SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

SAVEETHA SCHOOL OF ENGINEERING

Engineer to Excel

LABORATORY MANUAL

ITA0447 Statistics with R Programming for NLP

Submitted By :

Register Number :

LIST OF LAB EXPERIMENTS

A. **BASIC OPERATIONS IN R**
B. **DATA STRUCTURES IN R**
C. **WORKING WITH LOOPING & FUNCTION IN R**
D. **IMPLEMENTATION OF VECTOR RECYCLING, APPLY FAMILY & RECURSION**
E. **CREATE AND MANIPULATING DATA FRAMES**
F. **RESHAPE FUNCTION IN R**
G. **MELTING AND CASTING IN R**
H. **FILE MANUPULATION IN R**
I. **UNIVARIATE ANALYSIS IN R - MEASURES OF CENTRAL TENDENCY**
J. **UNIVARIATE ANALYSIS IN R - MEASURES OF DISPERSION**
K. **BIVARIATE ANALYSIS IN R -CROSSTAB, COVARIANCE, CORRELATION**
L. **VISUALIZATION IN R**
M. **LINEAR REGRESSION ANALYSIS IN R**
N. **MULTIPLE REGRESSION ANALYSIS IN R**
O. **LOGISTIC REGRESSION ANALYSIS IN R**
P. **POISSON REGRESSION ANALYSIS IN R**

A. BASIC OPERATIONS IN R

1.Write the Commands to Perform Basic Arithmetic in R.
Aim:

Procedure:

Code & Output:

1. Addition (+) 2 + 3 # Output: [1] 5 # Adding a sequence of numbers 1:5 + 3 # Output: [1] 4 5 6 7 8	2. Subtraction (-) # Subtracting two numbers 10 - 3 # Output: [1] 7 # Subtracting a sequence of numbers 1:5 - 3 # Output: [1] -2 -1 0 1 2	3. Multiplication (*) # Multiplying two numbers 2 * 3 # Output: [1] 6 # Multiplying a sequence of numbers 1:5 * 3 # Output: [1] 3 6 9 12 15
4. Division (/) # Dividing two numbers 10 / 5 # Output: [1] 2 # Dividing a sequence of numbers 1:5 / 2 # Output: [1] 0.5 1.0 1.5 2.0 2.5	5. Exponentiation (^) # Raising a number to a power 2^3 # Output: [1] 8 # Raising a sequence of numbers to a power 1:5^2 # Output: [1] 1 4 9 16 25	6. Modulus (%%) # Finding the remainder of a division 10 %% 3 # Output: [1] 1 # Finding the remainder of a sequence of divisions 1:5 %% 2 # Output: [1] 1 0 1 0 1

Result:

2. Declare Variables In R And Also Write The Commands For Retrieving The Value of The Stored Variables In R Console.
Aim:

Procedure:

Code:	**Output:**

# declare variables 	
x <- 5
y <- "Hello, World!"
z <- TRUE
# retrieve values and display output 	
x
	# Output: [1] 5
	y
	# Output: [1] "Hello, World!"
	z
	# Output: [1] TRUE

Result:

3. Write R script to calculate the area of Rectangle.
Aim:

Procedure:

Code:	**Output:**

# declare variables 	
length <- 5
width <- 3
 	
# calculate area 	
area <- length * width
 	
# display result 	
cat("The area of the rectangle is", area)
	The area of the rectangle is 15

Result:

4.Enumerate The Process To Check Whether A Given Input Is Numeric, Integer, Double, Complex in R.
Aim:

Procedure:

Code:	**Output:**

# declare variables 	
x <- 5
y <- 3.14
z <- 2+3i
a <- "hello"
 	
# check if variables are numeric 	
is.numeric(x) # TRUE
is.numeric(y) # TRUE
is.numeric(z) # TRUE
is.numeric(a) # FALSE
 	
# check if variables are integer 	
is.integer(x) # TRUE
is.integer(y) # FALSE
is.integer(z) # FALSE
is.integer(a) # FALSE
 	
# check if variables are double 	
is.double(x) # TRUE
is.double(y) # TRUE
is.double(z) # FALSE
is.double(a) # FALSE
 	
# check if variables are complex 	
is.complex(x) # FALSE
is.complex(y) # FALSE
is.complex(z) # TRUE
is.complex(a) # FALSE
	> is.numeric(x) # TRUE
	[1] TRUE
	> is.numeric(y) # TRUE
	[1] TRUE
	> is.numeric(z) # TRUE
	[1] TRUE
	> is.numeric(a) # FALSE
	[1] FALSE
	> is.integer(x) # TRUE
	[1] TRUE
	> is.integer(y) # FALSE
	[1] FALSE
	> is.integer(z) # FALSE
	[1] FALSE
	> is.integer(a) # FALSE
	[1] FALSE
	> is.double(x) # TRUE
	[1] TRUE
	> is.double(y) # TRUE
	[1] TRUE
	> is.double(z) # FALSE
	[1] FALSE
	> is.double(a) # FALSE
	[1] FALSE
	> is.complex(x) # FALSE
	[1] FALSE
	> is.complex(y) # FALSE
	[1] FALSE
	> is.complex(z) # TRUE
	[1] TRUE
	> is.complex(a) # FALSE
	[1] FALSE

Result:

5. Illustration of Vector Arithmetic.

Aim:

Procedure:

Code & Output:

# create two vectors 	
a <- c(1, 2, 3)
b <- c(4, 5, 6)
	# add the two vectors
	c <- a + b
	print(c) # output: 5 7 9
		# subtract the two vectors
		d <- b - a
		print(d) # output: 3 3 3

# multiply the two vectors element-wise 	
e <- a * b
print(e) # output: 4 10 18
	# divide the two vectors element-wise
	f <- b / a
	print(f) # output: 4 2.5 2
		# calculate the dot product of the two vectors
		g <- sum(a * b)
		print(g) # output: 32

Result:

6. Write an R Program to Take Input From User.

Input name as "Jack" and age as 17.

The program should display the output as

"Hai , Jack next year you will be 18 years old"

Aim:

Procedure:

Code:	**Output:**

# take input from user 	
name <- readline(prompt = "Enter your name: ")
age <- readline(prompt = "Enter your age: ")
 	
# convert age to integer and add 1 	
age_next_year <- as.integer(age) + 1
 	
# display output 	
cat("Hai, ", name, " next year you will be ",
age_next_year, " years old.", "\n")
	Enter your name: Jack
	Enter your age: 17
	Hai, Jack next year you will be 18 years old.

Result:

B.DATA STRUCTURES IN R

1) Perform Addition, Subtraction, Multiplication and Transpose of a Matrix in R
Aim:

Procedure:

Program:	**Output:**

matrix1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
matrix2 <- matrix(c(7, 8, 9, 10, 11, 12), nrow = 2, ncol = 3)
addition_result <- matrix1 + matrix2
subtraction_result <- matrix1 - matrix2
multiplication_result <- matrix1 %*% t(matrix2) # Use %*% for matrix multiplication
transpose_result <- t(matrix1)
print("Addition Result:")
print(addition_result)
print("Subtraction Result:")
print(subtraction_result)
print("Multiplication Result:")
print(multiplication_result)
print("Transpose Result:")
print(transpose_result)
	[1] "Addition Result:"
	[,1] [,2] [,3]
	[1,] 8 10 12
	[2,] 14 16 18
	
	[1] "Subtraction Result:"
	[,1] [,2] [,3]
	[1,] -6 -6 -6
	[2,] -6 -6 -6
	
	[1] "Multiplication Result:"
	[,1] [,2]
	[1,] 50 68
	[2,] 122 167
	
	[1] "Transpose Result:"
	[,1] [,2]
	[1,] 1 4
	[2,] 2 5
	[3,] 3 6

Result:

2) Find Transpose of a matrix in R and deconstruct a matrix.
Aim:

Procedure:

Program:	**Output:**

# Transpose of Matrix 	
mat <- matrix(1:6, nrow = 2)
transpose <- t(mat)
print(mat)
print(transpose)
 	
# Deconstruct of matrix 	
mat <- matrix(1:6, nrow = 2)
element_1 <- mat[1, 1] # Extracts the element at row 1, column 1
element_2 <- mat[2, 3] # Extracts the element at row 2, column 3
row_1 <- mat[1,]
column_2 <- mat[, 2]
print(element_1)
	Output:
	[,1] [,2] [,3]
	[1,] 1 3 5
	[2,] 2 4 6
	
	[,1] [,2]
	[1,] 1 2
	[2,] 3 4
	[3,] 5 6
	
	Output:
	
	[1] 1

print(element_2) print(row_1) print(column_2)	[1] 6 [1] 1 3 5 [1] 3 4

Result:

3) Perform the operation of combining matrices in R using cbind() and rbind() functions.
Aim :

Procedure:

Program:	**Output:**
mat1 <- matrix(1:4, nrow = 2) mat2 <- matrix(5:8, nrow = 2) combined <- cbind(mat1, mat2) print(combined)	[,1] [,2] [,3] [,4] [1,] 1 3 5 7 [2,] 2 4 6 8 **Output:** [,1] [,2] [1,] 1 2 [2,] 3 4 [3,] 5 6 [4,] 7 8

Result:

4) Perform array manipulation in R .
Aim:

Procedure:

Program:	**Output:**

#Reshaping Arrays in R program
mat <- matrix(1:12, nrow = 3, ncol = 4)
reshaped_array <- array(mat, dim = c(2, 6))
print(reshaped_array)
 	
#Subsetting Arrays in R
mat <- matrix(1:12, nrow = 3, ncol = 4)
subset_array <- mat[2,]
print(subset_array)
	[,1] [,2] [,3] [,4] [,5] [,6]
	[1,] 1 4 7 10 1 4
	[2,] 2 5 8 11 2 5
	
	Output:
	[1] 2 5 8 11
	
	Output:
	[,1] [,2] [,3] [,4] [,5] [,6]

#Merging Arrays in R mat1 <- matrix(1:6, nrow = 2) mat2 <- matrix(7:12, nrow = 2) merged_array <- cbind(mat1, mat2) print(merged_array) **#Transforming Arrays** mat <- matrix(1:12, nrow = 3, ncol = 4) transformed_array <- apply(mat, 2, sum) print(transformed_array)	[1,] 1 3 5 7 9 11 [2,] 2 4 6 8 10 12 **Output:** [1] 6 15 24 33

Result:

5) Perform calculations across array elements in an array using the apply() function.
Aim:

Procedure:

Program: my_array <- array(1:12, dim = c(3, 4)) print(my_array) product <- function(x) { prod(x)} result <- apply(my_array, 1, product) print(result)	**Output:**> print(my_array) [,1] [,2] [,3] [,4] [1,] 1 4 7 10 [2,] 2 5 8 11 [3,] 3 6 9 12 > print(result) [1] 280 1760 11880

Result:

6) Create a data frame and print the structure of the data frame in R.
Aim:

Procedure:

Programm: df <- data.frame(Name = c("John", "Jane", "Mike"), Age = c(25, 30, 35), Height = c(170, 165, 180), StringsAsFactors = FALSE)	**Output:** 'data.frame': 3 obs. of 3 variables: $ Name : chr "John" "Jane" "Mike" $ Age : num 25 30 35 $ Height: num 170 165 180

str(df)	

Result :

7) Demonstrate the creation of S3 class and S4 class in R.
Aim:

Procedure:

S3 Class	**S4 Class:**

Program:	**Output:**

S3 Class
setClass("Person", representation(name = "character", age = "numeric"))
person <- structure(list(name = "John Doe", age = 30), class = "Person")
print.Person <- function(obj)
{
cat("Name:", obj$name, "\n")
cat("Age:", obj$age, "\n")
}
print(person)
	Name: John Doe
	Age: 30

S4 Class setClass("Rectangle", slots = c(length = "numeric", width = "numeric"), prototype = list(length = 0, width = 0)) setMethod("area", "Rectangle", function(object) { object@length * object@width }) rectangle <- new("Rectangle", length = 5, width = 3) area(rectangle)	**Output:** **[1] 15**

Result:

8) Demonstrate the creation of S3 class and S4 class in R. Also illustrate how the fields of the object can be accessed using the $ operator. Modify the Name field by reassigning the name to Paul.
Aim :

Procedure:

Program:	**Output:**
person <- list(name = "John Doe", age = 30) cat("Original Name:", person$name, "\n") cat("Original Age:", person$age, "\n") person$name <- "Paul" cat("Modified Name:", person$name, "\n")	Original Name: John Doe Original Age: 30 Modified Name: Paul

Result :

C. WORKING WITH LOOPING AND FUNCTIONS IN R

1.Write a program to check whether an integer (entered by the user) is a prime number or not using control statements.

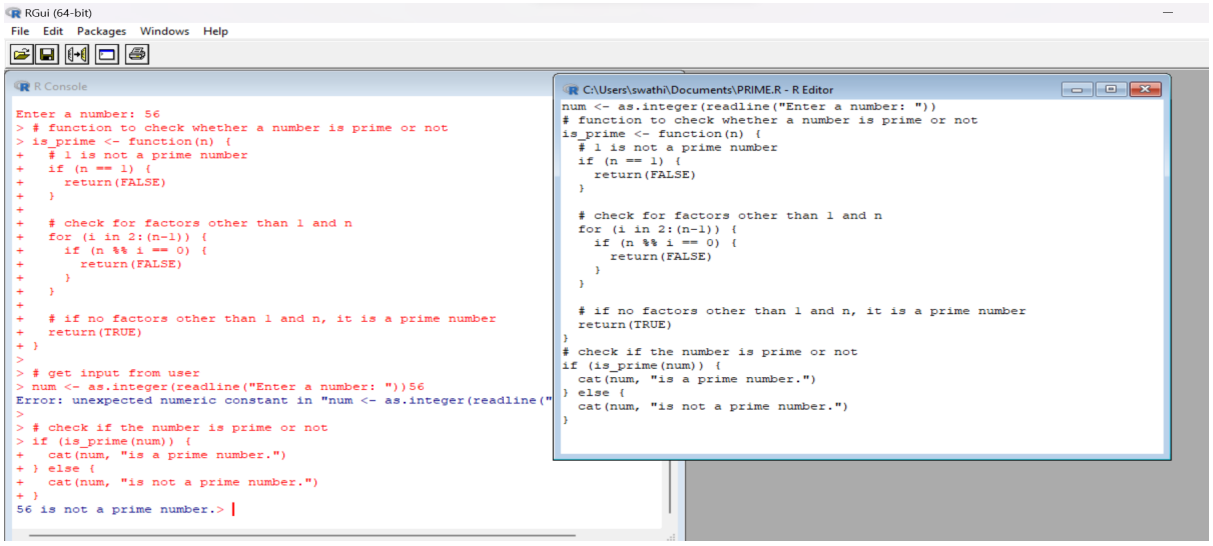
AIM:

PROCEDURE:

CODE:

```
num <- as.integer(readline("Enter a number: "))
# function to check whether a number is prime or not
is_prime <- function(n) {
  # 1 is not a prime number
  if (n == 1) {
    return(FALSE)
  }

  # check for factors other than 1 and n
  for (i in 2:(n-1)) {
    if (n %% i == 0) {
      return(FALSE)
    }
  }
  # if no factors other than 1 and n, it is a prime number
  return(TRUE)
}
# check if the number is prime or not
if (is_prime(num)) {
  cat(num, "is a prime number.")
} else {
  cat(num, "is not a prime number.")
}
```

OUTPUT:



RESULT:

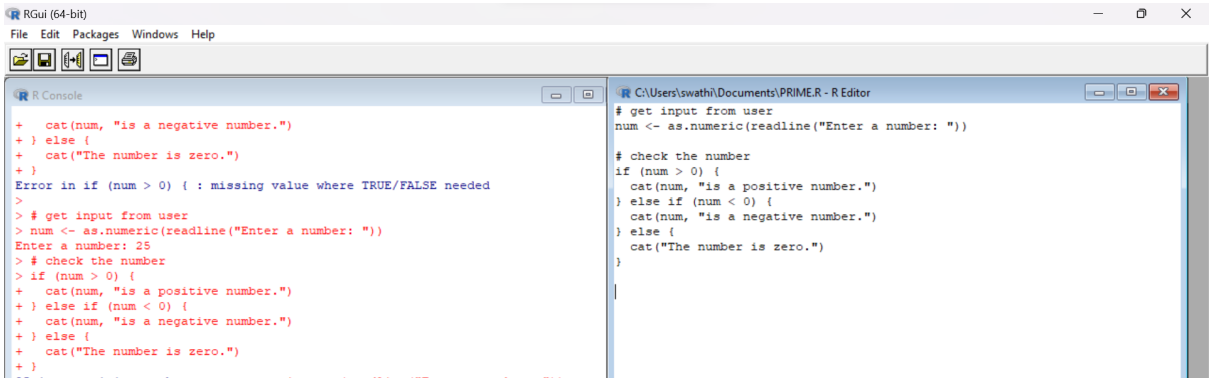
2.Write a program to check whether a number entered by the user is positive number or a negative number or zero.
AIM:

PROCEDURE:

CODE:

```
# get input from user
num <- as.numeric(readline("Enter a number: "))

# check the number
if (num > 0) {
  cat(num, "is a positive number.")
} else if (num < 0) {
  cat(num, "is a negative number.")
} else {
  cat("The number is zero.")
}
```

OUTPUT:



RESULT:

3.Write a program to check whether a number is an Armstrong number or not using a while loop.
AIM:

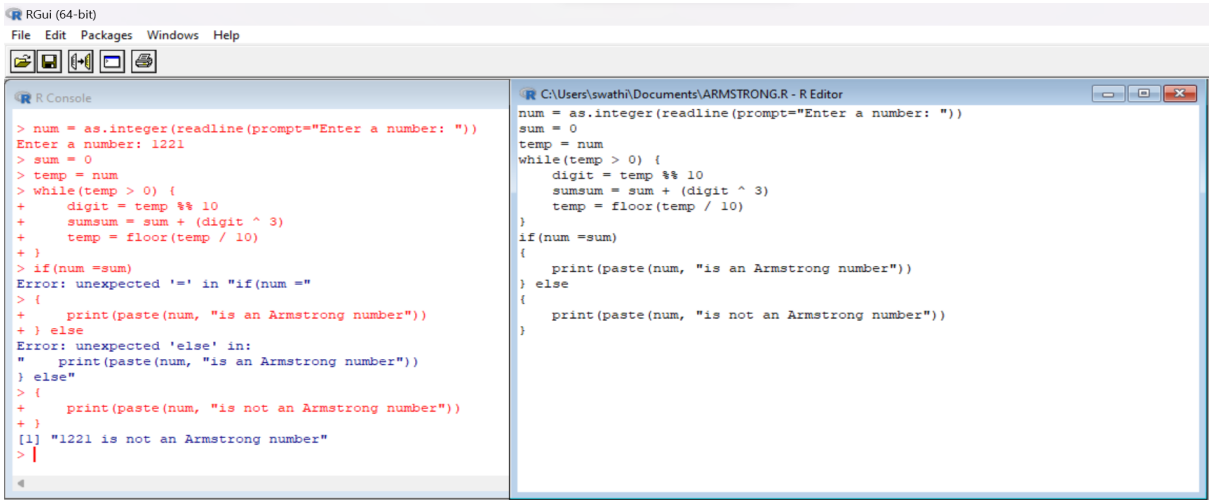
PROCEDURE:

CODE:

```
num = as.integer(readline(prompt="Enter a number: "))
sum = 0
temp = num
while(temp > 0) {
   digit = temp %% 10
   sumsum = sum + (digit ^ 3)
   temp = floor(temp / 10)
}
if(num =sum)
{
```

```
    print(paste(num, "is an Armstrong number"))
} else
{
    print(paste(num, "is not an Armstrong number"))
}
```

OUTPUT:



RESULT:

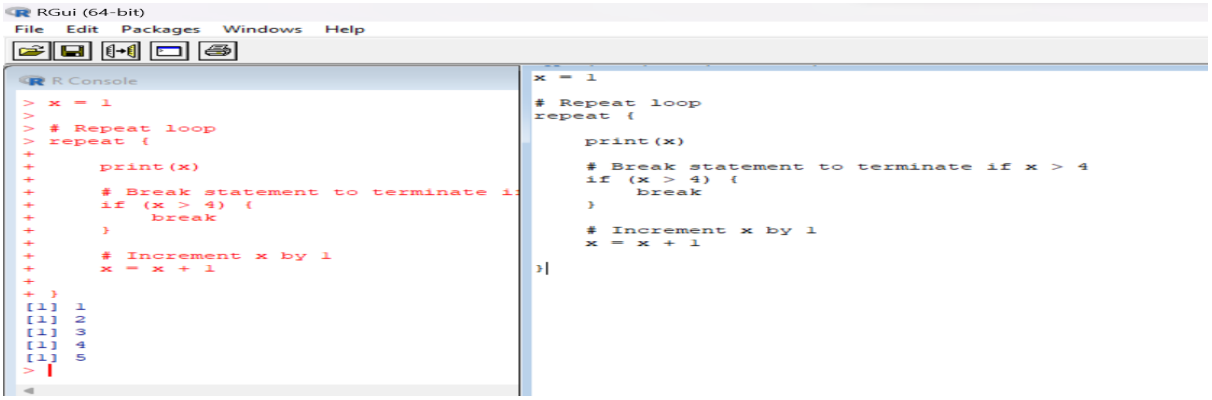
4.Write a program to demonstrate Repeat Loop in R
AIM:

PROCEDURE:

CODE:

```
x = 1
# Repeat loop
repeat {
    print(x)

    # Break statement to terminate if x > 4
    if (x > 4) {
        break
    }

    # Increment x by 1
    x = x + 1
}
```

OUTPUT:



RESULT:

5.Using functions develop a simple calculator in R.
AIM:

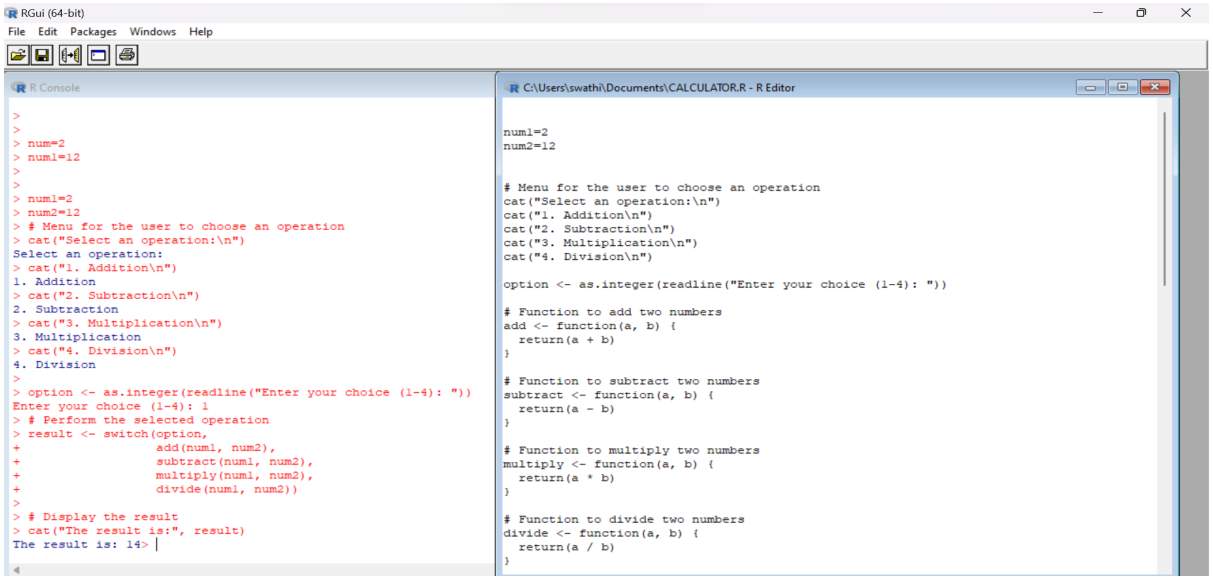
PROCEDURE:

CODE:

```
num1=2
num2=12

# Menu for the user to choose an operation
cat("Select an operation:\n")
cat("1. Addition\n")
cat("2. Subtraction\n")
cat("3. Multiplication\n")
cat("4. Division\n")

option <- as.integer(readline("Enter your choice (1-4): "))

# Function to add two numbers
add <- function(a, b) {
  return(a + b)
}

# Function to subtract two numbers
subtract <- function(a, b) {
  return(a - b)
}

# Function to multiply two numbers
multiply <- function(a, b) {
  return(a * b)
}

# Function to divide two numbers
divide <- function(a, b) {
  return(a / b)
}

# Get input from the user
```

```
# Function to add two numbers
add <- function(a, b) {
  return(a + b)
}

# Function to subtract two numbers
subtract <- function(a, b) {
  return(a - b)
}

# Function to multiply two numbers
multiply <- function(a, b) {
  return(a * b)
}

# Function to divide two numbers
divide <- function(a, b) {
  return(a / b)
}

# Get input from the user
num1=12
num2=2

# Menu for the user to choose an operation
cat("Select an operation:\n")
cat("1. Addition\n")
cat("2. Subtraction\n")
cat("3. Multiplication\n")
cat("4. Division\n")

option <- as.integer(readline("Enter your choice (1-4): "))

# Perform the selected operation
result <- switch(option,
                 add(num1, num2),
                 subtract(num1, num2),
                 multiply(num1, num2),
                 divide(num1, num2))

# Display the result
cat("The result is:", result)

# Perform the selected operation
result <- switch(option,
                 add(num1, num2),
                 subtract(num1, num2),
                 multiply(num1, num2),
                 divide(num1, num2))

# Display the result
cat("The result is:", result)
```

OUTPUT:



RESULT:

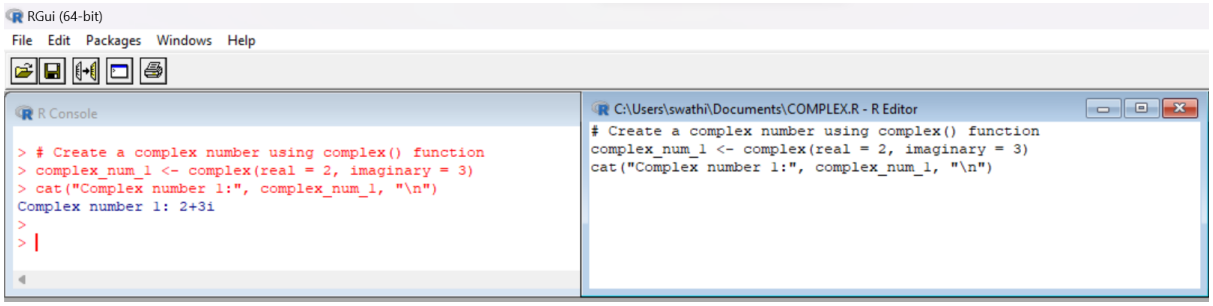
6. Demonstrate the creation of a complex number in R.
AIM:

PROCEDURE:

CODE:

```
# Create a complex number using complex() function
complex_num_1 <- complex(real = 2, imaginary = 3)
cat("Complex number 1:", complex_num_1, "\n")
```

OUTPUT:



RESULT:

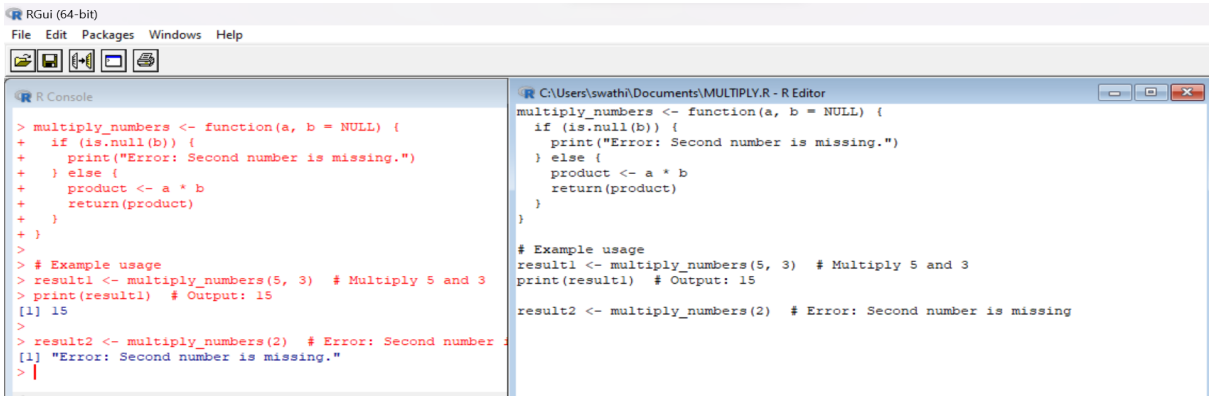
7.Write a program to multiply two numbers using a function with a default value. Assume default value as NULL.
AIM:

PROCEDURE:

CODE:

```
multiply_numbers <- function(a, b = NULL) {
  if (is.null(b)) {
    print("Error: Second number is missing.")
  } else {
    product <- a * b
    return(product)
  }
}
# Example usage
result1 <- multiply_numbers(5, 3)  # Multiply 5 and 3
print(result1)  # Output: 15

result2 <- multiply_numbers(2)  # Error: Second number is missing
```

OUTPUT:



RESULT:

8.Find sum, mean and product of vector elements using built-in functions.

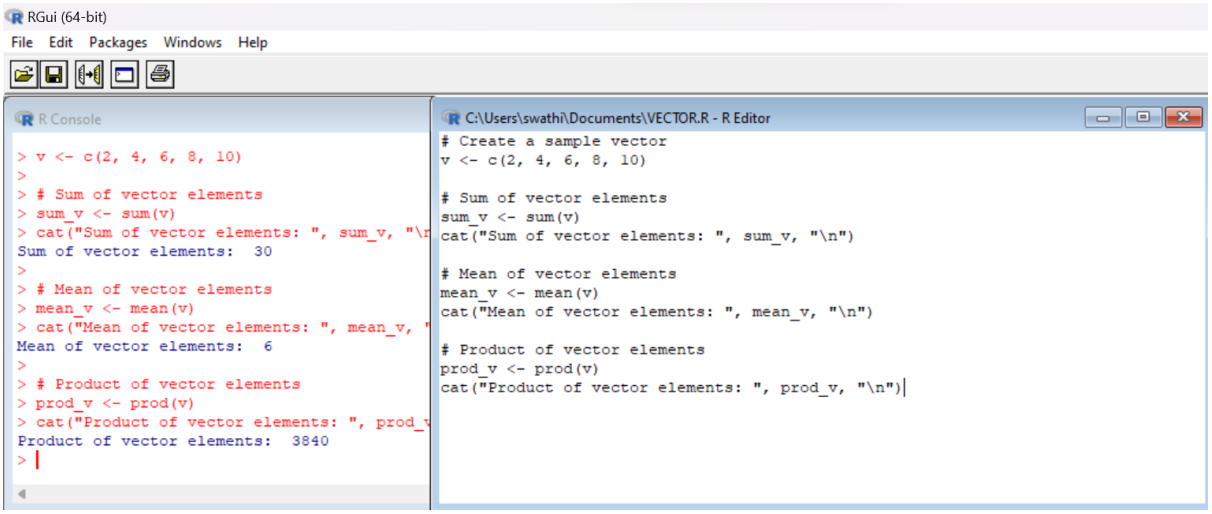
AIM:

PROCEDURE:

CODE:

```
# Create a sample vector
v <- c(2, 4, 6, 8, 10)

# Sum of vector elements
sum_v <- sum(v)
cat("Sum of vector elements: ", sum_v, "\n")

# Mean of vector elements
mean_v <- mean(v)
cat("Mean of vector elements: ", mean_v, "\n")

# Product of vector elements
prod_v <- prod(v)
cat("Product of vector elements: ", prod_v, "\n")
```

OUTPUT:



RESULT:

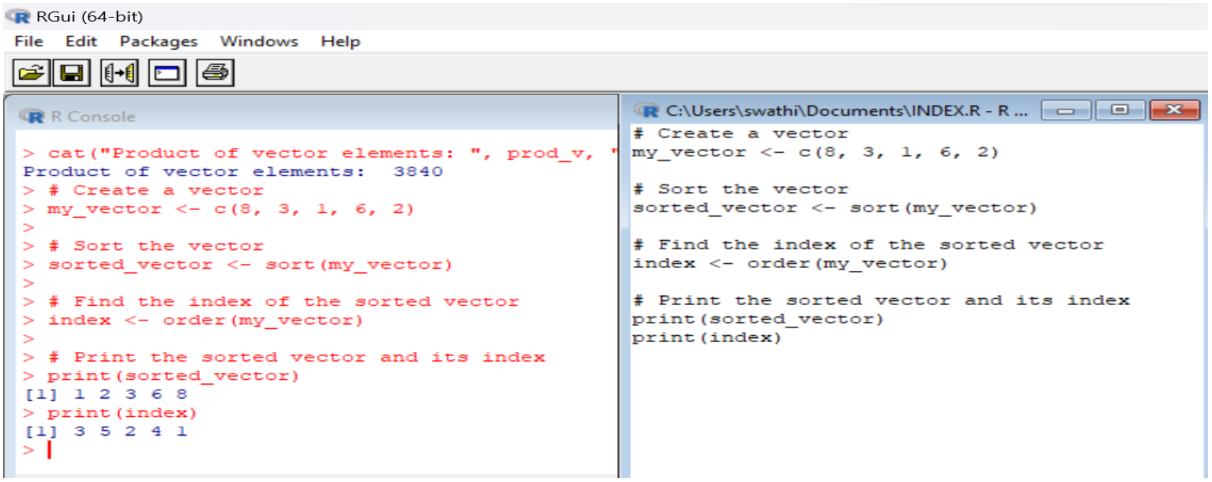
9.Sort a vector in R using sort() function. Also find the index of the sorted vector.
AIM:

PROCEDURE:

CODE:

```
# Create a vector
my_vector <- c(8, 3, 1, 6, 2)

# Sort the vector
sorted_vector <- sort(my_vector)

# Find the index of the sorted vector
index <- order(my_vector)

# Print the sorted vector and its index
print(sorted_vector)
print(index)
```

OUTPUT:



RESULT:

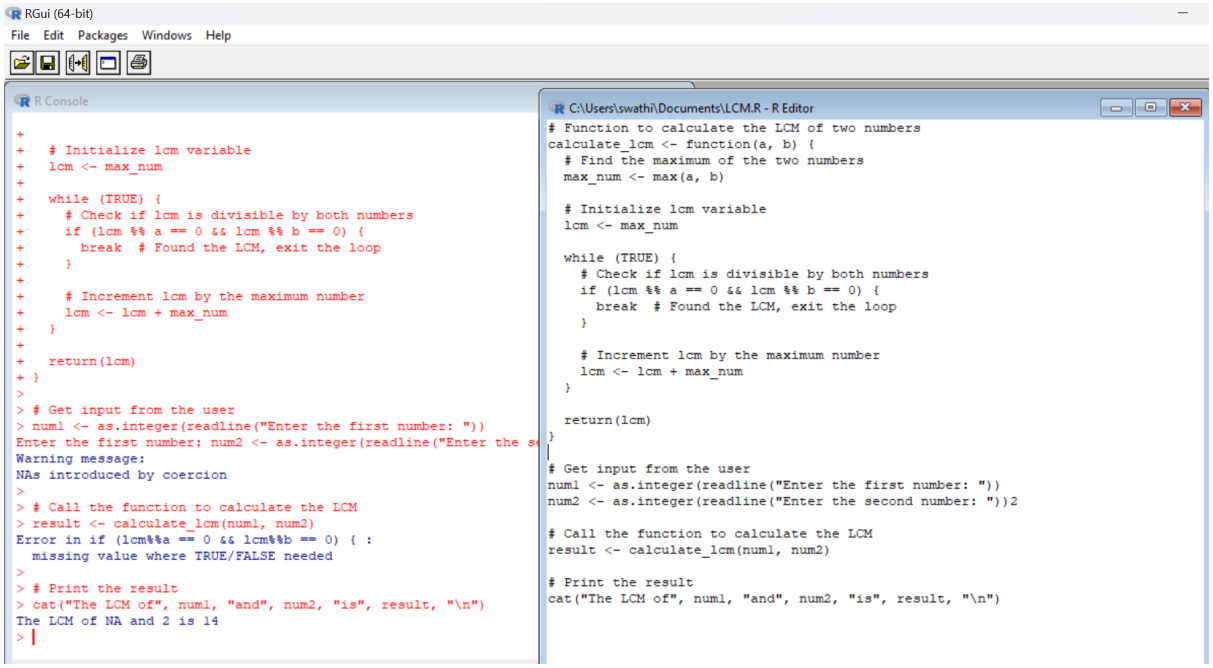
10.Find the L.C.M of two numbers entered by the user by creating a user-defined function.
AIM:

PROCEDURE:

CODE:

```
# Function to calculate the LCM of two numbers
calculate_lcm <- function(a, b) {
  # Find the maximum of the two numbers
  max_num <- max(a, b)

  # Initialize lcm variable
  lcm <- max_num
  while (TRUE) {
    # Check if lcm is divisible by both numbers
    if (lcm %% a == 0 && lcm %% b == 0) {
      break  # Found the LCM, exit the loop
    }

    # Increment lcm by the maximum number
    lcm <- lcm + max_num
  }
  return(lcm)
}

# Get input from the user
num1 <- as.integer(readline("Enter the first number: "))
num2 <- as.integer(readline("Enter the second number: "))2

# Call the function to calculate the LCM
result <- calculate_lcm(num1, num2)

# Print the result
cat("The LCM of", num1, "and", num2, "is", result, "\n")
```

OUTPUT:


RGui (64-bit)
File Edit Packages Windows Help
R Console
+
+ # Initialize lcm variable
+ lcm <- max_num
+
+ while (TRUE) {
+ # Check if lcm is divisible by both numbers
+ if (lcm %% a == 0 && lcm %% b == 0) {
+ break # Found the LCM, exit the loop
+ }
+
+ # Increment lcm by the maximum number
+ lcm <- lcm + max_num
+ }
+
+ return(lcm)
+ }
>
> # Get input from the user
> num1 <- as.integer(readline("Enter the first number: "))
Enter the first number: num2 <- as.integer(readline("Enter the s
Warning message:
NAs introduced by coercion
>
> # Call the function to calculate the LCM
> result <- calculate_lcm(num1, num2)
Error in if (lcm%%a == 0 && lcm%%b == 0) { :
missing value where TRUE/FALSE needed
>
> # Print the result
> cat("The LCM of", num1, "and", num2, "is", result, "\n")
The LCM of NA and 2 is 14
>
C:\Users\swathi\Documents\LCM.R - R Editor
# Function to calculate the LCM of two numbers
calculate_lcm <- function(a, b) {
# Find the maximum of the two numbers
max_num <- max(a, b)
# Initialize lcm variable
lcm <- max_num
while (TRUE) {
# Check if lcm is divisible by both numbers
if (lcm %% a == 0 && lcm %% b == 0) {
break # Found the LCM, exit the loop
}
# Increment lcm by the maximum number
lcm <- lcm + max_num
}
return(lcm)
}
# Get input from the user
num1 <- as.integer(readline("Enter the first number: "))
num2 <- as.integer(readline("Enter the second number: "))2
# Call the function to calculate the LCM
result <- calculate_lcm(num1, num2)
# Print the result
cat("The LCM of", num1, "and", num2, "is", result, "\n")

RESULT:

D. IMPLEMENTATION OF VECTOR RECYCLING, APPLY FAMILY & RECURSION

1. Demonstrate Vector Recycling in R.

AIM:

PROCEDURE:

CODE:	**OUTPUT:**
a<-c(23,45,67,89) b<-c(4,5) print(a+b)	27 50 71 94

RESULT:

2.Demonstrate the usage of apply function in R

AIM:

PROCEDURE:

CODE:	**OUTPUT:**
m<- matrix((1:12), nrow=3) print(m) print(apply(m,1,max)) print(apply(m,2,sum))	[,1] [,2] [,3] [,4] [1,] 1 4 7 10 [2,] 2 5 8 11 [3,] 3 6 9 12 > print(apply(m,1,max)) [1] 10 11 12 > print(apply(m,2,sum)) [1] 6 15 24 33

RESULT:

3.Demonstrate the usage of lapply function in R

AIM:

PROCEDURE:

CODE:	OUTPUT:

names <- c("nikitha", "abhi","pawan",
"hitler","deva")
print("original data:")
names
# apply lapply() function 	
print("data after lapply():")
lapply(names, toupper)
	[1] "nikitha" "abhi" "pawan" "hitler"
	"deva"
	
	[[1]]
	[1] "NIKITHA"
	[[2]]
	[1] "ABHI"
	[[3]]
	[1] "PAWAN"
	[[4]]
	[1] "HITLER"
	[[5]]
	[1] "DEVA"

RESULT:

4. Demonstrate the usage of sapply function in R
AIM:

PROCEDURE:

CODE:	OUTPUT:

sample_data<- data.frame(
x=c(1,2,3,4,5,6),y=c(3,2,4,2,34,5))
print("original data:")
sample_data
# apply sapply() function 	
print("data after sapply():")
sapply(sample_data, max)
	x y
	1 1 3
	2 2 2
	3 3 4
	4 4 2
	5 5 34
	6 6 5
	[1] "data after sapply():"
	> sapply(sample_data, max)
	x y
	6 34

RESULT:

5. Demonstrate the usage of tapply function in R
AIM:

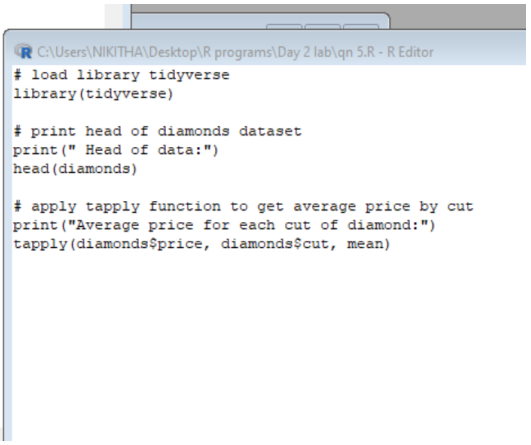
PROCEDURE:

CODE:

```
# load library tidyverse
library(tidyverse)

# print head of diamonds dataset
print(" Head of data:")
head(diamonds)
# apply tapply function to get average price by cut
print("Average price for each cut of diamond:")
tapply(diamonds$price, diamonds$cut, mean)
```

OUTPUT:

```
>
> # print head of diamonds dataset
> print(" Head of data:")
[1] " Head of data:"
> head(diamonds)
# A tibble: 6 × 10
  carat cut       color clarity depth table price     x     y     z
  <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
>
> # apply tapply function to get average price by cut
> print("Average price for each cut of diamond:")
[1] "Average price for each cut of diamond:"
> tapply(diamonds$price, diamonds$cut, mean)
     Fair      Good Very Good   Premium     Ideal
 4358.758  3928.864  3981.760  4584.258  3457.542
>
```



RESULT:

6. Demonstrate the usage of mapply function in R
AIM:

PROCEDURE:

CODE:

```
mapply(rep, 1:3, times=5)
```

OUTPUT:

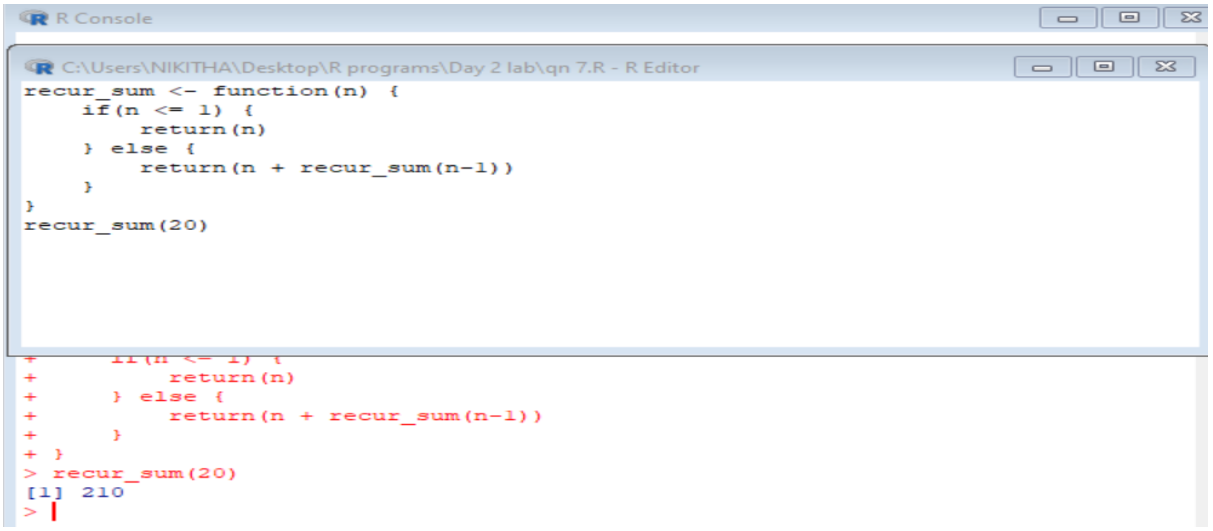
```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
[3,]    1    2    3
[4,]    1    2    3
[5,]    1    2    3
```

RESULT:

7. Sum of Natural Numbers using Recursion
AIM:

PROCEDURE:

CODE & OUTPUT:



RESULT:

8. Write a program to generate Fibonacci sequence using Recursion in R
AIM:

PROCEDURE:

CODE:

```
fibonacci <- function(n) {
  if(n <= 1) {
    return(n)
  } else {
    return(fibonacci(n-1) + fibonacci(n-2))
  }
}

for(i in 0:10) {
  result <- fibonacci(i)
  cat(result, " ")
}
```

OUTPUT:

0 1 1 2 3 5 8 13 21 34 55

RESULT:

9. Write a program to find factorial of a number in R using recursion.
AIM:

PROCEDURE:

CODE:

```
factorial <- function(n) {
  if(n <= 1) {
    return(1)
  } else {
    return(n * factorial(n-1))
  }
}
result <- factorial(9)
cat("9! =", result)
```

OUTPUT:

9!=362880

RESULT:

E. CREATION AND MANIPULATION OF DATAFRAMES IN R

1. Consider two vectors: x=seq(1,43,along.with=Id) y=seq(-20,0,along.with=Id)
Create a data frame 'df' as shown below.

```
>df
Id Letter x y
1 1 a 1.000000 -20.000000
2 1 b 4.818182 -18.181818
3 1 c 8.636364 -16.363636
4 2 a 12.454545 -14.545455
5 2 b 16.272727 -12.727273
6 2 c 20.090909 -10.909091
7 3 a 23.909091 -9.090909
8 3 b 27.727273 -7.272727
9 3 c 31.545455 -5.454545
10 4 a 35.363636 -3.636364
11 4 b 39.181818 -1.818182
12 4 c 43.000000 0.000000
```

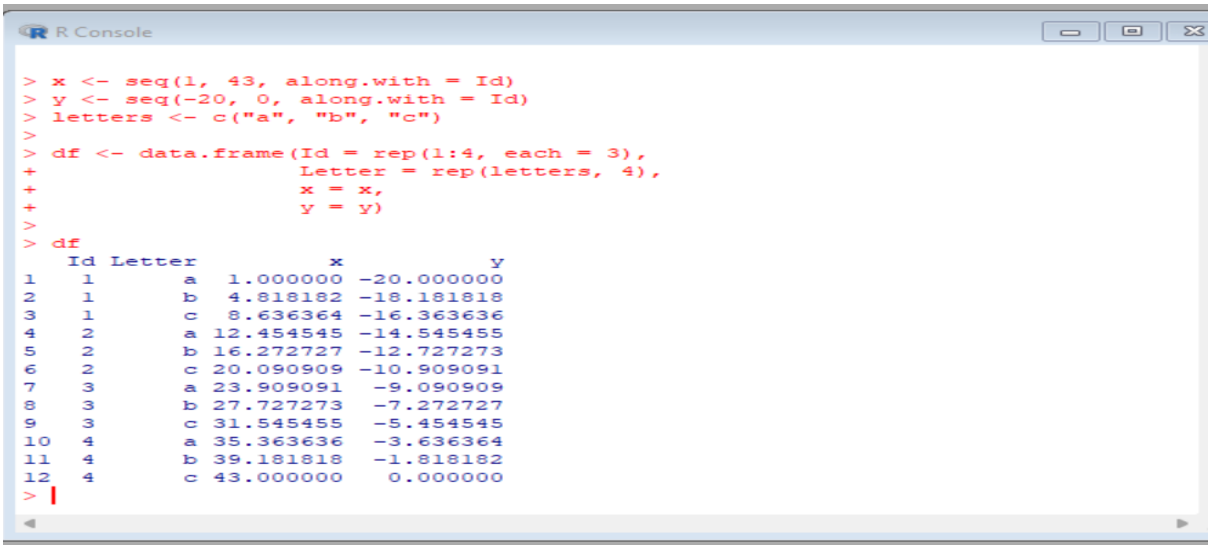
AIM:

PROCEDURE:

CODE:

```
x <- seq(1, 43, along.with = Id)
y <- seq(-20, 0, along.with = Id)
letters <- c("a", "b", "c")

df <- data.frame(Id = rep(1:4, each = 3),
                 Letter = rep(letters, 4),
                 x = x,
                 y = y)

df
```

OUTPUT:



RESULT:

2. Create two data frame df1 and df2:

> df1

Id Age

1 1 14

2 2 12

3 3 15

4 4 10

> df2

Id Sex Code

1 1 F a

2 2 M b

3 3 M c

4 4 F d

From df1 and df2 create M:

>M

Id Age Sex Code

1 1 14 F a

2 2 12 M b

3 3 15 M c 4 4 10 F d

AIM:

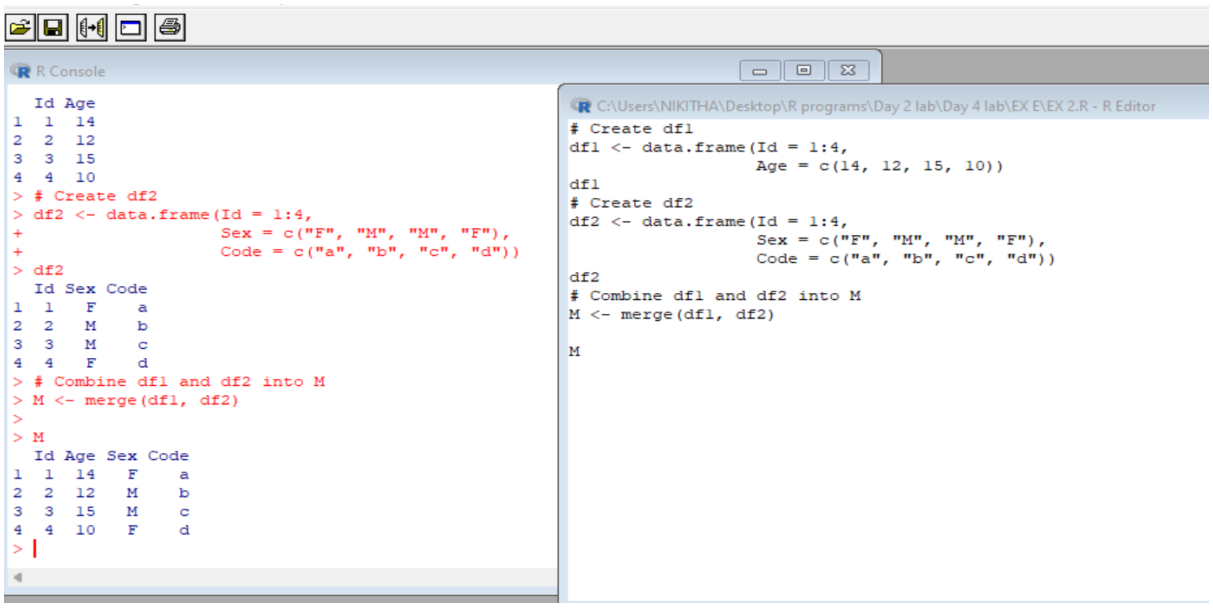
PROCEDURE:

CODE:

```
# Create df1
df1 <- data.frame(Id = 1:4,
```

```
Age = c(14, 12, 15, 10))
df1
# Create df2
df2 <- data.frame(Id = 1:4,
Sex = c("F", "M", "M", "F"),
Code = c("a", "b", "c", "d"))
df2
# Combine df1 and df2 into M
M <- merge(df1, df2)
M
```

OUTPUT:



RESULT:

3. Create a data frame df3:

> df3 id2 score 1 4 100
2 3 98
3 2 94
4 1 99

From M (used in Exercise-3) and df3 create N:

Id Age Sex Code score
1 1 14 F a 99
2 2 12 M b 94
3 3 15 M c 98 4 4 10 F d 100

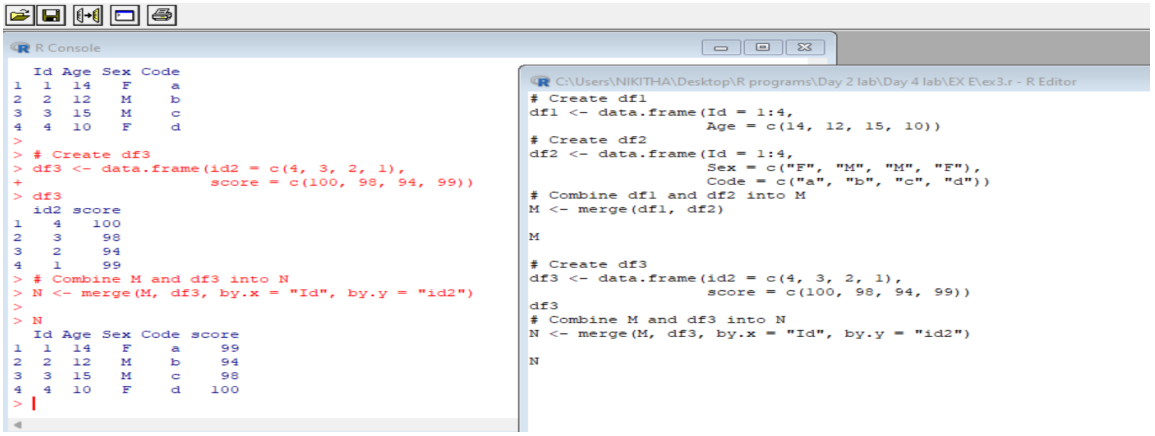
AIM:

PROCEDURE:

CODE:

```
# Create df1
df1 <- data.frame(Id = 1:4,
                  Age = c(14, 12, 15, 10))
# Create df2
df2 <- data.frame(Id = 1:4,
                  Sex = c("F", "M", "M", "F"),
                  Code = c("a", "b", "c", "d"))
# Combine df1 and df2 into M
M <- merge(df1, df2)
M
# Create df3
df3 <- data.frame(id2 = c(4, 3, 2, 1),
                  score = c(100, 98, 94, 99))
df3
# Combine M and df3 into N
N <- merge(M, df3, by.x = "Id", by.y = "id2")
N
```

OUTPUT:



RESULT:

4. Consider the previous one data frame N:

1) Remove the variables Sex and Code

2) From N, create a data frame:

values ind

1 1 Id
2 2 Id
3 3 Id
4 4 Id
5 14 Age
6 12 Age
7 15 Age
8 10 Age
9 99 score
10 94 score
11 98 score
12 100 score

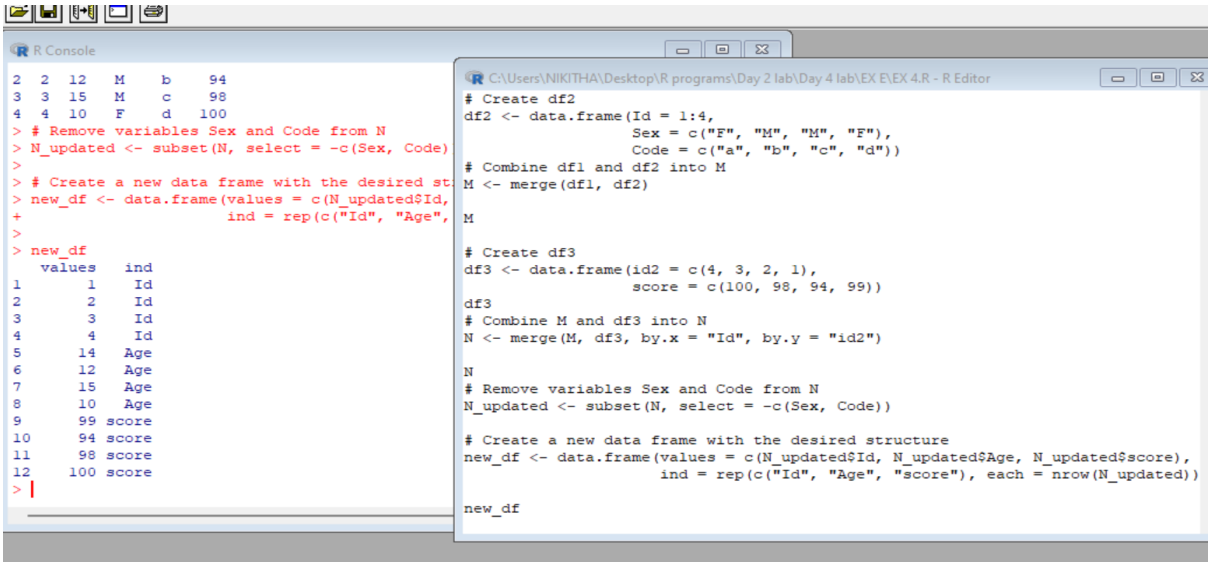
AIM:

PROCEDURE:

CODE:

```
# Create df1
df1 <- data.frame(Id = 1:4,
                  Age = c(14, 12, 15, 10))
# Create df2
df2 <- data.frame(Id = 1:4,
                  Sex = c("F", "M", "M", "F"),
                  Code = c("a", "b", "c", "d"))
# Combine df1 and df2 into M
M <- merge(df1, df2)
M
# Create df3
df3 <- data.frame(id2 = c(4, 3, 2, 1),
                  score = c(100, 98, 94, 99))
df3
# Combine M and df3 into N
N <- merge(M, df3, by.x = "Id", by.y = "id2")
N
# Remove variables Sex and Code from N
N_updated <- subset(N, select = -c(Sex, Code))

# Create a new data frame with the desired structure
new_df <- data.frame(values = c(N_updated$Id, N_updated$Age, N_updated$score),
                     ind = rep(c("Id", "Age", "score"), each = nrow(N_updated)))

new_df
```

OUTPUT:



RESULT:

6. For this exercise, we'll use the (built-in) dataset trees.

a) **Make sure the object is a data frame, if not change it to a data frame.**

b) **Create a new data frame A:**

>A

Girth Height Volume

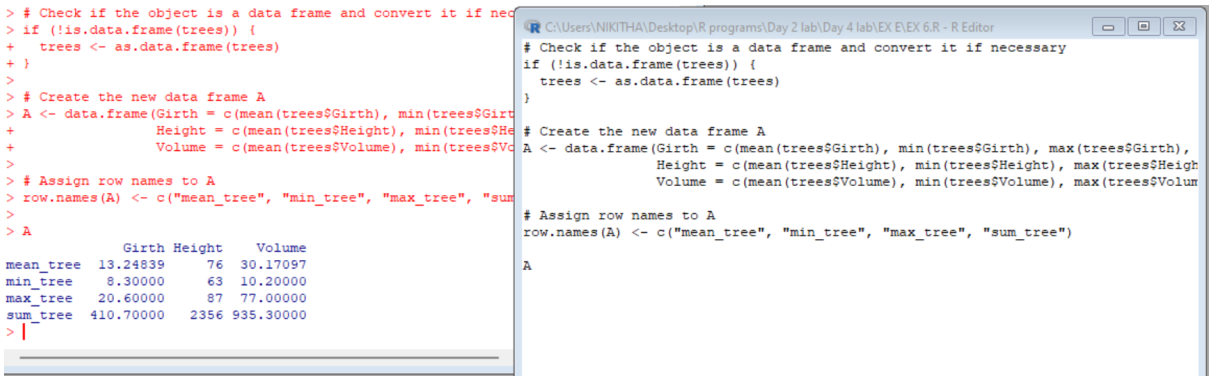
mean_tree 13.24839 76 30.17097
min_tree 8.30000 63 10.20000
max_tree 20.60000 87 77.00000
sum_tree 410.70000 2356 935.30000
AIM:

PROCEDURE:

CODE:

```
# Check if the object is a data frame and convert it if necessary
if (!is.data.frame(trees)) {
  trees <- as.data.frame(trees)
}

# Create the new data frame A
A <- data.frame(Girth = c(mean(trees$Girth), min(trees$Girth), max(trees$Girth), sum(trees$Girth)),
                Height = c(mean(trees$Height), min(trees$Height), max(trees$Height), sum(trees$Height)),
                Volume = c(mean(trees$Volume), min(trees$Volume), max(trees$Volume), sum(trees$Volume)))

# Assign row names to A
row.names(A) <- c("mean_tree", "min_tree", "max_tree", "sum_tree")

A
```

OUTPUT:



RESULT:

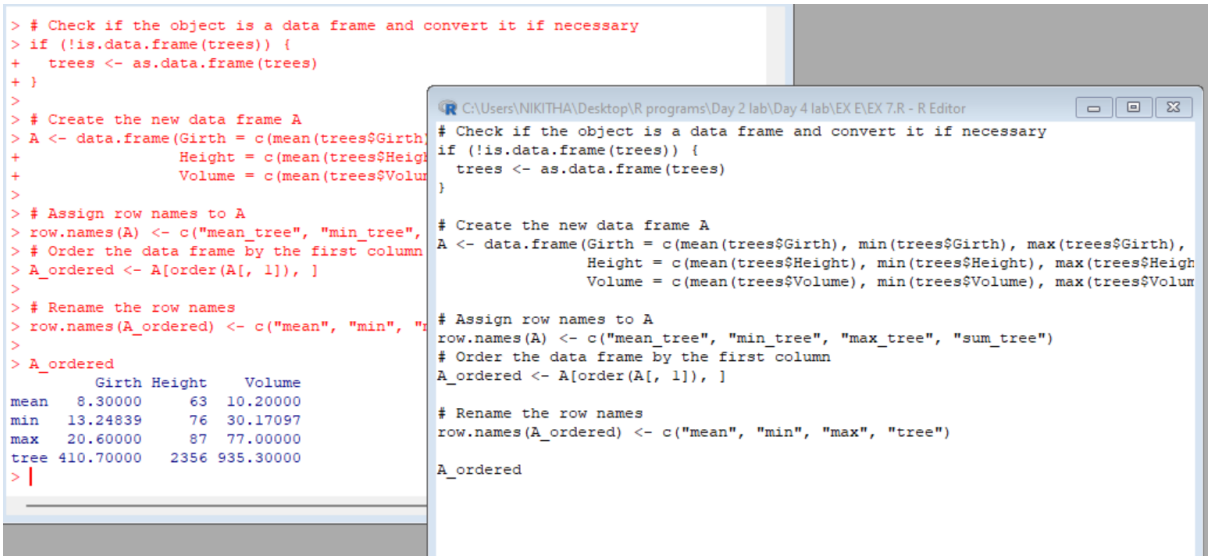
7. Consider the data frame A:
a)Order the entire data frame by the first column.
b)Rename the row names as follows: mean, min, max, tree
AIM:

PROCEDURE:

CODE:

```
# Check if the object is a data frame and convert it if necessary
if (!is.data.frame(trees)) {
  trees <- as.data.frame(trees)
}
# Create the new data frame A
A <- data.frame(Girth = c(mean(trees$Girth), min(trees$Girth), max(trees$Girth), sum(trees$Girth)),
                Height = c(mean(trees$Height), min(trees$Height), max(trees$Height), sum(trees$Height)),
                Volume = c(mean(trees$Volume), min(trees$Volume), max(trees$Volume), sum(trees$Volume)))
# Assign row names to A
row.names(A) <- c("mean_tree", "min_tree", "max_tree", "sum_tree")
# Order the data frame by the first column
A_ordered <- A[order(A[, 1]), ]
# Rename the row names
row.names(A_ordered) <- c("mean", "min", "max", "tree")
A_ordered
```

OUTPUT:



RESULT:

8. Create a data frame XY

X=c(1,2,3,1,4,5,2)
Y=c(0,3,2,0,5,9,3)
> XY
X Y
1 0
2 3
3 2
1 0
4 5
5 9
2 3

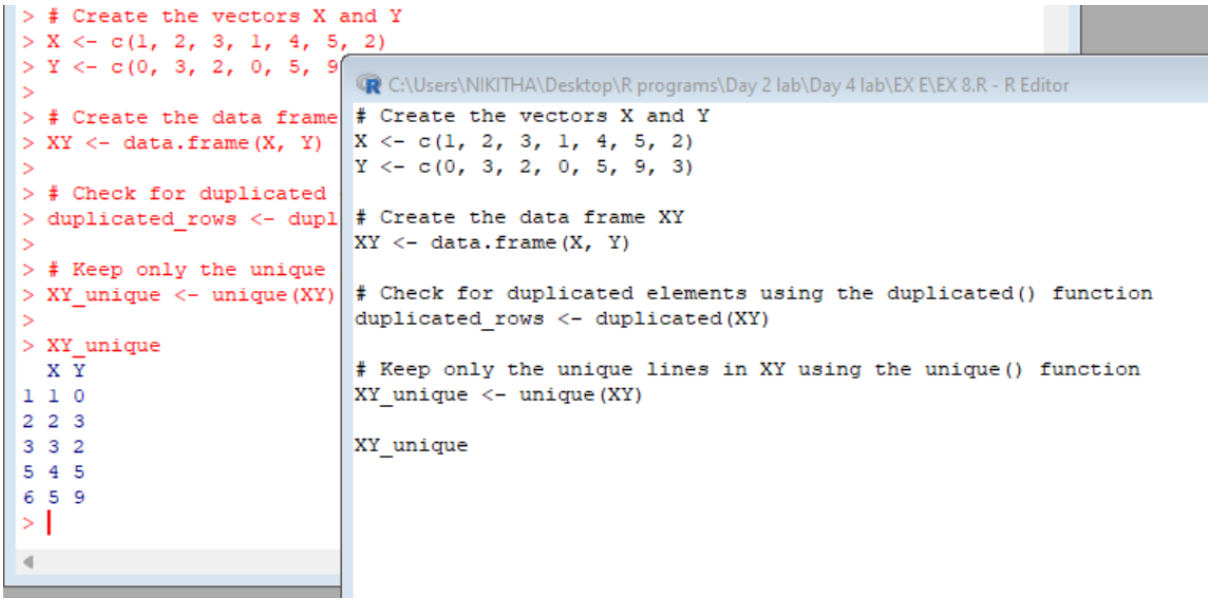
look at duplicated elements using a provided R function. keep only the unique lines on XY using a provided R function.

AIM:

PROCEDURE:

CODE:

```
# Create the vectors X and Y
X <- c(1, 2, 3, 1, 4, 5, 2)
Y <- c(0, 3, 2, 0, 5, 9, 3)
# Create the data frame XY
XY <- data.frame(X, Y)
# Check for duplicated elements using the duplicated() function
duplicated_rows <- duplicated(XY)
# Keep only the unique lines in XY using the unique() function
XY_unique <- unique(XY)
XY_unique
```



9. Use the (built-in) dataset Titanic.

Make sure the object is a data frame, if not change it to a data frame.

Define a data frame with value 1st in Class variable, and value NO in Survived variable and variables Sex, Age and Freq.

Sex Age Freq

1 Male Child 0

5 Female Child 0

9 Male Adult 118

13 Female Adult 4

AIM:

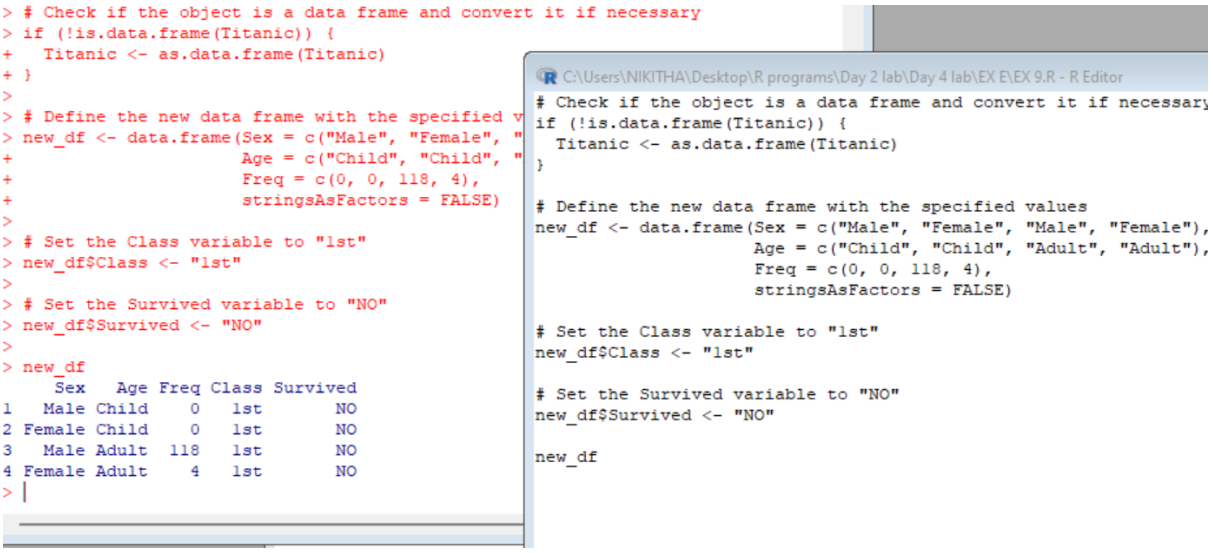
PROCEDURE:

CODE:

```
# Check if the object is a data frame and convert it if necessary
if (!is.data.frame(Titanic)) {
  Titanic <- as.data.frame(Titanic)
}
```

```
# Define the new data frame with the specified values
new_df <- data.frame(Sex = c("Male", "Female", "Male", "Female"),
                     Age = c("Child", "Child", "Adult", "Adult"),
                     Freq = c(0, 0, 118, 4),
                     stringsAsFactors = FALSE)

# Set the Class variable to "1st"
new_df$Class <- "1st"

# Set the Survived variable to "NO"
new_df$Survived <- "NO"

new_df
```

OUTPUT:



```
> # Check if the object is a data frame and convert it if necessary
> if (!is.data.frame(Titanic)) {
+   Titanic <- as.data.frame(Titanic)
+ }
>
> # Define the new data frame with the specified v
> new_df <- data.frame(Sex = c("Male", "Female", "
+                      Age = c("Child", "Child", "
+                      Freq = c(0, 0, 118, 4),
+                      stringsAsFactors = FALSE)
>
> # Set the Class variable to "1st"
> new_df$Class <- "1st"
>
> # Set the Survived variable to "NO"
> new_df$Survived <- "NO"
>
> new_df
     Sex   Age Freq Class Survived
1   Male Child    0   1st       NO
2 Female Child    0   1st       NO
3   Male Adult  118   1st       NO
4 Female Adult    4   1st       NO
>
```

C:\Users\NIKITHA\Desktop\R programs\Day 2 lab\Day 4 lab\EX E\EX 9.R - R Editor

```
# Check if the object is a data frame and convert it if necessary
if (!is.data.frame(Titanic)) {
  Titanic <- as.data.frame(Titanic)
}

# Define the new data frame with the specified values
new_df <- data.frame(Sex = c("Male", "Female", "Male", "Female"),
                     Age = c("Child", "Child", "Adult", "Adult"),
                     Freq = c(0, 0, 118, 4),
                     stringsAsFactors = FALSE)

# Set the Class variable to "1st"
new_df$Class <- "1st"

# Set the Survived variable to "NO"
new_df$Survived <- "NO"

new_df
```

RESULT:

MERGING DATAFRAMES

10 a) Create the following dataframes to merge:

buildings<- data.frame(location=c(1, 2, 3), name=c("building1", "building2","building3"))

data <- data.frame(survey=c(1,1,1,2,2,2),location=c(1,2,3,2,3,1),efficiency=c(51,64,70,7,80,58))

The dataframes, buildingsand datahave a common key variable called, “location”.

Use the merge() function to merge the two dataframes by “location”, into a new dataframe,“buildingStats”.

AIM:

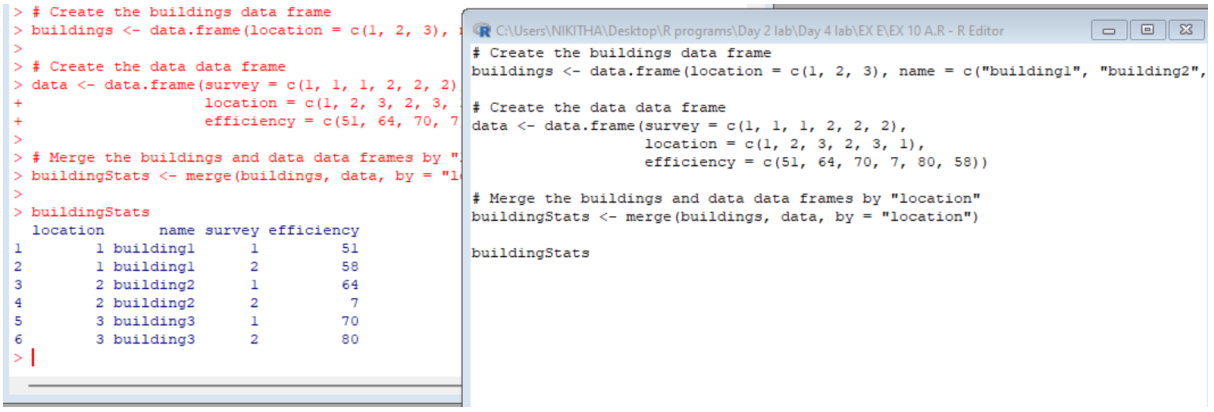
PROCEDURE:

CODE:

```
# Create the buildings data frame
buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2", "building3"))
# Create the data data frame
data <- data.frame(survey = c(1, 1, 1, 2, 2, 2),
                   location = c(1, 2, 3, 2, 3, 1),
```

efficiency = c(51, 64, 70, 7, 80, 58))
# Merge the buildings and data data frames by "location"
buildingStats <- merge(buildings, data, by = "location")
buildingStats

OUTPUT:



RESULT:

b) Give the dataframes different key variable names:
buildings<- data.frame(location=c(1, 2, 3), name=c("building1","building2", "building3"))
data <- data.frame(survey=c(1,1,1,2,2,2), LocationID=c(1,2,3,2,3,1), efficiency=c(51,64,70,71,80,58))
The dataframes, buildings and data have corresponding variables called, location, and LocationID. Use the merge() function to merge the columns of the two dataframes by the corresponding variables.

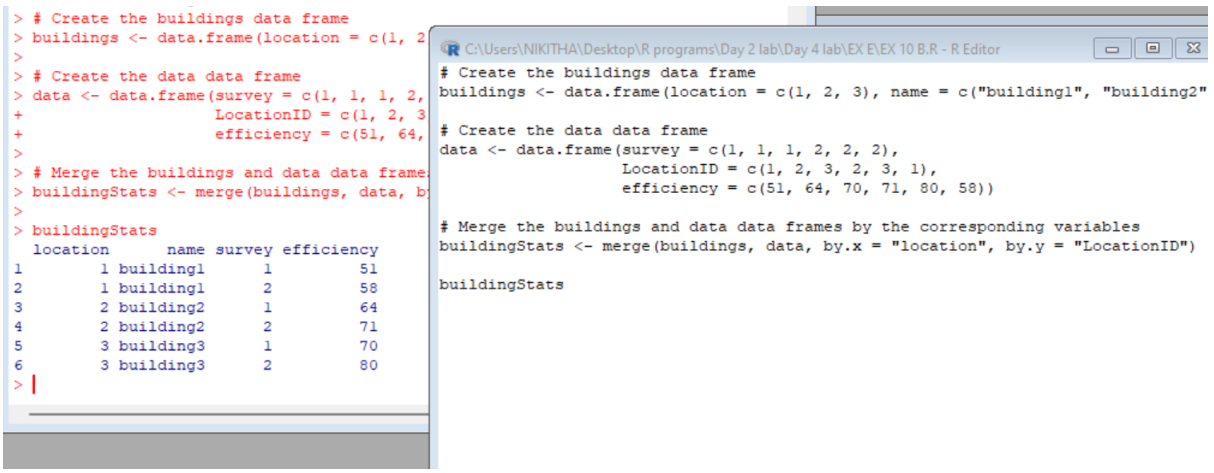
AIM:

PROCEDURE:

CODE:

```
# Create the buildings data frame
buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2", "building3"))
# Create the data data frame
data <- data.frame(survey = c(1, 1, 1, 2, 2, 2),
                   LocationID = c(1, 2, 3, 2, 3, 1),
                   efficiency = c(51, 64, 70, 71, 80, 58))
# Merge the buildings and data data frames by the corresponding variables
buildingStats <- merge(buildings, data, by.x = "location", by.y = "LocationID")
buildingStats
```

OUTPUT:



RESULT:

DIFFERENT TYPES OF MERGE IN R

11 **a)InnerJoin:**

The R merge() function automatically joins the frames by common variable names. In that case, demonstrate how you would perform the merge in Exercise 10 a without specifying the key variable.

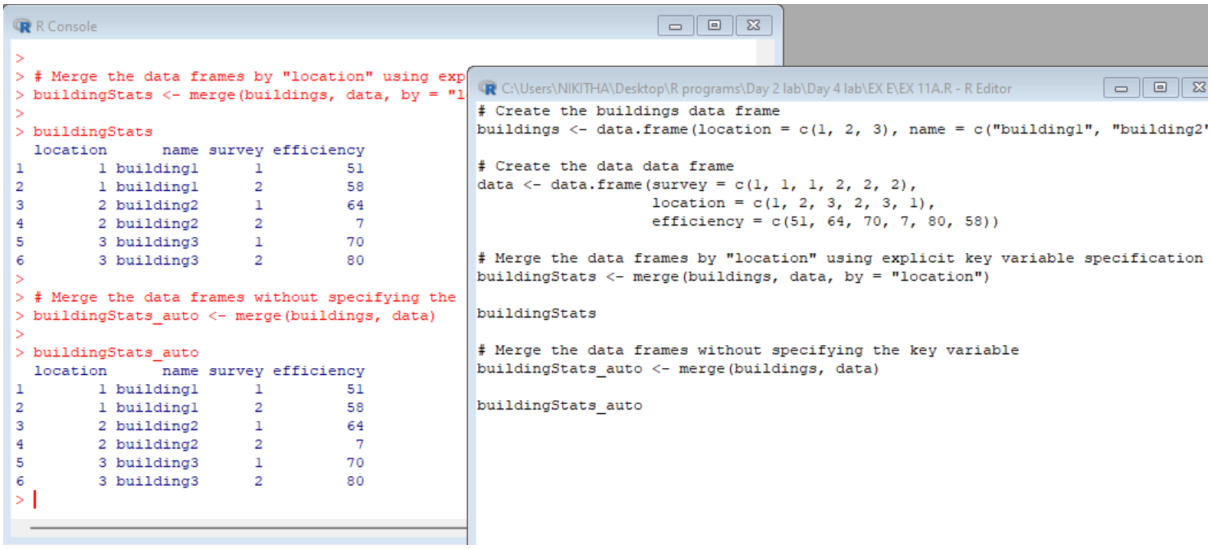
AIM:

PROCEDURE:

CODE:

```
# Create the buildings data frame
buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2", "building3"))
# Create the data data frame
data <- data.frame(survey = c(1, 1, 1, 2, 2, 2),
                   location = c(1, 2, 3, 2, 3, 1),
                   efficiency = c(51, 64, 70, 7, 80, 58))
# Merge the data frames by "location" using explicit key variable specification
buildingStats <- merge(buildings, data, by = "location")
buildingStats
# Merge the data frames without specifying the key variable
buildingStats_auto <- merge(buildings, data)
buildingStats_auto
```

OUTPUT:



RESULT:

b)OuterJoin:

Merge the two dataframes from Exercise 10 a. Use the "all=" parameter in the merge() function to return all records from both tables. Also, merge with the key variable, "location".

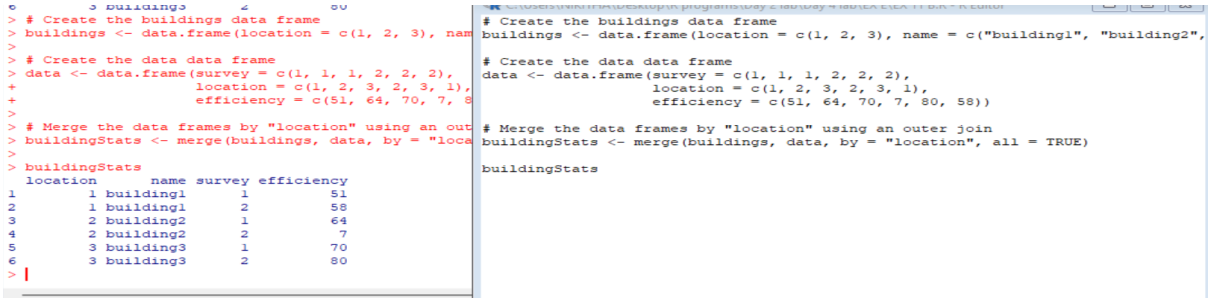
AIM:

PROCEDURE:

CODE:

```
# Create the buildings data frame
buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2", "building3"))
# Create the data data frame
data <- data.frame(survey = c(1, 1, 1, 2, 2, 2),
                   location = c(1, 2, 3, 2, 3, 1),
                   efficiency = c(51, 64, 70, 7, 80, 58))
# Merge the data frames by "location" using an outer join
buildingStats <- merge(buildings, data, by = "location", all = TRUE)
buildingStats
```

OUTPUT:



c)Left Join:

Merge the two dataframes from Exercise 10 a, and return all rows from the left table. Specify the matching key from Exercise 10 a.

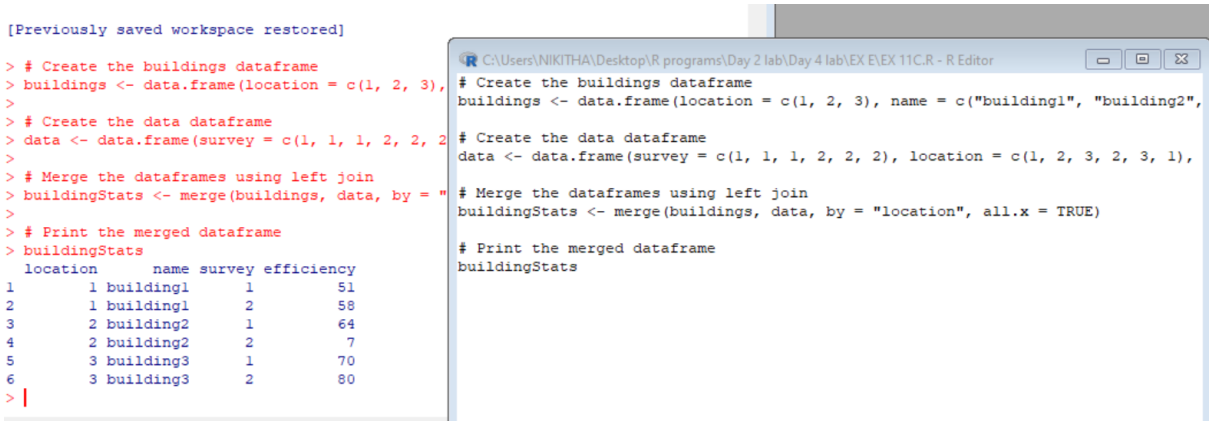
AIM:

PROCEDURE:

CODE:

```
# Create the buildings dataframe
buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2", "building3"))
# Create the data dataframe
data <- data.frame(survey = c(1, 1, 1, 2, 2, 2), location = c(1, 2, 3, 2, 3, 1), efficiency = c(51, 64, 70, 7, 80, 58))
# Merge the dataframes using left join
buildingStats <- merge(buildings, data, by = "location", all.x = TRUE)
# Print the merged dataframe
BuildingStats
```

OUTPUT:



```
[Previously saved workspace restored]

> # Create the buildings dataframe
> buildings <- data.frame(location = c(1, 2, 3),
>
> # Create the data dataframe
> data <- data.frame(survey = c(1, 1, 1, 2, 2, 2
>
> # Merge the dataframes using left join
> buildingStats <- merge(buildings, data, by = "
>
> # Print the merged dataframe
> buildingStats
  location      name survey efficiency
1        1 building1      1         51
2        1 building1      2         58
3        2 building2      1         64
4        2 building2      2          7
5        3 building3      1         70
6        3 building3      2         80
>
```

C:\Users\NIKITHA\Desktop\R programs\Day 2 lab\Day 4 lab\EX E\EX 11C.R - R Editor

```
# Create the buildings dataframe
buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2",

# Create the data dataframe
data <- data.frame(survey = c(1, 1, 1, 2, 2, 2), location = c(1, 2, 3, 2, 3, 1),

# Merge the dataframes using left join
buildingStats <- merge(buildings, data, by = "location", all.x = TRUE)

# Print the merged dataframe
buildingStats
```

RESULT:

d)Right Join:

Merge the two dataframes from Exercise 10a, and return all rows from the right table. Use the matching key from Exercise 10a to return matching rows from the left table.

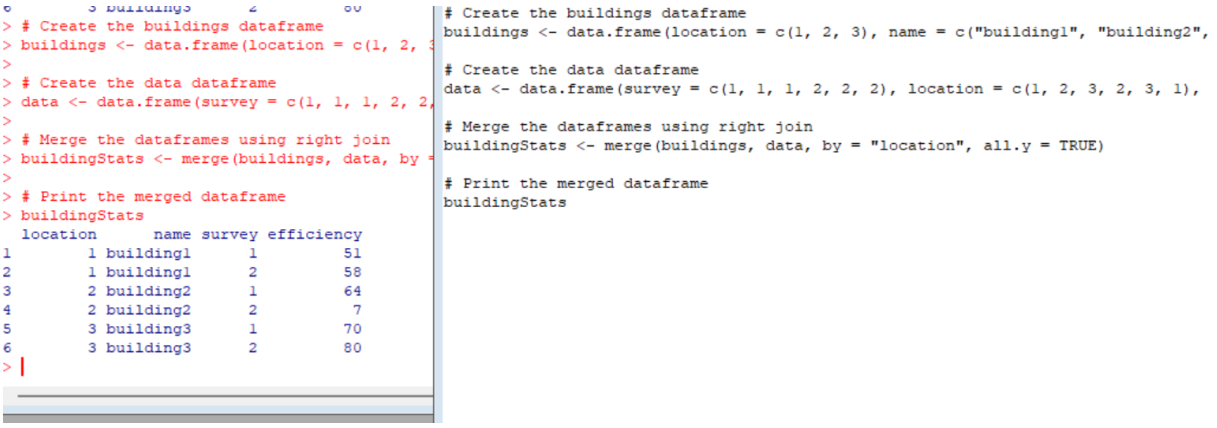
AIM:

PROCEDURE:

CODE:

```
# Create the buildings dataframe
buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2", "building3"))
# Create the data dataframe
data <- data.frame(survey = c(1, 1, 1, 2, 2, 2), location = c(1, 2, 3, 2, 3, 1), efficiency = c(51, 64, 70, 7, 80, 58))
# Merge the dataframes using right join
buildingStats <- merge(buildings, data, by = "location", all.y = TRUE)
# Print the merged dataframe
BuildingStats
```

OUTPUT:



```
> # Create the buildings dataframe
> buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2",
>
> # Create the data dataframe
> data <- data.frame(survey = c(1, 1, 1, 2, 2, 2), location = c(1, 2, 3, 2, 3, 1),
>
> # Merge the dataframes using right join
> buildingStats <- merge(buildings, data, by = "location", all.y = TRUE)
>
> # Print the merged dataframe
> buildingStats
  location      name survey efficiency
1        1 building1      1         51
2        1 building1      2         58
3        2 building2      1         64
4        2 building2      2          7
5        3 building3      1         70
6        3 building3      2         80
>
```

e)Cross Join:

Merge the two dataframes from Exercise 10a, into a "Cross Join" with each row of "buildings" matched to each row of "data". What new column names are created in "buildingStats"?

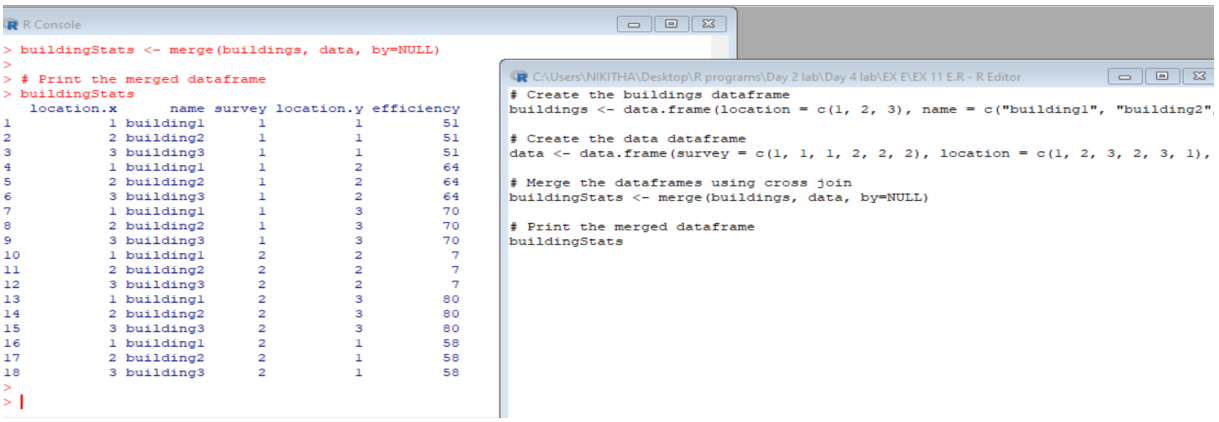
AIM:

PROCEDURE:

CODE:

```
# Create the buildings dataframe
buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2", "building3"))
# Create the data dataframe
data <- data.frame(survey = c(1, 1, 1, 2, 2, 2), location = c(1, 2, 3, 2, 3, 1), efficiency = c(51, 64, 70, 7, 80, 58))
# Merge the dataframes using cross join
buildingStats <- merge(buildings, data, by=NULL)
# Print the merged dataframe
BuildingStats
```

OUTPUT:



```
R Console
> buildingStats <- merge(buildings, data, by=NULL)
>
> # Print the merged dataframe
> buildingStats
   location.x      name survey location.y efficiency
1           1 building1      1          1         51
2           2 building2      1          1         51
3           3 building3      1          1         51
4           1 building1      1          2         64
5           2 building2      1          2         64
6           3 building3      1          2         64
7           1 building1      1          3         70
8           2 building2      1          3         70
9           3 building3      1          3         70
10          1 building1      2          2          7
11          2 building2      2          2          7
12          3 building3      2          2          7
13          1 building1      2          3         80
14          2 building2      2          3         80
15          3 building3      2          3         80
16          1 building1      2          1         58
17          2 building2      2          1         58
18          3 building3      2          1         58
>
>
```

```
C:\Users\NIKITHA\Desktop\R programs\Day 2 lab\Day 4 lab\EX E\EX 11 E.R - R Editor
# Create the buildings dataframe
buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2",

# Create the data dataframe
data <- data.frame(survey = c(1, 1, 1, 2, 2, 2), location = c(1, 2, 3, 2, 3, 1),

# Merge the dataframes using cross join
buildingStats <- merge(buildings, data, by=NULL)

# Print the merged dataframe
buildingStats
```

RESULT:

12 MergingDataframe rows:

To join two data frames (datasets) vertically, use the rbind function. The two data frames must have the same variables, but they do not have to be in the same order.

Merge the rows of the following two dataframes:

buildings<- data.frame(location=c(1, 2, 3), name=c("building1",
"building2", "building3"))
buildings2 <- data.frame(location=c(5, 4, 6), name=c("building5", "building4", "building6"))
Also, specify the new dataframe as, “allBuidings”.

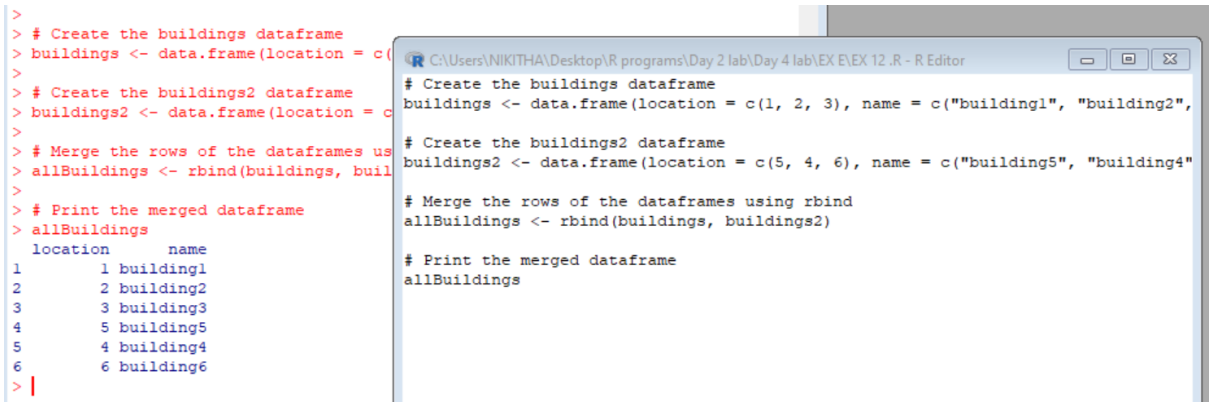
AIM:

PROCEDURE:

CODE:

```
# Create the buildings dataframe
buildings <- data.frame(location = c(1, 2, 3), name = c("building1", "building2", "building3"))
# Create the buildings2 dataframe
buildings2 <- data.frame(location = c(5, 4, 6), name = c("building5", "building4", "building6"))
# Merge the rows of the dataframes using rbind
allBuildings <- rbind(buildings, buildings2)
# Print the merged dataframe
AllBuildings
```

OUTPUT:



RESULT:

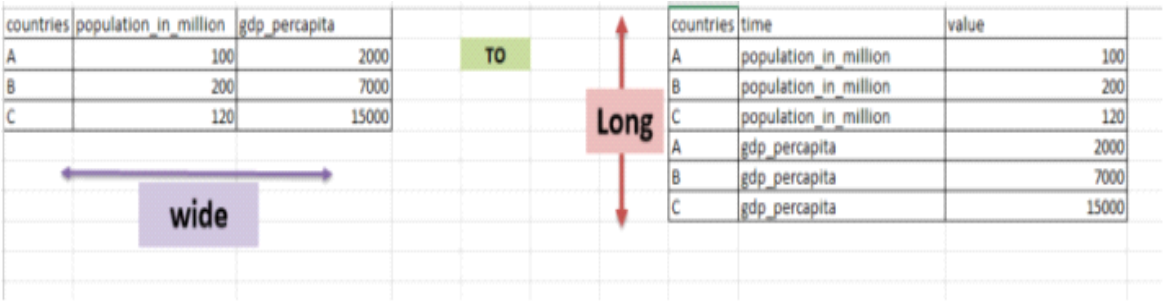
F. RESHAPE FUNCTION IN R

- Construct the following data frame ‘country’.

	countries	value.population_in_million	value.gdp_percapita
1	A	100	2000
2	B	200	7000
3	C	120	15000

- **Reshape in R from wide to long:**

Reshape the above data frame from wide to long format in R.



- data frame "country" is passed to reshape function
- idvar is the variable which need to be left unaltered which is "countries"
- varying are the ones that needs to converted from wide to long
- v.names are the values that should be against the times in the resultant data frame.
- new.row.names is used to assign row names to the resultant dataset
- direction is, to which format the data needs to be transformed

Aim:

Procedure:

Code:

```
> country<-data.frame(c("A","B","C"),c(100,200,120),c(2000,7000,15000))
> colnames(country)<- c("countries","population_in_million","gdp_percapita")
> country
```

OUTPUT:

```
  countries population_in_million gdp_percapita
1         A                  100          2000
2         B                  200          7000
3         C                  120         15000
```

Code: Reshape in R from wide to long

```
> country_w_to_L<- reshape(data=country, idvar="countries",
+                          varying = c("population_in_million","gdp_percapita"),
+                          v.name=c("value"),
+                          times=c("population_in_million","gdp_percapita"),
+                          new.row.names = 1:1000,
+                          direction="long")
> country_w_to_L
```

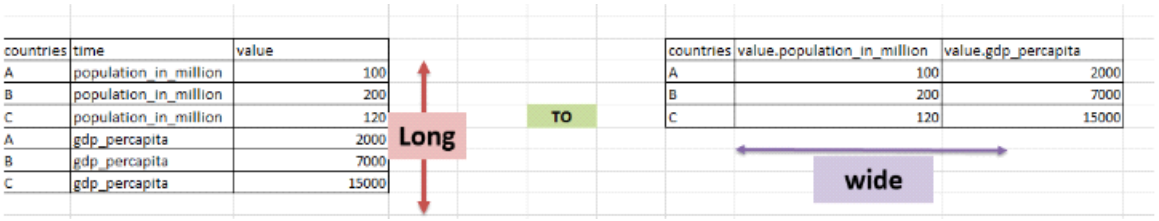
OUTPUT:

```
  countries                time value
1         A population_in_million   100
2         B population_in_million   200
```

3 C population_in_million 120
4 A gdp_percapita 2000
5 B gdp_percapita 7000
6 C gdp_percapita 15000

Result:

- **Reshape in R from long to wide:**



- data (country_w_to_L) which is in long format, is passed to reshape function
- idvar is the variable which need to be left unaltered, which is "countries"
- timevar are the variables that needs to converted to wide format
- v.names are the value variable
- direction is, to which format the data needs to be transformed

Aim:

Procedure:

Code:

```
library(tidyr)
data_long = gather(country, detail, value, population_in_million:gdp_percapita, factor_key=TRUE)
data_long
```

OUTPUT:

countries detail value
1 A population_in_million 100
2 B population_in_million 200
3 C population_in_million 120
4 A gdp_percapita 2000
5 B gdp_percapita 7000
6 C gdp_percapita 15000

Result:

G. MELTING AND CASTING IN R

1. **Melt airquality data set and display as a long – format data ?**
Aim :

Procedure :

Code & Output :

```
> library(reshape2)
>
> # Load the airquality dataset
> data(airquality)
>
> # Melt the dataset
> airquality_long <- melt(airquality, id.vars = c("Month", "Day"))
>
> # View the first few rows of the long-format dataset
> head(airquality_long)
  Month Day variable value
1     5   1    Ozone    41
2     5   2    Ozone    36
3     5   3    Ozone    12
4     5   4    Ozone    18
5     5   5    Ozone    NA
6     5   6    Ozone    28
>
```

RESULT :

2.Melt airquality data and specify month and day to be "ID variables" ?

AIM :

Procedure :

Code & Output :

```
> library(reshape2)
>
> # Load the airquality dataset
> data(airquality)
>
> # Melt the dataset and specify Month and Day as ID variables
> airquality_long <- melt(airquality, id.vars = c("Month", "Day"), variable.name = "Measurements", value.name = "Values")
>
> # View the first few rows of the long-format dataset
> head(airquality_long)
  Month Day Measurements Values
1     5   1       Ozone     41
2     5   2       Ozone     36
3     5   3       Ozone     12
4     5   4       Ozone     18
5     5   5       Ozone     NA
6     5   6       Ozone     28
>
```

Result :

3.Cast the molten airquality data set .
Aim :

Procedure :

Code & Output :

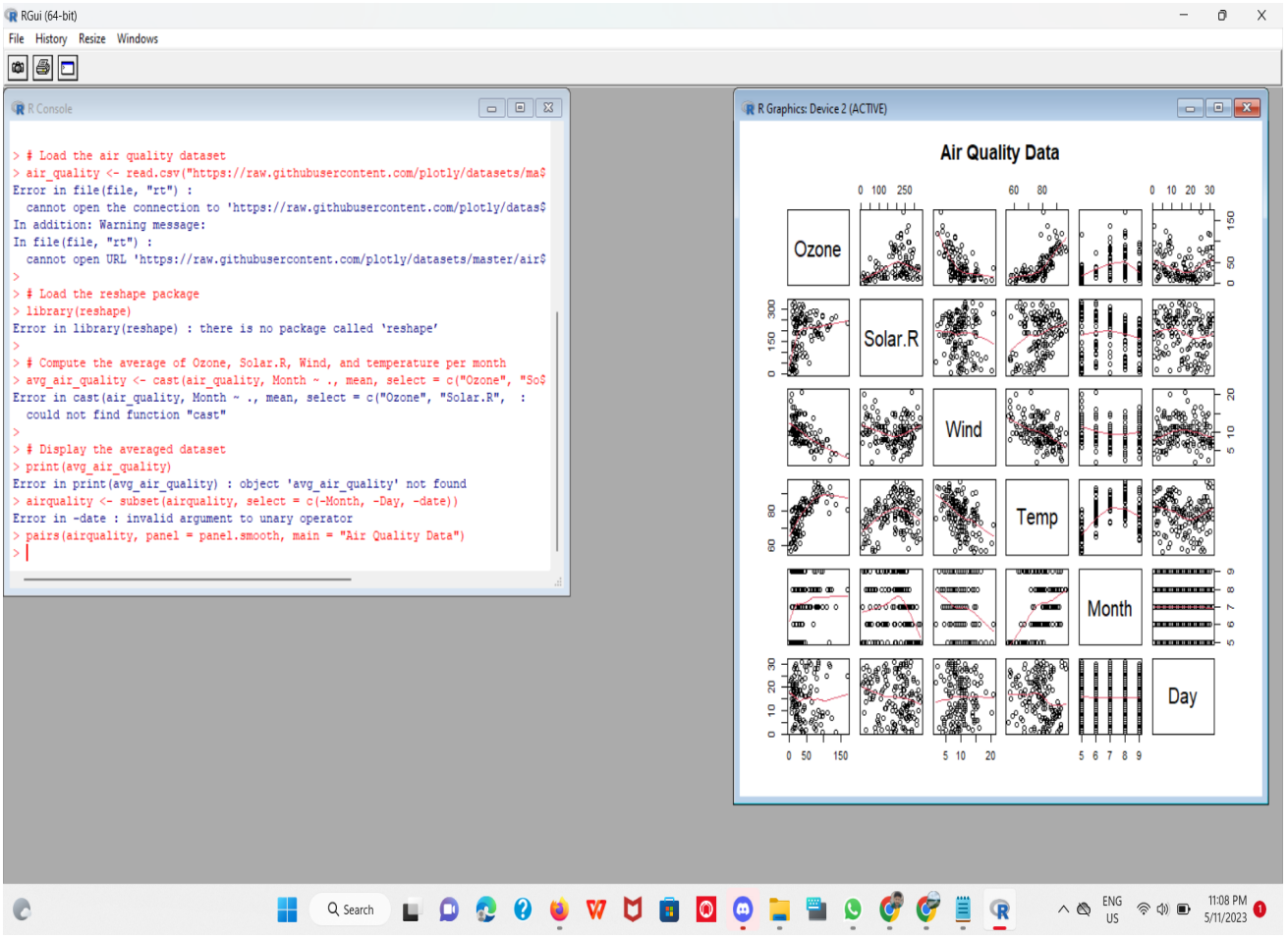
```
> library(reshape2)
>
> # Load the airquality dataset
> data(airquality)
>
> # Melt the dataset
> airquality_long <- melt(airquality, id.vars = c("Month", "Day"))
>
> # Cast the dataset back to its original wide-format
> airquality_wide <- dcast(airquality_long, Month + Day ~ variable)
>
> # View the first few rows of the wide-format dataset
> head(airquality_wide)
  Month Day Ozone Solar.R Wind Temp
1     5   1    41     190  7.4   67
2     5   2    36     118  8.0   72
3     5   3    12     149 12.6   74
4     5   4    18     313 11.5   62
5     5   5    NA      NA 14.3   56
6     5   6    28      NA 14.9   66
>
```

Result :

4.Use cast function appropriately and compute the average of Ozone, Solar.R , Wind and temperature per month ?
Aim :

Procedure :

Code & Output :



Result :

H. FILE MANUPULATION IN R

1. Consider the following data present. Create this file using windows notepad . Save the file as **input.csv** using the save As All files(*.*) option in notepad.

```
id,name,salary,start_date,dept
1,Rick,623.3,2012-01-01,IT
2,Dan,515.2,2013-09-23,Operations
3,Michelle,611,2014-11-15,IT
4,Ryan,729,2014-05-11,HR
5,Gary,843.25,2015-03-27,Finance
6,Nina,578,2013-05-21,IT
7,Simon,632.8,2013-07-30,Operations
8,Guru,722.5,2014-06-17,Finance
```

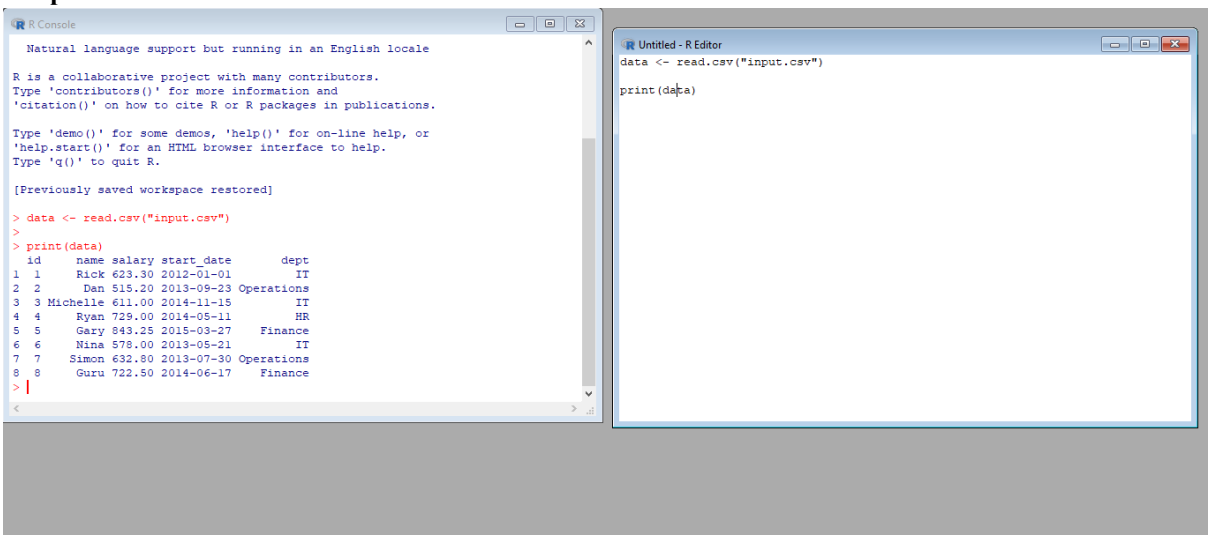
Aim:

Procedure:

Program :

```
data <- read.csv("input.csv")
print(data)
```

Output:



Result:

2. Use appropriate R commands to read **input.csv** file.

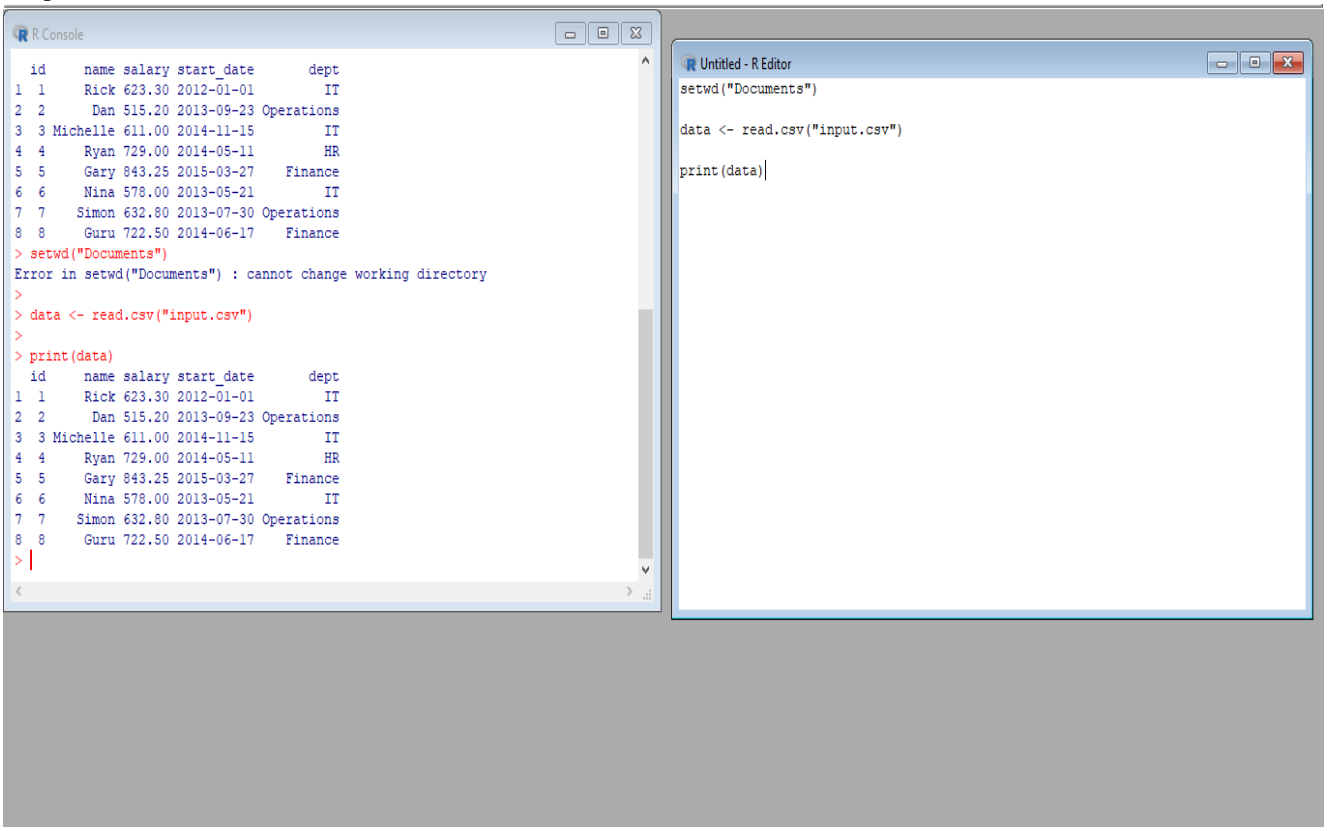
Aim:

Procedure:

Program:

```
setwd("Documents")
data <- read.csv("input.csv")
print(data)
```

Output:



Result:

3. Analyze the CSV File and compute the following.

a. Get the maximum salary
b. Get the details of the person with max salary
c. Get all the people working in IT department
d. Get the persons in IT department whose salary is greater than 600
e. Get the people who joined on or after 2014

Aim:

Procedure:

Program:

```
setwd("Documents")
data <- read.csv("input.csv")
    a)  max_salary <- max(data$salary)
print(paste("Maximum Salary:", max_salary))
    b)  person_with_max_salary <- data[data$salary == max_salary,]
print("Details of person with max salary:")
print(person_with_max_salary)
    c)  it_dept_employees <- data[data$dept == "IT",]
print("Employees working in IT department:")
print(it_dept_employees)
    d)  it_dept_high_salary_employees <- it_dept_employees[it_dept_employees$salary > 600,]
print("Employees working in IT department with salary greater than 600:")
print(it_dept_high_salary_employees)
    e)  employees_joined_after_2014 <- data[as.Date(data$start_date) >= as.Date("2014-01-01"),]
print("Employees who joined on or after 2014:")
print(employees_joined_after_2014)
```

Output:

a)

```
> max_salary <- max(data$salary)
> print(paste("Maximum Salary:", max_salary))
[1] "Maximum Salary: 843.25"
```

b)

```
[1] "Details of person with max salary:"
> print(person_with_max_salary)
  id name salary start_date    dept
5  5 Gary 843.25 2015-03-27 Finance
```

c)

```
[1] "Employees working in IT department:"
> print(it_dept_employees)
  id      name salary start_date dept
1  1      Rick  623.3 2012-01-01   IT
3  3 Michelle  611.0 2014-11-15   IT
6  6      Nina  578.0 2013-05-21   IT
>
```

d)

```
[1] "Employees working in IT department with salary greater than 600:"
> print(it_dept_high_salary_employees)
  id      name salary start_date dept
1  1      Rick  623.3 2012-01-01   IT
3  3 Michelle  611.0 2014-11-15   IT
>
```

e)

```
[1] "Employees who joined on or after 2014:"
> print(employees_joined_after_2014)
  id      name salary start_date     dept
3  3 Michelle 611.00 2014-11-15       IT
4  4      Ryan 729.00 2014-05-11       HR
5  5      Gary 843.25 2015-03-27 Finance
8  8      Guru 722.50 2014-06-17 Finance
>
```

Result:

4. Get the people who joined on or after 2014 and write the output onto a file called output.csv

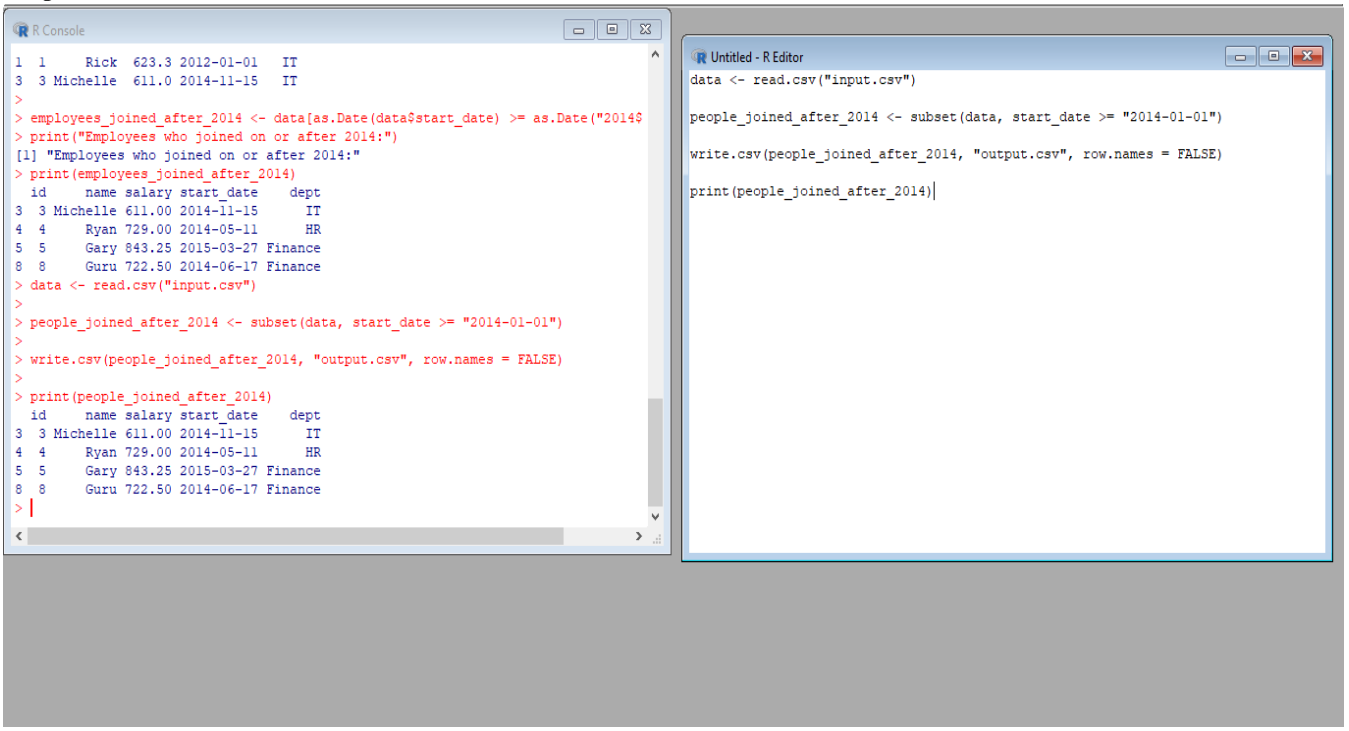
Aim:

Procedure:

Program:

```
data <- read.csv("input.csv")
people_joined_after_2014 <- subset(data, start_date >= "2014-01-01")
write.csv(people_joined_after_2014, "output.csv", row.names = FALSE)
print(people_joined_after_2014)
```

Output:



Result:

I. UNIVARIATE ANALYSIS IN R - MEASURES OF CENTRAL TENDENCY

1. ARITHMETIC MEAN

a) Write suitable R code to compute the average of the following values 12,7,3,4.2,18,2,54, -21,8, -5

Aim:

Procedure:

CODE:

```
values<- c(12, 7, 3, 4.2, 18, 2, 54, -21, 8, -5)
mean_value <- mean(values)
print(mean_value)
```

OUTPUT:

```
[1] 8.22
```

b)Compute the mean after applying the trim option and removing 3 values from each end.

Aim:

Procedure:

CODE:

```
values <- c(12, 7, 3, 4.2, 18, 2, 54, -21, 8, -5)
trimmed_mean <- mean(values, trim = 0.3)
print(trimmed_mean)
```

OUTPUT:

```
[1] 5.55
```

c) Compute the mean of the following vector. (12,7,3,4.2,18,2,54,-21,8,-5,NA)

#If there are missing values, then the mean function returns NA.
Find mean dropping NA values.
#To drop the missing values from the calculation use na.rm = TRUE

Aim:

Procedure:

CODE:

```
values <- c(12, 7, 3, 4.2, 18, 2, 54, -21, 8, -5, NA)
mean_without_na <- mean(values, na.rm = TRUE)
print(mean_without_na)
```

OUTPUT:

[1] 8.22

RESULT:

2.MEDIAN

Write suitable R code to compute the median of the following values. 12,7,3,4.2,18,2,54,-21,8,-5

AIM:

PROCEDURE:

CODE:

```
a<- c(12, 7, 3, 4.2, 18, 2, 54, -21, 8, -5)
median_of_a <- median(a)
print(median_of_a)
```

OUTPUT:

[1] 5.6

RESULT:

3. MODE

Calculate the mode for the following numeric as well as character data set in R.
(2,1,2,3,1,2,3,4,1,5,5,3,2,3) , ("o","it","the","it","it")

AIM:

PROCEDURE:

CODE:

```
get_mode <- function(data) {
  unique_values <- unique(data)
  frequencies <- tabulate(match(data, unique_values))
  mode <- unique_values[which.max(frequencies)]
  return(mode)
}

numeric_data <- c(2, 1, 2, 3, 1, 2, 3, 4, 1, 5, 5, 3, 2, 3)
```

```
mode_numeric <- get_mode(numeric_data)
print(mode_numeric)
character_data <- c("o", "it", "the", "it", "it")
mode_character <- get_mode(character_data)
print(mode_character)
```

OUTPUT:

```
> print(mode_numeric)
[1] 2
> print(mode_character)
[1] "it"
```

RESULT:

J. UNIVARIATE ANALYSIS IN R - MEASURES OF DISPERSION

1. Download mpg dataset which contains Fuel economy data from 1999 and 2008 for 38 popular models of car from the URL given below.

https://vincentarelbundock.github.io/Rdatasets/datasets.html

Aim:

Procedure:

Answer the following queries

- **Find the car which gives maximum city miles per gallon**

Code:

```
data("mtcars")

# Find the car with the maximum city miles per gallon
max_city_mpg <- mtcars[which.max(mtcars$mpg), ]

# Print the car name and its city miles per gallon
cat("The car with the maximum city miles per gallon is", rownames(max_city_mpg), "with", max_city_mpg$mpg, "mpg.")
```

Output:

The car with the maximum city miles per gallon is Toyota Corolla with 33.9 mpg.

Result:

- **Find the cars which gives minimum disp in compact and subcompact class**

Code:

```
# Load the MTcars dataset
data(mtcars)

# Create a subset of the data containing only the compact and subcompact cars
compact_cars <- subset(mtcars, cyl == 4 & (am == 0 | gear == 4))

# Find the car with the minimum displacement in the compact and subcompact classes
min_disp_compact <- compact_cars[which.min(compact_cars$disp), ]

# Print the car names and their displacements
cat("The car(s) with the minimum displacement in the compact and subcompact classes is/are:")
print(data.frame(disp = min_disp_compact$disp, car = rownames(min_disp_compact)))
```

Output:

The car(s) with the minimum displacement in the compact and subcompact classes is/are:

disp car

1 71.1 Toyota Corolla

Result:

2. Use the same dataset as used in Exercise 1 and perform the following queries

Aim:

Procedure:

- Find the standard deviation of city milles per gallon

Code:

```
# Load the MTcars dataset
data(mtcars)
# Calculate the standard deviation of city miles per gallon (mpg)
sd_mpg_city <- sd(mtcars$mpg)

# Print the standard deviation of city miles per gallon
cat("The standard deviation of city miles per gallon is:", sd_mpg_city)
```

Output:

The standard deviation of city miles per gallon is: 6.026948

Result:

- **Find the variance of highway milles per gallon**

Code:

```
data("mtcars")
# Calculate the variance of highway miles per gallon (mpg)
var_mpg_highway <- var(mtcars$hp)

# Print the variance of highway miles per gallon
cat("The variance of highway miles per gallon is:", var_mpg_highway)
```

Output:

The variance of highway miles per gallon is: 4700.867

Result:

3. Use the same dataset and perform the following queries

Aim:

Procedure:

- **Find the range of the disp in the data set mpg**

Code:

```
# Load the MTcars dataset
data(mtcars)

# Calculate the range of displacement (disp) values in the dataset
range_disp <- range(mtcars$disp)

# Print the range of displacement values
cat("The range of displacement values in the dataset is:", range_disp)
```

Output:

The range of displacement values in the dataset is: 71.1 472

Result:

- **Find the Quartile of the disp in the data set mpg**

Code:

```
# Load the MTcars dataset
```

```
data(mtcars)


# Calculate the quartiles of displacement (disp) values in the dataset
quartiles_disp <- quantile(mtcars$disp, probs = c(0.25, 0.5, 0.75))

# Print the quartiles of displacement values
cat("The quartiles of displacement values in the dataset are:")
print(quartiles_disp)
```

Output:
The variance of highway miles per gallon is: 4700.867
Result:

- **Find the IQR of the disp column in the data set mpg**

Code:

```
data("mtcars")
# Calculate the variance of highway miles per gallon (mpg)
var_mpg_highway <- var(mtcars$hp)
# Print the variance of highway miles per gallon
cat("The variance of highway miles per gallon is:", var_mpg_highway)
```

Output:
The quartiles of displacement values in the dataset are:
25% 50% 75%
120.825 196.300 326.000

Result:

4.

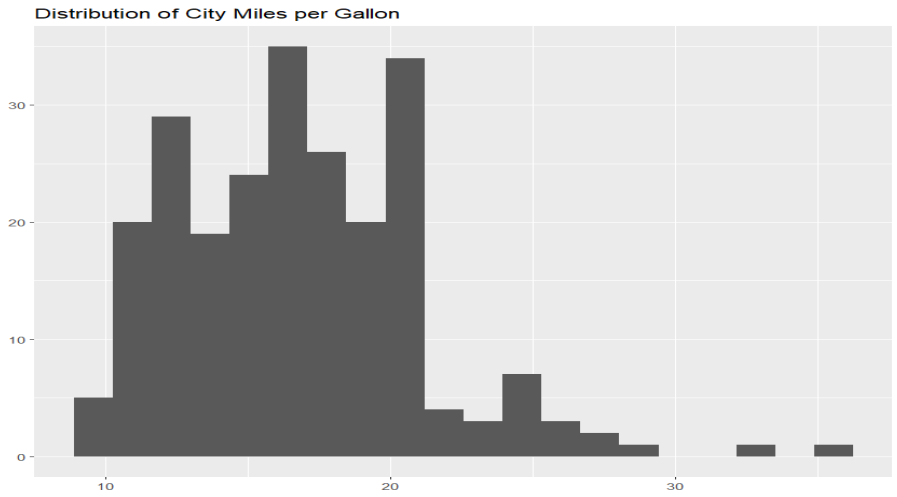
```
#Install Library

library(e1071)
```

- Find the skewness of city miles per mileage in the data set mpg ?
  Use qplot function and display the graph for the city miles per mileage column

Code:

```
# Load the mpg dataset
library(ggplot2)
data(mpg)
# Calculate the skewness of city miles per gallon
library(moments)
skew_mpg_city <- skewness(mpg$cty)
# Print the skewness value
cat("The skewness of city miles per gallon is:", skew_mpg_city)
# Create a histogram of city miles per gallon
qplot(x = cty, data = mpg, geom = "histogram", bins = 20,
     main = "Distribution of City Miles per Gallon", xlab = "City MPG")
```

Output:



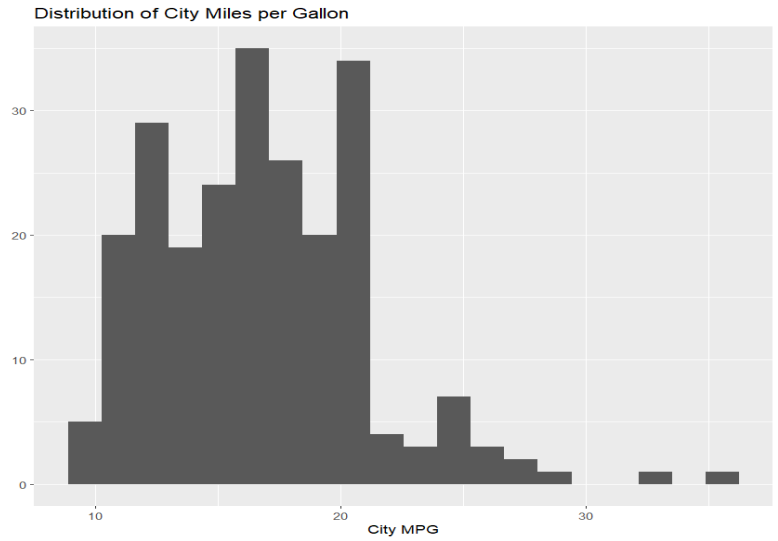
Result:

- Find the kurtosis of city miles per mileage in the data set mpg
  Use qplot function and display the graph for the city miles per mileage column

  Code:

```
# Load the mpg dataset
library(ggplot2)
data(mpg)

# Calculate the kurtosis of city miles per gallon
library(moments)
kurt_mpg_city <- kurtosis(mpg$cty)

# Print the kurtosis value
cat("The kurtosis of city miles per gallon is:", kurt_mpg_city)

# Create a histogram of city miles per gallon
qplot(x = cty, data = mpg, geom = "histogram", bins = 20,
      main = "Distribution of City Miles per Gallon", xlab = "City MPG")
```

  Output:



  Result:

K. BIVARIATEANALYSIS IN R -COVARIANCE, CORRELATION, CROSSTAB

1

	Reference	Status	Gender	TestNewOrFollowUp
1	KRXH	Accepted	Female	Test1 New
2	KRPT	Accepted	Male	Test1 New
3	FHRA	Rejected	Male	Test2 New
4	CZKK	Accepted	Female	Test3 New
5	CQTN	Rejected	Female	Test1 New
6	PZXW	Accepted	Female	Test4 Follow-up
7	SZRZ	Rejected	Male	Test4 New
8	RMZE	Rejected	Female	Test2 New
9	STNX	Accepted	Female	Test3 New
10	TMDW	Accepted	Female	Test1 New

i) Load the dataset and Create a data frame and name it as dataframe1

ii) Load the function for crosstab

xtabs(~colname , data=Data frame name)

AIM:

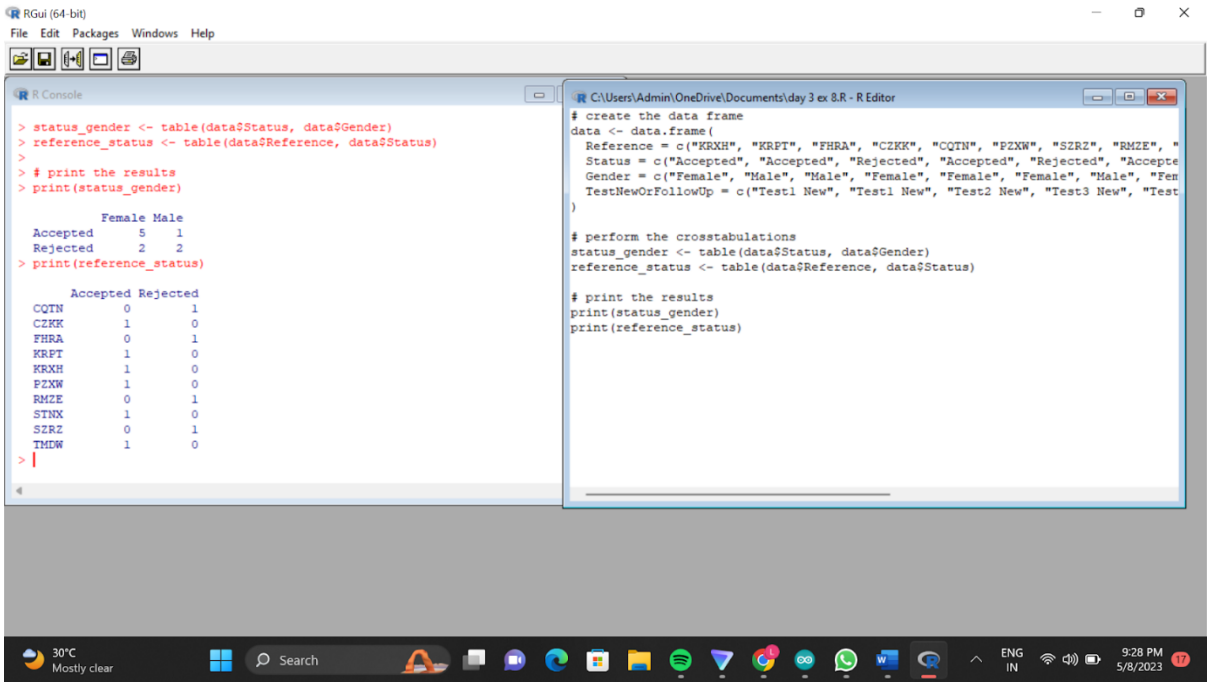
PROCEDURE:

CODE:

```
# load the data from dataframe1
dataframe1 <- data.frame(
  Reference = c("KRXH", "KRPT", "FHRA", "CZKK", "CQTN", "PZXW", "SZRZ", "RMZE", "STNX",
"TMDW"),
  Status = c("Accepted", "Accepted", "Rejected", "Accepted", "Rejected", "Accepted", "Rejected", "Rejected",
"Accepted", "Accepted"),
  Gender = c("Female", "Male", "Male", "Female", "Female", "Female", "Male", "Female", "Female",
"Female"),
  TestNewOrFollowUp = c("Test1 New", "Test1 New", "Test2 New", "Test3 New", "Test1 New",
"Follow-up", "Test4 New", "Test2 New", "Test3 New", "Test1 New")
)
```

```
# i) explore the relationship between two categorical variables
# contingency table between Gender and Status
gender_status <- table(dataframe1$Gender, dataframe1$Status)
print(gender_status)

# ii) create a table between Reference and Status
reference_status <- xtabs(~ Reference + Status, data = dataframe1)
print(reference_status)

# iii) save the file in the name of dataframe2
dataframe2 <- reference_status
```

OUTPUT:



RESULT:

2

i) Use Two Categorical Variables and Discover the relationships within a dataset

ii) Next, using the xtabs() function, apply two variables from "dataframe1 ", to create a table delineating the relationship between the "Reference" category, and the "Status" category.

iii) Save the file in the name of dataframe2

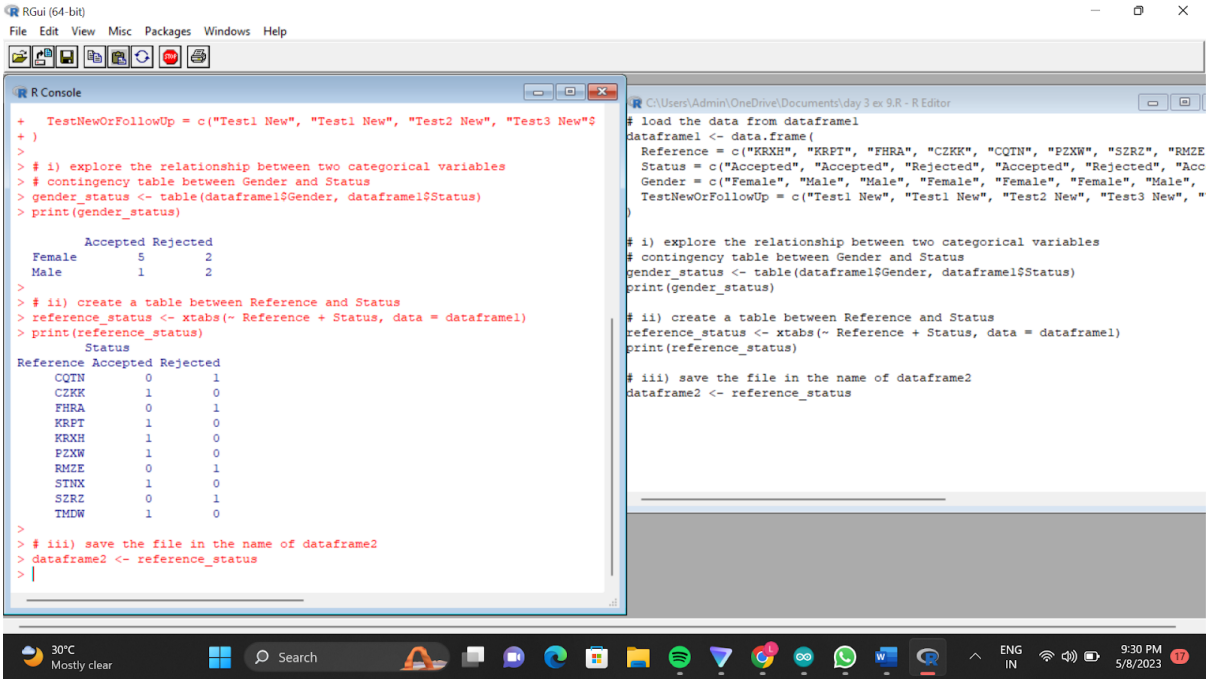
AIM:

PROCEDURE:

CODE:

```
# load the data from dataframe1
dataframe1 <- data.frame(
  Reference = c("KRXH", "KRPT", "FHRA", "CZKK", "CQTN", "PZXW", "SZRZ", "RMZE", "STNX", "TMDW"),
  Status = c("Accepted", "Accepted", "Rejected", "Accepted", "Rejected", "Accepted", "Rejected", "Rejected", "Accepted",
"Accepted"),
  Gender = c("Female", "Male", "Male", "Female", "Female", "Female", "Male", "Female", "Female", "Female"),
  TestNewOrFollowUp = c("Test1 New", "Test1 New", "Test2 New", "Test3 New", "Test1 New", "Follow-up", "Test4
New", "Test2 New", "Test3 New", "Test1 New")
)

# i) explore the relationship between two categorical variables
# contingency table between Gender and Status
gender_status <- table(dataframe1$Gender, dataframe1$Status)
print(gender_status)

# ii) create a table between Reference and Status
reference_status <- xtabs(~ Reference + Status, data = dataframe1)
print(reference_status)

# iii) save the file in the name of dataframe2
dataframe2 <- reference_status
```

OUTPUT:



RESULT:

3

Use the same data frame using three Categorical Variables create a Multi-Dimensional Table

Apply three variables from "dataframe1" to create a Multi-Dimensional Cross-Tabulation of "Status", "Gender", and "Test".

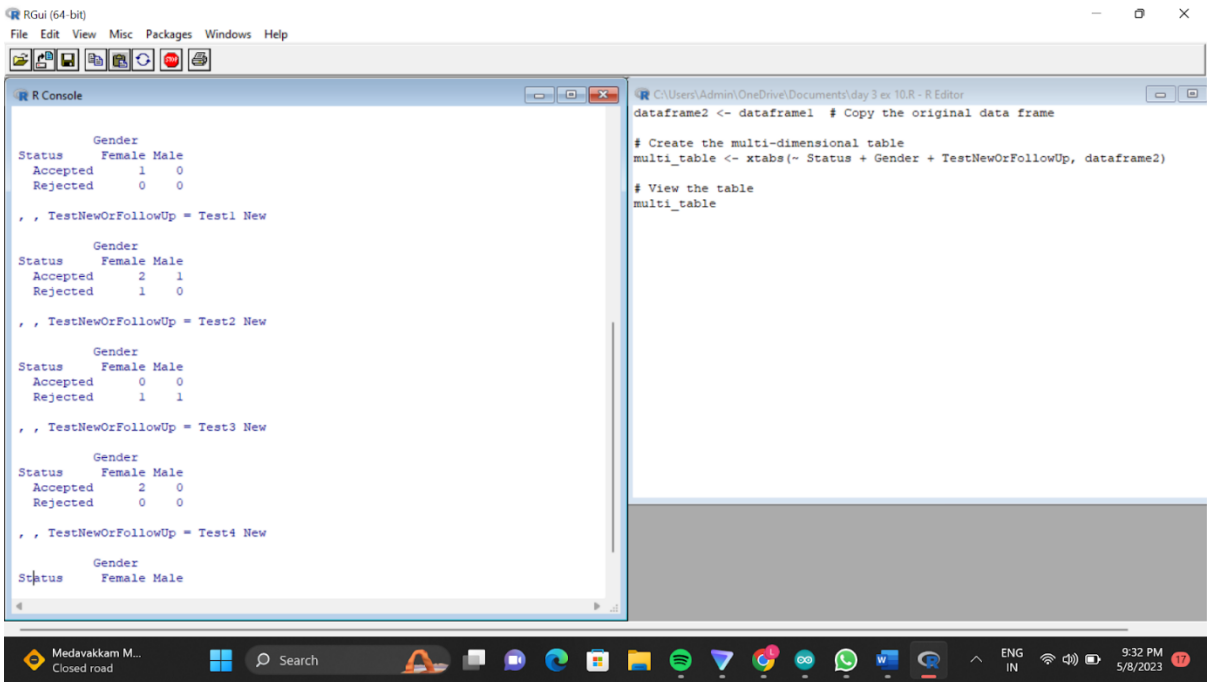
AIM:

PROCEDURE:

CODE:

```
dataframe2 <- dataframe1  # Copy the original data frame

# Create the multi-dimensional table
multi_table <- xtabs(~ Status + Gender + TestNewOrFollowUp, dataframe2)

# View the table
multi_table
```

OUTPUT:



RESULT:

4 Covariance

i) For the Dataframe1 created from exercise 2 calculate the covariance between Refrence column and Status column

ii) Display the covariance matrix

AIM:

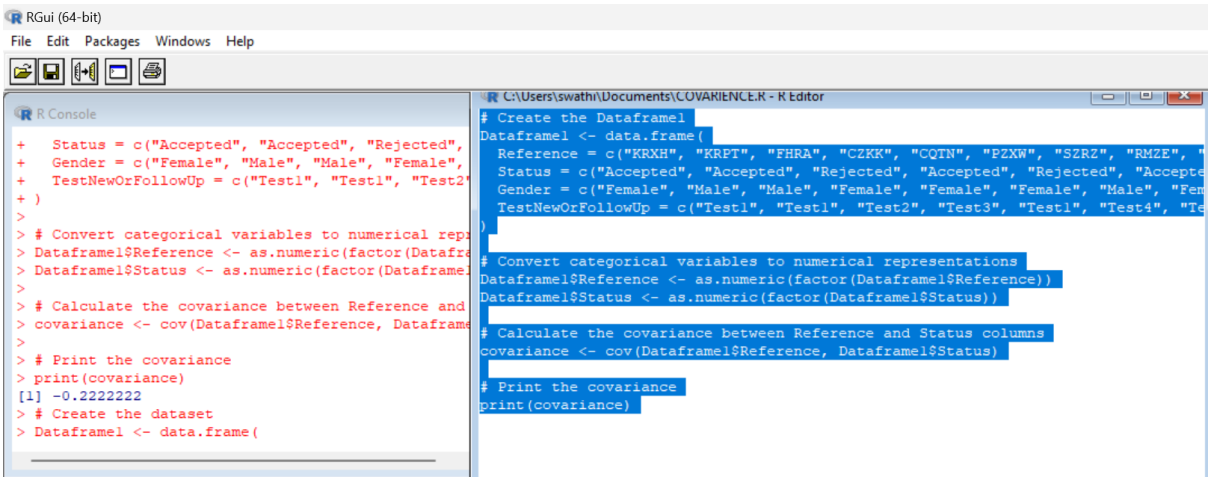
PROCEDURE:

CODE:

```
# Create the Dataframe1
Dataframe1 <- data.frame(
  Reference = c("KRXH", "KRPT", "FHRA", "CZKK", "CQTN", "PZXW", "SZRZ", "RMZE", "STNX", "TMDW"),
    Status = c("Accepted", "Accepted", "Rejected", "Accepted", "Rejected", "Accepted", "Rejected", "Rejected",
"Accepted", "Accepted"),
  Gender = c("Female", "Male", "Male", "Female", "Female", "Female", "Male", "Female", "Female", "Female"),
```

```
  TestNewOrFollowUp = c("Test1", "Test1", "Test2", "Test3", "Test1", "Test4", "Test4", "Test2", "Test3", "Test1")
)

# Convert categorical variables to numerical representations
Dataframe1$Reference <- as.numeric(factor(Dataframe1$Reference))
Dataframe1$Status <- as.numeric(factor(Dataframe1$Status))

# Calculate the covariance between Reference and Status columns
covariance <- cov(Dataframe1$Reference, Dataframe1$Status)

# Print the covariance
print(covariance)
```

OUTPUT:



RESULT:

5 Correlation

Find the Correlation between gender and status. what kind of correlation does exist between the two?

AIM:

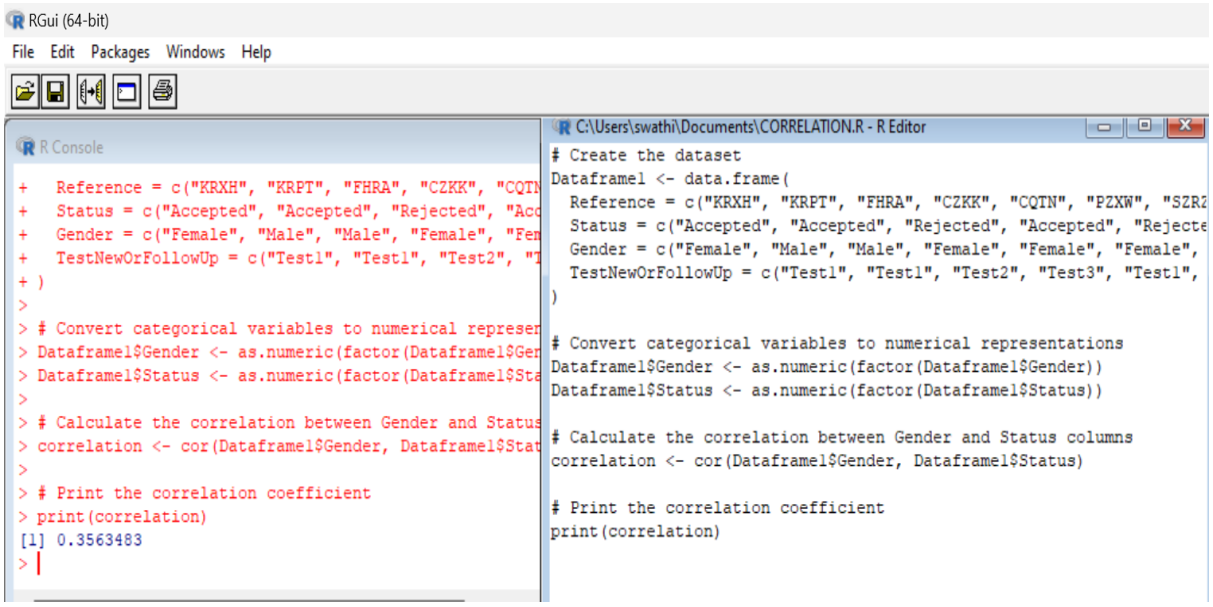
PROCEDURE:

CODE:

```
# Create the dataset
Dataframe1 <- data.frame(
  Reference = c("KRXH", "KRPT", "FHRA", "CZKK", "CQTN", "PZXW", "SZRZ", "RMZE", "STNX", "TMDW"),
   Status = c("Accepted", "Accepted", "Rejected", "Accepted", "Rejected", "Accepted", "Rejected", "Rejected", "Accepted",
"Accepted"),
  Gender = c("Female", "Male", "Male", "Female", "Female", "Female", "Male", "Female", "Female", "Female"),
  TestNewOrFollowUp = c("Test1", "Test1", "Test2", "Test3", "Test1", "Test4", "Test4", "Test2", "Test3", "Test1")
)

# Convert categorical variables to numerical representations
Dataframe1$Gender <- as.numeric(factor(Dataframe1$Gender))
Dataframe1$Status <- as.numeric(factor(Dataframe1$Status))

# Calculate the correlation between Gender and Status columns
correlation <- cor(Dataframe1$Gender, Dataframe1$Status)
```

# Print the correlation coefficient
print(correlation)

OUTPUT:



RESULT:

L. VISUALIZATION IN R

1. Write a program for creating a pie-chart in R using the input vector(21,62,10,53). Provide labels for the chart as 'London', 'New York', 'Singapore', 'Mumbai'. Add a title to the chart as 'city pie-chart' and add a legend at the top right corner of the chart.

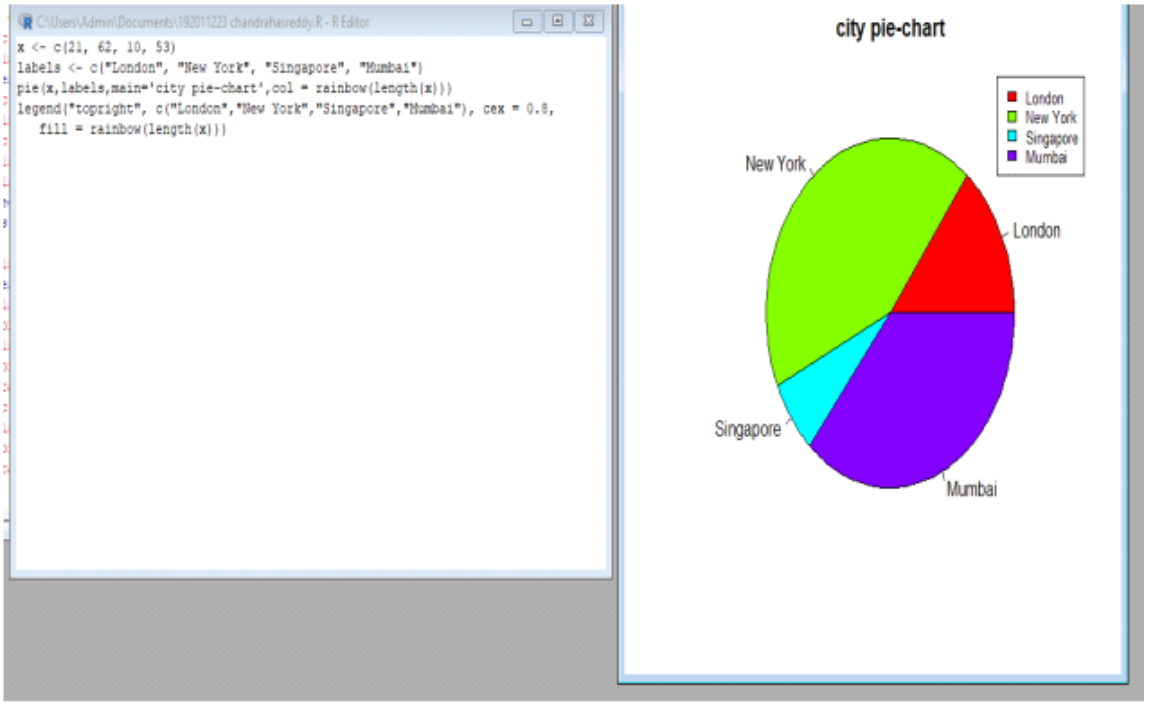
Aim:

Procedure:

Program:

A) x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")
pie(x,labels,main='city pie-chart',col = rainbow(length(x)))
legend("topright", c("London","New York","Singapore","Mumbai"), cex = 0.8,
fill = rainbow(length(x))

Output:



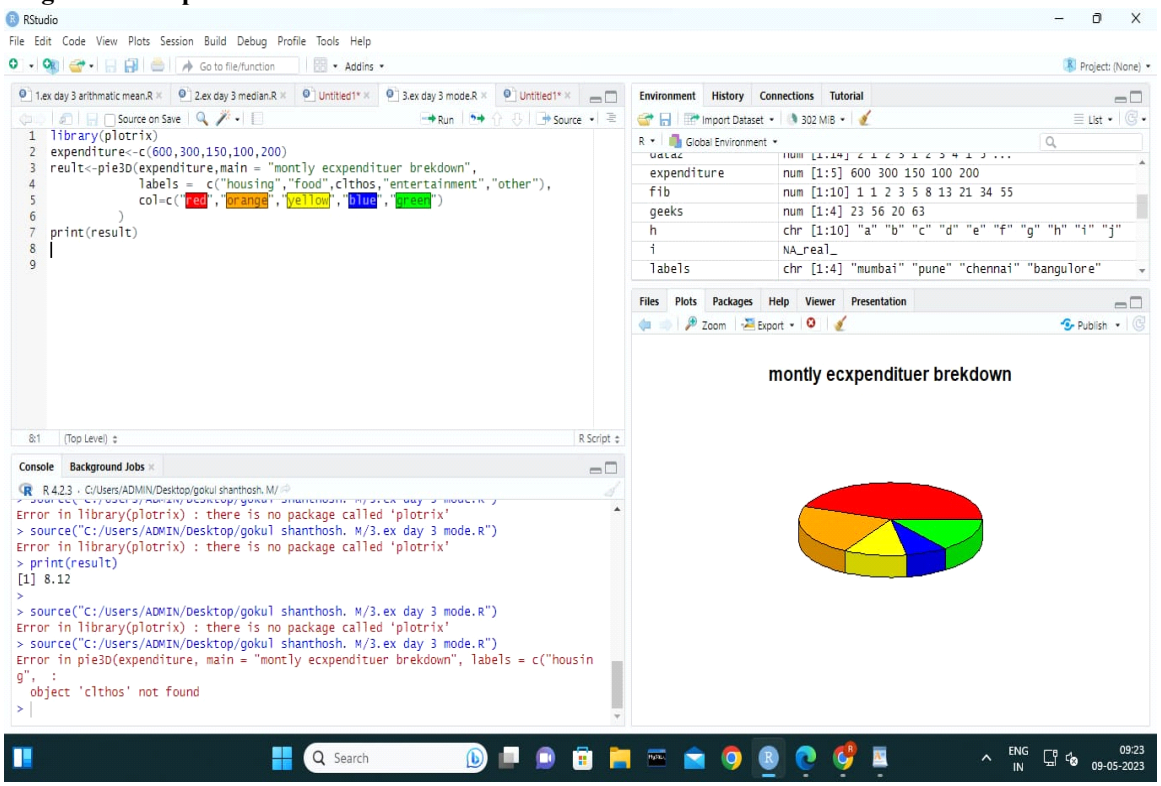
Result:

2. Create a 3D Pie Chart for the dataset "political Knowledge" with suitable labels, colours and a legend at the top right corner of the charT

Aim:

Procedure:

Program & Output:



Result:

3. Write a program for creating a bar chart using the vectors H=c(7,12,28,3,41) and M=c(“mar”, “apr”, “may”, “jun”, “jul”). Add a title to the chart as “Revenue chart”

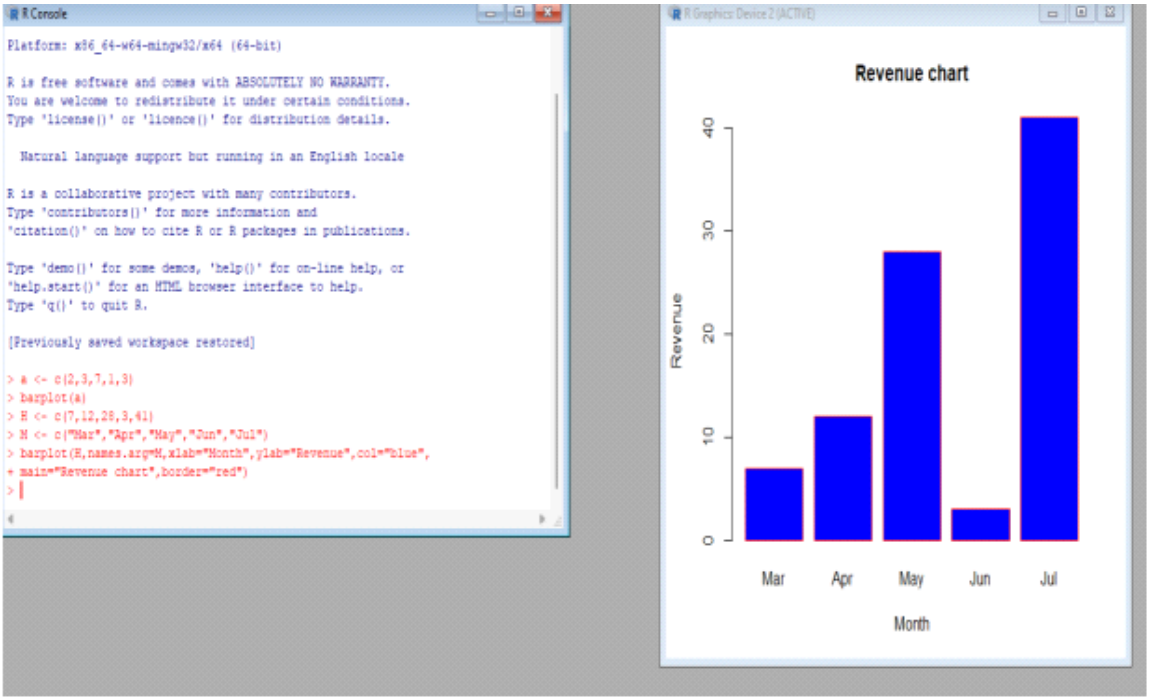
Aim:

Procedure:

Program:

A] a <- c(2,3,7,1,3)
> barplot(a)
> H <- c(7,12,28,3,41)
> M <- c("Mar","Apr","May","Jun","Jul")
> barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="blue",
+ main="Revenue chart",border="red")

Output:



Result:

4.Make a histogram for the "Air Passengers "dataset, start at 100 on the x-axis, and from values 200 to 700, make the bins 200 wide

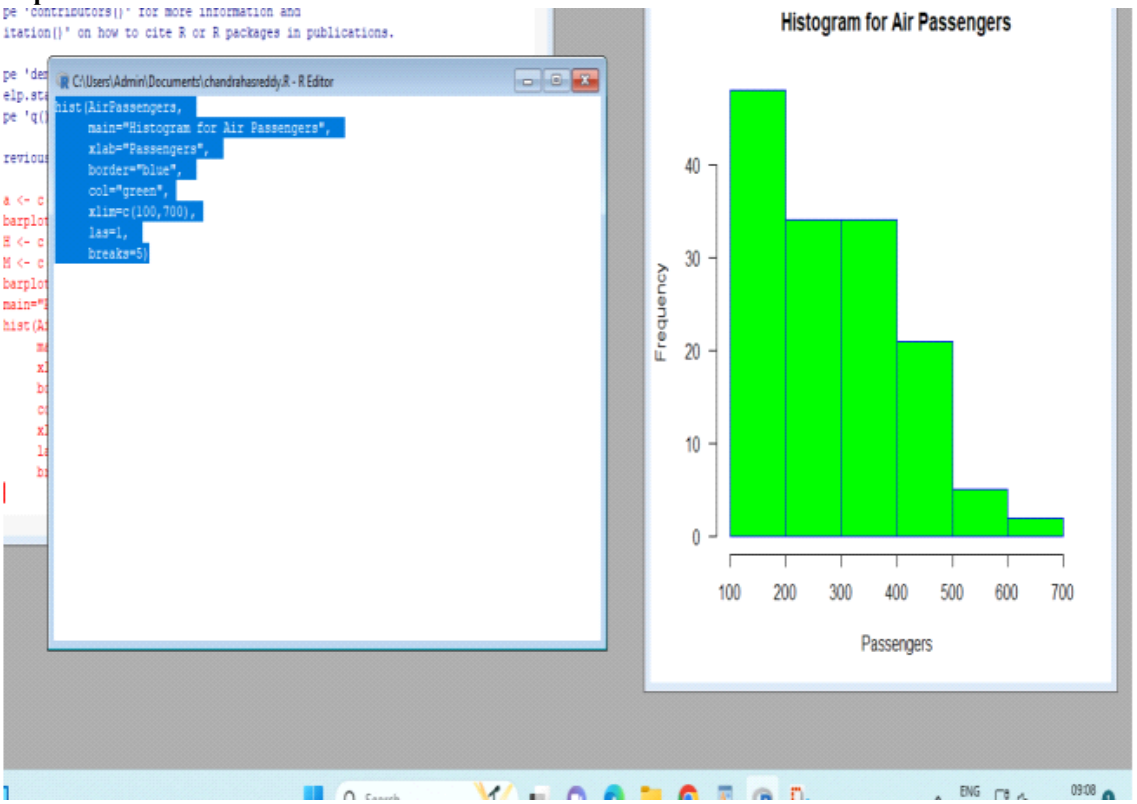
Aim:

Procedure:

Program:
A)hist(AirPassengers,
main="Histogram for Air Passengers",

```
 xlab="Passengers",
 border="blue",
 col="green",
 xlim=c(100,700),
 las=1,
 breaks=5)
```

Output:



Result:

5.Create a Boxplot graph for the relation between "mpg"(miles per galloon) and "cyl"(number of Cylinders) for the dataset "mtcars" available in R Environment.

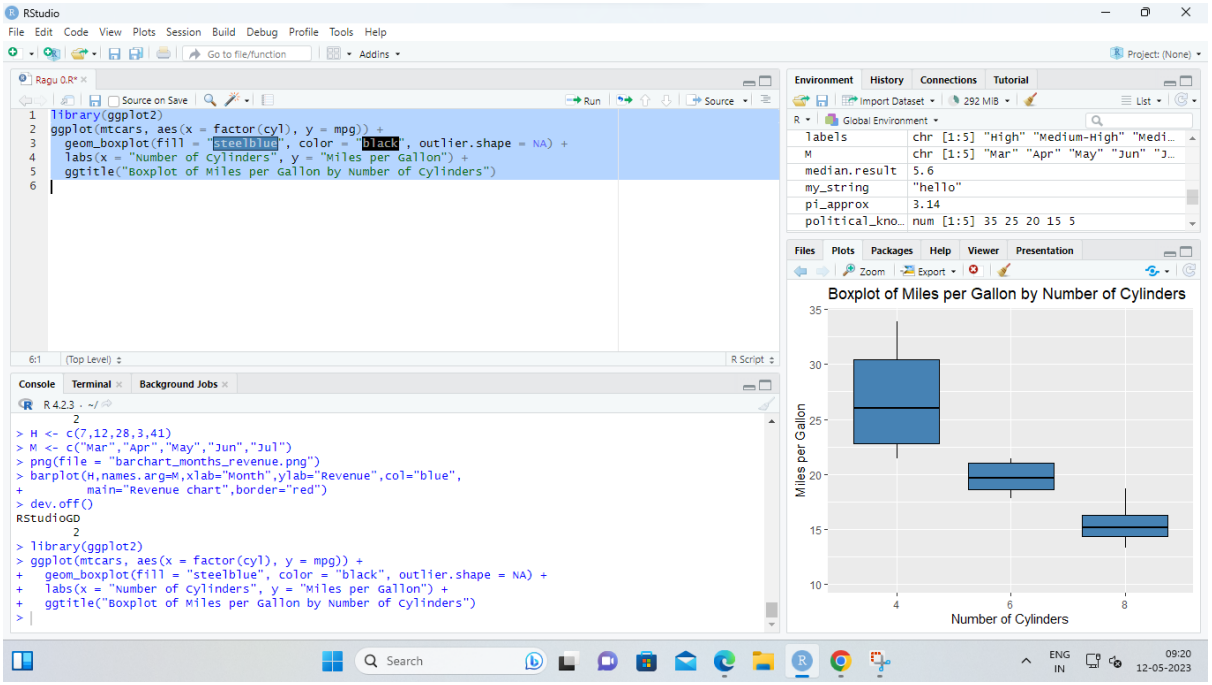
Aim:

Procedure:

Program:

```
library(ggplot2)
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_boxplot(fill = "steelblue", color = "black", outlier.shape = NA) +
  labs(x = "Number of Cylinders", y = "Miles per Gallon") +
  ggtitle("Boxplot of Miles per Gallon by Number of Cylinders")
```

Output:



Result:

M. LINEAR REGRESSION ANALYSIS IN R

1. Using linear regression analysis establish a relationship between height and weight of a person using the input vector given below.
   # Values of height
   151, 174, 138, 186, 128, 136, 179, 163, 152, 131
   # Values of weight.
   63, 81, 56, 91, 47, 57, 76, 72, 62, 48
   Predict the weight of a person with height 170. Visualize the regression graphically.
   Aim:

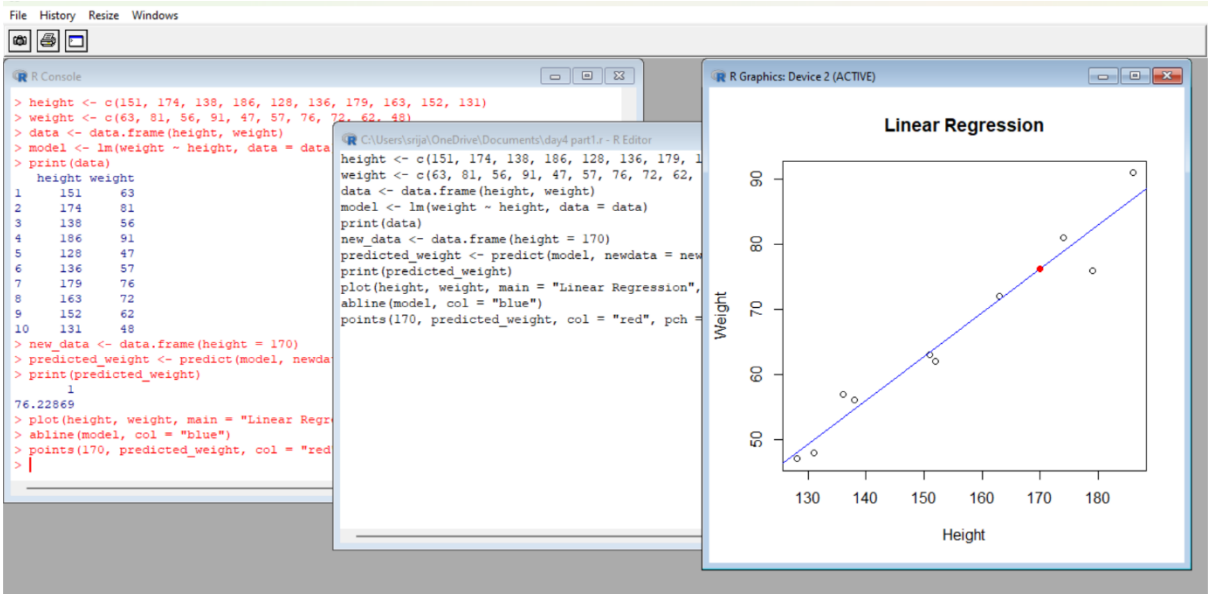
   Procedure:

   Program:

```
height <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
weight <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
data <- data.frame(height, weight)
model <- lm(weight ~ height, data = data)
print(data)
new_data <- data.frame(height = 170)
predicted_weight <- predict(model, newdata = new_data)
print(predicted_weight)
plot(height, weight, main = "Linear Regression", xlab = "Height", ylab = "Weight")
abline(model, col = "blue")
points(170, predicted_weight, col = "red", pch = 16)
```

Output:

Predicted weight: 76.22869



Result:

2. Download the Dataset "water". Find out whether there is a linear relation between attributes"mortality" and"hardness" by plot function. Fit the Data into the Linear Regression model.Predict the mortality for the hardness=88

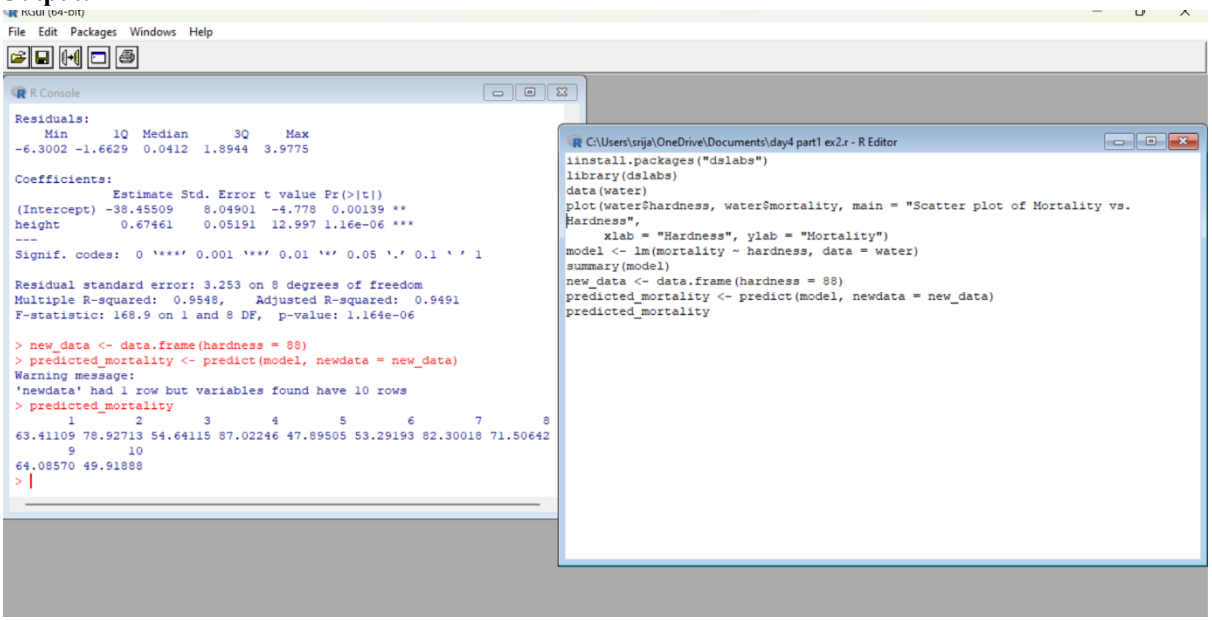
AIM:

PROCEDURE:

PROGRAM:

```
iinstall.packages("dslabs")
library(dslabs)
data(water)
plot(water$hardness, water$mortality, main = "Scatter plot of Mortality vs.Hardness",
xlab = "Hardness", ylab = "Mortality")
model <- lm(mortality ~ hardness, data = water)
summary(model)
new_data <- data.frame(hardness = 88)
predicted_mortality <- predict(model, newdata = new_data)
predicted_mortality
```

Output:



Result:

N. MULTIPLE REGRESSION ANALYSIS IN R

1.Generate a multiple regression model using the built in dataset mtcars.It gives a comparison between different car models in terms of mileage per gallon (mpg), cylinder displacement("disp"), horse power("hp"), weight of the car("wt") and some more parameters.
Establish the relationship between "mpg" as a response variable with "disp","hp" and "wt" as predictor variables. Predict the mileage of the car with dsp=221,hp=102 and wt=2.91.

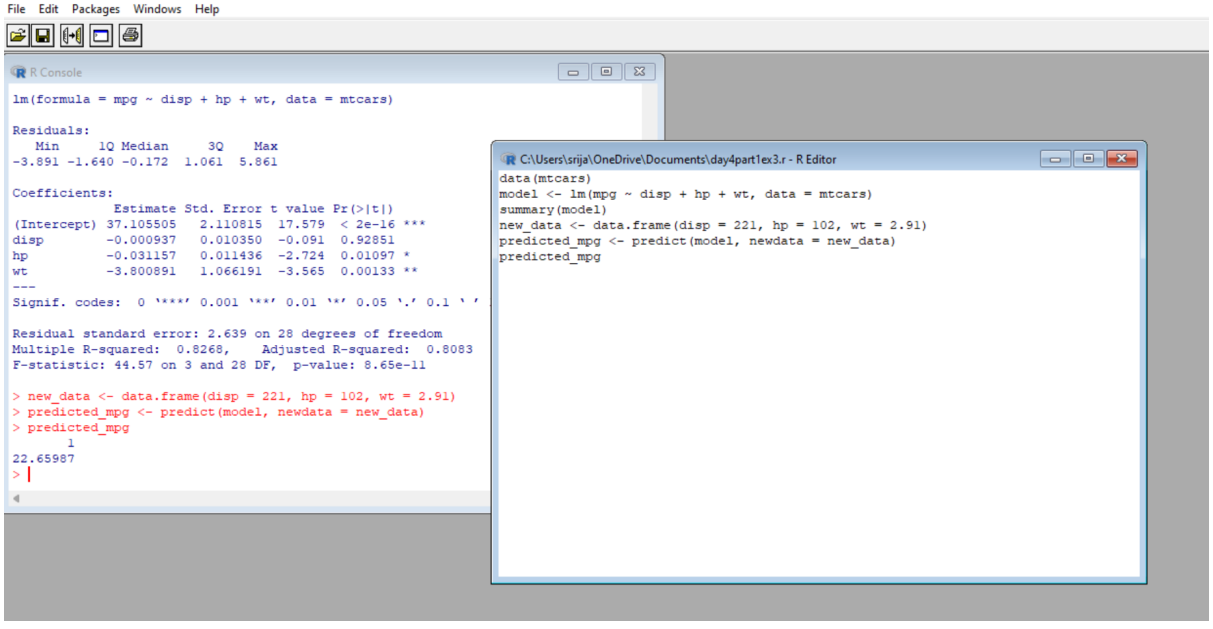
AIM:

PROCEDURE:

PROGRAM:

```
data(mtcars)
model <- lm(mpg ~ disp + hp + wt, data = mtcars)
summary(model)
new_data <- data.frame(disp = 221, hp = 102, wt = 2.91)
predicted_mpg <- predict(model, newdata = new_data)
predicted_mpg
```

Output:



Result:

2. Consider the data set "delivery" available in the R environment. It gives a deliverytime (“delTime”)of production materials(number of productions “n.prod”) with the given distance(“distance”) to reach the destination place.

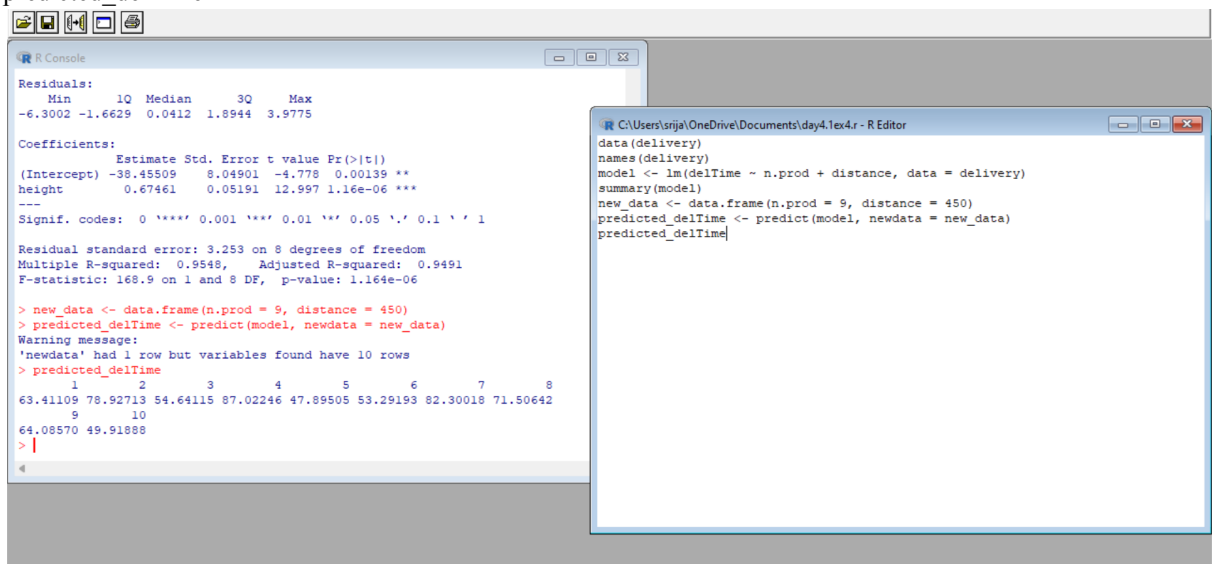
a)Create the model to establish the relationship between "delTime" as a response variable with "n.prod" and "distance" as predictor variables.
b)Predict the delTime for the given number of production(“n.prod”)=9 and distance(“distance”)=450

Aim:

Procedure:

Program & Output:

```
data(delivery)
names(delivery)
model <- lm(delTime ~ n.prod + distance, data = delivery)
summary(model)
new_data <- data.frame(n.prod = 9, distance = 450)
predicted_delTime <- predict(model, newdata = new_data)
predicted_delTime
```



Result:

O. LOGISTIC REGRESSION ANALYSIS IN R

1. Create a logistic regression model using the “mtcars” data set with the information given below.
The in-built data set "mtcars" describes different models of a car with their various engine specifications. In "mtcars" data set, the transmission mode (automatic or manual) is described by the column am which is a binary value (0 or 1). Create a logistic regression model between the columns "am" and 3 other columns - hp, wt and cyl.

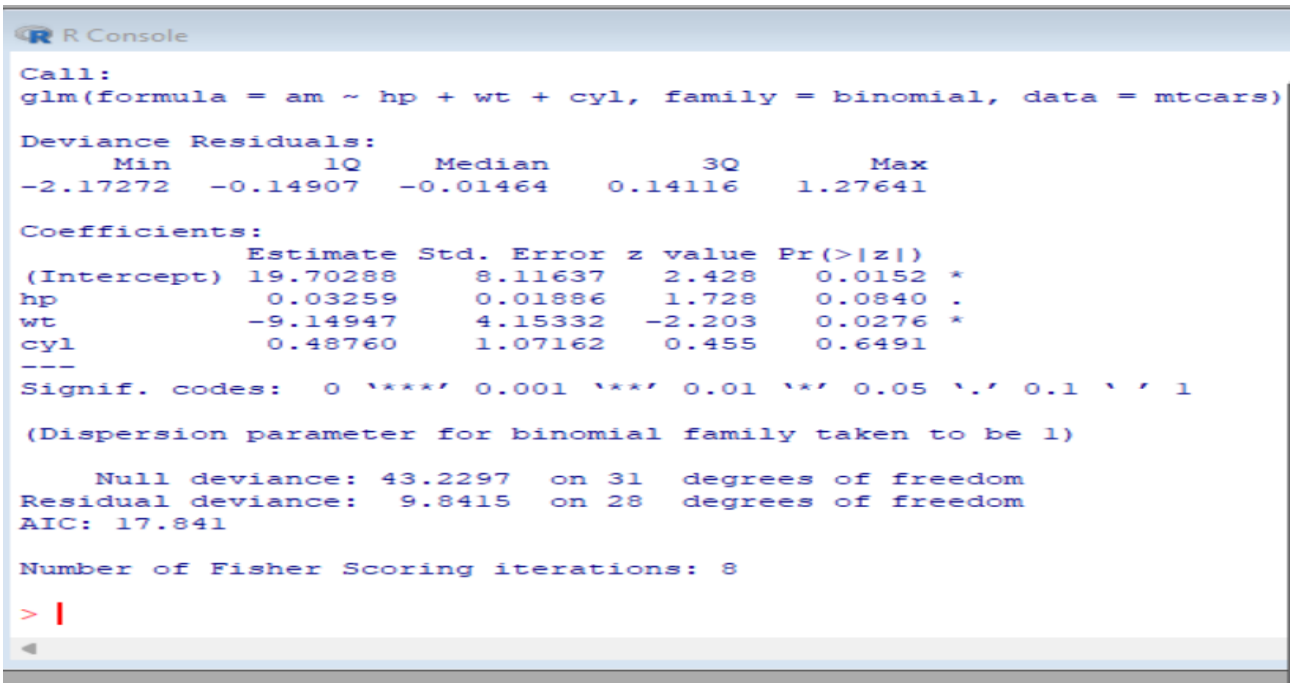
AIM:

PROCEDURE:

CODE:

```
# Load the mtcars dataset
data(mtcars)
# Create a logistic regression model
model <- glm(am ~ hp + wt + cyl, data = mtcars, family = binomial)
# Print the summary of the model
summary(model)
```

OUTPUT:



RESULT:

P. POISSON REGRESSION ANALYSIS IN R

1. Create a Poisson regression model using the in-built data set "warpbreaks" with information given below.

Aim :

Procedure:

Program & Output:

```
> input <- warpbreaks
> print(head(input))
```

OUTPUT:

```
breaks wool tension
1  26  A  L
2  30  A  L
3  54  A  L
4  25  A  L
5  70  A  L
6  52  A  L
```

```
> # Load the "warpbreaks" dataset
> data(warpbreaks)
>
> # Check the structure of the dataset
> str(warpbreaks)
```

OUTPUT:

```
'data.frame':   54 obs. of  3 variables:
 $ breaks : num  26 30 54 25 70 52 51 26 67 18 ...
 $ wool   : Factor w/ 2 levels "A","B": 1 1 1 1 1 1 1 1 1 1 ...
 $ tension: Factor w/ 3 levels "L","M","H": 1 1 1 1 1 1 1 1 1 2 ...
```

```
>
> # Fit a Poisson regression model
> model <- glm(breaks ~ wool + tension, data = warpbreaks, family = poisson)
>
> # Display the summary of the model
> summary(model)
```

OUTPUT:

```
Call:
glm(formula = breaks ~ wool + tension, family = poisson, data = warpbreaks)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.6871  -1.6503  -0.4269   1.1902   4.2616

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.69196    0.04541  81.302  < 2e-16 ***
woolB       -0.20599    0.05157  -3.994 6.49e-05 ***
tensionM    -0.32132    0.06027  -5.332 9.73e-08 ***
tensionH    -0.51849    0.06396  -8.107 5.21e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 297.37  on 53  degrees of freedom
Residual deviance: 210.39  on 50  degrees of freedom
AIC: 493.06

Number of Fisher Scoring iterations: 4
```

```
>
> # Make predictions on new data
> new_data <- data.frame(wool = "A", tension = "L")
> predict(model, newdata = new_data, type = "response")
```

OUTPUT:

```
       1
40.12354
```

Result: