cohorts = custom_cohorts_of_sample_pair(custom_cohort_tests, sample_i, sample_j)
        if not ('Infant-Infant' in custom_cohorts or 'Mother-Infant(all)' in custom_cohorts): #
Only look at mother-infant or infant-infant
            continue

        if 'Infant-Infant' in custom_cohorts:
            tp_type = 'II'
        if 'Mother-Infant(all)' in custom_cohorts:
            tp_type = 'MI'

        subject = sample_subject_map[sample_j][:-2] # Combine mother and infant subjects!
        day1, day2 = mi_sample_day_dict[sample_i], mi_sample_day_dict[sample_j]

        # Only consider Mother-Infant when mother timepoint is at delivery
        if tp_type == 'MI' and (day1 < 0 or day1 > 5):
            continue

        pdicts = snp_change_freqs_with_opps[species][(sample_i, sample_j)] # list of (vartype,
freq_dict, opp_dict tuples)
        npdict = snp_change_null_freqs[species][(sample_i, sample_j)] # dict: prev_cohort > list
of (freq, weight) tuples

        snp_change_val = snp_changes[species][(sample_i, sample_j)]

        nonsyn_diffs, nonsyn_opps, syn_diffs, syn_opps = dnds_info[species][(sample_i,
sample_j)]
```

```
        if type(snp_change_val) == type(1): # Replacement
            # continue # remove this to include replacement
            try:
                host_snp_changes[subject][species][(day1, day2, tp_type)] = 'replacement'
                host_snp_changes_freqs[subject][species][(day1, day2, tp_type)] = 'replacement'
            except:
                host_snp_changes[subject][species] = {(day1, day2, tp_type): 'replacement'}
                host_snp_changes_freqs[subject][species] = {(day1, day2, tp_type):
'replacement'}
        else: # elif len(snp_change_val) > 0: # formerly else to include no change
            try:
                host_snp_changes[subject][species][(day1, day2, tp_type)] =
snp_changes[species][(sample_i, sample_j)]
                host_snp_changes_freqs[subject][species][(day1, day2, tp_type)] = pdicts
            except:
                host_snp_changes[subject][species] = {(day1, day2, tp_type):
snp_changes[species][(sample_i, sample_j)]}
                host_snp_changes_freqs[subject][species] = {(day1, day2, tp_type): pdicts}


# In[52]:


num_host_species_pairs = 0
num_host_species_pairs_modification_twice = 0
desired_host_species_sites = []

for subject in host_snp_changes:
    for species in host_snp_changes[subject]:

       # Check if there are at least two non-overlapping modification events
        modification_time_tups = []
        qp_time_tups = set()
        for day1, day2, tp_type in sorted(host_snp_changes[subject][species].keys(), key=lambda
x: x[0]):
            qp_time_tups.add((day1, tp_type[0]))
            qp_time_tups.add((day2, tp_type[1]))
            snp_change_list = host_snp_changes[subject][species][(day1, day2, tp_type)]
            if snp_change_list != 'replacement' and len(snp_change_list) > 0: # If this is a
modification..
                modification_time_tups.append((day1, day2, tp_type))


        if len(qp_time_tups) >= 3:

            num_host_species_pairs += 1

        if len(modification_time_tups) > 1:

            num_candidates = 0

            # For each candidate, check if there is at least one other time tup that does not
overlap
            for day1, day2, tp_type in modification_time_tups:

                non_overlap_count = 0

                for other_day1, other_day2, other_tp_type in modification_time_tups:
                    if tp_type == 'II' and other_tp_type == 'II':
                        if other_day1 >= day2 or other_day2 <= day1:
                            non_overlap_count += 1
```

```
                    if tp_type == 'II' and other_tp_type == 'MI':
                        if day1 >= other_day2:
                            non_overlap_count += 1
                    if tp_type == 'MI' and other_tp_type == 'II':
                        if other_day1 >= day2:
                            non_overlap_count += 1
                    if tp_type == 'MI' and other_tp_type == 'MI':
                        if other_day1 != day1:
                            print("Weird")

                if non_overlap_count >= 1:
                    num_candidates += 1

            if num_candidates >= 1: # This is what we want: at least two non-overlapping
modification events

                num_host_species_pairs_modification_twice += 1

                # Print information
================================================================================
                print("Subject: %s, Species: %s" % (subject, species))
                sites = []

                for day1, day2, tp_type in sorted(host_snp_changes[subject][species].keys(),
key=lambda x: x[0]):

                    print('\n%s Day %i to day %i' % (tp_type, day1, day2))
                    snp_change_list = host_snp_changes[subject][species][(day1, day2, tp_type)]
                    snp_change_freqs_list = host_snp_changes_freqs[subject][species][(day1,
day2, tp_type)]

                    if snp_change_list == 'replacement':
                            print("Replacement")
                            continue

                    for snp_change, x_snp_change_freqs in zip(snp_change_list,
snp_change_freqs_list):
                        gene_id, contig, location, vartype, a1, d1, a2, d2 = snp_change

                        try:
                            gene_name = gene_id_name_map[gene_id]
                        except:
                            gene_name = 'Lack gene name info :('

                        vartype2, freq_dict, opp_dict = x_snp_change_freqs
                        if vartype != vartype2:
                            print("Weird")
                        sites.append((gene_id, contig, location))
                        print("Site: %s,%s,%s\t| Vartype: %s | %s" % (gene_id, contig, location,
vartype, gene_name))
                        print("\t%i %i %i %i | %.02f -> %.02f" % (a1, d1, a2, d2, float(a1)/d1,
float(a2)/d2))
                        print("\tInfant prev: %.02f | HMP prev: %.02f | Mother prev: %.02f" %
(freq_dict['nonpremie'], freq_dict['hmp'], freq_dict['mother']))


print("\n================================================================================\n
")
                #
================================================================================
                desired_host_species_sites.append(([subject], species, sites))
```

```
# In[53]:


num_host_species_pairs_modification_twice


# In[54]:


num_host_species_pairs
```

plot_metadata.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[61]:


import config, parse_midas_data, sample_utils as su, temporal_changes_utils,
stats_utils, midas_db_utils, parse_patric
from collections import defaultdict
import numpy as np
from numpy.random import binomial as sample_binomial
import math
import pickle
import sys
import random
from math import log10,ceil,log,exp

import matplotlib.cm as cmx
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import matplotlib.colors as mcolors
import matplotlib.patheffects as pe
from matplotlib.patches import Patch

# Cohort list
cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'olm', 'hmp']

# Plot directory
plot_dir = "%s/" % (config.analysis_directory)

# Species list
good_species_list = parse_midas_data.load_pickled_good_species_list()

# Sample-subject-order maps
sys.stderr.write("Loading sample metadata...\n")
subject_sample_map = su.parse_subject_sample_map()
sample_order_map = su.parse_sample_order_map()
sample_subject_map = su.parse_sample_subject_map()
same_mi_pair_dict = su.get_same_mi_pair_dict(subject_sample_map)
sys.stderr.write("Done!\n")

# Timepoint pair types
tp_pair_names = ['MM', 'MI', 'II', 'AA']

# Cohorts
```

```
cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
mi_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

# Samples for each cohort
samples = {cohort: su.get_sample_names(cohort) for cohort in cohorts}
hmp_samples = su.get_sample_names('hmp')
mother_samples = su.get_sample_names('mother')
infant_samples = su.get_sample_names('infant')
olm_samples = su.get_sample_names('olm')

# Sample-cohort map
sample_cohort_map = su.parse_sample_cohort_map()

# Sample-timepoint map
mi_sample_day_dict = su.get_mi_sample_day_dict(exclude_cohorts=['olm'])
mi_tp_sample_dict = su.get_mi_tp_sample_dict(exclude_cohorts=['olm']) # no binning
mi_tp_sample_dict_binned, mi_tp_binned_labels =
su.get_mi_tp_sample_dict(exclude_cohorts=['olm'], binned=True)

# Narrow down samples to no Olm, and mother only at delivery
# (Exception made for Backhed, since )
infant_samples = [sample for sample in infant_samples if sample not in olm_samples]
mother_samples = [sample for sample in mother_samples if
(mi_sample_day_dict[sample] <= 30 and mi_sample_day_dict[sample] > -7)]
mi_samples = [sample for sample in (mother_samples + infant_samples) if sample not
in olm_samples]


# In[47]:


# Distribution of mother sample timepoints
# To decide which ones to exclude

day_count_dict = defaultdict(int)

for sample in su.get_sample_names('mother'):
    day = mi_sample_day_dict[sample]
    day_count_dict[day] += 1

for day in day_count_dict:
    print("Day %i:\t%i mother samples" % (day, day_count_dict[day]))


# In[48]:


# Qin metadata
f = open('%s/qin_ids.txt' % config.metadata_directory)
header = f.readline()

qin_subjects = set()
qin_samples = set()
qin_subject_sample_map = defaultdict(list)

for line in f:
    subject_id, sample_id, run_accession, country, continent, t2d =
line.strip().split('\t')
    qin_subjects.add(subject_id)
    qin_samples.add(run_accession)
    qin_subject_sample_map[subject_id].append(run_accession)
```

```
    sample_cohort_map[run_accession] = 'qin' # Update sample_cohort_map

print("Qin: %i subjects, %i samples" % (len(qin_subjects), len(qin_samples)))
print("# samples per subject: %s" % str(set([len(subject_sample_map[subj]) for subj
in subject_sample_map])))


# In[49]:


# EVERY SAMPLE CONSIDERED IN THIS STUDY

all_samples = list(mi_samples) + list(hmp_samples) + list(qin_samples)

print("%i total samples from all cohorts" % len(all_samples))

# Make subject_cohort_map
subject_cohort_map = {}
# Combine mother infant as same subject
mi_subject_cohort_map = {}
cohort_subjects = defaultdict(list)

for subject in subject_sample_map: # Includes Qin
    for some_sample in subject_sample_map[subject]:
        if some_sample in all_samples:
            cohort = sample_cohort_map[some_sample]
            subject_cohort_map[subject] = cohort
        else:
            continue

        if some_sample in mi_samples:
            combined_subject = subject[:-2]
            mi_subject_cohort_map[combined_subject] = cohort
        else:
            mi_subject_cohort_map[subject] = cohort

for subject in mi_subject_cohort_map:
    cohort = mi_subject_cohort_map[subject]
    cohort_subjects[cohort].append(subject)

print("Total number of mother-infant subjects (excludes Qin): %i" %
len(mi_subject_cohort_map))

# EVERY SUBJECT
all_subjects = list(qin_subjects) + list(set([(sample_subject_map[sample]) for
sample in all_samples if sample not in qin_samples]))

print("Total number of subjects: %i" % (len(all_subjects)))


# In[50]:


# NOTE: there are 4 Shao subjects that ONLY have one mother sample.
# Not counting these in number of hosts in Table 1.

# Also, there is 1 Yassour subject that has ONLY one mother sample
# (when mother is restricted to delivery). Also not counting this host.
for cohort in cohort_subjects:
    print("%s: %i subjects" % (cohort, len(set(cohort_subjects[cohort]))))
```

```
# In[51]:


# Subjects per broad type
subjects_count = {'mother': set(), 'infant': set(), 'adult': set()}

# Subjects per cohort (infant)
cohorts_count = defaultdict(set)

# Subjects per cohort (mother)
cohorts_mother_count = defaultdict(set)

# Samples per cohort (combine mother and infant)
samples_cohort_count = defaultdict(int)

for sample in sample_subject_map:
    subject = sample_subject_map[sample]
    if sample in mother_samples:
        x = 'mother'
        cohort = sample_cohort_map[sample]
        cohorts_mother_count[cohort].add(subject)
        samples_cohort_count[cohort] += 1
    elif sample in infant_samples:
        x = 'infant'
        cohort = sample_cohort_map[sample]
        cohorts_count[cohort].add(subject)
        samples_cohort_count[cohort] += 1
    elif sample in hmp_samples:
        x = 'adult' # technically I mean HMP
        cohorts_count['hmp'].add(subject)
        samples_cohort_count['hmp'] += 1
    else:
        continue
    subjects_count[x].add(subject)

print("%i mother subjects, %i samples" % (len(subjects_count['mother']),
len(mother_samples)))
print("%i infant subjects, %i samples" % (len(subjects_count['infant']),
len(infant_samples)))
print("%i total MI samples" % len(mi_samples))
print("%i HMP subjects, %i samples" % (len(subjects_count['adult']),
len(hmp_samples)))

print('')

for cohort in cohorts_count:
    print("%s: %i subjects" % (cohort, len(cohorts_count[cohort])))


# In[52]:


# Summarize number of timepoints per host (combined mother-infant)

cohort_combined_subject_sample_dict = {cohort: defaultdict(set) for cohort in
['hmp', 'shao', 'ferretti', 'yassour', 'backhed', 'qin']}

for sample in all_samples:

    if sample in qin_samples:
```

```
        subject = sample
        cohort = 'qin'
    else:
        subject = sample_subject_map[sample]
        cohort = sample_cohort_map[sample]

    if sample in mi_samples:
        subject = subject[:-2]

    cohort_combined_subject_sample_dict[cohort][subject].add(sample)

for cohort in cohort_combined_subject_sample_dict:
    print(cohort)

print(sorted(list(set([len(cohort_combined_subject_sample_dict[cohort][subject])
for subject in cohort_combined_subject_sample_dict[cohort]]))))


# In[ ]:


# Num QP vs. non-QP samples in mother-infant
# Note, this is consistent with 7063 infant + 1159 mother QP samples

num_non = 0
num_qp = 0
for species in good_species_list:
    test = su.load_qp_samples(mi_samples, species)
    num_non += len(test['non-qp'])
    num_qp += len(test['qp'])

print("Number of QP mother-infant samples: %i" % num_qp)
print("Number of non-QP mother-infant samples: %i" % num_non)


# In[131]:


# Num QP vs. non-QP samples in HMP

num_non_hmp = 0
num_qp_hmp = 0
for species in good_species_list:
    test = su.load_qp_samples(hmp_samples, species)
    num_non_hmp += len(test['non-qp'])
    num_qp_hmp += len(test['qp'])

print("Number of QP HMP samples: %i" % num_qp_hmp)
print("Number of non-QP HMP samples: %i" % num_non_hmp)


# In[7]:


# Get some basic metadata numbers for report (once more)

cohort_infant_subjects_map = {cohort: set() for cohort in mi_cohorts}
cohort_mother_subjects_map = {cohort: set() for cohort in mi_cohorts}

for sample in mi_samples_no_olm:
    subject = sample_subject_map[sample]
```

```
    cohort = sample_cohort_map[sample]
    if sample in mother_samples:
        cohort_mother_subjects_map[cohort].add(subject)
    elif sample in infant_samples:
        cohort_infant_subjects_map[cohort].add(subject)
    else:
        print("Shouldn't happen!")

for cohort in mi_cohorts:
    print("%s: %i infants, %i mothers" % (cohort,
                                          len(cohort_infant_subjects_map[cohort]),
                                          len(cohort_mother_subjects_map[cohort])))

print("Total infant subjects: %i" % (sum([len(cohort_infant_subjects_map[cohort])
for cohort in mi_cohorts])))
print("Total mother subjects: %i" % (sum([len(cohort_mother_subjects_map[cohort])
for cohort in mi_cohorts])))


# In[65]:


# Make sure tp bin and precise day match
for tp_bin in mi_tp_binned_labels:
    day_set = set()
    cohort_set = set()
    for sample in mi_tp_sample_dict_binned['infant'][tp_bin]:
        day_set.add(mi_sample_day_dict[sample])
        cohort_set.add(sample_cohort_map[sample])
    print("%s:\t%i-%i\t\t%s" % (tp_bin, min(day_set), max(day_set),
str(cohort_set)))


# In[76]:


# Supplemental Figure S1

fig, ax = plt.subplots(figsize=(10, 5))

# mi_tp_binned_labels.remove('6 day')


cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
tp_labels = ['Mother'] + mi_tp_binned_labels[1:] + ['HMP']
tp_labels_abbreviated = ['M'] + [(label.split(' ')[1][0].upper() + label.split('
')[0]) for label in mi_tp_binned_labels[1:]] + ['HMP']

bin_counts_per_cohort = {cohort: np.zeros(len(tp_labels)) for cohort in cohorts}

for sample in mother_samples: # mother_samples should already not include Yassour
+/- 3 months
    cohort = sample_cohort_map[sample]
    bin_counts_per_cohort[cohort][0] += 1

for infant_tp_bin, idx in zip(mi_tp_binned_labels[1:], np.arange(1,
1+len(mi_tp_binned_labels[1:]))):

    if infant_tp_bin == '1 day': # Merge birth and 1 day into 1 day
        for sample in mi_tp_sample_dict_binned['infant']['birth']:
            cohort = sample_cohort_map[sample]
```

```
            bin_counts_per_cohort[cohort][idx] += 1

    for sample in mi_tp_sample_dict_binned['infant'][infant_tp_bin]:
        cohort = sample_cohort_map[sample]
        bin_counts_per_cohort[cohort][idx] += 1

for sample in hmp_samples:
    cohort = sample_cohort_map[sample]
    bin_counts_per_cohort[cohort][1+len(mi_tp_binned_labels[1:])] += 1

indices = np.arange(len(tp_labels))

acc = np.zeros(len(tp_labels))
for cohort in cohorts:
    cohort_label = 'HMP' if cohort == 'hmp' else cohort.capitalize()
    ax.bar(indices, bin_counts_per_cohort[cohort], label=cohort_label, bottom=acc)
    acc += bin_counts_per_cohort[cohort]

ax.set_ylabel("Number of samples")
# ax.set_yscale('log')
ax.set_xticks(indices)
ax.set_xticklabels(tp_labels_abbreviated)

ax.set_ylim((0, 700))
ax.legend(frameon=False, loc='upper center', ncol=len(cohorts))

plt.show()
fig.savefig('%s/samples_by_tp_cohorts.pdf' %
(config.analysis_directory),bbox_inches='tight')
fig.savefig('%s/samples_by_tp_cohorts.png' %
(config.analysis_directory),bbox_inches='tight', dpi=500)


# In[8]:


# Set up data and custom bins for metadata overview figure

custom_bins = ['Mother', 'Birth\n(meconium)', 'Day 1', 'Day 3-4', 'Day 6',
               'Week 1', 'Week 2', 'Week 3', 'Month 1', 'Month 2', 'Month 3',
               'Month 4', 'Month 12', 'Month 4-14']
custom_infant_bins = custom_bins[1:]
custom_bin_defs = {'Mother': (-100, 100), 'Birth\n(meconium)': (0, 0), 'Day 1': (1,
1), 'Day 3-4': (3,4), 'Day 6': (6,6),
                   'Week 1': (7, 13), 'Week 2': (14, 20), 'Week 3': (21, 21), 'Month
1': (30, 31), 'Month 2': (60, 62),
                   'Month 3': (91, 93), 'Month 4': (122, 122), 'Month 12': (366,
366), 'Month 4-14': (124, 440)}

def get_infant_tp_bin(tp, custom_infant_bins, custom_bin_defs):
    for bin_label in custom_infant_bins:
        start, end = custom_bin_defs[bin_label]
        if tp >= start and tp <= end:
            return bin_label
    return str(tp)

timepoints_by_cohort = {cohort: [] for cohort in mi_cohorts}
subject_ids_by_cohort = {cohort: [] for cohort in mi_cohorts}
all_timepoints = set()

custom_bins_by_cohort = {cohort: [] for cohort in mi_cohorts}
```

```
i = 0
for subject in subject_sample_map:
    for sample in subject_sample_map[subject]:
        if sample not in mi_samples_no_olm:
            continue
        cohort = sample_cohort_map[sample]
        day = mi_sample_day_dict[sample]

        if sample in mother_samples:
            if not (day >= 0 and day <= 7): # ignore other mom timepoints here
                continue
            custom_bin = 'Mother'
        else:
            custom_bin = get_infant_tp_bin(day, custom_infant_bins,
custom_bin_defs)
        custom_bins_by_cohort[cohort].append(custom_bin)

        timepoints_by_cohort[cohort].append(day)
        subject_ids_by_cohort[cohort].append(i)

        all_timepoints.add(day)
    i += 1

# Custom tweak to Shao
custom_bins_by_cohort['shao'].remove('Month 12')


# In[13]:


# Show distribution of timepoints per subject by cohorts

fig, ax = plt.subplots(4, 1, figsize=(12,8), sharex=True,
gridspec_kw={'wspace':0.025, 'hspace':0})

cohort_i = 0
for cohort in mi_cohorts:
    timepoints = timepoints_by_cohort[cohort]
    subject_ids = subject_ids_by_cohort[cohort]
    ax[cohort_i].set_yticks([])
    ax[cohort_i].set_xticks([0, 3, 7, 14, 30.5, 61])
    # ax[cohort_i].set_xticklabels(['Birth\n(meconium)'])
    ax[cohort_i].plot(timepoints, subject_ids, '.')

    cohort_i += 1

plt.show()


# In[14]:


# Show timepoints by cohort
fig, ax = plt.subplots(4, 1, figsize=(14,2.4), gridspec_kw={'wspace':0.025,
'hspace':0})

mi_cohort_labels = []

cohort_i = 0
for cohort in mi_cohorts:
```

```
    custom_bin_idxs = []
    for custom_bin in set(custom_bins_by_cohort[cohort]):
        custom_bin_idxs.append(custom_bins.index(custom_bin))
    ax[cohort_i].set_xlim((-0.5, len(custom_bins)-0.5))
    ax[cohort_i].plot(custom_bin_idxs, [cohort_i]*len(custom_bin_idxs), 'o',
markersize=14)
    ax[cohort_i].set_yticks([cohort_i])
    ax[cohort_i].set_yticklabels([cohort.capitalize()], fontsize=14)
    ax[cohort_i].set_xticks([])
    for x in np.arange(0.5, len(custom_bins), step=1):
        ax[cohort_i].axvline(x=x)
    cohort_i += 1

ax[0].set_xticks(np.arange(len(custom_bins)))
ax[0].set_xticklabels(custom_bins)
ax[0].xaxis.tick_top()

plt.tick_params(top=False)
fig.savefig('%s/cohort_timepoints_v1.pdf' %
(config.analysis_directory),bbox_inches='tight')
plt.show()


# In[15]:


# Alternate: include numbers of samples per timepoint-cohort pair
fig, ax = plt.subplots(4, 1, figsize=(14,2.4), gridspec_kw={'wspace':0.025,
'hspace':0})

mi_cohort_labels = []
for cohort in mi_cohorts:
    mi_cohort_labels.append('%s\n%i subjects')

cohort_i = 0
for cohort in mi_cohorts:
    custom_bin_idxs = []
    for custom_bin in set(custom_bins_by_cohort[cohort]):
        custom_bin_idxs.append(custom_bins.index(custom_bin))
    ax[cohort_i].set_xlim((-0.5, len(custom_bins)-0.5))
    for custom_bin_idx in custom_bin_idxs:
        custom_bin = custom_bins[custom_bin_idx]
        ax[cohort_i].text(custom_bin_idx, 0.5, 'n=%i' %
(custom_bins_by_cohort[cohort].count(custom_bin)),
                          ha='center', va='center', fontsize=12)
    ax[cohort_i].set_yticks([0.5])
    ax[cohort_i].set_yticklabels([cohort.capitalize()], fontsize=14)
    ax[cohort_i].set_xticks([])
    for x in np.arange(0.5, len(custom_bins), step=1):
        ax[cohort_i].axvline(x=x)
    cohort_i += 1

ax[0].set_xticks(np.arange(len(custom_bins)))
ax[0].set_xticklabels(custom_bins)
ax[0].xaxis.tick_top()

plt.tick_params(top=False)
fig.savefig('%s/cohort_timepoints_v2.pdf' %
(config.analysis_directory),bbox_inches='tight')
plt.show()
```

plot_qp.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


from utils import parse_midas_data, sample_utils as su, config
from collections import defaultdict
from matplotlib import pyplot as plt
import numpy as np
import pickle

# Load species list, timepoint-sample dict
good_species_list = parse_midas_data.load_pickled_good_species_list()
mi_tp_sample_dict = su.get_mi_tp_sample_dict(exclude_cohorts = ['olm'])
mi_sample_day_dict = su.get_mi_sample_day_dict()
olm_samples = su.get_sample_names('olm')
hmp_samples = su.get_sample_names('hmp')
mother_samples = su.get_sample_names('mother')
mother_samples_no_olm = [sample for sample in mother_samples if sample not in
olm_samples]
infant_samples = su.get_sample_names('infant')
infant_samples_no_olm = [sample for sample in infant_samples if sample not in
olm_samples]
mother_tps_ordered = sorted(mi_tp_sample_dict['mother'].keys())
infant_tps_ordered = sorted(mi_tp_sample_dict['infant'].keys())

# From pickle_qp.py
sample_species_qp_dict = pickle.load(open("%s/pickles/sample_species_qp_dict.pkl" %
config.data_directory, 'rb'))

# Narrow down samples to no Olm, and mother only at delivery
infant_samples = [sample for sample in infant_samples if sample not in olm_samples]
mother_samples = [sample for sample in mother_samples if
(mi_sample_day_dict[sample] < 7 and mi_sample_day_dict[sample] > -7)]
mi_samples = [sample for sample in (mother_samples + infant_samples) if sample not
in olm_samples]

# All considered samples excluding Qin
all_samples = list(mi_samples) + list(hmp_samples)


# In[ ]:


# For binned plots
mi_tp_sample_dict, infant_tps_ordered = su.get_mi_tp_sample_dict(exclude_cohorts =
['olm'], binned = True)
mother_tps_ordered = sorted(mi_tp_sample_dict['mother'].keys())


# In[ ]:


# ======================================================================
# Build qp information for infants (species -> tp -> qp samples count)
# ======================================================================
```

```
num_qp_dict = {species: {tp: 0 for tp in infant_tps_ordered} for species in
good_species_list}
num_non_dict = {species: {tp: 0 for tp in infant_tps_ordered} for species in
good_species_list}
num_lowcov_dict = {species: {tp: 0 for tp in infant_tps_ordered} for species in
good_species_list}

for species in good_species_list:

    for tp in mi_tp_sample_dict['infant']:
        samples = mi_tp_sample_dict['infant'][tp]
        num_samples = len(samples)

        qp_sample_sets = su.load_qp_samples(samples, species)
        num_qp = len(qp_sample_sets['qp'])
        num_non = len(qp_sample_sets['non-qp'])
        num_lowcov = len(qp_sample_sets['low-coverage'])

        alt_num_qp = 0
        alt_num_non = 0
        alt_num_lowcov = 0
        # Sanity check with alternative pickle
        for sample in samples:
            qp_status = sample_species_qp_dict['infant'][sample][species]
            if qp_status == 'qp':
                alt_num_qp += 1
            elif qp_status == 'non-qp':
                alt_num_non += 1
            elif qp_status == 'low-coverage':
                alt_num_lowcov += 1

        if (num_qp != alt_num_qp) or (num_non != alt_num_non) or (num_lowcov !=
alt_num_lowcov):
            print("Weird")

        num_qp_dict[species][tp] = num_qp
        num_non_dict[species][tp] = num_non
        num_lowcov_dict[species][tp] = num_lowcov


# In[ ]:


# Construct with alternative pickle
num_qp_dict = {cat: defaultdict(dict) for cat in ['mother', 'infant', 'hmp']}
num_non_dict = {cat: defaultdict(dict) for cat in ['mother', 'infant', 'hmp']}
num_lowcov_dict = {cat: defaultdict(dict) for cat in ['mother', 'infant', 'hmp']}

for cat in ['mother', 'infant', 'hmp']:
    for sample in sample_species_qp_dict[cat]:

        # Dummy day if HMP adult
        tp = -1 if cat == 'hmp' else mi_sample_day_dict[sample]

        for species in sample_species_qp_dict[cat][sample]:
            qp_status = sample_species_qp_dict[cat][sample][species]
            if qp_status == 'qp':
                if tp not in num_qp_dict[cat][species]:
                    num_qp_dict[cat][species][tp] = 0
                num_qp_dict[cat][species][tp] += 1
            elif qp_status == 'non-qp':
```

```
                if tp not in num_non_dict[cat][species]:
                    num_non_dict[cat][species][tp] = 0
                num_non_dict[cat][species][tp] += 1
            elif qp_status == 'low-coverage':
                if tp not in num_lowcov_dict[cat][species]:
                    num_lowcov_dict[cat][species][tp] = 0
                num_lowcov_dict[cat][species][tp] += 1


# In[ ]:


# QP info PER SAMPLE
# Construct with alternative pickle

# cat -> sample -> num QP/non/lowcov species
sample_num_qp_dict = {cat: defaultdict(int) for cat in ['mother', 'infant', 'hmp']}
sample_num_non_dict = {cat: defaultdict(int) for cat in ['mother', 'infant',
'hmp']}
sample_num_lowcov_dict = {cat: defaultdict(int) for cat in ['mother', 'infant',
'hmp']}

for cat in ['mother', 'infant', 'hmp']:
    for sample in sample_species_qp_dict[cat]:

        # Dummy day if HMP adult
        tp = -1 if cat == 'hmp' else mi_sample_day_dict[sample]

        for species in sample_species_qp_dict[cat][sample]:
            qp_status = sample_species_qp_dict[cat][sample][species]
            if qp_status == 'qp':
                sample_num_qp_dict[cat][sample] += 1
            elif qp_status == 'non-qp':
                sample_num_non_dict[cat][sample] += 1
            elif qp_status == 'low-coverage':
                sample_num_lowcov_dict[cat][sample] += 1


# In[ ]:


# Construct with alternative pickle (binned version)
num_qp_dict = {cat: defaultdict(dict) for cat in ['mother', 'infant', 'hmp']}
num_non_dict = {cat: defaultdict(dict) for cat in ['mother', 'infant', 'hmp']}
num_lowcov_dict = {cat: defaultdict(dict) for cat in ['mother', 'infant', 'hmp']}

for cat in ['mother', 'infant', 'hmp']:
    for sample in sample_species_qp_dict[cat]:

        # Dummy day if HMP adult
        if cat == 'hmp':
            tp = 'Adult'
        else:
            for tp in mi_tp_sample_dict[cat]:
                if sample in mi_tp_sample_dict[cat][tp]:
                    break

        for species in sample_species_qp_dict[cat][sample]:
            qp_status = sample_species_qp_dict[cat][sample][species]
            if qp_status == 'qp':
                if tp not in num_qp_dict[cat][species]:
```

```
                    num_qp_dict[cat][species][tp] = 0
                num_qp_dict[cat][species][tp] += 1
            elif qp_status == 'non-qp':
                if tp not in num_non_dict[cat][species]:
                    num_non_dict[cat][species][tp] = 0
                num_non_dict[cat][species][tp] += 1
            elif qp_status == 'low-coverage':
                if tp not in num_lowcov_dict[cat][species]:
                    num_lowcov_dict[cat][species][tp] = 0
                num_lowcov_dict[cat][species][tp] += 1


# In[ ]:


# ====================================================================
# Pickle
# ====================================================================

import pickle
ddir = config.data_directory

pickle.dump(num_qp_dict, open("%s/pickles/num_qp_dict.pkl" % ddir, 'wb'))
pickle.dump(num_non_dict, open("%s/pickles/num_non_dict.pkl" % ddir, 'wb'))
pickle.dump(num_lowcov_dict, open("%s/pickles/num_lowcov_dict.pkl" % ddir, 'wb'))


# In[ ]:


import pickle
ddir = config.data_directory

num_qp_dict = pickle.load(open("%s/pickles/num_qp_dict.pkl" % ddir, 'rb'))
num_non_dict = pickle.load(open("%s/pickles/num_non_dict.pkl" % ddir, 'rb'))
num_lowcov_dict = pickle.load(open("%s/pickles/num_lowcov_dict.pkl" % ddir, 'rb'))


# In[ ]:


# Get statistics for paper

cats = ['infant', 'hmp', 'mother']
qp_counts_agg = {cat: defaultdict(int) for cat in cats}

for cat in cats:

    num_samples_considered = 0

    for sample in sample_species_qp_dict[cat]:

        if sample not in all_samples:
            continue

        num_samples_considered += 1

        for species in sample_species_qp_dict[cat][sample]:
            qp_status = sample_species_qp_dict[cat][sample][species]
            qp_counts_agg[cat][qp_status] += 1
```

```
    print("# samples in category %s: %i" % (cat, num_samples_considered))

for cat in cats:
    lowcov = qp_counts_agg[cat]['low-coverage']
    non = qp_counts_agg[cat]['non-qp']
    qp = qp_counts_agg[cat]['qp']
    print("Category %s:" % cat)
    print("%i lowcov, %i non-qp, %i qp" % (lowcov, non, qp))
    print("Proportion QP: %.03f" % (qp/float(non+qp)))


# In[ ]:


print(len(good_species_list))


# In[ ]:


# ========================================================================
# Bar plots comparing QP vs. non-QP sample counts over timepoints
# aggregated over species + combining datasets
# ========================================================================

import numpy as np
from matplotlib import pyplot as plt

num_qp_agg_species = []
num_non_agg_species = []
num_lowcov_agg_species = []

for tp in infant_tps_ordered:
       total_num_qp = 0
       total_num_non = 0
       total_num_lowcov = 0

       for species in good_species_list:
              total_num_qp += num_qp_dict[species][tp]
              total_num_non += num_non_dict[species][tp]
              total_num_lowcov += num_lowcov_dict[species][tp]

       num_qp_agg_species.append(total_num_qp)
       num_non_agg_species.append(total_num_non)
       num_lowcov_agg_species.append(total_num_lowcov)

labels = [str(tp) for tp in infant_tps_ordered]
xticks = np.arange(len(labels))

fig, ax = plt.subplots(figsize=(28,8))
ax.set_yscale('log')
ax.set_xlim((-0.5, max(infant_tps_ordered) + 0.5))
ax.bar(infant_tps_ordered, num_qp_agg_species, label='QP', color='orange',
linewidth=0)
ax.bar(infant_tps_ordered, num_non_agg_species, label='non-QP',
bottom=num_qp_agg_species, color='blue', linewidth=0)
num_highcov_agg_species = np.array(num_qp_agg_species) +
np.array(num_non_agg_species)
ax.bar(infant_tps_ordered, num_lowcov_agg_species, label='low cov',
bottom=num_highcov_agg_species, color='gray', linewidth=0)
# ax.set_xticklabels(labels)
```

```
ax.hlines(10, -0.5, max(infant_tps_ordered) + 0.5, linestyles='dashed')
ax.set_ylabel("Number of samples (from any cohort)")
ax.set_xlabel("Timepoint (days)")
ax.set_title("Number of QP samples by timepoint")
ax.legend()

plt.show()
# fig.savefig("%s/num_qp_over_time.pdf" % config.analysis_directory,
bbox_inches='tight')


# In[ ]:


# ======================================================================
# Bar/line plots comparing QP vs. non-QP proportions over timepoints
# ======================================================================

prop_qp_agg_species = []
prop_non_agg_species = []
new_infant_tps_ordered = []

for q, n, tp in zip(num_qp_agg_species, num_non_agg_species, infant_tps_ordered):
       num_total = float(q + n)
       if num_total != 0:
              prop_qp_agg_species.append(q/num_total)
              prop_non_agg_species.append(n/num_total)
              new_infant_tps_ordered.append(tp)

fig, ax = plt.subplots(figsize=(28,8))
ax.set_xlim((-0.5, max(new_infant_tps_ordered) + 0.5))
ax.bar(new_infant_tps_ordered, prop_qp_agg_species, label='QP', color='orange',
linewidth=0)
ax.bar(new_infant_tps_ordered, prop_non_agg_species, label='non-QP',
bottom=prop_qp_agg_species, color='blue', linewidth=0)
ax.set_ylabel("Proportion of high coverage samples (from any cohort)\n")
ax.set_xlabel("Timepoint (days)")
ax.set_title("Proportion of QP samples by timepoint")
ax.legend()

plt.show()
# fig.savefig("%s/prop_qp_over_time.pdf" % config.analysis_directory,
bbox_inches='tight')


# In[ ]:


fig, ax = plt.subplots(figsize=(10,6))
ax.plot(new_infant_tps_ordered, prop_qp_agg_species)
ax.set_ylabel("Proportion of high coverage samples with are QP (from any
cohort)\n")
ax.set_xlabel("Timepoint (days)")
ax.set_title("Proportion of QP samples by timepoint")

plt.show()
# fig.savefig("%s/prop_qp_over_time_line.pdf" % config.analysis_directory,
bbox_inches='tight')


# In[ ]:
```

```
# ========================================================================
# Aggregate over species, for each timepoint
# ========================================================================

tps_ordered_dict = {'mother': mother_tps_ordered, 'infant': infant_tps_ordered,
'hmp': [-1]}

num_qp_agg_species = {'infant': [], 'mother': [], 'hmp': []}
num_non_agg_species = {'infant': [], 'mother': [], 'hmp': []}
num_lowcov_agg_species = {'infant': [], 'mother': [], 'hmp': []}

for cat in ['mother', 'infant', 'hmp']:
    for tp in tps_ordered_dict[cat]:

        total_num_qp, total_num_non, total_num_lowcov = 0, 0, 0

        for species in good_species_list:
            if tp in num_qp_dict[cat][species]:
                total_num_qp += num_qp_dict[cat][species][tp]
            if tp in num_non_dict[cat][species]:
                total_num_non += num_non_dict[cat][species][tp]
            if tp in num_lowcov_dict[cat][species]:
                total_num_lowcov += num_lowcov_dict[cat][species][tp]

        num_qp_agg_species[cat].append(total_num_qp)
        num_non_agg_species[cat].append(total_num_non)
        num_lowcov_agg_species[cat].append(total_num_lowcov)


# In[ ]:


# ========================================================================
# Bar plots comparing QP vs. non-QP sample counts over timepoints
# aggregated over species + combining datasets
# ========================================================================

import numpy as np
from matplotlib import pyplot as plt

labels = [str(tp) for tp in infant_tps_ordered]
xticks = np.arange(len(labels))

fig, ax = plt.subplots(figsize=(28,8))
ax.set_yscale('log')
ax.set_xlim((-0.5, max(infant_tps_ordered) + 0.5))
ax.bar(infant_tps_ordered, num_qp_agg_species['infant'], label='QP',
color='orange', linewidth=0)
ax.bar(infant_tps_ordered, num_non_agg_species['infant'], label='non-QP',
bottom=num_qp_agg_species['infant'], color='blue', linewidth=0)
num_highcov_agg_species = np.array(num_qp_agg_species['infant']) +
np.array(num_non_agg_species['infant'])
ax.bar(infant_tps_ordered, num_lowcov_agg_species['infant'], label='low cov',
bottom=num_highcov_agg_species, color='gray', linewidth=0)
# ax.set_xticklabels(labels)
ax.hlines(10, -0.5, max(infant_tps_ordered) + 0.5, linestyles='dashed')
ax.set_ylabel("Number of samples (from any cohort)")
ax.set_xlabel("Timepoint (days)")
ax.set_title("Number of QP samples by timepoint")
```

```
ax.legend()

plt.show()
# fig.savefig("%s/num_qp_over_time.pdf" % config.analysis_directory,
bbox_inches='tight')


# In[ ]:


# ======================================================================
# Same as above but binned
# ======================================================================

labels = [str(tp) for tp in infant_tps_ordered]
xticks = np.arange(len(labels))

fig, ax = plt.subplots(figsize=(18,8))
ax.set_yscale('log')
ax.bar(xticks, num_qp_agg_species['infant'], label='QP', color='orange',
linewidth=0)
ax.bar(xticks, num_non_agg_species['infant'], label='non-QP',
bottom=num_qp_agg_species['infant'], color='blue', linewidth=0)
num_highcov_agg_species = np.array(num_qp_agg_species['infant']) +
np.array(num_non_agg_species['infant'])
ax.bar(xticks, num_lowcov_agg_species['infant'], label='low cov',
bottom=num_highcov_agg_species, color='gray', linewidth=0)
ax.set_xticks(np.array(xticks) + 0.5)
ax.set_xticklabels(labels)
ax.set_ylabel("Number of samples (from any cohort)")
ax.set_xlabel("Timepoint")
ax.set_title("Number of QP samples by timepoint")
ax.legend()

plt.show()
# fig.savefig("%s/num_qp_over_time_binned.pdf" % config.analysis_directory,
bbox_inches='tight')


# In[ ]:


# ======================================================================
# Store proportion QP/non-QP for each timepoint
# ======================================================================

tps_ordered_dict = {'mother': mother_tps_ordered, 'infant': infant_tps_ordered,
'hmp': ['Adult']}

num_qp_agg_species = {'infant': [], 'mother': [], 'hmp': []}
num_non_agg_species = {'infant': [], 'mother': [], 'hmp': []}
num_lowcov_agg_species = {'infant': [], 'mother': [], 'hmp': []}

for cat in ['mother', 'infant', 'hmp']:
    for tp in tps_ordered_dict[cat]:

        total_num_qp, total_num_non, total_num_lowcov = 0, 0, 0

        for species in good_species_list:
            if tp in num_qp_dict[cat][species]:
                total_num_qp += num_qp_dict[cat][species][tp]
```

```
            if tp in num_non_dict[cat][species]:
                total_num_non += num_non_dict[cat][species][tp]
            if tp in num_lowcov_dict[cat][species]:
                total_num_lowcov += num_lowcov_dict[cat][species][tp]

        num_qp_agg_species[cat].append(total_num_qp)
        num_non_agg_species[cat].append(total_num_non)
        num_lowcov_agg_species[cat].append(total_num_lowcov)

prop_qp_agg_species = {'infant': [], 'mother': [], 'hmp': []}
prop_non_agg_species = {'infant': [], 'mother': [], 'hmp': []}
new_tps_ordered = {'infant': [], 'mother': [], 'hmp': []}

for cat in ['mother', 'infant', 'hmp']:
       for q, n, tp in zip(num_qp_agg_species[cat], num_non_agg_species[cat],
tps_ordered_dict[cat]):
             num_total = float(q + n)
             if num_total != 0:
                    prop_qp_agg_species[cat].append(q/num_total)
                    prop_non_agg_species[cat].append(n/num_total)
                    new_tps_ordered[cat].append(tp)

total_qp_mother_combined = 0
total_non_mother_combined = 0
total_highcov_mother_combined = 0
for tp in [0,1,2]:
    total_qp_mother_combined += num_qp_agg_species['mother'][tp]
    total_non_mother_combined += num_non_agg_species['mother'][tp]
    total_highcov_mother_combined += (num_non_agg_species['mother'][tp] +
num_qp_agg_species['mother'][tp])

prop_qp_agg_species_mother_combined =
total_qp_mother_combined/float(total_highcov_mother_combined)
prop_non_agg_species_mother_combined =
total_non_mother_combined/float(total_highcov_mother_combined)


# In[ ]:


tps_ordered_dict['mother']


# In[ ]:


# ========================================================================
# Bar/line plots comparing QP vs. non-QP proportions over timepoints
# ========================================================================

fig, ax = plt.subplots(figsize=(28,8))
ax.set_xlim((-0.5, max(new_infant_tps_ordered) + 0.5))
ax.bar(new_infant_tps_ordered, prop_qp_agg_species['infant'], label='QP',
color='orange', linewidth=0)
ax.bar(new_infant_tps_ordered, prop_non_agg_species['infant'], label='non-QP',
       bottom=prop_qp_agg_species['infant'], color='blue', linewidth=0)
ax.set_ylabel("Proportion of high coverage samples (from any cohort)\n")
ax.set_xlabel("Timepoint (days)")
ax.set_title("Proportion of QP samples by timepoint")
ax.legend()
```

```
# fig.savefig("%s/prop_qp_over_time.pdf" % config.analysis_directory,
bbox_inches='tight')

fig, ax = plt.subplots(figsize=(10,6))
ax.plot(new_infant_tps_ordered, prop_qp_agg_species['infant'])
ax.set_ylabel("Proportion of high coverage samples with are QP (from any
cohort)\n")
ax.set_xlabel("Timepoint (days)")
ax.set_title("Proportion of QP samples by timepoint")

plt.show()
# fig.savefig("%s/prop_qp_over_time_line.pdf" % config.analysis_directory,
bbox_inches='tight')


# In[ ]:


# ======================================================================
# Same as above but binned
# ======================================================================

fig, ax = plt.subplots(figsize=(18,8))

labels = new_tps_ordered['infant']

short_labels = ['birth']
for long_label in labels[1:]:
    # num, unit = long_label.split()
    unit = 'day'
    num = str(long_label)
    short_label = num + unit[0]
    short_labels.append(short_label)

xticks = np.arange(len(labels))

ax.bar(xticks, prop_qp_agg_species['infant'], label='QP', color='orange',
linewidth=0)
ax.bar(xticks, prop_non_agg_species['infant'], label='non-QP',
bottom=prop_qp_agg_species['infant'], color='blue', linewidth=0)

ax.set_xticks(np.array(xticks) + 0.5)
ax.set_xticklabels(labels)
ax.set_ylabel("Proportion of high coverage samples (from any cohort)\n")
ax.set_xlabel("Timepoint (days)")
ax.set_title("Proportion of QP samples by timepoint")
ax.legend()

plt.show()
# fig.savefig("%s/prop_qp_over_time_binned.pdf" % config.analysis_directory,
bbox_inches='tight')


# In[ ]:


# Line version

short_labels = ['M\n-3m', 'M\ndlv', 'M\n1d', 'M\n2d', 'M\n3m', 'birth']
for long_label in new_tps_ordered['infant'][1:]:
       num, unit = long_label.split()
```

```
       short_label = num + unit[0]
       short_labels.append(short_label)

mother_xticks = np.arange(len(new_tps_ordered['mother']))
xticks = np.arange(len(short_labels))
infant_xticks = np.arange(len(new_tps_ordered['mother']), len(xticks))

fig, ax = plt.subplots(figsize=(10,4))

ax.plot(np.array(mother_xticks) + 0.5, prop_qp_agg_species['mother'], 'r.-')
ax.plot(np.array(infant_xticks) + 0.5, prop_qp_agg_species['infant'], 'g.-')
ax.set_xticks(np.array(xticks) + 0.5)
ax.set_xticklabels(short_labels, fontsize=11)
ax.set_ylabel("Proportion of high coverage samples\nwhich are QP (from any
cohort)")
ax.set_xlabel("Timepoint")
ax.set_title("Proportion of QP samples by timepoint")

plt.show()
# fig.savefig("%s/prop_qp_over_time_line_binned.pdf" % config.analysis_directory,
bbox_inches='tight')


# In[ ]:


# Line version

short_labels = ['birth']
for long_label in new_tps_ordered['infant'][1:]:
    num, unit = long_label.split()
    short_label = num + unit[0]
    short_labels.append(short_label)
short_labels.append('M')
short_labels.append('A')

xticks = np.arange(len(short_labels))
infant_xticks = np.arange(len(prop_qp_agg_species['infant']))
mother_xticks = [len(infant_xticks)]
adult_xticks = [len(infant_xticks) + 1]

mother_delivery_combined = prop_qp_agg_species['mother']

fig, ax = plt.subplots(figsize=(10,4))

ax.bar(np.array(infant_xticks) + 0.5, prop_qp_agg_species['infant'],
color='#77acff')
ax.bar(np.array(mother_xticks) + 0.5, [prop_qp_agg_species_mother_combined],
color='#7bb551')
ax.bar(np.array(adult_xticks) + 0.5, prop_qp_agg_species['hmp'], color='#396651')
ax.set_xticks(np.array(xticks) + 0.5)
ax.set_xticklabels(short_labels, fontsize=11)
ax.set_ylabel("Proportion of high coverage samples\nwhich are QP (from any
cohort)")
ax.set_xlabel("Timepoint")
ax.set_title("Proportion of QP samples by timepoint")

plt.show()
fig.savefig("%s/prop_qp_over_time_bar_binned.pdf" % config.analysis_directory,
bbox_inches='tight')
```

```
# In[ ]:


# These custom cohorts are for a single sample

custom_cohorts_ordered = ['Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1',
'Mother', 'HMP (Adult)']

def get_custom_cohort(cat, sample):
    if cat == 'infant':
        day = mi_sample_day_dict[sample]
        if day >= 0 and day <= 7:
            return 'Day 0-Week 1'
        elif day <= 32:
            return 'Week 1-Month 1'
        elif day <= 366:
            return 'Month 1-Year 1'
    elif cat == 'hmp':
        return 'HMP (Adult)'
    elif cat == 'mother':
        return 'Mother'


# In[ ]:


# Final version?
# Boxplot, plot proportion of QP samples per host
# Remove 6 days
# Use timepoint categories from rates plots
# Remove title

props_qp_by_cohort = defaultdict(list)

for cat in sample_num_qp_dict:
    for sample in sample_num_qp_dict[cat]:
        custom_cohort = get_custom_cohort(cat, sample)

        # Make sure we're only including mother at delivery
        if custom_cohort == 'Mother' and (mi_sample_day_dict[sample] < -1 or
mi_sample_day_dict[sample] > 4):
                continue

        num_qp = (sample_num_qp_dict[cat][sample])
        num_non = (sample_num_non_dict[cat][sample])
        num_lowcov = (sample_num_lowcov_dict[cat][sample])
        prop_qp = num_qp/float(num_qp+num_non)

        props_qp_by_cohort[custom_cohort].append(prop_qp)

fig, ax = plt.subplots(figsize=(10, 4))

medianprops = dict(color='black')
flierprops = dict(marker='.')

data = [props_qp_by_cohort[cohort] for cohort in custom_cohorts_ordered]
boxplots = ax.boxplot(data, patch_artist=True,
                          medianprops=medianprops,
                          flierprops=flierprops)
```

```
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
boxplots['boxes'][4].set_facecolor('#396651')
boxplots['boxes'][3].set_facecolor('#7bb551')

ax.set_xticklabels(custom_cohorts_ordered, fontsize=11)
ax.set_ylabel("Proportion of high coverage species\nwhich are QP in a sample",
fontsize=12)

plt.show()
fig.savefig("%s/prop_qp_per_sample_over_time_boxplot.pdf" %
config.analysis_directory, bbox_inches='tight')


# In[ ]:


# Try survival curve version

# Define function needed for survival curve plots

import matplotlib.cm as cmx

def calculate_unnormalized_survival_from_vector(counts):
    counts = sorted(counts)
    xs = [0]
    ns = [len(counts)]
    ns_cur = len(counts)
    min_count = -1
    for count in counts:
        if count > min_count:
            ns.append(ns_cur) # Number of elements greater or equal
            xs.append(count)
            min_count = count
        ns_cur -= 1
    xs.append(xs[len(xs)-1]+1)
    ns.append(0)
    return xs, np.array(ns)

fig_snp, ax_snp = plt.subplots(figsize=(10,6))

colormap = cmx.get_cmap('viridis', 8)
colors = [colormap(x) for x in np.array([x for x in range(0,8)])/8.0]
colors = ['lightblue', 'blue', '#00f0b6', '#7bb551', '#0e750e']

modification_difference_threshold = config.modification_difference_threshold
replacement_difference_threshold = config.replacement_difference_threshold

# ax_snp.set_xscale('log')
# ax_snp.set_yscale('log')
ax_snp.set_ylabel(r'Fraction host samples $\geq$ $p$', fontsize=14)
ax_snp.set_xlabel(r'Proportion ($p$) QP', fontsize=14)

ax_snp.spines['top'].set_visible(False)
ax_snp.spines['right'].set_visible(False)
ax_snp.get_xaxis().tick_bottom()
ax_snp.get_yaxis().tick_left()

color_i = 0

ymin, ymax = 0.0001, 1.1
```

```
ax_snp.set_ylim([ymin,ymax])
ax_snp.set_xlim([0, 1.1])

for cohort in custom_cohorts_ordered:
    xs, ns =
calculate_unnormalized_survival_from_vector(props_qp_by_cohort[cohort])
    ax_snp.step(xs,ns/float(ns[0]),'-',color=colors[color_i],linewidth=1.4,
label=cohort, where='pre',zorder=4)
    color_i += 1

# Save
ax_snp.legend(loc='lower left', fontsize=14, numpoints=1, ncol=1, handlelength=1,
frameon=False)
fig_snp.savefig('%s/prop_qp_per_sample_over_time_survival.pdf' %
(config.analysis_directory),bbox_inches='tight')
fig_snp.savefig('%s/prop_qp_per_sample_over_time_survival.png' %
(config.analysis_directory),bbox_inches='tight',dpi=600)

plt.show()


# In[ ]:


# Make sure to have binned version for this
# Now replicate breakdown by species

tps_ordered_dict = {'mother': mother_tps_ordered, 'infant': infant_tps_ordered,
'hmp': ['Adult']}

num_qp_agg_tps = {cat: {} for cat in ['infant', 'mother', 'hmp']}
num_non_agg_tps = {cat: {} for cat in ['infant', 'mother', 'hmp']}
num_lowcov_agg_tps = {cat: {} for cat in ['infant', 'mother', 'hmp']}
num_total_agg_tps = {cat: {} for cat in ['infant', 'mother', 'hmp']}

for cat in ['mother', 'infant', 'hmp']:
    for species in good_species_list:

        total_num_qp, total_num_non, total_num_lowcov = 0, 0, 0

        for tp in tps_ordered_dict[cat]:
            if tp in num_qp_dict[cat][species]:
                total_num_qp += num_qp_dict[cat][species][tp]
            if tp in num_non_dict[cat][species]:
                total_num_non += num_non_dict[cat][species][tp]
            if tp in num_lowcov_dict[cat][species]:
                total_num_lowcov += num_lowcov_dict[cat][species][tp]

        num_qp_agg_tps[cat][species] = total_num_qp
        num_non_agg_tps[cat][species] = total_num_non
        num_lowcov_agg_tps[cat][species] = total_num_lowcov
        num_total_agg_tps[cat][species] =
total_num_qp+total_num_non+total_num_lowcov


# In[ ]:


# Plot infants first

from matplotlib.patches import Patch
```

```
ordered_species_list = []
for species, count in sorted(num_qp_agg_tps['infant'].items(), key=lambda x: x[1],
reverse=True):
    ordered_species_list.append(species)

ordered_species_list_subset = ordered_species_list[:30][::-1]

all_num_qp_by_cat = {}
all_num_non_by_cat = {}
all_num_lowcov_by_cat = {}

for cat in ['mother', 'infant', 'hmp']:
    all_num_qp_by_cat[cat] = np.array([num_qp_agg_tps[cat][species] for species in
ordered_species_list_subset])
    all_num_non_by_cat[cat] = np.array([num_non_agg_tps[cat][species] for species
in ordered_species_list_subset])
    all_num_lowcov_by_cat[cat] = np.array([num_lowcov_agg_tps[cat][species] for
species in ordered_species_list_subset])

fig, ax = plt.subplots(figsize=(8,12))

yticks = np.arange(len(ordered_species_list_subset))

acc = np.zeros(len(all_num_qp_by_cat['infant']))
ax.barh(np.array(yticks) + 0.5, all_num_qp_by_cat['infant'], color='#08519c'); acc
+= all_num_qp_by_cat['infant']
ax.barh(np.array(yticks) + 0.5, all_num_non_by_cat['infant'], left=acc,
color='#77acff'); acc += all_num_non_by_cat['infant']
ax.barh(np.array(yticks) + 0.5, all_num_qp_by_cat['mother'] +
all_num_qp_by_cat['hmp'], left=acc, color='#396651'); acc +=
all_num_qp_by_cat['mother'] + all_num_qp_by_cat['hmp']
ax.barh(np.array(yticks) + 0.5, all_num_non_by_cat['mother'] +
all_num_non_by_cat['hmp'], left=acc, color='#7bb551')

ax.set_yticks(np.array(yticks) + 0.5)
ax.set_yticklabels(ordered_species_list_subset, size=14)
ax.set_xlabel("Number of QP samples per species", size=16)

legend_elements = [Patch(facecolor='#08519c', label='QP (Infant)'),
Patch(facecolor='#77acff', label='non-QP (Infant)'),
                   Patch(facecolor='#396651', label='QP (Adult)'),
Patch(facecolor='#7bb551', label='non-QP (Adult)')]
ax.legend(handles=legend_elements, loc='best', frameon=False, fontsize=14)

plt.show()
fig.savefig("%s/count_qp_by_species_barh_infant_adult.pdf" %
config.analysis_directory, bbox_inches='tight')
fig.savefig("%s/count_qp_by_species_barh_infant_adult.png" %
config.analysis_directory, bbox_inches='tight', dpi=500)


# In[ ]:


# Plot infants first
# Expanded version: include all species with at least 10 QP samples

from matplotlib.patches import Patch

ordered_species_list = []
```

```
for species, count in sorted(num_qp_agg_tps['infant'].items(), key=lambda x: x[1],
reverse=True):
    ordered_species_list.append(species)

# ordered_species_list_subset = ordered_species_list[:30][::-1]

ordered_species_list_subset = []
for species in ordered_species_list[::-1]:
    total_qp_count = sum([num_qp_agg_tps[cat][species] for cat in num_qp_agg_tps])
    if total_qp_count > 10:
        ordered_species_list_subset.append(species)

print(len(ordered_species_list_subset))
if len(set(ordered_species_list_subset)) != len(ordered_species_list_subset):
    print("sus")

all_num_qp_by_cat = {}
all_num_non_by_cat = {}
all_num_lowcov_by_cat = {}

for cat in ['mother', 'infant', 'hmp']:
    all_num_qp_by_cat[cat] = np.array([num_qp_agg_tps[cat][species] for species in
ordered_species_list_subset])
    all_num_non_by_cat[cat] = np.array([num_non_agg_tps[cat][species] for species
in ordered_species_list_subset])
    all_num_lowcov_by_cat[cat] = np.array([num_lowcov_agg_tps[cat][species] for
species in ordered_species_list_subset])

fig, ax = plt.subplots(figsize=(8,40))

yticks = np.arange(len(ordered_species_list_subset))

acc = np.zeros(len(all_num_qp_by_cat['infant']))
ax.barh(np.array(yticks) + 0.5, all_num_qp_by_cat['infant'], color='#08519c')
acc += all_num_qp_by_cat['infant']
ax.barh(np.array(yticks) + 0.5, all_num_non_by_cat['infant'], left=acc,
color='#77acff')
acc += all_num_non_by_cat['infant']
ax.barh(np.array(yticks) + 0.5, all_num_qp_by_cat['mother'] +
all_num_qp_by_cat['hmp'], left=acc, color='#396651')
acc += (all_num_qp_by_cat['mother'] + all_num_qp_by_cat['hmp'])
ax.barh(np.array(yticks) + 0.5, all_num_non_by_cat['mother'] +
all_num_non_by_cat['hmp'], left=acc, color='#7bb551')

ax.set_ylim((min(yticks), max(yticks)+1))
ax.set_yticks(np.array(yticks) + 0.5)
ax.set_yticklabels(ordered_species_list_subset, size=14)
ax.set_xlabel("Number of QP samples per species", size=16)

legend_elements = [Patch(facecolor='#08519c', label='QP (Infant)'),
Patch(facecolor='#77acff', label='non-QP (Infant)'),
                   Patch(facecolor='#396651', label='QP (Adult)'),
Patch(facecolor='#7bb551', label='non-QP (Adult)')]
ax.legend(handles=legend_elements, loc='lower right', frameon=False, fontsize=14)

plt.show()
fig.savefig("%s/count_qp_by_species_barh_infant_adult.pdf" %
config.analysis_directory, bbox_inches='tight')
fig.savefig("%s/count_qp_by_species_barh_infant_adult.png" %
config.analysis_directory, bbox_inches='tight', dpi=500)
```

```
# In[ ]:


# Plot infant and HMP side by side
# Expanded version: include all species with at least 10 QP samples (this time in
either infant or HMP separately)

from matplotlib.patches import Patch

ordered_species_list_infant = []

for species, count in sorted(num_qp_agg_tps['infant'].items(), key=lambda x: x[1],
reverse=True):
    ordered_species_list_infant.append(species)

ordered_species_list_infant_subset = []
for species in ordered_species_list_infant[::-1]:
    total_qp_count = sum([num_qp_agg_tps[cat][species] for cat in ['infant']])
    if total_qp_count > 10:
        ordered_species_list_infant_subset.append(species)

ordered_species_list_adult = []

for species, count in sorted(num_qp_agg_tps['hmp'].items(), key=lambda x: x[1],
reverse=True):
    ordered_species_list_adult.append(species)

ordered_species_list_adult_subset = []
for species in ordered_species_list_adult[::-1]:
    total_qp_count = sum([num_qp_agg_tps[cat][species] for cat in ['mother',
'hmp']])
    if total_qp_count > 10:
        ordered_species_list_adult_subset.append(species)

all_num_qp_by_cat = {}
all_num_non_by_cat = {}
all_num_lowcov_by_cat = {}

for cat in ['mother', 'infant', 'hmp']:
    ordered_species_list_subset = ordered_species_list_infant_subset
    '''
    if cat == 'infant':
        ordered_species_list_subset = ordered_species_list_infant_subset
    else:
        ordered_species_list_subset = ordered_species_list_adult_subset
    '''
    all_num_qp_by_cat[cat] = np.array([num_qp_agg_tps[cat][species] for species in
ordered_species_list_subset])
    all_num_non_by_cat[cat] = np.array([num_non_agg_tps[cat][species] for species
in ordered_species_list_subset])
    all_num_lowcov_by_cat[cat] = np.array([num_lowcov_agg_tps[cat][species] for
species in ordered_species_list_subset])

fig, ax = plt.subplots(1, 2, figsize=(12,40), sharey=True)



yticks = np.arange(len(ordered_species_list_infant_subset))
acc = np.zeros(len(all_num_qp_by_cat['infant']))
```

```
ax[0].barh(np.array(yticks) + 0.5, all_num_qp_by_cat['infant'], color='#08519c');
acc += all_num_qp_by_cat['infant']
ax[0].barh(np.array(yticks) + 0.5, all_num_non_by_cat['infant'], left=acc,
color='#77acff'); acc += all_num_non_by_cat['infant']
ax[0].set_yticks(np.array(yticks) + 0.5)
ax[0].set_yticklabels(ordered_species_list_infant_subset, size=14)
ax[0].set_xlabel("Number of QP samples per species", size=16)

yticks = np.arange(len(ordered_species_list_infant_subset))
acc = np.zeros(len(all_num_qp_by_cat['hmp']))

ax[1].barh(np.array(yticks) + 0.5, all_num_qp_by_cat['mother'] +
all_num_qp_by_cat['hmp'], left=acc, color='#396651'); acc +=
all_num_qp_by_cat['mother'] + all_num_qp_by_cat['hmp']
ax[1].barh(np.array(yticks) + 0.5, all_num_non_by_cat['mother'] +
all_num_non_by_cat['hmp'], left=acc, color='#7bb551')
ax[1].set_yticks(np.array(yticks) + 0.5)
# ax[1].set_yticklabels(ordered_species_list_infant_subset, size=14)
ax[1].set_xlabel("Number of QP samples per species", size=16)

legend_elements = [Patch(facecolor='#08519c', label='QP (Infant)'),
Patch(facecolor='#77acff', label='non-QP (Infant)'),
                   Patch(facecolor='#396651', label='QP (Adult)'),
Patch(facecolor='#7bb551', label='non-QP (Adult)')]
ax[0].legend(handles=legend_elements, loc='best', frameon=False, fontsize=14)

plt.show()
# fig.savefig("%s/count_qp_by_species_barh_infant_adult.pdf" %
config.analysis_directory, bbox_inches='tight')
# fig.savefig("%s/count_qp_by_species_barh_infant_adult.png" %
config.analysis_directory, bbox_inches='tight', dpi=500)


# In[ ]:


# Split by life stage
# Match colors in PLOS figure 1f
# Do combined version with HMP, mother at delivery, and infants;
# have two tones of yellow where one is baby, one is adult, same for non-QP


# In[ ]:


# Might there be a relationship between proportion QP samples and abundance?
```

plot_qp_pairs.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


# Basically replicate pickle_everything but track info about how many QP pairs are
considered


# In[9]:
```

```
from utils import sample_utils as su, parse_midas_data, substitution_rates_utils,
config, temporal_changes_utils, snps_utils, core_gene_utils, gene_diversity_utils
import numpy as np
from numpy.random import choice, random as np_random, randint
import random
from collections import defaultdict
import pickle
import os, sys

# ====================================================
# Examines all consecutive timepoint pairs within hosts
# across all cohorts, and pickles SNP/gene change info
# ====================================================

# Parameters
sweep_type = 'full' # assume full for now
pp_prev_cohort = 'all'
min_coverage = 0
thresholds = {'full': (0.2, 0.8), 'partial': (0.35, 0.65)}
lower_threshold, upper_threshold = thresholds[sweep_type]

clade_divergence_threshold = 3e-02 # TODO: change to top level clade definition
later

min_sample_size = 3
variant_types = ['1D','4D']
within_host_type = 'nonconsecutive' # consecutive timepoints
min_snp_change_sample_size = 5

# For partitioning SNVs according to prevalence
derived_freq_bins = np.array([-1,0,0.01,0.1,0.5,0.9,0.99,1,2])
derived_virtual_freqs = np.arange(0,len(derived_freq_bins)-1)
derived_virtual_xticks = list(derived_virtual_freqs[:-1]+0.5)
derived_virtual_xticklabels = ['0','.01','.1','.5','.9','.99','1']

# For partitioning genes into different prevalence classes
gene_freq_bins = np.array([-1,0.1,0.5,0.9,2])
gene_freq_xticks                  = [-4, -3,   -2,          -1,          0,     1,
              2,          3, 4]
gene_freq_xticklabels = ['0','0.1','0.5', '0.9','1','0.9','0.5', '0.1','0']
gene_gain_virtual_freqs = np.array([3.5,2.5,1.5,0.5])
gene_loss_virtual_freqs = np.array([-3.5,-2.5,-1.5,-0.5])

# Sample-subject-order maps
sys.stderr.write("Loading sample metadata...\n")
subject_sample_map = su.parse_subject_sample_map()
sample_order_map = su.parse_sample_order_map()
sample_subject_map = su.parse_sample_subject_map()
sys.stderr.write("Done!\n")

# Timepoint pair types
tp_pair_names = ['MM', 'MI', 'II', 'AA']

# Cohorts
cohorts = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
mi_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

# Prevalence cohorts
prev_cohorts = ['all', 'hmp', 'infant', 'nonpremie', 'mother']
```

```
# Samples for each cohort
samples = {cohort: su.get_sample_names(cohort) for cohort in cohorts}
hmp_samples = su.get_sample_names('hmp')
olm_samples = su.get_sample_names('olm')
mother_samples = [sample for sample in su.get_sample_names('mother') if sample not
in olm_samples]
infant_samples = [sample for sample in su.get_sample_names('infant') if sample not
in olm_samples]

# Species list
good_species_list = parse_midas_data.load_pickled_good_species_list()


# In[14]:


mi_sample_day_dict = su.get_mi_sample_day_dict()


# In[7]:


species_cohort_qp_pair_count = defaultdict(dict)
species_qp_pair_dict = defaultdict(list)
num_species_enough_haploid = 0

for species_name in good_species_list[::-1]:

      sys.stderr.write("\nProcessing %s...\n" % species_name)

      # Grab QP samples for this species
      qp_sample_lists = {}
      for cohort in cohorts:
            qp_sample_lists[cohort] = sorted(su.load_qp_samples(samples[cohort],
species_name, prev_cohort=pp_prev_cohort)['qp'])

      combined_qp_samples = sorted(su.flatten([qp_sample_lists[cohort] for cohort
in cohorts]))
      combined_sample_name_idx_map = {combined_qp_samples[i] : i for i in
range(len(combined_qp_samples))}

      # Using all QP samples to threshold on sample size
      if len(combined_qp_samples) < min_sample_size:
            sys.stderr.write("Not enough haploid samples!\n")
            continue

      num_species_enough_haploid += 1

      # Loop over different cohorts
      for cohort in cohorts:
            desired_samples = qp_sample_lists[cohort]

            # These indices are w.r.t. desired_samples
            same_subject_idxs =
su.calculate_mi_ordered_same_subject_pairs(sample_order_map, desired_samples,
within_host_type=within_host_type, one_per_mi_pair=False)
            for i, j in zip(same_subject_idxs[0], same_subject_idxs[1]):
                  sample_i = desired_samples[i]
                  sample_j = desired_samples[j]
                  species_qp_pair_dict[species_name].append((sample_i, sample_j))
```

```
            species_cohort_qp_pair_count[species_name][cohort] =
len(same_subject_idxs[0])


# In[3]:


cohort_qp_pair_counts = defaultdict(int)
for species in species_cohort_qp_pair_count:
    for cohort in species_cohort_qp_pair_count[species]:
        qp_count = species_cohort_qp_pair_count[species][cohort]
        cohort_qp_pair_counts[cohort] += qp_count


# In[4]:


print("Number of QP sample pairs per cohort:")
print(cohort_qp_pair_counts)


# In[5]:


mi_qp_pair_count = 0
for cohort in mi_cohorts:
    mi_qp_pair_count += cohort_qp_pair_counts[cohort]

print("Number of QP sample pairs for all mother-infant: %i" % mi_qp_pair_count)


# In[6]:


cohort_qp_pair_counts


# In[11]:


print("There are %i out of %i 'good' species which additionally have at least %i
haploid samples" % (num_species_enough_haploid,

len(good_species_list),

min_sample_size))
print("Recall definition of good species: must have at least 10 samples with marker
gene coverage >= 10")


# In[21]:


mi_tt_qp_pair_count = 0
mi_deliv_tt_qp_pair_count = 0
ii_tt_qp_pair_count = 0

for species in species_qp_pair_dict:
    for sample_i, sample_j in species_qp_pair_dict[species]:
        if sample_i in mother_samples and sample_j in infant_samples:
            mi_tt_qp_pair_count += 1
```

```
            if mi_sample_day_dict[sample_i] >= -1 and mi_sample_day_dict[sample_i]
< 6:
                mi_deliv_tt_qp_pair_count += 1
        if sample_i in infant_samples and sample_j in infant_samples:
            ii_tt_qp_pair_count += 1
        if sample_i in infant_samples and sample_j in mother_samples:
            print("Weird")


# In[22]:


print("Number of QP sample pairs for all mother-infant: %i" % mi_tt_qp_pair_count)
print("Number of QP sample pairs for all mother-infant (mother at delivery
only): %i" % mi_deliv_tt_qp_pair_count)
print("Number of QP sample pairs for all infant-infant: %i" % ii_tt_qp_pair_count)


# In[ ]:


# Plot QP pairs per species
# Plot infants first

from matplotlib.patches import Patch

cohort = 'hmp'

for species in species_cohort_qp_pair_count:

ordered_species_list = []
for species, count in sorted(num_qp_agg_tps[cat].items(), key=lambda x: x[1],
reverse=True):
    ordered_species_list.append(species)

ordered_species_list_subset = ordered_species_list[:40][::-1]

all_num_qp = [species_cohort_qp_pair_count[cat][species] for species in
ordered_species_list_subset]
all_num_non = [num_non_agg_tps[cat][species] for species in
ordered_species_list_subset]
all_num_lowcov = [num_lowcov_agg_tps[cat][species] for species in
ordered_species_list_subset]

fig, ax = plt.subplots(figsize=(8,12))

yticks = np.arange(len(all_num_qp))

ax.barh(np.array(yticks) + 0.5, all_num_qp, color='orange')
ax.barh(np.array(yticks) + 0.5, all_num_non, left=all_num_qp, color='#77acff')
# ax.barh(np.array(yticks) + 0.5, all_num_lowcov,
left=np.array(all_num_qp)+np.array(all_num_non), color='#396651')

ax.set_yticks(np.array(yticks) + 0.5)
ax.set_yticklabels(ordered_species_list_subset)
ax.set_title("Number of QP samples per species")

legend_elements = [Patch(facecolor='orange', label='QP'),
Patch(facecolor='#77acff', label='non-QP')]
ax.legend(handles=legend_elements, loc='center right', frameon=False)
```

```
plt.show()
fig.savefig("%s/count_qp_by_species_barh.pdf" % config.analysis_directory,
bbox_inches='tight')
fig.savefig("%s/count_qp_by_species_barh.png" % config.analysis_directory,
bbox_inches='tight', dpi=500)
```

plot_revisions.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import matplotlib
from matplotlib.patches import Patch
from matplotlib import pyplot as plt
import plot_utils
import config, sample_utils as su, parse_midas_data as pmd
from collections import defaultdict
import bz2, sys
import numpy as np
import math
import scipy
from scipy import stats
import pickle


# In[2]:


# Pickle directory
pickle_dir = "%s/pickles" % config.data_directory

# Plot directory
plot_dir = "%s/" % (config.analysis_directory)

# Load pickles
sample_species_polymorphism_dict = pickle.load(open("%s/sample_species_polymorphism_dict.pkl" %
(pickle_dir), 'rb'),
                                               encoding='latin1')


# In[3]:


# Load subject and sample metadata
sample_order_map = su.parse_sample_order_map()
subject_sample_map = su.parse_subject_sample_map()
sample_subject_map = su.parse_sample_subject_map()
sample_cohort_map = su.parse_sample_cohort_map()
subject_feeding_mode_map = su.parse_subject_feeding_mode_map()
subject_delivery_mode_map = su.parse_subject_delivery_mode_map()

# Bootleg load HMP female subjects
f = open('%s/HMP1-2_female_subjects.txt' % config.metadata_directory, 'r')
hmp_female_subjects = [line.strip() for line in f]

all_samples = su.get_sample_names('all') # Note: c's removed
hmp_samples = su.get_sample_names('hmp')
```

```
olm_samples = su.get_sample_names('olm')
mother_samples_orig = su.get_sample_names('mother')
infant_samples = [sample for sample in su.get_sample_names('infant') if sample not in
olm_samples]

mi_tp_sample_dict, infant_tps_ordered = su.get_mi_tp_sample_dict(exclude_cohorts = ['olm'],
binned = True)
mother_tps_ordered = sorted(mi_tp_sample_dict['mother'].keys())
tps_ordered_dict = {'mother': mother_tps_ordered, 'infant': infant_tps_ordered}
mi_sample_day_dict = su.get_mi_sample_day_dict(exclude_cohorts=['olm'])

# Remove -92 and 92
mother_tps_ordered.remove(-92)
mother_tps_ordered.remove(92)
mother_samples = []
for sample in mother_samples_orig:
    if mi_sample_day_dict[sample] == -92 or mi_sample_day_dict[sample] == 92:
        continue
    mother_samples.append(sample)

# Species list
species_list = pmd.parse_species_list()
good_species_list = pmd.load_pickled_good_species_list()

# Qin metadata
qin_sample_subject_map = {}
qin_subject_sample_map = {}
qin_samples = []

qin_ids_fpath = "%s/qin_ids.txt" % (config.metadata_directory)
with open(qin_ids_fpath, 'r') as qin_file:
    header = qin_file.readline()
    for line in qin_file:
        subject, _, sample, _, _, _ = line.strip().split('\t')
        qin_sample_subject_map[sample] = subject
        qin_subject_sample_map[subject] = sample
        qin_samples.append(sample)

# Bootleg load Qin gender
f = open('%s/Qin_metadata.txt' % config.metadata_directory)
header = f.readline()
qin_sample_gender_dict = {}
for line in f:
    subject, gender, age = line.strip().split('\t')
    subject = subject.replace(' ', '-')
    age = int(age)
    if subject in qin_subject_sample_map:
        sample = qin_subject_sample_map[subject]
        qin_sample_gender_dict[sample] = gender

# Utility functions
def round_down(num, divisor):
    return num - (num % divisor)

# Plot function
def plot_interval(y, xstart, xstop, color='b', tickh=0.1):
    """Plot interval at y from xstart to xstop with given color."""
    plt.hlines(y, xstart, xstop, color, lw=1)
    plt.vlines(xstart, y+tickh, y-tickh, color, lw=1)
    plt.vlines(xstop, y+tickh, y-tickh, color, lw=1)
```

```
# Plot function on specific ax
def plot_interval_on_ax(ax, y, xstart, xstop, color='b', tickh=0.1):
    """Plot interval at y from xstart to xstop with given color."""
    ax.hlines(y, xstart, xstop, color, lw=1)
    ax.vlines(xstart, y+tickh, y-tickh, color, lw=1)
    ax.vlines(xstop, y+tickh, y-tickh, color, lw=1)

# function to calculate Cohen's d for independent samples
def cohenD(d1, d2):
    n1, n2 = len(d1), len(d2)
    s1, s2 = np.var(d1, ddof=1), np.var(d2, ddof=1)
    s = np.sqrt(((n1 - 1) * s1 + (n2 - 1) * s2) / (n1 + n2 - 2))
    u1, u2 = np.mean(d1), np.mean(d2)
    return (u1 - u2) / s

def summarize_ttest(a, b, simple=False):
    t, p = scipy.stats.ttest_ind(a, b)
    if simple:
        print("t=%.04f" % t)
        print("P=" + str(p))
        print("d=" + str(cohenD(a, b)))
    else:
        print("Group 1 size: %i | Group 2 size: %i" % (len(a), len(b)))
        print("T-statistic: %.04f" % t)
        print("P-value: " + str(p))
        print("Cohen's D: " + str(cohenD(a, b)))

def MWU_effect_size(p, n1, n2):
    return stats.norm.isf(p/2.0)/np.sqrt(n1 + n2)

def summarize_utest(a, b, simple=False):
    U, p = scipy.stats.mannwhitneyu(a, b)
    if simple:
        print("U=%.04f" % U)
        print("P=" + str(p))
        print("Effect size: " + str(MWU_effect_size(p, len(a), len(b))))
    else:
        print("Group 1 size: %i | Group 2 size: %i" % (len(a), len(b)))
        print("U-statistic: %.04f" % U)
        print("P-value: " + str(p))
        print("Effect size: " + str(MWU_effect_size(p, len(a), len(b))))


# In[4]:


# Alpha diversity at different timepoints

# Relative abundance file
relab_fpath = "%s/species/relative_abundance.txt.bz2" % (config.data_directory)
relab_file = bz2.open(relab_fpath, 'rt')
data = [line.split('\t') for line in relab_file]
data.pop() # Get rid of extra element due to terminal newline
header = su.parse_merged_sample_names(data[0])

# Generate alpha diversity dictionary
alpha_div_dict = {}
richness_dict = {}
for i in range(1, len(header)):
    acc = 0
    richness = 0
```

```
    for row in data[1:]:
        rel_ab = float(row[i])
        if rel_ab != 0:
            acc += (rel_ab * math.log(rel_ab))
            richness += 1
    alpha_div_dict[header[i]] = (acc*-1)
    richness_dict[header[i]] = richness


# In[5]:


# Poyet, Korpela, Qin adult data too

korpela_data_dir = '/u/home/d/daisyche/dbd/data_korpela/'
relab_fpath = "%s/species/relative_abundance.txt.bz2" % (korpela_data_dir)
relab_file = bz2.open(relab_fpath, 'rt')
data = [line.split('\t') for line in relab_file]
header = su.parse_merged_sample_names(data[0])

# Generate alpha diversity dictionary
korpela_alpha_div_dict = {}
korpela_richness_dict = {}
for i in range(1, len(header)):
    acc = 0
    richness = 0
    for row in data[1:]:
        rel_ab = float(row[i])
        if rel_ab != 0:
            acc += (rel_ab * math.log(rel_ab))
            richness += 1
    korpela_alpha_div_dict[header[i]] = (acc*-1)
    korpela_richness_dict[header[i]] = richness

poyet_data_dir = '/u/home/d/daisyche/dbd/data_poyet/'
relab_fpath = "%s/species/relative_abundance.txt.bz2" % (poyet_data_dir)
relab_file = bz2.open(relab_fpath, 'rt')
data = [line.split('\t') for line in relab_file]
header = su.parse_merged_sample_names(data[0])

# Generate alpha diversity dictionary
poyet_alpha_div_dict = {}
poyet_richness_dict = {}
for i in range(1, len(header)):
    acc = 0
    richness = 0
    for row in data[1:]:
        rel_ab = float(row[i])
        if rel_ab != 0:
            acc += (rel_ab * math.log(rel_ab))
            richness += 1
    poyet_alpha_div_dict[header[i]] = (acc*-1)
    poyet_richness_dict[header[i]] = richness

qin_data_dir = '/u/home/d/daisyche/mother_infant/Qin_species/'
relab_fpath = "%s/relative_abundance.txt.bz2" % (qin_data_dir)
relab_file = bz2.open(relab_fpath, 'rt')
data = [line.split('\t') for line in relab_file]
header = su.parse_merged_sample_names(data[0])

# Generate alpha diversity dictionary
```

```
qin_alpha_div_dict = {}
qin_richness_dict = {}
for i in range(1, len(header)):
    sample = header[i]
    if sample not in qin_sample_subject_map:
        continue
    acc = 0
    richness = 0
    for row in data[1:]:
        rel_ab = float(row[i])
        if rel_ab != 0:
            acc += (rel_ab * math.log(rel_ab))
            richness += 1
    qin_alpha_div_dict[sample] = (acc*-1)
    qin_richness_dict[sample] = richness


# In[6]:


# Load Qin polymorphism data and add to dictionary

f = open('%s/Qin_polymorphism.csv' % config.data_directory)
samples = f.readline().strip().split(',')[1:]
for line in f:
    items = line.strip().split(',')
    species = items[0]
    for sample, val in zip(samples, items[1:]):
        if val != '':
            sample_species_polymorphism_dict[sample][species] = float(val)
f.close()


# In[7]:


# Plot alpha diversity over time for each cohort

fig, ax = plt.subplots(2,2, figsize=(10,7), sharey=True, constrained_layout=True)

boxprops = dict(color='black')
medianprops = dict(color='black')
flierprops = dict(marker='.')

# HMP1-2
hmp_samples = su.get_sample_names('HMP')
order_alpha_div_dict = defaultdict(list)
order_subjects_dict = defaultdict(set)
for sample in hmp_samples:
    subject, order = sample_order_map[sample]
    if subject in order_subjects_dict[order]: # Ensure one sample per subject for particular
timepoint bin
        continue
    order_subjects_dict[order].add(subject)
    alpha_div = alpha_div_dict[sample]
    order_alpha_div_dict[order].append(alpha_div)

order_alpha_div_dict['Qin'] = list(qin_alpha_div_dict.values())

order_list = [1, 2, 3, 'Qin']
order_labels = ['HMP1-2:1', 'HMP1-2:2', 'HMP1-2:3', 'Qin']
```

```
order_labels_with_n = [label + '\n(n=%i)' % len(order_alpha_div_dict[o]) for label, o in
zip(order_labels, order_list)]
plot_data = [order_alpha_div_dict[o] for o in order_list]

boxplots = ax[0][0].boxplot(plot_data, patch_artist=True,
                boxprops=boxprops,
                medianprops=medianprops,
                flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
ax[0][0].set_xticklabels(order_labels_with_n)
ax[0][0].set_ylabel("Shannon alpha diversity")
ax[0][0].set_title("HMP and Qin", weight='bold')

# Backhed
backhed_samples = su.get_sample_names('Backhed')
order_alpha_div_dict = defaultdict(list)
order_subjects_dict = defaultdict(set)
for sample in backhed_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples_orig)
    subject, order = sample_order_map[sample]
    if subject in order_subjects_dict[tp]: # Ensure one sample per subject for particular
timepoint bin
        continue
    order_subjects_dict[tp].add(subject)
    alpha_div = alpha_div_dict[sample]
    order_alpha_div_dict[tp].append(alpha_div)

order_list = ['M1', 'I1', 'I2', 'I3']
order_labels = ['Mother', 'Birth (2-5 days)', '4 Months', '12 Months']
order_labels_with_n = [label + '\n(n=%i)' % len(order_alpha_div_dict[o]) for label, o in
zip(order_labels, order_list)]
plot_data = [order_alpha_div_dict[o] for o in order_list]

boxplots = ax[0][1].boxplot(plot_data, patch_artist=True,
                boxprops=boxprops,
                medianprops=medianprops,
                flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
ax[0][1].set_xticklabels(order_labels_with_n)
ax[0][1].set_title("Backhed", weight='bold')

# Ferretti
ferretti_samples = su.get_sample_names('Ferretti')
order_alpha_div_dict = defaultdict(list)
order_subjects_dict = defaultdict(set)
for sample in ferretti_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples_orig)
    subject, order = sample_order_map[sample]
    if subject in order_subjects_dict[tp]: # Ensure one sample per subject for particular
timepoint bin
        continue
    order_subjects_dict[tp].add(subject)
    alpha_div = alpha_div_dict[sample]
    order_alpha_div_dict[tp].append(alpha_div)

order_list = ['M1', 'I1', 'I2', 'I3', 'I4', 'I5']
order_labels = ['Mother', '1 day', '3 days', '1 wk', '1 mon', '4 mon']
order_labels_with_n = [label + '\n(n=%i)' % len(order_alpha_div_dict[o]) for label, o in
zip(order_labels, order_list)]
```

```
plot_data = [order_alpha_div_dict[o] for o in order_list]

boxplots = ax[1][0].boxplot(plot_data, patch_artist=True,
                 boxprops=boxprops,
                 medianprops=medianprops,
                 flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
ax[1][0].set_xticklabels(order_labels_with_n)
ax[1][0].set_ylabel("Shannon alpha diversity")
ax[1][0].set_title("Ferretti", weight='bold')

# Yassour
yassour_samples = su.get_sample_names('Yassour')
order_alpha_div_dict = defaultdict(list)
order_subjects_dict = defaultdict(set)
for sample in yassour_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples_orig)
    subject, order = sample_order_map[sample]
    if subject in order_subjects_dict[tp]: # Ensure one sample per subject for particular
timepoint bin
        continue
    order_subjects_dict[tp].add(subject)
    alpha_div = alpha_div_dict[sample]
    order_alpha_div_dict[tp].append(alpha_div)

order_list = ['M2', 'I1', 'I2', 'I3', 'I4', 'I5'] # Not including M1, M3
order_labels = ['Mother:\nGest', 'Mother:\nDelivery','Mother:\n3 mon', 'Birth', '1 wk', '2 wk',
'1 mon', '2 mon', '3 mon']
order_labels_with_n = [label + '\n(n=%i)' % len(order_alpha_div_dict[o]) for label, o in
zip(order_labels, order_list)]
plot_data = [order_alpha_div_dict[o] for o in order_list]

boxplots = ax[1][1].boxplot(plot_data, patch_artist=True,
                 boxprops=boxprops,
                 medianprops=medianprops,
                 flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
ax[1][1].set_xticklabels(order_labels_with_n)
ax[1][1].set_title("Yassour", weight='bold')

fig.savefig('%s/alpha_div_by_cohort_except_shao.pdf' % (config.analysis_directory),
bbox_inches='tight')
plt.show()


# In[8]:


# Compare timepoints

# Ferretti
ferretti_samples = su.get_sample_names('Ferretti')
order_alpha_div_dict = defaultdict(list)
for sample in ferretti_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples_orig)
    alpha_div = alpha_div_dict[sample]
    order_alpha_div_dict[tp].append(alpha_div)

order_list = ['M1', 'I1', 'I2', 'I3', 'I4', 'I5']
```

```
order_labels = ['Mother', '1 day', '3 days', '1 wk', '1 mon', '4 mon']

a = order_alpha_div_dict['I1']; b = order_alpha_div_dict['I2']
summarize_ttest(a, b)

# Backhed
backhed_samples = su.get_sample_names('Backhed')
order_alpha_div_dict = defaultdict(list)
for sample in backhed_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples_orig)
    alpha_div = alpha_div_dict[sample]
    order_alpha_div_dict[tp].append(alpha_div)

order_list = ['M1', 'I1', 'I2', 'I3']
order_labels = ['Mother', 'Birth (2-5 days)', '4 Months', '12 Months']

a = order_alpha_div_dict['I1']; b = order_alpha_div_dict['I3']
summarize_ttest(a, b)


# In[9]:


fig, ax = plt.subplots(figsize=(10, 3.6), sharey=True, constrained_layout=True)

boxprops = dict(color='black')
medianprops = dict(color='black')
flierprops = dict(marker='.')

# Shao
month_bins = np.arange(4, 15) * 30.5
order_bins = [0, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 21] + list(month_bins)

shao_samples = su.get_sample_names('Shao')
order_alpha_div_dict = defaultdict(list)
order_subjects_dict = defaultdict(set)

for sample in shao_samples:
    subject, order = sample_order_map[sample]
    if subject[-1] == 'M': # Mother
        tp = 0
    elif order > 21: # Convert to month bins
        tp = round_down(order, 30.5)
    else:
        tp = order

    if subject in order_subjects_dict[tp]: # Ensure one sample per subject for particular
timepoint bin
        continue
    order_subjects_dict[tp].add(subject)
    alpha_div = alpha_div_dict[sample]
    order_alpha_div_dict[tp].append(alpha_div)

plot_data = [order_alpha_div_dict[o] for o in order_bins if len(order_alpha_div_dict[o]) > 5]
order_labels = ['Mother', '4d', '6d', '7d', '8d', '9d', '10d', '11d', '12d', '13d', '14d',
'17d', '18d', '21d', '4m', '5m', '6m', '7m', '8m', '9m', '10m', '11m', '12m', '13m', '14m']
order_labels_with_n = [label + '\n(n=%i)' % len(order_alpha_div_dict[o]) for label, o in
zip(order_labels, order_bins) if len(order_alpha_div_dict[o]) > 5]

boxplots = ax.boxplot(plot_data, patch_artist=True,
                  boxprops=boxprops,
```

```
                medianprops=medianprops,
                flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

ax.set_xticklabels(order_labels_with_n)
ax.set_ylabel("Shannon alpha diversity")
ax.set_title("Shao", weight='bold')

summarize_ttest(order_alpha_div_dict[4], order_alpha_div_dict[335.5] + order_alpha_div_dict[366]
+ order_alpha_div_dict[305] + order_alpha_div_dict[396.5])

fig.savefig('%s/alpha_div_by_cohort_shao.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[10]:


# Store alpha diversity CSV for GLMM analysis

f = open('%s/alpha_div.csv' % config.analysis_directory, 'w')
f.write(','.join(['sample', 'subject', 'day', 'cohort', 'alpha_div', 'breast', 'formula',
'vaginal', 'csection']) + '\n')

for sample in infant_samples:
    cohort = sample_cohort_map[sample]
    subject = sample_subject_map[sample]
    day = mi_sample_day_dict[sample]
    alpha_div = alpha_div_dict[sample]

    feeding_mode = subject_feeding_mode_map[subject] if subject in subject_feeding_mode_map else
'NA'
    if feeding_mode == 'breast':
        breast = 1; formula = 0
    elif feeding_mode == 'mixed':
        breast = 1; formula = 1
    elif feeding_mode == 'formula':
        breast = 0; formula = 1
    else:
        breast = 'NA'; formula = 'NA'

    delivery_mode = subject_delivery_mode_map[subject] if subject in subject_delivery_mode_map
else 'NA'
    if delivery_mode == 'Vaginal':
        vaginal = 1; csection = 0
    elif delivery_mode == 'C-section':
        vaginal = 0; csection = 1
    else:
        vaginal = 'NA'; csection = 'NA'

    f.write(','.join([str(val) for val in [sample, subject, day, cohort, alpha_div, breast,
formula, vaginal, csection]]) + '\n')

f.close()


# In[11]:


# Compare C section and vaginal delivery
```

```
subject_delivery_mode_map = su.parse_subject_delivery_mode_map()

vaginal_days = []
vaginal_alpha_divs = []
csection_days = []
csection_alpha_divs = []

for sample in infant_samples:
    day = mi_sample_day_dict[sample]
    subject = sample_subject_map[sample]
    delivery_mode = subject_delivery_mode_map[subject]
    alpha_div = alpha_div_dict[sample]

    if delivery_mode == 'Vaginal':
        vaginal_days.append(day); vaginal_alpha_divs.append(alpha_div)
    elif delivery_mode == 'C-section':
        csection_days.append(day); csection_alpha_divs.append(alpha_div)

fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(vaginal_days, vaginal_alpha_divs, 'b.', label='Vaginal')
ax.plot(csection_days, csection_alpha_divs, 'g.', label='C-section')
# ax.set_xscale('log')
ax.set_ylabel("Shannon alpha diversity")
ax.set_xlabel('Days after birth')
ax.legend()
plt.show()


# In[9]:


# Compare feeding modes
subject_feeding_mode_map = su.parse_subject_feeding_mode_map()

days_by_category = defaultdict(list)
alpha_divs_by_category = defaultdict(list)

cats = ['breast', 'mixed', 'formula']

for sample in infant_samples:
    day = mi_sample_day_dict[sample]
    subject = sample_subject_map[sample]

    if subject in subject_feeding_mode_map:
        feeding_mode = subject_feeding_mode_map[subject]
        alpha_div = alpha_div_dict[sample]
        days_by_category[feeding_mode].append(day)
        alpha_divs_by_category[feeding_mode].append(alpha_div)

fig, ax = plt.subplots(figsize=(10, 4))

for cat in cats:
    ax.plot(days_by_category[cat], alpha_divs_by_category[cat], '.', label=cat)

# ax.set_xscale('log')
ax.set_ylabel("Shannon alpha diversity")
ax.set_xlabel('Days after birth')
ax.legend()
plt.show()
```

```
# In[16]:


fig, ax = plt.subplots(figsize=(20,4))

infant_tps = [tp for tp in tps_ordered_dict['infant']]
infant_tps.remove('6 day')
all_tps = infant_tps

# Alpha diversity ========================================
alpha_divs = [] # list of sample values for each tp
labels = []

for tp in infant_tps:
    num_samples = len(mi_tp_sample_dict['infant'][tp])

    vaginal_samples = []; csection_samples = []
    for sample in mi_tp_sample_dict['infant'][tp]:
        delivery_mode = subject_delivery_mode_map[sample_subject_map[sample]]
        if delivery_mode == 'Vaginal':
            vaginal_samples.append(sample)
        elif delivery_mode == 'C-section':
            csection_samples.append(sample)

    if tp != 'birth':
        num, unit = tp.split(' ')
        tp = num+unit[0]
    alpha_divs.append([alpha_div_dict[sample] for sample in vaginal_samples])
    labels.append(tp + "\n" + ("n=%i" % len(vaginal_samples)))
    alpha_divs.append([alpha_div_dict[sample] for sample in csection_samples])
    labels.append(tp + "\n" + ("n=%i" % len(csection_samples)))

boxprops = dict(color='#77acff')
medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(alpha_divs, patch_artist=True,
                          medianprops=medianprops,
                          flierprops=flierprops)

for i in np.arange(len(alpha_divs), step=2):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Vaginal
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # C-section

legend_elements = [Patch(facecolor=plot_utils.col_blue, label='Vaginal'),
Patch(facecolor=plot_utils.col_orange, label='C-section')]
ax.legend(handles=legend_elements, loc='upper left', frameon=False)

ax.set_ylabel("Shannon\nalpha diversity", fontsize=14)
ax.set_xticklabels(labels)


# In[21]:


fig, ax = plt.subplots(figsize=(24,4))

infant_tps = [tp for tp in tps_ordered_dict['infant']]
infant_tps.remove('6 day')
all_tps = infant_tps
```

```
cats = ['breast', 'mixed', 'formula']

# Alpha diversity ========================================
alpha_divs = [] # list of sample values for each tp
labels = []

for tp in infant_tps:
    num_samples = len(mi_tp_sample_dict['infant'][tp])

    samples_by_category = defaultdict(list)
    for sample in mi_tp_sample_dict['infant'][tp]:
        subject = sample_subject_map[sample]
        if subject in subject_feeding_mode_map:
            feeding_mode = subject_feeding_mode_map[subject]
            samples_by_category[feeding_mode].append(sample)

    if tp != 'birth':
        num, unit = tp.split(' ')
        tp = num+unit[0]

    for cat in cats:
        alpha_divs.append([alpha_div_dict[sample] for sample in samples_by_category[cat]])
        labels.append(tp + "\n" + ("n=%i" % len(samples_by_category[cat])))

boxprops = dict(color='#77acff'); medianprops = dict(color='black'); flierprops =
dict(marker='.')
boxplots = ax.boxplot(alpha_divs, patch_artist=True, medianprops=medianprops,
flierprops=flierprops)

for i in np.arange(len(alpha_divs), step=3):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Breast
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # Mixed
    boxplots['boxes'][i+2].set_facecolor(plot_utils.col_darkgreen) # Formula

legend_elements = [Patch(facecolor=plot_utils.col_blue, label='Breast'),
                   Patch(facecolor=plot_utils.col_orange, label='Mixed'),
                   Patch(facecolor=plot_utils.col_darkgreen, label='Formula')]
ax.legend(handles=legend_elements, loc='upper left', frameon=False)

ax.set_ylabel("Shannon\nalpha diversity", fontsize=14)
ax.set_xticklabels(labels)


# In[23]:


# quick overview of feeding mode info

cohort_cat_dict = {cohort: defaultdict(int) for cohort in ['backhed', 'ferretti', 'yassour',
'shao']}

for sample in infant_samples:
    cohort = sample_cohort_map[sample]
    subject = sample_subject_map[sample]
    if subject in subject_feeding_mode_map:
        cat = subject_feeding_mode_map[subject]
        cohort_cat_dict[cohort][cat] += 1

cohort_cat_dict
```

```
# In[25]:


fig, ax = plt.subplots(figsize=(24,4))

infant_tps = [tp for tp in tps_ordered_dict['infant']]
infant_tps.remove('6 day')
all_tps = infant_tps

cats = ['breast', 'mixed', 'formula']

# Polymorphism for E. coli ================================
desired_species = 'Escherichia_coli_58110'
polymorphisms = [] # list of sample values for each tp
labels = []

for tp in infant_tps:
    num_samples = len(mi_tp_sample_dict['infant'][tp])

    samples_by_category = defaultdict(list)
    for sample in mi_tp_sample_dict['infant'][tp]:
        subject = sample_subject_map[sample]
        if desired_species in sample_species_polymorphism_dict[sample] and subject in
subject_feeding_mode_map:
            feeding_mode = subject_feeding_mode_map[subject]
            samples_by_category[feeding_mode].append(sample)

    if tp != 'birth':
        num, unit = tp.split(' ')
        tp = num+unit[0]

    for cat in cats:
        polymorphisms.append([sample_species_polymorphism_dict[sample][desired_species] for
sample in samples_by_category[cat]])
        labels.append(tp + "\n" + ("n=%i" % len(samples_by_category[cat])))

boxprops = dict(color='#77acff'); medianprops = dict(color='black'); flierprops =
dict(marker='.')
boxplots = ax.boxplot(polymorphisms, patch_artist=True, medianprops=medianprops,
flierprops=flierprops)

for i in np.arange(len(alpha_divs), step=3):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Breast
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # Mixed
    boxplots['boxes'][i+2].set_facecolor(plot_utils.col_darkgreen) # Formula

legend_elements = [Patch(facecolor=plot_utils.col_blue, label='Breast'),
                   Patch(facecolor=plot_utils.col_orange, label='Mixed'),
                   Patch(facecolor=plot_utils.col_darkgreen, label='Formula')]
ax.legend(handles=legend_elements, loc='upper left', frameon=False)

ax.set_ylabel("Polymorphism", fontsize=14)
ax.set_xticklabels(labels)
ax.set_yscale('log')


# In[ ]:


fig, ax = plt.subplots(figsize=(20,4))
```

```
infant_tps = [tp for tp in tps_ordered_dict['infant']]
infant_tps.remove('6 day')
all_tps = infant_tps

# Polymorphism for E. coli ================================
desired_species = 'Escherichia_coli_58110'
polymorphisms = [] # list of sample values for each tp
labels = []

for tp in infant_tps:
    num_samples = len(mi_tp_sample_dict['infant'][tp])

    vaginal_samples = []; csection_samples = []
    for sample in mi_tp_sample_dict['infant'][tp]:
        if desired_species in sample_species_polymorphism_dict[sample]:
            delivery_mode = subject_delivery_mode_map[sample_subject_map[sample]]
            if delivery_mode == 'Vaginal':
                vaginal_samples.append(sample)
            elif delivery_mode == 'C-section':
                csection_samples.append(sample)

    if tp != 'birth':
        num, unit = tp.split(' ')
        tp = num+unit[0]
    polymorphisms.append([sample_species_polymorphism_dict[sample][desired_species] for sample
in vaginal_samples])
    labels.append(tp + "\n" + ("n=%i" % len(vaginal_samples)))
    polymorphisms.append([sample_species_polymorphism_dict[sample][desired_species] for sample
in csection_samples])
    labels.append(tp + "\n" + ("n=%i" % len(csection_samples)))

boxprops = dict(color='#77acff')
medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(polymorphisms, patch_artist=True,
                          medianprops=medianprops,
                          flierprops=flierprops)

for i in np.arange(len(alpha_divs), step=2):
    boxplots['boxes'][i].set_facecolor(plot_utils.col_blue) # Vaginal
    boxplots['boxes'][i+1].set_facecolor(plot_utils.col_orange) # C-section

legend_elements = [Patch(facecolor=plot_utils.col_blue, label='Vaginal'),
Patch(facecolor=plot_utils.col_orange, label='C-section')]
ax.legend(handles=legend_elements, loc='upper left', frameon=False)

ax.set_ylabel("Polymorphism", fontsize=14)
ax.set_xticklabels(labels)
ax.set_yscale('log')


# ## Wait, of course there aren't multiple same-subject timepoints here (only grouping together
the infancy timepoints by month for Shao), but there could be for grouping by week in Shao
later.

# In[9]:


# Compare datasets
```

```
cohorts = ['hmp', 'backhed', 'ferretti', 'yassour', 'shao']
cohort_tp_alpha_div_dict = {cohort: defaultdict(list) for cohort in cohorts}

for sample in sample_cohort_map:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples_orig)
    cohort = sample_cohort_map[sample]
    if cohort == 'shao':
        subject, order = sample_order_map[sample]
        if subject[-1] == 'M': # Mother
            tp = 0
        elif order > 21: # Convert to month bins
            tp = round_down(order, 30.5)
        else:
            tp = order
    if cohort not in cohorts:
        continue
    cohort_tp_alpha_div_dict[cohort][tp].append(alpha_div_dict[sample])

fig, ax = plt.subplots(1, 5, figsize=(13, 4),
                       sharey=True,
                       gridspec_kw={'width_ratios': [3, 2, 3, 2, 2]})

# 1 week
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]
plot_data = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

boxplots = ax[0].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j in zip([(0, 1), (1, 2), (0, 2)], [3.5, 3.15, -0.25]):
    i1, i2 = itup
    t, p = scipy.stats.ttest_ind(plot_data[i1], plot_data[i2])
    print("Cohen's D: %s" % str(cohenD(plot_data[i1], plot_data[i2])))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[0], j, i1+1, i2+1, color=color, tickh=0.05)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[0].text(i1+1+((i2-i1)/2)-offset, j+0.04, label, color=color)

ax[0].set_title('1 week')
ax[0].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_alpha_div_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])
ax[0].set_ylabel('Shannon alpha diversity')

# 2 weeks
cohort_tp_tups = [('yassour', 'I3'), ('shao',14)]
plot_data = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

boxplots = ax[1].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
```

```
for itup, j in zip([(0, 1)], [-0.25]):
    i1, i2 = itup
    t, p = scipy.stats.ttest_ind(plot_data[i1], plot_data[i2])
    print("Cohen's D: %s" % str(cohenD(plot_data[i1], plot_data[i2])))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[1], j, i1+1, i2+1, color=color, tickh=0.05)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[1].text(i1+1+((i2-i1)/2)-offset, j+0.04, label, color=color)

ax[1].set_title('2 weeks')
ax[1].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_alpha_div_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])

# 1 month
cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
plot_data = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

boxplots = ax[2].boxplot(plot_data, patch_artist=True, widths=0.6, boxprops=dict(color='black'),
medianprops=dict(color='black'), flierprops=dict(marker='.'))
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j in zip([(0, 1), (1, 2), (0, 2)], [3.5, 3.15, -0.25]):
    i1, i2 = itup
    t, p = scipy.stats.ttest_ind(plot_data[i1], plot_data[i2])
    print("Cohen's D: %s" % str(cohenD(plot_data[i1], plot_data[i2])))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[2], j, i1+1, i2+1, color=color, tickh=0.05)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[2].text(i1+1+((i2-i1)/2)-offset, j+0.04, label, color=color)

ax[2].set_title('1 month')
ax[2].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_alpha_div_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])

# 4 months
cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
plot_data = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

boxplots = ax[3].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j in zip([(0, 1)], [-0.25]):
    i1, i2 = itup
    t, p = scipy.stats.ttest_ind(plot_data[i1], plot_data[i2])
    print("Cohen's D: %s" % str(cohenD(plot_data[i1], plot_data[i2])))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[3], j, i1+1, i2+1, color=color, tickh=0.05)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
```

```
    ax[3].text(i1+1+((i2-i1)/2)-offset, j+0.04, label, color=color)

ax[3].set_title('4 months')
ax[3].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_alpha_div_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])

# 12 months
a = []
for tp in cohort_tp_alpha_div_dict['shao']:
    if tp > 300:
        a += cohort_tp_alpha_div_dict['shao'][tp]
b = cohort_tp_alpha_div_dict['backhed']['I3']
plot_data = [a, b]

boxplots = ax[4].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j in zip([(0, 1)], [-0.25]):
    i1, i2 = itup
    t, p = scipy.stats.ttest_ind(plot_data[i1], plot_data[i2])
    print("Cohen's D: %s" % str(cohenD(plot_data[i1], plot_data[i2])))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[4], j, i1+1, i2+1, color=color, tickh=0.05)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[4].text(i1+1+((i2-i1)/2)-offset, j+0.04, label, color=color)

ax[4].set_title('12 months')
ax[4].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(), len(vals))                          for
(cohort, vals) in zip(['shao', 'backhed'], [a, b])])

fig.savefig('%s/alpha_div_compare_cohort_matched_tp.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[10]:


# Store data for R Q-Q plots

# 1 week
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]
for (cohort, tp) in cohort_tp_tups:
    alpha_divs = cohort_tp_alpha_div_dict[cohort][tp]
    with open('%s/alpha_divs_%s_%s.txt' % (config.analysis_directory, cohort, '1wk'), 'w') as f:
        f.write('\n'.join([str(ad) for ad in alpha_divs]))

# 2 weeks
cohort_tp_tups = [('yassour', 'I3'), ('shao',14)]
for (cohort, tp) in cohort_tp_tups:
    alpha_divs = cohort_tp_alpha_div_dict[cohort][tp]
    with open('%s/alpha_divs_%s_%s.txt' % (config.analysis_directory, cohort, '2wk'), 'w') as f:
        f.write('\n'.join([str(ad) for ad in alpha_divs]))

# 1 month
```

```
cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
for (cohort, tp) in cohort_tp_tups:
    alpha_divs = cohort_tp_alpha_div_dict[cohort][tp]
    with open('%s/alpha_divs_%s_%s.txt' % (config.analysis_directory, cohort, '1mo'), 'w') as f:
        f.write('\n'.join([str(ad) for ad in alpha_divs]))

# 4 months
cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
for (cohort, tp) in cohort_tp_tups:
    alpha_divs = cohort_tp_alpha_div_dict[cohort][tp]
    with open('%s/alpha_divs_%s_%s.txt' % (config.analysis_directory, cohort, '4mo'), 'w') as f:
        f.write('\n'.join([str(ad) for ad in alpha_divs]))

# 12 months
a = []
for tp in cohort_tp_alpha_div_dict['shao']:
    if tp > 300:
        a += cohort_tp_alpha_div_dict['shao'][tp]
b = cohort_tp_alpha_div_dict['backhed']['I3']
for cohort, alpha_divs in zip(['shao', 'backhed'], [a, b]):
    with open('%s/alpha_divs_%s_%s.txt' % (config.analysis_directory, cohort, '1yr'), 'w') as f:
        f.write('\n'.join([str(ad) for ad in alpha_divs]))


# In[11]:


# Fake Q-Q plots to check validity of T test

fig, ax = plt.subplots(1, 5, figsize=(15, 3),
                       sharey=True)

# 1 week
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
y = [[np.quantile(x[j], i) for i in np.arange(0, 1, step=(1/len(x[0])))] for j in [0, 1, 2]]
ax[0].plot(y[0], y[1], '.-', markersize=10)
ax[0].plot(y[1], y[2], '.-', markersize=10)
ax[0].plot(y[0], y[2], '.-', markersize=10)
ax[0].set_title("Alpha diversities matched to equal quantiles (based on linear interpolation
between min and max)\n1 week", loc='left')

# 2 weeks
cohort_tp_tups = [('yassour', 'I3'), ('shao',14)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
y = [[np.quantile(x[j], i) for i in np.arange(0, 1, step=(1/len(x[0])))] for j in [0, 1]]
ax[1].plot(y[0], y[1], '.-', markersize=10)
ax[1].set_title("2 weeks", loc='left')

# 1 month
cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
y = [[np.quantile(x[j], i) for i in np.arange(0, 1, step=(1/len(x[0])))] for j in [0, 1, 2]]
ax[2].plot(y[0], y[1], '.-', markersize=10)
ax[2].plot(y[1], y[2], '.-', markersize=10)
ax[2].plot(y[0], y[2], '.-', markersize=10)
ax[2].set_title("1 month", loc='left')

# 4 months
cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
```

```
y = [[np.quantile(x[j], i) for i in np.arange(0, 1, step=(1/len(x[0])))] for j in [0, 1]]
ax[3].plot(y[0], y[1], '.-', markersize=10)
ax[3].set_title("4 months", loc='left')

# 12 months
a = []
for tp in cohort_tp_alpha_div_dict['shao']:
    if tp > 300:
        a += cohort_tp_alpha_div_dict['shao'][tp]
b = cohort_tp_alpha_div_dict['backhed']['I3']
x = [a, b]
y = [[np.quantile(x[j], i) for i in np.arange(0, 1, step=(1/len(x[0])))] for j in [0, 1]]
ax[4].plot(y[0], y[1], '.-', markersize=10)
ax[4].set_title("12 months", loc='left')

plt.tight_layout()


# In[13]:


# Actual Q-Q plots to check validity of T test

fig, ax = plt.subplots(1, 5, figsize=(15, 3),
                       sharey=True)

# 1 week
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
all_vals = sorted(list(set(x[0]+x[1]+x[2])))
y = [[stats.percentileofscore(x[j], val) for val in all_vals] for j in [0, 1, 2]]
ax[0].plot(np.arange(100), np.arange(100), '-', color='gray')
ax[0].plot(y[0], y[1], '.-', markersize=5)
ax[0].plot(y[1], y[2], '.-', markersize=5)
ax[0].plot(y[0], y[2], '.-', markersize=5)
ax[0].set_title("Percentile-percentile plot\n1 week", loc='left')

# 2 weeks
cohort_tp_tups = [('yassour', 'I3'), ('shao',14)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
all_vals = sorted(list(set(x[0]+x[1])))
y = [[stats.percentileofscore(x[j], val) for val in all_vals] for j in [0, 1]]
ax[1].plot(np.arange(100), np.arange(100), '-', color='gray')
ax[1].plot(y[0], y[1], '.-', markersize=5)
ax[1].set_title("2 weeks", loc='left')

# 1 month
cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
all_vals = sorted(list(set(x[0]+x[1]+x[2])))
y = [[stats.percentileofscore(x[j], val) for val in all_vals] for j in [0, 1, 2]]
ax[2].plot(np.arange(100), np.arange(100), '-', color='gray')
ax[2].plot(y[0], y[1], '.-', markersize=5)
ax[2].plot(y[1], y[2], '.-', markersize=5)
ax[2].plot(y[0], y[2], '.-', markersize=5)
ax[2].set_title("1 month", loc='left')

# 4 months
cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
all_vals = sorted(list(set(x[0]+x[1])))
```

```
y = [[stats.percentileofscore(x[j], val) for val in all_vals] for j in [0, 1]]
ax[3].plot(np.arange(100), np.arange(100), '-', color='gray')
ax[3].plot(y[0], y[1], '.-', markersize=5)
ax[3].set_title("4 months", loc='left')

# 12 months
a = []
for tp in cohort_tp_alpha_div_dict['shao']:
    if tp > 300:
        a += cohort_tp_alpha_div_dict['shao'][tp]
b = cohort_tp_alpha_div_dict['backhed']['I3']
x = [a, b]
all_vals = sorted(list(set(x[0]+x[1])))
y = [[stats.percentileofscore(x[j], val) for val in all_vals] for j in [0, 1]]
ax[4].plot(np.arange(100), np.arange(100), '-', color='gray')
ax[4].plot(y[0], y[1], '.-', markersize=5)
ax[4].set_title("12 months", loc='left')

plt.tight_layout()


# In[43]:


from scipy import stats

fig, ax = plt.subplots(3, 4, figsize=(9, 9),sharey=True, sharex=True)

# 1 week
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
stats.probplot(x[0], plot=ax[0][0]); ax[0][0].set_title("Ferretti - 1wk")
stats.probplot(x[1], plot=ax[0][1]); ax[0][1].set_title("Yassour - 1wk")
stats.probplot(x[2], plot=ax[0][2]); ax[0][2].set_title("Shao - 1wk")

# 2 weeks
cohort_tp_tups = [('yassour', 'I3'), ('shao',14)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
stats.probplot(x[0], plot=ax[0][3]); ax[0][3].set_title("Yassour - 2wk")
stats.probplot(x[1], plot=ax[1][0]); ax[1][0].set_title("Shao - 2wk")

# 1 month
cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
stats.probplot(x[0], plot=ax[1][1]); ax[1][1].set_title("Ferretti - 1mo")
stats.probplot(x[1], plot=ax[1][2]); ax[1][2].set_title("Yassour - 1mo")
stats.probplot(x[2], plot=ax[1][3]); ax[1][3].set_title("Shao - 21d")

# 4 months
cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
stats.probplot(x[0], plot=ax[2][0]); ax[2][0].set_title("Ferretti - 4mo")
stats.probplot(x[1], plot=ax[2][1]); ax[2][1].set_title("Backhed - 4mo")

# 12 months
a = []
for tp in cohort_tp_alpha_div_dict['shao']:
    if tp > 300:
        a += cohort_tp_alpha_div_dict['shao'][tp]
b = cohort_tp_alpha_div_dict['backhed']['I3']
x = [a, b]
```

```
stats.probplot(x[0], plot=ax[2][2]); ax[2][2].set_title("Shao - 12+mo")
stats.probplot(x[1], plot=ax[2][3]); ax[2][3].set_title("Backhed - 12mo")


# In[11]:


from scipy import stats

fig, ax = plt.subplots(3, 4, figsize=(9, 9),sharey=True, sharex=True)
xs = np.arange(-3, 3, 0.01)

# 1 week
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
vals = stats.probplot(x[0]); ax[0][0].set_title("Ferretti - 1wk")
osm, osr = vals[0]; m, b, r = vals[1]; ax[0][0].plot(xs, (m*xs)+b, color='gray');
ax[0][0].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[1]); ax[0][1].set_title("Yassour - 1wk")
osm, osr = vals[0]; m, b, r = vals[1]; ax[0][1].plot(xs, (m*xs)+b, color='gray');
ax[0][1].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[2]); ax[0][2].set_title("Shao - 1wk")
osm, osr = vals[0]; m, b, r = vals[1]; ax[0][2].plot(xs, (m*xs)+b, color='gray');
ax[0][2].plot(osm, osr, '.', color='black', mfc='none')

# 2 weeks
cohort_tp_tups = [('yassour', 'I3'), ('shao',14)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
vals = stats.probplot(x[0]); ax[0][3].set_title("Yassour - 2wk")
osm, osr = vals[0]; m, b, r = vals[1]; ax[0][3].plot(xs, (m*xs)+b, color='gray');
ax[0][3].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[1]); ax[1][0].set_title("Shao - 2wk")
osm, osr = vals[0]; m, b, r = vals[1]; ax[1][0].plot(xs, (m*xs)+b, color='gray');
ax[1][0].plot(osm, osr, '.', color='black', mfc='none')

# 1 month
cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
vals = stats.probplot(x[0]); ax[1][1].set_title("Ferretti - 1mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[1][1].plot(xs, (m*xs)+b, color='gray');
ax[1][1].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[1]); ax[1][2].set_title("Yassour - 1mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[1][2].plot(xs, (m*xs)+b, color='gray');
ax[1][2].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[2]); ax[1][3].set_title("Shao - 21d")
osm, osr = vals[0]; m, b, r = vals[1]; ax[1][3].plot(xs, (m*xs)+b, color='gray');
ax[1][3].plot(osm, osr, '.', color='black', mfc='none')

# 4 months
cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
x = [cohort_tp_alpha_div_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
vals = stats.probplot(x[0]); ax[2][0].set_title("Ferretti - 4mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[2][0].plot(xs, (m*xs)+b, color='gray');
ax[2][0].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[1]); ax[2][1].set_title("Backhed - 4mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[2][1].plot(xs, (m*xs)+b, color='gray');
ax[2][1].plot(osm, osr, '.', color='black', mfc='none')

# 12 months
a = []
for tp in cohort_tp_alpha_div_dict['shao']:
```

```
    if tp > 300:
        a += cohort_tp_alpha_div_dict['shao'][tp]
b = cohort_tp_alpha_div_dict['backhed']['I3']
x = [a, b]
vals = stats.probplot(x[0]); ax[2][2].set_title("Shao - 12+mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[2][2].plot(xs, (m*xs)+b, color='gray');
ax[2][2].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[1]); ax[2][3].set_title("Backhed - 12mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[2][3].plot(xs, (m*xs)+b, color='gray');
ax[2][3].plot(osm, osr, '.', color='black', mfc='none')

for i in range(3):
    ax[i][0].set_ylabel("Observed")

for i in range(4):
    ax[2][i].set_xlabel("Theoretical quantile")

fig.savefig('%s/alpha_diversity_qq_plots.pdf' % plot_dir)


# In[11]:


alpha_divs = [] # list of sample values for each tp
labels = []
sample_sizes = []

for tp in infant_tps_ordered:
    num_samples = len(mi_tp_sample_dict['infant'][tp])
    if num_samples < 10:
        continue # Skip timepoints with not enough data
    labels.append(tp + "\n" + ("n=%i" % num_samples))
    alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['infant'][tp]]
    alpha_divs.append(alpha_divs_tp)
    sample_sizes.append(num_samples)


alpha_divs_mother_combined = []
for i in range(len(mother_tps_ordered)):
    tp = mother_tps_ordered[i]
    num_samples = len(mi_tp_sample_dict['mother'][tp])
    alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['mother'][tp]]
    if tp == -92 or tp == 92: # Skip 3month/-3month
        continue
    alpha_divs_mother_combined += alpha_divs_tp

alpha_divs.append(alpha_divs_mother_combined)
labels.append('Mother'+ "\n" + ("n=%i" % len(alpha_divs_mother_combined)))
sample_sizes.append(len(alpha_divs_mother_combined))

alpha_divs_hmp = [alpha_div_dict[sample] for sample in hmp_samples]
alpha_divs.append(alpha_divs_hmp)
labels.append('Adult' + "\n" + ("n=%i" % len(alpha_divs_hmp)))
sample_sizes.append(len(alpha_divs_hmp))

fig, ax = plt.subplots(figsize=(14, 4))

boxprops = dict(color='black')
medianprops = dict(color='black')
flierprops = dict(marker='.')
```

```
boxplots = ax.boxplot(alpha_divs, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

boxplots['boxes'][-1].set_facecolor('#396651')
boxplots['boxes'][-2].set_facecolor('#7bb551')

ax.set_ylabel("Shannon alpha diversity\nper sample")
ax.set_title("Alpha diversity by timepoint (infants exclude Olm)")
ax.axvline(19.5, color='gray', linestyle='-')
ax.set_xticklabels(labels)

summarize_ttest(alpha_divs[1], alpha_divs[2])

plt.show()


# In[17]:


# Try downsampling hosts to 20 (second lowest number besides 5 months)

alpha_divs = [] # list of sample values for each tp
labels = []
sample_sizes = []
subject_sizes = []

for tp in infant_tps_ordered:
    num_samples = len(mi_tp_sample_dict['infant'][tp])
    if num_samples < 10:
        continue # Skip timepoints with not enough data

    subjects = set()
    for sample in mi_tp_sample_dict['infant'][tp]:
        subject = sample_subject_map[sample]
        subjects.add(subject)

    if num_samples < 20:
        subjects_downsampled = subjects
    else:
        subjects_downsampled = np.random.choice(list(subjects), 20, replace=False)

    # Use downsampled subjects
    alpha_divs_tp = []
    for sample in mi_tp_sample_dict['infant'][tp]:
        if sample_subject_map[sample] in subjects_downsampled:
            alpha_divs_tp.append(alpha_div_dict[sample])

    labels.append(tp + "\n" + ("n=%i" % len(alpha_divs_tp)))
    # alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['infant'][tp]]
    alpha_divs.append(alpha_divs_tp); sample_sizes.append(num_samples);
subject_sizes.append(len(subjects_downsampled))

# Mother ================================================================================
alpha_divs_mother_combined = []
mother_subjects = set()

for i in range(len(mother_tps_ordered)):
    tp = mother_tps_ordered[i]
    num_samples = len(mi_tp_sample_dict['mother'][tp])
```

```
    for sample in mi_tp_sample_dict['mother'][tp]:
        subject = sample_subject_map[sample]
        mother_subjects.add(subject)
    # alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['mother'][tp]]
    if tp == -92 or tp == 92: # Skip 3month/-3month
        continue
    # alpha_divs_mother_combined += alpha_divs_tp

mother_subjects_downsampled = np.random.choice(list(mother_subjects), 20, replace=False)

# Use downsampled subjects
alpha_divs_mother_combined = []

for i in range(len(mother_tps_ordered)):
    tp = mother_tps_ordered[i]
    for sample in mi_tp_sample_dict['mother'][tp]:
        subject = sample_subject_map[sample]
        if subject in mother_subjects_downsampled:
            alpha_divs_mother_combined.append(alpha_div_dict[sample])

alpha_divs.append(alpha_divs_mother_combined)
labels.append('Mother'+ "\n" + ("n=%i" % len(alpha_divs_mother_combined)))
sample_sizes.append(len(alpha_divs_mother_combined))

# HMP ============================================================================================
hmp_subjects = set()
for sample in hmp_samples:
    hmp_subjects.add(sample_subject_map[sample])

hmp_subjects_downsampled = np.random.choice(list(hmp_subjects), 20, replace=False)

# Use downsampled subjects
alpha_divs_hmp = []
subjects_so_far = set()
for sample in hmp_samples:
    subject = sample_subject_map[sample]
    if subject in subjects_so_far:
        continue
    if subject in hmp_subjects_downsampled:
        alpha_divs_hmp.append(alpha_div_dict[sample])
        subjects_so_far.add(subject)

alpha_divs.append(alpha_divs_hmp)
labels.append('Adult' + "\n" + ("n=%i" % len(alpha_divs_hmp)))
sample_sizes.append(len(alpha_divs_hmp))

# Plot ===========================================================================================
fig, ax = plt.subplots(figsize=(14, 4))

boxprops = dict(color='black')
medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(alpha_divs, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

boxplots['boxes'][-1].set_facecolor('#396651')
boxplots['boxes'][-2].set_facecolor('#7bb551')
```

```
ax.set_ylabel("Shannon alpha diversity\nper sample")
ax.set_title("Alpha diversity by timepoint (infants exclude Olm)")
ax.axvline(19.5, color='gray', linestyle='-')
ax.set_xticklabels(labels)

print('\nBirth vs. 1 day:')
summarize_ttest(alpha_divs[0], alpha_divs[1], simple=True)
print('\n1 day vs 3 day:')
summarize_ttest(alpha_divs[1], alpha_divs[2], simple=True)
print('\n3 day vs 1 year:')
summarize_ttest(alpha_divs[2], alpha_divs[-3], simple=True)

fig.savefig('%s/alpha_div_over_time_downsampled.pdf' % (config.analysis_directory),
bbox_inches='tight')
plt.show()


# In[11]:


# Try downsampling many times TODO

p1s = []; p2s = []; p3s = []
t1s = []; t2s = []; t3s = []

for _ in range(1000):
    alpha_divs = [] # list of sample values for each tp

    for tp in infant_tps_ordered:
        num_samples = len(mi_tp_sample_dict['infant'][tp])
        if num_samples < 10:
            continue # Skip timepoints with not enough data

        subjects = set()
        for sample in mi_tp_sample_dict['infant'][tp]:
            subject = sample_subject_map[sample]
            subjects.add(subject)

        if num_samples < 20:
            subjects_downsampled = subjects
        else:
            subjects_downsampled = np.random.choice(list(subjects), 20, replace=False)

        # Use downsampled subjects
        alpha_divs_tp = []
        for sample in mi_tp_sample_dict['infant'][tp]:
            if sample_subject_map[sample] in subjects_downsampled:
                alpha_divs_tp.append(alpha_div_dict[sample])

        alpha_divs.append(alpha_divs_tp); sample_sizes.append(num_samples);
subject_sizes.append(len(subjects_downsampled))

    # Mother
================================================================================
    alpha_divs_mother_combined = []
    mother_subjects = set()

    for i in range(len(mother_tps_ordered)):
        tp = mother_tps_ordered[i]
        num_samples = len(mi_tp_sample_dict['mother'][tp])
        for sample in mi_tp_sample_dict['mother'][tp]:
```

```
            subject = sample_subject_map[sample]
            mother_subjects.add(subject)
        # alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['mother'][tp]]
        if tp == -92 or tp == 92: # Skip 3month/-3month
            continue
        # alpha_divs_mother_combined += alpha_divs_tp

    mother_subjects_downsampled = np.random.choice(list(mother_subjects), 20, replace=False)

    # Use downsampled subjects
    alpha_divs_mother_combined = []

    for i in range(len(mother_tps_ordered)):
        tp = mother_tps_ordered[i]
        for sample in mi_tp_sample_dict['mother'][tp]:
            subject = sample_subject_map[sample]
            if subject in mother_subjects_downsampled:
                alpha_divs_mother_combined.append(alpha_div_dict[sample])

    alpha_divs.append(alpha_divs_mother_combined)
    labels.append('Mother'+ "\n" + ("n=%i" % len(alpha_divs_mother_combined)))
    sample_sizes.append(len(alpha_divs_mother_combined))

    # HMP ================================================================================
    hmp_subjects = set()
    for sample in hmp_samples:
        hmp_subjects.add(sample_subject_map[sample])

    hmp_subjects_downsampled = np.random.choice(list(hmp_subjects), 20, replace=False)

    # Use downsampled subjects
    alpha_divs_hmp = []
    subjects_so_far = set()
    for sample in hmp_samples:
        subject = sample_subject_map[sample]
        if subject in subjects_so_far:
            continue
        if subject in hmp_subjects_downsampled:
            alpha_divs_hmp.append(alpha_div_dict[sample])
            subjects_so_far.add(subject)

    alpha_divs.append(alpha_divs_hmp)

    # print('\nBirth vs. 1 day:')
    t, p = scipy.stats.ttest_ind(alpha_divs[0], alpha_divs[1])
    t1s.append(t); p1s.append(p)
    # print('\n1 day vs 3 day:')
    t, p = scipy.stats.ttest_ind(alpha_divs[1], alpha_divs[2])
    t2s.append(t); p2s.append(p)
    # print('\n3 day vs 1 year:')
    t, p = scipy.stats.ttest_ind(alpha_divs[2], alpha_divs[-3])
    t3s.append(t); p3s.append(p)


# In[12]:


print(sum(np.array(p1s) >= 0.05))
print(sum(np.array(p2s) >= 0.05))
print(sum(np.array(p3s) >= 0.05))
```

```
# In[10]:


# Statistical significance to asterisk representation mapping
def get_sig_str(pval):
    if pval <= 0.001:
        return '***'
    elif pval <= 0.01:
        return '**'
    elif pval <= 0.05:
        return '*'
    elif pval > 0.05:
        return 'ns'


# In[26]:


# THIS IS A SUPPLEMENTAL FIGURE
# Compare alpha diversity for Mom from each cohort, HMP1-2, Poyet, Korpela

alpha_divs_mother_by_cohort = defaultdict(list)

sample_cohort_map = su.parse_sample_cohort_map()
mother_samples = su.get_sample_names('mother')
hmp_samples = su.get_sample_names('hmp')
mother_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

for tp in mother_tps_ordered:
    if tp == -92 or tp == 92: # Skip 3month/-3month
        continue
    for sample in mi_tp_sample_dict['mother'][tp]:
        cohort = sample_cohort_map[sample]
        alpha_div = alpha_div_dict[sample]
        alpha_divs_mother_by_cohort[cohort].append(alpha_div)

alpha_divs = []
alpha_divs_all_mothers = []
labels = []
for cohort in mother_cohorts:
    alpha_divs.append(alpha_divs_mother_by_cohort[cohort])
    alpha_divs_all_mothers += alpha_divs_mother_by_cohort[cohort]
    labels.append('Mother\n%s\nn=%i' % (cohort.capitalize(),
len(alpha_divs_mother_by_cohort[cohort])))
'''
hmp_alpha_divs = [alpha_div_dict[sample] for sample in hmp_samples]
alpha_divs.append(hmp_alpha_divs)
labels.append('Adult\nHMP1-2\nn=%i' % (len(hmp_alpha_divs)))
'''
hmp_female_alpha_divs = []
subjects_so_far = set()
for sample in hmp_samples:
    subject = sample_subject_map[sample]
    if subject in subjects_so_far:
        continue
    if subject in hmp_female_subjects:
        hmp_female_alpha_divs.append(alpha_div_dict[sample])
        subjects_so_far.add(subject)
alpha_divs.append(hmp_female_alpha_divs)
labels.append('Adult\n(Female)\nHMP1-2\nn=%i' % (len(hmp_female_alpha_divs)))
```

```
'''
alpha_divs.append(list(poyet_alpha_div_dict.values()))
labels.append('Adult\nPoyet\nn=%i' % (len(poyet_alpha_div_dict)))

alpha_divs.append(list(korpela_alpha_div_dict.values()))
labels.append('Adult\nKorpela\nn=%i' % (len(korpela_alpha_div_dict)))

alpha_divs.append(list(qin_alpha_div_dict.values()))
labels.append('Adult\nQin\nn=%i' % (len(qin_alpha_div_dict)))
'''
qin_female_alpha_divs = [qin_alpha_div_dict[sample] for sample in qin_samples
if qin_sample_gender_dict[sample] == 'female']
alpha_divs.append(qin_female_alpha_divs)
labels.append('Adult\n(Female)\nQin\nn=%i' % (len(qin_female_alpha_divs)))


fig, ax = plt.subplots(figsize=(9, 7))

boxprops = dict(color='black')
medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(alpha_divs, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops, widths=0.7)
for patch in boxplots['boxes'][:4]:
    patch.set_facecolor('#7bb551')
for patch in boxplots['boxes'][-2:]:
    patch.set_facecolor('#396651')

# Perform pairwise t-test
j = 0
for i in [0, 1, 2, 3]:
    a1 = alpha_divs[i]
    a2 = alpha_divs[4] # Compare to HMP first
    t, p = scipy.stats.ttest_ind(a1, a2); D = cohenD(a1, a2)
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax, 5.3 - (0.8*j), i+1, 5, color=color, tickh=0.05)
    ax.text(1+i+((4-i)/2.0)-0.45, 5.3 - (0.8*j) +0.02, '{:.1e}'.format(p) + (', D=%.01f' % D),
color=color)
    # ax[0].text(0.8+i+((4-i)/2.0), 5.3 - j+0.02, '{:.2e}'.format(p), color=color)
    j += 0.3

j = 0
for i in [0, 1, 2, 3]:
    a1 = alpha_divs[i]
    a2 = alpha_divs[5] # Compare to Qin
    t, p = scipy.stats.ttest_ind(a1, a2); D = cohenD(a1, a2)
    print(p)
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax, -0.5 + (0.8*j), i+1, 6, color=color, tickh=0.05)
    ax.text(1+i+((5-i)/2.0)-0.45, -0.5 + (0.8*j) + 0.02, '{:.1e}'.format(p) + (', D=%.01f' % D),
color=color)
    # ax[0].text(0.8+i+((4-i)/2.0), -0.5+j+0.02, '{:.2e}'.format(p), color=color)
    j += 0.3

ax.set_ylabel("Shannon alpha diversity")
# ax.set_title("Alpha diversity of adult samples by cohort")
ax.set_xticklabels(labels)

print("Mothers vs. HMP females")
summarize_ttest(alpha_divs_all_mothers, hmp_female_alpha_divs)
```

```
print("Mothers vs. Qin females")
summarize_ttest(alpha_divs_all_mothers, qin_female_alpha_divs)

plt.show()
fig.savefig('%s/alpha_div_mothers_vs_adults_by_cohort.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[13]:


# THIS IS A SUPPLEMENTAL FIGURE
# Compare alpha diversity for Mom from each cohort, HMP1-2, Poyet, Korpela
# This time remove tests

alpha_divs_mother_by_cohort = defaultdict(list)

sample_cohort_map = su.parse_sample_cohort_map()
mother_samples = su.get_sample_names('mother')
hmp_samples = su.get_sample_names('hmp')
mother_cohorts = ['backhed', 'ferretti', 'yassour', 'shao']

for tp in mother_tps_ordered:
    if tp == -92 or tp == 92: # Skip 3month/-3month
        continue
    for sample in mi_tp_sample_dict['mother'][tp]:
        cohort = sample_cohort_map[sample]
        alpha_div = alpha_div_dict[sample]
        alpha_divs_mother_by_cohort[cohort].append(alpha_div)

alpha_divs = []
alpha_divs_all_mothers = []
labels = []
for cohort in mother_cohorts:
    alpha_divs.append(alpha_divs_mother_by_cohort[cohort])
    alpha_divs_all_mothers += alpha_divs_mother_by_cohort[cohort]
    labels.append('Mother\n%s\nn=%i' % (cohort.capitalize(),
len(alpha_divs_mother_by_cohort[cohort])))
'''
hmp_alpha_divs = [alpha_div_dict[sample] for sample in hmp_samples]
alpha_divs.append(hmp_alpha_divs)
labels.append('Adult\nHMP1-2\nn=%i' % (len(hmp_alpha_divs)))
'''
hmp_female_alpha_divs = []
subjects_so_far = set()
for sample in hmp_samples:
    subject = sample_subject_map[sample]
    if subject in subjects_so_far:
        continue
    if subject in hmp_female_subjects:
        hmp_female_alpha_divs.append(alpha_div_dict[sample])
        subjects_so_far.add(subject)
alpha_divs.append(hmp_female_alpha_divs)
labels.append('Adult\n(Female)\nHMP1-2\nn=%i' % (len(hmp_female_alpha_divs)))
'''
alpha_divs.append(list(poyet_alpha_div_dict.values()))
labels.append('Adult\nPoyet\nn=%i' % (len(poyet_alpha_div_dict)))

alpha_divs.append(list(korpela_alpha_div_dict.values()))
labels.append('Adult\nKorpela\nn=%i' % (len(korpela_alpha_div_dict)))
```

```
alpha_divs.append(list(qin_alpha_div_dict.values()))
labels.append('Adult\nQin\nn=%i' % (len(qin_alpha_div_dict)))
'''
qin_female_alpha_divs = [qin_alpha_div_dict[sample] for sample in qin_samples
if qin_sample_gender_dict[sample] == 'female']
alpha_divs.append(qin_female_alpha_divs)
labels.append('Adult\n(Female)\nQin\nn=%i' % (len(qin_female_alpha_divs)))


fig, ax = plt.subplots(figsize=(6, 5))

boxprops = dict(color='black')
medianprops = dict(color='black')
flierprops = dict(marker='.')

boxplots = ax.boxplot(alpha_divs, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops, widths=0.7)
for patch in boxplots['boxes'][:4]:
    patch.set_facecolor('#7bb551')
for patch in boxplots['boxes'][-2:]:
    patch.set_facecolor('#396651')

ax.set_ylabel("Shannon alpha diversity")
# ax.set_title("Alpha diversity of adult samples by cohort")
ax.set_xticklabels(labels)

print("Mothers vs. HMP females")
summarize_ttest(alpha_divs_all_mothers, hmp_female_alpha_divs)
print("Mothers vs. Qin females")
summarize_ttest(alpha_divs_all_mothers, qin_female_alpha_divs)

plt.show()
fig.savefig('%s/alpha_div_mothers_vs_adults_by_cohort_no_tests.pdf' %
(config.analysis_directory), bbox_inches='tight')


# In[21]:


# Store data for R Q-Q plots

for i, label in zip(np.arange(len(alpha_divs)), ['backhed_mother', 'ferretti_mother',
'yassour_mother', 'shao_mother', 'HMP1-2_female', 'qin_female']):
    with open('%s/alpha_divs_%s.txt' % (config.analysis_directory, label), 'w') as f:
        f.write('\n'.join([str(ad) for ad in alpha_divs[i]]))


# In[22]:


# Fake Q-Q plots to check validity of T test

fig, ax = plt.subplots(1, 2, figsize=(8, 3),
                       sharey=True)

a = alpha_divs # For brevity
y = [[np.quantile(a[j], i) for i in np.arange(0, 1, step=0.01)] for j in [0, 1, 2, 3, 4, 5]]

# Comparisons to HMP(female)
ax[0].set_title("Alpha diversities matched to equal quantiles\n(based on linear interpolation
between min and max)\nMothers vs. HMP female", loc='left')
```

```
ax[0].plot(y[0], y[4], '.-', markersize=5, label="Backhed")
ax[0].plot(y[1], y[4], '.-', markersize=5, label="Ferretti")
ax[0].plot(y[2], y[4], '.-', markersize=5, label="Yassour")
ax[0].plot(y[3], y[4], '.-', markersize=5, label="Shao")
ax[0].legend()

# Comparisons to Qin(female)
ax[1].set_title("Mothers vs. Qin female", loc='left')
ax[1].plot(y[0], y[5], '.-', markersize=5, label="Backhed")
ax[1].plot(y[1], y[5], '.-', markersize=5, label="Ferretti")
ax[1].plot(y[2], y[5], '.-', markersize=5, label="Yassour")
ax[1].plot(y[3], y[5], '.-', markersize=5, label="Shao")
ax[1].legend()


# In[23]:


# Actual Q-Q plots to check validity of T test

fig, ax = plt.subplots(1, 2, figsize=(6, 3),
                       sharey=True)

a = alpha_divs # For brevity

# Comparisons to HMP(female)

ax[0].plot(np.arange(100), np.arange(100), '-', color='gray')
ax[0].set_title("Percentile-percentile plot\nMothers vs. HMP female", loc='left')
y = [[stats.percentileofscore(a[j], val) for val in sorted(a[0]+a[4])] for j in [0, 4]];
ax[0].plot(y[0], y[1], '.-', markersize=5, label="Backhed")
y = [[stats.percentileofscore(a[j], val) for val in sorted(a[1]+a[4])] for j in [1, 4]];
ax[0].plot(y[0], y[1], '.-', markersize=5, label="Ferretti")
y = [[stats.percentileofscore(a[j], val) for val in sorted(a[2]+a[4])] for j in [2, 4]];
ax[0].plot(y[0], y[1], '.-', markersize=5, label="Yassour")
y = [[stats.percentileofscore(a[j], val) for val in sorted(a[3]+a[4])] for j in [3, 4]];
ax[0].plot(y[0], y[1], '.-', markersize=5, label="Shao")
ax[0].legend()

ax[1].plot(np.arange(100), np.arange(100), '-', color='gray')
ax[1].set_title("Mothers vs. Qin female", loc='left')
y = [[stats.percentileofscore(a[j], val) for val in sorted(a[0]+a[5])] for j in [0, 5]];
ax[1].plot(y[0], y[1], '.-', markersize=5, label="Backhed")
y = [[stats.percentileofscore(a[j], val) for val in sorted(a[1]+a[5])] for j in [1, 5]];
ax[1].plot(y[0], y[1], '.-', markersize=5, label="Ferretti")
y = [[stats.percentileofscore(a[j], val) for val in sorted(a[2]+a[5])] for j in [2, 5]];
ax[1].plot(y[0], y[1], '.-', markersize=5, label="Yassour")
y = [[stats.percentileofscore(a[j], val) for val in sorted(a[3]+a[5])] for j in [3, 5]];
ax[1].plot(y[0], y[1], '.-', markersize=5, label="Shao")
ax[1].legend()

plt.tight_layout()


# In[24]:


fig, ax = plt.subplots()
ax.hist(a[4], bins=20, label='HMP(Female)', alpha=0.4)
ax.hist(a[0], bins=20, label='Backhed(Mother)', alpha=0.4)
```

```
# In[15]:


# Store polymorphism CSV for GLMM analysis

f = open('%s/polymorphism.csv' % config.analysis_directory, 'w')
f.write(','.join(['sample', 'species', 'subject', 'cohort', 'day', 'polymorphism']) + '\n')

for sample in infant_samples:
    cohort = sample_cohort_map[sample]
    subject = sample_subject_map[sample]
    day = mi_sample_day_dict[sample]
    feeding_mode = subject_feeding_mode_map[subject] if subject in subject_feeding_mode_map else
'NA'
    delivery_mode = subject_delivery_mode_map[subject] if subject in subject_delivery_mode_map
else 'NA'
    for species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][species]
        f.write(','.join([str(val) for val in [sample, species, subject, cohort, day,
polymorphism]]) + '\n')

f.close()


# In[16]:


# Store polymorphism CSV for GLMM analysis: UPDATED

f = open('%s/polymorphism.csv' % config.analysis_directory, 'w')
f.write(','.join(['sample', 'species', 'subject', 'cohort', 'day', 'polymorphism', 'breast',
'formula', 'vaginal', 'csection']) + '\n')

for sample in infant_samples:
    cohort = sample_cohort_map[sample]
    subject = sample_subject_map[sample]
    day = mi_sample_day_dict[sample]
    feeding_mode = subject_feeding_mode_map[subject] if subject in subject_feeding_mode_map else
'NA'

    if feeding_mode == 'breast':
        breast = 1; formula = 0
    elif feeding_mode == 'mixed':
        breast = 1; formula = 1
    elif feeding_mode == 'formula':
        breast = 0; formula = 1
    else:
        breast = 'NA'; formula = 'NA'

    delivery_mode = subject_delivery_mode_map[subject] if subject in subject_delivery_mode_map
else 'NA'
    if delivery_mode == 'Vaginal':
        vaginal = 1; csection = 0
    elif delivery_mode == 'C-section':
        vaginal = 0; csection = 1
    else:
        vaginal = 'NA'; csection = 'NA'

    for species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][species]
```

```
        f.write(','.join([str(val) for val in [sample, species, subject, cohort, day,
polymorphism, breast, formula, vaginal, csection]]) + '\n')

f.close()


# In[7]:


fig, ax = plt.subplots(2, 1, figsize=(12,6))

species_list = good_species_list[0:2]

all_tps = tps_ordered_dict['infant'] + ['Mother', 'Adult']

for i in range(len(species_list)):
    desired_species = species_list[i]
    data = []
    labels = []

    polymorphism_by_tp_dict = defaultdict(list)

    for sample in mother_samples:
        for species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][species]
            if species == desired_species:
                polymorphism_by_tp_dict['Mother'].append(polymorphism)

    for sample in hmp_samples:
        for species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][species]
            if species == desired_species:
                polymorphism_by_tp_dict['Adult'].append(polymorphism)

    for tp in mi_tp_sample_dict['infant']:
        for sample in mi_tp_sample_dict['infant'][tp]:
            for species in sample_species_polymorphism_dict[sample]:
                polymorphism = sample_species_polymorphism_dict[sample][species]
                if species == desired_species:
                    polymorphism_by_tp_dict[tp].append(polymorphism)

    for tp in all_tps:
        polymorphisms = polymorphism_by_tp_dict[tp]
        data.append(polymorphisms)
        labels.append("%s\nn=%i" % (tp, len(polymorphisms)))

    ax[i].set_yscale('log')
    boxprops = dict(color='black')
    medianprops = dict(color='black')
    flierprops = dict(marker='.')
    boxplots = ax[i].boxplot(data, patch_artist=True,
                             boxprops=boxprops,
                             medianprops=medianprops,
                             flierprops=flierprops)
    for patch in boxplots['boxes'][:-2]:
        patch.set_facecolor('#77acff')
    boxplots['boxes'][-1].set_facecolor('#396651')
    boxplots['boxes'][-2].set_facecolor('#7bb551')
    ax[i].set_title('%s' % desired_species)
    ax[i].axvline(x=20.5, color='gray')
    ax[i].set_ylabel("pS")
```

```
    ax[i].set_xticklabels(labels)

plt.tight_layout()
plt.show()


# In[32]:


fig, ax = plt.subplots(7, 2, figsize=(14,14)) # , sharex=True)

species_list = good_species_list[0:14] # species_infant_prev_ordered[:3] +
species_infant_prev_ordered[4:13]

# Shao
month_bins = np.arange(4, 15) * 30.5
order_bins = [0, 4, 7, 21] + list(month_bins[1:8]) # Limit to the desired timepoint bins only
shao_samples = su.get_sample_names('Shao')

i = 0; j = 0
for desired_species in species_list:

    order_polymorphism_dict = defaultdict(list)
    order_subjects_dict = defaultdict(set)

    for sample in shao_samples:
        subject, order = sample_order_map[sample]
        if subject in order_subjects_dict[tp]: # Ensure one sample per subject for particular
timepoint bin
            continue

        if desired_species in sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][desired_species]
        else:
            continue

        if subject[-1] == 'M': # Mother
            order_polymorphism_dict[0].append(polymorphism)
        elif order > 21: # Convert to month bins
            month_bin = round_down(order, 30.5)
            order_polymorphism_dict[month_bin].append(polymorphism)
        else:
            order_polymorphism_dict[order].append(polymorphism)

        order_subjects_dict[tp].add(subject)

    plot_data = [order_polymorphism_dict[o] for o in order_bins] # if
len(order_polymorphism_dict[o]) > 5]
    order_labels = ['Mother', '4d', '7d', '21d', '5m', '6m', '7m', '8m', '9m', '10m', '11m']
    n_labels = ['n=%i' % len(order_polymorphism_dict[o]) for o in order_bins]
    # n_labels = [('n=%i' % len(order_polymorphism_dict[o]) if len(order_polymorphism_dict[o]) >
0 else '') for o in order_bins]
    order_labels_with_n = [label + '\nn=%i' % len(order_polymorphism_dict[o]) for label, o in
zip(order_labels, order_bins)] # if len(order_polymorphism_dict[o]) > 5]

    ax[i][j].set_yscale('log')
    boxplots = ax[i][j].boxplot(plot_data, patch_artist=True, boxprops=boxprops,
medianprops=medianprops, flierprops=flierprops)
    for patch in boxplots['boxes']:
        patch.set_facecolor('#77acff')
```

```
    ax[i][j].set_ylabel("Polymorphism")
    ax[i][j].set_title('%s' % plot_utils.get_pretty_species_name(desired_species), fontsize=14)
    ax[i][j].set_ylabel("pS", fontsize=14)
    ax[i][j].set_xticks(np.arange(1, len(order_labels_with_n)+1))
    if i == 6:
        ax[i][j].set_xticklabels(order_labels_with_n, fontsize=12)
    else:
        ax[i][j].set_xticklabels(n_labels, fontsize=12)

    if i < 7:
        i += 1

    if i == 7:
        i = 0
        j = 1

plt.tight_layout()
plt.subplots_adjust(hspace=0.5)
plt.show()

fig.savefig('%s/polymorphism_top_14_prevalence_species_Shao_only.pdf' %
(config.analysis_directory), bbox_inches='tight')
fig.savefig('%s/polymorphism_top_14_prevalence_species_Shao_only.png' %
(config.analysis_directory), bbox_inches='tight', dpi=500)


# In[43]:


# Try downsampling hosts to 8 (second lowest number besides birth, one day)
# Now for E. coli

desired_species = 'Escherichia_coli_58110'

polymorphisms = [] # list of sample values for each tp
labels = []; sample_sizes = []; subject_sizes = []

for tp in infant_tps_ordered:
    num_samples = len(mi_tp_sample_dict['infant'][tp])
    if num_samples < 10:
        continue # Skip timepoints with not enough data

    subjects = set()
    for sample in mi_tp_sample_dict['infant'][tp]:
        if desired_species in sample_species_polymorphism_dict[sample]:
            subject = sample_subject_map[sample]
            subjects.add(subject)

    if len(subjects) < 8:
        subjects_downsampled = subjects
    else:
        subjects_downsampled = np.random.choice(list(subjects), 8, replace=False)

    # Use downsampled subjects
    polymorphisms_tp = []
    for sample in mi_tp_sample_dict['infant'][tp]:
        if sample_subject_map[sample] in subjects_downsampled and desired_species in
sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][desired_species]
            polymorphisms_tp.append(polymorphism)
```

```
    labels.append(tp + "\n" + ("n=%i" % len(polymorphisms_tp)))
    polymorphisms.append(polymorphisms_tp) # sample_sizes.append(num_samples);
subject_sizes.append(len(subjects_downsampled))

# Mother ================================================================================
polymorphisms_mother_combined = []
mother_subjects = set()

for tp in mother_tps_ordered:
    if tp == -92 or tp == 92: # Skip 3month/-3month
        continue
    for sample in mi_tp_sample_dict['mother'][tp]:
        if desired_species in sample_species_polymorphism_dict[sample]:
            subject = sample_subject_map[sample]
            mother_subjects.add(subject)

mother_subjects_downsampled = np.random.choice(list(mother_subjects), 8, replace=False)

# Use downsampled subjects
polymorphisms_mother_combined = []
for tp in mother_tps_ordered:
    for sample in mi_tp_sample_dict['mother'][tp]:
        if sample_subject_map[sample] in mother_subjects_downsampled and desired_species in
sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][desired_species]
            polymorphisms_mother_combined.append(polymorphism)

labels.append('Mother'+ "\n" + ("n=%i" % len(polymorphisms_mother_combined)))
polymorphisms.append(polymorphisms_mother_combined)

# HMP ===================================================================================
hmp_subjects = set()
for sample in hmp_samples:
    if desired_species in sample_species_polymorphism_dict[sample]:
        hmp_subjects.add(sample_subject_map[sample])

hmp_subjects_downsampled = np.random.choice(list(hmp_subjects), 8, replace=False)

# Use downsampled subjects
polymorphisms_hmp = []
subjects_so_far = set()
for sample in hmp_samples:
    subject = sample_subject_map[sample]
    if desired_species in sample_species_polymorphism_dict[sample]:
        if subject in subjects_so_far:
            continue
        if subject in hmp_subjects_downsampled:
            polymorphisms_hmp.append(sample_species_polymorphism_dict[sample][desired_species])
            subjects_so_far.add(subject)

labels.append('HMP' + "\n" + ("n=%i" % len(polymorphisms_hmp)))
polymorphisms.append(polymorphisms_hmp)

# Plot ==================================================================================
fig, ax = plt.subplots(figsize=(14, 4))

boxprops = dict(color='black')
medianprops = dict(color='black')
flierprops = dict(marker='.')
```

```
boxplots = ax.boxplot(polymorphisms, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

boxplots['boxes'][-1].set_facecolor('#396651')
boxplots['boxes'][-2].set_facecolor('#7bb551')

ax.set_ylabel("Polymorphism\nper sample")
ax.set_title("Polymorphism by timepoint (infants exclude Olm)")
ax.axvline(19.5, color='gray', linestyle='-')
ax.set_xticklabels(labels)
ax.set_yscale('log')

print('\nBirth vs. 1 day:')
summarize_ttest(polymorphisms[0], polymorphisms[1], simple=True)
print('\n1 day vs 3 day:')
summarize_ttest(polymorphisms[1], polymorphisms[2], simple=True)
print('\n3 day vs 1 year:')
summarize_ttest(polymorphisms[2], polymorphisms[-3], simple=True)

fig.savefig('%s/polymorphism_over_time_downsampled.pdf' % (config.analysis_directory),
bbox_inches='tight')
plt.show()


# In[49]:


# Combined downsampled alpha diversity and polymorphism plot

fig, ax = plt.subplots(2, 1, figsize=(13,7))
boxprops = dict(color='black'); medianprops = dict(color='black'); flierprops = dict(marker='.')

# Try downsampling hosts to 20 (second lowest number besides 5 months)
# For alpha diversity

alpha_divs = [] # list of sample values for each tp
labels = []
sample_sizes = []
subject_sizes = []

for tp in infant_tps_ordered:
    num_samples = len(mi_tp_sample_dict['infant'][tp])
    if num_samples < 10:
        continue # Skip timepoints with not enough data

    subjects = set()
    for sample in mi_tp_sample_dict['infant'][tp]:
        subject = sample_subject_map[sample]
        subjects.add(subject)

    if num_samples < 20:
        subjects_downsampled = subjects
    else:
        subjects_downsampled = np.random.choice(list(subjects), 20, replace=False)

    # Use downsampled subjects
    alpha_divs_tp = []
    for sample in mi_tp_sample_dict['infant'][tp]:
        if sample_subject_map[sample] in subjects_downsampled:
```

```
            alpha_divs_tp.append(alpha_div_dict[sample])

    labels.append(tp + "\n" + ("n=%i" % len(alpha_divs_tp)))
    # alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['infant'][tp]]
    alpha_divs.append(alpha_divs_tp); sample_sizes.append(num_samples);
subject_sizes.append(len(subjects_downsampled))

# Mother ================================================================================
alpha_divs_mother_combined = []
mother_subjects = set()

for i in range(len(mother_tps_ordered)):
    tp = mother_tps_ordered[i]
    num_samples = len(mi_tp_sample_dict['mother'][tp])
    for sample in mi_tp_sample_dict['mother'][tp]:
        subject = sample_subject_map[sample]
        mother_subjects.add(subject)
    # alpha_divs_tp = [alpha_div_dict[sample] for sample in mi_tp_sample_dict['mother'][tp]]
    if tp == -92 or tp == 92: # Skip 3month/-3month
        continue
    # alpha_divs_mother_combined += alpha_divs_tp

mother_subjects_downsampled = np.random.choice(list(mother_subjects), 20, replace=False)

# Use downsampled subjects
alpha_divs_mother_combined = []

for i in range(len(mother_tps_ordered)):
    tp = mother_tps_ordered[i]
    for sample in mi_tp_sample_dict['mother'][tp]:
        subject = sample_subject_map[sample]
        if subject in mother_subjects_downsampled:
            alpha_divs_mother_combined.append(alpha_div_dict[sample])

alpha_divs.append(alpha_divs_mother_combined)
labels.append('Mother'+ "\n" + ("n=%i" % len(alpha_divs_mother_combined)))
sample_sizes.append(len(alpha_divs_mother_combined))

# HMP ===================================================================================
hmp_subjects = set()
for sample in hmp_samples:
    hmp_subjects.add(sample_subject_map[sample])

hmp_subjects_downsampled = np.random.choice(list(hmp_subjects), 20, replace=False)

# Use downsampled subjects
alpha_divs_hmp = []
subjects_so_far = set()
for sample in hmp_samples:
    subject = sample_subject_map[sample]
    if subject in subjects_so_far:
        continue
    if subject in hmp_subjects_downsampled:
        alpha_divs_hmp.append(alpha_div_dict[sample])
        subjects_so_far.add(subject)

alpha_divs.append(alpha_divs_hmp)
labels.append('HMP' + "\n" + ("n=%i" % len(alpha_divs_hmp)))
sample_sizes.append(len(alpha_divs_hmp))

# Plot ==================================================================================
```

```
boxplots = ax[0].boxplot(alpha_divs, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

boxplots['boxes'][-1].set_facecolor('#396651')
boxplots['boxes'][-2].set_facecolor('#7bb551')

ax[0].set_ylabel("Shannon\nalpha diversity", fontsize=14)
ax[0].axvline(19.5, color='gray', linestyle='-')
ax[0].set_xticklabels(labels)
ax[0].text(-0.08, 0.92, 'A', size=20, transform=ax[0].transAxes, weight='bold')

print('\nAlpha diversity - Birth vs. 1 day:')
summarize_ttest(alpha_divs[0], alpha_divs[1], simple=True)
print('\nAlpha diversity - 1 day vs 3 day:')
summarize_ttest(alpha_divs[1], alpha_divs[2], simple=True)
print('\nAlpha diversity - 3 day vs 1 year:')
summarize_ttest(alpha_divs[2], alpha_divs[-3], simple=True)

# Now polymorphism
desired_species = 'Escherichia_coli_58110'

polymorphisms = [] # list of sample values for each tp
labels = []; sample_sizes = []; subject_sizes = []

for tp in infant_tps_ordered:
    num_samples = len(mi_tp_sample_dict['infant'][tp])
    if num_samples < 10:
        continue # Skip timepoints with not enough data

    subjects = set()
    for sample in mi_tp_sample_dict['infant'][tp]:
        if desired_species in sample_species_polymorphism_dict[sample]:
            subject = sample_subject_map[sample]
            subjects.add(subject)

    if len(subjects) < 8:
        subjects_downsampled = subjects
    else:
        subjects_downsampled = np.random.choice(list(subjects), 8, replace=False)

    # Use downsampled subjects
    polymorphisms_tp = []
    for sample in mi_tp_sample_dict['infant'][tp]:
        if sample_subject_map[sample] in subjects_downsampled and desired_species in
sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][desired_species]
            polymorphisms_tp.append(polymorphism)

    labels.append(tp + "\n" + ("n=%i" % len(polymorphisms_tp)))
    polymorphisms.append(polymorphisms_tp) # sample_sizes.append(num_samples);
subject_sizes.append(len(subjects_downsampled))

# Mother ================================================================================
polymorphisms_mother_combined = []
mother_subjects = set()

for tp in mother_tps_ordered:
    if tp == -92 or tp == 92: # Skip 3month/-3month
        continue
```

```
    for sample in mi_tp_sample_dict['mother'][tp]:
        if desired_species in sample_species_polymorphism_dict[sample]:
            subject = sample_subject_map[sample]
            mother_subjects.add(subject)

mother_subjects_downsampled = np.random.choice(list(mother_subjects), 8, replace=False)

# Use downsampled subjects
polymorphisms_mother_combined = []
for tp in mother_tps_ordered:
    for sample in mi_tp_sample_dict['mother'][tp]:
        if sample_subject_map[sample] in mother_subjects_downsampled and desired_species in
sample_species_polymorphism_dict[sample]:
            polymorphism = sample_species_polymorphism_dict[sample][desired_species]
            polymorphisms_mother_combined.append(polymorphism)

labels.append('Mother'+ "\n" + ("n=%i" % len(polymorphisms_mother_combined)))
polymorphisms.append(polymorphisms_mother_combined)

# HMP ============================================================================================
hmp_subjects = set()
for sample in hmp_samples:
    if desired_species in sample_species_polymorphism_dict[sample]:
        hmp_subjects.add(sample_subject_map[sample])

hmp_subjects_downsampled = np.random.choice(list(hmp_subjects), 8, replace=False)

# Use downsampled subjects
polymorphisms_hmp = []
subjects_so_far = set()
for sample in hmp_samples:
    subject = sample_subject_map[sample]
    if desired_species in sample_species_polymorphism_dict[sample]:
        if subject in subjects_so_far:
            continue
        if subject in hmp_subjects_downsampled:
            polymorphisms_hmp.append(sample_species_polymorphism_dict[sample][desired_species])
            subjects_so_far.add(subject)

labels.append('HMP' + "\n" + ("n=%i" % len(polymorphisms_hmp)))
polymorphisms.append(polymorphisms_hmp)

# Plot ===========================================================================================
boxplots = ax[1].boxplot(polymorphisms, patch_artist=True, boxprops=boxprops,
medianprops=medianprops,flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

boxplots['boxes'][-1].set_facecolor('#396651')
boxplots['boxes'][-2].set_facecolor('#7bb551')

ax[1].axvline(x=19.5, color='gray')
ax[1].set_ylabel("Within-sample\npolymorphism", fontsize=14)
ax[1].set_xticklabels(labels)
ax[1].set_yscale('log')
ax[1].text(-0.08, 0.92, 'B', size=20, transform=ax[1].transAxes, weight='bold')

polymorphisms_before_4m = []
for polymorphisms_tp in polymorphisms[:10]:
    polymorphisms_before_4m += polymorphisms_tp
```

```
polymorphisms_after_incl_4m = []
for polymorphisms_tp in polymorphisms[10:-2]:
    polymorphisms_after_incl_4m += polymorphisms_tp

print('\nPolymorphism - before vs after/including 4 months:')
summarize_utest(polymorphisms_before_4m, polymorphisms_after_incl_4m, simple=True)

fig.savefig('%s/alpha_div_and_polymorphism_over_time_downsampled.pdf' %
(config.analysis_directory), bbox_inches='tight')
plt.show()


# In[67]:


# Freeze the data
pickle.dump(polymorphisms, open('%s/polymorphisms_downsampled_8_subjects.pkl' % pickle_dir,
'wb'))
pickle.dump(alpha_divs, open('%s/alpha_divs_downsampled_20_subjects.pkl' % pickle_dir, 'wb'))


# In[63]:


# Check statistics again without regenerating the data

print('\nAlpha diversity - Birth vs. 1 day:')
summarize_ttest(alpha_divs[0], alpha_divs[1], simple=True)
print('\nAlpha diversity - 1 day vs 3 day:')
summarize_ttest(alpha_divs[1], alpha_divs[2], simple=True)
print('\nAlpha diversity - 3 day vs 1 year:')
summarize_ttest(alpha_divs[2], alpha_divs[-3], simple=True)

polymorphisms_before_4m = []
for polymorphisms_tp in polymorphisms[:10]:
    polymorphisms_before_4m += polymorphisms_tp

polymorphisms_after_incl_4m = []
for polymorphisms_tp in polymorphisms[10:-2]:
    polymorphisms_after_incl_4m += polymorphisms_tp

print('\nPolymorphism - before vs after/including 4 months:')
summarize_utest(polymorphisms_before_4m, polymorphisms_after_incl_4m, simple=True)


# In[57]:


labels[10:-2]


# In[19]:


# Split E. coli polymorphism by dataset

# fig, ax = plt.subplots(1, 4, figsize=(18,4), sharey=True, constrained_layout=True,
#                        gridspec_kw={'width_ratios': [2, 3, 3, 3]})
fig, ax = plt.subplots(2,2, figsize=(12,7), sharey=True, constrained_layout=True)
desired_species = 'Escherichia_coli_58110'
```

```
# HMP1-2
hmp_samples = su.get_sample_names('HMP')
order_polymorphism_dict = defaultdict(list)
for sample in hmp_samples:
    subject, order = sample_order_map[sample]
    if desired_species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][desired_species]
        order_polymorphism_dict[order].append(polymorphism)

order_list = [1, 2, 3]
order_labels = ['HMP1-2:1', 'HMP1-2:2', 'HMP1-2:3']
order_labels_with_n = [label + '\n(n=%i)' % len(order_polymorphism_dict[o]) for label, o in
zip(order_labels, order_list)]

plot_data = [order_polymorphism_dict[o] for o in order_list]

ax[0][0].set_yscale('log')
boxplots = ax[0][0].boxplot(plot_data, patch_artist=True,
                boxprops=boxprops,
                medianprops=medianprops,
                flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
ax[0][0].set_xticklabels(order_labels_with_n)
ax[0][0].set_ylabel("Polymorphism")
ax[0][0].set_title("HMP1-2")

# Backhed
backhed_samples = su.get_sample_names('Backhed')
order_polymorphism_dict = defaultdict(list)
for sample in backhed_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples)
    if desired_species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][desired_species]
        order_polymorphism_dict[tp].append(polymorphism)

order_list = ['M1', 'I1', 'I2', 'I3']
order_labels = ['Mother', 'Birth (2-5 days)', '4 Months', '12 Months']
order_labels_with_n = [label + '\n(n=%i)' % len(order_polymorphism_dict[o]) for label, o in
zip(order_labels, order_list)]

plot_data = [order_polymorphism_dict[o] for o in order_list]

ax[0][1].set_yscale('log')
boxplots = ax[0][1].boxplot(plot_data, patch_artist=True,
                boxprops=boxprops,
                medianprops=medianprops,
                flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
ax[0][1].set_xticklabels(order_labels_with_n)
ax[0][1].set_title("Backhed")

# Ferretti
ferretti_samples = su.get_sample_names('Ferretti')
order_polymorphism_dict = defaultdict(list)
for sample in ferretti_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples)
    if desired_species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][desired_species]
        order_polymorphism_dict[tp].append(polymorphism)
```

```
order_list = ['M1', 'I1', 'I2', 'I3', 'I4', 'I5']
order_labels = ['Mother', '1 day', '3 days', '1 wk', '1 mon', '4 mon']
order_labels_with_n = [label + '\n(n=%i)' % len(order_polymorphism_dict[o]) for label, o in
zip(order_labels, order_list)]

plot_data = [order_polymorphism_dict[o] for o in order_list]

ax[1][0].set_yscale('log')
boxplots = ax[1][0].boxplot(plot_data, patch_artist=True,
                boxprops=boxprops,
                medianprops=medianprops,
                flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
ax[1][0].set_xticklabels(order_labels_with_n)
ax[1][0].set_title("Ferretti")

# Yassour
yassour_samples = su.get_sample_names('Yassour')
order_polymorphism_dict = defaultdict(list)
for sample in yassour_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples)
    if desired_species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][desired_species]
        order_polymorphism_dict[tp].append(polymorphism)

order_list = ['M1', 'M2', 'M3', 'I1', 'I2', 'I3', 'I4', 'I5']
order_labels = ['Mother:\nGest', 'Mother:\nDelivery','Mother:\n3 mon', 'Birth', '1 wk', '2 wk',
'1 mon', '2 mon', '3 mon']
order_labels_with_n = [label + '\n(n=%i)' % len(order_polymorphism_dict[o]) for label, o in
zip(order_labels, order_list)]

plot_data = [order_polymorphism_dict[o] for o in order_list]

ax[1][1].set_yscale('log')
boxplots = ax[1][1].boxplot(plot_data, patch_artist=True,
                boxprops=boxprops,
                medianprops=medianprops,
                flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')
ax[1][1].set_xticklabels(order_labels_with_n)
ax[1][1].set_title("Yassour")

plt.show()


# In[20]:


fig, ax = plt.subplots(figsize=(12, 3.6), sharey=True, constrained_layout=True)

# Shao
month_bins = np.arange(4, 15) * 30.5
order_bins = [0, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 21] + list(month_bins)

shao_samples = su.get_sample_names('Shao')
order_polymorphism_dict = defaultdict(list)
order_subjects_dict = defaultdict(set)
```

```
for sample in shao_samples:
    subject, order = sample_order_map[sample]
    if subject in order_subjects_dict[tp]: # Ensure one sample per subject for particular
timepoint bin
        continue

    if desired_species in sample_species_polymorphism_dict[sample]:
        polymorphism = sample_species_polymorphism_dict[sample][desired_species]
    else:
        continue
    if subject[-1] == 'M': # Mother
        order_polymorphism_dict[0].append(polymorphism)
    elif order > 21: # Convert to month bins
        month_bin = round_down(order, 30.5)
        order_polymorphism_dict[month_bin].append(polymorphism)
    else:
        order_polymorphism_dict[order].append(polymorphism)

    order_subjects_dict[tp].add(subject)

plot_data = [order_polymorphism_dict[o] for o in order_bins if len(order_polymorphism_dict[o]) >
5]
order_labels = ['Mother', '4d', '6d', '7d', '8d', '9d', '10d', '11d', '12d', '13d', '14d',
'17d', '18d', '21d', '4m', '5m', '6m', '7m', '8m', '9m', '10m', '11m', '12m', '13m', '14m']
order_labels_with_n = [label + '\n(n=%i)' % len(order_polymorphism_dict[o]) for label, o in
zip(order_labels, order_bins) if len(order_polymorphism_dict[o]) > 5]

ax.set_yscale('log')
boxplots = ax.boxplot(plot_data, patch_artist=True,
                  boxprops=boxprops,
                  medianprops=medianprops,
                  flierprops=flierprops)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

ax.set_xticklabels(order_labels_with_n)
ax.set_ylabel("Polymorphism")
ax.set_title("Shao")


# In[8]:


# Now that Backhed has more granular timepoints, reevaluate distribution of cohorts by timepoint
days = sorted(list(set(mi_sample_day_dict.values())))
tp_cohort_sample_count_dict = {day: defaultdict(int) for day in days}
for sample in mi_sample_day_dict:
    if sample not in infant_samples:
        continue
    day = mi_sample_day_dict[sample]
    cohort = sample_cohort_map[sample]
    tp_cohort_sample_count_dict[day][cohort] += 1

for day in days:
    if len(tp_cohort_sample_count_dict[day]) > 1 and
sum(tp_cohort_sample_count_dict[day].values()) > 15:
        print("%i ====================================" % day)
        for cohort in tp_cohort_sample_count_dict[day]:
            print("\t%s: %i" % (cohort, tp_cohort_sample_count_dict[day][cohort]))
```

```
# In[8]:


# Compare datasets

desired_species = 'Escherichia_coli_58110'

cohorts = ['hmp', 'backhed', 'ferretti', 'yassour', 'shao']
cohort_tp_polymorphism_dict = {cohort: defaultdict(list) for cohort in cohorts}

for sample in sample_cohort_map:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples_orig)
    if sample_cohort_map[sample] == 'backhed' and mi_sample_day_dict[sample] == 7:
        tp = '7d'
    cohort = sample_cohort_map[sample]
    if cohort == 'shao':
        subject, order = sample_order_map[sample]
        if subject[-1] == 'M': # Mother
            tp = 0
        elif order > 21: # Convert to month bins
            tp = round_down(order, 30.5)
        else:
            tp = order
    if cohort not in cohorts:
        continue

    if desired_species in sample_species_polymorphism_dict[sample]:

cohort_tp_polymorphism_dict[cohort][tp].append(sample_species_polymorphism_dict[sample][desired_
species])

fig, ax = plt.subplots(1, 4, figsize=(11, 4),
                       sharey=True,
                       gridspec_kw={'width_ratios': [3, 3, 2, 2]})

# 1 week (Ferretti / Yassour / Shao / Backhed)
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]# , ('backhed', '7d')]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[0].set_yscale('log')
boxplots = ax[0].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1), (1, 2), (0, 2)], [0.8, 0.15, 1e-6], [0.18, 0.036, 3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[0], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[0].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[0].set_title('1 week')
```

```
ax[0].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])
ax[0].set_ylabel('Polymorphism rate')
'''
# 2 weeks
cohort_tp_tups = [('yassour', 'I3'), ('shao',14)]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[1].set_yscale('log')
boxplots = ax[1].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1)], [1e-6], [3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[1], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[1].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[1].set_title('2 weeks')
ax[1].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp])) \
                       for (cohort, tp) in cohort_tp_tups])
'''
# 1 month
cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[1].set_yscale('log')
boxplots = ax[1].boxplot(plot_data, patch_artist=True, widths=0.6, boxprops=dict(color='black'),
medianprops=dict(color='black'), flierprops=dict(marker='.'))
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1), (1, 2), (0, 2)], [0.8, 0.15, 1e-6], [0.18, 0.036, 3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[1], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[1].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[1].set_title('1 month')
ax[1].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])

# 4 months
cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
```

```
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[2].set_yscale('log')
boxplots = ax[2].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j in zip([(0, 1)], [1e-6]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[2], j, i1+1, i2+1, color=color, tickh=3e-7)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[2].text(i1+1+((i2-i1)/2)-offset, j+3e-7, label, color=color)

ax[2].set_title('4 months')
ax[2].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])

# 12 months
a = []
for tp in cohort_tp_polymorphism_dict['shao']:
    if tp > 300:
        a += cohort_tp_polymorphism_dict['shao'][tp]
b = cohort_tp_polymorphism_dict['backhed']['I3']
plot_data = [a, b]

ax[3].set_yscale('log')
boxplots = ax[3].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j in zip([(0, 1)], [1e-6]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[3], j, i1+1, i2+1, color=color, tickh=3e-7)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[3].text(i1+1+((i2-i1)/2)-offset, j+3e-7, label, color=color)

ax[3].set_title('12 months')
ax[3].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(), len(vals))                          for
(cohort, vals) in zip(['shao', 'backhed'], [a, b])])
ax[0].set_ylim(top=5)

fig.savefig('%s/polymorphism_compare_cohort_matched_tp.pdf' % (config.analysis_directory),
bbox_inches='tight')
```

```
# In[10]:


# 1 week (Ferretti / Yassour / Shao / Backhed) - do Backhed comparisons
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7), ('backhed', '7d')]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

for i1, i2 in [(0, 3), (1, 3), (2, 3)]:
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))


# In[21]:


# Compare datasets

desired_species = 'Escherichia_coli_58110'

cohorts = ['hmp', 'backhed', 'ferretti', 'yassour', 'shao']
cohort_tp_polymorphism_dict = {cohort: defaultdict(list) for cohort in cohorts} # Now tp is a
day (for infants only)

for sample in sample_cohort_map:
    if sample not in infant_samples:
        continue
    day = mi_sample_day_dict[sample]
    cohort = sample_cohort_map[sample]
    if desired_species in sample_species_polymorphism_dict[sample]:

cohort_tp_polymorphism_dict[cohort][day].append(sample_species_polymorphism_dict[sample][desired
_species])

fig, ax = plt.subplots(1, 3, figsize=(10, 4),
                       sharey=True,
                       gridspec_kw={'width_ratios': [3, 2, 2]})

# 1 day (Ferretti / Backhed) also throw in Yassour birth for good measure
cohort_tp_tups = [('yassour', 0), ('ferretti', 1), ('backhed', 1)]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[0].set_yscale('log')
boxplots = ax[0].boxplot(plot_data, patch_artist=True, boxprops=boxprops,
medianprops=medianprops, flierprops=flierprops, widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1), (1, 2), (0, 2)], [0.8, 0.15, 1e-6], [0.18, 0.036, 3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[0], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[0].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[0].set_title('1 day')
```

```
ax[0].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])
ax[0].set_ylabel('Polymorphism rate')

# 3 days (Ferretti / Backhed)
cohort_tp_tups = [('ferretti', 3), ('backhed', 3)]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[1].set_yscale('log')
boxplots = ax[1].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1)], [1e-6], [3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[1], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[1].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[1].set_title('3 days')
ax[1].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])

# 4 days (Backhed / Yassour)
cohort_tp_tups = [('shao', 4), ('backhed', 4)]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[2].set_yscale('log')
boxplots = ax[2].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1)], [1e-6], [3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[2], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[2].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[2].set_title('4 days')
ax[2].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])
```

```
# In[14]:


# Combine matched timepoint polymorphism rate comparion

# Compare datasets

desired_species = 'Escherichia_coli_58110'

cohorts = ['hmp', 'backhed', 'ferretti', 'yassour', 'shao']
cohort_tp_polymorphism_dict = {cohort: defaultdict(list) for cohort in cohorts} # Now tp is a
day (for infants only)

for sample in sample_cohort_map:
    if sample not in infant_samples:
        continue
    day = mi_sample_day_dict[sample]
    cohort = sample_cohort_map[sample]
    if desired_species in sample_species_polymorphism_dict[sample]:

cohort_tp_polymorphism_dict[cohort][day].append(sample_species_polymorphism_dict[sample][desired
_species])

fig, ax = plt.subplots(1, 7, figsize=(20, 4),
                       sharey=True,
                       gridspec_kw={'width_ratios': [3, 2, 2, 3, 3, 2, 2]})

# 1 day (Ferretti / Backhed) also throw in Yassour birth for good measure
i = 0
cohort_tp_tups = [('yassour', 0), ('ferretti', 1), ('backhed', 1)]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
plot_data[2] += cohort_tp_polymorphism_dict['backhed'][0] # Backhed actually both 0 and 1 days

ax[i].set_yscale('log')
boxplots = ax[i].boxplot(plot_data, patch_artist=True, boxprops=boxprops,
medianprops=medianprops, flierprops=flierprops, widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1), (1, 2), (0, 2)], [0.8, 0.15, 1e-6], [0.18, 0.036, 3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[i], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[i].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[i].set_title('1 day')
ax[i].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])
ax[i].set_ylabel('Polymorphism rate')

# 3 days (Ferretti / Backhed)
i += 1
cohort_tp_tups = [('ferretti', 3), ('backhed', 3)]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
```

```
ax[i].set_yscale('log')
boxplots = ax[i].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1)], [1e-6], [3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[i], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[i].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[i].set_title('3 days')
ax[i].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                         for (cohort, tp) in
cohort_tp_tups])

# 4 days (Backhed / Yassour)
i += 1
cohort_tp_tups = [('shao', 4), ('backhed', 4)]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[i].set_yscale('log')
boxplots = ax[i].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1)], [1e-6], [3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[i], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[i].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[i].set_title('4 days')
ax[i].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                         for (cohort, tp) in
cohort_tp_tups])

# Later timepoints
cohort_tp_polymorphism_dict = {cohort: defaultdict(list) for cohort in cohorts}

for sample in sample_cohort_map:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples_orig)
    if sample_cohort_map[sample] == 'backhed' and mi_sample_day_dict[sample] == 7:
        tp = '7d'
```

```
    cohort = sample_cohort_map[sample]
    if cohort == 'shao':
        subject, order = sample_order_map[sample]
        if subject[-1] == 'M': # Mother
            tp = 0
        elif order > 21: # Convert to month bins
            tp = round_down(order, 30.5)
        else:
            tp = order
    if cohort not in cohorts:
        continue

    if desired_species in sample_species_polymorphism_dict[sample]:

cohort_tp_polymorphism_dict[cohort][tp].append(sample_species_polymorphism_dict[sample][desired_
species])

# 1 week (Ferretti / Yassour / Shao / Backhed)
i += 1
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]# , ('backhed', '7d')]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[i].set_yscale('log')
boxplots = ax[i].boxplot(plot_data, patch_artist=True,
                boxprops=dict(color='black'),
                medianprops=dict(color='black'),
                flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1), (1, 2), (0, 2)], [0.8, 0.15, 1e-6], [0.18, 0.036, 3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[i], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[i].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[i].set_title('1 week')
ax[i].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                        for (cohort, tp) in
cohort_tp_tups])

# 1 month
i += 1
cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[i].set_yscale('log')
boxplots = ax[i].boxplot(plot_data, patch_artist=True, widths=0.6, boxprops=dict(color='black'),
medianprops=dict(color='black'), flierprops=dict(marker='.'))
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j, tick in zip([(0, 1), (1, 2), (0, 2)], [0.8, 0.15, 1e-6], [0.18, 0.036, 3e-7]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
```

```
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[i], j, i1+1, i2+1, color=color, tickh=tick)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[i].text(i1+1+((i2-i1)/2)-offset, j+tick, label, color=color)

ax[i].set_title('1 month')
ax[i].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])

# 4 months
i += 1
cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]

ax[i].set_yscale('log')
boxplots = ax[i].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j in zip([(0, 1)], [1e-6]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[i], j, i1+1, i2+1, color=color, tickh=3e-7)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[i].text(i1+1+((i2-i1)/2)-offset, j+3e-7, label, color=color)

ax[i].set_title('4 months')
ax[i].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(),
len(cohort_tp_polymorphism_dict[cohort][tp]))                          for (cohort, tp) in
cohort_tp_tups])

# 12 months
i += 1
a = []
for tp in cohort_tp_polymorphism_dict['shao']:
    if tp > 300:
        a += cohort_tp_polymorphism_dict['shao'][tp]
b = cohort_tp_polymorphism_dict['backhed']['I3']
plot_data = [a, b]

ax[i].set_yscale('log')
boxplots = ax[i].boxplot(plot_data, patch_artist=True,
                 boxprops=dict(color='black'),
                 medianprops=dict(color='black'),
                 flierprops=dict(marker='.'), widths=0.6)
for patch in boxplots['boxes']:
    patch.set_facecolor('#77acff')

for itup, j in zip([(0, 1)], [1e-6]):
    i1, i2 = itup
    U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
```

```
    print("Effect size: %s" % (MWU_effect_size(p, len(plot_data[i1]), len(plot_data[i2]))))
    print("p: %s" % str(p))
    color = 'red' if p < 0.05 else 'gray'
    plot_interval_on_ax(ax[i], j, i1+1, i2+1, color=color, tickh=3e-7)
    label = '{:.2e}'.format(p) if p < 0.01 else ('%.03f' % p)
    offset = 0.36 if p < 0.01 else 0.23
    ax[i].text(i1+1+((i2-i1)/2)-offset, j+3e-7, label, color=color)

ax[i].set_title('12 months')
ax[i].set_xticklabels(['%s\nn=%i' % (cohort.capitalize(), len(vals))                          for
(cohort, vals) in zip(['shao', 'backhed'], [a, b])])
ax[i].set_ylim(top=5)

plt.subplots_adjust(wspace=0.1)

fig.savefig('%s/polymorphism_compare_cohort_matched_tp_v2.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[21]:


# Now report statistics for all species
# Compare datasets
# Also store results in a table

f = open('%s/matched_timepoint_polymorphism_dataset_sig_tests.csv' % config.analysis_directory,
'w')
f.write(','.join(['species', 'timepoint', 'cohort1', 'n1', 'cohort2', 'n2', 'U', 'p']) + '\n')

cohorts = ['hmp', 'backhed', 'ferretti', 'yassour', 'shao']

min_sample_size = 10

total_total_number_tests = 0
total_total_sig_tests = 0

orig_p = 0.05
correction_factor = 18

for desired_species in good_species_list[:13]:
    print("============================================================")
    print(desired_species)
    print("============================================================")

    total_number_tests = 0
    total_sig_tests = 0

    cohort_tp_polymorphism_dict = {cohort: defaultdict(list) for cohort in cohorts} # Now tp is
a day (for infants only)
    for sample in sample_cohort_map:
        if sample not in infant_samples:
            continue
        day = mi_sample_day_dict[sample]
        cohort = sample_cohort_map[sample]
        if desired_species in sample_species_polymorphism_dict[sample]:

cohort_tp_polymorphism_dict[cohort][day].append(sample_species_polymorphism_dict[sample][desired
_species])

    # 1 day (Ferretti / Backhed)
```

```
    cohort_tp_tups = [('ferretti', 1), ('backhed', 1)]
    plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
    for i1, i2 in [(0, 1)]:
        if len(plot_data[i1]) >= min_sample_size and len(plot_data[i2]) >= min_sample_size:
            U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
            total_number_tests += 1
            if p < (orig_p/correction_factor):
                total_sig_tests += 1
                print("1 day/birth: %s vs. %s: p=%s" % (str(cohort_tp_tups[i1][0]),
str(cohort_tp_tups[i2][0]), str(p)))
        else:
            U = 'NA'; p = 'NA'
        f.write(','.join([str(x) for x in [desired_species, '1 day', cohort_tp_tups[i1][0],
len(plot_data[i1]),
                                           cohort_tp_tups[i2][0], len(plot_data[i2]), U, p]]) +
'\n')

    # 3 days (Ferretti / Backhed)
    cohort_tp_tups = [('ferretti', 3), ('backhed', 3)]
    plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
    for i1, i2 in [(0, 1)]:
        if len(plot_data[i1]) >= min_sample_size and len(plot_data[i2]) >= min_sample_size:
            U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
            total_number_tests += 1
            if p < (orig_p/correction_factor):
                total_sig_tests += 1
                print("3 days: %s vs. %s: p=%s" % (str(cohort_tp_tups[i1][0]),
str(cohort_tp_tups[i2][0]), str(p)))
        else:
            U = 'NA'; p = 'NA'
        f.write(','.join([str(x) for x in [desired_species, '3 days', cohort_tp_tups[i1][0],
len(plot_data[i1]),
                                           cohort_tp_tups[i2][0], len(plot_data[i2]), U, p]]) +
'\n')

    # 4 days (Backhed / Yassour)
    cohort_tp_tups = [('shao', 4), ('backhed', 4)]
    plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
    for i1, i2 in [(0, 1)]:
        if len(plot_data[i1]) >= min_sample_size and len(plot_data[i2]) >= min_sample_size:
            U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
            total_number_tests += 1
            if p < (orig_p/correction_factor):
                total_sig_tests += 1
                print("4 days: %s vs. %s: p=%s" % (str(cohort_tp_tups[i1][0]),
str(cohort_tp_tups[i2][0]), str(p)))
        else:
            U = 'NA'; p = 'NA'
        f.write(','.join([str(x) for x in [desired_species, '4 days', cohort_tp_tups[i1][0],
len(plot_data[i1]),
                                           cohort_tp_tups[i2][0], len(plot_data[i2]), U, p]]) +
'\n')

    # Reset
    cohort_tp_polymorphism_dict = {cohort: defaultdict(list) for cohort in cohorts}

    for sample in sample_cohort_map:
        tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples_orig)
        if sample_cohort_map[sample] == 'backhed' and mi_sample_day_dict[sample] == 7:
            tp = '7d'
        cohort = sample_cohort_map[sample]
```

```
        if cohort == 'shao':
            subject, order = sample_order_map[sample]
            if subject[-1] == 'M': # Mother
                tp = 0
            elif order > 21: # Convert to month bins
                tp = round_down(order, 30.5)
            else:
                tp = order
        if cohort not in cohorts:
            continue

        if desired_species in sample_species_polymorphism_dict[sample]:

cohort_tp_polymorphism_dict[cohort][tp].append(sample_species_polymorphism_dict[sample][desired_
species])

    # 1 week (Ferretti / Yassour / Shao)
    cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]
    plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
    for i1, i2 in [(0, 1), (1, 2), (0, 2)]:
        if len(plot_data[i1]) >= min_sample_size and len(plot_data[i2]) >= min_sample_size:
            U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
            total_number_tests += 1
            if p < (orig_p/correction_factor):
                total_sig_tests += 1
                print("1 week: %s vs. %s: p=%s" % (str(cohort_tp_tups[i1][0]),
str(cohort_tp_tups[i2][0]), str(p)))
        else:
            U = 'NA'; p = 'NA'
        f.write(','.join([str(x) for x in [desired_species, '1 week', cohort_tp_tups[i1][0],
len(plot_data[i1]),
                                           cohort_tp_tups[i2][0], len(plot_data[i2]), U, p]]) +
'\n')

    # 2 weeks
    cohort_tp_tups = [('yassour', 'I3'), ('shao',14)]
    plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
    for i1, i2 in [(0, 1)]:
        if len(plot_data[i1]) >= min_sample_size and len(plot_data[i2]) >= min_sample_size:
            U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
            total_number_tests += 1
            if p < (orig_p/correction_factor):
                total_sig_tests += 1
                print("2 weeks: %s vs. %s: p=%s" % (str(cohort_tp_tups[i1][0]),
str(cohort_tp_tups[i2][0]), str(p)))
        else:
            U = 'NA'; p = 'NA'
        f.write(','.join([str(x) for x in [desired_species, '2 weeks', cohort_tp_tups[i1][0],
len(plot_data[i1]),
                                           cohort_tp_tups[i2][0], len(plot_data[i2]), U, p]]) +
'\n')

    # 1 month
    cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
    plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
    for i1, i2 in [(0, 1), (1, 2), (0, 2)]:
        if len(plot_data[i1]) >= min_sample_size and len(plot_data[i2]) >= min_sample_size:
            U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
            total_number_tests += 1
            if p < (orig_p/correction_factor):
                total_sig_tests += 1
```

```
                print("1 month: %s vs. %s: p=%s" % (str(cohort_tp_tups[i1][0]),
str(cohort_tp_tups[i2][0]), str(p)))
        else:
            U = 'NA'; p = 'NA'
        f.write(','.join([str(x) for x in [desired_species, '1 month', cohort_tp_tups[i1][0],
len(plot_data[i1]),
                                           cohort_tp_tups[i2][0], len(plot_data[i2]), U, p]]) +
'\n')

    # 4 months
    cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
    plot_data = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
    for i1, i2 in [(0, 1)]:
        if len(plot_data[i1]) >= min_sample_size and len(plot_data[i2]) >= min_sample_size:
            U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
            total_number_tests += 1
            if p < (orig_p/correction_factor):
                total_sig_tests += 1
                print("4 months: %s vs. %s: p=%s" % (str(cohort_tp_tups[i1][0]),
str(cohort_tp_tups[i2][0]), str(p)))
        else:
            U = 'NA'; p = 'NA'
        f.write(','.join([str(x) for x in [desired_species, '4 months', cohort_tp_tups[i1][0],
len(plot_data[i1]),
                                           cohort_tp_tups[i2][0], len(plot_data[i2]), U, p]]) +
'\n')

    # 12 months
    a = []
    for tp in cohort_tp_polymorphism_dict['shao']:
        if tp > 300:
            a += cohort_tp_polymorphism_dict['shao'][tp]
    b = cohort_tp_polymorphism_dict['backhed']['I3']
    plot_data = [a, b]
    for i1, i2 in [(0, 1)]:
        if len(plot_data[i1]) >= min_sample_size and len(plot_data[i2]) >= min_sample_size:
            U, p = scipy.stats.mannwhitneyu(plot_data[i1], plot_data[i2])
            total_number_tests += 1
            if p < (orig_p/correction_factor):
                total_sig_tests += 1
                print("12 months: %s vs. %s: p=%s" % (str(cohort_tp_tups[i1][0]),
str(cohort_tp_tups[i2][0]), str(p)))
        else:
            U = 'NA'; p = 'NA'
        f.write(','.join([str(x) for x in [desired_species, '12 months', cohort_tp_tups[i1][0],
len(plot_data[i1]),
                                           cohort_tp_tups[i2][0], len(plot_data[i2]), U, p]]) +
'\n')

    print("**** %i/%i tests were significant" % (total_sig_tests, total_number_tests))
    total_total_number_tests += total_number_tests
    total_total_sig_tests += total_sig_tests

f.close()
print("GRAND TOTAL: %i/%i tests were significant" % (total_total_sig_tests,
total_total_number_tests))


# In[ ]:
```

```
# Make a table with species, number of tests, number of significant tests, p-values (corrected)
# Write up about these tests


# In[32]:


U, p1 = scipy.stats.mannwhitneyu(plot_data[0][:-2], plot_data[1])
W, p2 = scipy.stats.wilcoxon(plot_data[0][:-2], plot_data[1])
print("U: %s" % str(U))
print("p: %s" % str(p1))
print("W: %s" % str(W))
print("p: %s" % str(p2))


# In[24]:


from scipy import stats

fig, ax = plt.subplots(3, 4, figsize=(9, 9),sharey=True, sharex=True)
xs = np.arange(-3, 3, 0.01)

# 1 week
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]
x = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
vals = stats.probplot(x[0]); ax[0][0].set_title("Ferretti - 1wk")
osm, osr = vals[0]; m, b, r = vals[1]; ax[0][0].plot(xs, (m*xs)+b, color='gray');
ax[0][0].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[1]); ax[0][1].set_title("Yassour - 1wk")
osm, osr = vals[0]; m, b, r = vals[1]; ax[0][1].plot(xs, (m*xs)+b, color='gray');
ax[0][1].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[2]); ax[0][2].set_title("Shao - 1wk")
osm, osr = vals[0]; m, b, r = vals[1]; ax[0][2].plot(xs, (m*xs)+b, color='gray');
ax[0][2].plot(osm, osr, '.', color='black', mfc='none')

# 2 weeks
cohort_tp_tups = [('yassour', 'I3'), ('shao',14)]
x = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
vals = stats.probplot(x[0]); ax[0][3].set_title("Yassour - 2wk")
osm, osr = vals[0]; m, b, r = vals[1]; ax[0][3].plot(xs, (m*xs)+b, color='gray');
ax[0][3].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[1]); ax[1][0].set_title("Shao - 2wk")
osm, osr = vals[0]; m, b, r = vals[1]; ax[1][0].plot(xs, (m*xs)+b, color='gray');
ax[1][0].plot(osm, osr, '.', color='black', mfc='none')

# 1 month
cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
x = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
vals = stats.probplot(x[0]); ax[1][1].set_title("Ferretti - 1mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[1][1].plot(xs, (m*xs)+b, color='gray');
ax[1][1].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[1]); ax[1][2].set_title("Yassour - 1mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[1][2].plot(xs, (m*xs)+b, color='gray');
ax[1][2].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[2]); ax[1][3].set_title("Shao - 21d")
osm, osr = vals[0]; m, b, r = vals[1]; ax[1][3].plot(xs, (m*xs)+b, color='gray');
ax[1][3].plot(osm, osr, '.', color='black', mfc='none')

# 4 months
cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
```

```
x = [cohort_tp_polymorphism_dict[cohort][tp] for (cohort, tp) in cohort_tp_tups]
vals = stats.probplot(x[0]); ax[2][0].set_title("Ferretti - 4mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[2][0].plot(xs, (m*xs)+b, color='gray');
ax[2][0].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[1]); ax[2][1].set_title("Backhed - 4mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[2][1].plot(xs, (m*xs)+b, color='gray');
ax[2][1].plot(osm, osr, '.', color='black', mfc='none')

# 12 months
a = []
for tp in cohort_tp_polymorphism_dict['shao']:
    if tp > 300:
        a += cohort_tp_polymorphism_dict['shao'][tp]
b = cohort_tp_polymorphism_dict['backhed']['I3']
x = [a, b]
vals = stats.probplot(x[0]); ax[2][2].set_title("Shao - 12+mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[2][2].plot(xs, (m*xs)+b, color='gray');
ax[2][2].plot(osm, osr, '.', color='black', mfc='none')
vals = stats.probplot(x[1]); ax[2][3].set_title("Backhed - 12mo")
osm, osr = vals[0]; m, b, r = vals[1]; ax[2][3].plot(xs, (m*xs)+b, color='gray');
ax[2][3].plot(osm, osr, '.', color='black', mfc='none')

for i in range(3):
    ax[i][0].set_ylabel("Observed")

for i in range(4):
    ax[2][i].set_xlabel("Theoretical quantile")


# In[62]:


# Store data for R Q-Q plots

# 1 week
cohort_tp_tups = [('ferretti', 'I3'), ('yassour', 'I2'), ('shao',7)]
for (cohort, tp) in cohort_tp_tups:
    polymorphisms = cohort_tp_polymorphism_dict[cohort][tp]
    with open('%s/polymorphisms_%s_%s.txt' % (config.analysis_directory, cohort, '1wk'), 'w') as
f:
        f.write('\n'.join([str(p) for p in polymorphisms]))

# 2 weeks
cohort_tp_tups = [('yassour', 'I3'), ('shao',14)]
for (cohort, tp) in cohort_tp_tups:
    polymorphisms = cohort_tp_polymorphism_dict[cohort][tp]
    with open('%s/polymorphisms_%s_%s.txt' % (config.analysis_directory, cohort, '2wk'), 'w') as
f:
        f.write('\n'.join([str(p) for p in polymorphisms]))

# 1 month
cohort_tp_tups = [('ferretti', 'I4'), ('yassour', 'I4'), ('shao',21)]
for (cohort, tp) in cohort_tp_tups:
    polymorphisms = cohort_tp_polymorphism_dict[cohort][tp]
    with open('%s/polymorphisms_%s_%s.txt' % (config.analysis_directory, cohort, '1mo'), 'w') as
f:
        f.write('\n'.join([str(p) for p in polymorphisms]))

# 4 months
cohort_tp_tups = [('ferretti', 'I5'), ('backhed', 'I2')]
for (cohort, tp) in cohort_tp_tups:
```

```
    polymorphisms = cohort_tp_polymorphism_dict[cohort][tp]
    with open('%s/polymorphisms_%s_%s.txt' % (config.analysis_directory, cohort, '4mo'), 'w') as
f:
        f.write('\n'.join([str(p) for p in polymorphisms]))

# 12 months
a = []
for tp in cohort_tp_polymorphism_dict['shao']:
    if tp > 300:
        a += cohort_tp_polymorphism_dict['shao'][tp]
b = cohort_tp_polymorphism_dict['backhed']['I3']
for cohort, polymorphisms in zip(['shao', 'backhed'], [a, b]):
    with open('%s/polymorphisms_%s_%s.txt' % (config.analysis_directory, cohort, '1yr'), 'w') as
f:
        f.write('\n'.join([str(p) for p in polymorphisms]))


# In[55]:


def flatten(list_of_lists):
    flattened_list = []
    for l in list_of_lists:
        flattened_list += l
    return flattened_list


# In[61]:


# Compare before vs. after 4 months

desired_species = 'Escherichia_coli_58110'

cohorts = ['hmp', 'backhed', 'ferretti', 'yassour', 'shao']
cohort_tp_polymorphism_dict = {cohort: defaultdict(list) for cohort in cohorts}

for sample in sample_cohort_map:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples_orig)
    cohort = sample_cohort_map[sample]
    if cohort == 'shao':
        subject, order = sample_order_map[sample]
        if subject[-1] == 'M': # Mother
            tp = 0
        elif order > 21: # Convert to month bins
            tp = round_down(order, 30.5)
        else:
            tp = order
    if cohort not in cohorts:
        continue

    if desired_species in sample_species_polymorphism_dict[sample]:

cohort_tp_polymorphism_dict[cohort][tp].append(sample_species_polymorphism_dict[sample][desired_
species])

backhed_pre4m = flatten([cohort_tp_polymorphism_dict['backhed'][tp] for tp in ['I1', 'I2']])
backhed_post4m = flatten([cohort_tp_polymorphism_dict['backhed'][tp] for tp in ['I3']])
summarize_ttest(backhed_pre4m, backhed_post4m, simple=False)
```

```
ferretti_pre4m = flatten([cohort_tp_polymorphism_dict['ferretti'][tp] for tp in ['I1', 'I2',
'I3', 'I4', 'I5']])
ferretti_post4m = flatten([cohort_tp_polymorphism_dict['ferretti'][tp] for tp in []])
summarize_ttest(ferretti_pre4m, ferretti_post4m, simple=False)

yassour_pre4m = flatten([cohort_tp_polymorphism_dict['yassour'][tp] for tp in ['I1', 'I2', 'I3',
'I4']])
yassour_post4m = flatten([cohort_tp_polymorphism_dict['yassour'][tp] for tp in []])
summarize_ttest(yassour_pre4m, yassour_post4m, simple=False)

shao_pre4m = flatten([cohort_tp_polymorphism_dict['shao'][tp] for tp in
cohort_tp_polymorphism_dict['shao'] if tp <= 122])
shao_post4m = flatten([cohort_tp_polymorphism_dict['shao'][tp] for tp in
cohort_tp_polymorphism_dict['shao'] if tp > 122])
summarize_ttest(shao_pre4m, shao_post4m, simple=False)

# Overall
summarize_ttest(backhed_pre4m + ferretti_pre4m + yassour_pre4m + shao_pre4m, backhed_post4m +
shao_post4m, simple=False)

# fig.savefig('%s/polymorphism_compare_cohort_matched_tp.pdf' % (config.analysis_directory),
bbox_inches='tight')


# # QP Stuff

# In[7]:


# Grab QP samples for all species
cohorts = ['hmp', 'backhed', 'ferretti', 'yassour', 'shao']
samples = {cohort: su.get_sample_names(cohort) for cohort in cohorts}
sample_species_qp_dict = defaultdict(dict)

for species in good_species_list:
    for cohort in cohorts:
        qp_sample_sets = su.load_qp_samples(samples[cohort], species, prev_cohort='all')
        for qp_status in ['qp', 'non-qp', 'low-coverage']:
            for sample in qp_sample_sets[qp_status]:
                sample_species_qp_dict[sample][species] = qp_status


# In[8]:


# These custom cohorts are for a single sample

custom_cohorts_ordered = ['Day 0-Week 1', 'Week 1-Month 1', 'Month 1-Year 1', 'Mother', 'HMP
(Adult)']

def get_custom_cohort(cat, sample):
    if cat == 'infant':
        day = mi_sample_day_dict[sample]
        if day >= 0 and day <= 7:
            return 'Day 0-Week 1'
        elif day <= 32:
            return 'Week 1-Month 1'
        elif day <= 366:
            return 'Month 1-Year 1'
    elif cat == 'hmp':
        return 'HMP (Adult)'
```

```
    elif cat == 'mother':
        return 'Mother'


# In[9]:


cat_sample_species_qp_dict = {cat: defaultdict(dict) for cat in ['mother', 'infant', 'hmp']}
for sample in sample_species_qp_dict:
    if sample in infant_samples:
        cat = 'infant'
    elif sample in mother_samples:
        cat = 'mother'
    elif sample in hmp_samples:
        cat = 'hmp'
    else:
        pass # Should only be Yassour +/- 3 months
    for species in sample_species_qp_dict[sample]:
        cat_sample_species_qp_dict[cat][sample][species] =
sample_species_qp_dict[sample][species]

sample_num_qp_dict = {cat: defaultdict(int) for cat in ['mother', 'infant', 'hmp']}
sample_num_non_dict = {cat: defaultdict(int) for cat in ['mother', 'infant', 'hmp']}
sample_num_lowcov_dict = {cat: defaultdict(int) for cat in ['mother', 'infant', 'hmp']}

for cat in ['mother', 'infant', 'hmp']:
    for sample in cat_sample_species_qp_dict[cat]:
        for species in cat_sample_species_qp_dict[cat][sample]:
            qp_status = cat_sample_species_qp_dict[cat][sample][species]
            if qp_status == 'qp':
                sample_num_qp_dict[cat][sample] += 1
            elif qp_status == 'non-qp':
                sample_num_non_dict[cat][sample] += 1
            elif qp_status == 'low-coverage':
                sample_num_lowcov_dict[cat][sample] += 1


props_qp_by_dataset_by_cohort = {}
datasets = ['backhed', 'ferretti', 'yassour', 'shao', 'hmp']
dataset_samples = {dataset: su.get_sample_names(dataset) for dataset in datasets}

for dataset in datasets:
    props_qp_by_cohort = defaultdict(list)
    for cat in sample_num_qp_dict:
        for sample in sample_num_qp_dict[cat]:

            if sample not in dataset_samples[dataset]: # Restrict to individual dataset
                continue

            custom_cohort = get_custom_cohort(cat, sample)

            # Make sure we're only including mother at delivery
            if custom_cohort == 'Mother' and (mi_sample_day_dict[sample] < -1 or
mi_sample_day_dict[sample] > 30):
                    continue

            num_qp = (sample_num_qp_dict[cat][sample])
            num_non = (sample_num_non_dict[cat][sample])
            num_lowcov = (sample_num_lowcov_dict[cat][sample])
            prop_qp = num_qp/float(num_qp+num_non)
```

```
            props_qp_by_cohort[custom_cohort].append(prop_qp)

    props_qp_by_dataset_by_cohort[dataset] = props_qp_by_cohort


# In[44]:


# Analyze relationship between QP status and read count

# For each sample species pair, report: QP status, read count, host, dataset
output_file = open('%s/sample_qp_read_count.csv' % config.analysis_directory, 'w')
output_file.write(','.join(['species', 'sample', 'subject', 'qp_status', 'read_count',
'cohort']) + '\n')
sample_read_count_map = su.parse_sample_read_count_map()

for cohort in cohorts:
    for sample in samples[cohort]:
        subject = sample_subject_map[sample]
        read_count = sample_read_count_map[sample]
        for species in good_species_list:
            qp_status = sample_species_qp_dict[sample][species]
            output_file.write(','.join([species, sample, subject, qp_status, str(read_count),
cohort]) + '\n')

output_file.close()


# In[10]:


import matplotlib.cm as cmx

def calculate_unnormalized_survival_from_vector(counts):
    counts = sorted(counts)
    xs = [0]
    ns = [len(counts)]
    ns_cur = len(counts)
    min_count = -1
    for count in counts:
        if count > min_count:
            ns.append(ns_cur) # Number of elements greater or equal
            xs.append(count)
            min_count = count
        ns_cur -= 1
    xs.append(xs[len(xs)-1]+1)
    ns.append(0)
    return xs, np.array(ns)

fig, ax = plt.subplots(2, 3, figsize=(16,8))

colormap = cmx.get_cmap('viridis', 8)
colors = [colormap(x) for x in np.array([x for x in range(0,8)])/8.0]
colors = ['lightblue', 'blue', '#00f0b6', '#7bb551', '#0e750e']

modification_difference_threshold = config.modification_difference_threshold
replacement_difference_threshold = config.replacement_difference_threshold

# ax_snp.set_xscale('log')
# ax_snp.set_yscale('log')
dataset_idx = 0
```

```
for i in range(2):
    for j in range(3):
        if i == 1 and j >= 1:
            continue
        dataset = datasets[dataset_idx]
        ax[i][j].set_title(dataset.capitalize(), fontsize=16)

        ax[i][j].set_ylabel(r'Fraction host samples $\geq$ $p$', fontsize=14)
        ax[i][j].set_xlabel(r'Proportion ($p$) QP', fontsize=14)

        ax[i][j].spines['top'].set_visible(False)
        ax[i][j].spines['right'].set_visible(False)
        ax[i][j].get_xaxis().tick_bottom()
        ax[i][j].get_yaxis().tick_left()

        color_i = 0

        ymin, ymax = 0.0001, 1.1
        ax[i][j].set_ylim([ymin,ymax])
        ax[i][j].set_xlim([0, 1.1])

        for cohort in custom_cohorts_ordered:
            xs, ns =
calculate_unnormalized_survival_from_vector(props_qp_by_dataset_by_cohort[dataset][cohort])
            ax[i][j].step(xs,ns/float(ns[0]),'-',color=colors[color_i],linewidth=1.4,
label=cohort, where='pre',zorder=4)
            color_i += 1

        dataset_idx += 1

# All cohorts combined
# Include HMP too

i = 1; j = 1
ax[i][j].set_title("Combined", fontsize=16)
ax[i][j].set_ylabel(r'Fraction host samples $\geq$ $p$', fontsize=14)
ax[i][j].set_xlabel(r'Proportion ($p$) QP', fontsize=14)
ax[i][j].spines['top'].set_visible(False)
ax[i][j].spines['right'].set_visible(False)
ax[i][j].get_xaxis().tick_bottom()
ax[i][j].get_yaxis().tick_left()

color_i = 0
ymin, ymax = 0.0001, 1.1
ax[i][j].set_ylim([ymin,ymax])
ax[i][j].set_xlim([0, 1.1])

for cohort in custom_cohorts_ordered:
    combined_props = []
    for dataset in datasets:
        combined_props += props_qp_by_dataset_by_cohort[dataset][cohort]
    xs, ns = calculate_unnormalized_survival_from_vector(combined_props)
    ax[i][j].step(xs,ns/float(ns[0]),'-',color=colors[color_i],linewidth=1.4, label=cohort,
where='pre',zorder=4)
    color_i += 1

# Legend

color_i = 0
for cohort in custom_cohorts_ordered:
    ax[-1][-1].step([],[],'-',color=colors[color_i],linewidth=1.4, label=cohort)
```

```
    color_i += 1
ax[-1][-1].legend(loc='lower left', fontsize=16, numpoints=1, ncol=1, handlelength=1,
frameon=False)
ax[-1][-1].axis('off')

plt.tight_layout()
plt.show()

fig.savefig('%s/qp_proportion_survival_curves_all_cohorts.pdf' % (config.analysis_directory),
bbox_inches='tight')


# In[18]:


props_qp_by_dataset_by_cohort


# In[34]:


len(samples['hmp'])


# In[35]:


# Plot richness over time for each cohort

fig, ax = plt.subplots(2,2, figsize=(12,7), sharey=True, constrained_layout=True)

# HMP1-2
hmp_samples = su.get_sample_names('HMP')
order_richness_dict = defaultdict(list)
for sample in hmp_samples:
    subject, order = sample_order_map[sample]
    richness = richness_dict[sample]
    order_richness_dict[order].append(richness)

order_list = [1, 2, 3]
order_labels = ['1', '2', '3']
plot_data = [order_richness_dict[o] for o in order_list]

ax[0][0].boxplot(plot_data)
ax[0][0].set_xticklabels(order_labels)
ax[0][0].set_ylabel("Richness")
ax[0][0].set_title("HMP1-2")

# Backhed
backhed_samples = su.get_sample_names('Backhed')
order_richness_dict = defaultdict(list)
for sample in backhed_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples)
    richness = richness_dict[sample]
    order_richness_dict[tp].append(richness)

order_list = ['M1', 'I1', 'I2', 'I3']
order_labels = ['Mother', 'Birth (2-5 days)', '4 Months', '12 Months']
plot_data = [order_richness_dict[o] for o in order_list]

ax[0][1].boxplot(plot_data)
```

```
ax[0][1].set_xticklabels(order_labels)
ax[0][1].set_title("Backhed")

# Ferretti
ferretti_samples = su.get_sample_names('Ferretti')
order_richness_dict = defaultdict(list)
for sample in ferretti_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples)
    richness = richness_dict[sample]
    order_richness_dict[tp].append(richness)

order_list = ['M1', 'I1', 'I2', 'I3', 'I4', 'I5']
order_labels = ['Mother', '1 day', '3 days', '1 wk', '1 mon', '4 mon']
plot_data = [order_richness_dict[o] for o in order_list]

ax[1][0].boxplot(plot_data)
ax[1][0].set_xticklabels(order_labels)
ax[1][0].set_ylabel("Richness")
ax[1][0].set_title("Ferretti")

# Yassour
yassour_samples = su.get_sample_names('Yassour')
order_richness_dict = defaultdict(list)
for sample in yassour_samples:
    tp = su.sample_to_tp(sample, sample_order_map, hmp_samples, mother_samples)
    richness = richness_dict[sample]
    order_richness_dict[tp].append(richness)

order_list = ['M1', 'M2', 'M3', 'I1', 'I2', 'I3', 'I4', 'I5']
order_labels = ['Mother:\nGest', 'Mother:\nDelivery','Mother:\n3 mon', 'Birth', '1 wk', '2 wk',
'1 mon', '2 mon', '3 mon']
plot_data = [order_richness_dict[o] for o in order_list]

ax[1][1].boxplot(plot_data)
ax[1][1].set_xticklabels(order_labels)
ax[1][1].set_title("Yassour")

plt.show()


# In[36]:


fig, ax = plt.subplots(figsize=(12, 3.6), sharey=True, constrained_layout=True)

# Shao
month_bins = np.arange(4, 15) * 30.5
order_bins = [0, 1, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 21] + list(month_bins)

shao_samples = su.get_sample_names('Shao')
order_richness_dict = defaultdict(list)
for sample in shao_samples:
    subject, order = sample_order_map[sample]
    richness = richness_dict[sample]
    if subject[-1] == 'M': # Mother
        order_richness_dict[0].append(richness)
    if order > 21: # Convert to month bins
        month_bin = round_down(order, 30.5)
        order_richness_dict[month_bin].append(richness)
    else:
        order_richness_dict[order].append(richness)
```

```
plot_data = [order_richness_dict[o] for o in order_bins]
order_labels = ['Mother', '1d', '4d', '6d', '7d', '8d', '9d', '10d', '11d', '12d', '13d', '14d',
'17d', '18d', '21d', '4m', '5m', '6m', '7m', '8m', '9m', '10m', '11m', '12m', '13m', '14m']

ax.boxplot(plot_data)
ax.set_xticklabels(order_labels)
ax.set_ylabel("Richness")
ax.set_title("Shao")
```

plot_sfs.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import config, parse_midas_data, sample_utils as su, sfs_utils
import numpy as np
from matplotlib import pyplot as plt
from collections import defaultdict
import random


# In[2]:


sample_cohort_map = su.parse_sample_cohort_map()
good_species_list = parse_midas_data.load_pickled_good_species_list()
qp_color_map = {'qp': 'blue', 'non-qp': 'red', 'low-coverage': 'gray'}


# In[3]:


# Grab QP samples for all species
cohorts = ['hmp', 'backhed', 'ferretti', 'yassour', 'shao']
samples = {cohort: su.get_sample_names(cohort) for cohort in cohorts}
sample_species_qp_dict = defaultdict(dict)

for species in good_species_list:
    for cohort in cohorts:
        qp_sample_sets = su.load_qp_samples(samples[cohort], species,
prev_cohort='all')
        for qp_status in ['qp', 'non-qp', 'low-coverage']:
            for sample in qp_sample_sets[qp_status]:
                sample_species_qp_dict[sample][species] = qp_status


# In[4]:


species = good_species_list[1]
print(species)


# In[5]:
```

```
samples, sfs_maps = parse_midas_data.parse_within_sample_sfs(species,

allowed_variant_types=set(['4D']),
                                                                prev_cohort='all')


# In[6]:


infant_samples = su.get_sample_names('infant')
olm_samples = su.get_sample_names('olm')
infant_samples = [sample for sample in infant_samples if sample not in olm_samples]
desired_samples = [sample for sample in samples if sample in infant_samples]
# random.shuffle(desired_samples)


# In[7]:


cohort_count_dict = defaultdict(int)
for sample in desired_samples[:120]:
    cohort_count_dict[sample_cohort_map[sample]] += 1
print(cohort_count_dict) # Cohorts represented in 120 SFSs to be plotted


# In[20]:


nonqp_5050_sample = desired_samples[7]
qp_100_sample = desired_samples[24]
qp_90_sample = desired_samples[53]


# In[51]:


qp_color='#08519c'
nonqp_color='#77acff'
qp_color_map = {'qp': qp_color, 'non-qp': nonqp_color}

fig, ax = plt.subplots(3, 1, figsize=(3.5,7), sharex=True)

i = 0
letter_dict = {0: 'A', 1: 'B', 2: 'C'}
ymin_ymax_dict = {0: [0, 6000], 1: [0, 3000], 2: [0, 3000]}

for sample in [qp_100_sample, qp_90_sample, nonqp_5050_sample]:
    sfs_map = sfs_maps[sample]
    qp_status = sample_species_qp_dict[sample][species]
    color = qp_color_map[qp_status]
    major_freqs = []
    for key in sfs_map.keys():
        D,A = key
        n = sfs_map[key][0]

        alt_freq = A/float(D)
        major_freq = (1-alt_freq) if alt_freq < 0.5 else alt_freq
        major_freqs += ([major_freq]*n)

    ymin, ymax = ymin_ymax_dict[i]
    ax[i].hist(major_freqs, bins=50, color=color)
```

```
    ax[i].fill_between([0.8,1], [ymin,ymin],[ymax,ymax],color='0.8',zorder=-1)
    for axis in ['top','bottom','left','right']:
        ax[i].spines[axis].set_linewidth(2)

    ax[i].set_ylim((ymin, ymax))
    # ax[i].set_title("    Sample %s" % (sample), fontsize=16, loc='left')
    ax[i].text(0.09, 1.12, "Sample %s" % (sample), size=16,
transform=ax[i].transAxes)
    ax[i].text(-0.018, 1.12, letter_dict[i], size=20, transform=ax[i].transAxes,
weight='bold')
    ax[i].set_yticks([])
    ax[i].set_xticks([0.5, 0.6, 0.7, 0.8, 0.9, 1])
    ax[i].set_xlim((0.5, 1))
    ax[i].set_xticklabels(['50', '60', '70', '80', '90', '100'], fontsize=16)

    i += 1

ax[1].set_ylabel("Fraction of synonymous sites", fontsize=16, labelpad=14)
ax[2].set_xlabel("Major allele freq (%)", fontsize=16)

plt.subplots_adjust(hspace=0.4)
plt.margins(0.2)
plt.show()

fig.savefig('%s/SFS_examples.pdf' % config.analysis_directory)


# In[8]:


fig, ax = plt.subplots(6, 6, figsize=(20, 20))

x = 0; y = 0
for i in range(36):
    sample = desired_samples[i]
    sfs_map = sfs_maps[sample]
    qp_status = sample_species_qp_dict[sample][species]
    color = qp_color_map[qp_status]
    major_freqs = []
    for key in sfs_map.keys():
        D,A = key
        n = sfs_map[key][0]

        alt_freq = A/float(D)
        major_freq = (1-alt_freq) if alt_freq < 0.5 else alt_freq
        major_freqs += ([major_freq]*n)

    ax[x][y].hist(major_freqs, bins=30, color=color)
    ax[x][y].set_ylim((0, 5000))
    x += 1
    if x == 6:
        y += 1
        x = 0

plt.show()


# In[67]:


fig, ax = plt.subplots(6, 6, figsize=(20, 20))
```

```
x = 0; y = 0
for i in range(36):
    sample = desired_samples[i]
    sfs_map = sfs_maps[sample]
    qp_status = sample_species_qp_dict[sample][species]
    color = qp_color_map[qp_status]
    major_freqs = []
    for key in sfs_map.keys():
        D,A = key
        n = sfs_map[key][0]

        alt_freq = A/float(D)
        major_freq = (1-alt_freq) if alt_freq < 0.5 else alt_freq
        major_freqs += ([major_freq]*n)

    ax[x][y].hist(major_freqs, bins=30, color=color)
    ax[x][y].set_ylim((0, 5000))
    x += 1
    if x == 6:
        y += 1
        x = 0

plt.show()


# In[68]:


fig, ax = plt.subplots(6, 6, figsize=(20, 20))

x = 0; y = 0
for i in range(36, 36*2):
    sample = desired_samples[i]
    sfs_map = sfs_maps[sample]
    qp_status = sample_species_qp_dict[sample][species]
    color = qp_color_map[qp_status]
    major_freqs = []
    for key in sfs_map.keys():
        D,A = key
        n = sfs_map[key][0]

        alt_freq = A/float(D)
        major_freq = (1-alt_freq) if alt_freq < 0.5 else alt_freq
        major_freqs += ([major_freq]*n)

    ax[x][y].hist(major_freqs, bins=30, color=color)
    ax[x][y].set_ylim((0, 5000))
    x += 1
    if x == 6:
        y += 1
        x = 0

plt.show()


# In[69]:


fig, ax = plt.subplots(6, 6, figsize=(20, 20))
```

```
x = 0; y = 0
for i in range(36*2, 36*3):
    sample = desired_samples[i]
    sfs_map = sfs_maps[sample]
    qp_status = sample_species_qp_dict[sample][species]
    color = qp_color_map[qp_status]
    major_freqs = []
    for key in sfs_map.keys():
        D,A = key
        n = sfs_map[key][0]

        alt_freq = A/float(D)
        major_freq = (1-alt_freq) if alt_freq < 0.5 else alt_freq
        major_freqs += ([major_freq]*n)

    ax[x][y].hist(major_freqs, bins=30, color=color)
    ax[x][y].set_ylim((0, 5000))
    x += 1
    if x == 6:
        y += 1
        x = 0

plt.show()


# In[13]:


fig, ax = plt.subplots(6, 6, figsize=(20, 20))

x = 0; y = 0
for i in range(36*3, 36*4):
    sfs_map = sfs_maps[samples[i]]
    major_freqs = []
    for key in sfs_map.keys():
        D,A = key
        n = sfs_map[key][0]

        alt_freq = A/float(D)
        major_freq = (1-alt_freq) if alt_freq < 0.5 else alt_freq
        major_freqs += ([major_freq]*n)

    ax[x][y].hist(major_freqs, bins=30)
    ax[x][y].set_ylim((0, 5000))
    x += 1
    if x == 6:
        y += 1
        x = 0

plt.show()
```