Assessment 4_SLE777_R Project

Ameeta

2025-10-05

Gene expression Step 1: Downloading and reading the geneexpression file

The link for raw file were copied from the github and then downloaded in R-markdown using te download.file function The downloaded file was read using the read.table function and the head function was used display the first 6 rows, showing 6 gene identifiers. “‘{r} # download the gene expression file

download.file(“https://raw.githubusercontent.com/ghazkha/Assessment4/refs/heads/main/gene_expression.tsv”, destfile = “geneexpression.tsv”) # destfile = destination file path

Read the downloaded file

gene_data <- read.table(“geneexpression.tsv”, # path to the file header=TRUE, # indicates first row of the file contains column names sep = “ ̂”, # ̂is the standard for tsv files row.names = 1, #indicates that first colum contains row names stringsAsFactors = FALSE) # keep as plain character strings

display the first 6 genes of the file

head(gene_data, 6) # 6 inidcates number of rows to be displayed

```
Step 2: Generating a new column with mean value of other columns

A new column called meanexpression, which contain the mean value of other columns was created in the tal
```r}
gene_data$meanexpression <- rowMeans(gene_data) # rowMeans calculate means across all columns for each r
gene_data[1:6, c(1, ncol(gene_data))] # 1:6 selects forst 6 genes of the data
                                      # c(1, ncol(gene_data) selects the first and the last column
```

Step 3: Listing the 10 genes with the highest mean expression

The meanexpression in gene_data were ordered in the descending order using order (-)function. 10 genes with highest meanexpression were displayed using a drop argument.

```
{r} # order the "meanexpression" of the gene-data in descending order and save in gene_data-sorted
file gene_data_sorted <- gene_data[order(-gene_data$meanexpression), ] # order(-gene_data$meanofcolumns)
sorts the meanexpression in descending order # show 10 genes with the highest mea expression
values  gene_data_sorted[1:10, "meanexpression", drop = FALSE] # drop =FALSE ensures
datas are expressed in data frame format and not vector
```
Step 4. Determining the number of genes with a mean <10

To determine the genes with the meanexpression less than 10, the logical condition was created to line checks for every gene in the datasetiwith meanexpression value less than 10 and the sum() function was used to sum the logical vectors. “‘{r} # create logical vectors for genes with meanexpression <10 gene_data_mean_10 <- gene_data_sorted$meanexpression <10
```

count the total number of genes with mean <10

sum(gene_data_mean_10)

```
Step 5: Making a histogram plot of the mean values

A histogram was generated using hist() function to represent the distribution of the mean expression val
The majority of the genes have a very low mean expression as indicated by the tallest bin near zero. Hou
```

```{r}
# Step 6: Histogram of mean values of gene expression
data(gene_data)
hist(gene_data$meanexpression, # data to be plotted
     xlab="Mean expression", # label for the x-axis
     main="Mean value of gene expression") # Main title of the histogram
```

GROWTH DATA INTERPRETATION

Step 6: Importing the csv file into an R object

The csv raw file for growth data was downloaded using download.file function and the file was imported into R as a data frame using the read.csv() function. The colnames() function was used to display all the column names of the data set.

"'{r} # download the growth data file

download.file("https://raw.githubusercontent.com/ghazkha/Assessment4/refs/heads/main/growth_data.csv", destfile = "growthdata.csv")

Read file

growth_data <- read.csv("growthdata.csv") colnames(growth_data)

```
Step 7: Calculating the mean and standard deviation of tree circumference at the start and end of the s

The mean and standard deviation was created using the mean() and sd() function. The data.frame was creat
```

```{r}
# 1. Mean and sd for Northeast site at the start (Circumf_2005_cm)

meannortheast_start<- mean(growth_data$Circumf_2005_cm[growth_data$Site== "northeast"]) # growth_data$Si

sdnortheast_start <- sd(growth_data$Circumf_2005_cm[growth_data$Site== "northeast"])

# 2. Mean and sd for Northeast site at the start (Circumf_2020_cm)

meannortheast_end <- mean(growth_data$Circumf_2020_cm[growth_data$Site== "northeast"])

sdnortheast_end <- sd(growth_data$Circumf_2020_cm[growth_data$Site== "northeast"])

# 3. Mean for Southwest site at the start (Circumf_2005_cm)

meansouthwest_start<- mean(growth_data$Circumf_2005_cm[growth_data$Site== "southwest"])

sdsouthwest_start <- sd(growth_data$Circumf_2005_cm[growth_data$Site== "southwest"])


# 4. Mean and sd for Southwest site at the end (Circumf_2020_cm)
```

```
meansouthwest_end<- mean(growth_data$Circumf_2020_cm[growth_data$Site== "southwest"])

sdsouthwest_end <- sd(growth_data$Circumf_2020_cm[growth_data$Site== "southwest"])

# create summary table

summary_table <- data.frame(
  Site = c("Northeast", "Northeast", "Southwest", "Southwest"),
  Year = c("2005 (Start)", "2020 (End)", "2005 (Start)", "2020 (End)"),
  Mean = c(meannortheast_start, meannortheast_end, meansouthwest_start, meansouthwest_end),
  SD = c(sdnortheast_start, sdnortheast_end, sdsouthwest_start, sdsouthwest_end)
)
summary_table
```

step 8: Making a box plot for the circumference at the start and end of the study at both sites.

To create Boxplots, the datas were splitted into Northeast and Southwest. Then, the boxplot() function was used to display the tree circumferences for both sites at both time points.The boxplots were placed side by side by entering data sets for boths sites in the same boxplot() function. For both northeast and southwest, the tree circumference increased substantially from 2005 to 2020, indicating significant tree growth over period of time.

"'{r} # Splitting of data into Northeast and Southwest

northeast <- growth_data[growth_data$Site == "northeast",]southwest < −$growth_data[growth_data$Site == "southwest",]

Creating boxplots for two different sites at the start and end of the study periods

boxplot(northeast$Circumf_2005_cm, northeastCircumf_2020_cm, southwest$Circumf_2005_cm, southwestCircumf_2020_cm, names = c("NE Start","NE End", "SW Start", "SW End"), # Provides the names to each boxplot col = c("red", "blue", "red", "blue"), # provides red color to the boxplot at the start and blue color to the boxplot for the end for both the study sites ylab = "Tree Circumference (cm)", xlab = "Sites", main = "Tree Circumference at Start and End by Site")

```
Step 9: Calculating the mean growth over the last 10 years at each site.

The 10 year growth values were extracted by finding difference in the circumference between 2020 and 20
```
```{r}
# calculate growth data from 2010 to 2020

growth_data$ growth_10_years <- (growth_data$Circumf_2020_cm - growth_data$Circumf_2010_cm)

# Extract 10-year growth values for northsite
North_east_growth_data <- (growth_data$growth_10_years[growth_data$Site == "northeast"])

# calculate mean for 10-year growth values of northwest
North_east_mean_growth_data <- mean(North_east_growth_data)
North_east_mean_growth_data

# Extract 10-year growth values for southwest
South_west_growth_data <- (growth_data$growth_10_years[growth_data$Site == "southwest"])

# calculating mean for 10-year growth values of southwest
```

```
Southwest_mean_growth_data <- mean (South_west_growth_data)
Southwest_mean_growth_data
```

Step 10: Using the t.test to estimate the p-value

The p-value for the two sites were determined using t-test function to compare the 10 years growth between two sites.The study observed higher mean 10-year growth at the northeast site compared to soutwest site but the growth difference was not statistically significant at 5% significance level ```{r}
t.test(North_east_growth_data,South_west_growth_data)```

PART 2: Examining the biological sequence diversity

step 1: Downloading and counting CDS in the Saprospirale and ecoli sequences

The FASTA files were downloaded from the ENSEMBL website using the download.file() function and gunzip() function was used to uncompress the file. The read.fasta() function loaded the sequence into R as lists and the length() function was applied to determine the total number of CDS for each organism. The table was created using data.frame() function The Saprospirales contain slightly higher number of coding sequences (4527) than E.coli (4239). "'{r} suppressPackageStartupMessages({ library("seqinr") # is a package designed to process and analyse sequence data. library("R.utils") # general utilities like zip and unzip }) # loading e.coli and Saprospirales data library("R.utils")

Download FASTA file

URL="https://ftp.ensemblgenomes.ebi.ac.uk/pub/bacteria/release-62/fasta/bacteria_58_collection/saprospirales_bacterium_gca_003448025/cds/Saprospirales_bacterium_gca_003448025.ASM344802v1.cds.all.fa.gz" download.file(URL,destfile="saprospirales_cds.fa.gz")

URL="http://ftp.ensemblgenomes.org/pub/bacteria/release-53/fasta/bacteria_0_collection/escherichia_coli_str_k_12_substr_mg1655_gca_000005845/cds/Escherichia_coli_str_k_12_substr_mg1655_gca_000005845.ASM584v2.cds.all.fa.gz" download.file(URL,destfile="ecoli_cds.fa.gz")

Uncompress the FASTA files for e.coli and saprospirales

gunzip("ecoli_cds.fa.gz", overwrite = TRUE) # overwrite = TRUE tells R to replace the uncompressed .fa file if it already exist in the working directory

gunzip("saprospirales_cds.fa.gz", overwrite = TRUE)

Read the FASTA sequences

library("seqinr") e.coli_cds <- seqinr::read.fasta("ecoli_cds.fa") # read.fasta is used to read the FASTA file

saprospirales_cds <- seqinr::read.fasta("saprospirales_cds.fa")

Count the number of coding sequences

e.coli_number <- length(e.coli_cds)

saprospirales_num <- length(saprospirales_cds)

create table

cds_table <- data.frame(Bacteria = c("E.coli", "Saprospirales"), CDS_count = c(e.coli_number, saprospirales_num)) cds_table

```
Step 2: Determing and comparing the total coding DNA between the two organisms.

To determine the total length of the the sequences in both organisms, the length of each coding sequenc

It was observed that the Saprospirales has greater total coding DNA (4200321) than that of E.coli (3978

```{r}

# Calculate the CDS length for e.coli
e.coli_cds_length<- as.numeric(summary(e.coli_cds)[,1]) # 1 select the first column of the matrix

# Calculate the CDS length for Saprospirales
saprospirales_cds_length <- as.numeric(summary(saprospirales_cds)[,1])

# sum total coding DNA of E.coli
e.coli_total_cds_length <- sum(e.coli_cds_length)

#Sum total coding DNA of Saprospirales
saprospirales_total_cds_length <- sum(saprospirales_cds_length)

# create table for the total coding DNA for both organisms
cds_table <- data.frame(
  Bacteria = c("E.coli", "Saprospirales"),
  CDS_count = c(e.coli_number, saprospirales_num),
  total_coding_DNA = c(e.coli_total_cds_length, saprospirales_total_cds_length)
)
cds_table
```

Step 3: Calculating and making boxplot of CDS length

The length of each coding sequences was extracted from the respective organisms' cds using summary() function and the first column of the summary was converted to the numeric value using as.numeric() function. Then the boxplot() was used to display the distribution of cds lengths. The mean() and median() function was used to summarize the central tendency of CDS lengths and table was created to display the result. The mean coding sequence length of E.coli is 938.5534 and median is 831, indicating it contains higher cds length than with moderately long E.coli genes compared to Saprospirales with mean value of 927.8376 and the median is 690.

"'{r}

Calculate the CDS length

e.coli_cds_length<- as.numeric(summary(e.coli_cds)[,1]) # 1 select the first column of the matrix

saprospirales_cds_length <- as.numeric(summary(saprospirales_cds)[,1])

Generate box plot for E.coli and Saprospirales

boxplot(list(E.coli = e.coli_cds_length, Saprospirales = saprospirales_cds_length), col = c("red", "blue"), ylab = "CDS Length (bp)", main = "Distribution of Coding Sequence Lengths")

Calculate mean and median of E.coli

e.coli_mean_cds_length <- mean(e.coli_cds_length) e.coli_median_cds_length <- median(e.coli_cds_length)
```

Calculate mean and median of Saprospirales

saprospirales_mean_cds_length <- mean(saprospirales_cds_length) saprospirales_median_cds_length <- median(saprospirales_cds_length)

Create a table

cds_table <- data.frame(Bacteria = c("E.coli", "Saprospirales"), CDS_count = c(e.coli_number, saprospirales_num), total_coding_DNA = c(e.coli_total_cds_length, saprospirales_total_cds_length), mean_cds_length = c(e.coli_mean_cds_length, saprospirales_mean_cds_length), median_cds_length = c(e.coli_median_cds_length, saprospirales_median_cds_length)) cds_table

Step 4: Calculating the frequency of DNA bases and aminoacids

The unlist() function was used to turn the list of CDS into a single long vectors of single characters a

The lapply was used to translate the dna to protein and unlist() was used to give a long aminoacid vecto

The higher frequency of adenine (A) was recorded higher in Saprospirales while E.coli recorded slightly

In both the organisms, Leucine (L) and alanine (A) were the most abundant amino acids , followed by glyc

```{r}
# Unlist the E.coli cds
e.coli_dna <- unlist(e.coli_cds)

# Calculate frequency of dna bases in total coding sequences for e.coli
e.coli_dna_freq <- count(e.coli_dna, 1)   #1 =word size (single nucleotide count)

# Unlist the Saprosipirales cds sequences
saprospirales_dna <- unlist(saprospirales_cds)

# Calculate the frequency of dna bases in total coding sequences for Saprospirales
saprospirales_dna_freq <- count(saprospirales_dna, 1)   # 1 = word size (single nucleotide count)

# Combine into a data frame for plotting
Dna_freq_df <- data.frame(
  Base = c("A","C","G","T"),
  E.coli = e.coli_dna_freq,
  Saprospirales = saprospirales_dna_freq
)
Dna_freq_df

# Generate bar plot for nucleotide frequency
barplot(
  height = rbind(as.numeric(Dna_freq_df$E.coli.Freq), as.numeric(Dna_freq_df$Saprospirales.Freq)),
  beside = TRUE,      # Place the bar for E.coli and Saprospirales side-by-side
  names.arg = Dna_freq_df$Base, # Tells R to represent nucleotide bases (AGTC) at the x-axis
  col = c("red", "blue"), # Tells R to give red color to E.coli and blue to saprospirales
  main = "Nucleotide Frequency in CDS",
  ylab = "Count",
  xlab="nucleotides"
)
legend("topright", legend = c("E. coli", "Saprospirales"), fill = c("red", "blue"))
```

```
# topright specify the position of the legend, legend =c("E. coli", "Saprospirales") are the labels of t

# Translate the sequences for e.coli
e.coli_prot <-lapply(e.coli_cds, translate) # lapply applies the translate function to each element of t

# Unlist aminoacids of E.coli
e.coli_prot_unlist <- unlist(e.coli_prot)

# Count aminoacids
aa_alphabet <- c("A","R","N","D","C","Q","E","G","H","I","L","K","M","F","P","S","T","W","Y","V")

# Calculate E.coli aminoacid frequency
e.coli_aa_freq <- count(e.coli_prot_unlist, wordsize=1, alphabet=aa_alphabet)
e.coli_aa_freq


# Translate the sequences for Saprospirales
saprospirales_prot <- lapply(saprospirales_cds, translate)

# Unlist aminoacids of Saprospirales
saprospirales_prot_unlist <- unlist(saprospirales_prot)

# Calculate the frequency of aa of Saprospirales
Saprospirales_aa_freq <- count(saprospirales_prot_unlist, wordsize=1, alphabet=aa_alphabet)
Saprospirales_aa_freq


# combine the aa frequency into data.frame for plotting
aa_freq_df <- data.frame(
  AA = aa_alphabet,
  E_coli = e.coli_aa_freq,
  Saprospirales = Saprospirales_aa_freq
)
aa_freq_df

# Generate bar plot for amino acid frequency
barplot(
  height = rbind(as.numeric(aa_freq_df$E_coli.Freq), as.numeric(aa_freq_df$Saprospirales.Freq)),
  beside = TRUE,
  names.arg = aa_freq_df$AA, # Tells R to represent aa texts at the x-axis
  col = c("red", "blue"),
  main = "Amino acid frequency in E.coli and Saprospirales",
  ylab = "Count",
  xlab = "Aminoacids"
)
legend("topright", legend = c("E. coli", "Saprospirales"), fill = c("red", "blue"))
```

Step 5: Qunaitifying the codon usage bias among all coding sequences.

The uco() function was applied to count all codons (3-base sequences) in the DNA sequences. The codons were sorted using order() function to ensure that the codon table is consistent for both organisms and table was created. The index="rscu" was employed to compute relative synonymous codon usage which is a measure of codon usage bias. Here, the RSCU > 1 indicates that codon is used more frequently than expected for the amino acid while RSCU <1 indicates that codon is used less frequently than expected. The data.frame was kept TRUE to convert the result into plotting.

The barchart was generated for codon usage bias using rbind() function.

As shown in the chart, the E.coli shows stronger peaks for certain codons, indicating higher codon usage bias while Saprospirales exhibit more even distributions across synonymous codons, suggesting weaker codon preference. Moreover, some of the codons which are less used in saprospirales and simlarly, the codons which are are more used in Saprospirales are comparatively less used in E.coli. "'{r} # Determining codon usage for e.coli e.coli_codon_usage <- uco(e.coli_dna) e.coli_codon_usage <- e.coli_codon_usage[order(names(e.coli_codon_usage))] # sort codons

Determining codon usage for saprospirales

saprospirales_codon_usage <- uco(saprospirales_dna) # uco() returns counts of each codon. saprospirales_codon_usage <- saprospirales_codon_usage[order(names(saprospirales_codon_usage))] # sort codons

Creating table for codon_usage for E.coli and Saprospirales

codons <- names(e.coli_codon_usage) bacteria_codon_table <- data.frame(Codon = codons, E.coli_count = as.numeric(e.coli_codon_usage), Saprospirales_count = as.numeric(saprospirales_codon_usage))

Calculation of RSCU values for E.coli

e.coli_codon_usage_bias <- uco(e.coli_dna, index="rscu", as.data.frame=TRUE)

Calculation of RSCU values for Saprospirales

saprospirales_codon_usage_bias <- uco(saprospirales_dna, index="rscu", as.data.frame=TRUE) e.coli_codon_usage_bias

generate barchart for codon usage bias for e.coli and saprospirales

RCSU_matrix <-rbind(E.coli = e.coli_codon_usage_bias$freq, $Saprospirales = saprospirales_codon_usage_bias$freq) barplot(RCSU_matrix, beside = TRUE, names.arg = e.coli_codon_usage_bias$AA, legend.text = TRUE, ylab = "codon usage bias (RSCU)", xlab = "Codons", main = "Codon usage frequency usage comparison") grid()

```
Step 6: Identifying over- and under- expressed k-mers of length 3-5

The k-mer frequencies of 3- 4- and 5- were calculated using count() function and the custom top k-marke

When comparing the top over- and under-expressed k-mers in Saprospirales with E.coli, most k-mers were
```

```{r}
# Calculate K-mer frequency in E. coli protein sequences
e.coli_prot_3_count <- count(e.coli_prot_unlist, wordsize = 3, alphabet = aa_alphabet)

e.coli_prot_4_count <- count(e.coli_prot_unlist, wordsize = 4, alphabet = aa_alphabet)

e.coli_prot_5_count <-  count(e.coli_prot_unlist, wordsize = 5, alphabet = aa_alphabet)

# Calculate K-mer frequency in Saprospirales protein sequences
Saprospirales_prot_3_count <- count(saprospirales_prot_unlist, wordsize = 3, alphabet = aa_alphabet)
Saprospirales_prot_4_count <- count(saprospirales_prot_unlist, wordsize = 4, alphabet = aa_alphabet)
```

```
Saprospirales_prot_5_count <- count(saprospirales_prot_unlist, wordsize = 5, alphabet = aa_alphabet)

# create Function to get top N over- and under-represented k-mers
top_kmers <- function(kmer_counts, N = 10) {
  # Sort k-mers by frequency
  kmer_sorted <- sort(kmer_counts)
    # Get N least frequent (under-represented) k-mers
  under <- head(kmer_sorted[kmer_sorted > 0], N)

  # Get N most frequent (over-represented) k-mers
  over  <- tail(kmer_sorted, N)

  list(over = over, under = under)
}

# CalculatE the top over- and under-expressed k-mers
Saprospirales_3_mer<- top_kmers(Saprospirales_prot_3_count, N = 10)
Saprospirales_3_mer
Saprospirales_4_mer <- top_kmers(Saprospirales_prot_4_count, N = 10)
Saprospirales_4_mer
Saprospirales_5_mer <- top_kmers(Saprospirales_prot_5_count, N = 10)
Saprospirales_5_mer

e.coli_3_mer<- top_kmers(e.coli_prot_3_count, N = 10)
e.coli_3_mer
e.coli_4_mer <- top_kmers(e.coli_prot_4_count, N = 10)
e.coli_4_mer
e.coli_5_mer <- top_kmers(e.coli_prot_5_count, N = 10)
e.coli_5_mer

# Common over-expressed k-mers in both organisms

common_over_3 <- intersect(names(Saprospirales_3_mer$over), names(e.coli_3_mer$over))

common_over_4 <- intersect(names(Saprospirales_4_mer$over), names(e.coli_4_mer$over))

common_over_5 <- intersect(names(Saprospirales_5_mer$over), names(e.coli_5_mer$over))

# Common under-expressed k-mers in both organisms
common_under_3 <- intersect(names(Saprospirales_3_mer$under), names(e.coli_3_mer$under))
common_under_4 <- intersect(names(Saprospirales_4_mer$under), names(e.coli_4_mer$under))
common_under_5 <- intersect(names(Saprospirales_5_mer$under), names(e.coli_5_mer$under))

# over-expressed k-mers in Saprospirales but not over expressed in E.coli
unique_over_3 <- setdiff(names(Saprospirales_3_mer$over), names(e.coli_3_mer$over))

unique_over_4 <- setdiff(names(Saprospirales_4_mer$over), names(e.coli_4_mer$over))

unique_over_5 <- setdiff(names(Saprospirales_5_mer$over), names(e.coli_5_mer$over))

# for overexpressed 3-mers

# Names of top over-expressed 3-mers in Saprospirales
sapro_3mers <- names(Saprospirales_3_mer$over)
```

```
# Initialize matrix: rows = organisms, columns = k-mers
compare_matrix <- matrix(0, nrow = 2, ncol = length(sapro_3mers),
                         dimnames = list(c("Saprospirales", "E.coli"), sapro_3mers))

# Fill counts for Saprospirales
compare_matrix["Saprospirales", ] <- Saprospirales_3_mer$over[sapro_3mers]

# Fill counts for E. coli: take counts if present, otherwise 0
compare_matrix["E.coli", ] <- sapply(sapro_3mers, function(k) {
  if(k %in% names(e.coli_prot_3_count)){
  # k %in% names
    e.coli_prot_3_count[k]   # use the raw count from E. coli
  } else {
    0
  }
})

barplot(compare_matrix,
        beside = TRUE,                 # side-by-side bars
        col = c("steelblue", "tomato"),  # Sapro = blue, E. coli = red
        las = 2,                        # rotate x-axis labels
        main = "Top Over-Expressed 3-mers in Saprospirales vs Counts in E. coli",
        ylab = "Frequency",
        cex.names = 0.8)
legend("topright", legend = c("Saprospirales", "E. coli"), fill = c("steelblue", "tomato"))

# for overexpressed 4-mers

# Names of top over-expressed 3-mers in Saprospirales
sapro_4mers <- names(Saprospirales_4_mer$over)

# Initialize matrix: rows = organisms, columns = k-mers
compare_matrix <- matrix(0, nrow = 2, ncol = length(sapro_4mers),
                         dimnames = list(c("Saprospirales", "E.coli"), sapro_4mers))

# Fill counts for Saprospirales
compare_matrix["Saprospirales", ] <- Saprospirales_4_mer$over[sapro_4mers]

# Fill counts for E. coli: take counts if present, otherwise 0
compare_matrix["E.coli", ] <- sapply(sapro_4mers, function(k) {
  if(k %in% names(e.coli_prot_4_count)){
    e.coli_prot_4_count[k]   # use the raw count from E. coli
  } else {
    0
  }
})

barplot(compare_matrix,
        beside = TRUE,                 # side-by-side bars
        col = c("steelblue", "tomato"),  # Sapro = blue, E. coli = red
        las = 2,                        # rotate x-axis labels
        main = "Top Over-Expressed 4-mers in Saprospirales vs Counts in E. coli",
        ylab = "Frequency",
```

```
        cex.names = 0.8)
legend("topright", legend = c("Saprospirales", "E. coli"), fill = c("steelblue", "tomato"))


# for overexpressed 5-mers in Saprospirales

# Names of top over-expressed 5-mers in Saprospirales
sapro_5mers <- names(Saprospirales_5_mer$over)

# Initialize matrix: rows = organisms, columns = k-mers
compare_matrix <- matrix(0, nrow = 2, ncol = length(sapro_5mers),
                         dimnames = list(c("Saprospirales", "E.coli"), sapro_5mers))

# Fill counts for Saprospirales
compare_matrix["Saprospirales", ] <- Saprospirales_5_mer$over[sapro_5mers]

# Fill counts for E. coli: take counts if present, otherwise 0
compare_matrix["E.coli", ] <- sapply(sapro_5mers, function(k) {
  if(k %in% names(e.coli_prot_5_count)){
    e.coli_prot_5_count[k]   # use the raw count from E. coli
  } else {
    0
  }
})

barplot(compare_matrix,
        beside = TRUE,                  # side-by-side bars
        col = c("steelblue", "tomato"),  # Sapro = blue, E. coli = red
        las = 2,                         # rotate x-axis labels
        main = "Top Over-Expressed 5-mers in Saprospirales vs Counts in E. coli",
        ylab = "Frequency",
        cex.names = 0.8)
legend("topright", legend = c("Saprospirales", "E. coli"), fill = c("steelblue", "tomato"))

# For under-expressed 3-mer genes of Saprospirales

# Names of top under-expressed 3-mers in Saprospirales
sapro_3mers_under <- names(Saprospirales_3_mer$under)

# Initialize matrix: rows = organisms, columns = k-mers
compare_matrix <- matrix(0, nrow = 2, ncol = length(sapro_3mers_under),
                         dimnames = list(c("Saprospirales", "E.coli"), sapro_3mers_under))

# Fill counts for Saprospirales
compare_matrix["Saprospirales", ] <- Saprospirales_3_mer$under[sapro_3mers_under]

# Fill counts for E. coli: take counts if present, otherwise 0
compare_matrix["E.coli", ] <- sapply(sapro_3mers_under, function(k) {
  if(k %in% names(e.coli_prot_3_count)){
    e.coli_prot_3_count[k]   # use the raw count from E. coli
  } else {
    0
  }
})
```

```
barplot(compare_matrix,
        beside = TRUE,                  # side-by-side bars
        col = c("steelblue", "tomato"),  # Sapro = blue, E. coli = red
        las = 2,                         # rotate x-axis labels
        main = "Top Under-Expressed 3-mers in Saprospirales vs Counts in E. coli",
        ylab = "Frequency",
        cex.names = 0.8)
legend("topright", legend = c("Saprospirales", "E. coli"), fill = c("steelblue", "tomato"))

# For under-expressed 4-mer genes of Saprospirales

# Names of top under-expressed 4-mers in Saprospirales
sapro_4mers_under <- names(Saprospirales_4_mer$under)

# Initialize matrix: rows = organisms, columns = k-mers
compare_matrix <- matrix(0, nrow = 2, ncol = length(sapro_4mers_under),
                         dimnames = list(c("Saprospirales", "E.coli"), sapro_4mers_under))

# Fill counts for Saprospirales
compare_matrix["Saprospirales", ] <- Saprospirales_4_mer$under[sapro_4mers_under]

# Fill counts for E. coli: take counts if present, otherwise 0
compare_matrix["E.coli", ] <- sapply(sapro_4mers_under, function(k) {
  if(k %in% names(e.coli_prot_4_count)){
    e.coli_prot_4_count[k]   # use the raw count from E. coli
  } else {
    0
  }
})

barplot(compare_matrix,
        beside = TRUE,                  # side-by-side bars
        col = c("steelblue", "tomato"),  # Sapro = blue, E. coli = red
        las = 2,                         # rotate x-axis labels
        main = "Top Under-Expressed 4-mers in Saprospirales vs Counts in E. coli",
        ylab = "Frequency",
        cex.names = 0.8)
legend("topright", legend = c("Saprospirales", "E. coli"), fill = c("steelblue", "tomato"))

# For under-expressed 5-mer genes of Saprospirales

# Names of top under-expressed 5-mers in Saprospirales
sapro_5mers_under <- names(Saprospirales_5_mer$under)

# Initialize matrix: rows = organisms, columns = k-mers
compare_matrix <- matrix(0, nrow = 2, ncol = length(sapro_5mers_under),
                         dimnames = list(c("Saprospirales", "E.coli"), sapro_5mers_under))

# Fill counts for Saprospirales
compare_matrix["Saprospirales", ] <- Saprospirales_5_mer$under[sapro_5mers_under]

# Fill counts for E. coli: take counts if present, otherwise 0
compare_matrix["E.coli", ] <- sapply(sapro_5mers_under, function(k) {
```

```
  if(k %in% names(e.coli_prot_5_count)){
    e.coli_prot_5_count[k]   # use the raw count from E. coli
  } else {
    0
  }
})

barplot(compare_matrix,
        beside = TRUE,                 # side-by-side bars
        col = c("steelblue", "tomato"),  # Sapro = blue, E. coli = red
        las = 2,                        # rotate x-axis labels
        main = "Top Under-Expressed 5-mers in Saprospirales vs Counts in E. coli",
        ylab = "Frequency",
        cex.names = 0.8)
legend("topright", legend = c("Saprospirales", "E. coli"), fill = c("steelblue", "tomato"))
```

The chatgpt was used to find the error in the code and to find the suitable codes when required.