Hardy Weinburg Rotation Project

Emma Wade

```
#install.packages("reticulate")
library(reticulate)
```

Sample parts of the genome for simulations – from Swetha

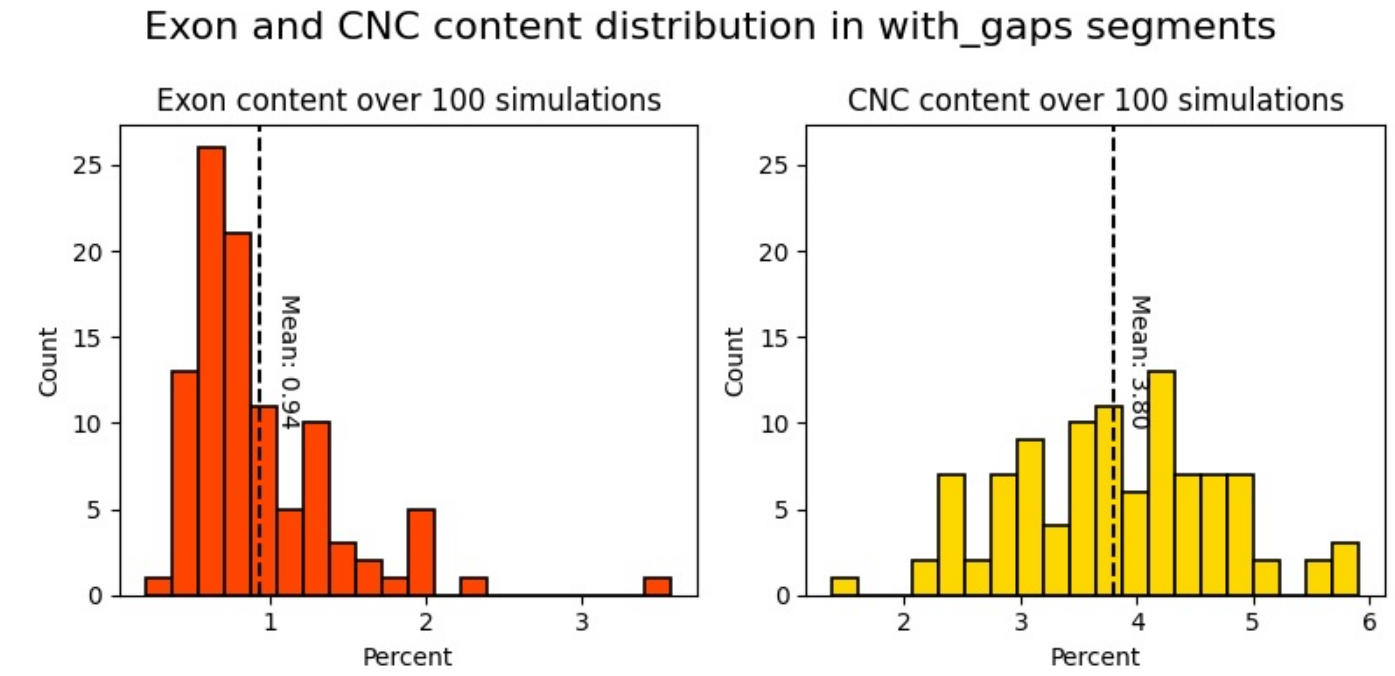
```
import os
import stdpopsim
import numpy as np
import pandas as pd
import getpass

if getpass.getuser() == "eewade":
    os.chdir("/u/home/e/eewade/project-klohmuel/bgnc")
else:
    os.chdir("/Users/swetha/work/lohmueller_lab/bgnc")


def check_intervals_overlap(starts, ends):
    # intervals should be sorted already
    for i in range(len(starts) - 1):
        if starts[i+1] < ends[i]:
            print(i)
            return True
    return False


# bedtools subtract -a $statebed -b $statequi
# trivial and obvious (not)
# this function DOES NOT allow overlapping intervals in a
def sub_intervals(xs, ys):
    if len(xs) == 0:
        return []
    i = j = 0
    rs = []
    cur_lo = xs[0][0]
    while i < len(xs) and j < len(ys):
        x_lo, x_hi = xs[i]
        cur_lo = max(cur_lo, x_lo)
        y_lo, y_hi = ys[j]
        if cur_lo < y_lo:
            rs.append((cur_lo, min(y_lo, x_hi)))
            if x_hi < y_lo:
                i += 1
            else:
                cur_lo = y_lo
        else:
            if y_hi < x_hi:
                cur_lo = y_hi
                j += 1
            else:
                i += 1
    if i < len(xs):
        rs.append((max(cur_lo, xs[i][0]), xs[i][1])) #rs.append((cur_lo, xs[i][1]))
    rs.extend(xs[i+1:])
    return rs


def merge_intervals(intervals, sort=True):
    if len(intervals) == 0:
        return []
    if not sort:
        intervals = sorted(intervals, key=lambda x: x[0])
    rs = []
    cur_lo, cur_hi = intervals[0]
    for lo, hi in intervals[1:]:
        if lo <= cur_hi:
            cur_hi = max(cur_hi, hi)
        else:
            rs.append((cur_lo, cur_hi))
            cur_lo, cur_hi = lo, hi
    rs.append((cur_lo, cur_hi))
```

```
    return rs


def fill_in_annots(annots, fill_name, fill_col):
    """
    >>> df = pd.DataFrame({'start': [1, 5, 10, 15], 'end': [3, 9, 15, 21], 'type': ['exon']*4})
    >>> fill_in_annots(df, 'fill', 'type')
       start  end   type
    0      1    3   exon
    1      3    5   fill
    2      5    9   exon
    3      9   10   fill
    4     10   15   exon
    6     15   21   exon
    """
    new_annots = []
    for i in range(len(annots) - 1):
        s, e, f = annots.iloc[i]
        new_annots.append((s, e, f))
        next_s = annots.iloc[i + 1, 0]
        if e < next_s:
            new_annots.append((e, next_s, fill_name))
    new_annots.append((annots.iloc[-1, 0], annots.iloc[-1, 1], annots.iloc[-1, 2]))
    return pd.DataFrame(new_annots, columns=['start', 'end', fill_col])


# run this once
def make_chrom_annots():
    if all([os.path.exists(f"./mysimfiles/20241113/{chrom}_annotations.csv") for chrom in chr_names]):
        print("Annotations already exist")
        return
    # centromere and gap locations from ucsc genome browser
    centromeres = pd.read_csv('./other_stuff/centromeres.txt', sep='\t', header=None,
                              names=['bin', 'chrom', 'start', 'end', 'name'])
    gaps = pd.read_csv('./other_stuff/gap.txt', sep='\t', header=None,
                       names=['bin', 'chrom', 'start', 'end', 'ix', 'n', 'size', 'type', 'bridge'])
    # top 5% CNC regions from Chenlu (this is already merged)
    cnc = pd.read_csv("./exons_and_cncs/top5_nc_470way.bed", sep='\t', header=None,
                      names=['chrom', 'start', 'end'])
    # CDS annots from stdpopsim
    cds = species.get_annotations('ensembl_havana_104_CDS')
    # for each chromosome, save all types of annotations to file
    for chrom in chr_names:
        # get annotations for each type for this chromosome
        cds_chrom = cds.get_chromosome_annotations(chrom)
        cnc_chrom = cnc.loc[cnc['chrom'] == chrom, ['start', 'end']]
        centromeres_chrom = centromeres.loc[centromeres['chrom'] == chrom, ['start', 'end']]
        gaps_chrom = gaps.loc[gaps['chrom'] == chrom, ['start', 'end']]
        # check for overlaps within each type of annotation
        check_intervals_overlap(cds_chrom[:, 0], cds_chrom[:, 1])
        check_intervals_overlap(cnc_chrom['start'].values, cnc_chrom['end'].values)
        check_intervals_overlap(centromeres_chrom['start'].values, centromeres_chrom['end'].values)
        check_intervals_overlap(gaps_chrom['start'].values, gaps_chrom['end'].values)
        # subtraction
        cds_chrom = sub_intervals(cds_chrom, centromeres_chrom.values)
        cds_chrom = sub_intervals(cds_chrom, gaps_chrom.values)
        cnc_chrom = sub_intervals(cnc_chrom.values, centromeres_chrom.values)
        cnc_chrom = sub_intervals(cnc_chrom, gaps_chrom.values)
        cnc_chrom = sub_intervals(cnc_chrom, cds_chrom)
        cds_chrom = pd.DataFrame(cds_chrom, columns=['start', 'end'])
        cnc_chrom = pd.DataFrame(cnc_chrom, columns=['start', 'end'])
        # add column to each dataframe for each type of annotation
        cds_chrom['type'] = 'exon'
        cnc_chrom['type'] = 'cnc'
        centromeres_chrom['type'] = 'exclude'
        gaps_chrom['type'] = 'exclude'
        # combine all annotations for this chromosome and sort by start and end
        chrom_annots = pd.concat([cds_chrom, cnc_chrom, centromeres_chrom, gaps_chrom])
        chrom_annots = chrom_annots.sort_values(by=['start', 'end'])
        # check for overlaps
        if check_intervals_overlap(chrom_annots['start'].values, chrom_annots['end'].values):
            print(f"Overlapping intervals in {chrom}")
            continue
        # fill in gaps between annotations
        chrom_annots = fill_in_annots(chrom_annots, 'bkgd', 'type')
        # save to file
        chrom_annots.to_csv(f"./mysimfiles/20241113/{chrom}_annotations.csv", index=False)
```

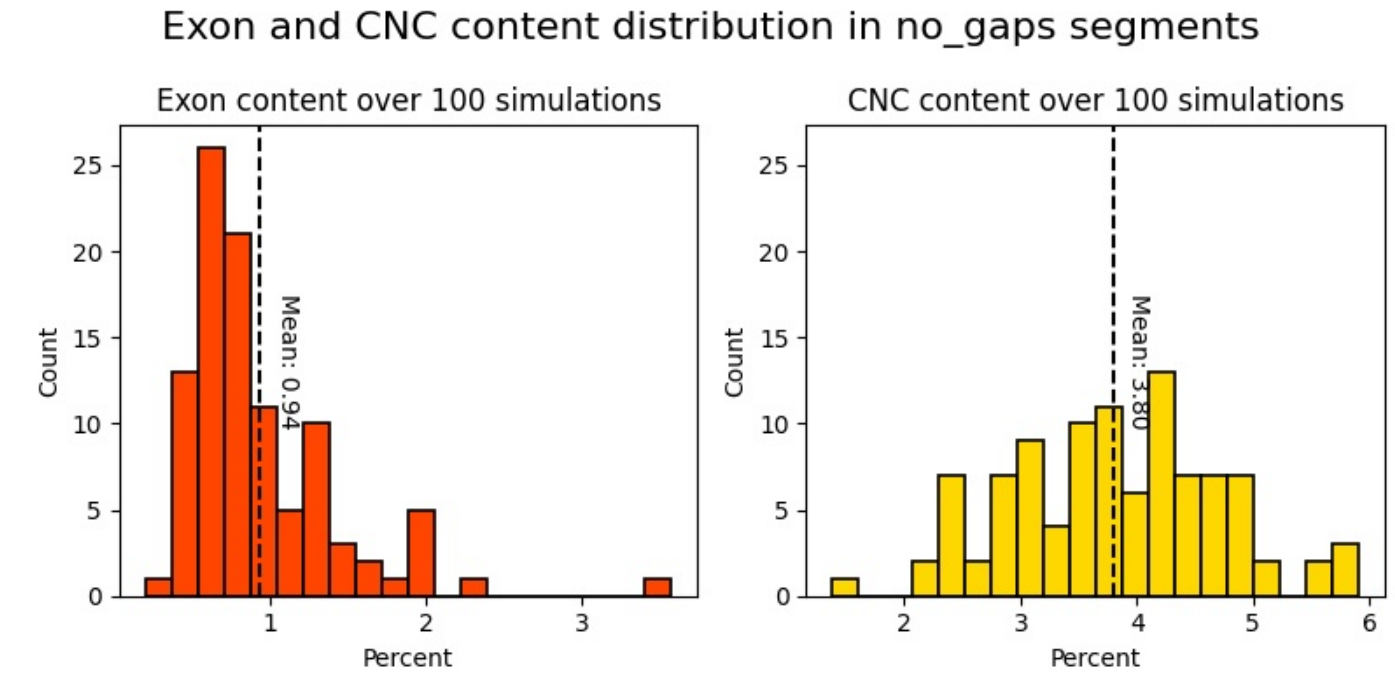
```
def sample_chrom_segment(chr_lengths, chr_names, chr_probs, length=20_000_000):
    chrom = np.random.choice(chr_names, p=chr_probs)
    chrom_length = chr_lengths[chrom]
    start = np.random.randint(1, chrom_length - length)
    end = start + length
    return chrom, start, end


def subset_annotations(chrom_annots, start, end):
    """
    >>> df = pd.DataFrame({'start': [1, 3, 5, 9, 10, 15], 'end': [3, 5, 9, 10, 15, 21]})
    >>> subset_annotations(df, 2, 9)
       start  end
    0      2    3
    1      3    5
    2      5    9
    >>> subset_annotations(df, 2, 11)
       start  end
    0      2    3
    1      3    5
    2      5    9
    3      9   10
    4     10   11
    >>> subset_annotations(df, 10, 21)
       start  end
    2     10   15
    3     15   21
    """
    # get the left bin index for start and end
    start_idx = np.digitize(start, chrom_annots['start'].values) - 1
    end_idx = np.digitize(end, chrom_annots['end'].values)
    if chrom_annots.iloc[end_idx - 1, 1] == end:
        end_idx -= 1
    subset_annots = chrom_annots.iloc[start_idx:end_idx + 1].copy()
    subset_annots.iloc[0, 0] = start
    subset_annots.iloc[-1, 1] = end
    return subset_annots


#%%
# chromosome details from stdpopsim
species = stdpopsim.get_species("HomSap")
chr_lengths = {f'chr{i.id}': i.length for i in species.genome.chromosomes if i.id not in ["X", "Y", "MT"]}
chr_probs = np.array(list(chr_lengths.values())) / sum(chr_lengths.values())
chr_names = list(chr_lengths.keys())

make_chrom_annots()

#%%

# sample segments for 100 simulations without including gaps and nan recomb rates
np.random.seed(29752806)
segment_details = []
for i in range(1, 101):
    # make sure there are no nans in recomb rates
    while True:
        # sample chromosome segment
        c, s, e = sample_chrom_segment(chr_lengths, chr_names, chr_probs)
        # read annots of segment
        chrom_annots = pd.read_csv(f"./mysimfiles/20241113/{c}_annotations.csv")
        segment_annots = subset_annotations(chrom_annots, s, e)
        # make sure there are no "exclude"s in segment annots
        if "exclude" in segment_annots['type'].values:
            continue
        # get recomb rates
        segment = species.get_contig(chromosome=c, left=s, right=e, genetic_map="HapMapII_GRCh38")
        recomb_rate = segment.recombination_map.rate
        # make sure there are no nans in recomb rates
        if np.sum(np.isnan(recomb_rate)) == 0:
            break
    # make the annots start at 0 and end at length
    segment_annots['start'] = segment_annots['start'] - s
    segment_annots['end'] = segment_annots['end'] - s - 1  # end is inclusive in slim
    segment_annots.to_csv(f'./mysimfiles/20241113/sim_{i}_annots.csv', index=False)
    # save recomb rates to file
    recomb_pos = segment.recombination_map.position.astype(int)
    # slim only uses recomb rates at the end of each interval
    recomb_ends = recomb_pos[1:]
    recomb_df = pd.DataFrame({'end': recomb_ends, 'rate': recomb_rate})
```

```
    recomb_df.to_csv(f'./mysimfiles/20241113/sim_{i}_recomb.csv', index=False)
    # save segment details
    segment_details.append((c, s, e))
    # save segment details to file
    segment_details_df = pd.DataFrame(segment_details, columns=['chrom', 'start', 'end'])
    segment_details_df.to_csv(f'./mysimfiles/20241113/sim_segment_details.csv', index=False)

# %%
# make distribution of exon and cnc percentages
import matplotlib.pyplot as plt


def plot_exon_cnc_perc_dist(sample_wo_gaps):
    exon_percs, cnc_percs = [], []
    total_len = 2e7
    for i in range(1, 101):
        annots = pd.read_csv(f'./mysimfiles/20241113/sim_{i}_annots.csv')
        exons, cnc = annots.loc[annots['type'] == 'exon'], annots.loc[annots['type'] == 'cnc']
        exon_len = sum(exons['end'] - exons['start'])
        cnc_len = sum(cnc['end'] - cnc['start'])
        exon_percs.append(exon_len / total_len * 100)
        cnc_percs.append(cnc_len / total_len * 100)
    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 4), sharey=True)
    axes = axes.ravel()
    text_y = -np.inf
    for ax, percs, color, percs_name in zip(axes, [exon_percs, cnc_percs], ['orangered', 'gold'], ['Exon', 'CNC']
):
        ax.hist(percs, bins=20, color=color, edgecolor='black', linewidth=1.2)
        ax.axvline(x=np.mean(percs), color='black', linestyle='--')
        text_y = np.max([text_y, ax.get_ylim()[1]])
        ax.text(np.mean(percs) + 0.1, text_y / 2, f"Mean: {np.mean(percs):.2f}", rotation=270, ha='left', va='cen
ter')
        ax.set_title(f"{percs_name} content over 100 simulations")
        ax.set_xlabel("Percent")
        ax.set_ylabel("Count")
    axes[1].yaxis.set_tick_params(labelleft=True)
    fig.suptitle(f"Exon and CNC content distribution in {sample_wo_gaps} segments", fontsize=16)
    fig.tight_layout()
    fig.show()
    fig.savefig(f'./mysimfiles/20241113/{sample_wo_gaps}_plot_exon.png')   # save the figure to file
    plt.close(fig)    # close the figure wind


plot_exon_cnc_perc_dist('no_gaps')
plot_exon_cnc_perc_dist('with_gaps')

###### Haven't tried these! #######
# %%
# plot segment locations
import matplotlib.pyplot as plt


chr_ends = {k: v for k, v in zip(chr_names, np.cumsum(list(chr_lengths.values())))}
chr_starts = {k: v - chr_lengths[k] for k, v in chr_ends.items()}
chr_boundaries = np.array([0] + list(chr_ends.values()))
chr_boundary_midpts = (chr_boundaries[1:] + chr_boundaries[:-1]) / 2

# first plot the segment locations
segment_details = pd.read_csv(f'./mysimfiles/20241113/sim_segment_details.csv')
segment_details['chrom'] = segment_details['chrom'].str.strip('chr').astype(int)
segment_details = segment_details.sort_values(by=['chrom', 'start', 'end'])
segment_details.reset_index(drop=True, inplace=True)
fig, ax = plt.subplots(figsize=(18, 4))
for i, row in segment_details.iterrows():
    chrom, start, end = row
    new_start, new_end = start + chr_starts[f'chr{chrom}'], end + chr_starts[f'chr{chrom}']
    ax.plot([new_start, new_end], [i, i], color='red')

# now we plot the regions where recomb rate is NA
for chrom in chr_names:
    chr_recomb = species.get_contig(chrom, genetic_map="HapMapII_GRCh38").recombination_map
    rate, pos = chr_recomb.rate, chr_recomb.position
    pos_starts = np.where(np.isnan(rate))[0]
    pos_ends = pos_starts + 1
    for s, e in zip(pos_starts, pos_ends):
        ax.axvspan(pos[s] + chr_starts[chrom], pos[e] + chr_starts[chrom], color='gray', alpha=0.5)

# adjust x-axis to show chromosome boundaries and chromosome labels
ax.set_xlim(0, chr_ends['chr22'])
```

```
ax.set_xticks([])
ax.set_xticks(chr_boundary_midpts, minor=True)
ax.set_xticklabels(map(lambda x: x.strip('chr'), chr_ends.keys()), minor=True)
for v in chr_boundaries[1:-1]:
    ax.axvline(x=v, color='black', linestyle='--')
ax.set_xlabel("Genome position")
ax.set_ylabel("Chromosome")
ax.set_title(f"Segment details")
fig.tight_layout()
fig.show()
fig.savefig(f'./mysimfiles/20241113/segment_locations.png')   # save the figure to file
plt.close(fig)
```



A caption



A caption

SLiM architecture

```
/// Run this script as follows:
// slim -d model=<model_number> -d seed=<seed> -d out_path=<out_path> -d perc_del=<perc_del> -d sim_annots=<path_
to_sim_annots> sim_recomb=<path_to_sim_recomb> bgnc_real_dem.slim
initialize() {
  initializeSLiMOptions(keepPedigrees=F);

  if (exists("slimgui")) {
    defineConstant("model", '2p2');
```

```
    defineConstant("seed", 12345);
    defineConstant("perc_del", 0);
    defineConstant("sim_annots", '/Users/swetha/work/lohmueller_lab/bgnc/other_stuff/run7_sim_files/sim_1_annots_
with_gaps.csv');
    defineConstant("sim_recomb", '/Users/swetha/work/lohmueller_lab/bgnc/other_stuff/run7_sim_files/sim_1_recomb_
with_gaps.csv');
    defineConstant("out_path", "/Users/swetha/work/lohmueller_lab/bgnc/output/test_script");
  }

  print(sim_annots);
  print(sim_recomb);
  print(out_path);

  // command line constants
  setSeed(seed);

  // print model and seed and perc_del
  catn("Model: " + model);
  catn("Seed: " + seed);
  catn("perc_del: " + perc_del);

  // from Rodrigues 2023
  initializeMutationRate(2e-8);

  //draw deleterious mutations from Kim 2017 human DFE
  // this will have to change!!!!
  // initializeMutationType("m1", 0.5, "g", -0.01314833, 0.186);

  // set dominance coefficients for different deleterious mutation types
  defineConstant("h_neut", 0.5);
  defineConstant("h_nearNeut", 0.45);
  defineConstant("h_wkDel", 0.25);
  defineConstant("h_modDel", 0.15);
  defineConstant("h_strDel", 0.05);

  initializeMutationType("m1", h_neut, "s", "return runif(1, -2e-5, 0.0);");
  initializeMutationType("m2", h_nearNeut, "s", "return runif(1, -2e-4, -2e-5);");
  initializeMutationType("m3", h_wkDel, "s", "return runif(1, -2e-3, -2e-4);");
  initializeMutationType("m4", h_modDel, "s", "return runif(1, -2e-2, -2e-3);");
  initializeMutationType("m5", h_strDel, "s", "return runif(1, -1.0, -2e-2);");
  initializeMutationType("m6", h_neut, "f", 0.0); // add sixth mutation type representing synonymous mutations

  // convert fixed mutation to substitution
  m1.convertToSubstitution = T;
  m2.convertToSubstitution = T;
  m3.convertToSubstitution = T;
  m4.convertToSubstitution = T;
  m5.convertToSubstitution = T;
  m6.convertToSubstitution = T;

  // neutral mutations - a separate type for each genomic element noncoding type to calculate sfs
  initializeMutationType("m7", h_neut, "f", 0.0);
  initializeMutationType("m8", h_neut, "f", 0.0);

  //non-coding region mutation types
  //deleterious mutations in noncoding regions - parameters from Torgerson et al 2009
  initializeMutationType("m9", 0.4, "g", -0.001036043, 0.0415);
  initializeMutationType("m10", 0.4, "g", -0.001036043, 0.0415);

  // coding regions
  // ratio of different deleterious mutation types taken from Kim 2017 DFE (sum to 100 below)
  // assume ratio of deleterious to neutral muts of 2.31:1
  // giving 100/2.31=43.3 for neutral mutations below
  // initializeGenomicElementType("g1", c(m1, m2), c(100, 43.3));
  initializeGenomicElementType("g1", c(m1,m2,m3,m4,m5,m6), c(0.2009326, 0.2221936, 0.01764828, 0.285706, 0.273519
5, 0.432));

  // non-coding regions
  if (model == '1') {
    assert(perc_del == 0, "perc_del is not 0");
    initializeGenomicElementType("g2", c(m7), 1); // background non-coding with neutral only
    initializeGenomicElementType("g3", c(m8), 1); // conserved non-coding with neutral only
  } else if ((model == '2') | (model == '2p2')) {
    assert(perc_del == 0, "perc_del is not 0");
    initializeGenomicElementType("g2", c(m7), 1); // background non-coding with neutral only
    initializeGenomicElementType("g3", c(m10), 1); // conserved non-coding with deleterious only
  } else if (model == '3') {
    assert(perc_del != 0, "perc_del shouldn't be 0");
    initializeGenomicElementType("g2", c(m9, m7), c(perc_del, 1 - perc_del)); // background non-coding with del_n
```

```
c and neutral
    initializeGenomicElementType("g3", c(m10), 1); // conserved non-coding with deleterious only
  } else {
    stop(paste("Invalid model:", model));
  }

  // create chromosome
  annots = readCSV(sim_annots);
  for (row in 0:(annots.nrow - 1)) {
    start = annots.subset(row, 'start');
    end = annots.subset(row, 'end');
    elem_type = annots.subset(row, 'type');
    if (elem_type == 'exon') {
      initializeGenomicElement(g1, start, end);
    } else if (elem_type == 'bkgd') {
      initializeGenomicElement(g2, start, end);
    } else if (elem_type == 'cnc') {
      initializeGenomicElement(g3, start, end);
    } else {
      next;  // these are 'exclude' regions
    }
  }

  // set recombination rate
  recomb = readCSV(sim_recomb);
  rates = recomb.getValue('rate');
  ends = recomb.getValue('end');
  initializeRecombinationRate(rates, ends);

}

/// Demography:
  /// parameters here taken Kim 2017 1kG model
/// in tables S1 and S2
/// parameters scaled in terms of diploids and generations
1 early() {
  sim.addSubpop("p1", 12378);
  cat("gen,popSize" + "\n");
}

//30 late() {

//  samples = c(10, 50, 100, 500, 1000, 5000, 10000);
  //for (ss in samples)
  //{
    //p1.outputVCFSample(sampleSize=ss, outputMultiallelics = F , filePath=out_path + "test_" + ss + ".vcf");
  //}

//}

1:126269 late() {
    if (sim.cycle % 1000 == 0) {
      print(sim.cycle);
        //cat(sim.cycle + "," + p1.individuals.size() + "\n");
    }
}

// bottleneck to 1048 for 248 generations
123780 early() {
  p1.setSubpopulationSize(1048);
}

// population growth to 13625 for 1744 generations
124028 early() {
  p1.setSubpopulationSize(13625);
}

// exponential growth for final 497 generations
// to current population size of 659551
125772:126269 early() {
  t = sim.cycle - 125772;
  p1_size = round(13625 * (1 + 0.007831051)^t);
  p1.setSubpopulationSize(asInteger(p1_size));
}

// end simulation
126269 early() {
  sim.simulationFinished();
}
```

```
// output sfs
126269 late() {

  samples = c(10, 50, 100, 500, 1000, 5000, 10000, 20000, 50000);
  for (ss in samples)
  {
    p1.outputVCFSample(sampleSize=ss, outputMultiallelics = F , filePath=out_path + "complete_" + ss + ".vcf");
  }
  sim.simulationFinished();

}
```

Run simulation

```
#!/bin/bash
#$ -t 1-100:1 # ended up changing to 10
# #$ -wd ~/u/home/e/eewade/project-klohmuel/rotation/2024/10/adapted-swetha-sim/all/
#$ -wd /u/scratch/e/eewade/swetha-sims/
#$ -l h_data=52G,h_rt=23:59:00
#$ -o /u/scratch/e/eewade/logs/ancestral_again_$TASK_ID.eo
#$ -e /u/scratch/e/eewade/logs/ancestral_again_$TASK_ID.eo
#$ -M eew226@g.ucla.edu
# Notify when
#$ -m e

start_time=$(date +%s)

# INPUT ARGUMENTS
seed=1

# load the job environment:
. /u/local/Modules/default/init/modules.sh
## Edit the line below as needed:
module load mamba

mamba activate slim

slim -d "model='2p2'" \
  -d seed=12345 \
  -d "out_path='/u/scratch/e/eewade/swetha-sims/sim_output_${SGE_TASK_ID}'" \
  -d perc_del=0 \
  -d "sim_annots='/u/home/e/eewade/project-klohmuel/bgnc/mysimfiles/20241113/sim_${SGE_TASK_ID}_annots.csv'" \
  -d "sim_recomb='/u/home/e/eewade/project-klohmuel/bgnc/mysimfiles/20241113/sim_${SGE_TASK_ID}_recomb.csv'" \
  ~/rotations/kirk/112024/swetha_dominance.slim

# Calculate runtime
end_time=$(date +%s)
runtime=$((end_time - start_time))
```

Jobs that ran

```
# deleted the runs that didn't have all 9 samples sizes
ls /u/scratch/e/eewade/swetha-sims
```

Combine outputs, fix simulation output VCF to get unique names

```
#!/bin/bash
#$ -t 1-9:1
#$ -l h_data=10G,h_rt=10:59:00
#$ -o /u/scratch/e/eewade/logs/fixoutput_$TASK_ID.eo
#$ -e /u/scratch/e/eewade/logs/fixoutput_$TASK_ID.eo
# Email address to notify
#$ -M eew226@g.ucla.edu
# Notify when
#$ -m bea

# get bcftools
. /u/local/Modules/default/init/modules.sh
module load bcftools
# Set the base directory
BASE_DIR="/u/home/e/eewade/project-klohmuel/rotation/2024/10/adapted-swetha-sim/all"

# Array of sample sizes
SAMPLE_SIZES=(10 50 100 500 1000 5000 10000 20000 50000)

# Function to process files for a given sample size
process_sample_size() {
    local index=$1
    local size=${SAMPLE_SIZES[$index-1]}
    echo "Processing sample size: $size"

    # Combine VCF files for this sample size
    bcftools concat ${BASE_DIR}/sim_output_*complete_${size}.vcf -Ou | \
    bcftools annotate --set-id +'%INFO/MID' - | \
    bcftools annotate -x INFO/MID -Oz -o ${BASE_DIR}/combined_${size}.vcf.gz

    # Index the output file
    bcftools index ${BASE_DIR}/combined_${size}.vcf.gz
}

# Use SGE_TASK_ID as the index
if [ -n "$SGE_TASK_ID" ]; then
    process_sample_size $SGE_TASK_ID
else
    echo "Error: SGE_TASK_ID is not set. This script should be run as part of a Sun Grid Engine array job."
    exit 1
fi

echo "Processing complete for task ID $SGE_TASK_ID!"
```

Run Plink HWE and combine with information

```
#!/bin/bash
#$ -t 1-9:1
#$ -l h_data=10G,h_rt=10:59:00
#$ -o /u/scratch/e/eewade/logs/plink_$TASK_ID.eo
#$ -e /u/scratch/e/eewade/logs/plink_$TASK_ID.eo
#$ -M eew226@g.ucla.edu
#$ -m bea

. /u/local/Modules/default/init/modules.sh
module load bcftools
module load plink

# Set the base directory
BASE_DIR="/u/scratch/e/eewade/swetha-sims/"

# Array of sample sizes
SAMPLE_SIZES=(10 50 100 500 1000 5000 10000 20000 50000)

# Function to process a given sample size
process_sample_size() {
    local size=$1
    echo "Processing sample size: $size"

    # Extract all INFO fields from VCF
    bcftools query -f '%CHROM\t%POS\t%ID\t%REF\t%ALT\t%INFO/S\t%INFO/DOM\t%INFO/PO\t%INFO/TO\t%INFO/MT\t%INFO/AC\
t%INFO/DP\n' \
        "${BASE_DIR}/combined_${size}.vcf.gz" > "${BASE_DIR}/vcf_info_${size}.txt"

    # Run PLINK's hardy function
    plink --vcf "${BASE_DIR}/combined_${size}.vcf.gz" \
          --hardy \
          --out "${BASE_DIR}/dominance_hwe_testing_${size}"
}

# Use SGE_TASK_ID as the index
if [ -n "$SGE_TASK_ID" ]; then
    # SGE_TASK_ID is 1-based, but array indices are 0-based
    index=$((SGE_TASK_ID - 1))
    if [ $index -lt ${#SAMPLE_SIZES[@]} ]; then
        size=${SAMPLE_SIZES[$index]}
        process_sample_size $size
    else
        echo "Error: SGE_TASK_ID ($SGE_TASK_ID) is out of range. Max is ${#SAMPLE_SIZES[@]}."
        exit 1
    fi
else
    echo "Error: SGE_TASK_ID is not set. This script should be run as part of a Sun Grid Engine array job."
    exit 1
fi

echo "Processing complete for sample size $size!"
```

Get Fsel

```
def calculate_fsel(k,p):
  return 1 + (k - 4p(p-1)k(k-1)+k^2))/(2p(1-p)(1-k))

def calculate_k(s, h):
  ## k = (fitness heterozygote ab^2) / (fitnessa x fitnessb)
  #deleterious dominant k = 1 - s, deleterious recessive allele k = 1/(1 - s), overdominance k = (1 + s)^2, under
dominance k = (1 - s)
  if h == 0:
    return 1/(1-s)
  elif h == 1:
    return 1 - s
  else 0 < h < 1:
    return (1 + s)^2

vcf =
```

Prep data for plotting

```
library(ggplot2)
library(dplyr)
library(readr)
library(gsubfn)
library(data.table)
library(stats)

samples = c("10", "50", "100", "500", "1000", "5000", "10000", "20000", "50000")

df = data.frame()
for (s in samples) {
  ### VCF Info for each SNP
  infoname = paste("/u/scratch/e/eewade/swetha-sims/vcf_info_", s, ".txt", sep="")
  info = fread(infoname,h=F,stringsAsFactors = F,col.names=c("CHROM", "POS","ID", "REF", "ALT","S", "DOM", "PO","
TO","MT","AC","DP"))

  ### HWE Results
  hweresults = paste("/u/scratch/e/eewade/swetha-sims/dominance_hwe_testing_", s, ".hwe", sep="")
# Read the file, skipping the header
  hwe <- fread(hweresults, skip = 1, header = FALSE,col.names = c("CHR", "SNP", "TEST", "A1", "A2", "GENO", "O_HE
T", "E_HET", "P"))

  table = merge(info, hwe, by.x="ID", by.y="SNP")
  table = table[table$AC > 1,] # removing singletons! will always be p > 0.05
  table$ss = s
  table$P_ADJ_BF <- p.adjust(table$P, method = "bonferroni") ## bonferroni correcting p-values within each sample
size
  table$P_ADJ_FDR <- p.adjust(table$P, method = "fdr") ## bonferroni correcting p-values within each sample size
  df = rbind(df, table)
  rm(table)
}

df$inarea = "Coding"
df$inarea[df$MT %in% c("7", "10")] = "Noncoding"

#defineConstant("h_neut", 0.5);
#defineConstant("h_nearNeut", 0.45);
#defineConstant("h_wkDel", 0.25);
#defineConstant("h_modDel", 0.15);
#defineConstant("h_strDel", 0.05);

#initializeMutationType("m1", h_neut, "s", "return runif(1, -2e-5, 0.0);");
#initializeMutationType("m2", h_nearNeut, "s", "return runif(1, -2e-4, -2e-5);");
#initializeMutationType("m3", h_wkDel, "s", "return runif(1, -2e-3, -2e-4);");
#initializeMutationType("m4", h_modDel, "s", "return runif(1, -2e-2, -2e-3);");
#initializeMutationType("m5", h_strDel, "s", "return runif(1, -1.0, -2e-2);");
#initializeMutationType("m6", h_neut, "f", 0.0); // add sixth mutation type representing synonymous mutations

long_MT = c(
  "1" = "Neutral in coding h = 0.5",
  "2" = "Nearly neutral in coding h = 0.45",
  "3" = "Weakly deleterious in coding h = 0.25",
  "4" = "Moderately deleterious in coding h = 0.15",
  "5" = "Strongly deleterious in coding 0.05",
  "6" = "Syn in coding h = 0",
  "7" = "Neutral in noncoding h = 0.0",
  "10" = "Deleterious in noncoding h = 0.4"
)

df$long_MT = long_MT[df$MT]

head(df)
```

QQ plot

```
library(MetBrewer)

ggplot(df[df$ss %in% c("10", "100", "500", "1000", "5000", "10000"),], aes(sample = O_HET, color=factor(ss, level
s=c("10", "100", "500", "1000", "5000", "10000")))) +
  stat_qq() +
  stat_qq_line()+
  theme_minimal()+
  labs(color="Sample Size", title="Observed Heterozygotes")+
  scale_color_manual(values=met.brewer("Isfahan1"))+
  geom_abline()

ggplot(df[df$ss %in% c("10", "100", "500", "1000", "5000", "10000"),], aes(sample = P, color=factor(ss, levels=c(
"10", "100", "500", "1000", "5000", "10000")))) +
  stat_qq() +
  stat_qq_line()+
  theme_minimal()+
  labs(color="Sample Size", title="P-Values")+
  scale_color_manual(values=met.brewer("Isfahan1"))+
  geom_abline()

ggplot(df[df$ss %in% c("10", "100", "500", "1000", "5000", "10000"),], aes(sample = P, color=factor(ss, levels=c(
"10", "100", "500", "1000", "5000", "10000")))) +
  stat_qq() +
  stat_qq_line()+
  facet_grid(~factor(MT), scales = "free")+
  theme_minimal()+
  labs(color="Sample Size", title="P-Values")+
  scale_color_manual(values=met.brewer("Isfahan1"))+
  geom_abline()
```

Proportions

Play with colors for p-values

continue palette met.brewer(“Troy”, n=15, type=“continuous”)

```
library(dplyr)
library(tidyr)
library(MetBrewer)

result_table <- df %>%
  group_by(MT, ss) %>%
  summarise(
    Total = n(),
    Prop_P_less_0.05 = sum(P < 0.05, na.rm = TRUE)/n(),
    Prop_P_bf_less_0.05 = sum(P_ADJ_BF < 0.05, na.rm=T) / n(),
    Prop_P_fdr_less_0.05 = sum(P_ADJ_FDR < 0.05, na.rm=T) / n(),

    #Percentage_P_less_0.05 = Count_P_less_0.05 / Total * 100,
    #Percentage_P_greater_0.05 = Count_P_greater_0.05 / Total * 100
  ) %>%
  arrange(MT, ss)

# Print the result
print(result_table)

# Reshape the data for plotting
plot_data <- result_table %>%
  pivot_longer(cols = c(Prop_P_less_0.05, Prop_P_bf_less_0.05, Prop_P_fdr_less_0.05),
               names_to = "P_category",
               values_to = "Count")

plot_data$long_MT = long_MT[plot_data$MT]
plot_data$long_MT[plot_data$MT == 10] = "Deleterious in noncoding h = 0.4"

# Create the plot
ggplot(plot_data, aes(x = factor(ss, levels=c("10", "50", "100", "500", "1000", "5000", "10000", "20000", "50000"
)), y = Count, fill = factor(long_MT, levels = c("Neutral in coding h = 0.5","Nearly neutral in coding h = 0.45",
"Weakly deleterious in coding h = 0.25","Moderately deleterious in coding h = 0.15","Strongly deleterious in codi
ng 0.05","Syn in coding h = 0","Neutral in noncoding h = 0.0","Deleterious in noncoding h = 0.4")))) +
  geom_bar(stat = "identity", position = "stack") +
  facet_wrap(~factor(P_category), scales = "free") +
  scale_fill_manual(values = met.brewer("Archambault", n=8))+
  labs(x = "N",
       y = "Proportion HWE p < 0.05 ",
       fill = "Mutation Type") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        strip.text = element_text(face = "bold"))
```

```
#library(ggplot2)
# Create the plot
#ggplot(df[ss %in% c("10", "1000", "5000"),], aes(x = S, y = DOM)) +
#  geom_point(aes(color = P < 0.05)) +
#  scale_color_manual(values = c("TRUE" = "red", "FALSE" = "blue")) +
#  labs(x = "Selection Coefficient (S)",
#       y = "Dominance (h)",
#       color = "P < 0.05") +
#  facet_wrap(~ss)+
#  theme_minimal() +
#  theme(legend.position = "bottom")

df$P_quant = " > 0.05"
df$P_quant[df$P < 1.0490e-02] = "< 1.0490e-02 (0% quantile)"
df$P_quant[df$P < 2.3770e-02 & df$P > 1.0490e-02] = "< 2.3770e-02 (25% quantile)"
df$P_quant[df$P > 2.3770e-02 & df$P < 3.7185e-02] = "< 3.7185e-02 (50% quantile)"
df$P_quant[df$P > 3.7185e-02 & df$P < 0.05] = "< 0.05 (75% quantile)"


ggplot(df[df$P_quant != " > 0.05" & df$inarea == "Coding",], aes(x = S, y = DOM, color=factor(P_quant, levels = c
("< 1.0490e-02 (0% quantile)","< 2.3770e-02 (25% quantile)", "< 3.7185e-02 (50% quantile)", "< 0.05 (75% quantile
)")))) +
  geom_point(size=3) +
  labs(x = "Selection Coefficient (S)",
       y = "Dominance (h)") +
  facet_wrap(~factor(ss, levels=c("10","50","100","500","1000","5000","10000","20000","50000")))+
  theme_minimal() +
  labs(color="P")+
  scale_color_manual(values=met.brewer("Hokusai1"))+
  theme(legend.position = "bottom")
```

Rethinking pipeline

Run simulation

```
#!/bin/bash
#$ -t 1-10 # Adjusted task range
#$ -wd /u/scratch/e/eewade/swetha-sims123/
#$ -l h_data=100G,h_rt=48:00:00,highp
#$ -o /u/scratch/e/eewade/logs/ancestral_again2_$TASK_ID.eo
#$ -e /u/scratch/e/eewade/logs/ancestral_again2_$TASK_ID.eo
#$ -M eew226@g.ucla.edu
#$ -m e

start_time=$(date +%s)

# INPUT ARGUMENTS
seed=1

# Load the job environment:
. /u/local/Modules/default/init/modules.sh

# Load necessary modules
module load mamba
mamba activate slim

# Define base directory for output files
BASE_DIR="/u/scratch/e/eewade/swetha-sims123"

# Run SLiM simulation to generate a VCF file with 50,000 samples
slim -d "model='2p2'" \
  -d seed=12345 \
  -d "out_path='${BASE_DIR}/sim_output_${SGE_TASK_ID}'" \
  -d perc_del=0 \
  -d "sim_annots='/u/home/e/eewade/project-klohmuel/bgnc/mysimfiles/20241113/sim_${SGE_TASK_ID}_annots.csv'" \
  -d "sim_recomb='/u/home/e/eewade/project-klohmuel/bgnc/mysimfiles/20241113/sim_${SGE_TASK_ID}_recomb.csv'" \
  ~/rotations/kirk/112024/swetha_dominance.slim

module load bcftools
module load plink

# Define input parameters
input_vcf="${BASE_DIR}/sim_output_${SGE_TASK_ID}complete_50000.vcf"
output_prefix="subset"
sizes=(10 50 100 500 1000 50000 10000 20000 50000)

# Extract all sample IDs
all_samples="${BASE_DIR}/all_samples_${SGE_TASK_ID}.txt"
if ! bcftools query -l "$input_vcf" > "$all_samples"; then
    echo "Error: Failed to extract sample IDs from $input_vcf"
    exit 1
fi

prev_samples=""
for size in "${sizes[@]}"; do
    ###### Downsample and subset #######
    if [ "$size" -eq 50000 ]; then
        # Use the original VCF for the full sample size of 50,000
        this_vcf="$input_vcf"
    else
        # Randomly sample from the original VCF file for smaller sizes
        this_samples="${BASE_DIR}/samples_${size}_${SGE_TASK_ID}.txt"
        if [[ -n "$prev_samples" ]]; then
            # Find the remaining unique samples not in prev_samples
            remaining_samples="${BASE_DIR}/remaining_samples_${SGE_TASK_ID}.txt"
            comm -23 <(sort "$all_samples") <(sort "$prev_samples") > "$remaining_samples"

            # Calculate the number of new samples to add
            new_samples_count=$((size - $(wc -l < "$prev_samples" | xargs)))
            if [[ $new_samples_count -le 0 ]]; then
                cp "$prev_samples" "$this_samples"
            else
                # Add new unique samples and combine with prev_samples
                shuf -n "$new_samples_count" "$remaining_samples" | cat "$prev_samples" - > "$this_samples"
            fi
            rm "$remaining_samples"  # Clean up
        else
            # First iteration: randomly sample without previous samples
            shuf -n "$size" "$all_samples" > "$this_samples"
        fi
```

```
        # Subset the VCF
        this_vcf="${BASE_DIR}/sim_output_${SGE_TASK_ID}complete_${size}.vcf.gz"
        if ! bcftools view -S "$this_samples" -o "$this_vcf" -O z "$input_vcf"; then
            echo "Error: Failed to create subset VCF for size $size"
            exit 1
        fi
    fi

    ###### Run things I want #######
    # Extract info fields from the sampled VCF using bcftools query
    bcftools query \
        -f '%CHROM\t%POS\t%ID\t%REF\t%ALT\t%INFO/S\t%INFO/DOM\t%INFO/PO\t%INFO/TO\t%INFO/MT\t%INFO/AC\t%INFO/DP\n
' \
        "$this_vcf" > "${BASE_DIR}/vcf_info_${size}_${SGE_TASK_ID}.txt"

    # Run PLINK's Hardy-Weinberg equilibrium test on the sampled VCF
    plink --vcf "$this_vcf" --hardy --out "${BASE_DIR}/dominance_hwe_testing_${size}_${SGE_TASK_ID}"

    # Delete the sampled VCF file after processing, except for the original large VCF file
    if [ "$size" -ne 50000 ]; then
        rm "$this_vcf"
    fi

    # Update previous sample list
    prev_samples="$this_samples"
done

echo "Downsampling completed successfully!"

# Delete no longer needed files
rm $input_vcf
rm ${BASE_DIR}/all_samples_${SGE_TASK_ID}.txt
rm ${BASE_DIR}/samples_*_${SGE_TASK_ID}.txt
rm ${BASE_DIR}/dominance_hwe_testing_*_${SGE_TASK_ID}.log
rm ${BASE_DIR}/dominance_hwe_testing_*_${SGE_TASK_ID}.nosex

end_time=$(date +%s)
echo "Execution time: $(($end_time - $start_time)) seconds"
```

Also want to try adding constant demography, any different?

```
/// Run this script as follows:
  // slim -d model=<model_number> -d seed=<seed> -d out_path=<out_path> -d perc_del=<perc_del> -d sim_annots=<pat
h_to_sim_annots> sim_recomb=<path_to_sim_recomb> bgnc_real_dem.slim
  initialize() {
    initializeSLiMOptions(keepPedigrees=F);
    initializeTreeSeq();

    if (exists("slimgui")) {
      defineConstant("model", '2p2');
      defineConstant("seed", 12345);
      defineConstant("perc_del", 0);
      defineConstant("sim_annots", '/Users/swetha/work/lohmueller_lab/bgnc/other_stuff/run7_sim_files/sim_1_annot
s_with_gaps.csv');
      defineConstant("sim_recomb", '/Users/swetha/work/lohmueller_lab/bgnc/other_stuff/run7_sim_files/sim_1_recom
b_with_gaps.csv');
      defineConstant("out_path", "/Users/swetha/work/lohmueller_lab/bgnc/output/test_script");
    }

    print(sim_annots);
    print(sim_recomb);
    print(out_path);

    // command line constants
    setSeed(seed);

    // print model and seed and perc_del
    catn("Model: " + model);
    catn("Seed: " + seed);
    catn("perc_del: " + perc_del);

    // from Rodrigues 2023
    initializeMutationRate(2e-8);

    //draw deleterious mutations from Kim 2017 human DFE
    // this will have to change!!!!
      // initializeMutationType("m1", 0.5, "g", -0.01314833, 0.186);
```

```
    // set dominance coefficients for different deleterious mutation types
    defineConstant("h_neut", 0.5);
    defineConstant("h_nearNeut", 0.45);
    defineConstant("h_wkDel", 0.25);
    defineConstant("h_modDel", 0.15);
    defineConstant("h_strDel", 0.05);

    initializeMutationType("m1", h_neut, "s", "return runif(1, -2e-5, 0.0);");
    initializeMutationType("m2", h_nearNeut, "s", "return runif(1, -2e-4, -2e-5);");
    initializeMutationType("m3", h_wkDel, "s", "return runif(1, -2e-3, -2e-4);");
    initializeMutationType("m4", h_modDel, "s", "return runif(1, -2e-2, -2e-3);");
    initializeMutationType("m5", h_strDel, "s", "return runif(1, -1.0, -2e-2);");
    initializeMutationType("m6", h_neut, "f", 0.0); // add sixth mutation type representing synonymous mutations

    // convert fixed mutation to substitution
    m1.convertToSubstitution = T;
    m2.convertToSubstitution = T;
    m3.convertToSubstitution = T;
    m4.convertToSubstitution = T;
    m5.convertToSubstitution = T;
    m6.convertToSubstitution = T;

    // neutral mutations - a separate type for each genomic element noncoding type to calculate sfs
    initializeMutationType("m7", h_neut, "f", 0.0);
    initializeMutationType("m8", h_neut, "f", 0.0);

    //non-coding region mutation types
    //deleterious mutations in noncoding regions - parameters from Torgerson et al 2009
    initializeMutationType("m9", 0.4, "g", -0.001036043, 0.0415);
    initializeMutationType("m10", 0.4, "g", -0.001036043, 0.0415);

    // coding regions
    // ratio of different deleterious mutation types taken from Kim 2017 DFE (sum to 100 below)
    // assume ratio of deleterious to neutral muts of 2.31:1
    // giving 100/2.31=43.3 for neutral mutations below
    // initializeGenomicElementType("g1", c(m1, m2), c(100, 43.3));
    initializeGenomicElementType("g1", c(m1,m2,m3,m4,m5,m6), c(0.2009326, 0.2221936, 0.01764828, 0.285706, 0.2735
195, 0.432));

    // non-coding regions
    if (model == '1') {
      assert(perc_del == 0, "perc_del is not 0");
      initializeGenomicElementType("g2", c(m7), 1); // background non-coding with neutral only
      initializeGenomicElementType("g3", c(m8), 1); // conserved non-coding with neutral only
    } else if ((model == '2') | (model == '2p2')) {
      assert(perc_del == 0, "perc_del is not 0");
      initializeGenomicElementType("g2", c(m7), 1); // background non-coding with neutral only
      initializeGenomicElementType("g3", c(m10), 1); // conserved non-coding with deleterious only
    } else if (model == '3') {
      assert(perc_del != 0, "perc_del shouldn't be 0");
      initializeGenomicElementType("g2", c(m9, m7), c(perc_del, 1 - perc_del)); // background non-coding with del
_nc and neutral
      initializeGenomicElementType("g3", c(m10), 1); // conserved non-coding with deleterious only
    } else {
      stop(paste("Invalid model:", model));
    }

    // create chromosome
    annots = readCSV(sim_annots);
    for (row in 0:(annots.nrow - 1)) {
      start = annots.subset(row, 'start');
      end = annots.subset(row, 'end');
      elem_type = annots.subset(row, 'type');
      if (elem_type == 'exon') {
        initializeGenomicElement(g1, start, end);
      } else if (elem_type == 'bkgd') {
        initializeGenomicElement(g2, start, end);
      } else if (elem_type == 'cnc') {
        initializeGenomicElement(g3, start, end);
      } else {
        next;  // these are 'exclude' regions
      }
    }

    // set recombination rate
    recomb = readCSV(sim_recomb);
    rates = recomb.getValue('rate');
    ends = recomb.getValue('end');
```

```
    initializeRecombinationRate(rates, ends);

  }

  /// Demography:
    /// parameters here taken Kim 2017 1kG model
  /// in tables S1 and S2
  /// parameters scaled in terms of diploids and generations
  1 early() {
    sim.addSubpop("p1", 12378);
    cat("gen,popSize" + "\n");
  }

  // end simulation
  //30 early() {
    //  sim.simulationFinished();
    //}

  //30 late() {

    //  samples = c(1000);
    //  for (ss in samples)
      //  {
        //    p1.outputVCFSample(sampleSize=ss, outputMultiallelics = F , filePath=out_path + "_constant_demo_com
plete_" + ss + ".vcf");
        //      sim.treeSeqOutput(out_path +"_overlay.trees");
        //  }
    //}



  1:123780 late() {
    if (sim.cycle % 10000 == 0) {
      print(sim.cycle);
    }
  }

  // output sfs
  123780 late() {
    sim.treeSeqOutput(out_path +"_constant_demo_overlay.trees");
    samples = c(10000);
    for (ss in samples)
    {
      p1.outputVCFSample(sampleSize=ss, outputMultiallelics = F , filePath=out_path + "constant_demo_" + ss + ".v
cf");
    }

    sim.simulationFinished();

  }
```

And submit

```
#!/bin/bash
#$ -t 1-20 # Adjusted task range
#$ -wd /u/scratch/e/eewade/swetha-sims123/
#$ -l h_data=50G,h_rt=24:00:00
#$ -o /u/scratch/e/eewade/logs/ancestral_constant_$TASK_ID.eo
#$ -e /u/scratch/e/eewade/logs/ancestral_constant_$TASK_ID.eo
#$ -M eew226@g.ucla.edu
#$ -m e

start_time=$(date +%s)

# INPUT ARGUMENTS
seed=1

# Load the job environment:
. /u/local/Modules/default/init/modules.sh

# Load necessary modules
module load mamba
mamba activate slim

# Define base directory for output files
BASE_DIR="/u/scratch/e/eewade/swetha-sims123"
```

```
# Run SLiM simulation to generate a VCF file with 50,000 samples
slim -d "model='2p2'" \
  -d seed=12345 \
  -d "out_path='${BASE_DIR}/sim_output_${SGE_TASK_ID}'" \
  -d perc_del=0 \
  -d "sim_annots='/u/home/e/eewade/project-klohmuel/bgnc/mysimfiles/20241113/sim_${SGE_TASK_ID}_annots.csv'" \
  -d "sim_recomb='/u/home/e/eewade/project-klohmuel/bgnc/mysimfiles/20241113/sim_${SGE_TASK_ID}_recomb.csv'" \
  /u/project/klohmuel/eewade/rotation/2024/11/swetha_dominance_constant_demo.slim

module load bcftools
module load plink

# Define input parameters
input_vcf="${BASE_DIR}/sim_output_${SGE_TASK_ID}constant_demo_10000.vcf"
output_prefix="subset"
sizes=(10 50 100 500 1000 5000 10000)

# Extract all sample IDs
all_samples="${BASE_DIR}/all_samples_constant_${SGE_TASK_ID}.txt"
if ! bcftools query -l "$input_vcf" > "$all_samples"; then
    echo "Error: Failed to extract sample IDs from $input_vcf"
    exit 1
fi

prev_samples=""
for size in "${sizes[@]}"; do
    ###### Downsample and subset #######
    if [ "$size" -eq 10000 ]; then
        # Use the original VCF for the full sample size of 50,000
        this_vcf="$input_vcf"
    else
        # Randomly sample from the original VCF file for smaller sizes
        this_samples="${BASE_DIR}/samples_constant_${size}_${SGE_TASK_ID}.txt"
        if [[ -n "$prev_samples" ]]; then
            # Find the remaining unique samples not in prev_samples
            remaining_samples="${BASE_DIR}/remaining_samples_constant_${SGE_TASK_ID}.txt"
            comm -23 <(sort "$all_samples") <(sort "$prev_samples") > "$remaining_samples"

            # Calculate the number of new samples to add
            new_samples_count=$((size - $(wc -l < "$prev_samples" | xargs)))
            if [[ $new_samples_count -le 0 ]]; then
                cp "$prev_samples" "$this_samples"
            else
                # Add new unique samples and combine with prev_samples
                shuf -n "$new_samples_count" "$remaining_samples" | cat "$prev_samples" - > "$this_samples"
            fi
            rm "$remaining_samples"  # Clean up
        else
            # First iteration: randomly sample without previous samples
            shuf -n "$size" "$all_samples" > "$this_samples"
        fi

        # Subset the VCF
        this_vcf="${BASE_DIR}/sim_output_${SGE_TASK_ID}constant_${size}.vcf.gz"
        if ! bcftools view -S "$this_samples" -o "$this_vcf" -O z "$input_vcf"; then
            echo "Error: Failed to create subset VCF for size $size"
            exit 1
        fi
    fi

    ###### Run things I want #######
    # Extract info fields from the sampled VCF using bcftools query
    bcftools query \
        -f '%CHROM\t%POS\t%ID\t%REF\t%ALT\t%INFO/S\t%INFO/DOM\t%INFO/PO\t%INFO/TO\t%INFO/MT\t%INFO/AC\t%INFO/DP\n
' \
        "$this_vcf" > "${BASE_DIR}/vcf_info_constant_${size}_${SGE_TASK_ID}.txt"

    # Run PLINK's Hardy-Weinberg equilibrium test on the sampled VCF
    plink --vcf "$this_vcf" --hardy --out "${BASE_DIR}/dominance_hwe_testing_constant_${size}_${SGE_TASK_ID}"

    # Delete the sampled VCF file after processing, except for the original large VCF file
    if [ "$size" -ne 10000 ]; then
        rm "$this_vcf"
    fi

    # Update previous sample list
    prev_samples="$this_samples"
done
```

```
echo "Downsampling completed successfully!"

# Delete no longer needed files
rm $input_vcf
rm ${BASE_DIR}/all_samples_${SGE_TASK_ID}.txt ##### NEED TO FIX IF RUNNING AGAIN
rm ${BASE_DIR}/samples_*_${SGE_TASK_ID}.txt
rm ${BASE_DIR}/dominance_hwe_testing_*_${SGE_TASK_ID}.log
rm ${BASE_DIR}/dominance_hwe_testing_*_${SGE_TASK_ID}.nosex

end_time=$(date +%s)
echo "Execution time: $(($end_time - $start_time)) seconds"
```

Prep data for plotting

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(readr)
library(gsubfn)
```

```
## Loading required package: proto
```

```
## Warning in fun(libname, pkgname): couldn't connect to display ":0"
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(stats)
df = data.frame()
for (s in c("10", "50", "100", "500", "1000", "5000", "10000")) {
  for (rep in 1:20) {
    for (demo in c("_", "_constant_") ) {
      ### VCF Info for each SNP
      infoname = paste("/u/scratch/e/eewade/swetha-sims123/vcf_info", demo, s, "_", rep, ".txt", sep="")
      # if replicate or sample size doesn't exist yet just continue
      if (!file.exists(infoname)) {
        next
      }
      print(paste(s, "-", rep, "-", demo))
      info = fread(infoname,h=F,stringsAsFactors = F,col.names=c("CHROM", "POS","ID", "REF", "ALT","S", "DOM", "P
O","TO","MT","AC","DP"))
      ### HWE Results
      hweresults = paste("/u/scratch/e/eewade/swetha-sims123/dominance_hwe_testing", demo, s, "_", rep, ".hwe", s
ep="")
    # Read the file, skipping the header
      hwe <- fread(hweresults, skip = 1, header = FALSE,col.names = c("CHR", "SNP", "TEST", "A1", "A2", "GENO", "
O_HET", "E_HET", "P"))

      table = cbind(info, hwe)
      table = table[table$AC > 1,] # removing singletons and 0s! will always be p > 0.05
      table$ss = s
      table$rep = rep
      table$demo = demo
      #table$P_ADJ_BF <- p.adjust(table$P, method = "bonferroni") ## bonferroni correcting p-values within each s
ample size
      #table$P_ADJ_FDR <- p.adjust(table$P, method = "fdr") ## bonferroni correcting p-values within each sample
size
      df = rbind(df, table)
      rm(table)
    }
  }
}
```

```
## [1] "10 - 1 - _"
## [1] "10 - 1 - _constant_"
## [1] "10 - 2 - _"
## [1] "10 - 2 - _constant_"
## [1] "10 - 3 - _"
## [1] "10 - 3 - _constant_"
## [1] "10 - 4 - _"
## [1] "10 - 5 - _"
## [1] "10 - 5 - _constant_"
## [1] "10 - 6 - _"
## [1] "10 - 6 - _constant_"
## [1] "10 - 7 - _"
## [1] "10 - 8 - _"
## [1] "10 - 9 - _"
## [1] "10 - 9 - _constant_"
## [1] "10 - 10 - _"
## [1] "10 - 11 - _"
## [1] "10 - 11 - _constant_"
## [1] "10 - 12 - _"
## [1] "10 - 12 - _constant_"
## [1] "10 - 13 - _"
## [1] "10 - 14 - _"
## [1] "10 - 14 - _constant_"
## [1] "10 - 15 - _"
## [1] "10 - 15 - _constant_"
## [1] "10 - 16 - _"
## [1] "10 - 17 - _"
## [1] "10 - 17 - _constant_"
## [1] "10 - 18 - _"
## [1] "10 - 19 - _"
## [1] "10 - 20 - _"
## [1] "10 - 20 - _constant_"
## [1] "50 - 1 - _"
## [1] "50 - 1 - _constant_"
## [1] "50 - 2 - _"
## [1] "50 - 2 - _constant_"
## [1] "50 - 3 - _"
## [1] "50 - 3 - _constant_"
## [1] "50 - 4 - _"
## [1] "50 - 5 - _"
## [1] "50 - 5 - _constant_"
```

```
## [1] "50 - 6 - _"
## [1] "50 - 6 - _constant_"
## [1] "50 - 7 - _"
## [1] "50 - 8 - _"
## [1] "50 - 9 - _"
## [1] "50 - 9 - _constant_"
## [1] "50 - 10 - _"
## [1] "50 - 11 - _"
## [1] "50 - 11 - _constant_"
## [1] "50 - 12 - _"
## [1] "50 - 12 - _constant_"
## [1] "50 - 13 - _"
## [1] "50 - 14 - _"
## [1] "50 - 14 - _constant_"
## [1] "50 - 15 - _"
## [1] "50 - 15 - _constant_"
## [1] "50 - 16 - _"
## [1] "50 - 17 - _"
## [1] "50 - 17 - _constant_"
## [1] "50 - 18 - _"
## [1] "50 - 19 - _"
## [1] "50 - 20 - _"
## [1] "50 - 20 - _constant_"
## [1] "100 - 1 - _"
## [1] "100 - 1 - _constant_"
## [1] "100 - 2 - _"
## [1] "100 - 2 - _constant_"
## [1] "100 - 3 - _"
## [1] "100 - 3 - _constant_"
## [1] "100 - 4 - _"
## [1] "100 - 5 - _"
## [1] "100 - 5 - _constant_"
## [1] "100 - 6 - _"
## [1] "100 - 6 - _constant_"
## [1] "100 - 7 - _"
## [1] "100 - 8 - _"
## [1] "100 - 9 - _"
## [1] "100 - 9 - _constant_"
## [1] "100 - 10 - _"
## [1] "100 - 11 - _"
## [1] "100 - 11 - _constant_"
## [1] "100 - 12 - _"
## [1] "100 - 12 - _constant_"
## [1] "100 - 13 - _"
## [1] "100 - 14 - _"
## [1] "100 - 14 - _constant_"
## [1] "100 - 15 - _"
## [1] "100 - 15 - _constant_"
## [1] "100 - 16 - _"
## [1] "100 - 17 - _"
## [1] "100 - 17 - _constant_"
## [1] "100 - 18 - _"
## [1] "100 - 19 - _"
## [1] "100 - 20 - _"
## [1] "100 - 20 - _constant_"
## [1] "500 - 1 - _"
## [1] "500 - 1 - _constant_"
## [1] "500 - 2 - _"
## [1] "500 - 2 - _constant_"
## [1] "500 - 3 - _"
## [1] "500 - 3 - _constant_"
## [1] "500 - 4 - _"
## [1] "500 - 5 - _"
## [1] "500 - 5 - _constant_"
## [1] "500 - 6 - _"
## [1] "500 - 6 - _constant_"
## [1] "500 - 7 - _"
## [1] "500 - 8 - _"
## [1] "500 - 9 - _"
## [1] "500 - 9 - _constant_"
## [1] "500 - 10 - _"
## [1] "500 - 11 - _"
## [1] "500 - 11 - _constant_"
## [1] "500 - 12 - _"
## [1] "500 - 12 - _constant_"
## [1] "500 - 13 - _"
## [1] "500 - 14 - _"
## [1] "500 - 14 - _constant_"
## [1] "500 - 15 - _"
```

```
## [1] "500 - 15 - _constant_"
## [1] "500 - 16 - _"
## [1] "500 - 17 - _"
## [1] "500 - 17 - _constant_"
## [1] "500 - 18 - _"
## [1] "500 - 19 - _"
## [1] "500 - 20 - _"
## [1] "500 - 20 - _constant_"
## [1] "1000 - 1 - _"
## [1] "1000 - 1 - _constant_"
## [1] "1000 - 2 - _"
## [1] "1000 - 2 - _constant_"
## [1] "1000 - 3 - _"
## [1] "1000 - 3 - _constant_"
## [1] "1000 - 4 - _"
## [1] "1000 - 5 - _"
## [1] "1000 - 5 - _constant_"
## [1] "1000 - 6 - _"
## [1] "1000 - 6 - _constant_"
## [1] "1000 - 7 - _"
## [1] "1000 - 8 - _"
## [1] "1000 - 9 - _"
## [1] "1000 - 9 - _constant_"
## [1] "1000 - 10 - _"
## [1] "1000 - 11 - _"
## [1] "1000 - 11 - _constant_"
## [1] "1000 - 12 - _"
## [1] "1000 - 12 - _constant_"
## [1] "1000 - 13 - _"
## [1] "1000 - 14 - _"
## [1] "1000 - 14 - _constant_"
## [1] "1000 - 15 - _"
## [1] "1000 - 15 - _constant_"
## [1] "1000 - 16 - _"
## [1] "1000 - 17 - _"
## [1] "1000 - 17 - _constant_"
## [1] "1000 - 18 - _"
## [1] "1000 - 19 - _"
## [1] "1000 - 20 - _"
## [1] "1000 - 20 - _constant_"
## [1] "5000 - 1 - _"
## [1] "5000 - 1 - _constant_"
## [1] "5000 - 2 - _"
## [1] "5000 - 2 - _constant_"
## [1] "5000 - 3 - _"
## [1] "5000 - 4 - _"
## [1] "5000 - 5 - _"
## [1] "5000 - 5 - _constant_"
## [1] "5000 - 6 - _"
## [1] "5000 - 6 - _constant_"
## [1] "5000 - 7 - _"
## [1] "5000 - 8 - _"
## [1] "5000 - 9 - _"
## [1] "5000 - 9 - _constant_"
## [1] "5000 - 10 - _"
## [1] "5000 - 11 - _"
## [1] "5000 - 11 - _constant_"
## [1] "5000 - 12 - _"
## [1] "5000 - 12 - _constant_"
## [1] "5000 - 13 - _"
## [1] "5000 - 14 - _"
## [1] "5000 - 14 - _constant_"
## [1] "5000 - 15 - _"
## [1] "5000 - 15 - _constant_"
## [1] "5000 - 16 - _"
## [1] "5000 - 17 - _"
## [1] "5000 - 17 - _constant_"
## [1] "5000 - 18 - _"
## [1] "5000 - 19 - _"
## [1] "5000 - 20 - _"
## [1] "5000 - 20 - _constant_"
## [1] "10000 - 1 - _"
## [1] "10000 - 1 - _constant_"
## [1] "10000 - 2 - _"
## [1] "10000 - 2 - _constant_"
## [1] "10000 - 3 - _"
## [1] "10000 - 4 - _"
## [1] "10000 - 5 - _"
## [1] "10000 - 5 - _constant_"
```

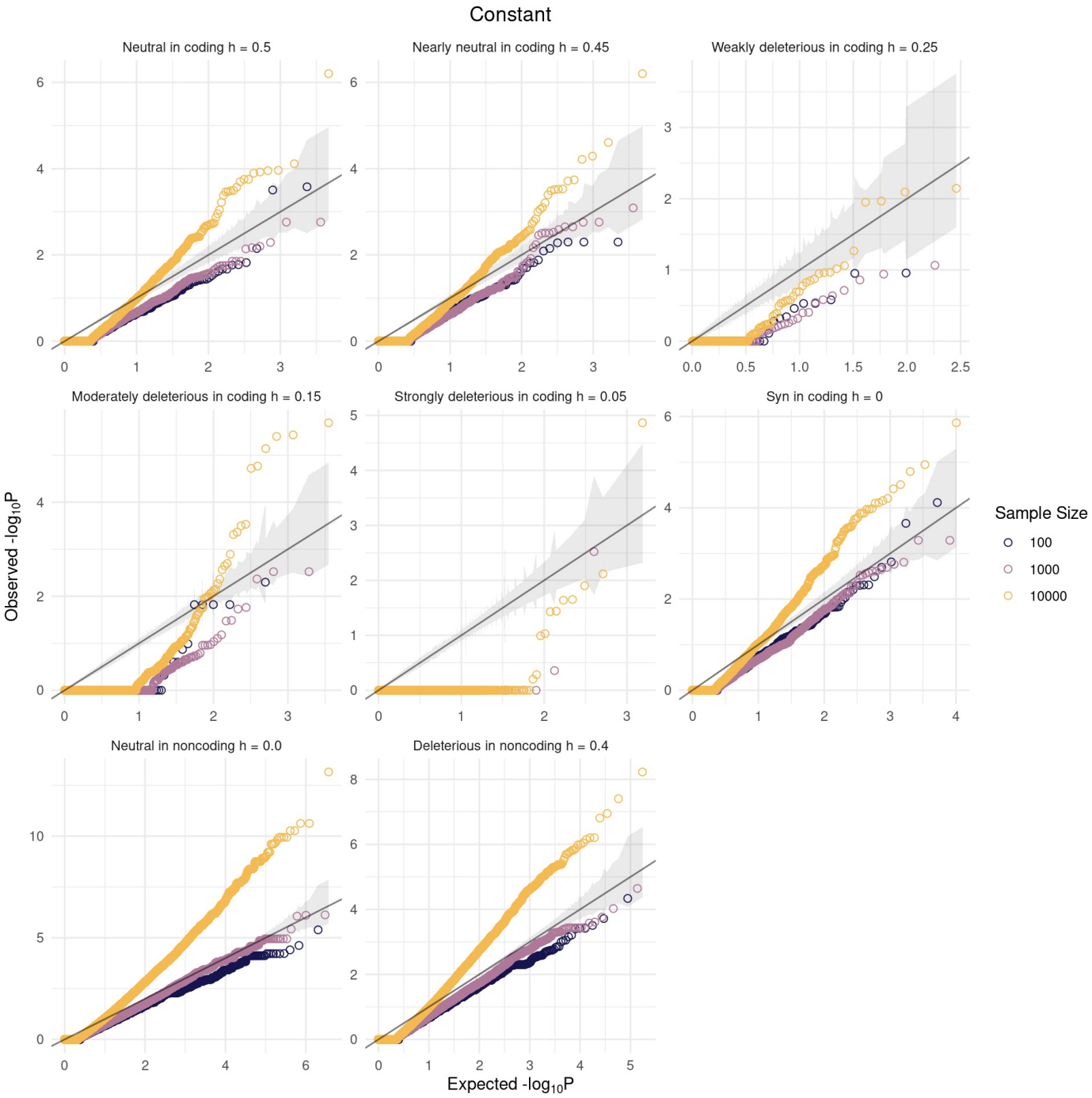
```
## [1] "10000 - 6 - _"
## [1] "10000 - 6 - _constant_"
## [1] "10000 - 7 - _"
## [1] "10000 - 8 - _"
## [1] "10000 - 9 - _"
## [1] "10000 - 9 - _constant_"
## [1] "10000 - 10 - _"
## [1] "10000 - 11 - _"
## [1] "10000 - 11 - _constant_"
## [1] "10000 - 12 - _"
## [1] "10000 - 12 - _constant_"
## [1] "10000 - 13 - _"
## [1] "10000 - 14 - _"
## [1] "10000 - 14 - _constant_"
## [1] "10000 - 15 - _"
## [1] "10000 - 15 - _constant_"
## [1] "10000 - 16 - _"
## [1] "10000 - 17 - _"
## [1] "10000 - 17 - _constant_"
## [1] "10000 - 18 - _"
## [1] "10000 - 19 - _"
## [1] "10000 - 20 - _"
## [1] "10000 - 20 - _constant_"
```

```
df$inarea = "Coding"
df$inarea[df$MT %in% c("7", "10")] = "Noncoding"

#defineConstant("h_neut", 0.5);
#defineConstant("h_nearNeut", 0.45);
#defineConstant("h_wkDel", 0.25);
#defineConstant("h_modDel", 0.15);
#defineConstant("h_strDel", 0.05);

#initializeMutationType("m1", h_neut, "s", "return runif(1, -2e-5, 0.0);");
#initializeMutationType("m2", h_nearNeut, "s", "return runif(1, -2e-4, -2e-5);");
#initializeMutationType("m3", h_wkDel, "s", "return runif(1, -2e-3, -2e-4);");
#initializeMutationType("m4", h_modDel, "s", "return runif(1, -2e-2, -2e-3);");
#initializeMutationType("m5", h_strDel, "s", "return runif(1, -1.0, -2e-2);");
#initializeMutationType("m6", h_neut, "f", 0.0); // add sixth mutation type representing synonymous mutations

long_MT = c(
  "1" = "Neutral in coding h = 0.5",
  "2" = "Nearly neutral in coding h = 0.45",
  "3" = "Weakly deleterious in coding h = 0.25",
  "4" = "Moderately deleterious in coding h = 0.15",
  "5" = "Strongly deleterious in coding h = 0.05",
  "6" = "Syn in coding h = 0",
  "7" = "Neutral in noncoding h = 0.0",
  "10" = "Deleterious in noncoding h = 0.4"
)

df$long_MT = long_MT[df$MT]
library(data.table)
df$long_MT[df$MT == "10"] = "Deleterious in noncoding h = 0.4" #don't feel like figuring out what's wrong
mt_list <- unname(long_MT)


head(df)
```

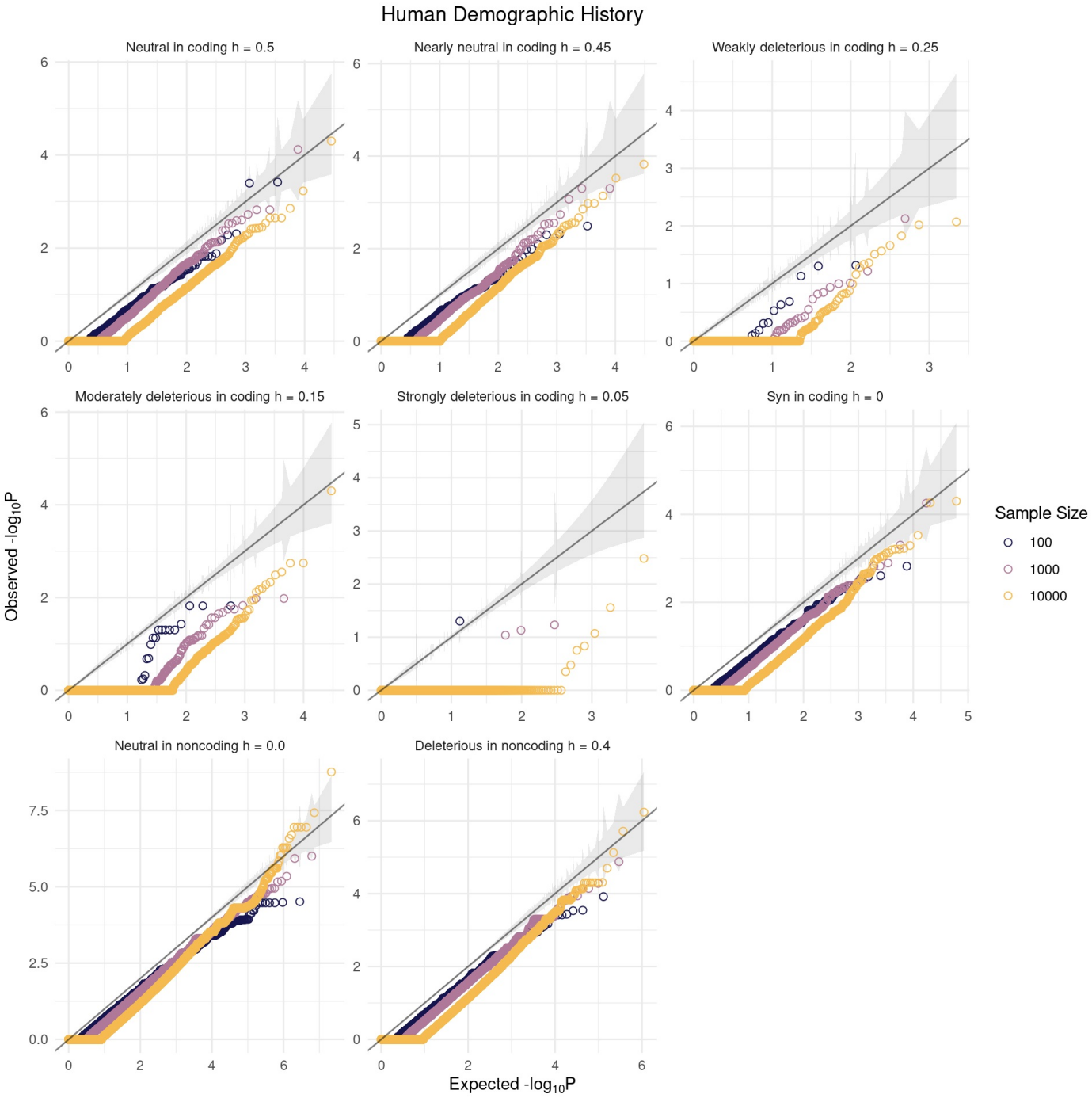
```
##    CHROM  POS ID REF ALT S DOM PO     TO MT AC   DP CHR SNP    TEST A1 A2  GENO
## 1:     1  226  .   A   T 0 0.5  1 112983  7 10 1000   1   . ALL(NP)  T  A 2/6/2
## 2:     1 1707  .   A   T 0 0.5  1 122084  7  6 1000   1   . ALL(NP)  T  A 1/4/5
## 3:     1 2161  .   A   T 0 0.5  1 112134  7  8 1000   1   . ALL(NP)  T  A 1/6/3
## 4:     1 2611  .   A   T 0 0.5  1 113872  7 10 1000   1   . ALL(NP)  T  A 2/6/2
## 5:     1 2968  .   A   T 0 0.5  1  96152  7 10 1000   1   . ALL(NP)  T  A 2/6/2
## 6:     1 4752  .   A   T 0 0.5  1  90237  7 10 1000   1   . ALL(NP)  T  A 2/6/2
##    O_HET E_HET P ss rep demo    inarea                     long_MT
## 1:   0.6  0.50 1 10   1    _ Noncoding Neutral in noncoding h = 0.0
## 2:   0.4  0.42 1 10   1    _ Noncoding Neutral in noncoding h = 0.0
## 3:   0.6  0.48 1 10   1    _ Noncoding Neutral in noncoding h = 0.0
## 4:   0.6  0.50 1 10   1    _ Noncoding Neutral in noncoding h = 0.0
## 5:   0.6  0.50 1 10   1    _ Noncoding Neutral in noncoding h = 0.0
## 6:   0.6  0.50 1 10   1    _ Noncoding Neutral in noncoding h = 0.0
```

QQ plot

```r
library(MetBrewer)

#' @param ps Vector of p-values.
#' @param ci Size of the confidence interval, 95% by default.
#' @return A ggplot2 plot.
#' @examples
#' library(ggplot2)
#' gg_qqplot(runif(1e2)) + theme_grey(base_size = 24)
library(ggplot2)

library(ggplot2)
library(dplyr)

# Define the QQ plot function
gg_qqplot <- function(data, ci = 0.95) {
  data <- data %>%
    group_by(ss, MT) %>%
    mutate(
      observed = -log10(sort(P)),
      expected = -log10(ppoints(n())),
      clower   = -log10(qbeta(p = (1 - ci) / 2, shape1 = 1:n(), shape2 = n():1)),
      cupper   = -log10(qbeta(p = (1 + ci) / 2, shape1 = 1:n(), shape2 = n():1))
    )

  log10Pe <- expression(paste("Expected -log"[10], plain(P)))
  log10Po <- expression(paste("Observed -log"[10], plain(P)))

  ggplot(data) +
    geom_ribbon(
      mapping = aes(x = expected, ymin = clower, ymax = cupper),
      alpha = 0.1
    ) +
    geom_point(aes(expected, observed, color = factor(ss, levels = c("10", "50", "100", "500", "1000", "5000", "1
0000"))), shape = 1, size = 2) +
    geom_abline(intercept = 0, slope = 1, alpha = 0.5) +
    xlab(log10Pe) +
    ylab(log10Po) +
    theme_minimal() +
    facet_wrap(factor(long_MT, levels=mt_list) ~ ., scales = "free") + # Facet grid by MT values
    labs(color="Sample Size")+
    theme(plot.title = element_text(hjust = 0.5))
}

gg_qqplot(df[df$demo == "_constant_" & df$ss %in% c("100", "1000", "10000")])+
   scale_color_manual(values=met.brewer("Renoir", 3))+labs(title="Constant")
```




```
gg_qqplot(df[df$demo == "_" & df$ss %in% c("100", "1000", "10000")])+
   scale_color_manual(values=met.brewer("Renoir", 3))+labs(title="Human Demographic History")
```



Proportions

Play with colors for p-values

continue palette met.brewer("Troy", n=15, type="continuous")

```
library(dplyr)
library(tidyr)
library(MetBrewer)

result_table <- df[which(df$AC > 2)] %>%
  group_by(MT, ss, demo) %>%
  summarise(
    Total = n(),
    Prop_P_less_0.05 = sum(P < 0.05, na.rm = TRUE)/n(),
    Num_P_less_0.05 = sum(P < 0.05, na.rm = TRUE)

    #Percentage_P_less_0.05 = Count_P_less_0.05 / Total * 100,
    #Percentage_P_greater_0.05 = Count_P_greater_0.05 / Total * 100
  ) %>%
  arrange(MT, ss, demo)
```

```
## `summarise()` has grouped output by 'MT', 'ss'. You can override using the `.groups` argument.
```

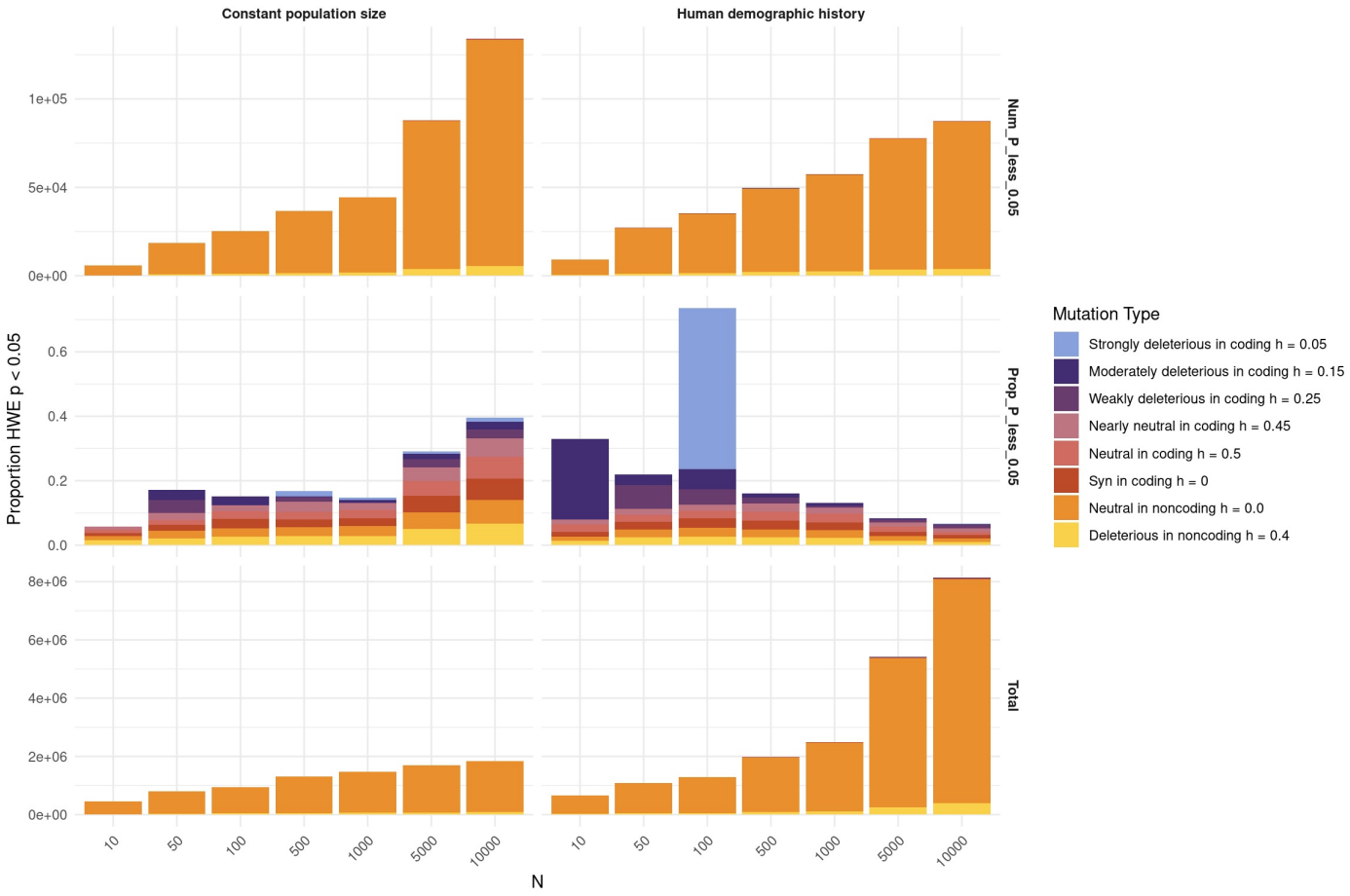
```
# Print the result
print(result_table)
```

```
## # A tibble: 110 × 6
## # Groups:   MT, ss [55]
##       MT ss    demo       Total Prop_P_less_0.05 Num_P_less_0.05
##    <int> <chr> <chr>      <int>            <dbl>           <int>
##  1     1 10    _            755          0.0238               18
##  2     1 10    _constant_   471          0.0106                5
##  3     1 100   _           1504          0.0239               36
##  4     1 100   _constant_  1034          0.0222               23
##  5     1 1000  _           2934          0.0273               80
##  6     1 1000  _constant_  1678          0.0256               43
##  7     1 10000 _          10069          0.00963              97
##  8     1 10000 _constant_  2219          0.0685              152
##  9     1 50    _           1257          0.0207               26
## 10     1 50    _constant_   870          0.0149               13
## # … with 100 more rows
```

```
# Reshape the data for plotting
plot_data <- result_table %>%
  pivot_longer(cols = c(Prop_P_less_0.05, Num_P_less_0.05, Total),
               names_to = "P_category",
               values_to = "Count")

plot_data$long_MT = long_MT[plot_data$MT]
plot_data$long_MT[plot_data$MT == 10] = "Deleterious in noncoding h = 0.4"
plot_data$pretty_demo[plot_data$demo == "_"] = "Human demographic history"
```

```
## Warning: Unknown or uninitialised column: `pretty_demo`.
```

```
plot_data$pretty_demo[plot_data$demo == "_constant_"] = "Constant population size"

mt_order = c("Strongly deleterious in coding h = 0.05","Moderately deleterious in coding h = 0.15","Weakly delete
rious in coding h = 0.25","Nearly neutral in coding h = 0.45", "Neutral in coding h = 0.5","Syn in coding h = 0",
"Neutral in noncoding h = 0.0","Deleterious in noncoding h = 0.4")

# Create the plot
ggplot(plot_data, aes(x = factor(ss, levels=c("10", "50", "100", "500", "1000", "5000", "10000")), y = Count, fil
l = factor(long_MT,mt_order))) +
  geom_bar(stat = "identity", position = "stack") +
  facet_grid(factor(P_category)~factor(pretty_demo), scales = "free") +
  scale_fill_manual(values = met.brewer("Archambault", n=8))+
  labs(x = "N",
       y = "Proportion HWE p < 0.05 ",
       fill = "Mutation Type") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        strip.text = element_text(face = "bold"))
```




```
result_table <- df %>%
  group_by(MT, ss, demo, rep) %>%
  summarise(
    Total = n(),
    Prop_P_less_0.05 = sum(P < 0.05, na.rm = TRUE)/n(),
    Num_P_less_0.05 = sum(P < 0.05, na.rm = TRUE)

    #Percentage_P_less_0.05 = Count_P_less_0.05 / Total * 100,
    #Percentage_P_greater_0.05 = Count_P_greater_0.05 / Total * 100
  ) %>%
  arrange(MT, ss, demo, rep)
```

```
## `summarise()` has grouped output by 'MT', 'ss', 'demo'. You can override using the `.groups` argument.
```

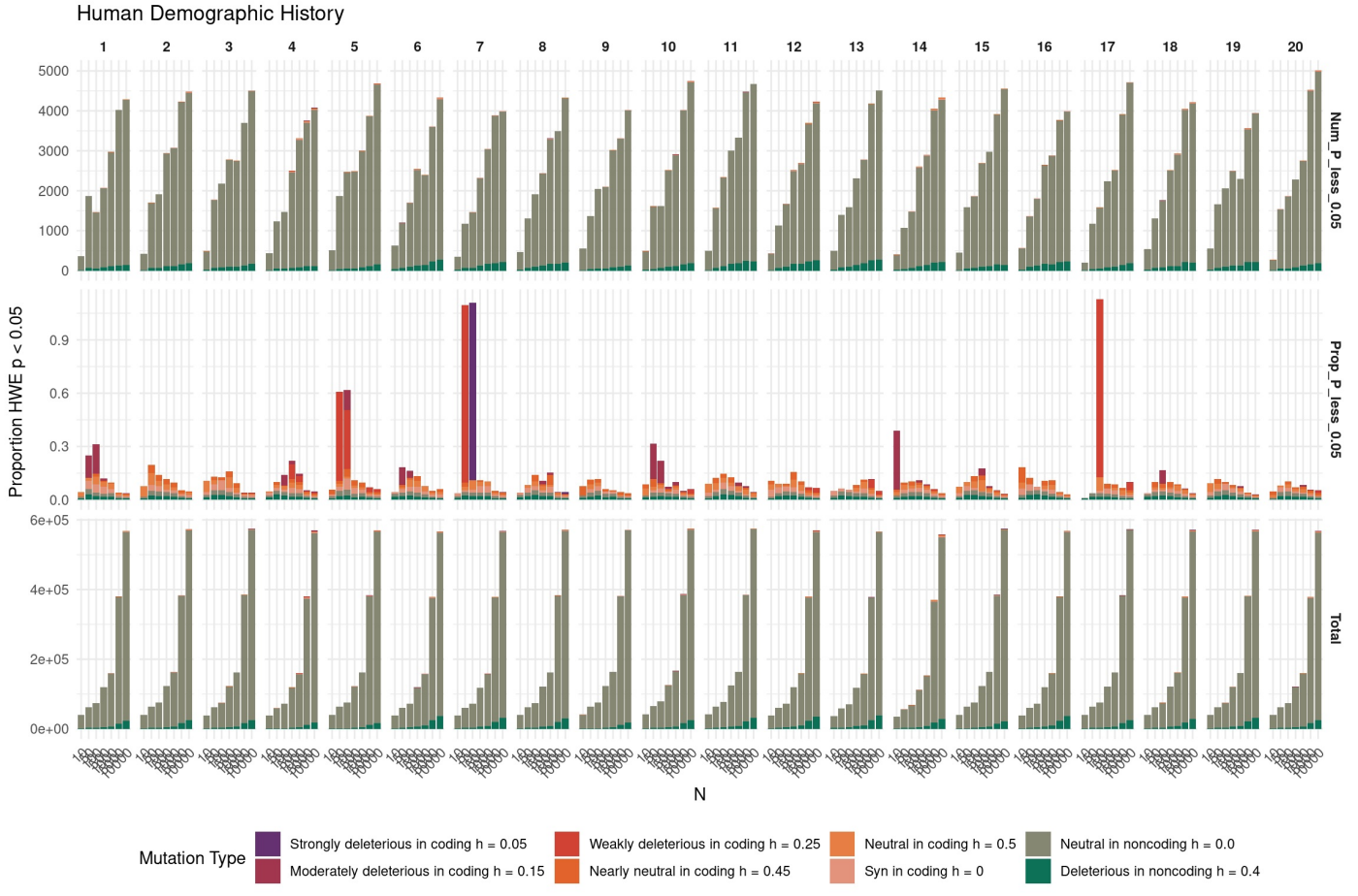
```
# Print the result
print(result_table)
```

```
## # A tibble: 1,660 × 7
## # Groups:   MT, ss, demo [111]
##       MT ss    demo    rep Total Prop_P_less_0.05 Num_P_less_0.05
##    <int> <chr> <chr> <int> <int>            <dbl>           <int>
##  1     1 10    _         1    36          0.0278                1
##  2     1 10    _         2    34          0.0588                2
##  3     1 10    _         3    32          0.0625                2
##  4     1 10    _         4   117          0.00855               1
##  5     1 10    _         5    32          0                     0
##  6     1 10    _         6    66          0                     0
##  7     1 10    _         7    24          0                     0
##  8     1 10    _         8    38          0                     0
##  9     1 10    _         9    30          0                     0
## 10     1 10    _        10    53          0.0377                2
## # … with 1,650 more rows
```

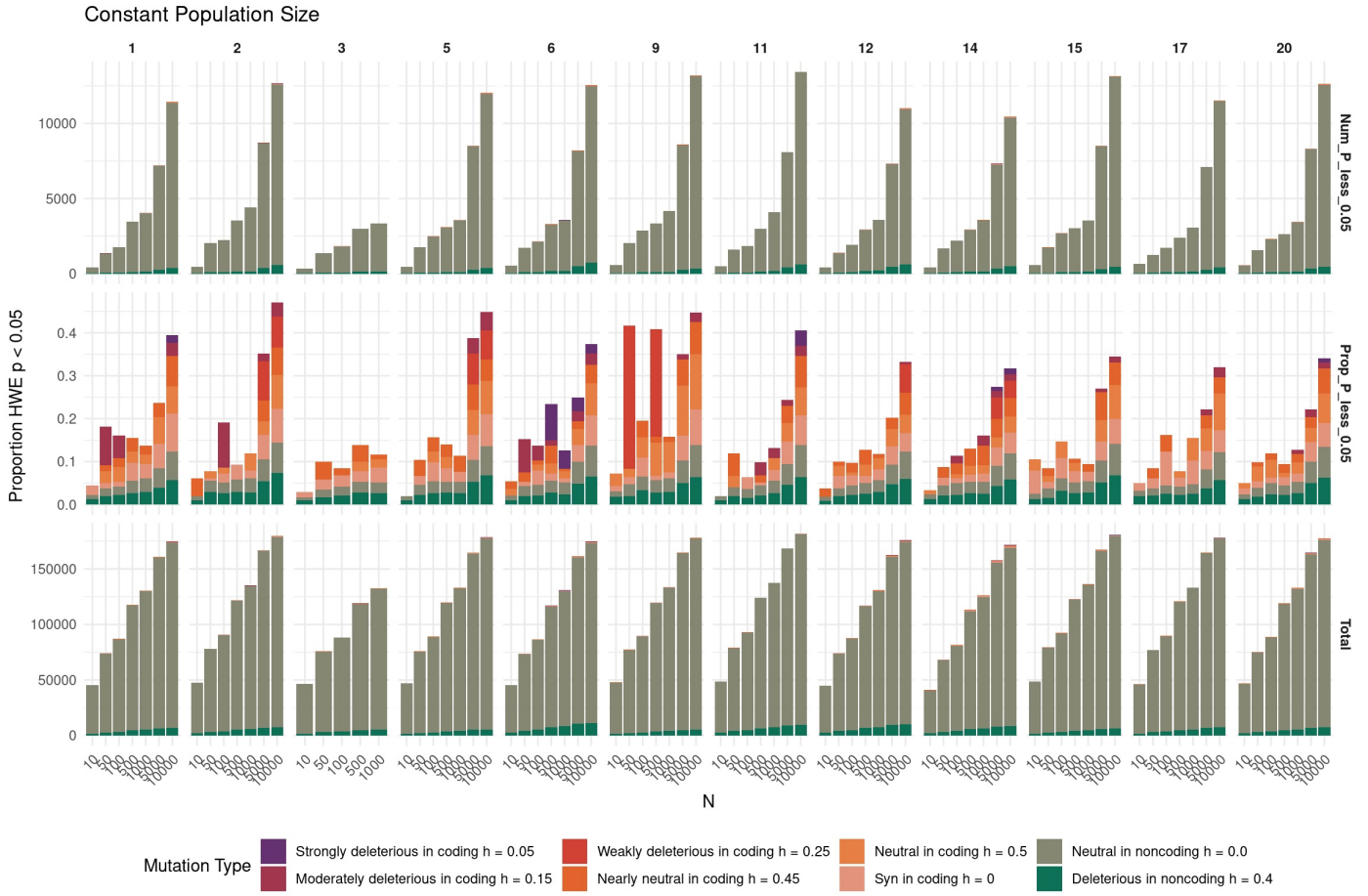
```
# Reshape the data for plotting
plot_data <- result_table %>%
  pivot_longer(cols = c(Prop_P_less_0.05, Num_P_less_0.05, Total),
               names_to = "P_category",
               values_to = "Count")

plot_data$long_MT = long_MT[plot_data$MT]
plot_data$long_MT[plot_data$MT == 10] = "Deleterious in noncoding h = 0.4"
plot_data$pretty_demo[plot_data$demo == "_"] = "Human demographic history"
```

```
## Warning: Unknown or uninitialised column: `pretty_demo`.
```

```
plot_data$pretty_demo[plot_data$demo == "_constant_"] = "Constant population size"

mt_order = c("Strongly deleterious in coding h = 0.05","Moderately deleterious in coding h = 0.15","Weakly delete
rious in coding h = 0.25","Nearly neutral in coding h = 0.45", "Neutral in coding h = 0.5","Syn in coding h = 0",
"Neutral in noncoding h = 0.0","Deleterious in noncoding h = 0.4")


# Create the plot
ggplot(plot_data[plot_data$pretty_demo == "Human demographic history",], aes(x = factor(ss, levels=c("10", "50",
"100", "500", "1000", "5000", "10000")), y = Count, fill = factor(long_MT, levels=mt_order))) +
  geom_bar(stat = "identity", position = "stack") +
  facet_grid(factor(P_category)~factor(rep), scales = "free") +
  scale_fill_manual(values = met.brewer("Java", n=8))+
  labs(x = "N",
       y = "Proportion HWE p < 0.05 ",
       fill = "Mutation Type",
       title="Human Demographic History") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        strip.text = element_text(face = "bold"),legend.position = "bottom")
```



```
ggplot(plot_data[plot_data$pretty_demo != "Human demographic history",], aes(x = factor(ss, levels=c("10", "50",
"100", "500", "1000", "5000", "10000")), y = Count, fill = factor(long_MT, levels =mt_order))) +
  geom_bar(stat = "identity", position = "stack") +
  facet_grid(factor(P_category)~factor(rep), scales = "free") +
  scale_fill_manual(values = met.brewer("Java", n=8))+
  labs(x = "N",
       y = "Proportion HWE p < 0.05 ",
       fill = "Mutation Type",
       title="Constant Population Size") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        strip.text = element_text(face = "bold"),legend.position = "bottom")
```




```
result_table <- df %>%
  group_by(MT, ss, demo) %>%
  summarise(
    Total = n(),
    Prop_P_less_0.05 = sum(P < 0.05, na.rm = TRUE)/n(),
    Num_P_less_0.05 = sum(P < 0.05, na.rm = TRUE)

    #Percentage_P_less_0.05 = Count_P_less_0.05 / Total * 100,
    #Percentage_P_greater_0.05 = Count_P_greater_0.05 / Total * 100
  ) %>%
  arrange(MT, ss, demo)
```

```
## `summarise()` has grouped output by 'MT', 'ss'. You can override using the `.groups` argument.
```

```
# Print the result
print(result_table)
```

```
## # A tibble: 111 × 6
## # Groups:   MT, ss [56]
##       MT ss    demo       Total Prop_P_less_0.05 Num_P_less_0.05
##    <int> <chr> <chr>      <int>            <dbl>           <int>
##  1     1 10    _            898          0.0200              18
##  2     1 10    _constant_   591          0.00846              5
##  3     1 100   _           1742          0.0207              36
##  4     1 100   _constant_  1176          0.0196              23
##  5     1 1000  _           3854          0.0208              80
##  6     1 1000  _constant_  1821          0.0236              43
##  7     1 10000 _          14205          0.00690             98
##  8     1 10000 _constant_  2350          0.0647             152
##  9     1 50    _           1465          0.0191              28
## 10     1 50    _constant_  1000          0.013               13
## # … with 101 more rows
```

```
# Reshape the data for plotting
plot_data <- result_table %>%
  pivot_longer(cols = c(Prop_P_less_0.05, Num_P_less_0.05, Total),
               names_to = "P_category",
               values_to = "Count")

plot_data$long_MT = long_MT[plot_data$MT]
plot_data$long_MT[plot_data$MT == 10] = "Deleterious in noncoding h = 0.4"
plot_data$pretty_demo[plot_data$demo == "_"] = "Human demographic history"
```

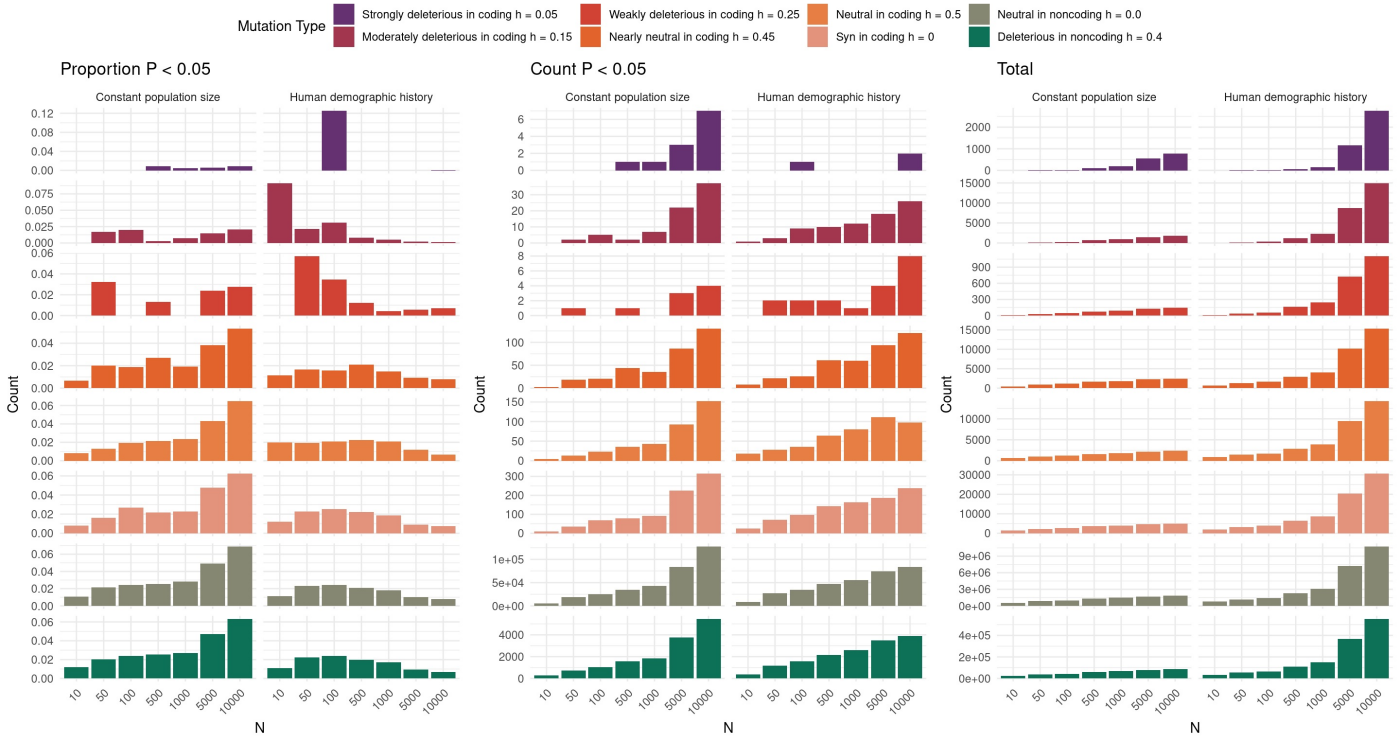
```
## Warning: Unknown or uninitialised column: `pretty_demo`.
```

```
plot_data$pretty_demo[plot_data$demo == "_constant_"] = "Constant population size"

mt_order = c("Strongly deleterious in coding h = 0.05","Moderately deleterious in coding h = 0.15","Weakly delete
rious in coding h = 0.25","Nearly neutral in coding h = 0.45", "Neutral in coding h = 0.5","Syn in coding h = 0",
"Neutral in noncoding h = 0.0","Deleterious in noncoding h = 0.4")


g1 = ggplot(plot_data[plot_data$P_category == "Prop_P_less_0.05",], aes(x = factor(ss, levels=c("10", "50", "100"
, "500", "1000", "5000", "10000")), y = Count, fill = factor(long_MT, levels=mt_order))) +
  geom_bar(stat = "identity", position = "stack") +
  facet_grid(rows=vars(factor(long_MT, levels=mt_order)), cols=vars(pretty_demo), scales = "free") +
  scale_fill_manual(values = met.brewer("Java", n=8))+
  labs(x = "N",
       fill = "Mutation Type",
       title="Proportion P < 0.05") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        strip.text.y = element_blank(),legend.position="none")

g2 = ggplot(plot_data[plot_data$P_category == "Num_P_less_0.05",], aes(x = factor(ss, levels=c("10", "50", "100",
"500", "1000", "5000", "10000")), y = Count, fill = factor(long_MT, levels=mt_order))) +
  geom_bar(stat = "identity", position = "stack") +
  facet_grid(rows=vars(factor(long_MT, levels=mt_order)), cols=vars(pretty_demo), scales = "free") +
  scale_fill_manual(values = met.brewer("Java", n=8))+
  labs(x = "N",
       fill = "Mutation Type",
       title="Count P < 0.05") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        strip.text.y = element_blank(),legend.position ="none")

g3 = ggplot(plot_data[plot_data$P_category == "Total",], aes(x = factor(ss, levels=c("10", "50", "100", "500", "1
000", "5000", "10000")), y = Count, fill = factor(long_MT, levels=mt_order))) +
  geom_bar(stat = "identity", position = "stack") +
  facet_grid(rows=vars(factor(long_MT, levels=mt_order)), cols=vars(pretty_demo), scales = "free") +
  scale_fill_manual(values = met.brewer("Java", n=8))+
  labs(x = "N",
       fill = "Mutation Type",
       title="Total") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        strip.text.y = element_blank(),
        legend.position="none")

library(cowplot)
library(ggpubr)
```

```
##
## Attaching package: 'ggpubr'
```

```
## The following object is masked from 'package:cowplot':
##
##     get_legend
```

```
ggarrange(g1, g2, g3,nrow=1,
  common.legend = TRUE)
```




```
#library(ggplot2)
# Create the plot
#ggplot(df[ss %in% c("10", "1000", "5000"),], aes(x = S, y = DOM)) +
#  geom_point(aes(color = P < 0.05)) +
#  scale_color_manual(values = c("TRUE" = "red", "FALSE" = "blue")) +
#  labs(x = "Selection Coefficient (S)",
#       y = "Dominance (h)",
#       color = "P < 0.05") +
#  facet_wrap(~ss)+
#  theme_minimal() +
#  theme(legend.position = "bottom")

df$P_quant = " > 0.05"
df$P_quant[df$P < 1.0490e-02] = "< 1.0490e-02 (0% quantile)"
df$P_quant[df$P < 2.3770e-02 & df$P > 1.0490e-02] = "< 2.3770e-02 (25% quantile)"
df$P_quant[df$P > 2.3770e-02 & df$P < 3.7185e-02] = "< 3.7185e-02 (50% quantile)"
df$P_quant[df$P > 3.7185e-02 & df$P < 0.05] = "< 0.05 (75% quantile)"


ggplot(df[df$P_quant != " > 0.05" & df$inarea == "Coding" & demo == "_",], aes(x = S, y = DOM, color=factor(P_qua
nt, levels = c("< 1.0490e-02 (0% quantile)","< 2.3770e-02 (25% quantile)", "< 3.7185e-02 (50% quantile)", "< 0.05
(75% quantile)")))) +
  geom_point(size=3, shape=2) +
  labs(x = "Selection Coefficient (S)",
       y = "Dominance (h)") +
  facet_wrap(~factor(ss, levels=c("10","50","100","500","1000","5000","10000","20000","50000")))+
  theme_minimal() +
  labs(color="P")+
  scale_color_manual(values=met.brewer("Hokusai1"))+
  theme(legend.position = "bottom")
```