R_report_Bassey-AsuquoNkaepe_c2766519_R

Bassey-Asuquo,Nkaepe

2024-05-16

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.

Answers:

R is a programming language for statistical computing and data visualization. It has been widely adopted in fields such as data mining, bioinformatics, and data analysis.

Key points:

• Free and Open Source: R is an open-source language, which means it's freely available for anyone to use and modify. You can download it from the official R Project website.

• Statistical Computing: R is particularly well-suited for statistical analysis and modeling. Researchers, data scientists, and statisticians use it to perform tasks like hypothesis testing, regression analysis, and data visualization.

• Platform-Independent: R runs on various platforms, including UNIX, Windows, and MacOS. This makes it accessible to users regardless of their operating system.

• Extension Packages: The core R language is augmented by a large number of extension packages. These packages contain reusable code, documentation, and sample data. Users can easily create their own functions and packages in R.

• Community and Events: The R community is active and supportive. There are regular conferences and events, such as the annual use R! conference, where users can learn, share knowledge, and collaborate. (R: The R Project for Statistical Computing (r-project.org)).

• In R, operations can be performed in vectors, which are one-dimensional arrays.

• Loop operations are performed on each element within the vector, which is not implemented in other scripting languages.

• In R script, when loop operations are performed in vector form, it produces an optimized command. Before performing statistical operations, the data can be visualized in R. Here, we can analyze and solve missing values, outliers, and scale transformations, and observe the relationship between variables by directly plotting a plot. Especially non-programmers can work at a higher level if they have experience in SPSS.

• In RStudio, we can also read data that is not structured in the console. This can be done using the read.csv, from JSON, and DBI functions. The reading of data can be visualized with

the help of str, head, and summary functions. Along with that, NAs removal and missing values can be visualized by table and if_else functions.

• R has a number of functions available, providing knowledge on statistical operations where functions are available for all mathematical, statistical, and machine learning concepts predefined in CRAN (Comprehensive R Archive Network), which is impossible in other scripting languages. As R packages are high in number and continuously added in CRAN, R is in a top position. R programming language is an open-source package that can be downloaded from the website, where it can be easily installed in an accessible environment. (Cummin et. al., 2022; Gupta et. al., 2020; Li et. al., 2023)

Advantages of R Programming Language:

1. Rich Package Ecosystem:

• R boasts an extensive collection of packages available on CRAN (Comprehensive R Archive Network), Bioconductor, and GitHub. These packages cover a wide range of data analysis and statistical tasks, providing ready-to-use functions and tools(Top Advantages and Disadvantages of R programming (theknowledgeacademy.com)).

2. Statistical Analysis:

• R's strong foundation in statistics makes it a powerful tool for various analyses: o Linear and nonlinear modeling o Time series analysis o Hypothesis testing o Machine learning tasks

3. Graphics Capabilities:

• R offers flexible graphics capabilities for data visualization. • Base graphics are included, and packages like ggplot2 offer advanced plotting features.

4. Data Handling:

• R possesses an array of tools and packages for importing, cleaning, and wrangling data, making it a versatile tool for data analysis.

5. Community Support: • The vibrant R community contributes to its strength. Users can find help, share knowledge, and collaborate through forums, mailing lists, and conferences.

Disadvantages of R Programming Language:

1. Memory Usage:

• R tends to use more memory compared to some other languages.

• Large datasets or complex computations may require substantial memory resources.

2. Learning Curve:

• R has a unique programming language and structure.

• Beginners without a background in statistical computing and coding may find it challenging initially. (Discover the Crucial Aspects: Advantages and Disadvantages of R Programming (itblog.dev))

3. Speed and Efficiency:

• While strides have been made to improve memory management, speed, and efficiency, R still faces challenges in these areas.

• Some operations may be slower compared to other languages.

4. Quirks:

• People transitioning from other programming languages might find R quirky due to its idiosyncrasies.

In R programming, several basic data structures are commonly used for organizing and storing data. Let's explore them along with brief definitions: Data Structures in R Programming - GeeksforGeeks

1.Vectors:

• A vector is an ordered collection of basic data types (e.g., numeric, character, logical) of a given length.

• All elements within a vector must be of the same data type (homogeneous).

• Vectors are one-dimensional data structures.

Example:

R program to illustrate Vector Vectors (ordered collection of same data type) X = c (1, 3, 5, 7, 8) Printing those elements in console print(X) Output: [1] 1 3 5 7 8

2. Lists: • A list is a generic object consisting of an ordered collection of heterogeneous objects.

• Lists can contain various data types, including vectors, matrices, characters, and functions.

• Lists are also one-dimensional data structures.

Example: R program to illustrate a List The first attributes is a numeric vector containing the employee IDs which is created using the 'c' command here empId = c(1, 2, 3, 4)

The second attribute is the employee's name which is created using this line of code here which is the character vector

empName = c("Debi", "Sandeep", "Subham", "Shiba")

The third attribute is the number of employees which is a single numeric variable.

numberOfEmp = 4

We can combine all these three different data types into a list containing the details of employees which can be done using a list command

empList = list(empId, empName, numberOfEmp)

print(empList)

Output: [[1]] [1] 1 2 3 4

[[2]] [1] “Debi” “Sandeep” “Subham” “Shiba”

[[3]] [1] 4

3. Dataframes:

• Dataframes are two-dimensional, tabular data objects used to store structured data.

• They are lists of vectors of equal lengths.

• Each column in a dataframe can have a different data type.

• Constraints: Column names are required, rows must have unique names, and each column must have the same number of items.

Example:

R program to illustrate dataframe A vector which is a character vector

Name = c(“Amiya”, “Raj”, “Asish”)

A vector which is a character vector Language = c(“R”, “Python”, “Java”)

A vector which is a numeric vector Age = c(22, 25, 45)

To create dataframe use data.frame command

and then pass each of the vectors

we have created as arguments

To the function data.frame()

df = data.frame(Name, Language, Age)

print(df)

Output: Name Language Age 1 Amiya R 22 2 Raj Python 25 3 Asish Java 45

4. Matrices:

• Matrices are two-dimensional arrays with elements of the same data type.

• They have fixed rows and columns.

Example :

R program to illustrate a matrix A = matrix(Taking sequence of elements c(1, 2, 3, 4, 5, 6, 7, 8, 9),

No of rows and columns nrow = 3, ncol = 3,

By default matrices are in column-wise order So this parameter decides how to arrange the matrix
byrow = TRUE
)

print(A)

Output: [,1] [,2] [,3] [1,] 1 2 3 [2,] 4 5 6 [3,] 7 8 9

5. Arrays:

a. Arrays are multi-dimensional data structures.

b. They can have more than two dimensions (e.g., 3D arrays).

Example

R program to illustrate an array

A = array(Taking sequence of elements c(1, 2, 3, 4, 5, 6, 7, 8),

Creating two rectangular matrices

Each with two rows and two columns dim = c(2, 2, 2)
)

print(A) Output: , , 1

```
     [,1] [,2]
```

[1,] 1 3 [2,] 2 4

, , 2

```
     [,1] [,2]
```

[1,] 5 7 [2,] 6 8

6. Factors:

• Factors represent categorical data (e.g., levels of a variable).

• They are useful for statistical modeling and plotting.

Example:

R program to illustrate factors

Creating factor using factor()

fac = factor(c(“Male”, “Female”, “Male”, “Male”, “Female”, “Male”, “Female”))

print(fac)

Output:

[1] Male Female Male Male Female Male Female

Levels: Female Male

7. Tibbles:

• Tibbles (from the dplyr package) are enhanced dataframes.

• They provide better printing, handling of missing values, and improved column names.

Example:

Load the tibble package

library(tibble) Create a tibble with three columns: name, age, and city

my_data <- tibble(name = c(“Vicky”, “Samuel”, “Amanda”), age = c(25, 30, 35), city = c(“Lagos”, “Uyo”, “Owerri”))

Print the tibble

print(my_data)

Output: name age city 1 Vicky 25 Lagos 2 Samuel 30 Uyo 3 Amanda 35 Owerri

Question 1d:

In R, there are several basic data types that allow you to store and manipulate different kinds of information.

• Numeric (Double):

• Numeric data types represent real numbers (including decimals).

Examples: 10.5, 55, 787 (R Data Types (w3schools.com))

• Integer:

• Integers are whole numbers without decimal points.

• Examples: 1L, 55L, 100L (where the letter “L” indicates an integer)

(3.1 Data types | An Introduction to R (intro2r.com))

• Complex:

• Complex numbers consist of a real part and an imaginary part (expressed as a + bi where i is the imaginary unit).

• Example: 9 + 3i (DATA TYPES in R 📗 [ATOMIC data types WITH EXAMPLES] (r-coder.com))

• Character (String):

• Character data types represent text or strings

• Examples: “k”, “R is exciting”, “FALSE”, “11.5” (Basic data types in R | R (datacamp.com))

• Logical (Boolean):

• Logical data types represent truth values (TRUE or FALSE).

• Example: TRUE (Data types in R - Stats and R)

Reference:

C Cummins, B Wasti, J Guo, B Cui… - … and Optimization …, 2022 - ieeexplore.ieee.org. Compilergym: Robust, performant compiler optimization environments for ai research. [PDF] Cited by 56

S Li, C Xu, J Liu, B Han - Ocean Engineering, 2023 - Elsevier. Data-driven docking control of autonomous double-ended ferries based on iterative learning model predictive control. [HTML] Cited by 12

A Gupta, R Chowdhury, A Chakrabarti… - arXiv preprint arXiv …, 2020 - arxiv.org. A 55-line code for large-scale parallel topology optimization in 2D and 3D. [PDF] Cited by 11

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:

2a: a. prints all the combinations of numbers from two dice.

Answers:

• Define the Sets of Numbers: Define two sets of numbers, Set1_Odd_Number and Set2_Even_Number, representing the numerical values on two different dice sets.

• Create All Combinations: Generate all possible combinations of one card from each set using the expand.grid() function.

• Calculate Sums: Calculate the sums of each combination using rowSums() function and store the sums in a new column named “Sum” in the Combinations data frame.

• Display All Combinations and Their Sums: Print the Combinations data frame and the total number of combinations.

```
#Define the number of dice in each set with their corresponding numerical val
ues
Set1_Odd_Number <- c(1,3,5,7,9,11)
Set2_Even_Number <- c(2,4,6,8,10,12)
```

```r
#Create all combinations of one card from each set
Combinations <- expand.grid(Set1_Odd_Number, Set2_Even_Number)
nrow(Combinations)
```

```
## [1] 36
```

```r
#Calculate the sums of each combination
Combinations$Sum <- rowSums(Combinations)
print(Combinations$Sum)
```

```
##  [1]  3  5  7  9 11 13  5  7  9 11 13 15  7  9 11 13 15 17  9 11 13 15 17
19 11
## [26] 13 15 17 19 21 13 15 17 19 21 23
```

```r
#Display all combinations and their sums
Total_Combination <- nrow(Combinations)
print(Total_Combination)
```

```
## [1] 36
```

Question 2b: b. uses your own function (not its internal function) to calculate the average value of all the sum of combinations.

Answer:

• Calculate Average Sum: Calculate the average value of all the sums of combinations by dividing the total sum of combinations by the total number of combinations.

```r
#calculate the average value of all the sum of combinations
Avg_sum_of_combination <- sum(Combinations$Sum/Total_Combination)
print(Avg_sum_of_combination)
```

```
## [1] 13
```

Question 2c: uses your own function (not its internal function) to find the maximum value of all the sum of combinations.

Answer: • Find Maximum Sum: Get the maximum value of all the sums of combinations using the max() function.

```r
#Get the maximum value of all the sum of combinations
Max_Sum_of_combination <- max(Combinations$Sum)
print(Max_Sum_of_combination)
```

```
## [1] 23
```

Question 2d: calculates the possibility of getting a sum of 15.

Answer:

• Calculate Chance of Sum Being 15: Count the number of combinations where the sum of dice cards is 15, then calculate the chance of getting a sum of 15 by dividing this count by the total number of combinations.

```r
#Calculate the chance that the sum of dice cards is 15
Sum_is_fifteen <- sum(Combinations$Sum == 15)
Probability_Sum_of_fiften <- Sum_is_fifteen / Total_Combination
print(Probability_Sum_of_fiften)
```

```
## [1] 0.1388889
```

Question 2e: calculates the possibility of getting a sum bigger than 15.

Answer: • Calculate Possibility of Sum Greater Than 15: Count the number of combinations where the sum of dice cards is greater than 15.

```r
#Calculate the possibility of getting a sum bigger than 15.
Sum_greater_fifteen <- sum(Combinations$Sum > 15)
Probability_Sum_greater_than_fiften <- Sum_greater_fifteen / Total_Combinatio
n

#Print
print(Probability_Sum_greater_than_fiften)
```

```
## [1] 0.2777778
```

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.

Answers:

• Define the S4 class “laptop” with slots for brand, model, and cpu_type.

• Create a constructor function new_laptop() to instantiate objects of class “laptop”.

• Define a generic function showCPU() to display the CPU type of a laptop.

• Define a method for the showCPU() generic function for objects of class “laptop” to display the CPU type.

• Then, create an instance of the “laptop” class representing an Apple Macbook Pro and display its CPU type using the showCPU() method.

```r
# Define the laptop(S4) class
setClass(
  "laptop",
  slots = list(
    brand = "character",
    model = "character",
    cpu_type = "character"
  ))

# Create a constructor function for the laptop class
new_laptop <- function(brand, model, cpu_type) {
```

```
  new("laptop", brand = brand, model = model, cpu_type = cpu_type)
}

# Create an Apple Macbook Pro object
Apple_macbook_pro <- new_laptop(brand = "Apple", model = "Macbook Pro", cpu_t
ype = "M2 Max Chips")

print(Apple_macbook_pro)

## An object of class "laptop"
## Slot "brand":
## [1] "Apple"
##
## Slot "model":
## [1] "Macbook Pro"
##
## Slot "cpu_type":
## [1] "M2 Max Chips"

# Define a method to show CPU type
setGeneric("showCPU", function(object) standardGeneric("showCPU"))

## [1] "showCPU"

setMethod("showCPU", "laptop", function(object) {
  cat("CPU Type:", object@cpu_type, "\n")
})

# Display CPU type of the Macbook Pro
showCPU (Apple_macbook_pro)

## CPU Type: M2 Max Chips
```

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?

Answer:

• Defining a function named fibonacci that takes an integer n as input (number of elements in the sequence).

• Initializing a vector result of size n filled with NA values.

• Setting the first two elements of result to 1 and 2, which are the starting values of the Fibonacci sequence.

• Using a loop to iterate from the third element (index 3) to the nth element (index n).

• Inside the loop, calculating the current element by adding the previous two elements in the result vector and storing it at the current index.

• After the loop, the function returns the result vector containing the Fibonacci sequence.

```r
fibonacci <- function(n) {
  # Initialize a vector to store the sequence
  result <- rep(NA, n)

  # Set the first two elements as 1 and 2
  result[1] <- 1
  result[2] <- 2

  # Calculate the remaining elements using a loop
  for (i in 3:n) {
    result[i] <- result[i - 1] + result[i - 2]
  }

  # Return the calculated sequence
  return(result)
}

# Get the first 20 numbers in the Fibonacci sequence
fibonacci_numbers <- fibonacci(20)

# Print the results
print(fibonacci_numbers)
```

```
##  [1]     1     2     3     5     8    13    21    34    55    89   144   2
33
## [13]   377   610   987  1597  2584  4181  6765 10946
```

Question 4b:

• This code defines a sample vector my_vector.

• The even_numbers vector is created using the modulo operator (%%). It checks if each element in my_vector leaves no remainder when divided by 2 (even numbers). The result is a logical vector with TRUE for even numbers and FALSE for odd numbers.

• The sum function is used on the even_numbers vector. Since TRUE is treated as 1 and FALSE as 0, the sum essentially counts the number of TRUE values, which represents the number of even numbers in the original vector.

```r
# Create a vector
my_vector <- c(fibonacci_numbers)

# Check for even numbers using modulo (remainder after division by 2)
even_numbers <- my_vector %% 2 == 0

# Count the number of even numbers (sum of TRUE values in the logical vector)
number_of_even <- sum(even_numbers)

# Print the result
print(number_of_even)
```

```
## [1] 7
```

Question 4c:

1. Define the vector of numbers (numbers <- c(fibonacci_numbers)) • This line assumes there's a previously defined variable named fibonacci_numbers. It likely contains a sequence of Fibonacci numbers (1, 2, 3, 5, …).

• The c() function combines elements into a vector. In this case, it creates a new vector named numbers that copies the contents of fibonacci_numbers.

2. Select even numbers using modulo (%) (even_numbers <- numbers %% 2 == 0) • This line creates a new logical vector named even_numbers. • The %% operator performs the modulo operation, which calculates the remainder after division. Here, it checks if each element in the numbers vector leaves no remainder when divided by 2 (even numbers).

• The comparison == 0 checks if the remainder is exactly zero.

• The result is stored in the even_numbers vector. It will have TRUE values for elements that are even numbers in the original numbers vector and FALSE for odd numbers.

3. Extract even numbers based on the logical vector (even_numbers_subset <- numbers[even_numbers])

• This line extracts elements from the numbers vector based on the condition specified in the square brackets [].

• Inside the brackets, even_numbers is the logical vector created in the previous step.

• By using numbers[even_numbers], the code selects only the elements from numbers where the corresponding value in even_numbers is TRUE (i.e., even numbers).

• The result is stored in a new vector named even_numbers_subset. It will contain only the even numbers from the original numbers vector.

```
# Define the vector of numbers
numbers <- c(fibonacci_numbers)

# Select even numbers using modulo (%)
even_numbers <- numbers %% 2 == 0

# Extract even numbers based on the logical vector
even_numbers_subset <- numbers[even_numbers]

# Print the even numbers
print(even_numbers_subset)
```

```
## [1]     2     8    34   144   610  2584 10946
```

```r
#Calculate the total sum of these even numbers
Total_Sum_of_even_numbers_subset <- sum(even_numbers_subset)

#Print sum of even numbers
print(Total_Sum_of_even_numbers_subset)

## [1] 14328
```

Question 5: GSE9476_series_matrix.txt (tab-delimited) and GSE9476_meta_info.txt are the microarray gene expression dataset and sample information table about patients with Acute Myeloid Leukemia (AML) and normal subjects as Controls. The expression data have been log2 transformed. Please complete the following tasks.

Question 5.1: 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).

Answer:

Import data:

a. Read the gene expression dataset from a CSV file located at the specified file path using the read.csv function. The dataset is assumed to have a header row, and the data starts from row 84 (skipping the first 83 rows). Missing values are filled with TRUE, and the first column is used as row names.

b. Read the metadata for the gene expression dataset from another CSV file located at the specified file path using the read.csv function. The dataset is assumed to have a header row, and the data starts from row 2 (skipping the first row). Missing values are filled with TRUE.

c.Count the number of patients with AML disease by summing the logical values where the Disease_status column in the metadata dataset is equal to “AML”.

d.The result is stored in the AML_count variable and printed to the console.

e.Count the number of control patients by summing the logical values where the Disease_status column in the metadata dataset is equal to “Control”. The result is stored in the control_count variable and printed to the console.

f. Count the number of genes in the gene expression dataset by getting the number of rows using the nrow function. The result is stored in the Genes_count variable and printed to the console.

```r
#import data
Gene.exp.dataset.assess.raw <- read.csv("C:/Users/agape/OneDrive/Desktop/2nd sem. Exams/R assessment Dataset/Gene expression dataset assessment.csv", header = TRUE, sep = ",", skip = 83, fill = TRUE, row.names = 1)
metadata_gene_exp <- read.csv("C:/Users/agape/OneDrive/Desktop/2nd sem. Exams/R assessment Dataset/Gene expression dataset assessment metadata.csv", header = TRUE, sep = ",", fill = TRUE, skip = 1)
```

```r
Gene.exp.dataset.assess. <- na.omit(Gene.exp.dataset.assess.raw)

#Question 5.1
#Number of patients with AML disease
AML_count <- sum(metadata_gene_exp$Disease_status =="AML")
print(AML_count)
```

```
## [1] 26
```

```r
#Number of control patients
control_count <- sum(metadata_gene_exp$Disease_status == "Control")
print(control_count)
```

```
## [1] 38
```

```r
#Number of genes in dataset
Genes_count <- nrow(Gene.exp.dataset.assess.)
print(Genes_count)
```

```
## [1] 22283
```

Question 5.2:Create your R function to calculate the average age of AML patients and Control subjects separately. Check your results by using R mean function.

Answers:

Filter AML patients' ages:

Identify rows in the metadata_gene_exp dataframe where the Disease_status column equals "AML".

Select the corresponding ages from the Age column.

Calculate the mean (average) age of these AML patients.

Filter Control subjects' ages:

Identify rows in the metadata_gene_exp dataframe where the Disease_status column equals "Control".

Select the corresponding ages from the Age column.

Calculate the mean (average) age of these Control subjects.

Print the results:

Display the average age of AML patients.

Display the average age of Control subjects.

```r
#calculate the average age of AML patients and Control subjects separately
# Calculate average age for AML patients
AML_average_age <- mean(metadata_gene_exp$Age[metadata_gene_exp$Disease_statu
s == "AML"])

# Calculate average age for Control subjects
control_average_age <- mean(metadata_gene_exp$Age[metadata_gene_exp$Disease_s
tatus == "Control"], na.rm = TRUE)

# Print the results
cat("Average Age (AML):", AML_average_age, "\n")
```

```
## Average Age (AML): 56.11538
```

```r
cat("Average Age (Control):", control_average_age, "\n")
```

```
## Average Age (Control): 33.33333
```

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

Answers:

Subset the data: Extract the ages of individuals with AML (Acute Myeloid Leukemia) and those in the control group from the dataframe metadata_gene_exp. This is done using boolean indexing

(metadata_gene_exp$Disease_status == "AML" and metadata_gene_exp$Disease_status == “Control”).

Perform the t-test: Use the t.test() function in R to perform an independent two-sample t-test. The first argument is the vector of ages for individuals with AML, and the second argument is the vector of ages for individuals in the control group. Specify na.action = na.exclude to exclude any missing values from the analysis.

Store the result: Assign the result of the t-test to the variable t_test_result.

Print the result: Use the print() function to display the result of the t-test, which includes statistics such as the t-statistic, degrees of freedom, p-value, and confidence interval for the difference in means.

```r
#Use t_test to check for significant difference between sample ages of AMLs a
nd Controls? Assume the age follows a normal distribution.
t_test_result <- t.test(metadata_gene_exp$Age[metadata_gene_exp$Disease_statu
s == "AML"],
                        metadata_gene_exp$Age[metadata_gene_exp$Disease_statu
s == "Control"],
                        na.action = na.exclude)
print(t_test_result)
```

```
##
##  Welch Two Sample t-test
##
## data:  metadata_gene_exp$Age[metadata_gene_exp$Disease_status == "AML"] an
d metadata_gene_exp$Age[metadata_gene_exp$Disease_status == "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.

Answer:

Read Data:

Load the gene expression dataset from the specified file path using read.csv(), skipping the first 83 rows since they contain metadata.

Load the metadata file using read.csv().

Tidy Data Transformation:

Load the tidyr library.

Use pivot_longer() to transform the gene expression dataset from wide to long format, with columns rearranged such that each row corresponds to a single observation (gene expression) and columns for gene IDs and expression values.

Ensure to drop any NA values from the expression column using values_drop_na = TRUE.

Add Metadata:

Add a new column named “disease.status” to the tidy data frame and populate it with disease status information from the metadata file. Use rep() to repeat the disease status values to match the length of the tidy data frame.

Visualization:

Load the ggplot2 library.

Create a boxplot using ggplot() with the tidy gene expression dataset.

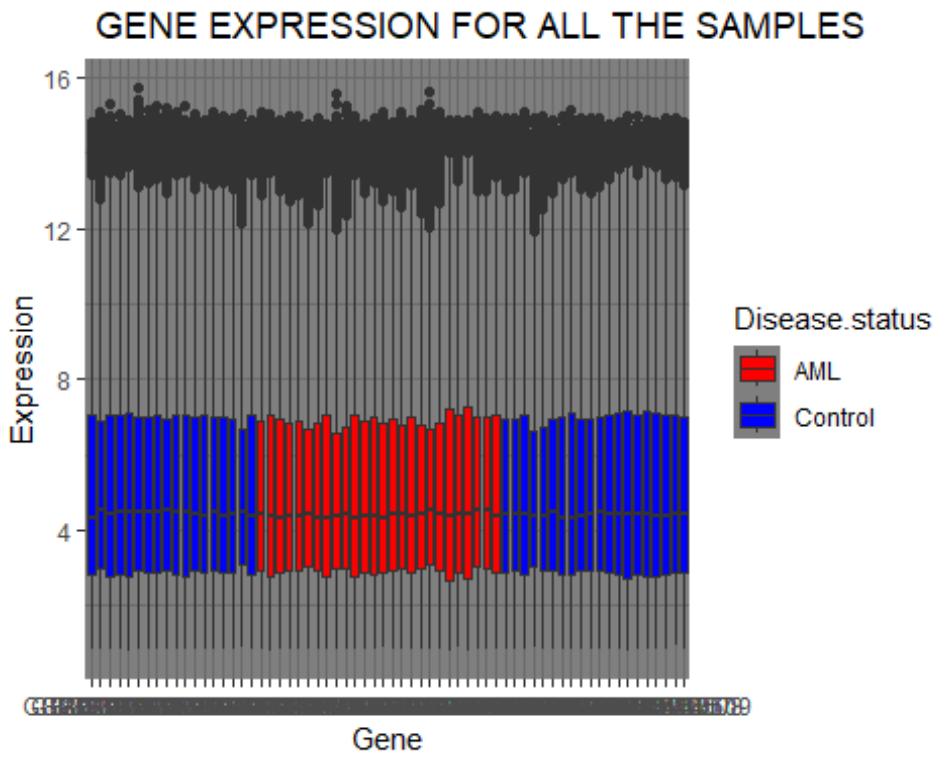
Map the x-axis to genes, y-axis to expression values, and fill colors based on disease status.

Customize the boxplot appearance, such as setting fill colors manually, using a dark theme, and adding appropriate axis labels and title.

```r
Gene.exp.dataset <- read.csv("C:/Users/agape/OneDrive/Desktop/2nd sem. Exams/R assessment Dataset/Gene expression dataset assessment.csv", header = TRUE, sep = ",", skip = 83, fill = TRUE)
metadata_gene <- read.csv("C:/Users/agape/OneDrive/Desktop/2nd sem. Exams/R assessment Dataset/Gene expression dataset assessment metadata.csv", header = TRUE, sep = ",", skip = 1, fill = TRUE)


library(tidyr)

tidy_Gene.exp.dataset<- pivot_longer(
  data = Gene.exp.dataset,
  col = -ID_REF,
  names_to = "Gene",
  values_to ="Expression",
  values_drop_na = TRUE
)

#To add a new column to the tidy data and populating with the disease status
tidy_Gene.exp.dataset$disease.status <- rep(metadata_gene$Disease_status, length.out = nrow(tidy_Gene.exp.dataset))

library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```r
#Making a boxplot of gene expression for all the samples, marking data points in red for AML samples and blue for Control samples
ggplot(data =tidy_Gene.exp.dataset, aes(x = Gene, y = Expression, fill= disease.status)) +
  geom_boxplot() +
  scale_fill_manual(values = c("red","blue")) +
  theme_dark() +
  labs(title = " GENE EXPRESSION FOR ALL THE SAMPLES", x ="Gene", y ="Expression", fill = "Disease.status")
```



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.

Answers:

Read Data:

Load the gene expression dataset from the specified file path using read.csv(), skipping the first 83 rows since they contain metadata. Also, set the row names to be the first column of the dataset. Load the metadata file using read.csv(). Subset Data:

Subset the gene expression dataset to extract samples labeled as “AML” and “Control” based on disease status information from the metadata. Calculate Statistics:

Create a data frame named Gene_Statistics to store statistics for each gene. Calculate mean, standard deviation, minimum, and maximum expression values for each gene in both AML and Control samples using apply() function. Calculate the Log Fold Change (Log_FC) as the difference between the mean expression values of Control and AML samples.

```
AML_Samples <- Gene.exp.dataset.assess.[, metadata_gene_exp$Disease_status ==
"AML"]
```

```r
Control_Samples <- Gene.exp.dataset.assess.[, metadata_gene_exp$Disease_statu
s == "Control"]

Gene.Exp.table <- na.omit(Gene.exp.dataset.assess.)

#Combine all the statistics into one data frame including the fold change
Gene_Statistics <- data.frame(

  Mean_AML = apply(AML_Samples, 1, mean),
  Sd_AML = apply(AML_Samples, 1, sd),
  min_AML = apply(AML_Samples, 1, min),
  max_AML = apply(AML_Samples, 1, max),

  Mean_Control = apply(Control_Samples, 1, mean),
  Sd_Control = apply(Control_Samples, 1, sd),
  min_Control = apply(Control_Samples, 1, min),
  max_Control = apply(Control_Samples, 1, max)
)

Gene_Statistics <- data.frame(

  Mean_AML = apply(AML_Samples, 1, mean),
  Sd_AML = apply(AML_Samples, 1, sd),
  min_AML = apply(AML_Samples, 1, min),
  max_AML = apply(AML_Samples, 1, max),

  Mean_Control = apply(Control_Samples, 1, mean),
  Sd_Control = apply(Control_Samples, 1, sd),
  min_Control = apply(Control_Samples, 1, min),
  max_Control = apply(Control_Samples, 1, max),
  Log_Fc = (Gene_Statistics$Mean_AML - Gene_Statistics$Mean_Control)

)
```

Question 5.6:With the table created above, make a scatterplot of the mean expression of AML vs. Control samples. Are mean_AML and mean_Control significantly different?

Answers:

Load Required Library:

Load the ggplot2 library to use its functions for data visualization.

Create Scatter Plot:

Use ggplot() function to initialize the plot with Gene_Statistics dataframe as the data source.

Map the x-axis to the mean expression in Control samples (Mean_Control) and the y-axis to the mean expression in AML samples (Mean_AML) using aes() function.

Add points to the plot using geom_point() to create a scatter plot, setting alpha parameter to control transparency.

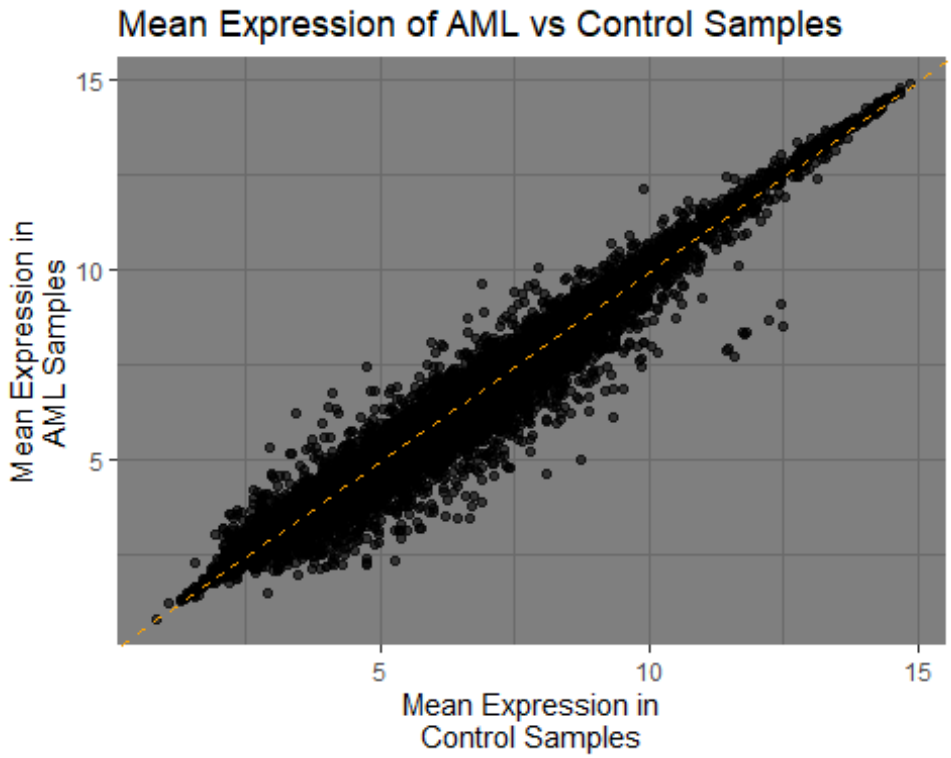
Add a dashed reference line with a slope of 1 and intercept of 0 using geom_abline() function. This line indicates where the mean expression values of AML and Control samples would be equal.

Customize Plot:

Apply a dark theme to the plot using theme_dark() function.

Add appropriate axis labels and title using labs() function.

```
# Plotting a scattered diagram of mean expression of AML against Control samp
les. A

library(ggplot2)
ggplot(Gene_Statistics, aes(x =
                              Mean_Control , y = Mean_AML)) +
  geom_point(alpha = 0.6) +
  geom_abline(slope = 1, intercept
              = 0, linetype = "dashed", color =
                "Orange") +
  theme_dark() +
  labs(title = "Mean Expression of AML vs Control Samples",
       x = "Mean Expression in
Control Samples",
       y = "Mean Expression in
AML Samples")
```



Perform T-test:

Use the t.test() function to perform a t-test comparing the mean expression values of AML samples (Gene_Statistics$Mean_AML) and Control samples (Gene_Statistics$Mean_Control). Store the result of the t-test in a variable named t_test_Mean. Print Results:

Print the result of the t-test using the print() function to display the test statistic, degrees of freedom, p-value, and whether the p-value is less than the significance level (usually 0.05). Interpret Results:

Check the p-value from the t-test result. If the p-value is greater than 0.05, it indicates that there is not a significant difference between the mean expression values of AML and Control samples.

```
#T-test to check significant difference for Mean_AML and Mean_Control
t_test_Mean <- t.test(Gene_Statistics$Mean_AML, Gene_Statistics$Mean_Control,
                      na.action = na.exclude)
print(t_test_Mean)

##
##  Welch Two Sample t-test
##
## data:  Gene_Statistics$Mean_AML and Gene_Statistics$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

#Pvalue = 0.4539 > 0.05 hence they are not significantly different.
```

Preallocate Vectors:

Create empty numeric vectors named t_test_P.Val and P.Val with a length equal to the number of genes (number of rows in Control_Samples). Loop through Genes:

Use a for loop to iterate over each gene in the samples data. Perform T-test:

Inside the loop, perform a t-test using the t.test() function comparing the expression values of the current gene in AML samples (AML_Samples[i,]) and Control samples (Control_Samples[i,]). Store Results:

Store the t-statistic and p-value from the t-test result in the corresponding position in the preallocated vectors t_test_P.Val and P.Val. Print Results:

After the loop, print the vectors t_test_P.Val and P.Val to display the t-statistics and p-values for each gene.

```
Gene.Exp.table <- na.omit(Gene.exp.dataset.assess.)
```

T-test of AML_Samples and Control_Samples Preallocate vectors to store t-statistics and p-values

```
AML_Samples <- Gene.Exp.table[ , metadata_gene_exp$Disease_status == "AML"]
Control_Samples <- Gene.Exp.table [ , metadata_gene_exp$Disease_status == "Co
ntrol"]

t_test_P.Val <- numeric(nrow(AML_Samples))

P.Val <-numeric(nrow(AML_Samples))

# Loop through each gene to perform t-tests

  for (i in 1:nrow(AML_Samples)) {
  # Performing t-test
t_test_Combine <- t.test(AML_Samples[i,], Control_Samples[i,], paired = FALSE
)

  # Store the t-statistic and p-value
  t_test_P.Val[i] <- t_test_Combine$statistic
  P.Val[i] <- t_test_Combine$p.value
}
Data <- data.frame( T_Statistic = unlist(t_test_P.Val), P_Value= P.Val)

#merging the tab
```

```
Combined.Data <- cbind(Gene_Statistics,Data)
```

Question 5.8:

a.) mean_AML > 3 and mean_Control > 3

Answer:

Filter the Data: Filter the Combined_Data dataframe to retain only rows where both the mean values in the "Mean_Control" and "Mean_AML" columns are greater than 3.

Count Filtered Rows: Count the number of rows in the filtered dataset Filter_Genes_Mean. Then return number of rows.

```
Filter_Genes_Mean <- Combined.Data[Combined.Data $Mean_Control > 3 &
                                   Combined.Data$Mean_AML > 3, ]
 nrow(Filter_Genes_Mean)

## [1] 16026
```

b.) max_AML < 14 or max_Control < 14

Answer:

Step 1: Filter rows from Combined_Data:

Identify rows where either the maximum value in the "max_Control" column is less than 14 or the maximum value in the "max_AML" column is less than 14.

Create a new dataset named Filter_Genes_Max containing only the rows that meet the filter condition.

Use the nrow() function to determine the number of rows in the Filter_Genes_Max dataset.

```
Filter_Genes_Max <- Combined.Data[Combined.Data$max_Control < 14 |
                                    Combined.Data$max_AML < 14, ]
nrow(Filter_Genes_Max)

## [1] 22171
```

c)Apply a) and b) filters together which is more stringent than applying a) or b) alone
Answer:

Filter rows of the Combined_Data dataframe based on conditions:

Rows where the mean value in the "Mean_Control" column is greater than 3 AND the mean value in the "Mean_AML" column is greater than 3.

Additionally, either the maximum value in the "max_AML" column is less than 14 OR the maximum value in the "max_Control" column is less than 14.

Assign the filtered result to a new dataframe named Filter_Genes_Mean_Max.

Calculate the number of rows in the Filter_Genes_Mean_Max dataframe using the nrow() function.

```
Filter_Genes_Mean_Max <- Combined.Data[(Combined.Data$Mean_Control > 3 &
                            Combined.Data$Mean_AML > 3) & (Combined.Data$max_A
ML < 14 | Combined.Data$max_Control < 14), ]
nrow(Filter_Genes_Mean_Max)

## [1] 15914
```

#Applying A and B together is more stringent the applying just A or B.

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_Statistics <- Filter_Genes_Mean_Max[abs(Filter_Genes_Mean_Max$Log_Fc >
1) & (Filter_Genes_Mean_Max$P_Value<0.05),]
nrow(Filter_Statistics)

## [1] 136

Filter_P_FC_t_test <- Combined.Data[abs(Combined.Data$Log_Fc)>1 & (Combined.D
ata$P_Value < 0.05), ]
nrow(Filter_P_FC_t_test)

## [1] 709

#Filter for Upregulated nd Downregulated
Upregulated <- Filter_Genes_Mean_Max[(Filter_Genes_Mean_Max$Log_Fc > 0) & (Fi
lter_Genes_Mean_Max$P_Value<0.05), ]
nrow(Upregulated)

## [1] 2575

Downregulated <-Filter_Genes_Mean_Max[(Filter_Genes_Mean_Max$Log_Fc < 0) & (F
ilter_Genes_Mean_Max$P_Value<0.05), ]
nrow(Downregulated)

## [1] 2321

#top 10 most upregulated DEGs and 10 most down-regulated DEGs including
#their p-value, log2FC, and mean values in each sample group
Upregulated_Top10_Genes <- head(Upregulated, 10)
#using the column positions Sd_AML = 2,min_AML = 3, max_AML = 4, Sd_Control =
6, min_Control =7, max_Control=8, T_Static = 10
Upregulated_Top10_Genes_remove <- Upregulated_Top10_Genes[,c(-2, -3, -4, -6,
-7, -8, -10)]
print(Upregulated_Top10_Genes_remove)
```

```
##              Mean_AML Mean_Control    Log_Fc      P_Value
## 121_at       7.449728     7.176393 0.2733345 1.518826e-03
## 1431_at      3.250006     3.134428 0.1155782 3.495726e-02
## 1487_at      5.847557     5.696597 0.1509598 2.864684e-02
## 160020_at    4.758708     4.578213 0.1804952 1.049015e-02
## 177_at       3.744790     3.392353 0.3524371 8.719128e-05
## 200001_at   10.230331     9.831523 0.3988082 7.327972e-04
## 200004_at   11.784068    11.600926 0.1831417 1.666260e-02
## 200007_at   12.271725    12.123623 0.1481026 2.723837e-02
## 200011_s_at  9.318028     8.885527 0.4325007 3.274727e-04
## 200017_at   13.252933    13.091976 0.1609571 2.920340e-02
```

```
Downregulated_Top10_Genes <- head(Downregulated, 10)
#using the column positions Sd_AML = 2,min_AML = 3, max_AML = 4, Sd_Control =
6, min_Control =7, max_Control=8, T_Static = 10
Downregulated_Top10_Genes_remove <- Downregulated_Top10_Genes[,c(-2, -3, -4,
-6, -7, -8, -10)]
print(Downregulated_Top10_Genes_remove)
```

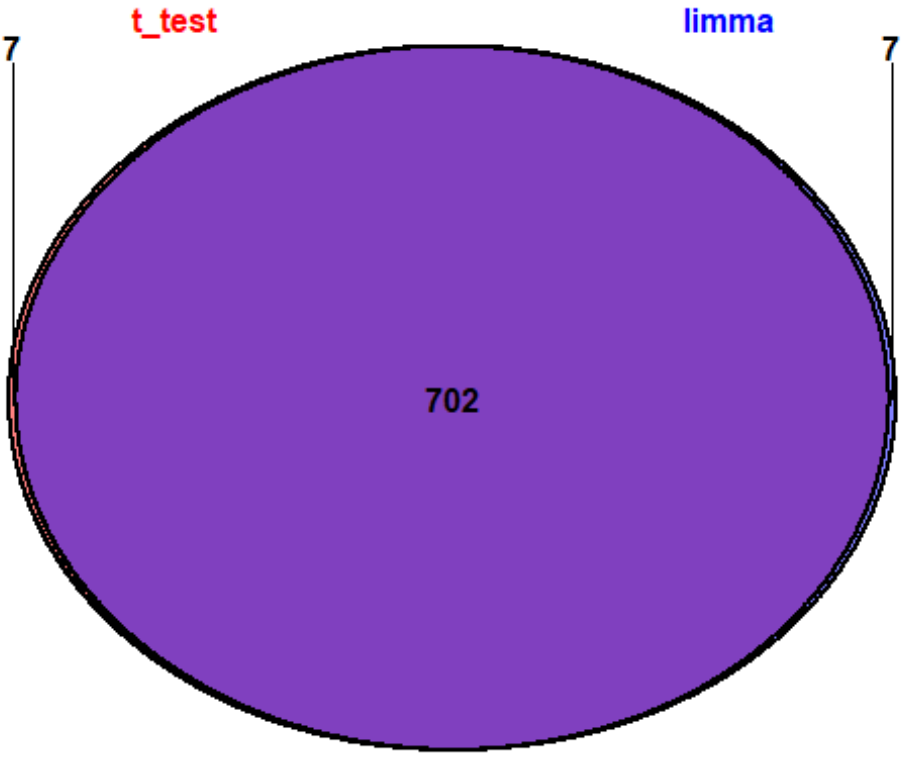
```
##              Mean_AML Mean_Control     Log_Fc      P_Value
## 117_at       5.352038     6.475112 -1.1230740 3.596652e-02
## 1729_at      6.044211     6.620357 -0.5761459 2.696712e-04
## 200035_at    8.397745     8.651002 -0.2532570 1.390343e-02
## 200050_at    8.261935     8.904615 -0.6426803 1.547248e-02
## 200066_at    8.799529     9.029772 -0.2302427 1.365264e-03
## 200069_at    5.066167     5.766136 -0.6999692 8.788131e-04
## 200597_at    7.101229     7.466525 -0.3652964 2.795891e-02
## 200602_at    4.919461     7.097208 -2.1777464 2.997055e-05
## 200603_at   10.403300    10.989821 -0.5865210 7.441166e-03
## 200604_s_at  7.666749     8.570604 -0.9038551 2.923918e-03
```

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.

```
library(limma)

Gene.exp.dataset <- read.csv("C:/Users/agape/OneDrive/Desktop/2nd sem. Exams/
R assessment Dataset/Gene expression dataset assessment.csv", header = TRUE,
sep = ",", skip = 83, fill = TRUE)
metadata_gene <- read.csv("C:/Users/agape/OneDrive/Desktop/2nd sem. Exams/R a
ssessment Dataset/Gene expression dataset assessment metadata.csv", header =
TRUE, sep = ",", skip = 1, fill = TRUE)

#once the data inputs into R, the gene name has to be the row names for limma
so they aren't classed as data
rownames(Gene.exp.dataset) <-  Gene.exp.dataset$ID_REF

#remove the original gene name column
Gene.exp.dataset$ID_REF <- NULL
```

```r
#Convert disease status to a factor for analysis. Necessary for categorical a
nalysis such as when specifying a variable in a model formula.
metadata_gene$Disease_status<- as.factor(metadata_gene$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.
#With the syntax, each coefficient directly represents the condition's effect
. Without it, coefficients represent the difference from the reference condit
ion.

design <- model.matrix(~ 0 + metadata_gene$Disease_status)

#Rename the columns of the design matrix to match the levels of the Disease_s
tatus in sample_info.
#This makes the output more interpretable by using disease status names inste
ad of automatically generated #names (e.g., sample_info$Disease_statusControl
becomes Control).
colnames(design) <-levels(metadata_gene$Disease_status)

#Fit the linear model
fit <- lmFit(Gene.exp.dataset, design)

#Determine the levels in the make Contrasts call
contrast.matrix <- makeContrasts(AML_vs_Control = AML - Control, levels = des
ign)

#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,this sorts by p value and list
an infinite number of genes
AML_Control_results <- topTable(fit2, coef="AML_vs_Control", adjust.method="B
H", sort.by="P", n=Inf)
nrow(AML_Control_results)

## [1] 22284

#Filter your data for logFC greater than 1 and adjusted P value less than 0.0
5
filtered_genes_Limma <- AML_Control_results[abs(AML_Control_results$logFC) >
1 & AML_Control_results$P.Value < 0.05, ]
nrow(filtered_genes_Limma)

## [1] 709
```

```r
#Venn Diagram for results

library(VennDiagram)

## Warning: package 'VennDiagram' was built under R version 4.3.3

## Loading required package: grid

## Loading required package: futile.logger

## Warning: package 'futile.logger' was built under R version 4.3.3

# summarize test results as "up", "down" or "not expressed"
T_test <- rownames(Filter_P_FC_t_test)

CLimma <- rownames(filtered_genes_Limma)

List_of_both <- list( t_test = T_test, limma = CLimma)

# Generate Venn diagram

Plot_Venn_Diagram <- venn.diagram(
  x = List_of_both,
  category.names = c("t_test","limma"),
  filename = NULL, # Use NULL to plot on screen
  output = "display",# Ensure it displays on screen in most environments
  height = 450,
  width = 450,
  cat.fontfamily = "sans",
  resolution = 300,
  compression = "lzw",
  lwd = 2,
  col = "black",
  fill = c("red", "blue"),
  cat.fontface = "bold",
  cat.dist = 0.05,
  alpha = 0.5,
  cex = 1,
  fontfamily = "sans",
  fontface = "bold",
  cat.cex = 1,
  cat.col = c("red", "blue"),
  cat.pos = c(330,30)
)
# Display the plot (this line is necessary in some environments)
if (!is.null(Plot_Venn_Diagram)) { # Check if venn.plot is not NULL
  grid.draw(Plot_Venn_Diagram)}
```




```

#Answer: The Venn diagram indicates that there are 543 overlapped DEGs identified by the two different approaches, limma and t_test.
```