INTRODUCTION

The Heliconius genus, or the brush-footed butterflies, known for their ecological adaptability and mimicry (Mallet, 2010; Jiggins, 2017), offers a unique model for studying the genetic basis of behavioral traits, such as pollination. Mitochondrial (COI) and nuclear (EF1α) genes in Heliconius butterflies evolve under different mutational pressures, with mitochondrial DNA typically showing higher rates of divergence (Rubinoff & Holland, 2005). This project aims to investigate whether these genes exhibit distinct clustering patterns, revealing potential ecological and evolutionary mechanisms underlying pollination behaviors. Understanding these clustering patterns is crucial, as they can elucidate how specific genetic adaptations influence pollination behaviors, providing insights into the evolutionary pressures that shape ecological interactions within the Heliconius genus.

Two contrasting hypotheses emerge: it can be expected that mitochondrial genes will cluster more tightly within species due to their rapid, maternally inherited evolution, whereas if both genes show similar clustering, this might indicate shared evolutionary pathways, suggesting a more complex relationship between gene type and ecological adaptation. Using clustering algorithms and unsupervised learning, this project will implement clustering metrics to evaluate whether differences in clustering reflect ecological adaptations specific to each gene type. This approach integrates an ecological application for the DNA sequence analysis to uncover gene-environment interactions within Heliconius.

CODE SECTION 1: DATA (DATA EXPLORATION AND QUALITY CONTROL)

```
# Libraries (assuming all are pre-installed)
library(rentrez)
library(Biostrings) # For DNAStringSet
library(stringr) # For string manipulation
library(dplyr) # For data manipulation
library(ggplot2) # For plotting
library(DECIPHER)
library(ape) # For dist.dna and distance calculation
library(cluster) # For k-means and hierarchical clustering
library(fpc) # For evaluation of clustering
library(ggdendro)
library(ggplot2)
library(ggfortify)

# Several functions were created in this assignment to simplify and generalize
the procedures of alignment and applying ML algorithms to DNA data from NCBI.
# All functions are shown and then are called for the specific variables.
# As a next step, a meta function/algorithm can be created integrating all
these functions for easier and effective implementation.

#### CODE SECTION 1: DATA (DATA EXPLORATION AND QUALITY CONTROL) -----

## ------ 1. Fetch Sequences from NCBI -----
```

```
# Before data was fetched, sequences were searched on NCBI for a general
overview. It was seen that COI had 2k+ sequences, whereas EF1a had only 240.
# To eliminate a strong imbalance and bias, 200 sequences were imported for
both.

fetch_sequences <- function(gene, organism, max_results = 200, output_file) {
  # Construct search term for NCBI query
  search_term <- paste0(organism, "[Organism] AND ", gene, "[Gene]")

  # Search NCBI database for the sequences
  search_result <- entrez_search(db = "nuccore", term = search_term, rettype =
"fasta", retmax = max_results)

  # Fetch sequences based on the search results
  fetched_sequences <- entrez_fetch(db = "nuccore", id = search_result$ids,
rettype = "fasta")

  # Write fetched sequences to the output file
  writeLines(fetched_sequences, output_file)

  # Check if sequences were fetched successfully
  if (length(search_result$ids) == 0) {
    cat("Warning: No sequences found for", gene, "in", organism, "\n")
  } else {
    cat("Fetched", length(search_result$ids), "sequences for", gene, "in",
organism, "\n")
  }
}

## ------ 2. Read Sequences from FASTA File into Data Frame -----

read_sequences <- function(fasta_file) {
  # Read the DNA sequences from the FASTA file
  string_set <- readDNAStringSet(fasta_file)

  # Create a data frame from the sequences
  df <- data.frame(
    Identifier = names(string_set),
    Nucleotide_Sequence = paste(string_set)
  )

  # Extract species name from the identifier
  df$Species_Name <- word(df$Identifier, 2L, 3L)

  # Select relevant columns for output
  df <- df[, c("Identifier", "Species_Name", "Nucleotide_Sequence")]

  # Check if the data frame is empty to ensure sequences are being imported
correctly
  if (nrow(df) == 0) {
    cat("Warning: No sequences read from", fasta_file, "\n")
  } else {
    cat("Read", nrow(df), "sequences from", fasta_file, "\n")
  }

  return(df)
}
```

```
## ------ 3. Exploratory Data Analysis (EDA) on Sequence Data -----

# Before data was filtered, EDA was conducted to determine what filtering steps
needed to be done.
# Iterations of filtering were done with multiple functions integrated within
one another. They were simplified and seperated into individual functions due
to errors that were occuring for manipulating between different data types.

# Function to get the lengths of sequences
get_sequence_lengths <- function(sequences) {
  sapply(sequences, nchar)
}

# Function to plot sequence lengths in histogram and boxplot
# Looking at sequence lengths in a visual form can give indications on the
limits that have to be implemented for sequences for alignment.
plot_sequence_lengths <- function(lengths, gene_name, hist_color="skyblue",
box_color="blue") {
  # Histogram of sequence lengths
  hist(lengths, breaks=30, main=paste(gene_name, "Sequence Lengths"),
       xlab="Sequence Length", col=hist_color)

  # Boxplot of sequence lengths
  boxplot(lengths, main=paste(gene_name, "Sequence Lengths"),
          ylab="Length", col=box_color)
}

# Function to check for duplicates in the sequences
# Duplicates need to be checked and removed if too many exist to ensure that a
bias does not occur for a specific species.
check_duplicates <- function(sequences) {
  unique_sequences <- unique(sequences)
  num_duplicates <- length(sequences) - length(unique_sequences)
  cat("Number of duplicate sequences: ", num_duplicates, "\n")

  # Return unique sequences
  return(unique_sequences)
}

## ------ 4. Execute Workflow with Data Exploration Before Filtering ------
# Functions that were created in 1-3 will be called and implemented for the
Genus: Heliconius (Butterfly) and Genes: COI, EF1a

# Fetch and Read COI Sequences
fetch_sequences("COI", "Heliconius", output_file = "COI_sequences.fasta")
coi_data <- read_sequences("COI_sequences.fasta")

# EDA for COI Sequences
COI_lengths <- get_sequence_lengths(coi_data$Nucleotide_Sequence) # Checking
sequence length for identifying possible outliers.
plot_sequence_lengths(COI_lengths, "COI", hist_color="skyblue",
box_color="blue") # Visualizing length distribution for identidying next steps
in filtering.

cat("COI Sequence Length Summary:\n")
```

```
print(summary(COI_lengths)) # Q1 and Q3 sequence lengths will be used to filter
sequences.

COI_unique <- check_duplicates(coi_data$Nucleotide_Sequence) # Identify and
count duplicates to remove in filtering, the count of duplicates also gives
indication of sequence diversity in dataset.
table(coi_data$Species_Name) # Table displays species distribution for
identifying if data has been retrieved correctly and species diversity.

# Some general observations can be made from the EDA. The sequence lengths vary
incredibly. Sequences from 0-15000 exist.
# The higher sequences at 15k may be an entire genome and should be filtered
accordingly. They are seen as outliers in the boxplot.
# The boxplot should also be made again with the removal of the outliers for a
better understanding of the data.
# Due to the wide spread of data, the summary is heavily skewed with the
minimum at 252 and maximum at 15k bp in the sequence length.
# The median and mean also vary heavily due to the influence of the 15k on the
mean.
# The gap between Q1 and Q3 is larger in COI than EF1a for the same reason.
# Total of 84 duplicates were found and should be removed.

# Fetch and Read EF1α Sequences
# The same steps as above as done for the EF1a gene.
fetch_sequences("EF1a", "Heliconius", output_file = "EF1a_sequences.fasta")
ef1a_data <- read_sequences("EF1a_sequences.fasta")

# EDA for EF1α Sequences
EF1a_lengths <- get_sequence_lengths(ef1a_data$Nucleotide_Sequence)
plot_sequence_lengths(EF1a_lengths, "EF1α", hist_color="lightgreen",
box_color="green")
cat("EF1α Sequence Length Summary:\n")
print(summary(EF1a_lengths))
EF1a_unique <- check_duplicates(ef1a_data$Nucleotide_Sequence)
table(ef1a_data$Species_Name)

# Some general observations can be made from this EDA. The sequence length does
not vary as much in EF1a as it does in COI and the overall lengths are
clustered closer together. It will be simpler to filter since there are less
outliers.
# This can be seen in the boxplot as well. The median is relatively towards the
lower half of the dataset, so instead of taking a value of nucleotides +/- for
the sequence, taking Q1-Q3 might be better for a more balanced set.
# The mean and median are relatively far apart with 937 and 876, respectively.
This can be indicative of the large peak of sequence frequency at 900 sequence
length.
# A total of 41 duplicates were found and will be removed to remain consistent.
# When looking at species diversity, 1 instance of Neruda metharme was
retrieved. This will be removed when filtering is done. It is not relevent in
this dataset.

## ------ 5. Filtering Based on EDA Insights + Quality Control ------

# Function to clean sequences, handling missing data, gaps, and duplicates
clean_sequences <- function(df, max_n_percentage = 5) {
  # 5% of N was taken as a standard value due to the N's seen in EDA, a
stronger constraint can be placed if higher values of N's were seen.
```

```
  # Filter out rows where Species_Name does not start with "Heliconius" as seen
in EDA
  df <- df %>% filter(str_starts(Species_Name, "Heliconius"))

  # Calculate metrics for N-count, missing data, sequence length, and N
percentage
  df <- df %>%
    mutate(
      N_count = ifelse(is.na(Nucleotide_Sequence), 0,
str_count(Nucleotide_Sequence, "N")),
      Missing_data = str_count(Nucleotide_Sequence, '-'), # Determine gaps to
be removed
      Length = nchar(Nucleotide_Sequence), # Determine length of sequences
      N_count_percentage = (N_count / Length) * 100 # Determine N % for removal
of sequences
    )

  # Remove the sequence with the largest percentage of N's
  if (nrow(df) > 0) {
    df <- df[-which.max(df$N_count_percentage), ]
  }

  # Filter out sequences with abnormal length (>15,000 bp)
  # This value is being used due to the outlier in COI sequences of 15k
  df <- df %>% filter(Length <= 15000)

  # Filter out sequences with high N content
  df <- df %>% filter(N_count_percentage <= max_n_percentage)

  # Remove duplicates based on Nucleotide_Sequence
  df <- df[!duplicated(df$Nucleotide_Sequence), ]

  return(df)
}

# Function to filter sequences by length within a specific range
# The range will differ depending on Q1 - Q3 values and should be different for
each gene.
filter_by_length <- function(df, min_length, max_length) {
  df <- df %>% filter(Length >= min_length & Length <= max_length)
  return(df)
}

# Function to process and clean sequences for a specific gene
# This function calls all the individual functions above for efficient
implementation.
process_gene_sequences <- function(gene, organism, fasta_file, gene_name,
min_length, max_length) {
  fetch_sequences(gene, organism, output_file = fasta_file)
  df <- read_sequences(fasta_file)
  df <- clean_sequences(df)
  df <- filter_by_length(df, min_length, max_length)

  # Add a column for gene name to differentiate which gene the sequence/row is
from
  df$Gene <- gene_name
```

```
  return(df)
}

# Process COI sequences
df_COI <- process_gene_sequences("COI", "Heliconius", "Heliconius_COI.fasta",
"COI", 376, 618)

# Process EF1a sequences
df_EF1a <- process_gene_sequences("EF1a", "Heliconius",
"Heliconius_EF1a.fasta", "EF1a", 798, 1189)

# Combine the cleaned data
df_Heliconius <- bind_rows(df_COI, df_EF1a)


## ------ 6. Visualizations -------

# Two visualizations were made for Code Part 1.
# Visualization 1 is for EDA to display the species diversity + frequency of
sequences based on the gene.
# The species with highest sequence counts, Heliconius melpomene and Heliconius
numata have significantly higher COI gene sequences compared to other species.
# COI gene generally has more sequences available across species, while EF1α is
less represented.
# Species like Heliconius hecale and Heliconius erato show a more balanced
distribution between COI and EF1α sequences.
# Species such as Heliconius ricini and Heliconius sara have very few available
sequences for both genes.
# The plot shows varying levels of genetic data availability, which can act as
a future research guide on species.

# Function to plot species frequencies as a stacked bar chart
plot_species_frequencies <- function(df_coi, df_ef1a) {
  # Combine the data frames and add a column to indicate the gene
  combined_df <- bind_rows(
    df_coi %>% mutate(Gene = "COI"),
    df_ef1a %>% mutate(Gene = "EF1α")
  )

  # Count the number of sequences for each species, grouped by gene
  species_counts <- combined_df %>%
    group_by(Species_Name, Gene) %>%
    summarise(Count = n()) %>%
    ungroup()

  # Create the stacked bar plot
  ggplot(species_counts, aes(x = Species_Name, y = Count, fill = Gene)) +
    geom_bar(stat = "identity", position = "stack") + # Use "stack" for stacked
bars
    labs(title = "Frequency of Sequences by Species and Gene",
         x = "Species",
         y = "Frequency of Sequences") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) + # Rotate x-axis
labels for readability
```

```
    scale_fill_manual(values = c("COI" = "lightblue", "EF1α" = "lightgreen")) #
Assign colors to genes
}

# Call the function to plot species frequencies for COI and EF1α as a stacked
bar chart
plot_species_frequencies(df_COI, df_EF1a)
```

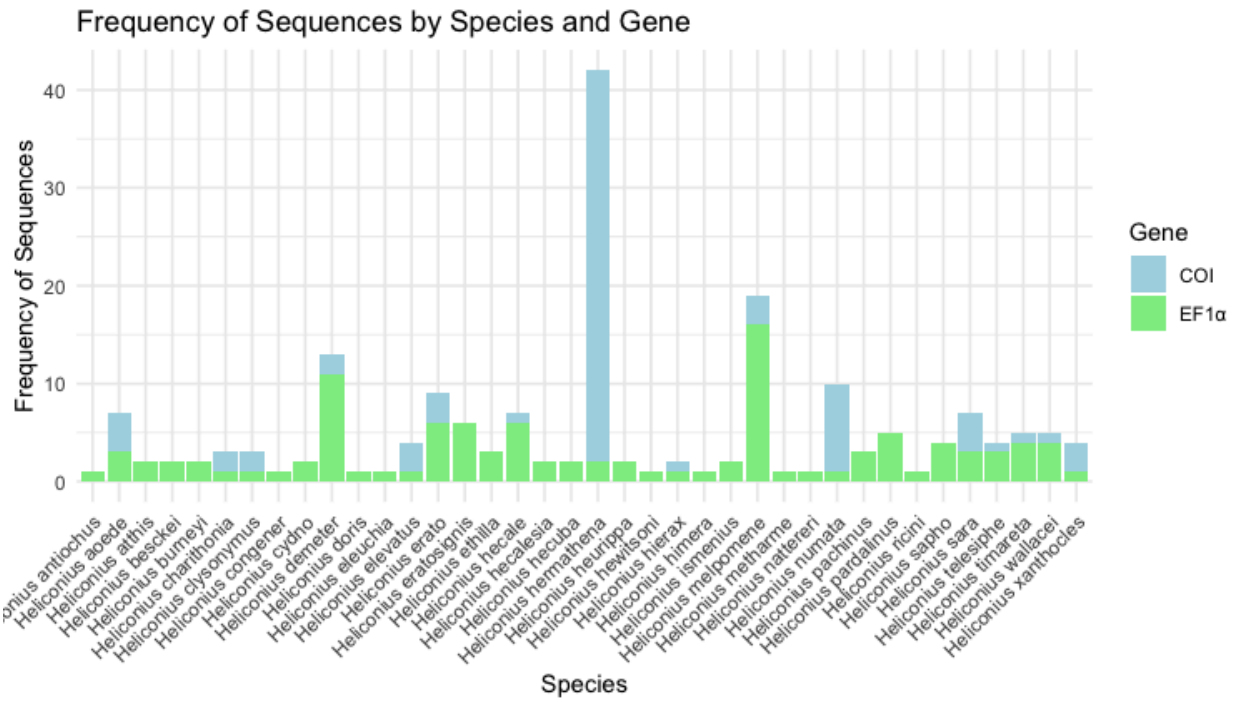


Figure 1: This figure displays the frequency of sequences by both species of Heliconius and by gene (COI and EF1a).

```
# Visualization 2 focuses on the quality control after filtering.
# EF1α gene has longer sequences, with a median length close to 1,000 base
pairs and a range extending up to 1,200 base pairs.
# The COI gene shows shorter sequences, with a median length around 500 base
pairs and a range extending from approximately 400 to 600 base pairs.
# EF1α sequences exhibit greater variability in length compared to COI
sequences, which are more tightly clustered. This will effect the balance of
the dataset.
# After filtering, there is a clear distinction between the lengths of the two
genes, with EF1α being substantially longer than COI.
# This difference in sequence length between mitochondrial (COI) and nuclear
(EF1α) genes reflects inherent differences in their genetic structures and may
influence downstream analysis, specifically in terms of pollination.

# Function to visualize sequence lengths with a boxplot for QC
plot_length_boxplot <- function(df) {
  # Create a boxplot using ggplot
  ggplot(df, aes(x = Gene, y = Length, fill = Gene)) +
    geom_boxplot(outlier.color = "red", outlier.size = 5) +
    labs(title = "Length Distribution by Gene after Filtering (Quality
Control)", x = "Gene", y = "Sequence Length") +
```

```
    scale_fill_manual(values = c("COI" = "lightblue", "EF1a" = "lightgreen")) +
# Assign colors
    theme_minimal() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          legend.title = element_blank()) + # Remove legend title
    guides(fill = guide_legend(title = "Gene")) # Add a title to the legend
}

# Prepare a combined data frame for boxplot visualization
df_COI$Gene <- "COI"
df_EF1a$Gene <- "EF1a"
combined_df <- rbind(df_COI, df_EF1a)

# Plot boxplot for quality control
plot_length_boxplot(combined_df)
```

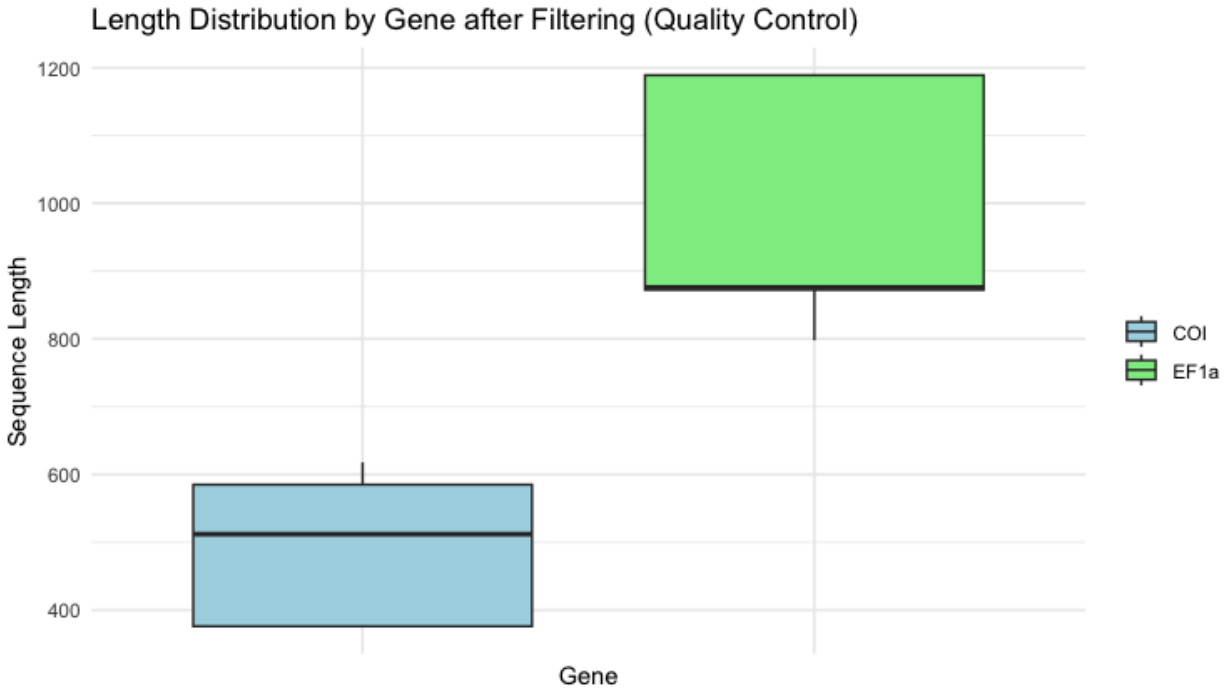


Figure 2: This figure displays a boxplot of the sequence length for the COI and EF1a gene after filtering.

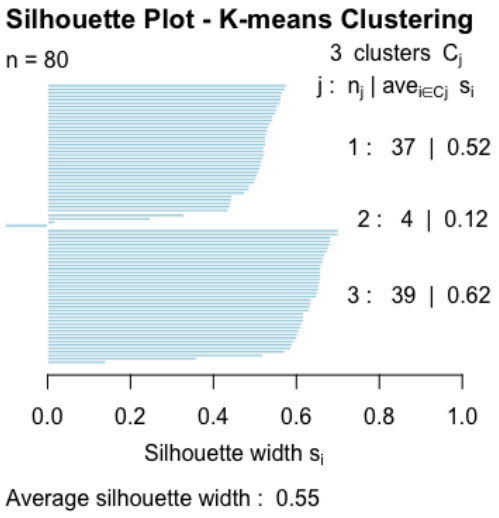
CODE SECTION 2: ANALYSIS TO ADDRESS QUESTION

```
#### CODE SECTION 2: ANALYSIS TO ADDRESS QUESTION ------

## ------ 1. Compute k-mer Profiles and Alignment -----

# Function to compute k-mer profiles
# Compute k-mer profiles using the oligonucleotideFrequency function that
counts the occurrences of each k-mer (subsequence of length k) across all
sequences, to understand the composition and variability of the sequences.
kmer_profile <- function(sequences, k) {
  # Convert the character vector of sequences to DNAStringSet object
```

```
  dna_sequences <- DNAStringSet(sequences)
  kmer_counts <- oligonucleotideFrequency(dna_sequences, width = k)

  return(kmer_counts)  # Return the k-mer counts as a matrix.
}

# Apply k-mer profile extraction for COI and EF1a genes with k = 3.
# k = 3 is commonly used to capture key sequence motifs while balancing the
trade-off between specificity and computational efficiency.
# If time permitted, next steps would consist of adjusting and trying different
k-mer frequencies to see which works best for this dataset.
kmer_COI <- kmer_profile(df_COI$Nucleotide_Sequence, k = 3)
kmer_EF1a <- kmer_profile(df_EF1a$Nucleotide_Sequence, k = 3)

## ------ 2. K-means Clustering ------

set.seed(123)  # Set seed for reproducibility of results in random processes.

# Perform k-means clustering on k-mer profiles for the COI sequence
# K-means clustering groups sequences into k distinct groups based on the
similarity of their k-mer profiles.
kmeans_COI <- kmeans(kmer_COI, centers = 3)

# Perform k-means clustering on k-mer profiles for the EF1a sequences
kmeans_EF1a <- kmeans(kmer_EF1a, centers = 3)

## ------ 3. Matrix + Hierachical Clustering ------

# Compute distance matrices for the k-mer profiles of COI and EF1a
# dist function calculates a distance matrix, which quantifies how dissimilar
the sequences are based on their k-mer composition
dist_COI <- dist(kmer_COI)  # Calculate distance for COI sequences
dist_EF1a <- dist(kmer_EF1a)  # Calculate distance for EF1a sequences

# Perform hierarchical clustering using the average linkage method (UPGMA)
# Reasons for using UPGMA:
# It uses a balance between compact clusters (complete linkage) and chaining
effects (single linkage)
# Assumes equal contributions from all data points, which is appropriate for
genetic sequence clustering
# Produces biologically meaningful clusters, which is helpful for DNA sequence
analysis
hc_COI <- hclust(dist_COI, method = "average")  # Hierarchical clustering for
COI.
hc_EF1a <- hclust(dist_EF1a, method = "average")  # Hierarchical clustering for
EF1a.

## ------ 4. Create Dendrogram -----

# Dendrograms can be used to visually see the effects of the clustering. A
function was created to do this, and called on the data after.

plot_dendrogram <- function(hc_result, gene_name) {
  # Convert hierarchical clustering result to a dendrogram
  dendro_data <- dendro_data(hc_result)

  # Create the dendrogram plot using ggplot2
```

```
  ggplot(dendro_data$segments) +
    geom_segment(aes(x = x, y = y, xend = xend, yend = yend),
                 color = "steelblue", size = 0.5) +
    geom_text(data = dendro_data$labels, aes(x = x, y = y, label = label),
              size = 3, color = "black", vjust = -0.5) +
    labs(title = paste("Hierarchical Clustering -", gene_name),
         x = "Cluster", y = "Height") +
    theme_minimal() +
    theme(axis.text.x = element_blank(),  # Hide x-axis text
          axis.ticks.x = element_blank())  # Hide x-axis ticks
}

# Plot the dendrogram for COI gene
plot_dendrogram(hc_COI, "COI Gene")

# Plot the dendrogram for EF1a gene
plot_dendrogram(hc_EF1a, "EF1a Gene")

# The dendrograms for hierarchical clustering of COI and EF1α genes show
distinct patterns of clustering.
# In the COI gene, there are two dominant large clusters, with sub-clusters
indicating a finer differentiation among sequences.
# For EF1α, the clustering pattern is slightly more varied, with one large
dominant cluster and multiple smaller, tighter clusters.
# This does suggest that EF1α sequences exhibit more hierarchical variation
compared to COI, potentially reflecting differences in sequence diversity or
evolutionary pressure across these two genes.

## ------ 5. Evaluate using Silhouette Index -----

# The silhouette plots assess clustering quality for COI and EF1α gene
datasets, each divided into three clusters.

perform_clustering_analysis <- function(kmeans_result, hclust_result,
dist_matrix, num_clusters = 3) {

  # Check if the inputs are valid
  # This step was added during the iteration process of the functions.
  if (is.null(kmeans_result) || is.null(hclust_result) || is.null(dist_matrix))
{
    stop("kmeans_result, hclust_result, and dist_matrix must not be NULL.")
  }

  # Compute silhouette scores for k-means
  sil_kmeans <- silhouette(kmeans_result$cluster, dist_matrix)

  # Compute silhouette scores for hierarchical clustering
  hc_clusters <- cutree(hclust_result, k = num_clusters)
  sil_hclust <- silhouette(hc_clusters, dist_matrix)

  # Set up the plot layout: 1 row, 2 columns
  par(mfrow = c(1, 2))

  # Plot silhouette for k-means
  plot(sil_kmeans, main = "Silhouette Plot - K-means Clustering", col =
"lightblue")
```

```
  # Plot silhouette for hierarchical clustering
  plot(sil_hclust, main = "Silhouette Plot - Hierarchical Clustering", col =
"lightgreen")

  # Compute average silhouette scores and handle NA values
  avg_sil_kmeans <- mean(sil_kmeans[, 3], na.rm = TRUE)
  avg_sil_hclust <- mean(sil_hclust[, 3], na.rm = TRUE)

  # Return results as a list
  return(list(avg_sil_kmeans = avg_sil_kmeans,
              avg_sil_hclust = avg_sil_hclust))
}

# Perform clustering analysis for COI
results_COI <- perform_clustering_analysis(kmeans_COI, hc_COI, dist_COI,
num_clusters = 3)

# Perform clustering analysis for EF1a
results_EF1a <- perform_clustering_analysis(kmeans_EF1a, hc_EF1a, dist_EF1a,
num_clusters = 3)

# The silhouette plots for the COI gene (n = 80) and EF1α gene (n = 110)
compare the clustering quality between K-means and hierarchical clustering.
# For the COI gene, K-means clustering results in an average silhouette width
of 0.55, with one poorly defined cluster (Cluster 2, silhouette width = 0.12).
# In contrast, hierarchical clustering for the COI gene shows a higher average
silhouette width of 0.63, indicating better cluster separation overall,
although one small cluster (Cluster 3, n = 1) is not well defined.

# For the EF1α gene, both K-means and hierarchical clustering yield an average
silhouette width of 0.54, suggesting similar clustering quality across methods.
# K-means shows good clustering for Cluster 1 (0.78) but poor results for
Cluster 3 (0.34), while hierarchical clustering provides better results for
Clusters 2 and 3 (0.76 each).
# Overall, hierarchical clustering appears to perform slightly better for COI,
whereas the methods show comparable performance for EF1α.
```



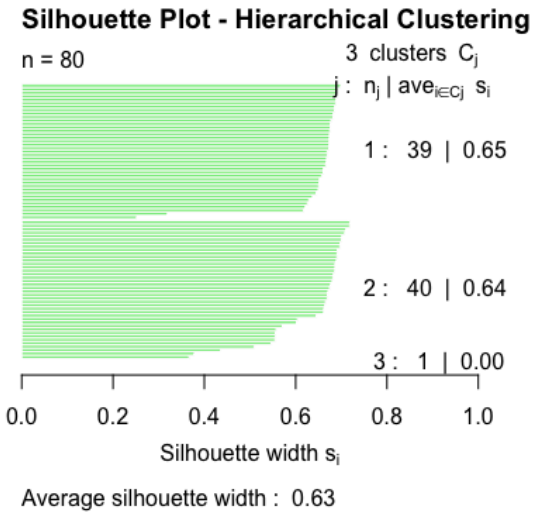


Figure 3 : This figure displays the Silhoutte plot for both the k-means and hierarchical clustering for the COI gene.

```
## ------ 6. PCA to Evaluate Clustering -----

# Perform PCA for COI
# Use principal component analysis (PCA) to reduce dimensionality of the
distance matrix for the COI gene
pca_result_coi <- prcomp(dist_COI, center = TRUE, scale. = TRUE)  # Perform PCA
with centering and scaling
# Create a data frame to hold PCA results along with cluster assignments from
k-means
pca_data_coi <- data.frame(pca_result_coi$x, cluster =
as.factor(kmeans_COI$cluster))

# Perform PCA for EF1a
# Repeat the PCA process for the EF1a gene distance matrix
pca_result_ef1a <- prcomp(dist_EF1a, center = TRUE, scale. = TRUE)  # Perform
PCA with centering and scaling
# Create a data frame to hold PCA results for EF1a along with cluster
assignments from k-means
pca_data_ef1a <- data.frame(pca_result_ef1a$x, cluster =
as.factor(kmeans_EF1a$cluster))

# Create a data frame for the PCA results of the COI gene
pca_COI_data <- data.frame(
  PC1 = pca_result_coi$x[, 1],  # Extract first principal component
  PC2 = pca_result_coi$x[, 2],  # Extract second principal component
  Cluster = kmeans_COI$cluster,  # Include cluster information from k-means
  Gene = "COI",  # Specify the gene for identification
  Species = df_COI$Species_Name  # Include species names from the original data
frame
)

# Create a data frame for the PCA results of the EF1a gene
pca_EF1a_data <- data.frame(
  PC1 = pca_result_ef1a$x[, 1],
  PC2 = pca_result_ef1a$x[, 2],
  Cluster = kmeans_EF1a$cluster,
  Gene = "EF1a",
  Species = df_EF1a$Species_Name
)

# Function to plot PCA with gene selection and optional species labels
plot_pca_genes <- function(show_COI = TRUE, show_EF1a = TRUE, show_labels =
FALSE) {
  # Combine PCA data for selected genes
  combined_pca_data <- rbind(
    if (show_COI) pca_COI_data else NULL,  # Include COI data if specified
    if (show_EF1a) pca_EF1a_data else NULL  # Include EF1a data if specified
  )

  # Ensure there's data to plot; stop execution if no data is selected
  if (nrow(combined_pca_data) == 0) {
    stop("No data selected. Please enable at least one gene.")
```

```
  }

  # Create the PCA plot with points colored by cluster and shaped by gene
  p <- ggplot(combined_pca_data, aes(x = PC1, y = PC2, color =
as.factor(Cluster), shape = Gene)) +
    geom_point(size = 3) +  # Plot points with size
    theme_minimal() +  # Use a minimal theme for the plot
    labs(
      title = "PCA of Clusters with Species Labels for Selected Genes",  #
Title of the plot
      x = "Principal Component 1",  # Label for x-axis
      y = "Principal Component 2",  # Label for y-axis
      color = "Cluster",  # Legend title for colors
      shape = "Gene"  # Legend title for shapes
    ) +
    scale_color_brewer(palette = "Set1")  # Use a color palette from
RColorBrewer

  # Add species labels to the plot if specified
  if (show_labels) {
    p <- p + geom_text(aes(label = Species), hjust = 0.5, vjust = -0.5, size =
3, check_overlap = TRUE)
  }

  # Print the plot
  print(p)
}

# Plot with labels for both genes and species names
plot_pca_genes(show_COI = TRUE, show_EF1a = TRUE, show_labels = TRUE)

# Plot with labels for only EF1a
plot_pca_genes(show_COI = FALSE, show_EF1a = TRUE, show_labels = TRUE)

# Plot without labels for clearer cluster visualization
plot_pca_genes(show_COI = TRUE, show_EF1a = TRUE, show_labels = FALSE)
```

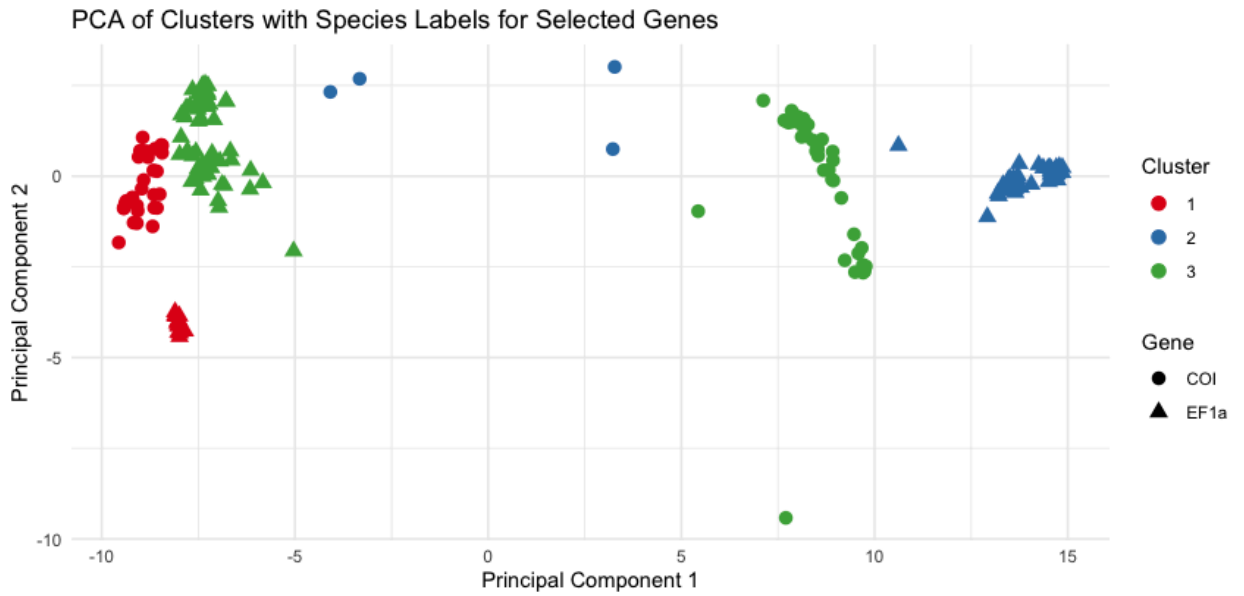


Figure 4: This figure displays a PCA for both COI and EF1a displaying the clustering patterns for the dataset.

DISCUSSION AND CONCLUSION

The results reveal distinct clustering patterns between the COI and EF1α genes, providing insight into their different evolutionary dynamics. The tight clustering of COI sequences, particularly in species like Heliconius melpomene, supports the hypothesis that mitochondrial DNA evolves more rapidly due to maternal inheritance and lack of recombination, leading to faster divergence (Rubinoff & Holland, 2005). This suggests that COI could play a role in species-specific adaptations, potentially influencing behaviors such as pollination. Alternatively, the more dispersed clustering of EF1α sequences in the PCA plot suggests slower evolutionary rates due to nuclear recombination, leading to more gradual divergence across species. The hierarchical clustering's higher silhouette score (0.63) compared to K-means indicates that hierarchical methods more accurately capture genetic structure, likely reflecting deeper evolutionary relationships (Wahlberg, Wheat, & Peña, 2009). These results suggest that different evolutionary pressures on mitochondrial and nuclear genes may influence ecological behaviors in distinct ways, offering insights into gene-environment interactions in Heliconius butterflies.

However, several limitations should be considered. The uneven sequence lengths between the genes, with EF1α being significantly longer, may have biased the clustering results by providing more phylogenetic information. Additionally, the sample sizes for some species were small, which may have limited the generalizability of the findings. As a next step, incorporating data from another genus closely related to Heliconius could provide a more comprehensive dataset, allowing for better generalization across taxa. Integrating ecological data on pollination behavior would also be valuable, as it could help to link genetic clustering with specific ecological traits.

From a bioinformatics perspective, implementing more advanced clustering algorithms—such as model-based clustering or network-based approaches (Tommasi et al., 2021), could provide a more nuanced view of the genetic relationships between species. Further, leveraging machine learning methods to predict ecological traits based on genetic data could open new avenues for understanding how evolutionary pressures shape behavior in butterflies.

ACKNOWLEDGMENTS

I want to express my gratitude to Brittany MacIntyre for helping me gain a clearer understanding of the various paths I can pursue for my project. Her suggestions for workarounds in the code and the time she dedicated to answering my questions were invaluable. I'd also like to thankAvery Murphy for her steadfast support while I was troubleshooting the code. Additionally, I appreciate Rebecca Choi and Frances Bonafe for their helpful suggestions during the troubleshooting process.

REFERENCES

Heliconius. (n.d.). NCBI. Retrieved October 25, 2024, from https://www.ncbi.nlm.nih.gov/datasets/taxonomy/33416/

Jiggins, C. D., & Lamas, G. (2016). *The ecology and evolution of heliconius butterflies*. https://academic.oup.com/book/26557

McMillan, W. O., Livraghi, L., Concha, C., & Hanly, J. J. (n.d.). Frontiers. *Frontiers in Ecology and Evolution*, *8*. https://doi.org/10.3389/fevo.2020.00221

Rubinoff, D., & Holland, B. S. (2005). Between Two Extremes: Mitochondrial DNA is neither the Panacea nor the Nemesis of Phylogenetic and Taxonomic Inference. *Systematic Biology*, *54*(6), 952–961. https://doi.org/10.1080/10635150500234674

Tommasi, N., Ferrari, A., Labra, M., Galimberti, A., & Biella, P. (2021). Harnessing the power of metabarcoding in the ecological interpretation of plant-pollinator DNA data: Strategies and consequences of filtering approaches. *Diversity*, *13*(9). https://doi.org/10.3390/d13090437

Van Belleghem, S. M., Lewis, J. J., Rivera, E. S., & Papa, R. (2021). Heliconius butterflies: A window into the evolution and development of diversity. *Current Opinion in Genetics & Development*, *69*, 72–81. https://doi.org/10.1016/j.gde.2021.01.010

Zuccarelli, E. "Jay." (2021, January 31). Performance metrics in ml-part 3: Clustering. *Towards Data Science*. https://towardsdatascience.com/performance-metrics-in-machine-learning-part-3-clustering-d69550662dc6