Bioinformatics Analysis and Visualisation of Medical Genomics Data

Assignment 1 - Alen Lovric

Dependencies

- Pipeline depends on the following packages: ‘tidyverse’, ‘ggrepel’, ‘remotes’, and ‘ggpubr’ among the rest.

```r
# Load libraries
load_libs <- c('tidyverse', 'ggrepel', 'remotes', 'ggpubr',
               'kableExtra', "tinytex", 'summarytools', 'remotes')

sapply(load_libs, require, character.only = T)
```

Task 4 - Using R example datasets

1. Use the R internal CO2 dataset.
2. Describe briefly the content of the CO2 dataset using the help function.
3. What is the average and median CO2 uptake of the plants from Quebec and Mississippi?

```r
# CO2 data set
# Attach data and take a look
data(CO2)
help(CO2)

# Calculate average and median CO2 uptake
# of the plants from Quebec and Mississippi?
co2_summary <- CO2 %>% group_by(Type) %>% summarise(mean = mean(uptake),
                                     median = median(uptake))

# Print table
kable(co2_summary, caption = 'CO2 data set.', format = 'latex', booktabs = T) %>%
            kable_styling(latex_options = c('striped',"HOLD_position"))
```

Table 1: CO2 data set.

Type	mean	median
Quebec	33.54286	37.15
Mississippi	20.88333	19.30

Description: The CO2 data frame has 84 rows and 5 columns of data from an experiment on the cold tolerance of the grass species Echinochloa crus-galli.

4. In the "airway" example data from Bioconductor, how many genes are expressed in each sample? How many genes are not expressed in any sample?

```r
# Install airway if not already present
# BiocManager::install("airway")
library(airway)
data(airway)

# Extract counts data from the airway assay
counts <- assay(airway) %>% as.data.frame()

# Use tidyverse to calulate no. of expressed and non expressed genes per sample
counts_summary <- counts %>%
          summarise_all(.funs = list(expressed = function(x) sum(x > 0),
                                     notExpressed = function(x) sum(x == 0))) %>%
          pivot_longer(cols = everything()) %>%
          separate(name, c('Sample', 'temp'),sep = '_') %>%
          pivot_wider(names_from = 'temp', values_from = 'value')

# Print table
kable(counts_summary, caption = 'Airway dataset.',
      format = 'latex', booktabs = T) %>%
            kable_styling(latex_options = c('striped',"HOLD_position"))
```

Table 2: Airway dataset.

Sample	expressed	notExpressed
SRR1039508	24633	39469
SRR1039509	24527	39575
SRR1039512	25699	38403
SRR1039513	23124	40978
SRR1039516	25508	38594
SRR1039517	25998	38104
SRR1039520	24662	39440
SRR1039521	23991	40111

Description: Number of expressed and non-expressed genes per sample.

Task 5 - Creating R Functions

1. Write a fun that calculates the ratio of the mean and the median of a given vector.
2. Write a fun that calculates mean and ignores lowest and highest value from a given vector.

```r
# Create a random numeric vector with fixed seed for reproducibility
set.seed(1)
numval <- sample(1:100, 50)

# Function that calualtes mean-to-median ratio
vectorInfo <- function(x){
    res <- mean(x)/median(x)
    return(cat(paste('mean-to-median ratio:', round(res, 2))))
}

# Call the function using previously created numeric vector
vectorInfo(numval)
```

```
## mean-to-median ratio: 1.03
```

```r
# Create trimmed mean function
# Both min and max values are removed from vector before mean calulation
trimmedMean <- function(x){
      temp_x <- x[-c(which.max(x), which.min(x))]
      res <- sum(temp_x)/length(temp_x)
      return(cat(paste('Trimmed mean:', res)))
}

# Call the function using previously created numeric vector
trimmedMean(numval)
```

```
## Trimmed mean: 54.5
```

3. Write a short explanation of why, how, and when not to use pipes.

The use of pipes (%>%) in R, often provided by packages like **magrittr** and widely adopted by the **dplyr** package, has become a popular tool for enhancing the readability and efficiency of R code. However, there are situations where using pipes may not be the best choice, and understanding when not to use them is crucial for writing clean and maintainable R code.

Pipes are incredibly useful when you need to perform a series of operations on a dataset or object, as they allow you to chain together functions in a left-to-right fashion, enhancing code readability. They shine in data manipulation tasks, where you apply multiple transformations to a data frame, for example.

However, here are scenarios where using pipes may not be ideal:

- **Simple Operations:** For basic operations that don't involve chaining multiple functions together, pipes can add unnecessary complexity to the code. In such cases, using standard R syntax is more straightforward and doesn't introduce unnecessary overhead. **Debugging:** While pipes make code more readable, they can make debugging more challenging. When you encounter an error, it can be trickier to pinpoint which step in the pipeline is causing the issue, especially if the data transformations are complex.
- **Non-linear Data Flow:** When your data processing doesn't follow a linear sequence, pipes may not be the best choice. Pipes work best for operations that naturally flow from one step to the next. If your workflow involves branching or looping, using pipes can lead to convoluted code.
- **Performance:** In some cases, using pipes can introduce a small performance overhead compared to writing the code in a more traditional way. While this overhead is usually negligible, it might matter in computationally intensive tasks.
- **Code Style and Team Consistency:** If you're working on a team project, it's essential to maintain code consistency. Some team members may prefer traditional R syntax over pipes, so it's crucial to follow the project's coding guidelines.

4. Write a short explanation of why they could be useful in your work.

The apply-family of functions, including **apply**, **lapply**, **sapply**, and others, play a pivotal role in streamlining and enhancing the efficiency of data analysis and manipulation tasks within my work. These functions provide a concise and flexible way to apply operations across data structures in R, such as matrices, lists, and data frames.

One of the key advantages of these functions is their ability to abstract away the complexities of looping constructs. Traditional for-loops in R can be cumbersome and error-prone, especially when dealing with multidimensional data. The apply functions, on the other hand, allow me to express my intentions more clearly and succinctly, reducing the chances of coding errors.

apply is particularly useful when working with matrices, as it efficiently applies a function to rows or columns, enabling me to calculate row-wise or column-wise statistics or perform custom operations effortlessly. For lists, **lapply** facilitates the application of a function to each element, returning the results as a list—a valuable approach for iterative tasks involving data structures of varying lengths and types.

The **sapply** function is a versatile tool that simplifies the conversion of the output from lapply into a more manageable format, such as a vector or data frame. This simplification is especially useful when I need to work with results in a unified structure.

Furthermore, the apply-family functions promote code modularity and maintainability. By encapsulating specific operations within functions and then applying them using the apply-family, I can create cleaner, more organized code. This modular approach enhances code reusability and allows for easier debugging and testing.

Task 6 - Basic visualization with R

1. Compare the distributions of the body heights of two species from the 'magic_guys.csv' dataset.

A. using the basic 'hist' function as well as 'ggplot' and 'geom_histogram' functions from the ggplot2 package. Optimize the plots for example by trying several different 'breaks'. Note that ggplot2-based functions give you many more options for changing the visualization parameters, try some of them.

```r
# Read in magic_guys and look at the summary by species
magic_guys <- read.csv('./data/magic_guys.csv')
#magic_guys %>% group_by(species) %>% dfSummary()

# Compare the hist visualization using base and ggplot
## First plot
par(mfrow=c(1, 2))
hist(magic_guys$length[magic_guys$species == 'jedi'],
     breaks = 5, las = 1, ylab = 'Frequency', xlab = '', main = '')

## Second plot
hist(magic_guys$length[magic_guys$species == 'sith'],
     breaks = 5, ann = FALSE, las = 1)

## Title
title(main = '', xlab = 'Jedi (left) and Sith height (right) by using break = 5',
      outer = TRUE, line = -2)
```

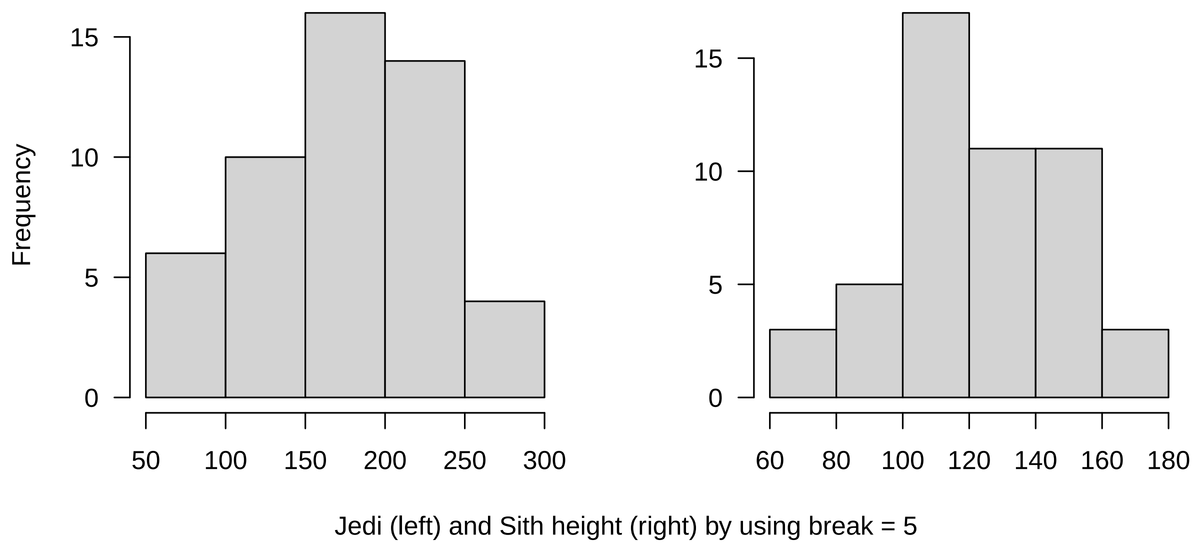

Figure 1: Histogram of Jedi and Sith heigth using base R functions

- Use hist function from base R to compare distribution of Jedi and Sith height
- Using breaks = 10

```r
# Compare the hist visualization using base and ggplot
## First plot - left half of x-axis, right margin set to 0 lines
par(mfrow=c(1, 2))
hist(magic_guys$length[magic_guys$species == 'jedi'],
     xlim = c(50, 300),
     breaks = 10, las = 1, ylab = 'Frequency', xlab = '', main = '')

## Cecond plot - right half of x-axis, left margin set to 0 lines
hist(magic_guys$length[magic_guys$species == 'sith'],
     breaks = 10, ann = FALSE, las = 1)

title(main = '', xlab = 'Jedi (left) and Sith height (right) by using break = 10',
      outer = TRUE, line = -2)
```

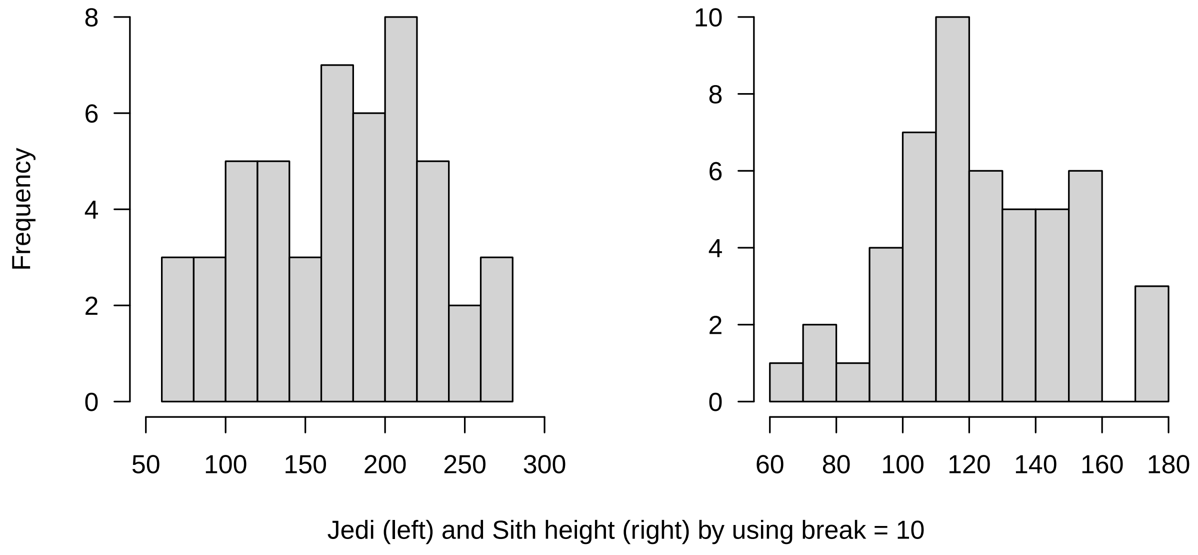

Figure 2: Histogram of Jedi and Sith heigth using base R functions

- Use geom_histogram function from ggplot library to compare distribution of Jedi and Sith height
- Using binwidth = 10 and 20

```r
# Use ggplot for histogram
# Here we do everything in parallel
## Small workaround - duplicated magic_guys and convert species to character
magic_guys_dp <- magic_guys %>% mutate(species = paste0(species, ': bin 10')) %>%
                              bind_rows(magic_guys %>%
                                            mutate(species = paste0(species, ': bin 20')))

# Plot the data using several layers of histogram referring to bandwidth used
ggplot(magic_guys_dp, aes(x = length)) +
     geom_histogram(data=subset(magic_guys_dp, species=="jedi: bin 10"),
                    binwidth=10, color = 'black', fill = alpha('#7ea4b3', 0.5)) +
     geom_histogram(data=subset(magic_guys_dp, species=="jedi: bin 20"),
                    binwidth=20, color = 'black', fill = '#7ea4b3') +
     geom_histogram(data=subset(magic_guys_dp, species=="sith: bin 10"),
                    binwidth=10, color = 'black', fill = alpha('#cfb7ae', 0.5)) +
     geom_histogram(data=subset(magic_guys_dp, species=="sith: bin 20"),
                    binwidth=20, color = 'black', fill = '#cfb7ae') +
     facet_grid(~species, scales="free_x") +
     labs(y = 'Counts', x = 'Heigth') +
     theme_bw()
```

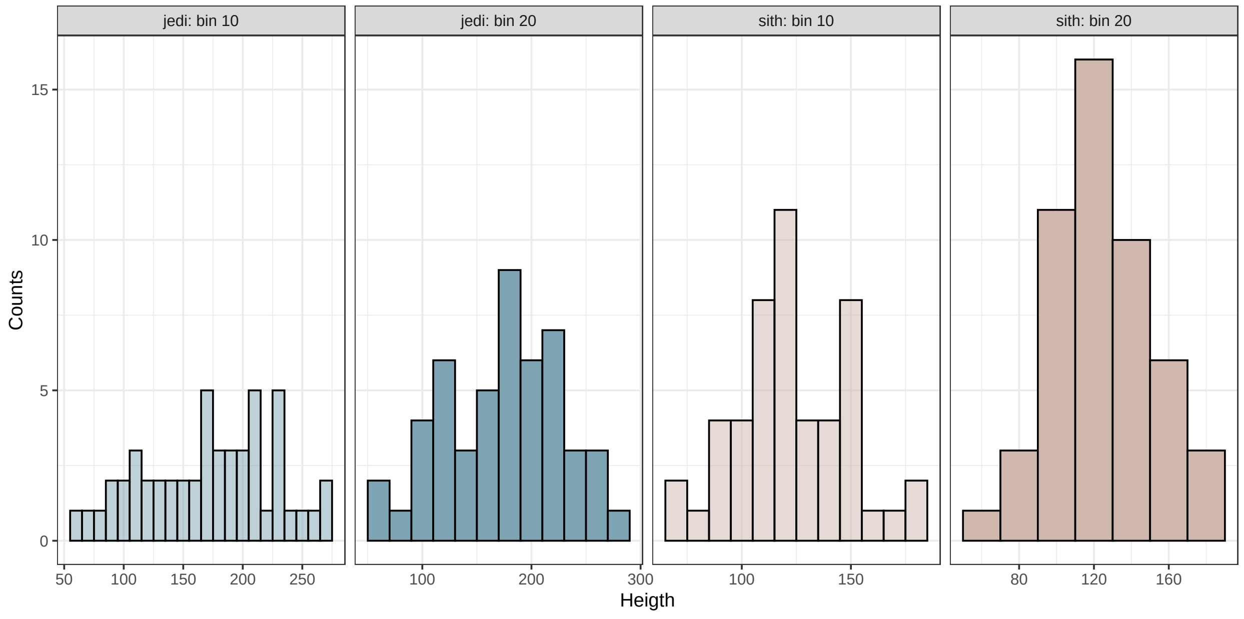

Figure 3: Histogram of Jedi and Sith heigth using ggplot.

B. Do the same comparison as in a. but with boxplots. If you want to use the ggplot2-package, use the functions 'ggplot' and 'geom_boxplot'.

```
# Use ggplot and geom_boxplot and fill by species
## change the theme and colors
gg <- ggplot(magic_guys, (aes(species, length))) +
            geom_boxplot(aes(fill = species), show.legend = F, notch = T) +
            scale_fill_manual(values = c('#DFB359', '#374684')) +
            scale_x_discrete(labels = str_to_title(unique(magic_guys$species))) +
            coord_flip() +
            labs(x = 'Species', y = 'Heigth') +
            theme_bw() +
            theme(panel.grid = element_blank())

plot(gg)
```

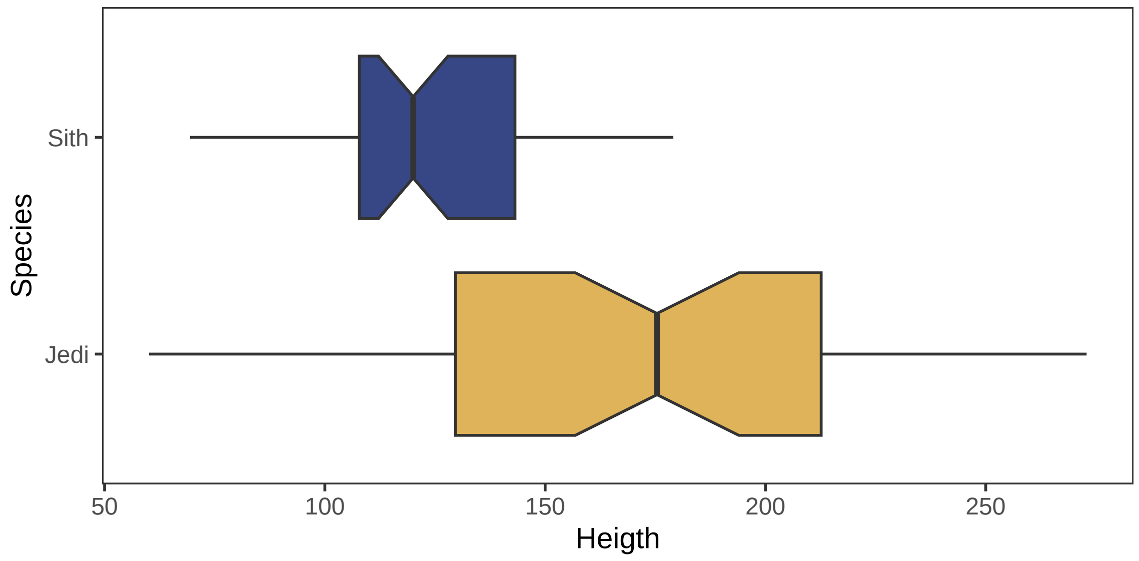

Figure 4: Boxplot of Jedi and Sith heigth using ggplot.

C. Save the plots with the 'png', 'pdf', and 'svg' formats.In which situation would you use which file format?

```
# Here we save only ggplot fig. using ggsave function
## We can do the same opening and closing  different developer such as pdf; dev.off
save_here <- './figures/'
save_type <- c('pdf', 'svg', 'png')

# Use lapply to save all 3
lapply(1:3, function(i) {
  ggsave(filename = paste0(save_here, 'Jedi_vs_Sith.', save_type[i]),
        device = save_type[i], width = 6, height = 4)
})
```

PNG (Portable Network Graphics):

Use Case: PNG is a raster graphics format, which means it's composed of pixels. It's a good choice when you want to save a plot as a static image with high-quality detail.

Situation: Use PNG when you need to share plots online, include them in reports or presentations, or display them on a website. PNG is suitable for situations where you want to preserve the appearance of the plot, and you don't need to edit it further as it is not easily editable once saved.

PDF (Portable Document Format):

Use Case: PDF is a vector format, which means it stores plots as scalable objects rather than pixels. It's great for documents and publications where you need high-quality graphics that can be resized without loss of quality.

Situation: Use PDF when you are creating documents, reports, or presentations where the plot needs to be printed or viewed at different sizes. PDFs are also useful when you want to keep the ability to edit or extract elements from the plot later, as vector graphics are more easily editable in programs like Adobe Illustrator.

SVG (Scalable Vector Graphics):

Use Case: SVG is a vector format designed specifically for scalable graphics. It's ideal for plots that need to be highly responsive and adapt to different screen sizes or resolutions.

Situation: Use SVG when you want to embed plots in web pages, where they may be displayed on devices with various screen sizes (e.g., desktops, tablets, smartphones). SVG files are smaller in size compared to PNG and maintain sharpness at any resolution. They are also suitable for situations where you might need to modify or animate the plot using tools that support SVG editing.

2. Load the gene expression data matrix from the 'microarray_data.tab'.

A. How big is the matrix in terms of rows and columns?

B. Count the missing values per gene and visualize this result.

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

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

```r
# Load library you'll nned
library(naniar)

# A.
# Read in magic_guys and look at the summary by species
# check dimensions
microarray <- read.csv('./data/microarray_data.tab', sep = '\t')
dim(microarray)
```

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

```r
# Count the missing values per gene and visualize
sum_na <- colSums(is.na(microarray))

# B.
# Lets plot this nice figure :)
# Will use original data for that with fun from 'naniar' pck
gg_before <- gg_miss_var(microarray, show_pct = TRUE) +
                  labs(x = 'Genes', y = 'Missing values (%)')

# C.
# Genes with certain percent of missing values - use base
ten_perc <- names(microarray)[colMeans(is.na(microarray)*100) <= 10]
twenty_perc <- names(microarray)[colMeans(is.na(microarray)*100) <= 20]
fifty_perc <- names(microarray)[colMeans(is.na(microarray)*100) <= 50]

# D.
# Replace missing values with the average expression value for each gene
microarray <- microarray %>% mutate(across(everything(.),
                                ~ifelse(is.na(.), mean(., na.rm = TRUE), .)))

# Will use original data for that with fun from 'naniar' pck
gg_after <- gg_miss_var(microarray, show_pct = TRUE) +
                  labs(x = 'Genes', y = 'Missing values (%)')

#Combine the plots with ggarrange and plot
ggarrange(gg_before, gg_after)
```

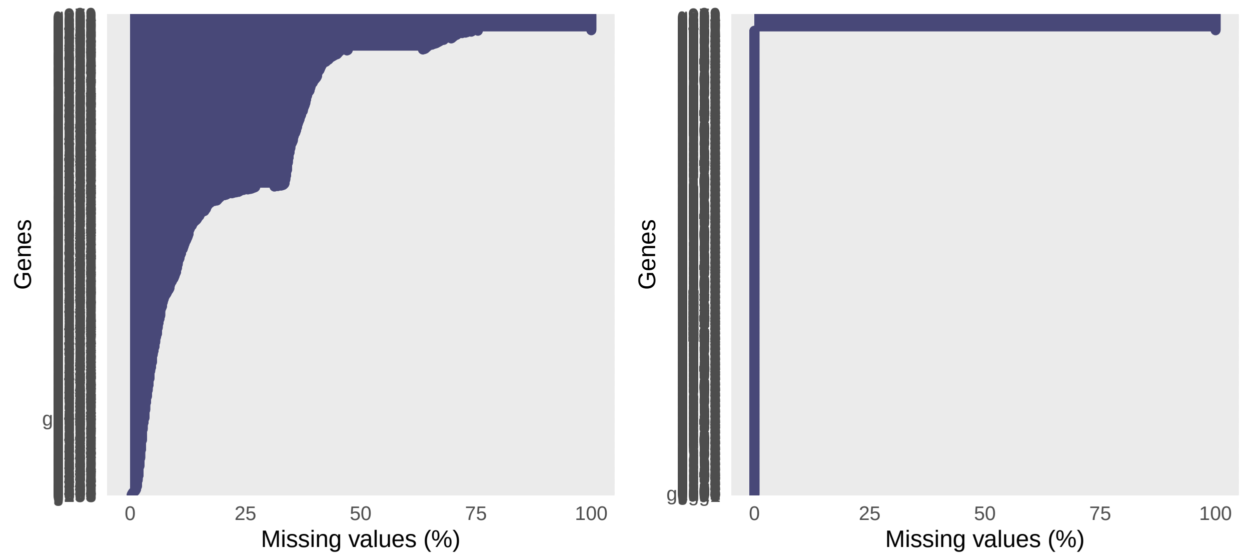

Figure 5: Percentage of missing values in MicroArray data set before (left) and after (rigth) missing values replacement.

Comment: It seems some genes do not express at all!

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

```r
# Combine Type and treatment in one column; use facet wrap for Plant and Place
CO2 %>% unite(Place, c(Type, Treatment), sep = '-') %>%
        ggplot(aes(conc, uptake)) +
        geom_line() + geom_point() +
        facet_wrap(Plant~Place, scales = 'free_x', ncol = 6) +
        labs(y = expression(paste(CO[2], ' uptake')),
             x = expression(paste('Ambient ', CO[2], ' concentration'))) +
        theme_bw() + theme(panel.grid.minor.x = element_blank())
```

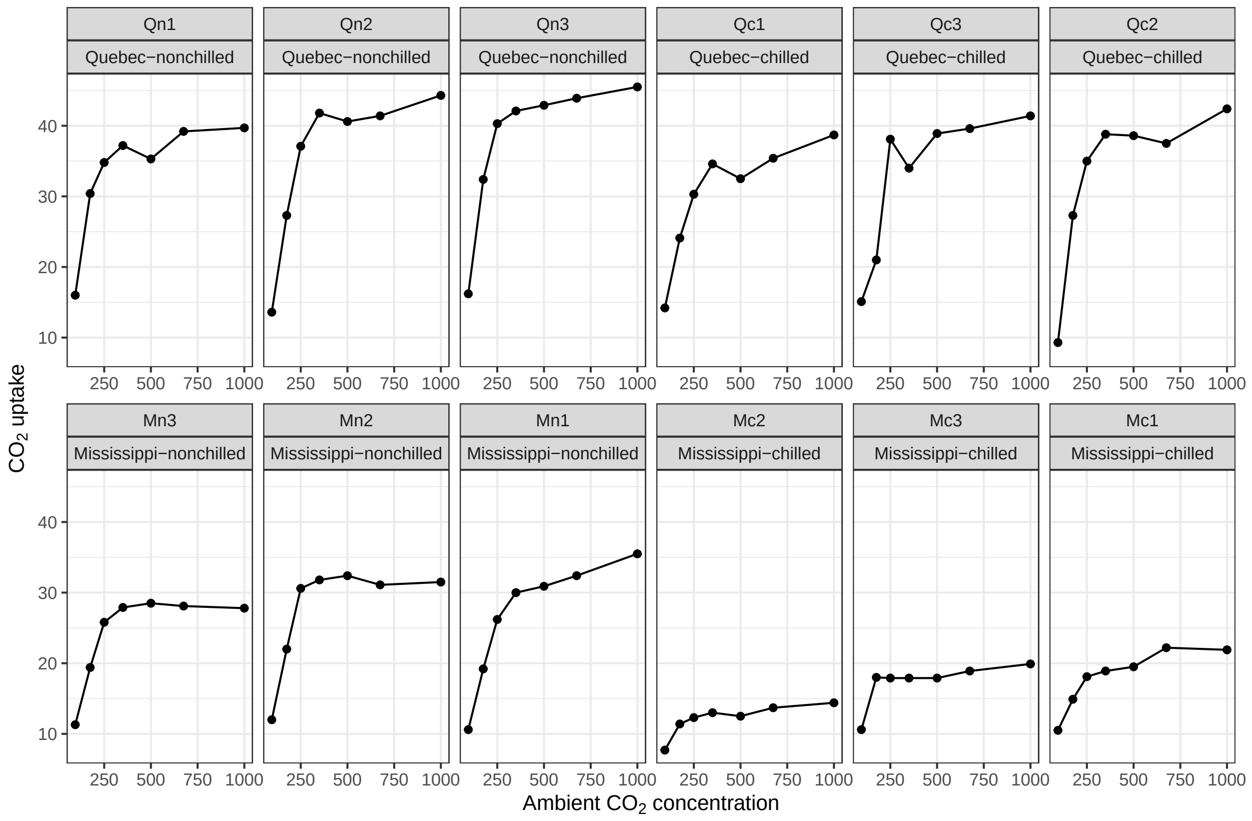

Figure 6: The CO2 uptake of six plants from Quebec and six plants from Mississippi was measured at several levels of ambient CO2 concentration. Half the plants of each type were chilled overnight before the experiment was conducted.

Task 7 - Tidybiology package

1. Install the Tidybiology package, which includes the data 'chromosome' and 'proteins'.

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.

```r
# Install tidybiology
# devtools::install_github("hirscheylab/tidybiology")
library(tidybiology)

# attach chromosome data
data("chromosome")

# summarize target variables
chromosome_summary <- chromosome %>% select(variations, protein_codinggenes, mi_rna) %>%
               pivot_longer(cols = everything(), names_to = 'Variable') %>%
               mutate(Variable = case_when(Variable == 'mi_rna' ~ 'miRNA',
                                 Variable == 'protein_codinggenes' ~ 'Protein coding genes',
                                 TRUE ~ 'Variations')) %>%
               group_by(Variable) %>%
               summarise(mean = mean(value), median = median(value), max = max(value)) %>%
               mutate(mean = round(mean)) %>%
               arrange(max)


# Print table
kable(chromosome_summary, caption = 'Chromosome dataset.',
      format = 'latex', booktabs = T) %>%
            kable_styling(latex_options = c('striped',"HOLD_position"))
```

Table 3: Chromosome dataset.

Variable	mean	median	max
miRNA	73	75	134
Protein coding genes	850	836	2058
Variations	6484572	6172346	12945965

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

```
# Plot distribution of the chromosome size
# Change plot theme and axis labels
ggplot(chromosome, aes(x = length_mm)) +
        geom_histogram(binwidth = 10, fill = "lightgray", color = "black") +
        labs(x = "Chromosome Length (mm)",
             y = "Frequency") +
        theme_bw() +
        theme(panel.grid.major = element_blank(),
              panel.grid.minor = element_blank())
```

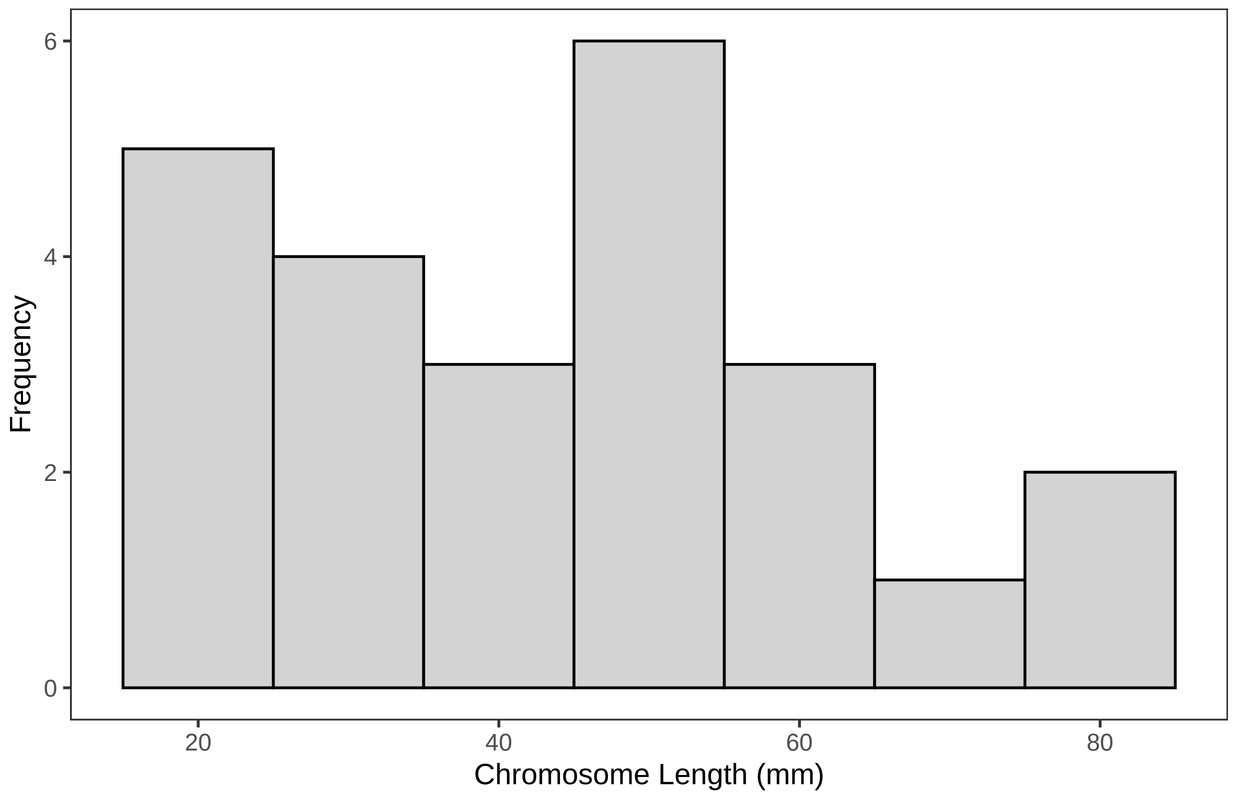

Figure 7: Chromosome data set. Distribution of Chromosome Sizes.

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.

```
# C1
# Plot length and number of protein coding genes
# Add the line and spearman correlation
# Change plot theme and axis labels
ggplot(chromosome, aes(x = length_mm, y = protein_codinggenes)) +
        geom_point() +
        geom_smooth(method = 'lm', fill = 'gray', alpha = 0.3) +
        stat_cor(method = 'spearman', cor.coef.name = 'rho') +
        labs(x = 'Chromosome size (mm)', y = 'Protein coding genes') +
        theme_bw() +
        theme(panel.grid.major = element_blank(),
              panel.grid.minor = element_blank())
```

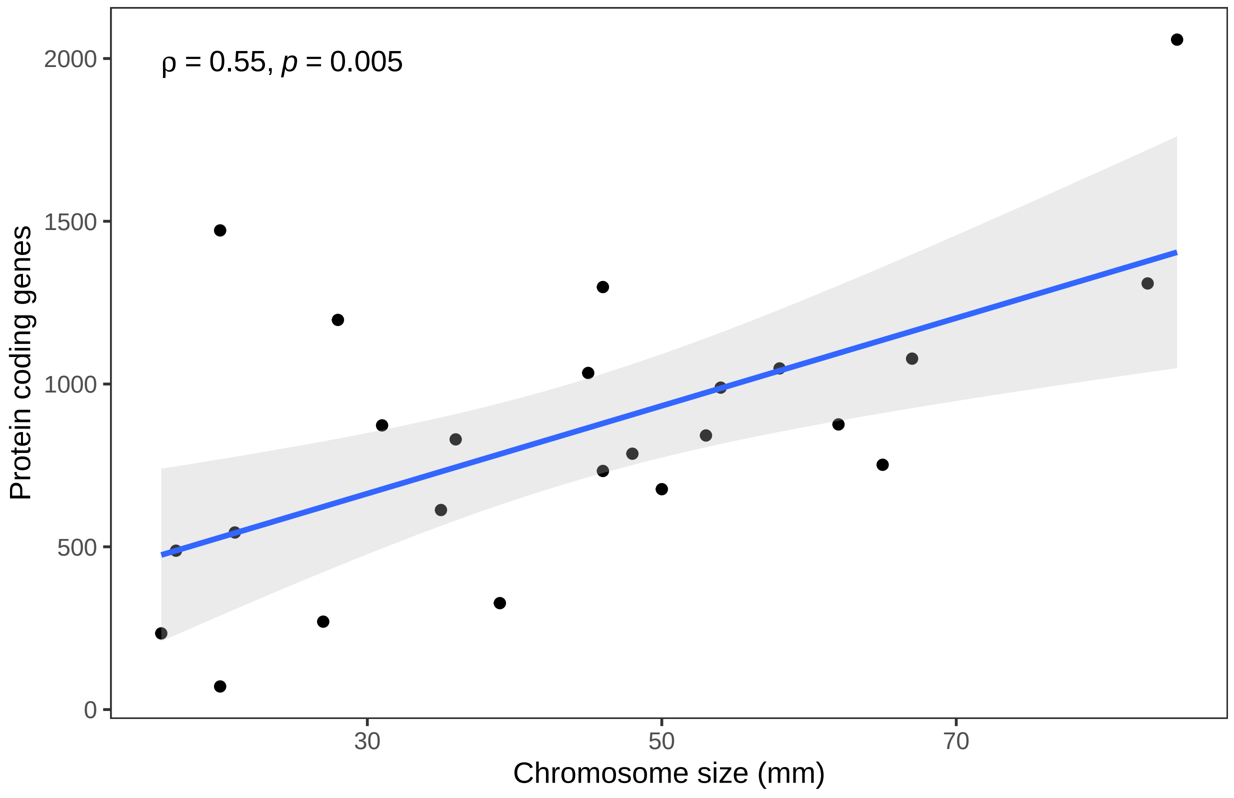

Figure 8: Chromosome data set. Association between chromosome length and number of protein coding genes.

```r
# C2
# plot length and number of miRNA
# add the line and spearman correaltion
# change plot theme and axis labels
ggplot(chromosome, aes(x = length_mm, y = mi_rna)) +
        geom_point() +
        stat_cor(lmethod = 'spearman', cor.coef.name = 'rho', label.y = 140) +
        geom_smooth(method = 'lm', fill = 'gray', alpha = 0.3) +
        labs(x = 'Chromosome size (mm)', y = 'miRNA', ) +
        theme_bw() +
        theme(panel.grid.major = element_blank(),
              panel.grid.minor = element_blank())
```

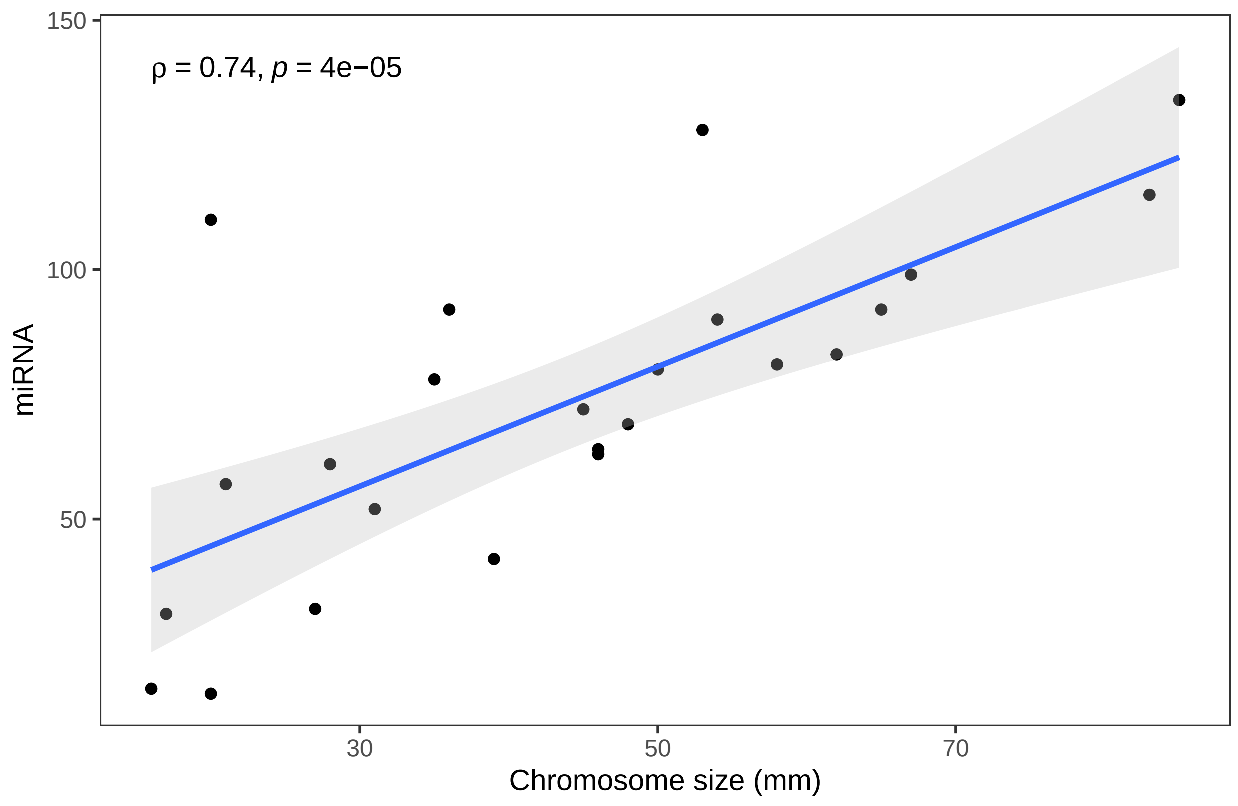

Figure 9: Chromosome data set. Association between chromosome length and miRNA.

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.

```r
# Attach the data
data("proteins")

# Summary stats of protein data
protein_summary <- proteins %>%
          select(length, mass) %>%
          summarise(across(everything(),
                    list(mean = mean, median = median, max = max)))

# Print table
kable(protein_summary %>%
                      pivot_longer(cols = everything()) %>%
                      separate(name, c('Info', 'group')) %>%
                      pivot_wider(names_from = 'group', values_from = 'value'),
              caption = 'Protein data set.',
              format = 'latex', booktabs = T) %>%
      kable_styling(latex_options = c('striped',"HOLD_position"))
```

Table 4: Protein data set.

Info	mean	median	max
length	557.1603	414.0	34350
mass	62061.3791	46140.5	3816030

```r
# Plot length vs mass
ggplot(proteins, aes(x = length, y = mass)) +
        geom_point(aes(color = length), size = 1.5) +
        scale_color_gradient(low = "blue", high = "red") +
        labs(title = ,
             x = "Protein length",
             y = "Protein mass") +
        theme_bw() +
        theme(legend.position = "none",
              panel.grid.major = element_blank(),
              panel.grid.minor = element_blank()) +
        geom_text_repel(aes(label = ifelse(mass > max(mass) * 0.20, gene_name, NA)),
                        size = 2.5,
                        max.overlaps = 15) +
        geom_text(data = protein_summary,
                  aes(x = length_mean, y = mass_max, label = "Median protein length"),
                  angle = 90, vjust = -1.5,
                  hjust = 1, size = 2) +
        geom_text(data = protein_summary,
                  aes(x = length_max, y = mass_median, label = "Median protein mass"),
                  vjust = -1, hjust = 1, size = 2) +
        geom_hline(yintercept = protein_summary$mass_median,
                   linetype = "dashed",
                   color = "gray") +
        geom_vline(xintercept = protein_summary$length_median,
                   linetype = "dashed",
                   color = "gray")
```

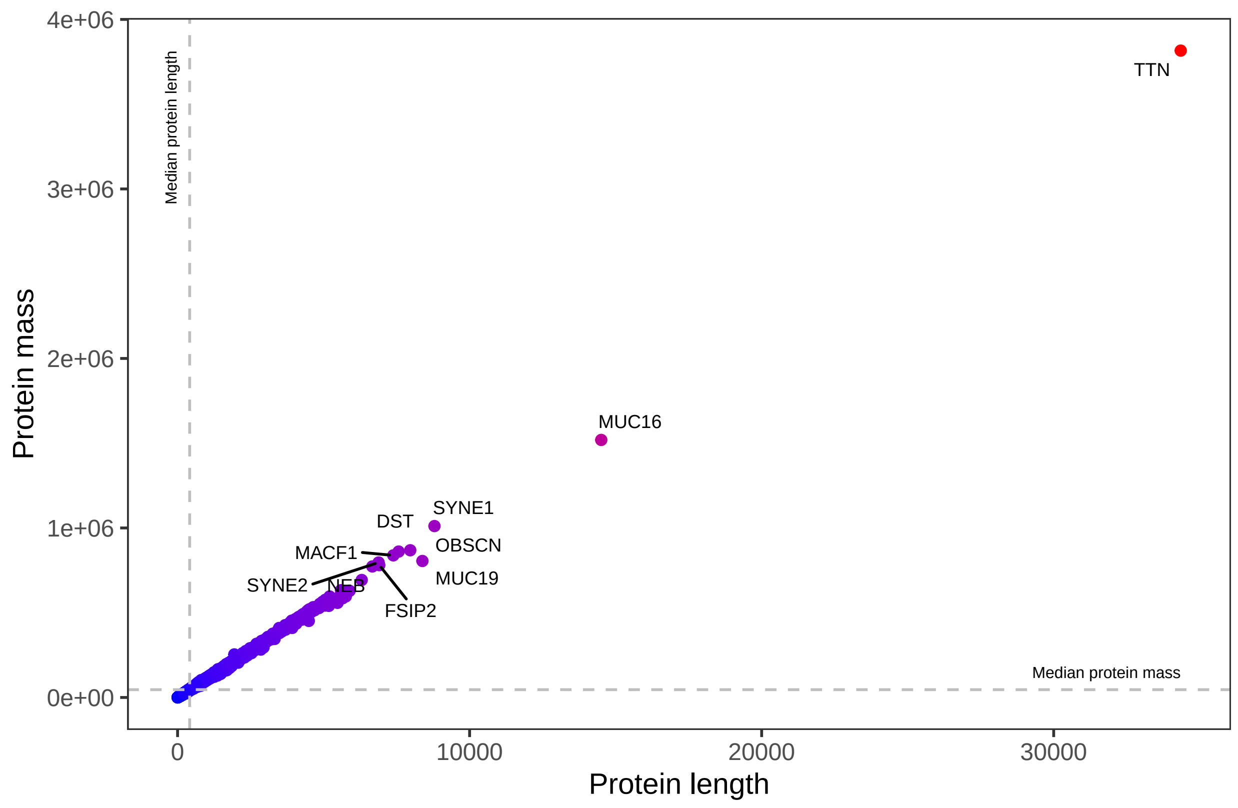

Figure 10: Protein data set. Relationship between Protein Length and Protein Mass.