Project Code Compilation

Source Directory: C:\Users\aaisu\Box\Carter Lab\Projects\Anstep_GA_Yemen\code_repo

Section: Root Directory

File: code_to_pdf.py

```
import os
import nbformat
from fpdf import FPDF

# --- Configuration ---
# You can add or remove file extensions here
SUPPORTED_EXTENSIONS = ('.py', '.sh', '.ipynb')
CODE_FONT_SIZE = 10
LINE_HEIGHT = 5  # PDF line height for code

class PDF(FPDF):
    """
    Custom PDF class to add a header and footer for page numbers.
    """
    def header(self):
        # We don't want a header on every page, so we leave this blank.
        pass

    def footer(self):
        # Add a page number to the bottom
        self.set_y(-15)  # Position 1.5 cm from bottom
        self.set_font('Helvetica', 'I', 8)
        self.cell(0, 10, f'Page {self.page_no()}', 0, 0, 'C')

def safe_text(text):
    """
    Encodes text to 'latin-1' with replacements.
    This prevents fpdf from crashing on unsupported Unicode characters
    by replacing them with a '?'.
    """
    return text.encode('latin-1', 'replace').decode('latin-1')

def get_code_from_notebook(file_path):
    """
    Extracts and concatenates code from all code cells in a .ipynb file.
    """
    code = ""
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            # Read the notebook using nbformat
            notebook = nbformat.read(f, as_version=nbformat.NO_CONVERT)

         code_cells = [cell['source'] for cell in notebook['cells'] if cell['cell_type']
== 'code']

        if not code_cells:
            return "# No code cells found in this notebook."
        for i, cell_source in enumerate(code_cells):
            code += f"# === Code Cell {i+1} ===\n"
            code += cell_source
            code += "\n\n# === End of Cell ===\n\n"

        return code
    except Exception as e:
        print(f"  [!] Error reading notebook {file_path}: {e}")
        return f"# Error reading notebook: {e}"

def get_code_from_text(file_path):
    """
    Reads code from a plain text file (.py, .sh, etc.).
    """
    try:
        with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
            return f.read()
    except Exception as e:
        print(f"  [!] Error reading file {file_path}: {e}")
        return f"# Error reading file: {e}"

def create_code_pdf(root_dir, output_pdf_file):
    """
    Main function to walk directories and generate the PDF.
    """
    pdf = PDF('P', 'mm', 'A4')  # Portrait, millimeters, A4 size
    pdf.set_auto_page_break(True, margin=15)
    pdf.add_page()

    # Add a main title page
    pdf.set_font('Helvetica', 'B', 24)
    pdf.cell(0, 30, 'Project Code Compilation', 0, 1, 'C')
    pdf.set_font('Helvetica', '', 12)
    pdf.cell(0, 10, f'Source Directory: {os.path.abspath(root_dir)}', 0, 1, 'C')
    pdf.ln(20)

    current_section = None

    # We sort the directories and files to ensure a consistent, alphabetical order
    for root, dirs, files in os.walk(root_dir, topdown=True):
        dirs.sort()
        files.sort()

        rel_dir = os.path.relpath(root, root_dir)

        # --- Add a New Section for Each Subfolder ---
        if rel_dir != current_section:
            current_section = rel_dir
                          section_name  =  "Root  Directory"  if  rel_dir  ==  "."  else
rel_dir.replace(os.path.sep, " > ")

            print(f"\nProcessing section: {section_name}")
            pdf.add_page()
            pdf.set_font('Helvetica', 'B', 20)
            pdf.cell(0, 15, f'Section: {section_name}', 0, 1, 'L')
            pdf.line(pdf.get_x(), pdf.get_y(), pdf.get_x() + 180, pdf.get_y())
            pdf.ln(10)

            files_found_in_section = False

        # --- Process each file in the folder ---
        for file in files:
            if file.endswith(SUPPORTED_EXTENSIONS):
                files_found_in_section = True
                file_path = os.path.join(root, file)
                print(f"  Adding file: {file}")

                # Add file title
                pdf.set_font('Helvetica', 'B', 14)
                pdf.cell(0, 10, f'File: {file}', 0, 1, 'L')
                pdf.ln(2)

                # Get code content
                if file.endswith('.ipynb'):
                    code_content = get_code_from_notebook(file_path)
                else:
                    code_content = get_code_from_text(file_path)

                # Add code content
                pdf.set_font('Courier', '', CODE_FONT_SIZE)
                # Use multi_cell for automatic wrapping and page breaks
                pdf.multi_cell(0, LINE_HEIGHT, safe_text(code_content))

                pdf.ln(10) # Add a 10mm space before the next file

        if not files_found_in_section:
            pdf.set_font('Helvetica', 'I', 12)
            pdf.cell(0, 10, 'No supported scripts found in this directory.', 0, 1, 'L')
            pdf.ln(5)

    # --- Save the PDF ---
    try:
        pdf.output(output_pdf_file)
        print(f"\n? Successfully generated PDF: {output_pdf_file}")
    except Exception as e:
        print(f"\n? Error saving PDF: {e}")

if __name__ == "__main__":
    print("--- Code-to-PDF Compiler ---")

    # Get user input for the directories
    # Note: You can hard-code these paths if you prefer.
    # example: root_directory = r"C:\Users\YourName\Projects\MyProject"
    # example: output_file = r"C:\Users\YourName\Desktop\MyProject_Code.pdf"

    root_directory = input("Enter the FULL path to your main code folder: ")
    output_file = input("Enter the FULL path for the output PDF (e.g., C:\\my_code.pdf):
```

```
")

    # Validate paths
    if not os.path.isdir(root_directory):
        print(f"Error: Directory not found: {root_directory}")
    elif not output_file.lower().endswith('.pdf'):
        print("Error: Output file name must end with .pdf")
    else:
        create_code_pdf(root_directory, output_file)
```

Section: WGS_analysis

No supported scripts found in this directory.

Section: WGS_analysis > Admixture

File: YmEthIndSml_WGS_admixture_plot_Wlocdet.py

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Input parameters
K = 3
q_file = f"final_admixture.{K}.Q"
fam_file = "final_admixture.fam"
metadata_file = "WGS_sample_loc_det.csv"
output_plot = f"YmEthSml_admixture_plot_K{K}_with_location_bars.png"

# Step 1: Load .Q file (ancestry proportions)
q_data = np.loadtxt(q_file)

# Step 2: Load .fam file (individual IDs)
fam_data = pd.read_csv(fam_file, delim_whitespace=True, header=None, usecols=[0, 1],
names=["FID", "IID"])

# Step 3: Load metadata
metadata = pd.read_csv(metadata_file)

# Step 4: Merge fam data with metadata
fam_data = fam_data.merge(metadata, left_on="IID", right_on="ind", how="left")

# Step 5: Combine ancestry proportions and individual IDs
df = pd.DataFrame(q_data, columns=[f"Cluster_{i+1}" for i in range(K)])
df["IID"] = fam_data["IID"]
df["Site"] = fam_data["Site"]
df["Country"] = fam_data["Country"]

# Step 6: Sort individuals by site and ancestry
# df_sorted = df.sort_values(by=["Country", "Site", f"Cluster_1"], ascending=[True,
True, False])
# df_sorted = df.set_index("IID").loc[fam_data["IID"]].reset_index()
# df_sorted = df.copy()  # Keep the original order

# Step 6: Sort individuals by site (or country, if you prefer)
df_sorted = df.sort_values(by=[f"Cluster_{i+1}" for i in range(K)], ascending=False)

# Step 7: Prepare plot
fig, ax = plt.subplots(figsize=(40, 8))

bottom = np.zeros(df_sorted.shape[0])
x = np.arange(len(df_sorted))

# Main admixture plot (cluster colors)
```

```
for i in range(K):
      ax.bar(x,  df_sorted[f"Cluster_{i+1}"],  bottom=bottom,  width=0.8,  label=f"Cluster
{i+1}")
    bottom += df_sorted[f"Cluster_{i+1}"]

# Step 8: Add population and country color bars on top

# Color palettes
# Define color palette for sites with enough unique colors
site_palette = sns.color_palette("husl", n_colors=df_sorted["Site"].nunique())
site_color_map = dict(zip(df_sorted["Site"].unique(), site_palette))

country_palette = sns.color_palette("Set2", n_colors=df_sorted["Country"].nunique())
country_color_map = dict(zip(df_sorted["Country"].unique(), country_palette))

# Define spacing
admixture_top = 1.0
gap_between_admixture_and_site = 0.12
site_bar_height = 0.02
gap_between_site_and_country = 0.58
country_bar_height = 0.05

# Add site color bar (with gap)
site_bar_bottom = admixture_top + gap_between_admixture_and_site
#for xi, site in zip(x, df_sorted["Site"]):
#       ax.bar(xi,  site_bar_height,  bottom=site_bar_bottom,  color=site_color_map[site],
width=1.0, edgecolor='none')

# Add site names as text labels above the bars
for xi, site in zip(x, df_sorted["Site"]):
    ax.text(xi, site_bar_bottom + (site_bar_height / 2) + 0.22, site,
            ha='center', va='center', rotation=90, fontsize=23, clip_on=False)

# Add country color bar (with gap)
country_bar_bottom = site_bar_bottom + site_bar_height + gap_between_site_and_country
for xi, country in zip(x, df_sorted["Country"]):
                       ax.bar(xi,      country_bar_height,      bottom=country_bar_bottom,
color=country_color_map[country], width=0.8, edgecolor='none')


# Step 9: Beautify plot
ax.set_title(f"Admixture of YmEthSml WGS samples (K={K})", fontsize=35)
ax.set_xlabel("Sample IDs", fontsize=26)
ax.set_ylabel("Ancestry Proportion", fontsize=26)
ax.set_xticks(x)
ax.set_xticklabels(df_sorted["IID"], rotation=90, fontsize=15)
ax.set_yticks(np.linspace(0, 1, 6))
ax.set_yticklabels(np.round(np.linspace(0, 1, 6), 2), fontsize=20)
ax.legend(loc="upper right", fontsize="small", title="Clusters")
ax.set_ylim(0, country_bar_bottom + country_bar_height + 0.05)  # Add some space

# Step 10: Create legends for site and country
from matplotlib.patches import Patch
```

```
# site_patches = [Patch(color=color, label=site) for site, color in site_color_map.items()]
country_patches = [Patch(color=color, label=country) for country, color in country_color_map.items()]

#legend1 = ax.legend(handles=site_patches, title="Site", loc='upper center', bbox_to_anchor=(0.5, -0.08), ncol=len(site_color_map), fontsize='small', title_fontsize='small')
#ax.add_artist(legend1)

legend2 = ax.legend(handles=country_patches, title="Country", loc='lower right', bbox_to_anchor=(0.9, -0.55), ncol=len(country_color_map), fontsize=20, title_fontsize=25)
ax.add_artist(legend2)

plt.tight_layout()
plt.savefig(output_plot, dpi=300, bbox_inches='tight')
plt.close()

print(f"Admixture plot saved as {output_plot}")
```

File: YmEthInd_allChrs_admixture.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=30

# locating the directory with data
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/biallelic_phased_VCFs/LD_decay

# setting background for conda environment activation
eval "$(conda shell.bash hook)"

# activating 'newbio' environment with conda
conda activate newbio

# ADMIXTURE does not accept chromosome names that are not human chromosomes. We will thus just exchange the first column by 0
awk '{$1="0";print $0}' YmEthOnly_allChrs_LD.bim > YmEthOnly_allChrs_LD.bim.tmp
mv YmEthOnly_allChrs_LD.bim.tmp YmEthOnly_allChrs_LD.bim

# Run admixture analysis for each value of K with cross validation
for K in $(seq 2 10)
do
    echo "Running admixture analysis for K=$K..."
    admixture --cv -j30 YmEthOnly_allChrs_LD.bed $K > admixture_K${K}.log

    # Extract cross-validation error
    CV_ERROR=$(grep -o "CV error.*" admixture_K${K}.log | awk '{print $3}')
    echo "K=$K: CV error = $CV_ERROR"
done
```

```
# Summarize results
echo "Admixture analysis completed. Cross-validation results:"
grep -h "CV error" admixture_K*.log
```

Section: WGS_analysis > FST_Calculations

File: Fst_analysis_1.14.2025.ipynb

```
# === Code Cell 1 ===
# Importing libraries
import os
import pandas as pd
import numpy as np
import re

# === End of Cell ===

# === Code Cell 2 ===
import pandas as pd
import os

def read_tab_delimited_files(directory_path, dict_name):
    """
    Reads all tab-delimited text files in the specified directory into pandas DataFrames
    and assigns them to the specified variable name in the global scope.

    Args:
         directory_path (str): Path to the directory containing the tab-delimited text
files.
        df_name (str): Name of the variable to store the resulting DataFrame dictionary.

    Returns:
        None
    """
    data_frames = {}

    try:
        # Iterate through files in the directory
        for filename in os.listdir(directory_path):
                # Check if the file has a .txt or .tsv extension (commonly used for
tab-delimited files)
            if filename.endswith(".fst"):
                file_path = os.path.join(directory_path, filename)

                # Read the file into a pandas DataFrame
                df = pd.read_csv(file_path, sep="\t")

                # Defining data frame name
                dfname = filename.split("_")[1] + "_" + filename.split("_")[2]

                # Store the DataFrame in the dictionary
                data_frames[dfname] = df

                print(f"Successfully read file: {filename}")
        # Assign the resulting dictionary to the given variable name in the global scope
        globals()[dict_name] = data_frames

    except Exception as e:
        print(f"An error occurred: {e}")

for    folders    in    ["Ethiopia_vs_India",    "Ethiopia_vs_Yemen",    "Yemen_vs_India",
"YemenNEthiopia_vs_India"]:
    # Example usage
    if __name__ == "__main__":
        # Specify the directory containing your tab-delimited text files
        directory = folders

        # Specify the name of the variable to store the DataFrame dictionary
        dictionary_name = folders

        # Read the files and assign to the specified variable name
        read_tab_delimited_files(directory, dictionary_name)

        # Access the DataFrame dictionary dynamically
        data_frames_dict = globals()[dictionary_name]

        # Display the first few rows of each DataFrame
        for file_name, df in data_frames_dict.items():
            print(f"\nData from file: {file_name}")
            print(df.head())


# === End of Cell ===

# === Code Cell 3 ===
# Importing the gene details

pcg_filePath = "Anstep_PCGs_1.12.2025.tsv"

PCG_det = pd.read_csv(pcg_filePath, sep='\t')

# === End of Cell ===

# === Code Cell 4 ===
def map_genes_to_fst_windows(fst_dicts, gene_df):
    """
     Maps gene symbols to Fst windows based on their positions and appends the list of
gene names
    to each Fst window in the Fst DataFrame.

    Args:
        fst_dicts (dict): Dictionary of DataFrames containing Fst windows.
         gene_df (pd.DataFrame): DataFrame with gene details, including columns 'Begin',
'End', and 'Symbol'.

    Returns:
          dict: Updated dictionary of DataFrames with a new column 'Genes' containing
lists of gene names.
```

```
    """
    # Ensure 'Begin', 'End', and 'Symbol' columns exist in gene_df
    required_columns = {'Begin', 'End', 'Symbol'}
    if not required_columns.issubset(gene_df.columns):
              raise  ValueError(f"The  gene  DataFrame  must  contain  the  columns:
{required_columns}")

    for fst_name, fst_df in fst_dicts.items():
        try:
            # Create a new column for genes in the Fst DataFrame
            fst_df['Genes'] = fst_df.apply(
                lambda row: gene_df[
                            (gene_df['Begin'] >= row['BIN_START']) & (gene_df['End'] <=
row['BIN_END']) & (gene_df['Accession'] == row['CHROM'])
                ]['Symbol'].tolist(), axis=1
            )
            print(f"Processed Fst DataFrame: {fst_name}")
        except Exception as e:
            print(f"An error occurred while processing {fst_name}: {e}")

    return fst_dicts

# Running the function on the data frames in dictionaries
for    dict    in    ["Ethiopia_vs_India",    "Ethiopia_vs_Yemen",    "Yemen_vs_India",
"YemenNEthiopia_vs_India"]:
    # Example usage
    if __name__ == "__main__":

        # Read the files and assign to the specified variable name
        globals()[dict] = map_genes_to_fst_windows(globals()[dict], PCG_det)

        # Access the DataFrame dictionary dynamically
        data_frames_dict = globals()[dict]

        # Display the first few rows of each DataFrame
        for file_name, df in data_frames_dict.items():
            print(f"\nData from file: {file_name}")
            print(df.head())


# === End of Cell ===

# === Code Cell 5 ===
def add_percent_category(data_frames):
    """
    Adds a column 'percent_category' to each DataFrame in the dictionary based on
    the percentiles (75%, 90%, 95%, 99%) of the 'WEIGHTED_FST' column.

    Args:
        data_frames (dict): Dictionary where values are pandas DataFrames.

    Returns:
        None
    """
    for name, df in data_frames.items():
        try:
            # Calculate percentiles
            percentiles = {
                75: df['WEIGHTED_FST'].quantile(0.75),
                90: df['WEIGHTED_FST'].quantile(0.90),
                95: df['WEIGHTED_FST'].quantile(0.95),
                99: df['WEIGHTED_FST'].quantile(0.99),
            }

            # Printing percentiles
            print(percentiles)

            # Define categorization function
            def categorize(value):
                if value >= percentiles[99]:
                    return 'above_99_pct'
                elif value >= percentiles[95]:
                    return 'bw_95N99_pct'
                elif value >= percentiles[90]:
                    return 'bw_90N95_pct'
                elif value >= percentiles[75]:
                    return 'bw_75N90_pct'
                else:
                    return 'below_75_pct'

            # Apply the categorization
            df['percent_category'] = df['WEIGHTED_FST'].apply(categorize)
            print(f"Added 'percent_category' to DataFrame: {name}")

        except Exception as e:
            print(f"An error occurred while processing DataFrame {name}: {e}")


# Applying the function to assign the percentile category to each value in
for    dict    in    ["Ethiopia_vs_India",    "Ethiopia_vs_Yemen",    "Yemen_vs_India",
"YemenNEthiopia_vs_India"]:

    # Access the DataFrame dictionary dynamically
    print(dict)
    data_frames_dict = globals()[dict]
    add_percent_category(data_frames_dict)

# === End of Cell ===

# === Code Cell 6 ===
def split_data_frames_by_category(data_frames):
    """
         Splits  DataFrames  in  a  dictionary  into  sub-dictionaries  based  on  the
'percent_category' column.

    Args:
        data_frames (dict): Dictionary where values are pandas DataFrames.
    Returns:
        dict: A dictionary of dictionaries, where keys are DataFrame names and values
              are dictionaries split by 'percent_category'.
    """
    split_dict = {}

    for name, df in data_frames.items():
        try:
            # Check if 'percent_category' column exists
            if 'percent_category' not in df.columns:
                    print(f"DataFrame {name} does not have 'percent_category' column.
Skipping.")
                continue

            # Create a sub-dictionary for this DataFrame
            category_dict = {}

            # Group by 'percent_category' and create a dictionary of DataFrames
            for category, group in df.groupby('percent_category'):
                category_dict[category] = group

            # Add the sub-dictionary to the main dictionary
            split_dict[name] = category_dict
            print(f"Split DataFrame {name} into categories.")

        except Exception as e:
            print(f"An error occurred while splitting DataFrame {name}: {e}")

    return split_dict


# Running the function on the data frames in dictionaries
for    dict    in    ["Ethiopia_vs_India",    "Ethiopia_vs_Yemen",    "Yemen_vs_India",
"YemenNEthiopia_vs_India"]:
    # Example usage
    if __name__ == "__main__":
        # Specify the name of the variable to store the DataFrame dictionary
        dictionary_name = dict + "_splitOnPct"

        # Read the files and assign to the specified variable name
        globals()[dictionary_name] = split_data_frames_by_category(globals()[dict])

        # Access the DataFrame dictionary dynamically
        data_frames_dict = globals()[dictionary_name]

        # Display the result structure
    for df_name, categories in data_frames_dict.items():
        print(f"\nDataFrame: {df_name}")
        for category, df in categories.items():
            print(f"Category: {category}, Number of Rows: {len(df)}")

# === End of Cell ===

# === Code Cell 7 ===
# Define the folder where text files will be saved
output_folder = "Fst_unique_genes_text_files"
os.makedirs(output_folder, exist_ok=True)

# Running the function on the data frames in dictionaries
for    dict    in    ["Ethiopia_vs_India_splitOnPct",    "Ethiopia_vs_Yemen_splitOnPct",
"Yemen_vs_India_splitOnPct", "YemenNEthiopia_vs_India_splitOnPct"]:

    # Access the DataFrame dictionary dynamically
    data_frames_dict = globals()[dict]

    # Loop over the dictionary of dictionaries
    for outer_key, inner_dict in data_frames_dict.items():
        for inner_key, df in inner_dict.items():
            # Extract all the unique genes from the 'Genes' column
            unique_genes = set()
            for gene_list in df['Genes']:
                unique_genes.update(gene_list)  # Combine all genes into the set

            # Create a meaningful and sanitized file name
            sanitized_outer_key = re.sub(r'[<>:"/\\|?*]', '_', outer_key)
            sanitized_inner_key = re.sub(r'[<>:"/\\|?*]', '_', inner_key)
                                                                          file_name      =
f"{dict}_{sanitized_outer_key}_{sanitized_inner_key}_unique_genes.txt".replace(" ", "_")
            file_path = os.path.join(output_folder, file_name)

            # Save the unique genes to the text file
            with open(file_path, 'w') as f:
                for gene in sorted(unique_genes):  # Sort genes alphabetically
                    f.write(f"{gene}\n")

            # Print a message to indicate progress
            print(f"Saved: {file_name}")

# === End of Cell ===
```

File: Fst_visualization.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=1

# Loading R module
module load R/4.4.0

# Navigating to the folder with R script and data
cd /data/gunarathnai/Ans_GA/all_variants/YmEtInd_all_variants/Fst_visualization

# Running R script
R --vanilla --file=Ans_GA_Fst_Vis.R
```

File: Plot_Fst.py

```
import pandas as pd
import matplotlib.pyplot as plt

# Define the input file path directly in the script
file_path                                                                                =
'YmEthSampleNames.txtIndianSampleNames.txtYmEthInd_allChrs_BA_Phased.vcf.gz.vcf.gz_Fst.w
indowed.weir.fst'  # Replace with the path to your Fst file

# Load the Fst data file
try:
    fst_data = pd.read_csv(file_path, sep='\t')
except Exception as e:
    print(f"Error loading file: {e}")
    exit(1)

# Ensure the column names match the data format
chromosome_col = 'CHROM'
fst_col = 'WEIGHTED_FST'
start_col = 'BIN_START'

# Filter out rows with missing Fst values
fst_data = fst_data.dropna(subset=[fst_col])

# Get unique chromosomes
chromosomes = fst_data[chromosome_col].unique()

# Determine the chromosome lengths
chromosome_lengths = {
      chrom:  fst_data[fst_data[chromosome_col]  ==  chrom][start_col].max()  for  chrom  in
chromosomes
}

# Scale chromosome lengths to avoid oversized figures
max_scaled_length = 50  # Set the maximum scaled width for the largest chromosome
scaling_factor = max_scaled_length / max(chromosome_lengths.values())
scaled_lengths   =   {chrom:   length   *   scaling_factor   for   chrom,   length   in
chromosome_lengths.items()}

# Create a multi-panel plot with scaled panel widths
fig, axes = plt.subplots(1, len(chromosomes),
                         figsize=(max_scaled_length + 4, 6),  # Limit total figure size
                           gridspec_kw={'width_ratios': [scaled_lengths[chrom] for chrom
in chromosomes]})

# If there's only one chromosome, wrap the single axis into a list for consistency
if len(chromosomes) == 1:
    axes = [axes]

# Plot each chromosome in its own panel
for ax, chrom in zip(axes, chromosomes):
    chrom_data = fst_data[fst_data[chromosome_col] == chrom]
    ax.scatter(chrom_data[start_col], chrom_data[fst_col], alpha=0.7, s=10)
    ax.set_title(f"Chromosome: {chrom}", fontsize=12)
    ax.set_xlim(0, chromosome_lengths[chrom])
    ax.set_xlabel("Genomic Position (bp)", fontsize=10)
    ax.set_ylabel("Weighted Fst", fontsize=10)
    ax.grid(axis='y', linestyle='--', alpha=0.7)

# Adjust layout
plt.tight_layout()

# Save the plot as a PNG file
output_file = "YemenNEthiopiaVsIndia_fst_scatter_plot_single_row.png"
plt.savefig(output_file, dpi=300)
plt.show()

print(f"Scatter plot saved to {output_file}.")
```

File: chr2_Fst_calc.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10

# locating the directory with data
cd /data/gunarathnai/Ans_GA/all_variants/YmEtInd_all_variants/chr_2

# setting background for conda environment activation
eval "$(conda shell.bash hook)"

# activating 'newbio' environment with conda
conda activate newbio

vcftools --gzvcf biallelic_phased_YmEtInd_chr2.vcf.gz.vcf.gz --weir-fst-pop Indian_sample_IDs.txt --weir-fst-pop Yemen_sample_IDs.txt --out chr2_YmVsInd_Fst

vcftools --gzvcf biallelic_phased_YmEtInd_chr2.vcf.gz.vcf.gz --weir-fst-pop EthiopianNSomaliLand_sample_IDs.txt --weir-fst-pop Yemen_sample_IDs.txt --out chr2_YmVsEtSml_Fst

vcftools --gzvcf biallelic_phased_YmEtInd_chr2.vcf.gz.vcf.gz --weir-fst-pop EthiopianNSomaliLand_sample_IDs.txt --weir-fst-pop Indian_sample_IDs.txt --out chr2_IndVsEtSml_Fst
```

File: chr3_Fst_calc.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10

# locating the directory with data
cd /data/gunarathnai/Ans_GA/all_variants/YmEtInd_all_variants/chr_3
```

```
# setting background for conda environment activation
eval "$(conda shell.bash hook)"

# activating 'newbio' environment with conda
conda activate newbio

vcftools --gzvcf biallelic_phased_YmEtInd_chr3.vcf.gz.vcf.gz --weir-fst-pop Indian_sample_IDs.txt --weir-fst-pop Yemen_sample_IDs.txt --out chr3_YmVsInd_Fst

vcftools --gzvcf biallelic_phased_YmEtInd_chr3.vcf.gz.vcf.gz --weir-fst-pop EthiopianNSomaliLand_sample_IDs.txt --weir-fst-pop Yemen_sample_IDs.txt --out chr3_YmVsEtSml_Fst

vcftools --gzvcf biallelic_phased_YmEtInd_chr3.vcf.gz.vcf.gz --weir-fst-pop EthiopianNSomaliLand_sample_IDs.txt --weir-fst-pop Indian_sample_IDs.txt --out chr3_IndVsEtSml_Fst
```

Section: WGS_analysis > GATK_scripts

File: BiAllelic_filtering_n_Phasing.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=30
#PBS -k doe

# setting working directory
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/IndYmEthSm
l_individual_gVCFs/filtered_YmEthIndSml_VCFs


for VCF in `ls YmEthIndSml_NC_*_fltpass.vcf.gz`

do

# getting sample name
chrID=$(echo "$VCF" | cut -d'_' -f3)

# filtering biallelic variants
#bcftools view -Oz -m 2 -M 2 ${VCF} -o YmEthIndSml_NC_${chrID}_BAFlt.vcf.gz --threads 30

# indexing the input file
bcftools index YmEthIndSml_NC_${chrID}_BAFlt.vcf.gz

# Pahsing biallelic VCF file using beagle 5.4
java -Xmx200g -jar /home/gunarathnai/bin/beagle/beagle_V5.4.jar \
 nthreads=30 burnin=5 iterations=20 \
 gt=YmEthIndSml_NC_${chrID}_BAFlt.vcf.gz \
 out=../biallelic_phased_VCFs/YmEthIndSml_NC_${chrID}_BA_Phased.vcf.gz

done
```

File: Combining_gVCFs.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10

# setting working directory
cd /data/gunarathnai/

# Set paths - Replace with the path to your Apptainer image
CONTAINER_PATH="/data/gunarathnai/Apptainer_containers/gatk4TCLab.sif"

for chr in `cat /data/gunarathnai/Ans_GA/gatk_variants/chrIDs.txt`

do
```

```
apptainer exec "$CONTAINER_PATH" gatk --java-options "-Xms8G -Xmx8G" GenomicsDBImport \
                                                              --genomicsdb-workspace-path
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/YmEthInd_g
db_workspace/${chr}_gdb \
  --intervals ${chr} \
                                                                                --tmp-dir
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/gdb_tmp \
  -R /data/gunarathnai/Ans_GA/gatk_variants/ref_genome/UCI_ANSTEP_V1.fna \
                                                                        --sample-name-map
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/YmEthInd.s
ample_map \
  --reader-threads 5

done
```

File: Joint_genotyping.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10

# setting working directory
cd /data/gunarathnai/

# Set paths - Replace with the path to your Apptainer image
CONTAINER_PATH="/data/gunarathnai/Apptainer_containers/gatk4TCLab.sif"

for chr in `cat /data/gunarathnai/Ans_GA/gatk_variants/chrIDs.txt`

do

apptainer exec "$CONTAINER_PATH" gatk --java-options "-Xmx4g" GenotypeGVCFs \
 -R /data/gunarathnai/Ans_GA/gatk_variants/ref_genome/UCI_ANSTEP_V1.fna \
 -V
gendb://Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/YmEthInd_gdb_workspa
ce/${chr}_gdb \
 -O
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/genotyped_
VCFs/YmEthInd_${chr}_genotyped.vcf.gz \
 -G StandardAnnotation -G AS_StandardAnnotation \
 --tmp-dir
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/gdb_tmp

done
```

File: Run_MD.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=35
```

```
# setting working directory
cd /data/gunarathnai/Ans_GA/Ethiopia_B2/sorted_bams

# Set paths
CONTAINER_PATH="/data/gunarathnai/Apptainer_containers/gatk4TCLab.sif"  #  Replace  with
the path to your Apptainer image

# Iterating over file names
for sbamfile in `ls *.sorted.bam`
do

# Extracting sample name from the input file name
sID=$(echo $sbamfile | cut -d "." -f 1)

# Adding read group data since it was not included in the bam files
apptainer    exec    "$CONTAINER_PATH"    gatk    AddOrReplaceReadGroups    I=${sbamfile}
O=${sID}.sorted.rg.bam RGID=1 RGLB=lib1 RGPL=illumina RGPU=unit1 RGSM=$sbamfile

# Running MarkDuplicate on bam files with read group data
apptainer  exec  "$CONTAINER_PATH"  gatk  MarkDuplicatesSpark  -I  ${sID}.sorted.rg.bam  -O
${sID}_dedup.bam -M ${sID}_metrics.txt --spark-master local[35] #--remove-all-duplicates

done
```

File: base_recalibrator.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=5

# setting working directory
cd /data/gunarathnai/

# Set paths - Replace with the path to your Apptainer image
CONTAINER_PATH="/data/gunarathnai/Apptainer_containers/gatk4TCLab.sif"

# Creating GATK compatible index for the reference genome
#apptainer     exec     "$CONTAINER_PATH"     gatk     CreateSequenceDictionary     -R
/data/gunarathnai/Ans_GA/WG_bam/sorted_bams/MD_bams/ref_genome/UCI_ANSTEP_V1.fna \
#-O /data/gunarathnai/Ans_GA/WG_bam/sorted_bams/MD_bams/ref_genome/UCI_ANSTEP_V1.dict


# Iterating over file names
for sbamfile in `ls /data/gunarathnai/Ans_GA/Ethiopia_B2/dedup_bams/*_dedup.bam`
do

apptainer    exec    "$CONTAINER_PATH"    gatk    --java-options    "-Xms4G    -Xmx4G
-XX:ParallelGCThreads=2" BaseRecalibrator \
 -I $sbamfile \
 -R /data/gunarathnai/Ans_GA/WG_bam/sorted_bams/MD_bams/ref_genome/UCI_ANSTEP_V1.fna \
 --known-sites
/data/gunarathnai/Ans_GA/WG_bam/sorted_bams/MD_bams/known_sites/Astp_IndCh.calls.norm.fl
t-indels.vcf.gz \
 -O ${sbamfile}_bqsr.report
done
```

File: bqsr_apply.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=8

# setting working directory
cd /data/gunarathnai/

# Set paths - Replace with the path to your Apptainer image
CONTAINER_PATH="/data/gunarathnai/Apptainer_containers/gatk4TCLab.sif"

# Iterating over file names
for sID in `cat /data/gunarathnai/Ans_GA/Ethiopia_B2/dedup_bams/samplenames.txt`
do

apptainer     exec     "$CONTAINER_PATH"     gatk     --java-options     "-Xms2G     -Xmx2G
-XX:ParallelGCThreads=2" ApplyBQSR \
 -I /data/gunarathnai/Ans_GA/Ethiopia_B2/dedup_bams/${sID}_dedup.bam \
 -R /data/gunarathnai/Ans_GA/WG_bam/sorted_bams/MD_bams/ref_genome/UCI_ANSTEP_V1.fna \
 --bqsr-recal-file
/data/gunarathnai/Ans_GA/Ethiopia_B2/dedup_bams/${sID}_dedup.bam_bqsr.report \
 -O /data/gunarathnai/Ans_GA/Ethiopia_B2/dedup_bams/${sID}_bqsr.bam

done
```

File: filtering_out.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10

# setting working directory
cd /data/gunarathnai/

# Set paths - Replace with the path to your Apptainer image
CONTAINER_PATH="/data/gunarathnai/Apptainer_containers/gatk4TCLab.sif"

for chr in `cat /data/gunarathnai/Ans_GA/gatk_variants/chrIDs.txt`

do

# Define paths for input VCF and reference genome
INPUT_VCF="/data/gunarathnai/Ans_GA/gatk_variants/genotyped_VCFs/EnI_${chr}_flt.vcf.gz"
# Replace with your VCF file
REFERENCE_GENOME="/data/gunarathnai/Ans_GA/WG_bam/sorted_bams/MD_bams/ref_genome/UCI_ANS
```

```
TEP_V1.fna"  # Replace with your reference genome file
OUTPUT_VCF="/data/gunarathnai/Ans_GA/gatk_variants/genotyped_VCFs/EnI_${chr}_fltpass.vcf
.gz"

# Run VariantFiltration with filters optimized for RNA-seq data
apptainer exec "$CONTAINER_PATH" gatk SelectVariants \
   -R $REFERENCE_GENOME \
   -V $INPUT_VCF \
   --exclude-filtered \
   -O $OUTPUT_VCF \

done
```

File: haplotypeCaller.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=36

# setting working directory
cd /data/gunarathnai/

# Set paths - Replace with the path to your Apptainer image
CONTAINER_PATH="/data/gunarathnai/Apptainer_containers/gatk4TCLab.sif"

# Iterating over file names
for sID in `cat /data/gunarathnai/Ans_GA/WG_bam/sorted_bams/BQSR_bams/sample_names.txt`
do

apptainer    exec    "$CONTAINER_PATH"    gatk    --java-options    "-Xms120G    -Xmx120G
-XX:ParallelGCThreads=15" HaplotypeCaller \
   -R /data/gunarathnai/Ans_GA/WG_bam/sorted_bams/MD_bams/ref_genome/UCI_ANSTEP_V1.fna \
   -I /data/gunarathnai/Ans_GA/WG_bam/sorted_bams/BQSR_bams/${sID}_bqsr.bam \
   -O /data/gunarathnai/Ans_GA/WG_bam/sorted_bams/BQSR_bams/${sID}.g.vcf \
   -ERC GVCF \
   --native-pair-hmm-threads 20

done
```

File: variant_filtering.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10

# setting working directory
cd /data/gunarathnai/

# Set paths - Replace with the path to your Apptainer image
CONTAINER_PATH="/data/gunarathnai/Apptainer_containers/gatk4TCLab.sif"

for chr in `cat /data/gunarathnai/Ans_GA/gatk_variants/chrIDs.txt`
```

```
do

# Define paths for input VCF and reference genome
INPUT_VCF="/data/gunarathnai/Ans_GA/gatk_variants/genotyped_VCFs/EnI_${chr}_genotyped.vc
f.gz"  # Replace with your VCF file
REFERENCE_GENOME="/data/gunarathnai/Ans_GA/WG_bam/sorted_bams/MD_bams/ref_genome/UCI_ANS
TEP_V1.fna"  # Replace with your reference genome file
OUTPUT_VCF="/data/gunarathnai/Ans_GA/gatk_variants/genotyped_VCFs/EnI_${chr}_flt.vcf.gz"

# Run VariantFiltration with filters optimized for RNA-seq data
apptainer exec "$CONTAINER_PATH" gatk VariantFiltration \
   -R $REFERENCE_GENOME \
   -V $INPUT_VCF \
   -O $OUTPUT_VCF \
   --filter-name "LowQual" --filter-expression "QUAL < 30.0" \
   --filter-name "LowQD" --filter-expression "QD < 2.0" \
   --filter-name "LowDepth" --filter-expression "DP < 10" \
   --filter-name "StrandBias" --filter-expression "FS > 60.0" \
   --filter-name "ReadPosRank" --filter-expression "ReadPosRankSum < -8.0" \
   --filter-name "MappingQuality" --filter-expression "MQ < 40.0" \
   --filter-name "SOR" --filter-expression "SOR > 3.0"

done
```

Section: WGS_analysis > LD_n_PCA

File: LD_decay_calc.py

```
#!/usr/bin/env python2

# read in gzip file

import gzip
import sys, argparse
import glob, re
import numpy as np

def getOptionValue(option): # needs sys
  if option in sys.argv:
    optionPos = sys.argv.index(option)
    optionValue = sys.argv[optionPos + 1]
    return optionValue
  else:
    print >> sys.stderr, "\nWarning, option", option, "not_specified.\n"

if "-i" in sys.argv:
  fileName = getOptionValue("-i")
else:
  print("\nplease specify input file name using -i <file_name> \n")
  sys.exit()

if "-o" in sys.argv:
  prefix = getOptionValue("-o")
else:
  print("\nplease specify output prefix using -o \n")
  sys.exit()

# prefix = "test"
# fileName = "./test.ld.gz"

f = gzip.open(fileName, 'r')

f.readline()

counter = 0
chr_distance_ld = {}

for line in f:

 line = line.strip().split()
 distance = abs(int(line[1])-int(line[4]))
 chr = line[0]
 if chr in chr_distance_ld:
  if distance in chr_distance_ld[chr]:
   chr_distance_ld[chr][distance].append(float(line[6]))
  else:
   chr_distance_ld[chr][distance] = [float(line[6])]
 else:
  chr_distance_ld[chr] = {}
  chr_distance_ld[chr][distance] = [float(line[6])]
  #print (chr_distance_ld)
 counter += 1
 if (counter % 100000000) == 0:
  print("Read %d entires." % counter)
  sys.stdout.flush()


print("Calculating average distances.")
sys.stdout.flush()

decay_output = open(prefix + ".ld_decay", "w")
#write header
decay_output.write("chr\tdistance\tavg_R2\tstd\n")
#decay_output.write("distance\tR2\n")
decay_output_bins = open(prefix + ".ld_decay_bins", "w")
#write header
decay_output_bins.write("chr\tdistance\tavg_R2\tstd\n")
#decay_output.write("distance\tR2\n")

#distance_avg_ld = {}
for chr in chr_distance_ld:
 distance_bin = {}
 for distance in sorted(chr_distance_ld[chr]):
  lds = np.array(chr_distance_ld[chr][distance])
  mean = np.mean(lds)
  std = np.std(lds)
  #sum(chr_distance_ld[chr][distance])/len(chr_distance_ld[chr][distance]
  decay_output.write("%s\t%d\t%g\t%g\n" % (chr, distance, mean, std))
  #round to closest 1000, and add to new structure No, round to 500 (all from 0-1000).
  bin_thousand = int(round(distance-500, -3)) + 500
  if bin_thousand in distance_bin:
   distance_bin[bin_thousand] = np.concatenate([distance_bin[bin_thousand], lds])
  else:
   distance_bin[bin_thousand] = lds
 for bin in sorted(distance_bin):
  mean = np.mean(distance_bin[bin])
  std = np.std(distance_bin[bin])
  decay_output_bins.write("%s\t%d\t%g\t%g\n" % (chr, bin, mean, std))

decay_output_bins.close()
decay_output.close()

f.close()
```

File: YmEthInd_allChrs_LD.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=20
```

```
# locating the directory with data
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/biallelic_
phased_VCFs

# registering VCF file name
VCF=YmEthOnly_allChrs_BA_Phased.vcf.gz

plink --vcf $VCF --double-id --allow-extra-chr \
--set-missing-var-ids @:# \
--maf 0.01 --geno 0.1 --mind 0.5 \
--thin 0.1 -r2 gz --ld-window 100 --ld-window-kb 1000 \
--ld-window-r2 0 \
--make-bed --out YmEthOnly_allChrs_LD
```

File: YmEthInd_sepChrs_LD.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=20

# locating the directory with data
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/biallelic_
phased_VCFs

for chr in `cat chrIDs.txt`

do

# registering VCF file name
VCF=YmEthInd_${chr}_BA_Phased.vcf.gz.vcf.gz

plink --vcf $VCF --double-id --allow-extra-chr \
--set-missing-var-ids @:# \
--maf 0.01 --geno 0.1 --mind 0.5 --chr ${chr} \
--thin 0.1 -r2 gz --ld-window 100 --ld-window-kb 1000 \
--ld-window-r2 0 \
--make-bed --out YmEthInd_${chr}_LD
done
```

File: autosomal_YmEthSml_plink_PCA.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=20
#PBS -k doe

# locating the directory with data
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/IndYmEthSm
```

```
l_individual_gVCFs/biallelic_phased_VCFs

# printing hostname on Kodiak
hostname

# linkage pruning
plink --vcf autosomal_YmEthSml_BA_Phased.vcf \
--double-id \
--allow-extra-chr \
--list-duplicate-vars \
--threads 20 \
--set-missing-var-ids @:#_\$1_\$2 \
--indep-pairwise 50 10 0.1 \
--out autosomal_YmEthSml_BAP_LD

# Performing PCA
# The "var-wts" flag will produce the loadings of the genes for each PC
plink --vcf autosomal_YmEthSml_BA_Phased.vcf \
--double-id \
--allow-extra-chr \
--set-missing-var-ids @:#_\$1_\$2 \
--extract autosomal_YmEthSml_BAP_LD.prune.in \
--list-duplicate-vars --threads 20 \
--make-bed \
--pca var-wts \
--out autosomal_YmEthSml_BAP_PCA
```

Section: WGS_analysis > Nucleotide_Diversity

File: nuc_div_per_site_WGS.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=5
#PBS -k doe

# locating the directory with data
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/IndYmEthSm
l_individual_gVCFs/biallelic_phased_VCFs

# setting background for conda environment activation
eval "$(conda shell.bash hook)"

# activating 'newbio' environment with conda
conda activate newbio

# Define your input and output file names
VCF_FILE="autosomal_YmEthIndSml_BA_Phased.vcf"
OUTPUT_FILE="nuc_div_results/autosomal_YmEthIndSml_BA_Phased.nucleotide_diversity.per_si
te.pi"

# Run VCFtools to calculate nucleotide diversity per SNP
vcftools \
    --vcf "$VCF_FILE" \
    --site-pi \
    --out nucleotide_diversity_per_site_output

# Rename the output file for convenience
mv nucleotide_diversity_per_site_output.sites.pi "$OUTPUT_FILE"

echo "Nucleotide diversity calculation per site completed. Results saved to
$OUTPUT_FILE"
```

File: nuc_div_window_WGS.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=5
#Pbs -k doe

# locating the directory with data
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/IndYmEthSm
l_individual_gVCFs/biallelic_phased_VCFs

# setting background for conda environment activation
```

```
eval "$(conda shell.bash hook)"

# activating 'newbio' environment with conda
conda activate newbio

# Define your input and output file names
VCF_FILE="autosomal_YmEthSml_BA_Phased.vcf"
OUTPUT_FILE="nuc_div_results/autosomal_YmEthSml_BA_Phased.nucleotide_diversity.windowed.
pi"

# Run VCFtools to calculate nucleotide diversity
vcftools \
    --vcf "$VCF_FILE" \
    --window-pi 100000 \
    --window-pi-step 50000 \
    --out nucleotide_diversity_output

# Rename the output file for convenience
mv nucleotide_diversity_output.windowed.pi "$OUTPUT_FILE"

echo "Nucleotide diversity calculation completed. Results saved to $OUTPUT_FILE"
```

Section: WGS_analysis > Phylogenetic_tree

File: generateNJTree.py

```
import allel
from Bio import Phylo
from Bio.Phylo.TreeConstruction import DistanceTreeConstructor, DistanceMatrix
import numpy as np
from ete3 import Tree #, TreeStyle

# Step 1: Load the VCF file and extract sample names and genotype data
vcf_file = 'YmEthInd_allChrs_BA_Phased.vcf.gz.vcf'  # Replace with your VCF file path
callset = allel.read_vcf(vcf_file)
sample_names = callset['samples']
genotypes = allel.GenotypeArray(callset['calldata/GT'])

# Step 2: Convert genotypes to haplotypes (0, 1, or 2 to represent allele counts)
haplotypes = genotypes.to_n_alt()  # Convert genotypes to allele counts

# Step 3: Calculate the pairwise distances and store in lower triangular format
def calculate_lower_triangle_distance_matrix(haplotypes):
    n_samples = haplotypes.shape[1]
    dist_matrix = []

    for i in range(n_samples):
        row = []
        for j in range(i):  # Only calculate for lower triangle
            dist = np.sum(haplotypes[:, i] != haplotypes[:, j])  # Count allele
differences
            row.append(dist)
        row.append(0)  # Diagonal elements (distance to self)
        dist_matrix.append(row)

    return dist_matrix

dist_matrix = calculate_lower_triangle_distance_matrix(haplotypes)

# Step 4: Convert the distance matrix to a format compatible with Bio.Phylo
labels = list(sample_names)  # Original sample names from VCF
matrix = DistanceMatrix(names=labels, matrix=dist_matrix)

# Step 5: Construct the tree using the distance matrix
constructor = DistanceTreeConstructor()
tree = constructor.nj(matrix)  # Using Neighbor-Joining (NJ) method

# Step 6: Save the tree to a Newick file using Bio.Phylo
newick_file = "temp_tree.nw"
Phylo.write(tree, newick_file, "newick")

# Step 7: Load the Newick file with ete3
ete_tree = Tree(newick_file, format=1)
```

```
# Step 8: Rename the leaves in the ete3 tree to match original sample names
# Create a dictionary mapping new leaf names to original sample names
leaf_names = ete_tree.get_leaf_names()
name_mapping = {leaf: sample for leaf, sample in zip(leaf_names, sample_names)}

# Rename leaves in the ete3 tree
for leaf in ete_tree:
    if leaf.name in name_mapping:
        leaf.name = name_mapping[leaf.name]

# Step 9: Save the renamed tree
ete_tree.write(format=1, outfile="phylogenetic_tree_renamed.nw")

# Optional: Save the tree as an image
ete_tree.render("phylogenetic_tree_renamed.png", w=800, units="px")
```

File: raxMLHPC_tree.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10
#PBS -k doe

# locating the directory with data
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/biallelic_
phased_VCFs/autosomal_fasta_alignments/conactenated_fasta_file

# setting background for conda environment activation
eval "$(conda shell.bash hook)"

# activating 'newbio' environment with conda
conda activate newbio

# running raxmlHPC to generate the tree files
raxmlHPC-PTHREADS  -T  10  -f  a  -x  95538  -p  95538  -N  1000  -m  GTRGAMMA  -k  -n
all_autosomal_BAPhased.tre -s all_autosomal_genomes.fasta
```

Section: WGS_analysis > RoH_n_Selection

File: Garuds_HStat_analysis.ipynb

```
# === Code Cell 1 ===
# Importing necessary libraries

import numpy as np
import pandas as pd
import os
import re

# === End of Cell ===

# === Code Cell 2 ===
### Reading the H statistics data

# Define the folder path where the CSV files are stored
folder_path                          =                          "C:/Users/aaisu/Box/Carter
Lab/Projects/Anstep_GA_Yemen/Selection_stats_12.15.2024/Garuds_Hstat/Hstat_data"

# Create a dictionary to store DataFrames, using file names (without extension) as keys
dataframes = {}

# Loop through all files in the folder
for file_name in os.listdir(folder_path):
    # Check if the file is a CSV
    if file_name.endswith(".csv"):
        # Construct the full file path
        file_path = os.path.join(folder_path, file_name)

        # Read the CSV file into a DataFrame
        df = pd.read_csv(file_path)

        # Store the DataFrame in the dictionary
        # Use the file name without the extension as the key
        key = os.path.splitext(file_name)[0]
        dataframes[key] = df

# Example: Accessing a specific DataFrame by its key
for key, df in dataframes.items():
    print(f"DataFrame from file {key}:")
    print(df.head())  # Display the first 5 rows
    print()


### Reading variant positions
# Define the folder path where the CSV files are stored
folder_path                          =                          "C:/Users/aaisu/Box/Carter
Lab/Projects/Anstep_GA_Yemen/Selection_stats_12.15.2024/Garuds_Hstat/"
```

```
# Create a dictionary to store DataFrames, using file names (without extension) as keys
variant_positions = {}

# Column names to be assigned
column_names = ['CHROM', 'POS', 'REF', 'ALT']

# Loop through all files in the folder
for file_name in os.listdir(folder_path):
    # Check if the file is a CSV
    if file_name.endswith(".txt"):
        # Construct the full file path
        file_path = os.path.join(folder_path, file_name)

        # Read the CSV file into a DataFrame
        df = pd.read_csv(file_path, sep=' ', header=None, names=column_names)

        # Store the DataFrame in the dictionary
        # Use the file name without the extension as the key
        key = os.path.splitext(file_name)[0]
        variant_positions[key] = df

# Example: Accessing a specific DataFrame by its key
for key, df in variant_positions.items():
    print(f"DataFrame from file {key}:")
    print(df.head())  # Display the first 5 rows
    print()

# === End of Cell ===

# === Code Cell 3 ===
# Assuming `variant_positions` is the dictionary containing your original DataFrames
# Initialize a dictionary to store the new DataFrames
windowed_data = {}

# Define window width and step size
window_width = 1000
step_size = 500

# Iterate over each DataFrame in the dictionary
for key, df in variant_positions.items():
    # Initialize a list to store start and end values for each window
    window_summary = []

    # Slide the window through the DataFrame
    for start in range(0, len(df), step_size):
        end = start + window_width
        # Ensure the end index does not exceed the DataFrame length
        if end > len(df):
            break

        # Get the start and end CHROM values of the current window
        start_chrom = df.iloc[start]['POS']
        end_chrom = df.iloc[end - 1]['POS']
        # Append the results to the list
        window_summary.append({'Start_POS': start_chrom, 'End_POS': end_chrom})

    # Convert the summary to a DataFrame
    windowed_data[key] = pd.DataFrame(window_summary)

# Example: Accessing a specific DataFrame by its key
for key, df in windowed_data.items():
    print(f"Windowed DataFrame for {key}:")
    print(df.shape)  # Display the the number of rows columns
    print(df.head())  # Display the first 5 rows
    print()


# === End of Cell ===

# === Code Cell 4 ===
print(dataframes.keys())
print(windowed_data.keys())

hStat_dfnames              =              pd.Series(['garuds_h_statistics_Ethiopia_chr2',
'garuds_h_statistics_Ethiopia_chr3',                 'garuds_h_statistics_Ethiopia_chrx',
'garuds_h_statistics_India_chr2',                       'garuds_h_statistics_India_chr3',
'garuds_h_statistics_India_chrx',                       'garuds_h_statistics_Yemen_chr2',
'garuds_h_statistics_Yemen_chr3', 'garuds_h_statistics_Yemen_chrx'], name="hStat_dfs")
varPos_dfnames              =              pd.Series(['YmEthInd_NC_050202.1_BAFlt_Pos',
'YmEthInd_NC_050203.1_BAFlt_Pos',      'YmEthInd_NC_050201.1_BAFlt_Pos']      *      3,
name="varPos_dfs")

hStat_N_varPos = pd.concat([hStat_dfnames, varPos_dfnames], axis=1)
hStat_N_varPos.head

# Initialize a dictionary to store the joined DataFrames
joined_data = {}

# Iterate over each row in the mapping DataFrame
for index, row in hStat_N_varPos.iterrows():
    # Extract the names of the DataFrames from the current row
    hStat_key = row['hStat_dfs']
    varPos_key = row['varPos_dfs']

    # Retrieve the corresponding DataFrames from the dictionaries
    if hStat_key in dataframes and varPos_key in windowed_data:
        df_hStat = dataframes[hStat_key]
        df_varPos = windowed_data[varPos_key]

        # Perform the join operation (e.g., inner join)
        joined_df = pd.concat([df_hStat, df_varPos], axis=1)

        # Store the joined DataFrame in the dictionary
        joined_data[f'{hStat_key}_joined_{varPos_key}'] = joined_df

# Example: Accessing one of the joined DataFrames
for key, df in joined_data.items():
    print(f"Joined DataFrame for {key}:")
    print(df.shape)
    print(df.head())  # Display the first 5 rows
    print(df.tail())  # Display the last 5 rows
    print()

# === End of Cell ===

# === Code Cell 5 ===
# Function to calculate percentiles and categorize
def categorize_by_percentile(df, column_name, percentiles):
    thresholds = df[column_name].quantile(percentiles).to_dict()
    # Create a new column for the category
    category_col = f"{column_name}_category"
    df[category_col] = df[column_name].apply(
        lambda x: (
            "? 75%" if x <= thresholds[0.75] else
            "75-90%" if x <= thresholds[0.90] else
            "90-95%" if x <= thresholds[0.95] else
            "95-99%" if x <= thresholds[0.99] else
            "> 99%"
        )
    )
    print(thresholds)
    return df

# Iterate over each DataFrame in the dictionary and categorize
percentiles = [0.75, 0.90, 0.95, 0.99]
for key, df in joined_data.items():
    print(key)
    df = categorize_by_percentile(df, "H1", percentiles)
    df = categorize_by_percentile(df, "H12", percentiles)
    joined_data[key] = df

# Example: Display the modified DataFrame
for key, df in joined_data.items():
    print(f"DataFrame '{key}':")
    print(df)
    print()

# === End of Cell ===

# === Code Cell 6 ===
# Initialize dictionaries to store category counts
H1_counts = {}
H12_counts = {}

# Process each DataFrame in the dictionary
for key, df in joined_data.items():
    # Count occurrences of each category in H1_category
    H1_counts[key] = df['H1_category'].value_counts().to_dict()

    # Count occurrences of each category in H12_category
    H12_counts[key] = df['H12_category'].value_counts().to_dict()
```

```
# Convert the results into DataFrames
H1_counts_df = pd.DataFrame(H1_counts).fillna(0).astype(int)
H12_counts_df = pd.DataFrame(H12_counts).fillna(0).astype(int)

# Example: Display the results
print("H1 Category Counts:")
print(H1_counts_df)
print("\nH12 Category Counts:")
print(H12_counts_df)

# You can save these DataFrames to files if needed
H1_counts_df.to_csv("H1_category_counts.csv")
H12_counts_df.to_csv("H12_category_counts.csv")

# === End of Cell ===

# === Code Cell 7 ===
# Reading TSV file
pcg_filePath = "Anstep_PCGs_1.12.2025.tsv"

PCG_det = pd.read_csv(pcg_filePath, sep='\t')

# === End of Cell ===

# === Code Cell 8 ===
# Creating a new dictionary to hold the output
Hstat_W_genes = {}

# Process each DataFrame in the dictionary
for key, df in joined_data.items():
    chrID = key.split("_")[7] + "_" + key.split("_")[8]

    # Filter PCG_det for the relevant chrID
    filtered_PCG_det = PCG_det[PCG_det['Accession'] == chrID]

    # Create a DataFrame to represent overlaps
    overlaps = (
        pd.merge(
            df[['Start_POS', 'End_POS']].reset_index(),
            filtered_PCG_det[['Begin', 'End', 'Symbol']],
            how='cross'
        )
    )

    # Find where windows and genes overlap
    overlaps = overlaps[
              ((overlaps['Start_POS']  <=  overlaps['End'])  &  (overlaps['End']  <=
overlaps['End_POS'])) |
            ((overlaps['Start_POS']  <=  overlaps['Begin'])  &  (overlaps['Begin']  <=
overlaps['End_POS']))
    ]

    # Group overlapping genes by window index
    gene_groups = (
        overlaps.groupby('index')['Symbol']
        .apply(list)
        .reindex(df.index, fill_value=[])
    )

    # Add the grouped genes as a new column
    df['Genes'] = gene_groups

    # Store the updated DataFrame in the output dictionary
    Hstat_W_genes[key] = df


# === End of Cell ===

# === Code Cell 9 ===
# Define the folder to save the files
output_folder = "Hstat_windows_with_genes"
os.makedirs(output_folder, exist_ok=True)

# Save each DataFrame as a CSV file
for key, df in Hstat_W_genes.items():
    file_path = os.path.join(output_folder, f"{key}.csv")
    df.to_csv(file_path, index=False)  # `index=False` to exclude the index

# === End of Cell ===

# === Code Cell 10 ===
# Define the folder where CSV files will be saved
output_folder = "Hstat_windows_with_genes"
os.makedirs(output_folder, exist_ok=True)


# Initialize a new dictionary of dictionaries to hold the subsets
by_H1_Category_Hstat_W_genes = {}

# Loop over each DataFrame in the original dictionary
for key, df in Hstat_W_genes.items():
    # Get the unique categories in the H1_category column
    categories = df['H1_category'].unique()

    # Create a dictionary to hold subsets for the current DataFrame
    by_H1_Category_Hstat_W_genes[key] = {}

    # Subset the DataFrame for each category
    for category in categories:
        # filtering the category
        subset_df = df[df['H1_category'] == category]

        # Replace invalid characters in the category name
              sanitized_category  =  re.sub(r'[<>:"/\\|?*]',  '_',  category.replace('  ',
'_').replace('%', 'pct'))

        # Save the subset DataFrame to a CSV file
        file_name = f"{key}_H1_category_{sanitized_category}.csv"
        file_path = os.path.join(output_folder, file_name)
        subset_df.to_csv(file_path, index=False)

        # Saving to the dictionary
        by_H1_Category_Hstat_W_genes[key][category] = subset_df

# Example: Accessing the subsets
for key, category_dict in by_H1_Category_Hstat_W_genes.items():
    print(f"DataFrame: {key}")
    for category, subset_df in category_dict.items():
        print(f"Category: {category}")
        print(subset_df.head())  # Print the first few rows of the subset


# === End of Cell ===

# === Code Cell 11 ===
# Define the folder where CSV files will be saved
output_folder = "Hstat_windows_with_genes/Separated_by_H12_Categories"
os.makedirs(output_folder, exist_ok=True)


# Initialize a new dictionary of dictionaries to hold the subsets
by_H12_Category_Hstat_W_genes = {}

# Loop over each DataFrame in the original dictionary
for key, df in Hstat_W_genes.items():
    # Get the unique categories in the H1_category column
    categories = df['H12_category'].unique()

    # Create a dictionary to hold subsets for the current DataFrame
    by_H12_Category_Hstat_W_genes[key] = {}

    # Subset the DataFrame for each category
    for category in categories:
        # filtering the category
        subset_df = df[df['H12_category'] == category]

        # Replace invalid characters in the category name
              sanitized_category = re.sub(r'[<>:"/\\|?*]', '_', category.replace(' ',
'_').replace('%', 'pct'))

        # Save the subset DataFrame to a CSV file
        file_name = f"{key}_H12_category_{sanitized_category}.csv"
        file_path = os.path.join(output_folder, file_name)
        subset_df.to_csv(file_path, index=False)

        # Saving to the dictionary
        by_H12_Category_Hstat_W_genes[key][category] = subset_df

# Example: Accessing the subsets
for key, category_dict in by_H12_Category_Hstat_W_genes.items():
    print(f"DataFrame: {key}")
    for category, subset_df in category_dict.items():
        print(f"Category: {category}")
        print(subset_df.head())  # Print the first few rows of the subset


# === End of Cell ===

# === Code Cell 12 ===
# Define the folder where text files will be saved
output_folder = "H12_unique_genes_text_files"
os.makedirs(output_folder, exist_ok=True)

# Loop over the dictionary of dictionaries
for outer_key, inner_dict in by_H12_Category_Hstat_W_genes.items():
    for inner_key, df in inner_dict.items():
        # Extract all the unique genes from the 'Genes' column
        unique_genes = set()
        for gene_list in df['Genes']:
            unique_genes.update(gene_list)  # Combine all genes into the set

        # Create a meaningful and sanitized file name
        sanitized_outer_key = re.sub(r'[<>:"/\\|?*]', '_', outer_key)
        sanitized_inner_key = re.sub(r'[<>:"/\\|?*]', '_', inner_key)
                                                                      file_name         =
f"{sanitized_outer_key}_{sanitized_inner_key}_unique_genes.txt".replace(" ", "_")
        file_path = os.path.join(output_folder, file_name)

        # Save the unique genes to the text file
        with open(file_path, 'w') as f:
            for gene in sorted(unique_genes):  # Sort genes alphabetically
                f.write(f"{gene}\n")

        # Print a message to indicate progress
        print(f"Saved: {file_name}")

# === End of Cell ===
```

File: H12_viz.py

```
import pandas as pd
import matplotlib.pyplot as plt
import sys
import os

# Check if the input file is provided
if len(sys.argv) != 2:
    print("Usage: python script.py <input_csv_file>")
    sys.exit(1)

# Input file from command-line argument
input_file = sys.argv[1]
```

```
# Extract the prefix from the file name
file_name = os.path.basename(input_file)
prefix = file_name.split("_")[3] + "_" + file_name.split("_")[4] # Use the first part of
the file name as prefix

try:
    # Read the input CSV file
    data = pd.read_csv(input_file)

    # Check if the required H12 column exists
    if 'H12' not in data.columns:
        raise ValueError("The input file must contain an 'H12' column.")

    # Define the window size and step size
    window_size = 1000  # Window size in base pairs (bp)
    step_size = 500     # Step size in base pairs (bp)

    # Calculate genomic positions
    data['Position'] = data.index * step_size + window_size // 2

    # Plot the scatter plot for H12 values
    plt.figure(figsize=(30, 3))
    plt.plot(data['Position'], data['H12'], color='blue', linewidth=0.5, label='H12')
    plt.title('Genome-wide Distribution of H12 Values', fontsize=16)
    plt.xlabel('Genomic Position (bp)', fontsize=14)
    plt.ylabel('H12', fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()

    # Save the plot as a PNG file
    output_file = prefix + "_H12_distribution.png"
    plt.savefig(output_file, format='png', dpi=300)
    #plt.show()

    print(f"The scatter plot has been saved as {output_file}.")

except Exception as e:
    print(f"Error: {e}")
```

File: RoH.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10

# locating the directory with data
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/filtered_V
CFs

for pop in `ls *SampleNames.txt`
```

```
do

POPNAME=$(echo ${pop} | cut -d'S' -f1)

for CHR in `cat chrIDs.txt`

do

# Input arguments
VCF="YmEthInd_${CHR}_fltpass.vcf.gz"            # Input VCF file
OUTPUT_PREFIX="${POPNAME}_${CHR}"  # Output prefix for results

# Run bcftools roh
echo "Calculating RoH using bcftools roh..."
bcftools roh -S ${pop} --AF-dflt 0.224768 -o ${OUTPUT_PREFIX}_roh.txt $VCF

done

done
```

File: RoH_Calculation.ipynb

```
# === Code Cell 1 ===
!pip install seaborn


# === End of Cell ===

# === Code Cell 2 ===
# Improting libraries
import allel
import zarr
import numcodecs
import pandas as pd
from hmmlearn import hmm
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# === End of Cell ===

# === Code Cell 3 ===
# providing path to the zarr files
allChrs_zarr_path                                                                          =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_allChrs
_fltpass.zarr'
chrx_zarr_path                                                                             =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chrx_BA
P.zarr'
chr2_zarr_path                                                                             =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr2_BA
P.zarr'
chr3_zarr_path                                                                             =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr3_BA
P.zarr'

# === End of Cell ===

# === Code Cell 4 ===
# Opening actual zarr files and reading in the variant data into the environment
chrx_callset = zarr.open_group('YmEthInd_Chrx_BAP.zarr', mode='r')
chrx_callset
chr2_callset = zarr.open_group('YmEthInd_Chr2_BAP.zarr', mode='r')
chr2_callset
chr3_callset = zarr.open_group('YmEthInd_Chr3_BAP.zarr', mode='r')
chr3_callset

# === End of Cell ===

# === Code Cell 5 ===
# Don't keep this snippt without commenting out
# chr3_callset.tree(expand=True)

# === End of Cell ===

# === Code Cell 6 ===
# Load genotype data (GT - shape: variants x samples x ploidy)
chrx_genotypes = allel.GenotypeArray(chrx_callset['calldata/GT'])
chr2_genotypes = allel.GenotypeArray(chr2_callset['calldata/GT'])
chr3_genotypes = allel.GenotypeArray(chr3_callset['calldata/GT'])

# Load variant positions
chrx_positions = chrx_callset['variants/POS'][:]
chr2_positions = chr2_callset['variants/POS'][:]
chr3_positions = chr3_callset['variants/POS'][:]

# Chromosome lengths are give as the contig sizes to calculate the fraction of RoH
# These values are taken from the NCBI reference sequence for An. stephensi - accession
ID - GCF_013141755.1
# Depending on the organism this should change
chrx_size=22713616
chr2_size=93706023
chr3_size=88747589


# === End of Cell ===

# === Code Cell 7 ===
# Load sample names (optional, for population assignment)
samples = chrx_callset['samples'][:]
len(samples)

# Define populations (indices of individuals in each population)
India_indices = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
# Replace with indices of population 1
Ethiopia_indices = [20, 21, 22, 23, 24, 25, 26, 27, 37, 38, 39, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55]  # Replace with indices of population 2
```

```
Yemen_indices = [28, 29, 30, 31, 32, 33, 34, 35, 36, 40, 41, 42]  # Replace with indices
of population 3

# Subset genotypes by population
chrx_genotypes_India = chrx_genotypes.take(India_indices, axis=1)
chr2_genotypes_India = chr2_genotypes.take(India_indices, axis=1)
chr3_genotypes_India = chr3_genotypes.take(India_indices, axis=1)

chrx_genotypes_Ethiopia = chrx_genotypes.take(Ethiopia_indices, axis=1)
chr2_genotypes_Ethiopia = chr2_genotypes.take(Ethiopia_indices, axis=1)
chr3_genotypes_Ethiopia = chr3_genotypes.take(Ethiopia_indices, axis=1)

chrx_genotypes_Yemen = chrx_genotypes.take(Yemen_indices, axis=1)
chr2_genotypes_Yemen = chr2_genotypes.take(Yemen_indices, axis=1)
chr3_genotypes_Yemen = chr3_genotypes.take(Yemen_indices, axis=1)

# before the main loops, define pop for each sample name
# e.g. a dict mapping sample name ? its population
pop_of = {
               'SRR15257906':'India',    'SRR15257907':'India',    'SRR15257908':'India',
'SRR15257909':'India',
                  'SRR15257910':'India',   'SRR15257911':'India',   'SRR15257912':'India',
'SRR15257913':'India',
                  'SRR15257914':'India',   'SRR15257915':'India',   'SRR15293885':'India',
'SRR15293886':'India',
                  'SRR15293887':'India',   'SRR15293888':'India',   'SRR15293889':'India',
'SRR15293890':'India',
                  'SRR15293891':'India',   'SRR15293892':'India',   'SRR15293893':'India',
'SRR15293894':'India',
         'X1296':'Ethiopia', 'X1307':'Ethiopia', 'X1402':'Ethiopia', 'X1403':'Ethiopia',
'X1404':'Ethiopia', 'X1408':'Ethiopia', 'X1409':'Ethiopia',
               'X1410':'Ethiopia',  'X1415':'Yemen',  'X1416':'Yemen',  'X1417':'Yemen',
'X1419':'Yemen', 'X1420':'Yemen', 'X1421':'Yemen',
               'X1423':'Yemen',  'X1424':'Yemen',  'X1425':'Yemen',  'X1580':'Ethiopia',
'X1581':'Ethiopia', 'X1583':'Ethiopia', 'X1585':'Yemen',
             'X1586':'Yemen',  'X1587':'Yemen',  'X1604':'Ethiopia',  'X1605':'Ethiopia',
'X1673':'Ethiopia', 'X1676':'Ethiopia', 'X1679':'Ethiopia',
         'X1680':'Ethiopia', 'X1735':'Ethiopia', 'X1736':'Ethiopia', 'X1738':'Ethiopia',
'X1740':'Ethiopia', 'X1742':'Ethiopia', 'X1743':'Ethiopia',
       'X1747':'Ethiopia'
}

samples

# === End of Cell ===

# === Code Cell 8 ===
# Parameters for roh_mhmm
phet_roh      = 0.001
phet_nonroh   = (0.005, 0.02)
transition    = 1e-06
min_roh       = 100_000

# Define populations and chromosomes
```

```
populations   = ["India", "Ethiopia", "Yemen"]
chromosomes   = ["chrx", "chr2", "chr3"]

# This dictionary will hold: results_roh[pop][chrom] = DataFrame of all runs for that
pop/chrom
results_roh = {}
for pop in populations:
    results_roh[pop] = {}

for chrom in chromosomes:
    # open or reference your callset for this chrom
    callset = locals()[f"{chrom}_callset"]

    # get that chromosome?s sample names
    samples = callset['samples'][:]               # shape = (n_samples,)
    # build a boolean mask for which belong to each pop
    for pop in populations:
        mask = np.vectorize(lambda s: pop_of[s])(samples) == pop
        sample_idxs = np.nonzero(mask)[0]         # these are now valid 0..n_samples-1

        # grab the GenotypeArray & positions
        genotypes = locals()[f"{chrom}_genotypes"]
        positions = locals()[f"{chrom}_positions"]
        contig_size = locals()[f"{chrom}_size"]

        dfs = []
        for si in sample_idxs:
            gv = genotypes[:, si, :]
            roh_df, froh = allel.roh_mhmm(
                gv, positions,
                # phet_roh=phet_roh,
                # phet_nonroh=phet_nonroh,
                # transition=transition,
                min_roh=min_roh,
                is_accessible=None,
                contig_size=contig_size
            )
            roh_df['sample'] = samples[si]  # store the sample name
            roh_df['froh']   = froh
            dfs.append(roh_df)

        # concatenate and store
        all_roh = pd.concat(dfs, ignore_index=True)
        results_roh[pop][chrom] = all_roh


# Now results_roh["India"]["chr2"] is a DataFrame containing all RoH runs
# for every Indian sample on chromosome 2, with columns:
#   ['start','stop','length','is_marginal','sample','froh']


# === End of Cell ===

# === Code Cell 9 ===
# Assuming `results_roh` dictionary is already in the environment as defined previously.

for pop, chrom_dict in results_roh.items():
    for chrom, df in chrom_dict.items():
        # Create figure with two subplots: ROH length distribution and fROH per sample
        fig, axes = plt.subplots(2, 1, figsize=(12, 8))

        # Histogram of ROH lengths
        sns.histplot(df['length'], bins=50, ax=axes[0])
        axes[0].set_title(f'{pop} - {chrom}: ROH Length Distribution')
        axes[0].set_xlabel('ROH length (bp)')
        axes[0].set_ylabel('Count')

        # Barplot of fROH per sample
        froh_df = df[['sample', 'froh']].drop_duplicates().sort_values('froh')
        sns.barplot(x='sample', y='froh', data=froh_df, ax=axes[1])
        axes[1].set_title(f'{pop} - {chrom}: Fraction of Genome in ROH (fROH)')
        axes[1].set_xlabel('Sample')
        axes[1].set_ylabel('fROH')
        axes[1].tick_params(axis='x', rotation=45)

        plt.tight_layout()
        plt.show()


# === End of Cell ===

# === Code Cell 10 ===
for pop in populations:
    for chrom in chromosomes:
        results_roh[pop][chrom]['length'].plot(kind='kde') # , bins=5, edgecolor='black'
        plt.title('Distribution of RoH lengths (Histogram)')
        plt.xlabel('Length')
        plt.ylabel('Frequency')
        plt.show()

# === End of Cell ===

# === Code Cell 11 ===
# your input: results_roh[pop][chrom] ? DataFrame with columns
# ['start','stop','length','is_marginal','sample','froh']

window_size = 1000  # 1 kb windows

# output dict
probabilities = {}

for pop, chrom_dict in results_roh.items():
    probabilities[pop] = {}

    for chrom, roh_df in chrom_dict.items():
        # 1) find contig length (use max 'stop' position as a lower bound)
        contig_len = int(roh_df['stop'].max())
        # 2) tile the chromosome into non-overlapping windows:
        #    [0?1000), [1000?2000), ?, up to contig_len
        window_starts = np.arange(0, contig_len, window_size)

        # 3) get the set of all samples in this pop × chrom
        all_samples = roh_df['sample'].unique()
        n_samples = len(all_samples)

        rows = []
        for ws in window_starts:
            we = ws + window_size
            # a) select RoH segments overlapping [ws, we)
            ov = roh_df[
                (roh_df['stop']  >= ws) &
                (roh_df['start'] <  we)
            ]
            # b) count how many distinct samples have ?1 overlapping RoH
            n_overlap = ov['sample'].nunique()
            prop = n_overlap / n_samples

            rows.append({
                'window_start': ws,
                'window_end':   we,
                'proportion':   prop
            })

        # 4) assemble into DataFrame and store
        prob_df = pd.DataFrame(rows)
        probabilities[pop][chrom] = prob_df

        # 5) quick line?plot of proportion vs position
        plt.figure(figsize=(8, 2.5))
        plt.plot(
            prob_df['window_start'],
            prob_df['proportion'],
            lw=1
        )
        plt.ylim(0, 1)
        plt.xlabel('Genomic position (bp)')
        plt.ylabel('Fraction in RoH')
        plt.title(f'{pop}: {chrom}')
        plt.tight_layout()
        plt.show()


# === End of Cell ===
```

File: RoH_viz.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10
```

```
# locating the directory with data
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/filtered_V
CFs/RoH_results

module load python

for pop in `ls *_roh.txt`

do

POPCHR=$(echo $pop | cut -d'.' -f1)

grep '^RG' "$pop" > "${POPCHR}_RG.txt"

python roh_viz2.py ${POPCHR}_RG.txt

done
```

File: fixedNselected_gene_stats.ipynb

```
# === Code Cell 1 ===
# Importing necessary python libraries

import allel
import numpy as np
import zarr
import os
import csv
import pandas as pd


# === End of Cell ===

# === Code Cell 2 ===
# providing path to the zarr files
zarrPaths                                                                              =
['/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_allChr
s_fltpass.zarr',

'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chrx_BA
P.zarr',

'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr2_BA
P.zarr',

'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr3_BA
P.zarr',]

zarrPaths[0]

# === End of Cell ===
# === Code Cell 3 ===
# Define populations (indices of individuals in each population)
India_indices = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
# Replace with indices of population 1
Ethiopia_indices = [20, 21, 22, 23, 24, 25, 26, 27, 37, 38, 39, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55]  # Replace with indices of population 2
Yemen_indices = [28, 29, 30, 31, 32, 33, 34, 35, 36, 40, 41, 42]  # Replace with indices
of population 3


# === End of Cell ===

# === Code Cell 4 ===
def calculate_fst_h12_from_zarr(zarr_file, start, end, pop1_indices, pop2_indices):
    """
    Calculate Fst using the Weir & Cockerham method for a specific region.

    Parameters:
        zarr_file (str): Path to the .zarr file containing genotype data.
        start (int): Start position of the region (0-based).
        end (int): End position of the region (1-based).
        pop1_indices (list of int): Indices of samples belonging to population 1.
        pop2_indices (list of int): Indices of samples belonging to population 2.

    Returns:
        float: Average Fst value for the selected region.
    """

    # Load the genotype data from the Zarr file
    callset = zarr.open_group(zarr_file, mode='r')

    # Extract variant positions
    positions = allel.SortedIndex(callset['variants/POS'][:])

    # Identify variant indices within the given region
    region_mask = (positions >= start) & (positions <= end)
    variant_indices = np.where(region_mask)[0]

    if len(variant_indices) == 0:
        print(f"No variants found in the region {start}-{end}. Returning NaN.")
        return np.nan

    # Extract genotype data for the selected region
    try:
          genotype_array = allel.GenotypeArray(callset['calldata/GT'][:])  # Ensure 3D
format
        genotype_region = genotype_array[variant_indices]  # Subset for selected region
    except Exception as e:
        print(f"Error loading genotype data: {e}")
        return np.nan

    # Validate genotype shape
    if genotype_region.ndim != 3 or genotype_region.shape[2] != 2:
        print(f"Unexpected genotype shape: {genotype_region.shape}. Returning NaN.")
        return np.nan

    # Compute allele counts for each population
    ac1 = genotype_region.count_alleles(subpop=pop1_indices)
    ac2 = genotype_region.count_alleles(subpop=pop2_indices)

    # Ensure non-empty allele counts
    if ac1.shape[0] == 0 or ac2.shape[0] == 0:
        print("Empty allele counts for the region. Returning NaN.")
        return [np.nan, "0"]

    # Calculate Hudson's Fst
     fst_hudson, _, _, _ = allel.average_hudson_fst(ac1, ac2, len(variant_indices)) # Can
change between Hudson and WC 1984

    # Return average Fst value for the region
    #return np.nanmean(fst_wc)  # Use np.nanmean to avoid NaN issues
    #return [fst_hudson, str(len(variant_indices))]

    # **Calculate Garud's H12 statistic**
    try:
        # Extract haplotypes from the genotype array
        haplotypes = genotype_region.to_haplotypes()

        # Compute haplotype frequencies
        h12_results = allel.garud_h(haplotypes)

        # Extract the H12 statistic
        h12_stat = h12_results[1]  # H12 is the second value in the returned tuple
    except Exception as e:
        print(f"Error calculating H12: {e}")
        h12_stat = np.nan

    # Return Fst, H12, and the number of variants in the region
    return [fst_hudson, h12_stat, str(len(variant_indices))]


# === End of Cell ===

# === Code Cell 5 ===
# Method testing block
# Calculating Fst value
stats_values                                                                           =
calculate_fst_h12_from_zarr('/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants
/Zarr_files/YmEthInd_Chr2_BAP.zarr',      67271604,      67277466,      Ethiopia_indices,
India_indices)
print(stats_values)


# === End of Cell ===

# === Code Cell 6 ===
# Giving path to the directory with CSV files
```

```
csv_dir                                                                                        =
"/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/Det_genes_fixedN
selected_filtered"

# Getting the names of the csv files
csv_files = [file for file in os.listdir(csv_dir) if file.endswith('.csv')]

# Iterating over the CSV files
for file_name in csv_files:
    file_path = os.path.join(csv_dir, file_name)
    csvdf = pd.read_csv(file_path)
    pop1 = file_name.split("_")[1]
    pop2 = file_name.split("_")[3]
    chrname = file_name.split("_")[4]
    chrname = chrname.split(".")[0]
    print("Population 1 - " + pop1 + " Population 2 - " + pop2 + " Chromosome name - " +
chrname)

    # Creating a new list to hold new row list with Fst values and number of variants in
the gene
    new_rows_list = []

    # Getting column names of the data frame
    df_col_names = csvdf.columns

    # Iterating over the rows of the data frame
    for index, row in csvdf.iterrows():
        row_list = row.to_list()
        start_position = row_list[1]
        end_position = row_list[2]
        pop1_sample_indices = []  # Indices of Population 1
        pop2_sample_indices = []  # Indices of Population 2

        # Setting the population 1
        if pop1 == "Yemen":
            pop1_sample_indices = Yemen_indices
        elif pop1 == "Ethiopia":
            pop1_sample_indices = Ethiopia_indices
        elif pop1 == "India":
            pop1_sample_indices = India_indices
        elif pop1 == "YemenNEthiopia":
            pop1_sample_indices = Yemen_indices + Ethiopia_indices

        # Setting the population 2
        if pop2 == "Yemen":
            pop2_sample_indices = Yemen_indices
        elif pop2 == "Ethiopia":
            pop2_sample_indices = Ethiopia_indices
        elif pop2 == "India":
            pop2_sample_indices = India_indices
        elif pop2 == "YemenNEthiopia":
            pop2_sample_indices = Yemen_indices + Ethiopia_indices

        # Picking the correct zarr path based on the chromosome
        if chrname == "ChrX":
            zarr_path = zarrPaths[1]
        elif chrname == "Chr2":
            zarr_path = zarrPaths[2]
        elif chrname == "Chr3":
            zarr_path = zarrPaths[3]

        # Calculating Fst value
                stats_values  =  calculate_fst_h12_from_zarr(zarr_path,  start_position,
end_position, pop1_sample_indices, pop2_sample_indices)

        # adding the Fst value and number of variants to the row list
        if isinstance(stats_values, list) and len(stats_values) == 3:
            row_list = row_list + stats_values
        else:
                print(f"Unexpected  return  type  from  function:  {type(stats_values)}.
Fixing...")
            row_list = row_list + [np.nan, np.nan, "0"]  # Ensure consistent list size

        # adding the new row to new row list
        new_rows_list.append(row_list)

    # Creating a data frame from the new rows list
      new_df  =  pd.DataFrame(new_rows_list,  columns=list(df_col_names)  +  ["Fst_value",
"H12_value", "number_of_variants"])
    # print(new_df.head)

    # Saving the new data frame as
    new_df.to_csv(file_name.split(".")[0] + "_WFstNH12.csv", index=False)




# === End of Cell ===
```

File: selection_stats.ipynb

```
# === Code Cell 1 ===
# Improting libraries
import allel
import zarr
import numcodecs
import pandas as pd

# === End of Cell ===

# === Code Cell 2 ===
# providing path to the zarr files
allChrs_zarr_path                                                                       =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_allChrs
```

```
_fltpass.zarr'
chrx_zarr_path                                                                                 =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chrx_BA
P.zarr'
chr2_zarr_path                                                                                 =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr2_BA
P.zarr'
chr3_zarr_path                                                                                 =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr3_BA
P.zarr'

# === End of Cell ===

# === Code Cell 3 ===
chrx_callset = zarr.open_group('YmEthInd_Chrx_BAP.zarr', mode='r')
chrx_callset
chr2_callset = zarr.open_group('YmEthInd_Chr2_BAP.zarr', mode='r')
chr2_callset
chr3_callset = zarr.open_group('YmEthInd_Chr3_BAP.zarr', mode='r')
chr3_callset

# === End of Cell ===

# === Code Cell 4 ===
# chrx_callset.tree(expand=True)

# === End of Cell ===

# === Code Cell 5 ===
# Load genotype data (GT - shape: variants x samples x ploidy)
chrx_genotypes = allel.GenotypeArray(chrx_callset['calldata/GT'])
chr2_genotypes = allel.GenotypeArray(chr2_callset['calldata/GT'])
chr3_genotypes = allel.GenotypeArray(chr3_callset['calldata/GT'])

# Load sample names (optional, for population assignment)
samples = chrx_callset['samples'][:]
samples

# === End of Cell ===

# === Code Cell 6 ===
# Define populations (indices of individuals in each population)
India_indices = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
# Replace with indices of population 1
Ethiopia_indices = [20, 21, 22, 23, 24, 25, 26, 27, 37, 38, 39, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55]  # Replace with indices of population 2
Yemen_indices = [28, 29, 30, 31, 32, 33, 34, 35, 36, 40, 41, 42]  # Replace with indices
of population 3

# Subset genotypes by population
chrx_genotypes_India = chrx_genotypes.take(India_indices, axis=1)
chr2_genotypes_India = chr2_genotypes.take(India_indices, axis=1)
chr3_genotypes_India = chr3_genotypes.take(India_indices, axis=1)
```

```
chrx_genotypes_Ethiopia = chrx_genotypes.take(Ethiopia_indices, axis=1)
chr2_genotypes_Ethiopia = chr2_genotypes.take(Ethiopia_indices, axis=1)
chr3_genotypes_Ethiopia = chr3_genotypes.take(Ethiopia_indices, axis=1)

chrx_genotypes_Yemen = chrx_genotypes.take(Yemen_indices, axis=1)
chr2_genotypes_Yemen = chr2_genotypes.take(Yemen_indices, axis=1)
chr3_genotypes_Yemen = chr3_genotypes.take(Yemen_indices, axis=1)

# === End of Cell ===

# === Code Cell 7 ===
# Convert genotypes to haplotypes (unphased or phased)
chrx_haplotypes_India = chrx_genotypes_India.to_haplotypes()
chr2_haplotypes_India = chr2_genotypes_India.to_haplotypes()
chr3_haplotypes_India = chr3_genotypes_India.to_haplotypes()

chrx_haplotypes_Ethiopia = chrx_genotypes_Ethiopia.to_haplotypes()
chr2_haplotypes_Ethiopia = chr2_genotypes_Ethiopia.to_haplotypes()
chr3_haplotypes_Ethiopia = chr3_genotypes_Ethiopia.to_haplotypes()

chrx_haplotypes_Yemen = chrx_genotypes_Yemen.to_haplotypes()
chr2_haplotypes_Yemen = chr2_genotypes_Yemen.to_haplotypes()
chr3_haplotypes_Yemen = chr3_genotypes_Yemen.to_haplotypes()

# === End of Cell ===

# === Code Cell 8 ===
# Define populations and chromosomes
populations = ["India", "Ethiopia", "Yemen"]
chromosomes = ["chrx", "chr2", "chr3"]

# Define window size (e.g., 100,000 variants) and step size (e.g., 50,000 variants)
window_size = 1000
step_size = 500

for pop in populations:
    for chrom in chromosomes:

        # Construct the variable name dynamically
        haplotypes_var_name = f"{chrom}_haplotypes_{pop}"

        # Access the variable from the local namespace
        haplotypes_pop = locals()[haplotypes_var_name]

        # Calculate Garud's H statistics in sliding windows for population 1
        h1_values, h12_values, h123_values, h2_h1_values = allel.moving_garud_h(
            haplotypes_pop,
            size=window_size,
            step=step_size
        )

        # Create a DataFrame for sliding window results
        df = pd.DataFrame({
            'H1': h1_values,
            'H12': h12_values,
            'H123': h123_values,
            'H2/H1': h2_h1_values
        })

        # Save to a CSV file
        df.to_csv(f"garuds_h_statistics_{pop}_{chrom}.csv", index=False)

# === End of Cell ===

# === Code Cell 9 ===


# === End of Cell ===
```

File: vcf2zarr_conversion.ipynb

```
# === Code Cell 1 ===
import allel
import sys
print(allel.__version__)

# === End of Cell ===

# === Code Cell 2 ===
!pip install zarr
!pip install numcodecs

# === End of Cell ===

# === Code Cell 3 ===
import zarr
import numcodecs

# === End of Cell ===

# === Code Cell 4 ===
vcf_path                                                                                    =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/filtered_
VCFs/YmEthInd_allChrs_fltpass.vcf'
zarr_path                                                                                   =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_allChrs
_fltpass.zarr'

# === End of Cell ===

# === Code Cell 5 ===
allel.vcf_to_zarr(vcf_path, zarr_path, fields='*', overwrite =True)

# === End of Cell ===
# === Code Cell 6 ===
callset_zarr = zarr.open_group('YmEthInd_allChrs_fltpass.zarr', mode='r')
callset_zarr

# === End of Cell ===

# === Code Cell 7 ===
callset_zarr.tree(expand=True)

# === End of Cell ===

# === Code Cell 8 ===
# Chromosome X
chrx_vcf_path                                                                            =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/biallelic
_phased_VCFs/YmEthInd_NC_050201.1_BA_Phased.vcf.gz.vcf.gz'
chrx_zarr_path                                                                           =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chrx_BA
P.zarr'
allel.vcf_to_zarr(chrx_vcf_path, chrx_zarr_path, fields='*', overwrite =True)

# Chromosome 2
chr2_vcf_path                                                                            =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/biallelic
_phased_VCFs/YmEthInd_NC_050202.1_BA_Phased.vcf.gz.vcf.gz'
chr2_zarr_path                                                                           =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr2_BA
P.zarr'
allel.vcf_to_zarr(chr2_vcf_path, chr2_zarr_path, fields='*', overwrite =True)

# Chromosome 3
chr3_vcf_path                                                                            =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/biallelic
_phased_VCFs/YmEthInd_NC_050203.1_BA_Phased.vcf.gz.vcf.gz'
chr3_zarr_path                                                                           =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr3_BA
P.zarr'
allel.vcf_to_zarr(chr3_vcf_path, chr3_zarr_path, fields='*', overwrite =True)



# === End of Cell ===

# === Code Cell 9 ===


# === End of Cell ===
```

Section: ddRAD_analysis

File: BAfilter_N_Phasing.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=5

# setting working directory
cd /data/gunarathnai/Ans_GA/ddRad_variants


# filtering biallelic variants
bcftools     view     -Oz     -m     2     -M     2     ddRAD_autosomal_ons.vcf.gz     -o
ddRAD_autosomal_ons_BAFlt.vcf.gz --threads 5

# indexing the input file
bcftools index ddRAD_autosomal_ons_BAFlt.vcf.gz -@ 5

# Pahsing biallelic VCF file using beagle 5.4
java -Xmx200g -jar /home/gunarathnai/bin/beagle/beagle_V5.4.jar \
 nthreads=5 burnin=5 iterations=20 \
 gt=ddRAD_autosomal_ons_BAFlt.vcf.gz \
 out=ddRAD_autosomal_ons_BA_Phased.vcf.gz
```

File: Fst_Calc.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10

# locating the directory with data
cd /data/gunarathnai/Ans_GA/ddRad_variants/Jeannes_samples_WOZabid/Fst_calculations

# setting background for conda environment activation
eval "$(conda shell.bash hook)"

# activating 'bio2' environment with conda
conda activate newbio

# Iterating over VCF files to generate Fst
for VCF in `cat VCFs4Fst.txt`
do

for pp in `cat pop_pairs.txt`
do

POP1=$(echo ${pp} | cut -d',' -f1)
POP2=$(echo ${pp} | cut -d',' -f2)
```

```
OUT="$POP1"
OUT+="$POP2"
OUT+="$VCF"

vcftools --gzvcf ${VCF} \
 --weir-fst-pop ${POP1} --weir-fst-pop ${POP2} \
 --fst-window-size 100000 \
 --fst-window-step 50000 \
 --out ${OUT}_Fst

done
done
```

File: admixture_YmEthSml_ddRAD_4.9.25.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=10
#PBS -k doe

set -e

echo "Job started on $(date)"
conda info
conda list

cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/IndYmEthSm
l_individual_gVCFs/biallelic_phased_VCFs/admixture_analysis/YmEthSml_ddRAD

# Prepare PLINK files
## Exclude SNPs with Minor Allele Frequency  lower than 0.01 and Genotype missingness
per SNP greater than 30%, Individual missingness > 10%

plink --vcf autosomal_YmEthSml_ddRAD_intersect.vcf.gz --double-id --allow-extra-chr \
--set-missing-var-ids @:# \
--maf 0.01 --geno 0.1 --mind 0.3 \
--make-bed --out autosomal_YmEthSml_ddRAD_intersect_LD

# LD pruning
## Filtering out linkage r sqrd greater than 0.3 for 50 variants window with a step size
of 5

plink  --bfile  autosomal_YmEthSml_ddRAD_intersect_LD  --indep-pairwise  50  5  0.3  --out
pruned_snps --allow-extra-chr
plink   --bfile   autosomal_YmEthSml_ddRAD_intersect_LD   --extract   pruned_snps.prune.in
--make-bed --out final_admixture --allow-extra-chr

# Activate conda environment
eval "$(conda shell.bash hook)"
conda activate newbio
```

```
# Rename chromosome names to "0"
awk '{$1="0";print $0}' final_admixture.bim > final_admixture.bim.tmp
mv final_admixture.bim.tmp final_admixture.bim

# Run admixture analysis
for K in $(seq 2 10)
do
    echo "Running admixture analysis for K=$K..."
    admixture --cv -j10 final_admixture.bed $K > admixture_K${K}.log

    CV_ERROR=$(grep -o "CV error.*" admixture_K${K}.log | awk '{print $3}')
    echo "K=$K: CV error = $CV_ERROR"
done

# Summarize CV errors
grep -h "CV error" admixture_K*.log > cross_validation_summary.txt
echo "Admixture analysis completed. Summary written to cross_validation_summary.txt"
```

File: admixture_ddRAD.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=3

# locating the directory with data
cd /data/gunarathnai/Ans_GA/ddRad_variants/Jeannes_samples_WOZabid/admixture

# setting background for conda environment activation
eval "$(conda shell.bash hook)"

# activating 'newbio' environment with conda
conda activate newbio

# ADMIXTURE does not accept chromosome names that are not human chromosomes. We will thus just exchange the first column by 0
awk '{$1="0";print $0}' final_admixture.bim > final_admixture.bim.tmp
mv final_admixture.bim.tmp final_admixture.bim

# Run admixture analysis for each value of K with cross validation
for K in $(seq 2 10)
do
    echo "Running admixture analysis for K=$K..."
    admixture --cv -j30 final_admixture.bed $K > admixture_K${K}.log

    # Extract cross-validation error
    CV_ERROR=$(grep -o "CV error.*" admixture_K${K}.log | awk '{print $3}')
    echo "K=$K: CV error = $CV_ERROR"
done

# Summarize results
echo "Admixture analysis completed. Cross-validation results:"
grep -h "CV error" admixture_K*.log
```

File: autosomal_LD_PCA_ddRAD_4.9.25.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=5
#PBS -k doe

# locating the directory with data
cd
/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/IndYmEthSm
l_individual_gVCFs/biallelic_phased_VCFs/ddRAD_intersections

# printing hostname on Kodiak
hostname

# linkage pruning
plink  --vcf  autosomal_YmEthSml_ddRAD_intersect.vcf.gz  --double-id  --allow-extra-chr
--list-duplicate-vars \
--set-missing-var-ids @:#_\$1_\$2 \
--indep-pairwise 50 10 0.1 --out autosomal_YmEthSml_ddRAD_intersect

# Performing PCA
# The "var-wts" flag will produce the loadings of the genes for each PC
plink  --vcf  autosomal_YmEthSml_ddRAD_intersect.vcf.gz  --double-id  --allow-extra-chr
--set-missing-var-ids @:#_\$1_\$2 \
--extract autosomal_YmEthSml_ddRAD_intersect.prune.in --list-duplicate-vars \
--make-bed --pca var-wts --out autosomal_YmEthSml_ddRAD_intersect_pca
```

File: ddRAD_admixture_input_new.sh

```
#!/usr/bin/bash
#PBS -l nodes=1:ppn=2

# locating the directory with data
cd /data/gunarathnai/Ans_GA/ddRad_variants

plink --vcf ddRAD_autosomal_WOZabid_BAFlt.vcf.gz --double-id --allow-extra-chr \
--set-missing-var-ids @:# \
--maf 0.01 --geno 0.1 --mind 0.1 \
--make-bed --out ddRAD_autosomal_WOZabid_BAFlt_LD
```

```
plink --bfile ddRAD_autosomal_WOZabid_BAFlt_LD --indep-pairwise 50 5 0.2 --out
pruned_snps --allow-extra-chr
plink --bfile ddRAD_autosomal_WOZabid_BAFlt_LD --extract pruned_snps.prune.in --make-bed
--out final_admixture --allow-extra-chr
```

File: generateNJTree.py

```
import allel
from Bio import Phylo
from Bio.Phylo.TreeConstruction import DistanceTreeConstructor, DistanceMatrix
import numpy as np
from ete3 import Tree #, TreeStyle

# Step 1: Load the VCF file and extract sample names and genotype data
vcf_file = 'ddRAD_autosomal_WOZabid_BAFlt.vcf'  # Replace with your VCF file path
callset = allel.read_vcf(vcf_file)
sample_names = callset['samples']
genotypes = allel.GenotypeArray(callset['calldata/GT'])

# Step 2: Convert genotypes to haplotypes (0, 1, or 2 to represent allele counts)
haplotypes = genotypes.to_n_alt()  # Convert genotypes to allele counts

# Step 3: Calculate the pairwise distances and store in lower triangular format
def calculate_lower_triangle_distance_matrix(haplotypes):
    n_samples = haplotypes.shape[1]
    dist_matrix = []

    for i in range(n_samples):
        row = []
        for j in range(i):  # Only calculate for lower triangle
                dist = np.sum(haplotypes[:, i] != haplotypes[:, j])  # Count allele
differences
            row.append(dist)
        row.append(0)  # Diagonal elements (distance to self)
        dist_matrix.append(row)

    return dist_matrix

dist_matrix = calculate_lower_triangle_distance_matrix(haplotypes)

# Step 4: Convert the distance matrix to a format compatible with Bio.Phylo
labels = list(sample_names)  # Original sample names from VCF
matrix = DistanceMatrix(names=labels, matrix=dist_matrix)

# Step 5: Construct the tree using the distance matrix
constructor = DistanceTreeConstructor()
tree = constructor.nj(matrix)  # Using Neighbor-Joining (NJ) method

# Step 6: Save the tree to a Newick file using Bio.Phylo
newick_file = "temp_tree.nw"
Phylo.write(tree, newick_file, "newick")
```

```
# Step 7: Load the Newick file with ete3
ete_tree = Tree(newick_file, format=1)

# Step 8: Rename the leaves in the ete3 tree to match original sample names
# Create a dictionary mapping new leaf names to original sample names
leaf_names = ete_tree.get_leaf_names()
name_mapping = {leaf: sample for leaf, sample in zip(leaf_names, sample_names)}

# Rename leaves in the ete3 tree
for leaf in ete_tree:
    if leaf.name in name_mapping:
        leaf.name = name_mapping[leaf.name]

# Step 9: Save the renamed tree
ete_tree.write(format=1, outfile="phylogenetic_tree_renamed.nw")

# Optional: Save the tree as an image
ete_tree.render("phylogenetic_tree_renamed.png", w=800, units="px")
```

File: selection_stats.ipynb

```
# === Code Cell 1 ===
# Improting libraries
import allel
import zarr
import numcodecs
import pandas as pd

# === End of Cell ===

# === Code Cell 2 ===
# providing path to the zarr files
allChrs_zarr_path                                                          =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_allChrs
_fltpass.zarr'
chrx_zarr_path                                                             =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chrx_BA
P.zarr'
chr2_zarr_path                                                             =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr2_BA
P.zarr'
chr3_zarr_path                                                             =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr3_BA
P.zarr'

# === End of Cell ===

# === Code Cell 3 ===
chrx_callset = zarr.open_group('YmEthInd_Chrx_BAP.zarr', mode='r')
chrx_callset
chr2_callset = zarr.open_group('YmEthInd_Chr2_BAP.zarr', mode='r')
```

```
chr2_callset
chr3_callset = zarr.open_group('YmEthInd_Chr3_BAP.zarr', mode='r')
chr3_callset

# === End of Cell ===

# === Code Cell 4 ===
# chrx_callset.tree(expand=True)

# === End of Cell ===

# === Code Cell 5 ===
# Load genotype data (GT - shape: variants x samples x ploidy)
chrx_genotypes = allel.GenotypeArray(chrx_callset['calldata/GT'])
chr2_genotypes = allel.GenotypeArray(chr2_callset['calldata/GT'])
chr3_genotypes = allel.GenotypeArray(chr3_callset['calldata/GT'])

# Load sample names (optional, for population assignment)
samples = chrx_callset['samples'][:]
samples

# === End of Cell ===

# === Code Cell 6 ===
# Define populations (indices of individuals in each population)
India_indices = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
# Replace with indices of population 1
Ethiopia_indices = [20, 21, 22, 23, 24, 25, 26, 27, 37, 38, 39, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55]  # Replace with indices of population 2
Yemen_indices = [28, 29, 30, 31, 32, 33, 34, 35, 36, 40, 41, 42]  # Replace with indices
of population 3

# Subset genotypes by population
chrx_genotypes_India = chrx_genotypes.take(India_indices, axis=1)
chr2_genotypes_India = chr2_genotypes.take(India_indices, axis=1)
chr3_genotypes_India = chr3_genotypes.take(India_indices, axis=1)

chrx_genotypes_Ethiopia = chrx_genotypes.take(Ethiopia_indices, axis=1)
chr2_genotypes_Ethiopia = chr2_genotypes.take(Ethiopia_indices, axis=1)
chr3_genotypes_Ethiopia = chr3_genotypes.take(Ethiopia_indices, axis=1)

chrx_genotypes_Yemen = chrx_genotypes.take(Yemen_indices, axis=1)
chr2_genotypes_Yemen = chr2_genotypes.take(Yemen_indices, axis=1)
chr3_genotypes_Yemen = chr3_genotypes.take(Yemen_indices, axis=1)

# === End of Cell ===

# === Code Cell 7 ===
# Convert genotypes to haplotypes (unphased or phased)
chrx_haplotypes_India = chrx_genotypes_India.to_haplotypes()
chr2_haplotypes_India = chr2_genotypes_India.to_haplotypes()
chr3_haplotypes_India = chr3_genotypes_India.to_haplotypes()

chrx_haplotypes_Ethiopia = chrx_genotypes_Ethiopia.to_haplotypes()
```

```
chr2_haplotypes_Ethiopia = chr2_genotypes_Ethiopia.to_haplotypes()
chr3_haplotypes_Ethiopia = chr3_genotypes_Ethiopia.to_haplotypes()

chrx_haplotypes_Yemen = chrx_genotypes_Yemen.to_haplotypes()
chr2_haplotypes_Yemen = chr2_genotypes_Yemen.to_haplotypes()
chr3_haplotypes_Yemen = chr3_genotypes_Yemen.to_haplotypes()

# === End of Cell ===

# === Code Cell 8 ===
# Define populations and chromosomes
populations = ["India", "Ethiopia", "Yemen"]
chromosomes = ["chrx", "chr2", "chr3"]

# Define window size (e.g., 100,000 variants) and step size (e.g., 50,000 variants)
window_size = 1000
step_size = 500

for pop in populations:
    for chrom in chromosomes:

        # Construct the variable name dynamically
        haplotypes_var_name = f"{chrom}_haplotypes_{pop}"

        # Access the variable from the local namespace
        haplotypes_pop = locals()[haplotypes_var_name]

        # Calculate Garud's H statistics in sliding windows for population 1
        h1_values, h12_values, h123_values, h2_h1_values = allel.moving_garud_h(
            haplotypes_pop,
            size=window_size,
            step=step_size
        )

        # Create a DataFrame for sliding window results
        df = pd.DataFrame({
            'H1': h1_values,
            'H12': h12_values,
            'H123': h123_values,
            'H2/H1': h2_h1_values
        })

        # Save to a CSV file
        df.to_csv(f"garuds_h_statistics_{pop}_{chrom}.csv", index=False)

# === End of Cell ===

# === Code Cell 9 ===


# === End of Cell ===
```

File: vcf2zarr_conversion.ipynb

```
# === Code Cell 1 ===
import allel
import sys
print(allel.__version__)

# === End of Cell ===

# === Code Cell 2 ===
callset                                                                                =
allel.read_vcf('/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Com
bined/filtered_VCFs/YmEthInd_allChrs_fltpass.vcf', fileds = '*')

# === End of Cell ===

# === Code Cell 3 ===
sorted(callset.keys())

# === End of Cell ===

# === Code Cell 4 ===
callset['samples']

# === End of Cell ===

# === Code Cell 5 ===
gt = allel.GenotypeArray(callset['calldata/GT'])
gt

# === End of Cell ===

# === Code Cell 6 ===
gt.is_het()
gt.count_het(axis=1)

# === End of Cell ===

# === Code Cell 7 ===
ac = gt.count_alleles()
ac

# === End of Cell ===

# === Code Cell 8 ===
!pip install zarr
!pip install numcodecs

# === End of Cell ===

# === Code Cell 9 ===
import zarr
import numcodecs
```

```
# === End of Cell ===

# === Code Cell 10 ===
vcf_path                                                                                        =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/filtered_
VCFs/YmEthInd_allChrs_fltpass.vcf'
zarr_path                                                                                       =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_allChrs
_fltpass.zarr'

# === End of Cell ===

# === Code Cell 11 ===
allel.vcf_to_zarr(vcf_path, zarr_path, fields='*', overwrite =True)

# === End of Cell ===

# === Code Cell 12 ===
callset_zarr = zarr.open_group('YmEthInd_allChrs_fltpass.zarr', mode='r')
callset_zarr

# === End of Cell ===

# === Code Cell 13 ===
callset_zarr.tree(expand=True)

# === End of Cell ===

# === Code Cell 14 ===
# Chromosome X
chrx_vcf_path                                                                                   =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/biallelic
_phased_VCFs/YmEthInd_NC_050201.1_BA_Phased.vcf.gz.vcf.gz'
chrx_zarr_path                                                                                  =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chrx_BA
P.zarr'
allel.vcf_to_zarr(chrx_vcf_path, chrx_zarr_path, fields='*', overwrite =True)

# Chromosome 2
chr2_vcf_path                                                                                   =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/biallelic
_phased_VCFs/YmEthInd_NC_050202.1_BA_Phased.vcf.gz.vcf.gz'
chr2_zarr_path                                                                                  =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr2_BA
P.zarr'
allel.vcf_to_zarr(chr2_vcf_path, chr2_zarr_path, fields='*', overwrite =True)

# Chromosome 3
chr3_vcf_path                                                                                   =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/YmEthInd_Combined/biallelic
_phased_VCFs/YmEthInd_NC_050203.1_BA_Phased.vcf.gz.vcf.gz'
chr3_zarr_path                                                                                  =
'/data/gunarathnai/Ans_GA/gatk_variants/YmEthSmlInd_variants/Zarr_files/YmEthInd_Chr3_BA
P.zarr'
```

```
allel.vcf_to_zarr(chr3_vcf_path, chr3_zarr_path, fields='*', overwrite =True)



# === End of Cell ===

# === Code Cell 15 ===


# === End of Cell ===
```