Assignment 1

Wanzhe Xu Laboratory of Functional Analysis in silico, Human Genome Center, The Institute of Medical Science, The University of Tokyo

2023-03-06

# Task 1 - Literature

## 1. Read the research article of the hands-on working group you are assigned to (see file “Student Groups.pdf” in shared folder General course material).

## 2. Answer the following questions

### a. What is the medically relevant insight from the article?

Answer: This paper provides an unbiased approach to integrate disparate single-cell transcriptome datasets for more accurate and reliable results. In medical research, single-cell transcriptome analysis has become an important tool for studying complex diseases and developing personalized therapeutic strategies. However, due to the high heterogeneity and technical bias of single-cell transcriptome data, there are large differences between different datasets, which makes the integration and interpretation of data difficult. With the unbiased integration approach proposed in this paper, different cellular subpopulations and differentially expressed genes can be more accurately identified while reducing batch effects and technical biases, thus deepening the understanding of disease pathogenesis and contributing to the discovery of new therapeutic targets and drugs.

### b. Which genomics technology/ technologies were used?

Answer: The technique used in this study is scRNA-seq, which allows researchers to sequence the transcriptome of individual cells to gain insights into gene expression patterns and cellular heterogeneity at the single-cell level.

## 3. Further related research questions

### a. List and explain at least three questions/ hypotheses you can think of that extend the analysis presented in the paper.

Answer: Question 1: How to deal with the effects of technical biases or batch effects? During the acquisition and processing of individual cell transcriptome data, it may be affected by various technical factors, such as sequencing depth, PCR amplification bias, etc. These technical factors may bias the integration results, so it is possible to explore how to correct the effects of these technical biases during the integration process to obtain more accurate results.

Question 2: How are differences between cell types handled when integrating individual cell transcriptomes? The approach in this paper focuses on duplicates of individual cells, but for samples with different cell types, there may be differences in gene expression levels between each cell type.

Hypotheses 3: Different integration algorithms may produce different results. Is it possible to reduce bias, such as the effect of noise, by algorithmic integration.

### b. [Optional] Devise a computational analysis strategy for (some of) the listed questions under 3a.

Answer: There may be unknown batch effects in single cell transcriptome data. To avoid the influence of batch effects on integration results, some batch effect correction methods can be used, such as ComBat or Limma.

# Task 4 - R basic operations

## What is the square root of 10?

cat(log2(32))

output: 3.162278

## What is the logarithm of 32 to the base 2?

cat(sum(1:1000))

output: 5

## What is the sum of the numbers from 1 to 1000?

cat(sum(1:1000))

output: 500500

1. **What is the sum of all even numbers from 2 to 1000?**

sum\_even <- 0

for (i in seq(2, 1000, by=2)) { # for loop through even numbers

sum\_even <- sum\_even + i # add each even number to the sum

}

cat(sum\_even)

## How many pairwise comparisons are there for 100 genes?

# The seq function is used to create an equal series from 2 to 1000 with a step size of 2, so that all even numbers are obtained.

sum\_even <- sum(seq(from = 2, to = 1000, by = 2))

sum\_even

output: 250500

## And how many ways to arrange 100 genes in triples?

# The NCR formula is used when some sort of ordering is done without considering the order of things. the R code is implemented as follows.

n <- 100

r <- 3

num\_triplets <- factorial(n) / (factorial(r) \* factorial(n - r))

print(num\_triplets)

output: 161700

# Task 5 - Using R example datasets

## 1. Use the R internal CO2 dataset (“data(CO2)”).

data(CO2)  
help(CO2)

## 2. Describe briefly the content of the CO2 dataset using the help function.

Answer: The CO2 dataset is a time series object containing atmospheric CO2 concentration data collected at the Mauna Loa Observatory in Hawaii from 1959 to 1997. The data set has two variables, which are

CO2: atmospheric carbon dioxide concentration. Plant: the type of plant used for the experiment.

There are 468 observations in the dataset.

## 3. What is the average and median CO2 uptake of the plants from Quebec and Mississippi?

# import lib dplyr first  
# install.packages("dplyr")  
library(dplyr)  
  
CO2 %>%  
 # select only Quebec and Mississippi  
 filter(Type %in% c("Quebec", "Mississippi")) %>%   
 # group by Type  
 group\_by(Type) %>%  
 # compute mean and median  
 summarize(mean\_uptake = mean(uptake), median\_uptake = median(uptake))   
## # A tibble: 2 × 3  
## Type mean\_uptake median\_uptake  
## <fct> <dbl> <dbl>  
## 1 Quebec 33.5 37.2  
## 2 Mississippi 20.9 19.3

The results are shown in the table and the mean for Quebec is 33.54286 and the median is 37.15. The mean for Mississippi is 20.88333 and the median is 19.30.

## 4. [Optional] In the “airway” example data from Bioconductor, how many genes are expressed in each sample? How many genes are not expressed in any sample?

#install packge pasilla for R version 4 or more:  
if (!requireNamespace("BiocManager", quietly = TRUE))  
install.packages("BiocManager")  
BiocManager::install("pasilla", version = "3.16")  
# Install packdge airway:  
if (!require("airway"))  
BiocManager::install("airway")  
  
# Load the "pasilla" package:  
# library(pasilla)  
  
# Load the "airway" dataset  
data(airway)  
  
# Extracts the data from the SingleCellExperiment object and converts it into a matrix  
airway\_mat <- as.matrix(assay(airway))  
  
# Remove missing values  
airway\_mat <- na.omit(airway\_mat)  
  
# Calculate the number of expressed and non-expressed genes  
expressed\_genes <- sum(rowSums(airway\_mat) > 0)  
not\_expressed\_genes <- sum(rowSums(airway\_mat) == 0)  
  
# print result  
cat("Number of expressed genes:", expressed\_genes, "\n")  
## Number of expressed genes: 33469  
cat("Number of not expressed genes:", not\_expressed\_genes, "\n")  
## Number of not expressed genes: 30633

# Task 6 - R Functions

## 1. Write a function that calculates the ratio of the mean and the median of a given vector.

mean\_median\_ratio <- function(vector) {  
 # Calculate mean and median  
 mean <- mean(vector)  
 median <- median(vector)  
   
 # Calculate ratio  
 ratio <- mean / median  
   
 # Return ratio  
 return(ratio)  
}

## 2. Write a function that ignores the lowest and the highest value from a given vector and calculate the mean.

trimmed\_mean <- function(vector) {  
 # Remove lowest and highest values  
 trimmed\_vector <- vector[-c(which.min(vector), which.max(vector))]  
   
 # Calculate mean of trimmed vector  
 mean(trimmed\_vector)  
}

## 3. Read about piping from here:<https://r4ds.had.co.nz/pipes.html#pipes> (you don’t have to learn everything, a basic understanding of the usage is enough). Write a short (max. 300 characters, no spaces) explanation of why, how, and when not to use pipes.

Pipes in R are used to chain together multiple operations, making code more readable and efficient. Pipes allow data to flow from one operation to the next, reducing the need for intermediate variables. However, pipes can be difficult to read when they become too complex or are nested too deeply. Additionally, some operations may not work well with pipes, such as functions that require multiple arguments or functions that require data to be grouped or sorted in a particular way. Therefore, pipes should be used judiciously, and not at the expense of code readability or functionality.

## 4. Familiarize yourself with the apply-family of functions (apply, lapply, sapply etc.) <http://uc-r.github.io/apply_family> Write a short explanation (max. 300 characters, no spaces) of why they could be useful in your work.

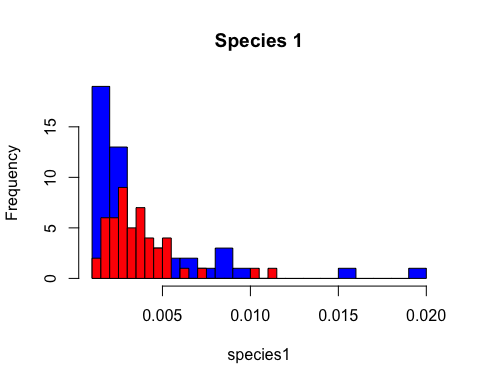
The apply-family of functions in R (apply, lapply, sapply, etc.) can be useful in my work because they allow for efficient and streamlined manipulation of data in arrays and lists. These functions provide a simpler and more concise way to apply a function to subsets of a dataset or to apply a function across multiple datasets, reducing the amount of repetitive code. The apply-family functions also allow for the output to be returned in various formats, such as a list, vector, or matrix, depending on the needs of the analysis.

# Task 7 - Basic visualization with R

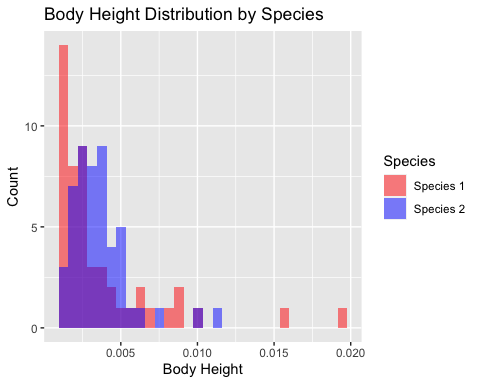
## 1. Compare the distributions of the body heights of the two species from the ‘magic\_guys.csv’ dataset graphically

### 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.

# Load the data  
magic\_guys <- read.csv("magic\_guys.csv")  
library(ggplot2)  
# Calculate body height  
body\_height <- magic\_guys$weight / magic\_guys$length^2  
  
# Subset the data by species  
species1 <- subset(body\_height, magic\_guys$species == "jedi")  
species2 <- subset(body\_height, magic\_guys$species == "sith")  
#print(species1)  
#print(species2)  
type(species1)  
## [1] "double"  
# Use the basic hist() function to plot histograms of the body height data for both species  
hist(species1, breaks = 20, col = "blue", main = "Species 1")  
hist(species2, breaks = 20, col = "red", add = TRUE)

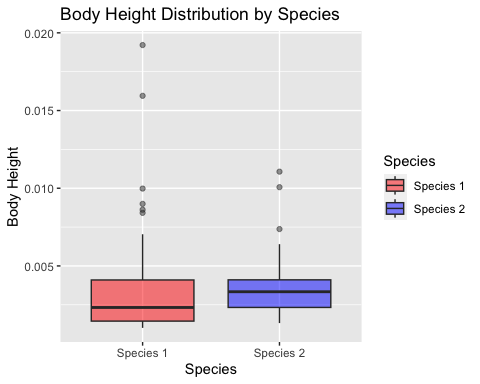


# Use ggplot2 and geom\_histogram() to plot histograms of the body height data for both species  
# Load the necessary library  
library(ggplot2)  
  
# Create a new data frame with the body heights of both species  
new\_df <- data.frame(Body\_Height = c(species1, species2),  
 Species = factor(rep(c("Species 1", "Species 2"), c(length(species1), length(species2)))))  
  
# Create the histogram plot  
ggplot(new\_df, aes(x=Body\_Height, fill=Species)) +  
 geom\_histogram(alpha=0.5, position="identity", bins=30) +  
 labs(title="Body Height Distribution by Species", x="Body Height", y="Count") +  
 scale\_fill\_manual(values=c("red", "blue"))



### 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’.

# Create the boxplot plot  
ggplot(new\_df, aes(x=Species, y=Body\_Height, fill=Species)) +  
 geom\_boxplot(alpha=0.5, position="dodge") +  
 labs(title="Body Height Distribution by Species", x="Species", y="Body Height") +  
 scale\_fill\_manual(values=c("red", "blue"))



### c. Save the plots with the ‘png’, ‘pdf’, and ‘svg’ formats. In which situation would you use which file format?

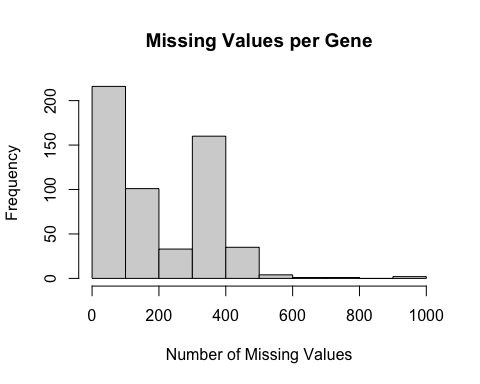
# Save the plot  
ggsave("temp\_save.png", plot = last\_plot(), width = 6, height = 4, dpi = 300)  
ggsave("temp\_save.pdf", plot = last\_plot(), width = 6, height = 4, dpi = 300)  
# install.packages("svglite") first  
library(svglite)  
ggsave("temp\_save.svg", plot = last\_plot(), width = 6, height = 4, dpi = 300, device = "svg")

If we need a lossless, web-compatible format that supports transparency, we might choose PNG. if we need a high-quality vector format that supports CMYK colors and can be embedded in a document, we might choose PDF. if we need a scalable, web-compatible vector format, we might choose SVG. ## 2. Load the gene expression data matrix from the ‘microarray\_data.tab’ dataset provided in the shared folder, it is a big tabular separated matrix. ### a. How big is the matrix in terms of rows and columns?

# Load the data  
data <- read.table("microarray\_data.tab", header = TRUE, sep = "\t")  
  
# Get the dimensions of the matrix  
dim(data)  
## [1] 553 1000

As shown, the matrix has 553rows and 1000columns. ### b. Count the missing values per gene and visualize this result.

# Count the missing values per gene  
missing\_values <- apply(data, 1, function(x) sum(is.na(x)))  
missing\_values  
## [1] 72 226 72 74 126 87 273 95 92 85 64 56 103 78 66  
## [16] 74 586 455 493 67 287 85 72 329 406 397 359 347 355 351  
## [31] 389 375 366 348 356 355 348 358 375 359 393 346 367 361 353  
## [46] 352 352 494 347 396 398 357 379 347 356 382 568 362 366 364  
## [61] 356 371 353 352 355 381 365 350 361 353 351 349 64 67 454  
## [76] 415 406 412 363 372 381 359 390 385 380 378 384 377 394 401  
## [91] 382 383 369 387 388 379 387 383 387 405 415 406 445 462 410  
## [106] 418 407 126 409 414 116 439 433 89 430 366 67 393 375 403  
## [121] 375 400 403 356 84 387 363 355 351 356 138 375 360 360 547  
## [136] 379 368 377 396 379 384 383 393 375 381 390 400 372 391 394  
## [151] 415 369 387 398 400 68 389 392 382 389 377 381 395 393 499  
## [166] 387 83 388 368 284 385 372 393 375 98 198 329 170 110 68  
## [181] 91 457 380 57 64 69 179 65 79 72 92 67 359 63 293  
## [196] 97 69 81 66 62 62 115 124 90 81 68 66 90 371 64  
## [211] 69 64 67 376 375 193 103 54 122 94 69 62 76 54 61  
## [226] 183 61 52 56 58 59 56 54 64 56 57 344 110 93 116  
## [241] 557 64 111 64 81 62 59 64 177 790 304 100 82 61 95  
## [256] 62 60 66 53 124 61 159 161 122 62 93 78 51 69 121  
## [271] 72 61 182 63 97 101 119 60 84 195 58 282 54 65 405  
## [286] 114 74 108 96 102 121 79 58 116 108 93 68 58 60 56  
## [301] 63 61 112 108 62 86 62 319 60 53 127 128 81 67 72  
## [316] 81 75 72 96 122 69 106 209 60 84 85 116 207 85 76  
## [331] 180 147 191 96 677 387 379 368 384 1000 369 420 373 356 349  
## [346] 350 348 363 358 356 404 354 371 362 363 356 356 435 362 360  
## [361] 367 445 358 365 357 361 376 354 360 60 291 124 331 109 108  
## [376] 109 280 88 102 89 83 221 197 92 100 105 177 157 105 102  
## [391] 139 109 172 106 325 275 100 85 88 105 210 89 78 83 100  
## [406] 102 88 90 81 81 76 85 82 83 79 100 101 281 125 104  
## [421] 90 88 86 409 87 82 324 261 78 80 87 103 93 81 242  
## [436] 232 210 77 170 199 230 185 210 239 335 338 242 266 171 284  
## [451] 219 177 121 96 302 118 112 113 96 100 426 121 84 83 155  
## [466] 95 81 87 114 83 77 87 93 80 90 76 96 130 102 79  
## [481] 78 90 103 95 93 89 94 88 100 91 87 100 131 92 118  
## [496] 90 100 247 80 179 97 116 165 106 109 238 99 75 104 80  
## [511] 107 97 119 472 398 117 146 132 267 238 122 324 101 91 79  
## [526] 90 104 94 181 85 100 88 91 223 82 241 110 82 78 92  
## [541] 1000 80 78 90 99 251 103 256 90 133 104 86 86  
# Visualize the result  
hist(missing\_values, main = "Missing Values per Gene", xlab = "Number of Missing Values")



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

# Calculate the percentage of missing values for each gene  
percent\_missing <- apply(is.na(data), 1, mean) \* 100  
  
# Find the genes with more than X% missing values, where X is 10%, 20%, and 50%  
X\_values <- c(10, 20, 50)  
for (X in X\_values) {  
 missing\_genes <- rownames(data)[percent\_missing > X]  
 cat("Genes with more than", X, "% missing values:\n")  
 print(missing\_genes)  
}  
## Genes with more than 10 % missing values:  
## [1] "2" "5" "7" "13" "17" "18" "19" "21" "24" "25" "26" "27"   
## [13] "28" "29" "30" "31" "32" "33" "34" "35" "36" "37" "38" "39"   
## [25] "40" "41" "42" "43" "44" "45" "46" "47" "48" "49" "50" "51"   
## [37] "52" "53" "54" "55" "56" "57" "58" "59" "60" "61" "62" "63"   
## [49] "64" "65" "66" "67" "68" "69" "70" "71" "72" "75" "76" "77"   
## [61] "78" "79" "80" "81" "82" "83" "84" "85" "86" "87" "88" "89"   
## [73] "90" "91" "92" "93" "94" "95" "96" "97" "98" "99" "100" "101"  
## [85] "102" "103" "104" "105" "106" "107" "108" "109" "110" "111" "112" "113"  
## [97] "115" "116" "118" "119" "120" "121" "122" "123" "124" "126" "127" "128"  
## [109] "129" "130" "131" "132" "133" "134" "135" "136" "137" "138" "139" "140"  
## [121] "141" "142" "143" "144" "145" "146" "147" "148" "149" "150" "151" "152"  
## [133] "153" "154" "155" "157" "158" "159" "160" "161" "162" "163" "164" "165"  
## [145] "166" "168" "169" "170" "171" "172" "173" "174" "176" "177" "178" "179"  
## [157] "182" "183" "187" "193" "195" "202" "203" "209" "214" "215" "216" "217"  
## [169] "219" "226" "237" "238" "240" "241" "243" "249" "250" "251" "260" "262"  
## [181] "263" "264" "270" "273" "276" "277" "280" "282" "285" "286" "288" "290"  
## [193] "291" "294" "295" "303" "304" "308" "311" "312" "320" "322" "323" "327"  
## [205] "328" "331" "332" "333" "335" "336" "337" "338" "339" "340" "341" "342"  
## [217] "343" "344" "345" "346" "347" "348" "349" "350" "351" "352" "353" "354"  
## [229] "355" "356" "357" "358" "359" "360" "361" "362" "363" "364" "365" "366"  
## [241] "367" "368" "369" "371" "372" "373" "374" "375" "376" "377" "379" "382"  
## [253] "383" "386" "387" "388" "389" "390" "391" "392" "393" "394" "395" "396"  
## [265] "400" "401" "406" "417" "418" "419" "420" "424" "427" "428" "432" "435"  
## [277] "436" "437" "439" "440" "441" "442" "443" "444" "445" "446" "447" "448"  
## [289] "449" "450" "451" "452" "453" "455" "456" "457" "458" "461" "462" "465"  
## [301] "469" "478" "479" "483" "493" "495" "498" "500" "502" "503" "504" "505"  
## [313] "506" "509" "511" "513" "514" "515" "516" "517" "518" "519" "520" "521"  
## [325] "522" "523" "527" "529" "534" "536" "537" "541" "546" "547" "548" "550"  
## [337] "551"  
## Genes with more than 20 % missing values:  
## [1] "2" "7" "17" "18" "19" "21" "24" "25" "26" "27" "28" "29"   
## [13] "30" "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41"   
## [25] "42" "43" "44" "45" "46" "47" "48" "49" "50" "51" "52" "53"   
## [37] "54" "55" "56" "57" "58" "59" "60" "61" "62" "63" "64" "65"   
## [49] "66" "67" "68" "69" "70" "71" "72" "75" "76" "77" "78" "79"   
## [61] "80" "81" "82" "83" "84" "85" "86" "87" "88" "89" "90" "91"   
## [73] "92" "93" "94" "95" "96" "97" "98" "99" "100" "101" "102" "103"  
## [85] "104" "105" "106" "107" "109" "110" "112" "113" "115" "116" "118" "119"  
## [97] "120" "121" "122" "123" "124" "126" "127" "128" "129" "130" "132" "133"  
## [109] "134" "135" "136" "137" "138" "139" "140" "141" "142" "143" "144" "145"  
## [121] "146" "147" "148" "149" "150" "151" "152" "153" "154" "155" "157" "158"  
## [133] "159" "160" "161" "162" "163" "164" "165" "166" "168" "169" "170" "171"  
## [145] "172" "173" "174" "177" "182" "183" "193" "195" "209" "214" "215" "237"  
## [157] "241" "250" "251" "282" "285" "308" "323" "328" "335" "336" "337" "338"  
## [169] "339" "340" "341" "342" "343" "344" "345" "346" "347" "348" "349" "350"  
## [181] "351" "352" "353" "354" "355" "356" "357" "358" "359" "360" "361" "362"  
## [193] "363" "364" "365" "366" "367" "368" "369" "371" "373" "377" "382" "395"  
## [205] "396" "401" "418" "424" "427" "428" "435" "436" "437" "441" "443" "444"  
## [217] "445" "446" "447" "448" "450" "451" "455" "461" "498" "506" "514" "515"  
## [229] "519" "520" "522" "534" "536" "541" "546" "548"  
## Genes with more than 50 % missing values:  
## [1] "17" "57" "135" "241" "250" "335" "340" "541"

### d. Replace the missing values by the average expression value for the particular gene. (Note: Imputing data has to be used with caution!)

# Replace missing values with the average expression value for the particular gene  
for (gene in colnames(data)) {  
 gene\_values <- data[,gene]  
 missing\_indices <- is.na(gene\_values)  
 if (any(missing\_indices)) {  
 avg\_value <- mean(gene\_values, na.rm=TRUE)  
 gene\_values[missing\_indices] <- avg\_value  
 data[,gene] <- gene\_values  
 }  
}  
head(data)  
## g1 g2 g3 g4 g5 g6 g7  
## 1 1.80200000 0.1656927 -0.182 1.31200000 3.49700000 0.4390000 0.777  
## 2 0.02547518 0.1656927 7.693 -0.06731957 0.19300000 -1.3830000 -1.309  
## 3 1.07900000 0.1656927 1.556 1.65200000 -0.01812288 0.4600000 0.715  
## 4 3.60700000 0.1656927 1.914 -0.06731957 1.40000000 1.1090000 2.143  
## 5 -1.70000000 0.1656927 0.943 -0.06731957 -0.17000000 -0.1571338 -0.041

# Task 8

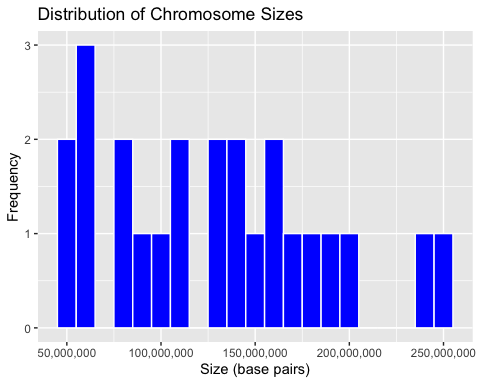
## 1. Install the Tidybiology package, which includes the data ‘chromosome’ and ‘proteins’ devtools::install\_github(“hirscheylab/tidybiology”)

### 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.

# Load the chromosome data  
library(ggplot2)  
data(chromosome)  
## Warning in data(chromosome): data set 'chromosome' not found  
library(tidybiology)  
# Extract summary statistics for variations, protein coding genes, and miRNAs  
chromosome %>%  
 summarize(  
 mean\_variations = mean(variations),  
 median\_variations = median(variations),  
 max\_variations = max(variations),  
 mean\_protein\_coding\_genes = mean(protein\_codinggenes),  
 median\_protein\_coding\_genes = median(protein\_codinggenes),  
 max\_protein\_coding\_genes = max(protein\_codinggenes),  
 mean\_miRNAs = mean(mi\_rna),  
 median\_miRNAs = median(mi\_rna),  
 max\_miRNAs = max(mi\_rna)  
 )  
## # A tibble: 1 × 9  
## mean\_variati…¹ media…² max\_v…³ mean\_…⁴ media…⁵ max\_p…⁶ mean\_…⁷ media…⁸ max\_m…⁹  
## <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl> <int>  
## 1 6484572. 6172346 1.29e7 850. 836 2058 73.2 75 134  
## # … with abbreviated variable names ¹​mean\_variations, ²​median\_variations,  
## # ³​max\_variations, ⁴​mean\_protein\_coding\_genes, ⁵​median\_protein\_coding\_genes,  
## # ⁶​max\_protein\_coding\_genes, ⁷​mean\_miRNAs, ⁸​median\_miRNAs, ⁹​max\_miRNAs

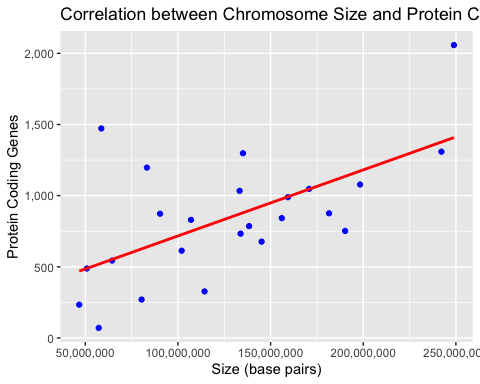
### b. How does the chromosome size distribute? Plot a graph that helps to visualize this by using ggplot2 package functions.

# Create a histogram of chromosome sizes  
ggplot(chromosome, aes(x = basepairs)) +  
 geom\_histogram(binwidth = 10000000, fill = "blue", color = "white") +  
 scale\_x\_continuous(labels = scales::comma) +  
 scale\_y\_continuous(labels = scales::comma) +  
 labs(title = "Distribution of Chromosome Sizes",  
 x = "Size (base pairs)", y = "Frequency")

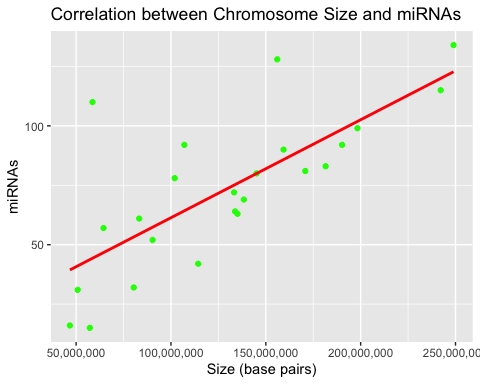


### c. Does the number of protein coding genes or miRNAs correlate with the length of the chromosome? Make two separate plots to visualize these relationships.

# Create a scatterplot for protein coding genes  
ggplot(chromosome, aes(x = basepairs, y = protein\_codinggenes)) +  
 geom\_point(color = "blue") +  
 geom\_smooth(method = "lm", se = FALSE, color = "red") +  
 scale\_x\_continuous(labels = scales::comma) +  
 scale\_y\_continuous(labels = scales::comma) +  
 labs(title = "Correlation between Chromosome Size and Protein Coding Genes",  
 x = "Size (base pairs)", y = "Protein Coding Genes")  
## `geom\_smooth()` using formula = 'y ~ x'



# Create a scatterplot for miRNAs  
ggplot(chromosome, aes(x = basepairs, y = mi\_rna)) +  
 geom\_point(color = "green") +  
 geom\_smooth(method = "lm", se = FALSE, color = "red") +  
 scale\_x\_continuous(labels = scales::comma) +  
 scale\_y\_continuous(labels = scales::comma) +  
 labs(title = "Correlation between Chromosome Size and miRNAs",  
 x = "Size (base pairs)", y = "miRNAs")  
## `geom\_smooth()` using formula = 'y ~ x'



### 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.

# To calculate the summary statistics for the 'proteins' data variables length and mass, we can use the summarize() function from the dplyr package.  
proteins\_summary <- proteins %>%  
 summarize(mean\_length = mean(length),  
 median\_length = median(length),  
 max\_length = max(length),  
 mean\_mass = mean(mass),  
 median\_mass = median(mass),  
 max\_mass = max(mass))  
  
proteins\_summary  
## # A tibble: 1 × 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

# Use the ggplot() function to create the plot, and add geom\_point() to add the points.  
ggplot(proteins, aes(x = length, y = mass)) +  
 geom\_point() +  
 labs(x = "Length", y = "Mass") +  
 theme\_classic()

