R\_assessment\_Report

Chisom\_Udeagbala\_Q2195099

2024-05-16

options(repos = c(CRAN = "https://cran.rstudio.com/"))  
  
# Install the readr package  
install.packages("readr")

## Installing package into 'C:/Users/Admin/AppData/Local/R/win-library/4.4'  
## (as 'lib' is unspecified)

## package 'readr' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\Admin\AppData\Local\Temp\RtmpkJC3Gu\downloaded\_packages

##QUESTION 1 What is R and its advantages/disadvantages; List and define some basic data structures in R; List and define some basic data types in R.

R is an open-source programming/scripting language and environment specifically designed for statistical computing and graphics. R is a comprehensive statistical and graphical programming language, a dialect of the S language, it was initially written by Ross Ihaka and Robert Gentleman at Department of Statistics of University of Auckland, New Zealand during 1990s. R provides a wide variety of statistical and graphical techniques, including linear and nonlinear modeling, time-series analysis, clustering, and more. It is commonly used by statisticians, data analysts and data scientists for data analysis, statistical modelling and data visualization across sectors/ industries such as Academia, healthcare, commerce, banking and finance, and social media. Since R is open source, it is freely available, making it accessible to anyone interested in statistical analysis or data visualization.

ADVANTAGES

It is open source and freely available.

It is easy to learn with relatively simple syntax.

It has built-in functions and additional packages for various statistical tests, models, and data analysis techniques.

It is widely used across different sectors and industries.

It is Available on all platforms.

It has a large and growing community making it easy to find support, tutorials, and answers to questions.

It has Over 15,000 packages available on the Comprehensive R Archive Network (CRAN), plus more on Bioconductor

It has strong graphical capabilities, offering fine control over plot features.

DISADVANTAGES

R has a limited memory and since it holds all objects in its memory during computation, it can be limiting when dealing with very large data sets.

R can be slower compared to other programming languages like Python or Java.

For beginners, R’s syntax can be somewhat opaque, making the learning curve steeper than some other programming languages

DATA STRUCTURES IN R

There is a wide variety of data structures in R, and the str() function is used to check the data structure of a dataset in R. Data structures in R include:

Vectors: Vectors are one-dimensional arrays that can hold numeric, character, or logical values. They are created using the c() function.

Data Frames: Data frames are two-dimensional tabular data structures similar to tables in a database or spreadsheet. They can hold different types of data in each column. Data frames are created using the data.frame() function.

Arrays: Arrays are multi-dimensional extensions of matrices. They can have more than two dimensions. Arrays are created using the array() function.

Factors: Factors are used to represent categorical data. They are created using the factor() function.

Lists: Lists are collections of objects that can be of different types (e.g., vectors, matrices, other lists). They are created using the list() function.

Matrices: Matrices are two-dimensional arrays with rows and columns. They can hold values of the same data type. Matrices are created using the matrix() function.

DATA TYPES IN R

There is a wide variety of data structures in R and the class() function is used to check the datatype of a dataset in R. Data types in R include:

Numeric: Any number with or without a decimal point: 23, 0.03 and the numeric null value NA. It is denoted with the ‘L’ suffix.

Integer: Integer data type is used to represent whole numbers. Unlike numeric data type, integers are stored without decimal points.

Character: it represents text strings enclosed within single (’ … ’) or double quotes (” … “). Sometimes it is referred to as “string”.

Logical: This data type only has two possible values — either TRUE or FALSE, “ON” or “OFF”, “YES” or “NO” (without quotes). It represents Boolean values used in conditional statements.

NA: This data type represents the absence of a value and is represented by the keyword NA (without quotes) but it has its own significance in the context of the different types. That is there is a numeric NA, a character NA, and a logical NA.

Complex: Complex data type is used to represent complex numbers, which consist of a real part and an imaginary part.

## QUESTION 2

You have two dice, one has six odd numbers: 1, 3, 5, 7, 9, 11; the other has six even numbers: 2, 4, 6, 8, 10, 12. Write a code that:  
a. prints all the combinations of numbers from two dice. b. uses your own function (not its internal function) to calculate the average value of all the sum of combinations. c. uses your own function (not its internal function) to find the maximum value of all the sum of combinations. d. calculates the possibility of getting a sum of 15. e. calculates the possibility of getting a sum bigger than 15.

#Define the two set of outcomes  
set1 <- c(1, 3, 5, 7, 9, 11) #odd numbers  
set2 <- c(2, 4, 6, 8, 10, 12) #even numbers  
  
#Create all combinations of the two dice  
combinations <- expand.grid(set1, set2)  
#count number of combinations  
total\_combinations <- nrow(combinations)  
#Display total combinations  
print(paste("total number of combinations is:", total\_combinations))

## [1] "total number of combinations is: 36"

#Calculate the sums of each combination  
combinations$Sum <- rowSums(combinations)  
#Display all combinations and their sums  
print(combinations)

## Var1 Var2 Sum  
## 1 1 2 3  
## 2 3 2 5  
## 3 5 2 7  
## 4 7 2 9  
## 5 9 2 11  
## 6 11 2 13  
## 7 1 4 5  
## 8 3 4 7  
## 9 5 4 9  
## 10 7 4 11  
## 11 9 4 13  
## 12 11 4 15  
## 13 1 6 7  
## 14 3 6 9  
## 15 5 6 11  
## 16 7 6 13  
## 17 9 6 15  
## 18 11 6 17  
## 19 1 8 9  
## 20 3 8 11  
## 21 5 8 13  
## 22 7 8 15  
## 23 9 8 17  
## 24 11 8 19  
## 25 1 10 11  
## 26 3 10 13  
## 27 5 10 15  
## 28 7 10 17  
## 29 9 10 19  
## 30 11 10 21  
## 31 1 12 13  
## 32 3 12 15  
## 33 5 12 17  
## 34 7 12 19  
## 35 9 12 21  
## 36 11 12 23

#calculate average  
# Define a Function to calculate average combination sum  
calculate\_average <- function(values) {  
 total\_sum <- sum(values) #total sum is the sum of all individual combination sums  
 total <- length(values) #total is the number of all combination sums  
 average <- total\_sum / total #average combination sum   
 return(average) #function returns the average combination sum  
}  
#call the average function to take the column "sums" of the combinations table and find the average sum  
average\_sum\_combinations <- calculate\_average(combinations$Sum)  
#display the average sum  
print(paste("The average combinations sum is", average\_sum\_combinations))

## [1] "The average combinations sum is 13"

#Define a function to calculate maximum value  
calculate\_max <- function(values) { #the function takes in "values" as input  
 maximum\_sum <- max(values) #maximum\_sum is the maximum of the values  
 return(maximum\_sum) #function returns maximum value of "values"  
}  
#call the csalculate\_max function and give the sum column of the combinations table as input  
max\_sum\_combinations <- calculate\_max(combinations$Sum)  
#Display the maximum combination sum  
print(paste("The maximun combination sum is", max\_sum\_combinations))

## [1] "The maximun combination sum is 23"

#Calculate the chance that the sum of two combinations is 15  
#create a vector to hold the count where sum is equal to 15  
sum\_is\_fifteen <- sum(combinations$Sum == 15)  
#define a vector to hold the count of the total combinations possible  
total\_combinations <- nrow(combinations)  
#calculate the probabilty of getting a sum equal to 15 and round up to two decimal place  
chance\_of\_sum\_fifteen <- round((sum\_is\_fifteen / total\_combinations), digits = 2)  
#Display the probability result  
print(paste("The chance of a sum of combinations equal to 15 is",chance\_of\_sum\_fifteen))

## [1] "The chance of a sum of combinations equal to 15 is 0.14"

#Calculate the chance that the sum of two combinations is > 15  
#create a vector to hold the count where sum is bigger than 15  
sum\_bigger\_fifteen <- sum(combinations$Sum > 15)  
#define a vector to hold the count of the total combinations possible  
total\_combinations <- nrow(combinations)  
#calculate the probabilty of getting a sum > 15 and round up to two decimal place  
chance\_sum\_bigger\_15 <- round((sum\_bigger\_fifteen / total\_combinations), digits = 2)  
#Display the probability result  
print(paste("The chance of a sum of combinations >15 is", chance\_sum\_bigger\_15))

## [1] "The chance of a sum of combinations >15 is 0.28"

## QUESTION 3

How would you create a new S4 class and its new method? Create a laptop class, then create an Apple Macbook Pro object and define a show function to display its CPU type to demonstrate the process.

# Define the S4 class for laptops  
setClass("Laptop", #set the classs name to Laptop  
 slots = c( #define the slots "brand", "model", "cpu\_type" and the data type  
 brand = "character",  
 model = "character",  
 cpu\_type = "character"  
 )  
)  
# Create an instance of the Laptop class for an Apple MacBook Pro  
Apple\_macbook\_pro <- new("Laptop",  
 brand = "Apple", #define the brand to be Apple  
 model = "MacBook Pro", #define model to MacBook Pro  
 cpu\_type = "Intel Core i9" #define cpu\_type to be intel core i9  
)  
# Define a method to display information of a Laptop objects  
setMethod("show",  
 signature = "Laptop",  
 function(object) {  
 cat("Brand:", object@brand, "\n") #Displays the brand of the object.  
 cat("Model:", object@model, "\n") # #Displays the model of the object  
 cat("CPU Type:", object@cpu\_type, "\n")# #Displays the Cpu of the object  
 }  
)  
# Display the MacBook Pro object  
Apple\_macbook\_pro

## Brand: Apple   
## Model: MacBook Pro   
## CPU Type: Intel Core i9

# QUESTION 4

Make a function to create 20 moving-sum numbers: 1, 2, 3, 5, 8, 13, 21, 34, … How many even numbers? What is the total sum of these even numbers?

#Define a function to generate a sequence of moving numbers  
generate\_moving\_sequence <- function(n) {   
#Creates a numeric vector of length n to store the sequence.  
 sequence <- numeric(n)  
 sequence[1] <- 0 # Start the sequence with 0  
 sequence[2] <- 1 # Second element is also 1  
#Iterate over the indices from 3 to n, calculating the next number in the sequence.  
 for (i in 3:n) {  
 sequence[i] <- sequence[i - 1] + sequence[i - 2] # Calculate the next number in the sequence  
 }  
   
 return(sequence) #return the sequence formed  
}  
generate\_moving\_sum <- function(numbers, window\_size) {  
#initialize an empty numeric vector, with length equal to the difference between the length of the numbers vector and the window\_size plus one.   
 moving\_sums <- numeric(length(numbers) - window\_size + 1)  
 #iterates over the indices of the numbers vector, considering each possible starting index for the moving sum calculation.   
 for (i in 1:(length(numbers) - window\_size + 1)) {  
#calculate the moving sum for the current window   
 moving\_sums[i] <- sum(numbers[i:(i + window\_size - 1)])  
 }  
 #return the sums  
 return(moving\_sums)  
}  
  
# Generate the moving sum sequence  
moving\_sum\_sequence <- generate\_moving\_sequence(21)  
  
# Calculate moving sums  
window\_size <- 2  
the\_moving\_sums <- generate\_moving\_sum(moving\_sum\_sequence, window\_size)  
  
# Display the moving sums  
# Count the number of even numbers using the sum function  
num\_even\_numbers <- sum(the\_moving\_sums %% 2 == 0)  
#  
Even\_moving\_nums <- the\_moving\_sums[the\_moving\_sums %% 2 == 0]  
total\_even\_sums <- sum(Even\_moving\_nums)  
print("the 20 moving sums are:")

## [1] "the 20 moving sums are:"

print(the\_moving\_sums)

## [1] 1 2 3 5 8 13 21 34 55 89 144 233  
## [13] 377 610 987 1597 2584 4181 6765 10946

# Print the result  
print(paste("Number of even numbers in the sequence:", num\_even\_numbers))

## [1] "Number of even numbers in the sequence: 7"

print(paste("total sum of all even numbers in the sequence:", total\_even\_sums))

## [1] "total sum of all even numbers in the sequence: 14328"

##QUESTION 5 #QUESTION 1 How many patients are with AML and how many are the control cases? How many genes(probesets) are we studying? (For the purpose of the assignment, each probeset is treated as a gene).

#Install and load the readr package  
install.packages("readr")

## Installing package into 'C:/Users/Admin/AppData/Local/R/win-library/4.4'  
## (as 'lib' is unspecified)

## package 'readr' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\Admin\AppData\Local\Temp\RtmpkJC3Gu\downloaded\_packages

library(readr)  
#Read a tab-delimited meta data file  
meta\_info\_data <- read.delim("GSE9476\_meta\_info.txt", header = TRUE, sep = "\t")  
#import and read the tab- delimited Gene Expression data  
Expression\_data <- read.delim("GSE9476\_series\_matrix.txt", header = TRUE, sep = "\t",skip = 83) #skip 83 columns of meta data  
#The last row of the expression data has NA values so we remove it  
Expression\_data <- Expression\_data[-nrow(Expression\_data), ]  
#subset the meta data to extract number of AML disease samples   
disease\_count <- sum(meta\_info\_data$Disease\_status =="AML")  
#subset the meta data to extract number of healthy control samples   
control\_count <- sum(meta\_info\_data$Disease\_status == "Control")  
#count the rows of gene ID ref to get number of genes  
genes\_count <- nrow(Expression\_data)  
  
#Display the number of disease samples  
print(paste("AML disease cases are", disease\_count, "in number"))

## [1] "AML disease cases are 26 in number"

#Display the number of control samples  
print(paste("control cases are", control\_count, "in number"))

## [1] "control cases are 38 in number"

#Display the number of genes  
print(paste("Number of genes are ", genes\_count))

## [1] "Number of genes are 22283"

#QUESTION5.2 Create your R function to calculate the average age of AML patients and Control subjects separately. Check your results by using R mean function.

age\_AML <- c() #initialize an empty vector to store the ages of AML patients  
age\_control <- c() #initialize an empty vector to store the ages of control patients  
#iterate through the indicies of Disease status column in the meta data   
for (i in 1:length(meta\_info\_data$Disease\_status)) {  
#check if the age value for the current individual is not missing (not NA).  
 if (!is.na(meta\_info\_data$Age[i])) {  
 #check if disease status is AML   
 if (meta\_info\_data$Disease\_status[i] == "AML") {  
 #append the age index to the vector "age\_AML"  
 age\_AML <- c(age\_AML, meta\_info\_data$Age[i])  
 #if Disease status not AML ie control  
 } else {  
 #append the age index to the vector "age\_AML"  
 age\_control <- c(age\_control, meta\_info\_data$Age[i])  
 }  
 }  
}  
#Define a function to take in an input "ages" and find the average  
find\_average\_age <- function(ages) {  
 total\_sum = sum(ages) #calculate total age  
 Average\_sum = total\_sum/length(ages) #find average of age   
 return(Average\_sum) #return the average age value  
}  
#find average age of AML/Diseaese patients/cases  
avg\_age\_AML <- round((find\_average\_age(age\_AML)), digits = 2)  
#find average age of control patients/cases  
avg\_age\_control <- round((find\_average\_age(age\_control)), digits = 2)  
  
#display the average age of disease and control patients  
print(paste( "average age of AML samples is: ", avg\_age\_AML))

## [1] "average age of AML samples is: 56.12"

print(paste( "average age of Control samples is: ", avg\_age\_control))#check average result with R built in mean function

## [1] "average age of Control samples is: 33.33"

print(paste("average age of AML samples is: ", mean(age\_AML)))

## [1] "average age of AML samples is: 56.1153846153846"

print(paste("average age of Control samples is: ", mean(age\_control)))

## [1] "average age of Control samples is: 33.3333333333333"

#QUESTION 5.3 Is there any significant difference between sample ages of AMLs and Controls? Assume the age follows a normal distribution.

Answer: Yes there is significant difference between the ages of the AML and control cases pvalue is less than 0.05.

#Use t.test function to test the hypothesis that the ages are different  
p\_value\_test <- t.test(age\_AML, age\_control, paired = FALSE, conf.level = 0.95)  
print(p\_value\_test)

##   
## Welch Two Sample t-test  
##   
## data: age\_AML and age\_control  
## t = 6.2687, df = 47.986, p-value = 9.743e-08  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## 15.47480 30.08931  
## sample estimates:  
## mean of x mean of y   
## 56.11538 33.33333

#QUESTION 5.4 Make a boxplot of gene expression for all the samples, marking data points in red for AML samples and blue for Control samples.

#Install the tidyr package  
install.packages("tidyr")

## Installing package into 'C:/Users/Admin/AppData/Local/R/win-library/4.4'  
## (as 'lib' is unspecified)

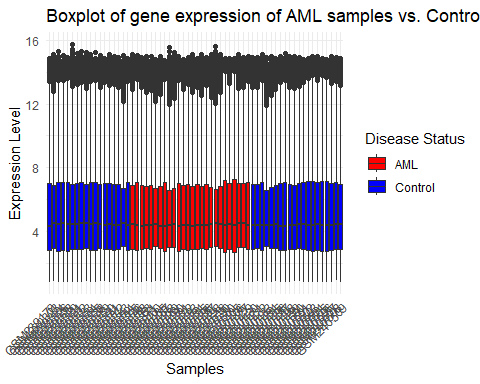
## package 'tidyr' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\Admin\AppData\Local\Temp\RtmpkJC3Gu\downloaded\_packages

install.packages("ggplot2")

## Installing package into 'C:/Users/Admin/AppData/Local/R/win-library/4.4'  
## (as 'lib' is unspecified)

## package 'ggplot2' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\Admin\AppData\Local\Temp\RtmpkJC3Gu\downloaded\_packages

#call the tidyr and ggplot libray  
library(tidyr)  
library(ggplot2)  
#convert the gene expression data from wide data to a long data  
gene\_expression\_long <- pivot\_longer(Expression\_data,   
 cols = -ID\_REF,  
 names\_to = "SampleID",  
 values\_to = "Expression\_Level")  
#merge the long expression data with the meta data by sampleID which is common to both  
merged\_long\_data <- merge(gene\_expression\_long, meta\_info\_data, by.x = "SampleID", by.y = "SampleID", all.x = TRUE)  
  
# make a boxplot of the gene expression for the samples  
#Initialize a ggplot object  
ggplot(merged\_long\_data, aes(x = SampleID, y = Expression\_Level, fill = Disease\_status)) +  
 geom\_boxplot() + #Add a boxplot layer to the ggplot object  
 #set the fill colors for the Disease\_status levels, bluo for control, red for disease  
 scale\_fill\_manual(values = c("Control" = "blue", "AML" = "red")) +  
 #set labels for x and y axis  
 labs(x = "Samples", y = "Expression Level", fill = "Disease Status") +  
 #set label for title  
 ggtitle("Boxplot of gene expression of AML samples vs. Control samples") + # Add title +  
 theme\_minimal() + #removes unnecessary background elements.  
 theme(axis.text.x = element\_text(angle = 45, hjust = 1)) #To avoid overlap



#QUESTION 5.5 Create a table to include mean\_AML, sd\_AML, min\_AML, max\_AML, mean\_Control, sd\_Control, min\_Control, max\_Control, and Fold\_change (i.e, mean\_AML – mean\_Control) for each gene. It is fine to use built-in functions. Hints: split the dataset into AML data and normal data sets, and then for each gene/probeset, calculate its mean, standard derivation, min, and max expression values across samples separately (using a built-in function), and further merge these statistical values for each gene into one table. Apply data.frame() and give the created table the same row names as the gene expression data table.

#install.packages("tidyr")  
library(tidyr)  
#install.packages("dplyr")  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

#install.packages("tidyverse")  
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ forcats 1.0.0 ✔ stringr 1.5.1  
## ✔ lubridate 1.9.3 ✔ tibble 3.2.1  
## ✔ purrr 1.0.2

## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

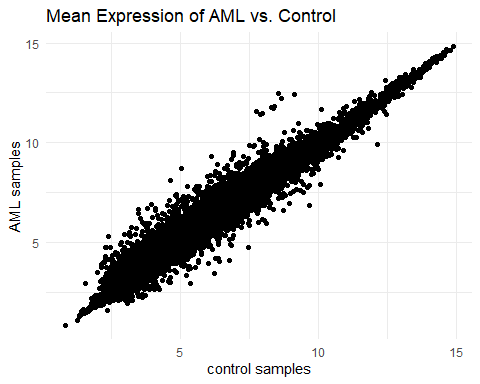
# Filter expression data based on disease status and extract disease and control data  
d\_disease\_data <- merged\_long\_data[merged\_long\_data$Disease\_status == "AML", ]  
d\_control\_data <- merged\_long\_data[merged\_long\_data$Disease\_status == "Control", ]  
#  
AML\_disease\_data <- d\_disease\_data[1:3]  
 #transpose the expression data so that the column names which is sampleID becomes row names.   
wide\_AML\_data <- AML\_disease\_data %>%  
 arrange(ID\_REF) %>%  
 pivot\_wider(names\_from = SampleID, values\_from = Expression\_Level) %>%  
 column\_to\_rownames(var = "ID\_REF")  
  
####  
  
AML\_control\_data <- d\_control\_data[1:3]  
   
wide\_control\_data <- AML\_control\_data %>%  
 arrange(ID\_REF) %>%  
 pivot\_wider(names\_from = SampleID, values\_from = Expression\_Level) %>%  
 column\_to\_rownames(var = "ID\_REF")  
# Filter expression data based on disease status and extract disease and control data  
#define a function to take in data as input and calculate statistics summary  
statistics <- function(data){  
 mean\_data <- apply(data, 1, function(x) mean(x, na.rm = TRUE))#calculate mean,removing Na values  
 max\_data <- apply(data, 1, function(x) max(x, na.rm = TRUE))#calculate max,removing Na values  
 min\_data <- apply(data, 1, function(x) min(x, na.rm = TRUE)) #calculate min,removing Na values  
 sd\_data <- apply(data, 1, function(x) sd(x, na.rm = TRUE)) #calculate standard deviation,removing Na values  
 #return the mean, max, min and standard deviation as a dataframe  
 return(data.frame(mean\_data, max\_data, min\_data, sd\_data, row.names = NULL))  
}  
#call the statistics function to calculate summary statistics for AML and controlcases  
new\_disease\_stats <- statistics(wide\_AML\_data)  
new\_control\_stats <- statistics(wide\_control\_data)  
#label the columns as "AML\_mean", "AML\_max", "AML\_min","AML\_sd\_data" for the two tables  
colnames(new\_disease\_stats)[c(1,2,3,4)] <- c("AML\_mean", "AML\_max", "AML\_min","AML\_sd\_data")  
colnames(new\_control\_stats)[c(1,2,3,4)] <- c("control\_mean", "control\_max", "control\_min","control\_sd\_data")  
#Add the gene IDs as row names to the newly formed disease and control stats table  
rownames(new\_disease\_stats) <- row.names(wide\_AML\_data)  
rownames(new\_control\_stats) <- row.names(wide\_AML\_data)  
# Merge the two disease and control statistics table  
merged\_exp\_data <- merge(new\_disease\_stats, new\_control\_stats, by = "row.names")  
merged\_exp\_data$ID\_REF <- merged\_exp\_data$Row.names  
# Define a function to Calculate the logFC   
fold\_change <- function(mean\_1, mean\_2){   
 #logfc is diference in mean between disease and control  
 fc = mean\_1 - mean\_2  
 return(fc) #return logfc  
}  
#call the function to calculate the logFC of the merged expression statistics data  
LogFC <- fold\_change(merged\_exp\_data$AML\_mean, merged\_exp\_data$control\_mean)  
#define rownames to be gene IDs  
rownames(LogFC) <- merged\_exp\_data$Gene\_Probeset  
#add the logfc to the merged expression statistics data  
merged\_exp\_data$logFC <- LogFC  
  
#Display the merged table  
#print(merged\_exp\_data)

#QUESTION 6 With the table created above, make a scatterplot of the mean expression of AMLvs. Control samples. Are mean\_AML and mean\_Control significantly different? Answer: mean AML is not significantly different from mean control. pvalue is 0.4539 and the mean AML is 5.091984 and mean control is 5.109899

#install ggplot2 package  
install.packages("ggplot2")

## Warning: package 'ggplot2' is in use and will not be installed

#call the ggplot libray  
library(ggplot2)  
#subset the column of mean AML disease from the merged table  
mean\_AML <- merged\_exp\_data$AML\_mean  
#subset the column of mean AML disease from the merged table  
mean\_control <- merged\_exp\_data$control\_mean  
#create a data frame of mean gene expresion for disease and control  
mean\_data <- data.frame(Group = c("AML", "Control"),  
 Mean\_Expression = c(mean\_AML, mean\_control))  
#make the group a factor so that R treats it as a factor not a number  
mean\_data$Group <- factor(mean\_data$Group)   
  
# Create a scatterplot with ggplot for mean disease and control  
ggplot(merged\_exp\_data, aes(x = mean\_AML, y = mean\_control)) +  
 geom\_point() + #Add a scatterplot layer to the ggplot object  
 labs(x = "control samples", y = "AML samples") + # Add axis labels  
 ggtitle("Mean Expression of AML vs. Control") + # Add title  
 theme\_minimal() # Use a minimal theme



#Test the hypothesis whether the mean expression for the disease is significantly different from that of control  
hypothesis\_testing <- t.test(mean\_AML, mean\_control, paired = FALSE, conf.level = 0.95)  
#display the t test result  
print(hypothesis\_testing)

##   
## Welch Two Sample t-test  
##   
## data: mean\_AML and mean\_control  
## t = -0.74899, df = 44560, p-value = 0.4539  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -0.06479628 0.02896624  
## sample estimates:  
## mean of x mean of y   
## 5.091984 5.109899

#Mean expression of disease cases is not significantly different from mean expressions for disease cases

#QUESTION 5.7 For each gene/probeset, conduct t-test between AML samples and Control samples, and then create a table including t scores and p-values from t-test, and merge to the table created in 6.

# First, split the data into disease and control groups  
the\_disease\_data <- subset(merged\_long\_data, Disease\_status == "AML")  
the\_control\_data <- subset(merged\_long\_data, Disease\_status == "Control")  
  
# Create an empty dataframe to store t-test results  
t\_test\_results <- data.frame(ID\_REF = character(), t\_statistic = numeric(), p\_value = numeric(), stringsAsFactors = FALSE)  
  
# Loop through each gene (ID\_REF)  
for (gene\_id in unique(merged\_long\_data$ID\_REF)) {  
 # Extract expression levels for the current gene in disease and control groups  
 disease\_expr <- subset(the\_disease\_data, ID\_REF == gene\_id)$Expression\_Level  
 control\_expr <- subset(the\_control\_data, ID\_REF == gene\_id)$Expression\_Level  
   
 # Perform t-test  
 t\_test\_result <- t.test(disease\_expr, control\_expr)  
   
 # Store t-test results in a dataframe  
 t\_test\_results <- rbind(t\_test\_results, data.frame(ID\_REF = gene\_id,   
 t\_statistic = t\_test\_result$statistic, p\_value = t\_test\_result$p.value))  
}  
  
# View the dataframe with t-test results  
#print(t\_test\_results)  
# Merge the subsets using cbind  
merged\_exp\_stats\_data <- merge(merged\_exp\_data, t\_test\_results, by = "ID\_REF")  
merged\_exp\_stats\_data$Row.names <- NULL  
#print(merged\_exp\_stats\_data)

#QUESTION 5.8 How many genes/probesets are left after the following filters? a.) mean\_AML > 3 and mean\_Control > 3 b.) max\_AML < 14 or max\_Control < 14  
c)Apply a) and b) filters together which is more stringent than applying a) or b) alone

# filter by mean expression > 3 for disease and control  
filter\_by\_mean <- merged\_exp\_stats\_data[merged\_exp\_stats\_data$AML\_mean > 3 & merged\_exp\_stats\_data$control\_mean > 3,]  
# filter by max > 3 for disease and control  
filter\_by\_max <- merged\_exp\_stats\_data[merged\_exp\_stats\_data$AML\_max < 14 | merged\_exp\_stats\_data$control\_max < 14, ]  
  
# filter by mean >3 and AML\_max > 14 or control\_max >14  
filter\_by\_mean\_max <- filter\_by\_mean[filter\_by\_mean$AML\_max < 14 | filter\_by\_mean$control\_max < 14,]  
# Count the number of genes/probesets remaining after the combined filter  
print(paste("the num of genes left after first filter is", nrow(filter\_by\_mean)))

## [1] "the num of genes left after first filter is 16026"

print(paste("the num of genes left after second filter is", nrow(filter\_by\_max)))

## [1] "the num of genes left after second filter is 22171"

print(paste("the num of genes left after combinedd filter is", nrow(filter\_by\_mean\_max)))

## [1] "the num of genes left after combinedd filter is 15914"

#QUESTION 5.9 Following 8, if we further filter gene/probeset by applying p-value < 0.05 and the absolute value of log2 Fold\_change bigger than 1, then how many genes will be left which we call them significantly expressed genes (DEGs)? Among them, how many are up-regulated (Fold\_change > 0) and how many are down-regulated (Fold\_change < 0)? List top 10 most upregulated DEGs and 10 most down-regulated DEGs including their p-value, log2FC, and mean values in each sample group.

# Filter rows based on P-value and logFC criteria from the filtered data  
filtered\_result <- filter\_by\_mean\_max[filter\_by\_mean\_max$p\_value < 0.05 & abs(filter\_by\_mean\_max$logFC) > 1, ]  
num\_genes\_left <- nrow(filtered\_result)  
  
# Print the number of genes/probesets left  
print(paste("Number of genes/probesets left after filtering by mean,max,pval and logfc:", num\_genes\_left))

## [1] "Number of genes/probesets left after filtering by mean,max,pval and logfc: 626"

# Count the number of up-regulated and down-regulated genes  
num\_upregulated <- sum(filtered\_result$logFC > 0)  
num\_downregulated <- sum(filtered\_result$logFC < 0)  
  
# Print the number of DEGs and the counts of up-regulated and down-regulated genes  
cat("Number of significantly expressed genes (DEGs):", num\_genes\_left, "\n")

## Number of significantly expressed genes (DEGs): 626

cat("Number of up-regulated genes:", num\_upregulated, "\n")

## Number of up-regulated genes: 136

cat("Number of down-regulated genes:", num\_downregulated, "\n")

## Number of down-regulated genes: 490

# creat a List of top 10 upregulated DEGs  
top\_ten\_upregulated <- head(filtered\_result[order(filtered\_result$logFC, decreasing = TRUE), ], 10)  
# create a List of top 10 down-regulated DEGs  
top\_ten\_downregulated <- head(filtered\_result[order(filtered\_result$logFC), ], 10)  
  
# Print top 10 upregulated DEGs  
cat("\nTop 10 upregulated DEGs:\n")

##   
## Top 10 upregulated DEGs:

print(top\_ten\_upregulated)

## ID\_REF AML\_mean AML\_max AML\_min AML\_sd\_data control\_mean  
## 14862 215489\_x\_at 6.237537 9.166895 2.341520 1.8459302 3.431159  
## 6200 206674\_at 9.669838 11.280655 7.616481 0.9546449 6.876417  
## 9843 210365\_at 7.441080 10.046149 4.669434 1.4212957 4.736884  
## 4174 204647\_at 6.764131 10.323205 2.693155 2.0727303 4.096730  
## 5593 206067\_s\_at 6.391956 9.334233 3.561072 1.8991179 4.054346  
## 633 201105\_at 12.127482 13.376602 8.839435 1.2303518 9.891301  
## 4658 205131\_x\_at 6.354558 10.683155 1.658505 2.6885262 4.204067  
## 13953 214575\_s\_at 8.107346 12.649552 3.310517 3.2255145 5.959960  
## 4909 205382\_s\_at 10.078486 13.446872 4.174919 2.4420972 7.931564  
## 11110 211709\_s\_at 8.731739 11.493808 4.727807 2.2094899 6.624619  
## control\_max control\_min control\_sd\_data logFC t\_statistic p\_value  
## 14862 5.038581 1.812112 1.0012402 2.806378 7.072818 3.021987e-08  
## 6200 9.445325 3.528771 1.8451665 2.793421 7.912149 8.237636e-11  
## 9843 7.475545 2.954536 1.3581058 2.704195 7.611153 5.128225e-10  
## 4174 6.261917 2.630884 1.1360273 2.667401 5.976441 8.013568e-07  
## 5593 4.414244 3.730570 0.1850671 2.337609 6.256051 1.433770e-06  
## 633 12.338433 7.373187 1.3947045 2.236181 6.760455 7.396949e-09  
## 4658 7.413472 2.082558 1.6603251 2.150490 3.632135 8.268710e-04  
## 13953 9.505510 3.101807 2.4872662 2.147387 2.862027 6.393440e-03  
## 4909 12.071110 3.467865 2.2600775 2.146922 3.559475 8.140686e-04  
## 11110 9.456413 2.162979 2.0826208 2.107120 3.834913 3.427277e-04

print(top\_ten\_upregulated$ID\_REF)

## [1] "215489\_x\_at" "206674\_at" "210365\_at" "204647\_at" "206067\_s\_at"  
## [6] "201105\_at" "205131\_x\_at" "214575\_s\_at" "205382\_s\_at" "211709\_s\_at"

# Print top 10 down-regulated DEGs  
cat("\nTop 10 down-regulated DEGs:\n")

##   
## Top 10 down-regulated DEGs:

print(top\_ten\_downregulated)

## ID\_REF AML\_mean AML\_max AML\_min AML\_sd\_data control\_mean  
## 8610 209116\_x\_at 8.538463 13.795138 3.481071 2.882742 12.489345  
## 16779 217414\_x\_at 7.740921 13.465267 2.265479 3.294756 11.578075  
## 11610 212224\_at 5.010834 8.752955 2.671007 1.981498 8.710812  
## 16599 217232\_x\_at 8.668270 13.467697 5.292753 2.342057 12.213326  
## 8951 209458\_x\_at 7.884379 13.398513 3.067269 3.053984 11.419415  
## 11100 211699\_x\_at 7.962659 13.422177 3.284826 2.916587 11.479616  
## 3545 204018\_x\_at 8.299976 13.581241 2.714353 3.012318 11.758847  
## 5733 206207\_at 4.638640 10.825961 2.105885 2.606206 8.090973  
## 11145 211745\_x\_at 8.387930 13.585314 3.550242 2.931243 11.802806  
## 13793 214414\_x\_at 8.370409 13.569854 2.820095 3.085795 11.726737  
## control\_max control\_min control\_sd\_data logFC t\_statistic  
## 8610 15.24995 6.785420 2.5617460 -3.950883 -5.630795  
## 16779 15.73261 4.692070 3.2248056 -3.837153 -4.615432  
## 11610 10.56486 6.509192 0.9785371 -3.699978 -8.814193  
## 16599 14.74952 6.766785 2.3543967 -3.545056 -5.934498  
## 8951 14.76805 4.266272 2.9579303 -3.535036 -4.606247  
## 11100 15.42275 4.770810 3.1506440 -3.516957 -4.584919  
## 3545 15.08012 5.761851 2.8825054 -3.458872 -4.590837  
## 5733 11.70940 3.145440 2.4928731 -3.452333 -5.297009  
## 11145 15.12502 5.735317 2.8510421 -3.414876 -4.628346  
## 13793 15.06595 5.052806 2.9107935 -3.356328 -4.372525  
## p\_value  
## 8610 8.365940e-07  
## 16779 2.521879e-05  
## 11610 3.104044e-10  
## 16599 2.163963e-07  
## 8951 2.630755e-05  
## 11100 2.560269e-05  
## 3545 2.807991e-05  
## 5733 2.389381e-06  
## 11145 2.428234e-05  
## 13793 5.951027e-05

print(top\_ten\_downregulated$ID\_REF)

## [1] "209116\_x\_at" "217414\_x\_at" "212224\_at" "217232\_x\_at" "209458\_x\_at"  
## [6] "211699\_x\_at" "204018\_x\_at" "206207\_at" "211745\_x\_at" "214414\_x\_at"

#QUESTION 5.10 if you apply the limma package rather than t-test, then how many DEGs will be discovered by limma? (p-value < 0.05 & abs(log2FC) > 1). How many overlapped DEGs by these two different approaches? please draw a Venn diagram for demonstration.

Answer: there are 620 overlapped genes between the two approaches. 6 genes in the t.test filtered result are not in the limma filtered genes while 88 genes in the limma filtered genes are not in the t.test filtered genes.

#carry out DEG analysis using limma to compare the result with the filtered pvalue/fold-change result.  
  
#install the Bioconductor package "BiocManager"#install.packages("BiocManager")  
install.packages("BiocManager")

## Installing package into 'C:/Users/Admin/AppData/Local/R/win-library/4.4'  
## (as 'lib' is unspecified)

## package 'BiocManager' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\Admin\AppData\Local\Temp\RtmpkJC3Gu\downloaded\_packages

#instal limma from the Biocmanager  
BiocManager::install("limma")

## 'getOption("repos")' replaces Bioconductor standard repositories, see  
## 'help("repositories", package = "BiocManager")' for details.  
## Replacement repositories:  
## CRAN: https://cran.rstudio.com/

## Bioconductor version 3.19 (BiocManager 1.30.23), R 4.4.0 (2024-04-24 ucrt)

## Warning: package(s) not installed when version(s) same as or greater than current; use  
## `force = TRUE` to re-install: 'limma'

## Installation paths not writeable, unable to update packages  
## path: C:/Program Files/R/R-4.4.0/library  
## packages:  
## KernSmooth, survival

## Old packages: 'fastmap', 'xfun'

#load the limma library  
library(limma)  
  
#make the gene names the row names for limma so they aren't classed as data  
Gene\_Expression\_data <- as.data.frame(Expression\_data)  
rownames(Expression\_data) <- Gene\_Expression\_data$ID\_REF  
  
#remove the original gene name column  
Gene\_Expression\_data$ID\_REF <- NULL  
  
#Convert disease status to a factor for analysis.   
meta\_info\_data$Disease\_status <- factor(meta\_info\_data$Disease\_status)  
  
#Generate a design matrix for statistical modelling, using Disease\_status as the explanatory variable.   
#The ~ 0 + syntax means that no intercept is included in the model, allowing for the estimation of effects for #each level of Disease\_status directly.  
  
design <- model.matrix(~ 0 + meta\_info\_data$Disease\_status)  
  
#Rename the columns of the design matrix to match the levels of the Disease\_status in meta\_info data.   
  
colnames(design) <-levels(meta\_info\_data$Disease\_status)  
  
#Fit the linear model  
fit <- lmFit(Gene\_Expression\_data, design)  
  
#Determine the levels in the makeContrasts call  
contrast.matrix <- makeContrasts(AML\_vs\_Control = AML - Control, levels = design)  
  
#Now apply the contrasts  
fit2 <- contrasts.fit(fit, contrast.matrix)  
  
#And then apply Bayesian adjustment  
fit2 <- eBayes(fit2)  
  
#Extract the results for the specific contrast  
# Assuming fit2 is a correctly fitted model object  
results <- topTable(fit2, coef="AML\_vs\_Control", adjust.method="BH", sort.by="P", n=Inf)  
  
#this sorts by p value and list an infinite number of genes  
  
#results  
  
#Filtering  
# Add a new column indicating the filter criteria  
LIMMA\_filtered\_genes <- results[abs(results$logFC) > 1 & results$P.Value < 0.05, ]  
limma\_filtered <- nrow(LIMMA\_filtered\_genes)  
print(paste("num of genes left after filtering result from limma is", limma\_filtered))

## [1] "num of genes left after filtering result from limma is 708"

#compare the filtered results from limma with the pval/fold-change filter  
limma\_genes <- row.names(LIMMA\_filtered\_genes)  
pvalue\_logFC\_genes <- filtered\_result$ID\_REF  
  
# Compare the gene sets for similarities  
common\_genes <- intersect(limma\_genes, pvalue\_logFC\_genes)  
  
# Count the number of common genes  
num\_common\_genes <- length(common\_genes)  
print(paste("Overlapped DEGs are ", num\_common\_genes))

## [1] "Overlapped DEGs are 0"

i noticed the “overlapped genes” prints “0” instead of 620 after knitting to HTML, hence i have attached the .Rmd file to this report file on submission.

Question 10 please draw a Venn diagram for demonstration

Answer: there are 620 overlapped genes between the two approaches. 6 genes in the t.test filtered result are not in the limma filtered genes while 88 genes in the limma filtered genes are not in the t.test filtered genes.

# Import necessary libraries  
install.packages("VennDiagram")

## Installing package into 'C:/Users/Admin/AppData/Local/R/win-library/4.4'  
## (as 'lib' is unspecified)

## package 'VennDiagram' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\Admin\AppData\Local\Temp\RtmpkJC3Gu\downloaded\_packages

library(VennDiagram)

## Loading required package: grid

## Loading required package: futile.logger

#compare the filtered results from limma with the pval/fold-change filter  
limma\_genes <- row.names(LIMMA\_filtered\_genes)  
pvalue\_logFC\_genes <- filtered\_result$ID\_REF  
  
# Compare the gene sets for similarities  
common\_genes <- intersect(limma\_genes, pvalue\_logFC\_genes)  
  
# Create the Venn diagram  
venn.plot <- venn.diagram(  
 x = list('Pval filtered' = pvalue\_logFC\_genes, 'Limma filtered' = limma\_genes),  
 category.names = c("Pval filtered", "Limma filtered"),  
 filename = NULL,  
 output = TRUE,  
 imagetype = "png",  
 height = 300,  
 width = 300,  
 resolution = 300,  
 compression = "lzw",  
 lwd = 2,  
 col = c("red", "blue"),  
 fill = c("red", "blue"),  
 alpha = 0.5,  
 cex = 1.5,  
 fontface = "bold",  
 cat.cex = 1.5,  
 cat.fontface = "bold",  
 cat.default.pos = "outer",  
 cat.pos = c(-20, 20),  
 cat.dist = c(0.05, 0.05),  
 cat.col = c("red", "blue")  
 )  
  
# Plot the Venn diagram  
grid.draw(venn.plot)

