KI-RIKEN joint course 2024

Shu

2024-09-20

Task 1 - Literature

1. Read the paper: https://academic.oup.com/bioinformatics/article/38/22/5126/6730725#supplementary-data
2. Answer to the questions

a. Medically relevant insight from the article:

The medical relevance of SCAFE's analysis is its potential to identify cell-type-specific cis-regulatory elements (CREs) such as enhancers and promoters, which are critical for understanding gene regulation in diseases. By identifying transcription start sites (TSS) at the single-cell level, SCAFE can help pinpoint CREs that may be dysregulated in disease contexts like cancer or genetic disorders. This offers insight into how specific regulatory elements may drive abnormal gene expression in diseased cells, leading to better understanding and potential targeting of these regulatory elements for therapeutic purposes.

b. Genomics technology/technologies used:

SCAFE (Single Cell Analysis of Five-prime Ends) provides an end-to-end solution for processing of single cell 5'end RNA-seq data.

3. Further related research questions

a. Three related research questions:

   How do cell-type-specific tCREs differ between healthy and diseased tissues? This question extends SCAFE's ability to identify regulatory elements to a comparative analysis between different states of health, potentially revealing new biomarkers or therapeutic targets.

   Can tCREs be used to predict cell fate decisions in developmental or differentiation processes? During development or cell differentiation, specific cis-regulatory elements become activated or repressed to guide cell fate decisions. By tracking changes in tCRE activity over time or across different developmental stages, we could uncover how regulatory elements guide lineage specification, and how disruptions in tCRE activity might lead to developmental disorders or diseases.

   How do specific transcription factors interact with tCREs in the regulation of gene expression across different cell types? This question delves into transcription factor (TF) binding at identified tCREs and how these interactions may vary across different cell types, providing a better understanding of cell-type-specific regulatory networks.

Task 2 - Git repositories and R Markdown

Github repository. https://github.com/xufengshu/xufengshu.github.io.git

Task 3 - Introduction to R and online R course

```
##  install Bioconductor
## if (!require("BiocManager", quietly = TRUE))
## install.packages("BiocManager")
## BiocManager::install(version = "3.19")
## install Tidyverse
## install.packages("tidyverse")
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

Task 4 - Using R example datasets

```
## Q: What is the average and median CO2 uptake of the plants from Quebec
# and Mississippi?

data("CO2")
help("CO2")
# extract out CO2 uptake of the plants from Quebec
Quebec_CO2_uptake = CO2[which(CO2$Type == "Quebec"), c(2,5)]
Quebec_average = mean(Quebec_CO2_uptake$uptake)
Quebec_average
```

```
## [1] 33.54286
```

```
Quebec_median = median(Quebec_CO2_uptake$uptake)
Quebec_median
```

```
## [1] 37.15
```

```
# extract out CO2 uptake of the plants from Mississippi
Mississippi_CO2_uptake = CO2[which(CO2$Type == "Mississippi"), c(2,5)]
Mississippi_average = mean(Mississippi_CO2_uptake$uptake)
Mississippi_average
```

```
## [1] 20.88333
```

```
Mississippi_median = median(Mississippi_CO2_uptake$uptake)
Mississippi_median
```

```
## [1] 19.3
```

Task 5 - R Functions

```
## use CO2 uptake vector as an example to execute the functions.

## Q: Write a function that calculates the ratio of the mean and the
# median of a given vector.
# function
ratio_mean_median = function(data_vector){
  mean = mean(data_vector)
  median = median(data_vector)
  ratio = mean / median
  return(ratio)
}
#test
ratio = ratio_mean_median(CO2$uptake)
print(ratio)
```

```
## [1] 0.9615935
```

```
## Q: Write a function that ignores the lowest and the highest value
# from a given vector and calculate the mean.
#function
calculate_mean_without_extremes <- function(data_vector) {
  # Check if the vector has at least 3 elements
  if (length(data_vector) < 3) {
    stop("Input vector must have at least 3 elements.")
  }
  # Sort the vector in ascending order
  sorted_data <- sort(data_vector)
  # Remove the lowest and highest values
  trimmed_data <- sorted_data[-c(1, length(sorted_data))]
  # Calculate the mean of the trimmed data
  mean_value <- mean(trimmed_data)
  return(mean_value)
}
#test
mean = calculate_mean_without_extremes(CO2$uptake)
print(mean)
```

```
## [1] 27.22805
```

```
## piping
library(magrittr)
```

```
##
## Attaching package: 'magrittr'

## The following object is masked from 'package:purrr':
##
##     set_names

## The following object is masked from 'package:tidyr':
##
##     extract
```

```
## Q3: Familiarize yourself with piping
##  *Why use pipes in R:
# **Readability**: Pipes allow you to chain together operations in a left-to-right fashion,
# making your code more readable and resembling a natural language flow.

# **Reduced Nesting**: Using pipes, you can avoid deeply nested function calls,
# which can be hard to read and debug. This leads to cleaner, more maintainable code.

# **Code Efficiency**: Pipes can make your code more efficient since intermediate results
# are stored automatically, and you don't have to repeatedly assign variables.

# **Interactive Data Exploration**: For interactive data exploration, pipes can be
# especially helpful because they allow you to build up your analysis step by step.
```

```
## *How to use pipes in R:
# the `%>%` operator passes the result of one operation as the first argument to the
# next operation, allowing you to chain together multiple data manipulation functions.

## *When not to use pipes in R:
# **Overly Complex Chains**: Avoid creating overly complex chains of operations.
# If your pipe chain becomes too long and convoluted, it may reduce code readability
# and make debugging more challenging.

# **Conditional Execution**: If you need conditional execution of functions or complex
# branching in your code, it might be better to use traditional control flow structures
# like `if`, `else`, or loops.

# **Performance Critical Code**: For highly performance-critical code, using pipes might
# introduce some overhead. In such cases, you may need to optimize your code differently.

# **Non-Pipe-Friendly Functions**: Some functions may not work well with pipes,
# especially if they have unconventional argument orders or side effects.




## Q4: Familiarize yourself with the apply-family of functions (apply, lapply, sapply etc.)

## **Applying Functions to Data Frames or Matrices**:
# `apply()`: Useful for applying a function to the rows or columns of a matrix or data frame.
# For example, calculating row-wise or column-wise means or sums.

# `lapply()`: Used to apply a function to each element of a list or data frame column,
# returning a list as output. This is handy for transforming and summarizing data.

## **Simplifying Code**:
#`sapply()`: Similar to `lapply()`, but it tries to simplify the result into a vector or
# matrix whenever possible, which can lead to cleaner code.

## **Multiple Input Lists**:
#`mapply()`: Useful when you have multiple input lists or vectors, and you want to
# apply a function element-wise to each combination of inputs.

## **Split-Apply-Combine Operations**:
#`tapply()`: Great for performing split-apply-combine operations, where you split your data
# into groups based on one or more factors, apply a function to each group, and then
# combine the results.

## **Working with Complex Data Structures**:
# The apply family can handle complex nested data structures and can help you extract,
# transform, or summarize data buried within lists or data frames.

## **Custom Function Application**:
# You can use these functions to apply your custom functions to data, allowing for
# flexible and customized data processing.

## **Speed and Efficiency**:
```

```
#In many cases, the apply family functions can be more efficient than using loops,
# especially when working with large datasets. They are often implemented in C,
# which can be faster than writing equivalent R code.

## **Functional Programming Paradigm**:
#The apply family functions align with the functional programming paradigm,
# which can lead to more concise and expressive code, enhancing the maintainability of your scripts.

## **Parallel Processing**:
#Some of the apply functions can be used in parallel environments,
# which can significantly speed up computations on multi-core machines.
```

Task 6 - Basic visualization with R

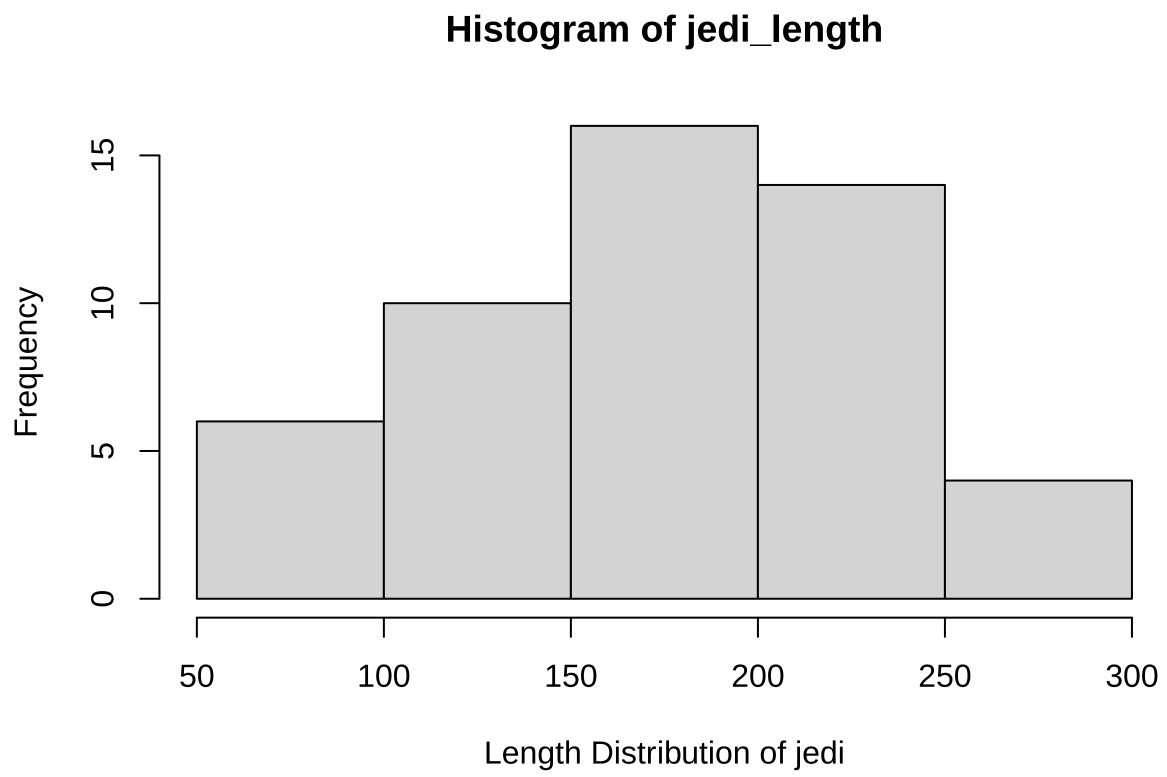
```
## install.packages("remotes")
library(remotes)
## installed
## install_url("http://emotion.utu.fi/wp-content/uploads/2019/11/nummenmaa_1.0.tar.gz",dependencies=TRU

library(tidyverse)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

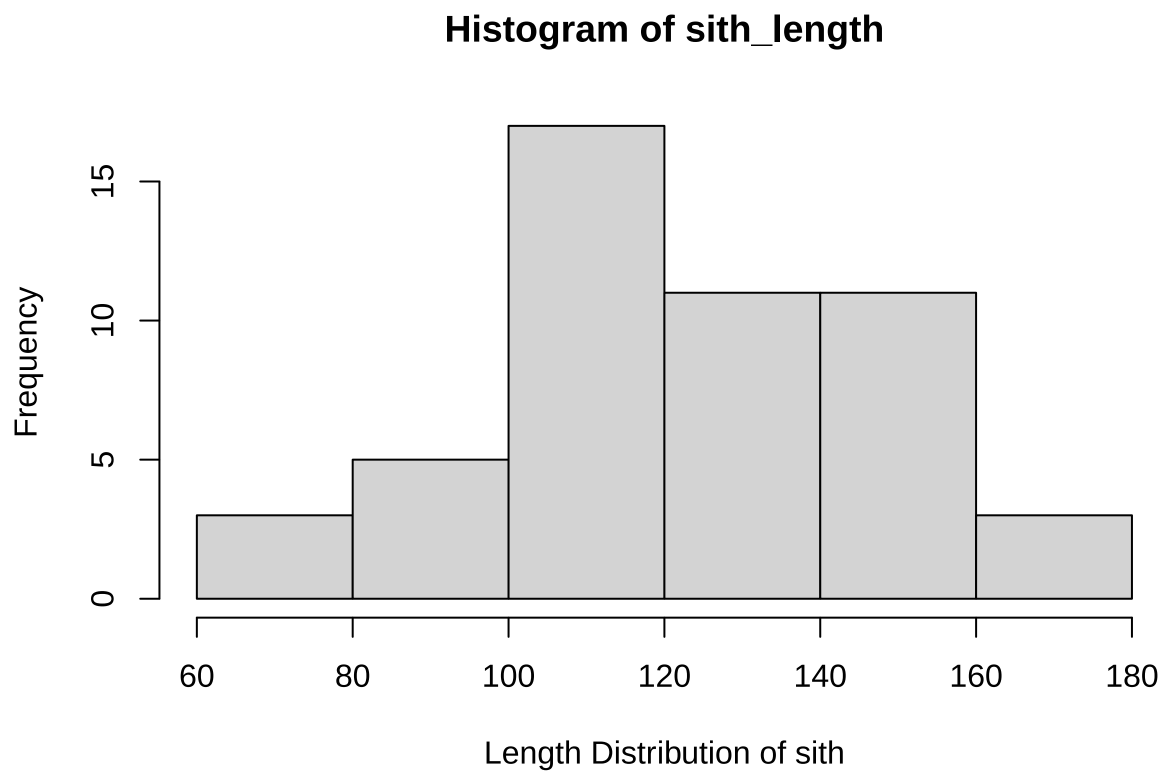
```
library(nummenmaa)


## Q1:
magic.guys <- read.csv("/home/winghin/T/magic_guys_2024.csv", header=TRUE, stringsAsFactors=FALSE)

head(magic.guys)
```

```
##   uniqId species length weight
## 1     p1    jedi  174.6   71.3
## 2     p2    jedi  252.2   70.8
## 3     p3    jedi  229.8   70.7
## 4     p4    jedi  176.2   80.4
## 5     p5    jedi  213.3   82.0
## 6     p6    jedi  112.5   64.2
```

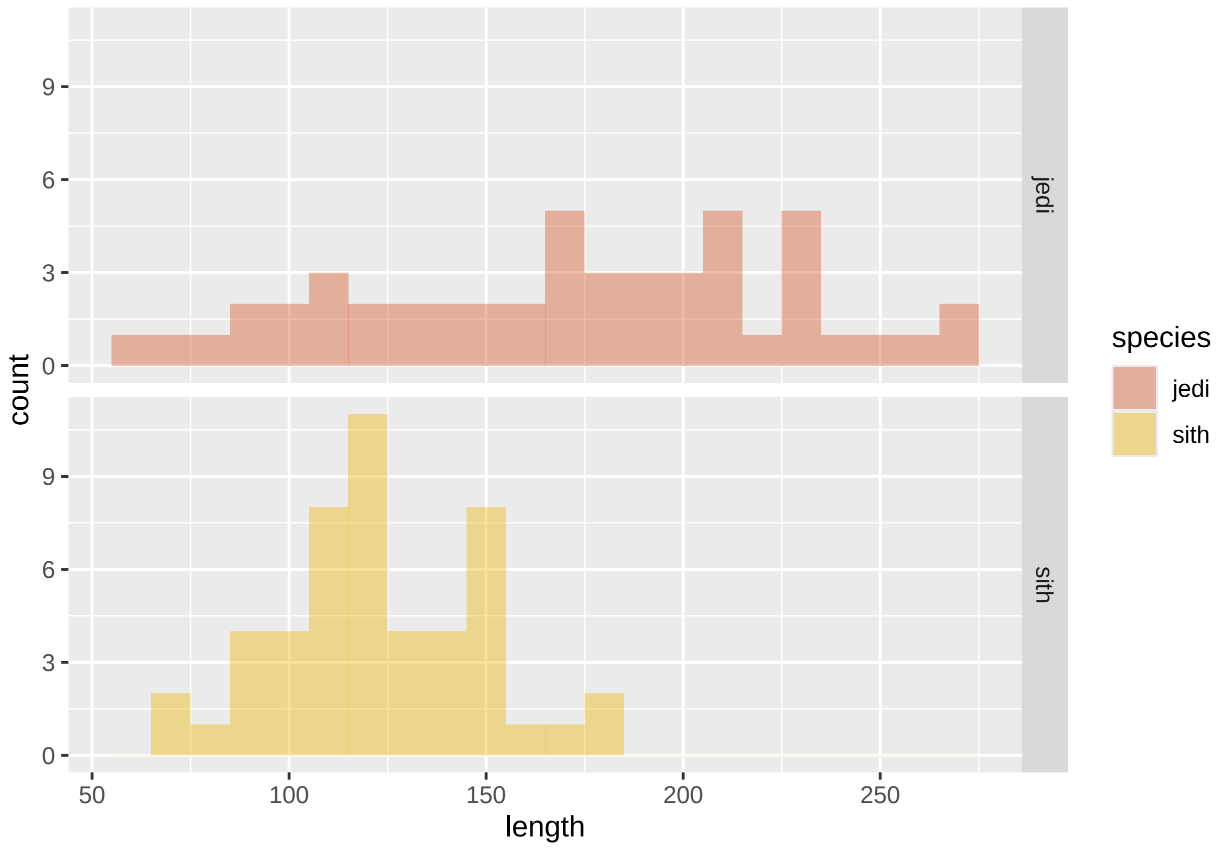
```
## histogram plot
jedi_length = magic.guys[which(magic.guys$species == "jedi"), 3]
sith_length = magic.guys[which(magic.guys$species == "sith"), 3]

hist(jedi_length, xlab = "Length Distribution of jedi")
```



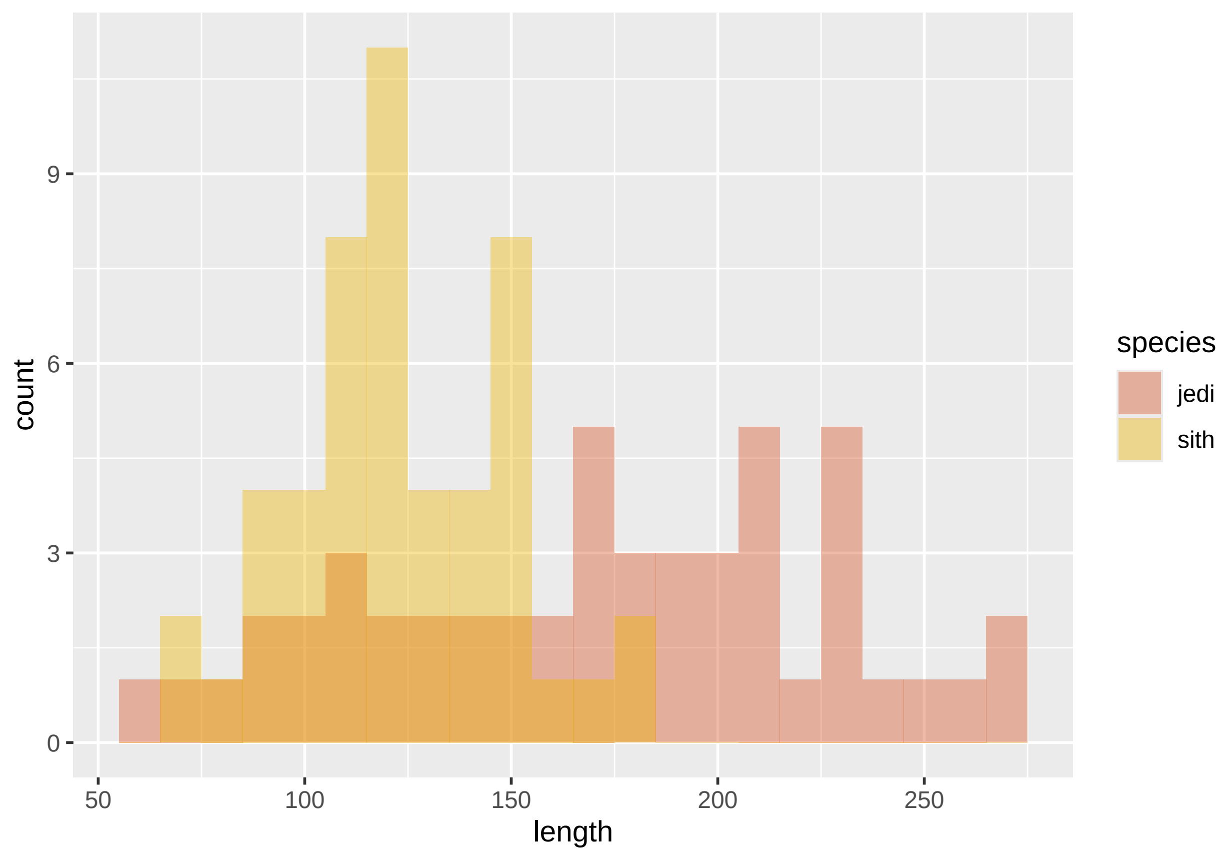
```r
hist(sith_length, xlab = "Length Distribution of sith")
```



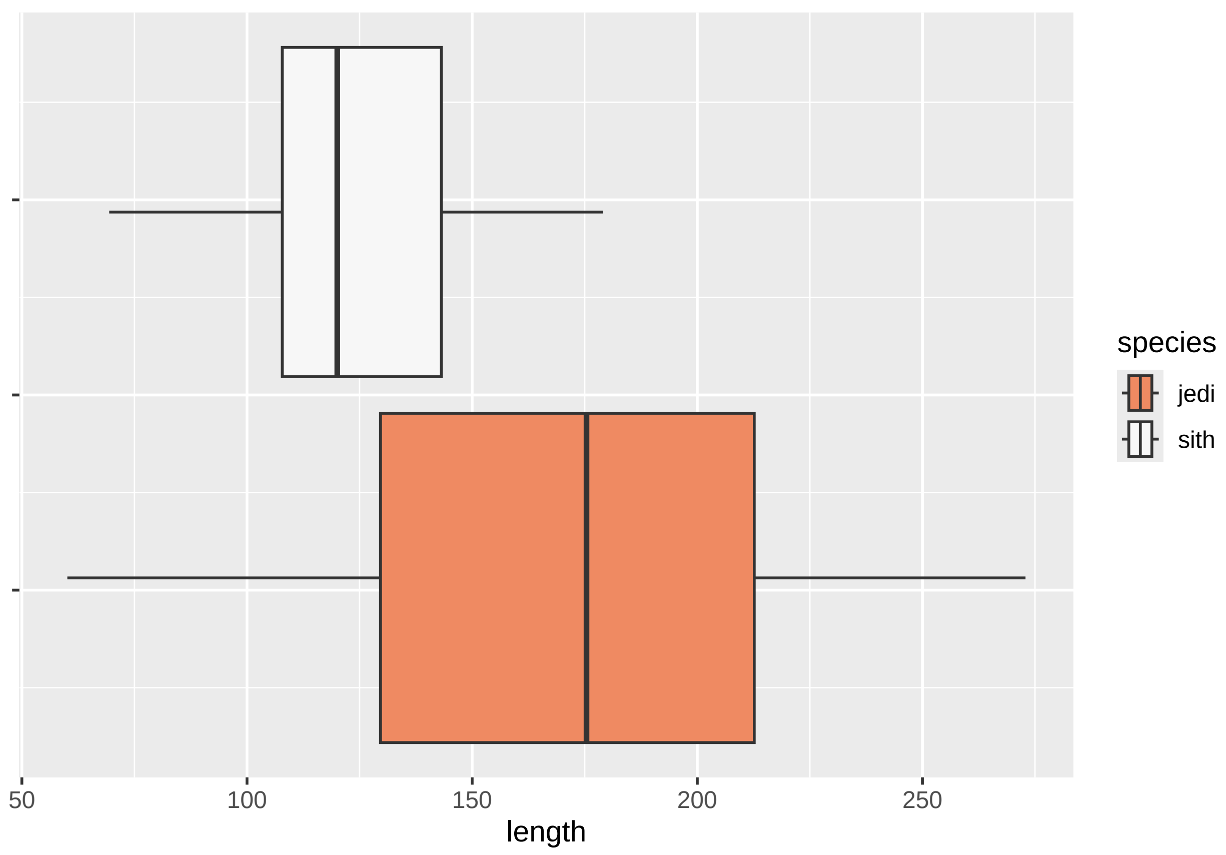
```
#seperate
compare.length = magic.guys %>%
  ggplot( aes(x=length, fill=species)) +
    geom_histogram(binwidth = 10, alpha=0.4, position = 'identity') +
    scale_fill_manual(values=c("#d75427","#eeb401")) +
    facet_grid(species ~ .)

compare.length
```



```
#merge
compare.length = magic.guys %>%
  ggplot( aes(x=length, fill=species)) +
    geom_histogram(binwidth = 10, alpha=0.4, position = 'identity') +
    scale_fill_manual(values=c("#d75427","#eeb401"))

compare.length
```



```
## box plot
compare.length = magic.guys %>%
  ggplot( aes(x=length, fill=species)) +
    geom_boxplot(outlier.colour="red", outlier.shape=8,outlier.size=4) +
    scale_fill_brewer(palette="RdBu") +
    theme(axis.text.y = element_blank())

compare.length
```



```
## save plots with pdf format
pdf("/osc-fs_home/xufeng/joint_KI_RIKEN/compare.length.pdf")
plot(compare.length)
dev.off()
```

```
## pdf
##   2
```

```
## save plots with svg format
svg("/osc-fs_home/xufeng/joint_KI_RIKEN/compare.length.svg")
plot(compare.length)
dev.off()
```

```
## pdf
##   2
```

```
## save plots with png format
png("/osc-fs_home/xufeng/joint_KI_RIKEN/compare.length.png")
plot(compare.length)
dev.off()
```

```
## pdf
##   2
```

```
# The PDF format for saving images is the most used for creating scientific documents,
# as they are easy to add to LaTeX and maintain the resolution even if you zoom in.
# However, if you need to edit the image after saving in order to add some decoration
# or perform some modifications you should use SVG. And png image file format is
# known to weight less than JPEG with better quality, as it supports transparent backgrounds.
```

```
## Q2:
microarray <- read.table("/home/winghin/T/microarray_data_2024.tab", header = T, stringsAsFactors=FALSE

nrow(microarray)
```

```
## [1] 553
```

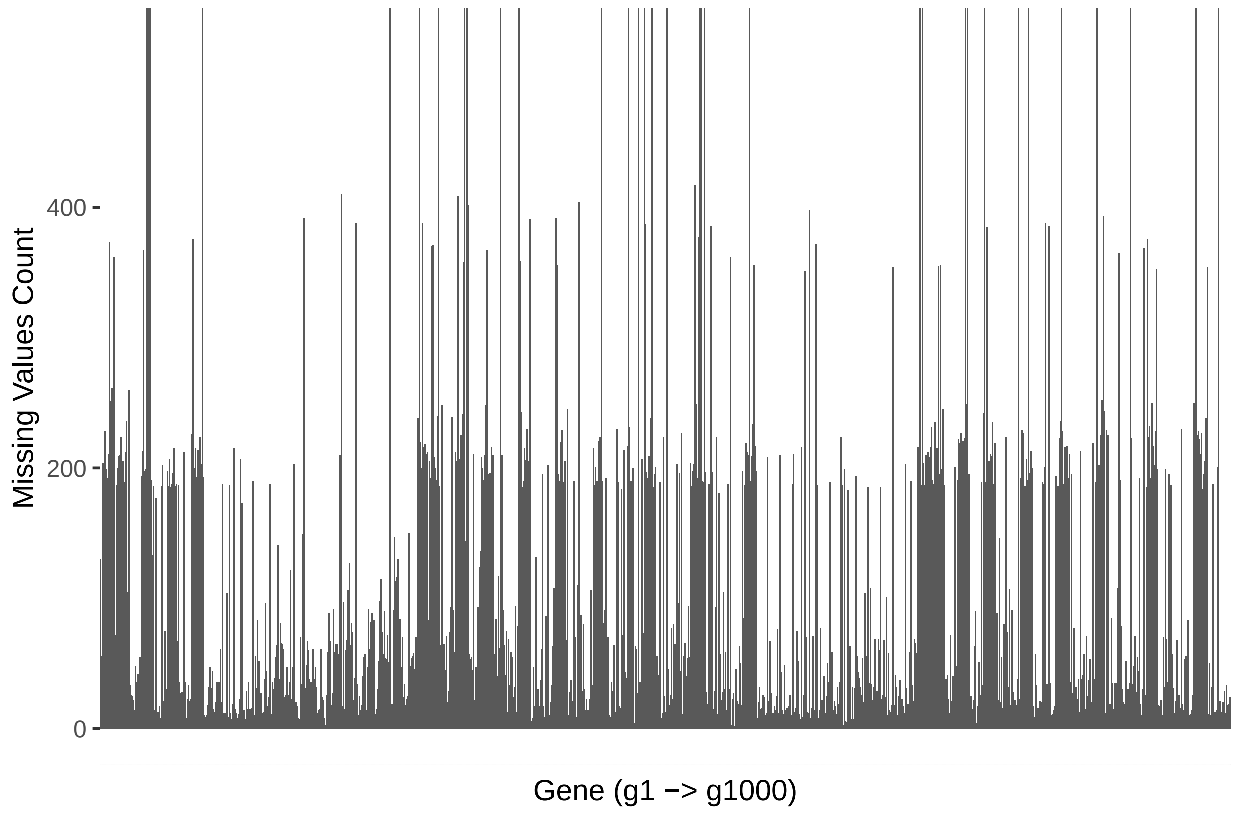
```
ncol(microarray)
```

```
## [1] 1000
```

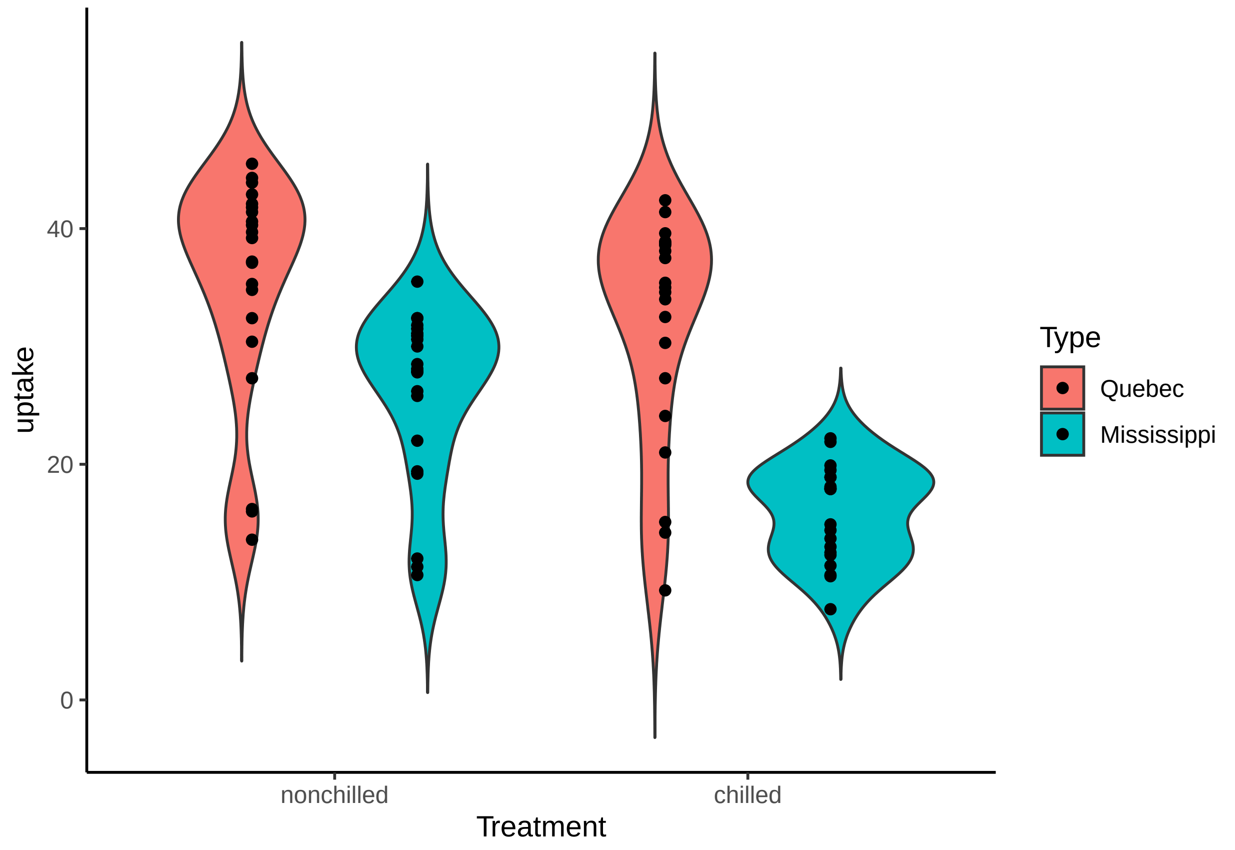
```
dim(microarray)
```

```
## [1]  553 1000
```

```
## Count the missing values per gene and visualize this result.
missing_values_per_gene = colSums(is.na(microarray))

gene_missing_counts <- data.frame(Gene = colnames(microarray), Missing_Values = missing_values_per_gene

ggplot(gene_missing_counts, aes(x = Gene, y = Missing_Values)) +
  geom_bar(stat = "identity") +
  labs(x = "Gene (g1 -> g1000)", y = "Missing Values Count") +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank())
```



```r
## Find the genes for which there are more than X% (X=10%, 20%, 50%) missing values.

p_10 = gene_missing_counts[which(gene_missing_counts$Missing_Values > (nrow(microarray)*0.1)), 1]

p_20 = gene_missing_counts[which(gene_missing_counts$Missing_Values > (nrow(microarray)*0.2)), 1]

p_50 = gene_missing_counts[which(gene_missing_counts$Missing_Values > (nrow(microarray)*0.5)), 1]


## Replace the missing values by the average expression value for the particular gene.

# Calculate the row-wise means for each gene, ignoring NA values
gene_means <- apply(microarray, 2, function(x) mean(x, na.rm = TRUE))
# Replace NA values with corresponding gene means
for (gene_col in 1:ncol(microarray)) {
  microarray[is.na(microarray[, gene_col]), gene_col] <- gene_means[gene_col]
}


## Q3: Visualize the data in the CO2 dataset in a way that gives you a deeper understanding of the
# data. What do you see?

# Load the CO2 dataset
data("CO2")

ggplot(data = CO2, aes(x = Treatment, y = uptake, fill = Type))+
  theme_classic() +
  geom_violin(trim = FALSE) +
  geom_point(position = position_dodge(width = 0.8))
```



```
# I found : nonchilled is a better condition for uptaking CO2. And in the same condition,
# Quebec can uptake more CO2 versus to Mississippi.
```

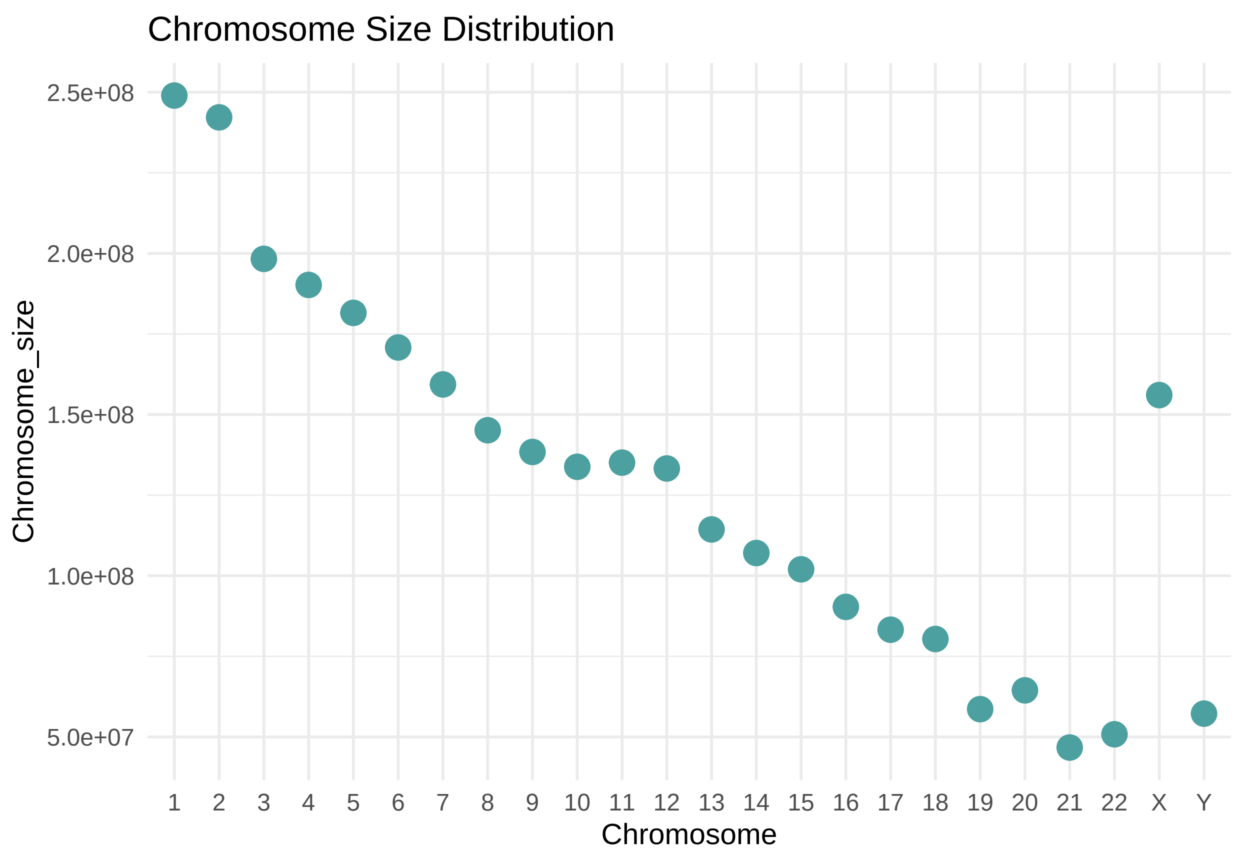
Task 7

```
## devtools::install_github("hirscheylab/tidybiology")

## a. Extract summary statistics (mean, median and maximum) for the following variables
# from the ‘chromosome’ data: variations, protein coding genes, and miRNAs.
# Utilize the tidyverse functions to make this as simply as possible.

library(tidybiology)
# Load the 'chromosome' data
data("chromosome")
# Use tidyverse functions to extract summary statistics
summary_stats <- chromosome %>%
  summarise(
    Mean_Variations = mean(variations, na.rm = TRUE),
    Median_Variations = median(variations, na.rm = TRUE),
    Max_Variations = max(variations, na.rm = TRUE),
    Mean_Protein_Coding_Genes = mean(protein_codinggenes, na.rm = TRUE),
    Median_Protein_Coding_Genes = median(protein_codinggenes, na.rm = TRUE),
    Max_Protein_Coding_Genes = max(protein_codinggenes, na.rm = TRUE),
    Mean_miRNAs = mean(mi_rna, na.rm = TRUE),
    Median_miRNAs = median(mi_rna, na.rm = TRUE),
    Max_miRNAs = max(mi_rna, na.rm = TRUE)
  )
```

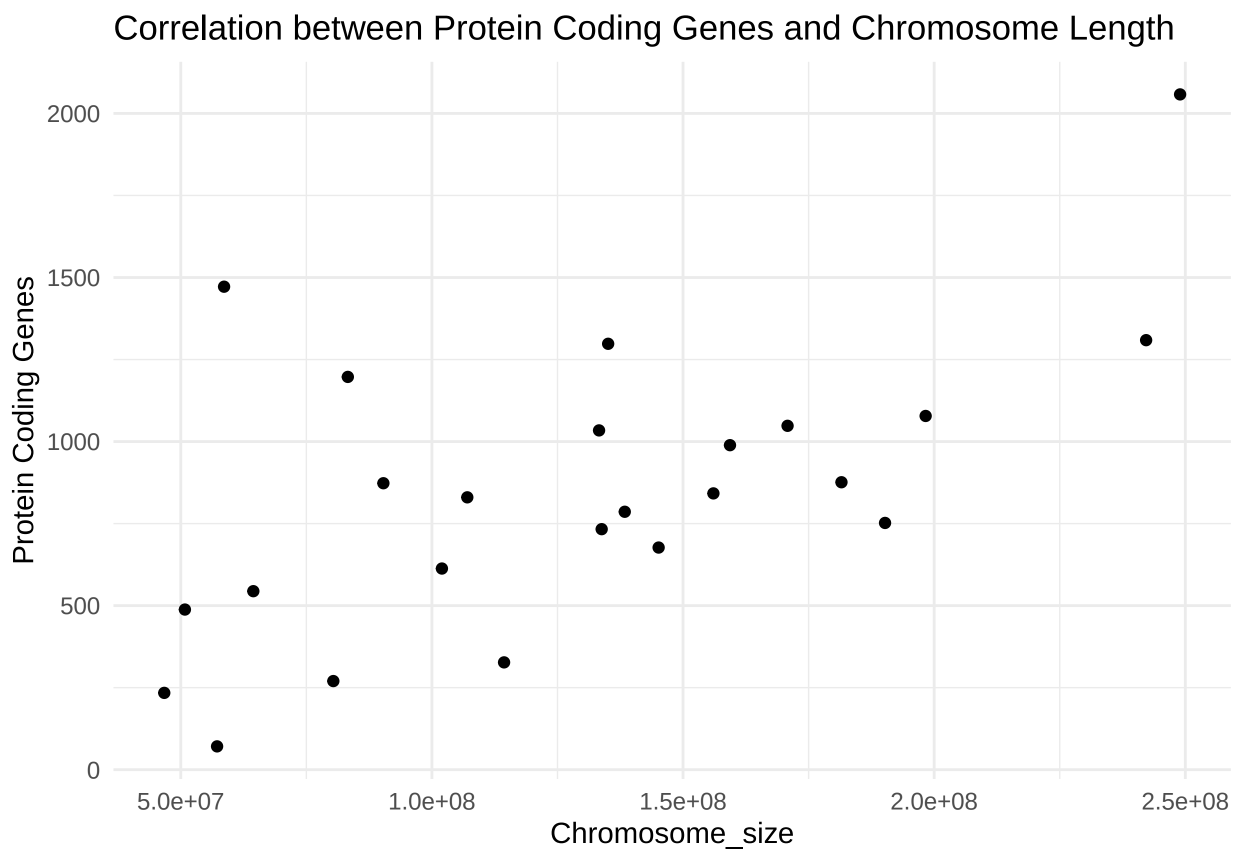
```r
summary_stats
```

```
## # A tibble: 1 x 9
##   Mean_Variations Median_Variations Max_Variations Mean_Protein_Coding_Genes
##             <dbl>             <dbl>          <dbl>                     <dbl>
## 1        6484572.           6172346       12945965                      850.
## # i 5 more variables: Median_Protein_Coding_Genes <dbl>,
## #   Max_Protein_Coding_Genes <int>, Mean_miRNAs <dbl>, Median_miRNAs <dbl>,
## #   Max_miRNAs <int>
```

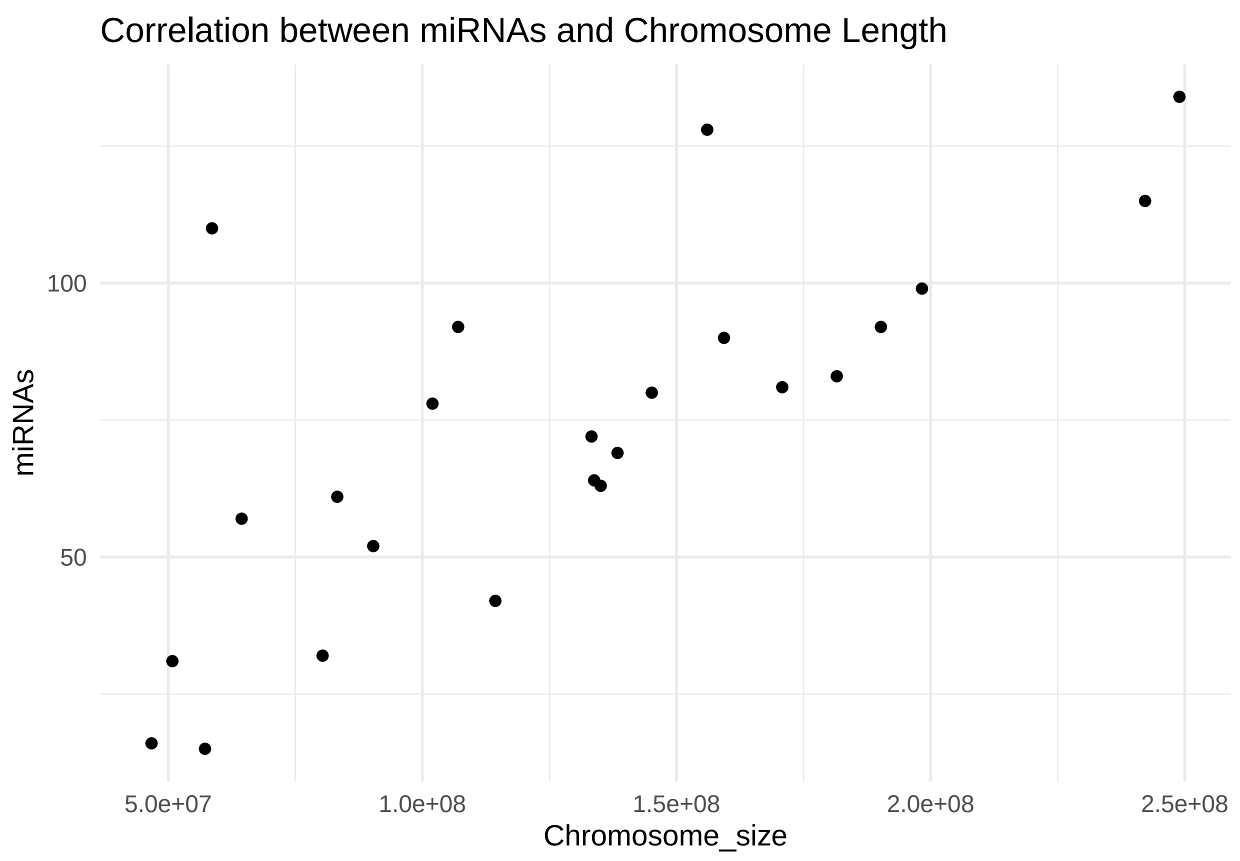
```r
## b. How does the chromosome size distribute? Plot a graph that helps to visualize
# this by using ggplot2 package functions.

# Create a scatter plot to visualize chromosome size distribution
ggplot(chromosome, aes(x = id, y = basepairs)) +
  geom_point(color = "#4da0a0", size = 4) +
  labs(title = "Chromosome Size Distribution",
       x = "Chromosome",
       y = "Chromosome_size") +
  theme_minimal()
```



```r
## c. Does the number of protein coding genes or miRNAs correlate with the length
# of the chromosome? Make two separate plots to visualize these relationships.

# Scatter plot to visualize correlation between protein coding genes and chromosome length
ggplot(chromosome, aes(x = basepairs, y = protein_codinggenes)) +
  geom_point() +
  labs(title = "Correlation between Protein Coding Genes and Chromosome Length",
       x = "Chromosome_size",
       y = "Protein Coding Genes") +
  theme_minimal()
```



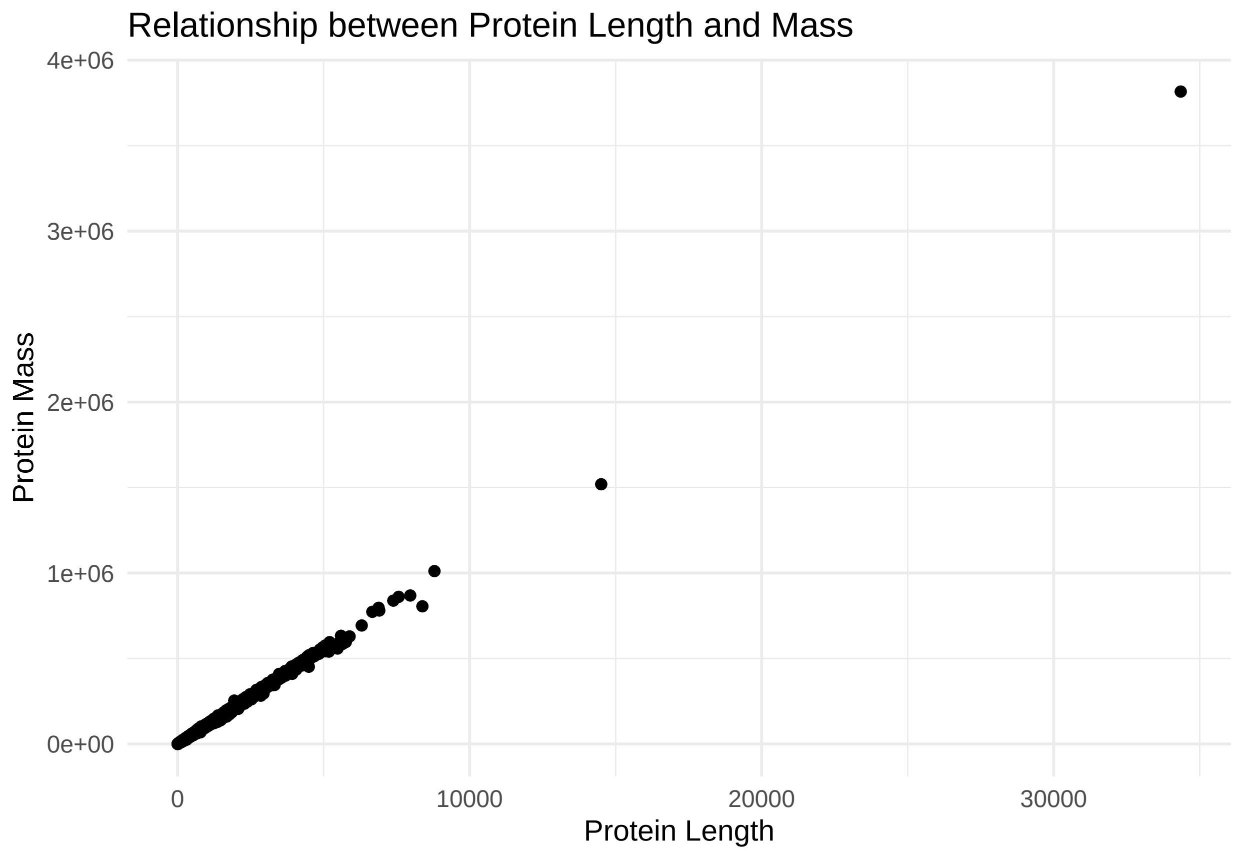
```
# Scatter plot to visualize correlation between miRNAs and chromosome length
ggplot(chromosome, aes(x = basepairs, y = mi_rna)) +
  geom_point() +
  labs(title = "Correlation between miRNAs and Chromosome Length",
       x = "Chromosome_size",
       y = "miRNAs") +
  theme_minimal()
```



```
## d. Calculate the same summary statistics for the ‘proteins’ data variables length and mass.
# Create a meaningful visualization of the relationship between these two variables by
# utilizing the ggplot2 package functions. Play with the colors, theme- and
# other visualization parameters to create a plot that pleases you.

data("proteins")
# Summary statistics for 'length' and 'mass' variables
summary_stats_proteins <- proteins %>%
  summarise(
    Mean_Length = mean(length, na.rm = TRUE),
    Median_Length = median(length, na.rm = TRUE),
    Max_Length = max(length, na.rm = TRUE),
    Mean_Mass = mean(mass, na.rm = TRUE),
    Median_Mass = median(mass, na.rm = TRUE),
    Max_Mass = max(mass, na.rm = TRUE)
  )

summary_stats_proteins
```

```
## # A tibble: 1 x 6
##   Mean_Length Median_Length Max_Length Mean_Mass Median_Mass Max_Mass
##         <dbl>         <dbl>      <dbl>     <dbl>       <dbl>    <dbl>
## 1        557.           414      34350    62061.      46140.  3816030
```

```
# Scatter plot to visualize the relationship between 'length' and 'mass'
ggplot(proteins, aes(x = length, y = mass)) +
  geom_point() +
  labs(title = "Relationship between Protein Length and Mass",
       x = "Protein Length",
       y = "Protein Mass") +
  theme_minimal()
```



Hi Xiaoxi

I am waiting for your comments and suggestions~

Best

Shu