**INDEX**

|  |  |
| --- | --- |
| **S.No** | **Title of Programs** |
| 1 | Write a program to generate prime number |
| 2 | Write a program to generate perfect number |
| 3 | Write a program to generate armstrong number |
| 4 | Write a program to generate fibonacci sequence |
| 5 | Write a program to generate product of prime |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |
| 30 |  |
| 31 |  |
| 32 |  |
| 33 |  |
| 34 |  |
| 35 |  |
| 36 |  |

1. **Generate prime number**

**Aim :**

To find and print all prime numbers less than a number entered by the user using R.

**Algorithm :**

**Step 1:** Start the process to find prime numbers below a given number.

**Step 2:** Open RStudio and write the program using readline() and while loops.

**Step 3:** Read an integer from the user and store it in a variable (e.g., num1).

**Step 4:** Set a loop variable (m = 2) and begin a loop that runs while m < num1.

**Step 5:** For each m, check if it is divisible by any number between 2 and m - 1. If not divisible, it is a prime number.

**Step 6:** Print the prime number, increase m by 1, and repeat the loop until all numbers below num1 are checked.

**Step 7:** End the program

**Program :**

num1 = as.integer(readline(prompt = "Enter A value: "))

m = 2

while (m < num1) {

flag = 1

num = m

i = 2

while (i < num) {

if ((num %% i) == 0) {

flag = 0

break

}

i = i + 1

}

if (flag == 1) {

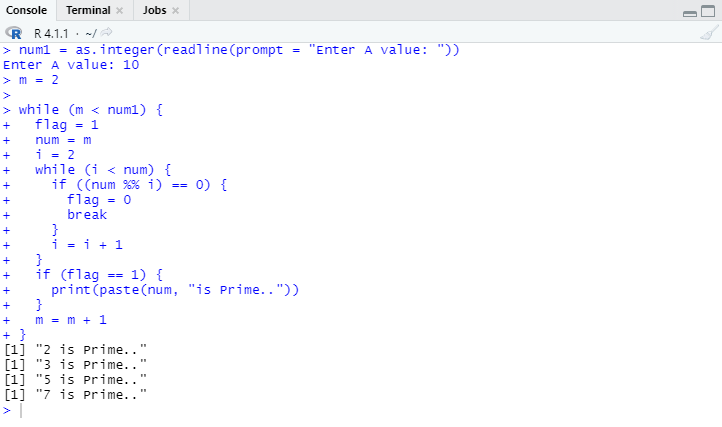
print(paste(num, "is Prime.."))

}

m = m + 1

}

**OUTPUT:**



**RESULT:**

Thus, our program has been successfully saved and executed.

1. **Generate perfect number**

**Aim :**

To find and print all perfect numbers less than a number entered by the user using R.

**Algorithm :**

**Step 1:** Start the process to check for perfect numbers below a given number.

**Step 2:** Open RStudio and write the program using readline(), while loops, and conditional statements.

**Step 3:** Read an integer input from the user and store it in NUM1.

**Step 4:** Initialize num = 1 and use a while loop to check each number less than NUM1.

**Step 5:** For each number, find the sum of its proper divisors (excluding the number itself) using another loop.

**Step 6:** If the sum of divisors equals the number, print it as a perfect number. Continue the loop until all numbers are checked

**Step 7:** End the program

**Program :**

NUM1 = as.integer(readline(prompt = "Enter No: "))

print(num)

num=1

while (num<NUM1) {

sum = 0

i = 1

while(i < num) {

if(num %% i == 0) {

sum = sum + i

}

i = i + 1

}

if(sum == num) {

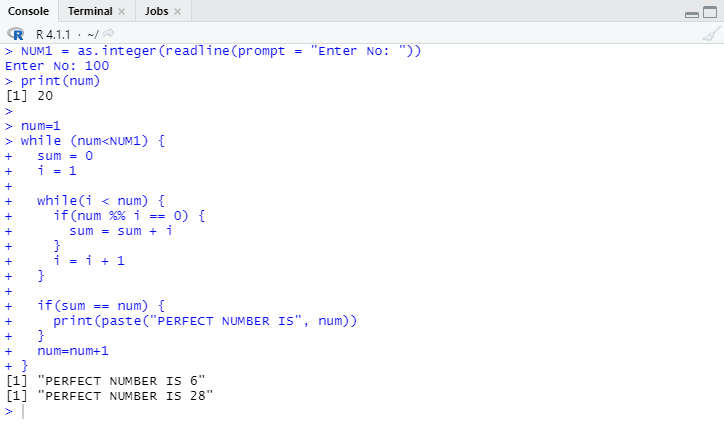
print(paste("PERFECT NUMBER IS", num))

}

num=num+1

}

**OUTPUT:**



**RESULT:**

Thus, our program has been successfully saved and executed.

1. **Generate armstrong number**

**Aim :**

To find and print all Armstrong numbers greater than or equal to 10 and less than a user-entered limit using R.

**Algorithm :**

**Step 1:** Start the process to identify Armstrong numbers less than a specified number.

**Step 2:** Open RStudio and write the program using readline() to take input and loops to calculate digit powers.

**Step 3:** Read an integer input from the user and store it in NUM1. Initialize num = 1.

**Step 4:** For each number from 1 to NUM1 - 1, count its digits using division by 10.

**Step 5:** Reset temp = num and calculate the sum of each digit raised to the power of the total number of digits.

**Step 6:** If the sum equals the original number and the number is ≥10, print it as an Armstrong number. Continue checking until all numbers are processed

**Step 7:** End the program

**Program :**

NUM1 = as.integer(readline(prompt = "Enter Limit: "))

num = 1

while (num < NUM1) {

temp = num

count = 0

while (temp > 0) {

count = count + 1

temp = temp %/% 10

}

temp = num

sum = 0

while (temp > 0) {

digit = temp %% 10

power = 1

i = 1

while (i <= count) {

power = power \* digit

i = i + 1

}

sum = sum + power

temp = temp %/% 10

}

if (sum == num && 10 <= num) {

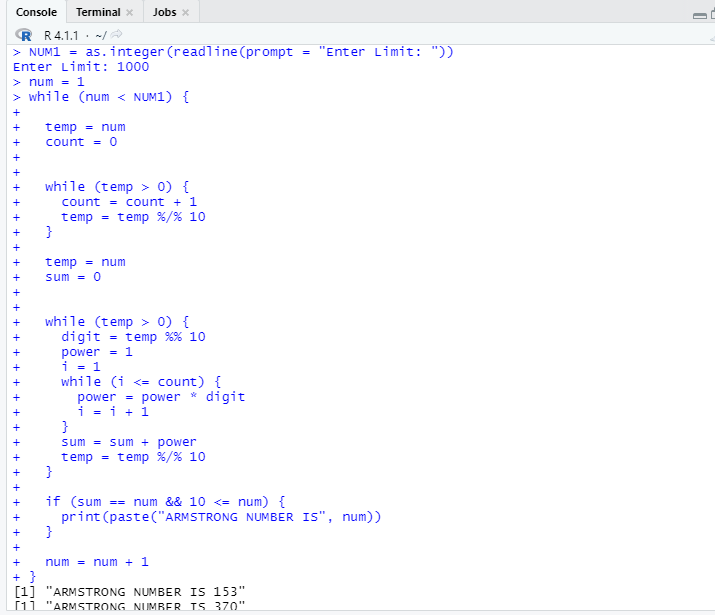
print(paste("ARMSTRONG NUMBER IS", num))

}

num = num + 1

}

**OUTPUT:**



**RESULT:**

Thus, our program has been successfully saved and executed.

1. **Generate fibonacci number**

**Aim :**

To generate and print the first ‘n’ terms of the Fibonacci sequence using R programming.

**Algorithm :**

**Step 1:** Start the process to generate a Fibonacci sequence for a given number of terms.

**Step 2:** Open RStudio and write the code using readline() to get input and a for loop to generate the sequence.

**Step 3:** Read an integer value n from the user representing the number of terms to generate.

**Step 4:** Initialize two variables a = 0 and b = 1, which represent the first two terms of the Fibonacci sequence.

**Step 5:** Use a for loop to iterate from 1 to n. In each iteration, print the current value of a.

**Step 6:** Calculate the next term by adding a and b, update a and b, and continue the loop. Stop when n terms are printed.

**Program :**

n = as.integer(readline(prompt = "Enter number of terms in Fibonacci sequence: "))

a = 0

b = 1

print("Fibonacci sequence:")

for (i in 1:n) {

print(a)

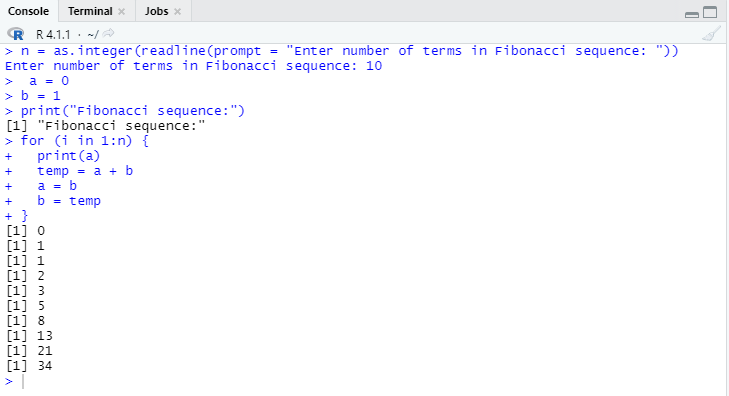
temp = a + b

a = b

b = temp

}

**OUTPUT:**



**RESULT:**

Thus, our program has been successfully saved and executed.

1. **Generate product of prime**

**Aim :**

To find and print all prime numbers up to a user-defined limit and calculate the product of those primes using R.

**Algorithm :**

**Step 1:** Start the process to identify prime numbers up to a given number and compute their product.

**Step 2:** Open RStudio and write the program using readline() to take input and while loops for processing.

**Step 3:** Read an integer input from the user and store it in NUM1. Initialize num = 2 and product = 1.

**Step 4:** Use a while loop to check each number from 2 to NUM1. For each number, set is\_prime = 1 and check divisibility using another loop.

**Step 5:** If a number has no divisors other than 1 and itself, it is prime. Multiply it with product and print the number.

**Step 6:** Repeat the process until all numbers up to NUM1 are checked. After the loop, print the final product of all prime numbers.

**Step 7:** End the program

**Program :**

NUM1 = as.integer(readline(prompt = "Enter Limit: "))

num = 2

product = 1

while (num <= NUM1) {

is\_prime = 1

i = 2

while (i <= num %/% 2) {

if (num %% i == 0) {

is\_prime = 0

break

}

i = i + 1

}

if (is\_prime == 1) {

product = product \* num

print(paste("PRIME NUMBER IS", num))

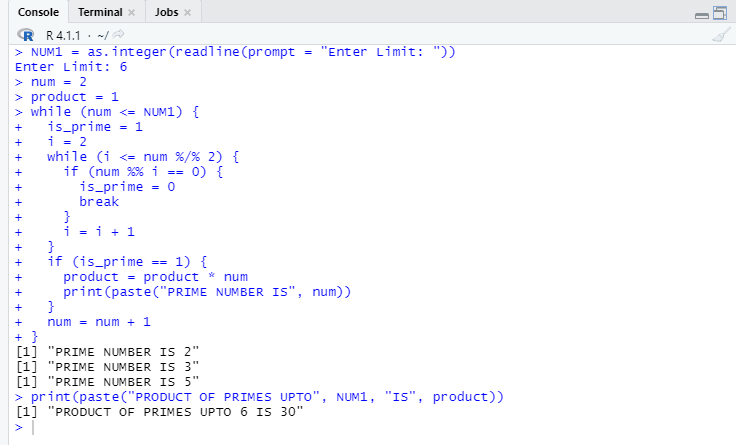
}

num = num + 1

}

print(paste("PRODUCT OF PRIMES UPTO", NUM1, "IS", product))

**OUTPUT:**



**RESULT:**

Thus, our program has been successfully saved and executed.

**6. Find Uniqueness of Data Using Array**

**Aim :**

To write an R program to find unique elements from a given array without using the factor() function.

**Algorithm :**

**Step 1:** Start the process to find the unique elements in an array.  
**Step 2:** Open RStudio and write the program using array(), while loop, and conditional checks.  
**Step 3:** Create an array (e.g., din) with repeated character elements.  
**Step 4:** Find the length of the array and store it in a variable (e.g., len).  
**Step 5:** Initialize an index variable i = 1.  
**Step 6:** Create an empty character vector a to store unique elements.  
**Step 7:** Begin a while loop that runs while i <= len.  
**Step 8:** Inside the loop, check if the current element din[i] is already present in a.If **not present**, add it to a.  
**Step 9:** Increment i by 1 and repeat Step 8 until all elements are processed.  
**Step 10:** Print the vector a which now contains only the unique elements.  
**Step 11:** End the program.

**Program :**

din<- array(c("E","E","W","R","T","W","R","T"))

len<-length(din)

i=1

a=character(0)

while(i<=len){

if(!(din[i] %in% a)){

a <- c(a, din[i])

}

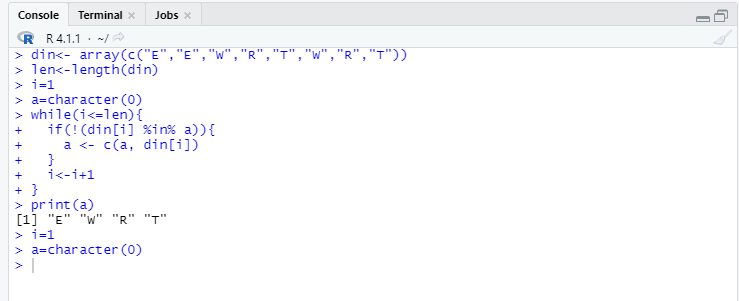
i<-i+1

}

print(a)

i=1

a=character(0)

**OUTPUT:**

**RESULT:**

This, our program has been successfully saved and executed.

**7. Mathematical Operation on Matrix**

**Aim :**

To write an R program to perform addition, subtraction, multiplication, and division on two matrices using user-defined functions.

**Algorithm :**

**Step 1:** Start the process to perform matrix operations.  
**Step 2:** Open RStudio and write the program using functions.  
**Step 3:** Create a function create\_matrix() that generates a 4×4 matrix with random numbers.  
**Step 4:** Define functions for:

* **Addition:** Add two matrices element-wise.
* **Subtraction:** Subtract two matrices element-wise.
* **Multiplication:** Multiply two matrices using %\*% (matrix product).
* **Division:** Divide two matrices element-wise.

**Step 5:** Generate two random matrices (matrix1 and matrix2) using create\_matrix().  
**Step 6:** Print both matrices.  
**Step 7:** Call each function to perform addition, subtraction, multiplication, and division.  
**Step 8:** Print the results of all operations.  
**Step 9:** End the program.

**Program :**

create\_matrix <- function() {

matrix(sample(1:10, 16, replace=TRUE), nrow=4, ncol=4)

}

matrix\_addition <- function(A, B) {

return(A + B)

}

matrix\_subtraction <- function(A, B) {

return(A - B)

}

matrix\_multiplication <- function(A, B) {

return(A %\*% B)

}

matrix\_division <- function(A, B) {

return(A / B)

}

matrix1 <- create\_matrix()

matrix2 <- create\_matrix()

cat("Matrix 1:\n")

print(matrix1)

cat("\nMatrix 2:\n")

print(matrix2)

cat("\nAddition:\n")

print(matrix\_addition(matrix1, matrix2))

cat("\nSubtraction:\n")

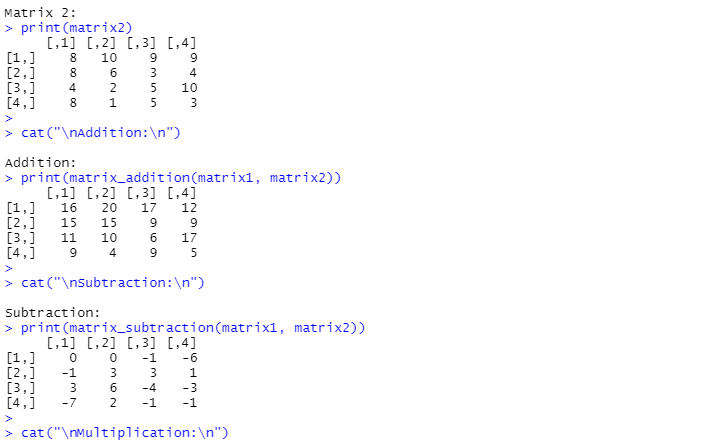
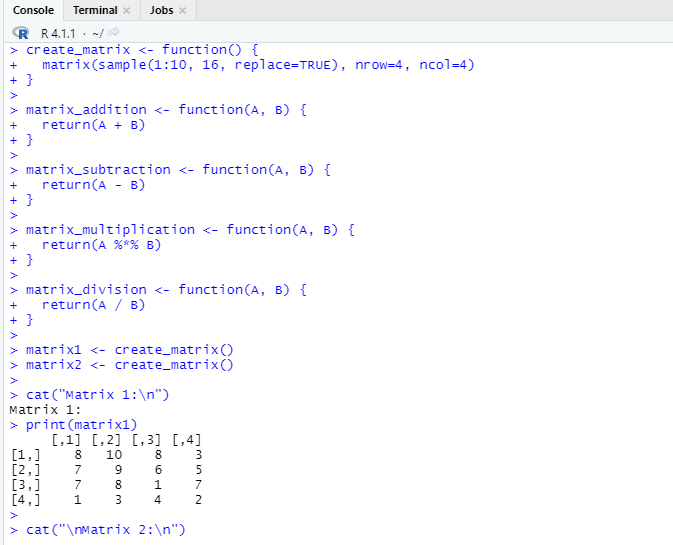
print(matrix\_subtraction(matrix1, matrix2))

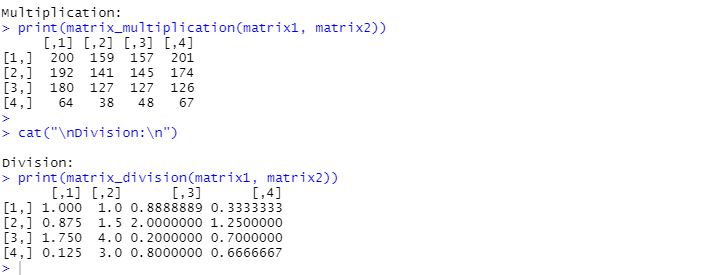
cat("\nMultiplication:\n")

print(matrix\_multiplication(matrix1, matrix2))

cat("\nDivision:\n")

print(matrix\_division(matrix1, matrix2))

**OUTPUT:**

****

**RESULT:**

This, our program has been successfully saved and executed.

**8. Converted As Data Frame**

**Aim :**

To write an R program to Three vectors of name,age,gender. It should be converted as data frame. Provide atleast 10 dataset. Filter data as follows:

a. Age >25

b. Age in between 10 40

c. Age >25 and gender= female

d. Age in between 10 and 45 and gender =male

e. Name starting with a character "a"

.

**Algorithm :**

**Step 1:** Start the process.

**Step 2:** Set the folder where your CSV file is stored using setwd().

setwd("D:/R lab")

**Step 3:** Read the CSV file into a variable using read.csv().

data = read.csv("Details.csv")

**Step 4** Use the subset() function to extract rows where AGE is greater than 25.

res\_age25 = subset(data, AGE > 25) print(res\_age25)

**Step 5:** Use logical operators (&) to filter AGE between 10 and 40.

res\_agebetween = subset(data, AGE > 10 & AGE < 40)

print(res\_agebetween)

**Step 6:** Apply two conditions: AGE and GENDER.

res\_agegender = subset(data, AGE > 25 & GENDER == "girl")

print(res\_agegender)

**Step 7:** Combine multiple conditions with &.

res\_agegender = subset(data, AGE > 20 & AGE < 40 & GENDER == "male")

print(res\_agegender)

**step 8**: Use grepl() with regular expressions to match names starting with "a" (case- sensitive).

res\_a = subset(data, grepl("^a", NAME, ignore.case = TRUE))

print(res\_a)

step 9 : End the process

**Program :**

getwd()

setwd("D:/R lab")

getwd()

data=read.csv("Details.csv")

print(data)

res\_age25=subset(data,AGE>25)

print(res\_age25)

res\_agebetween=subset(data,AGE>10 & AGE<40)

print(res\_agebetween)

res\_agegender=subset(data,AGE>25 & GENDER == "girl")

print(res\_agegender)

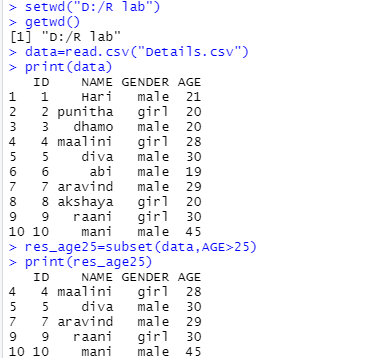
res\_agegender=subset(data,AGE>20 & AGE<40 & GENDER == "male")

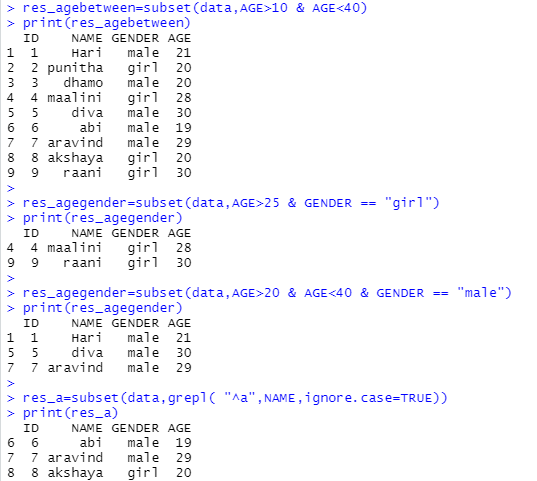
print(res\_agegender)

res\_a=subset(data,grepl( "^a",NAME,ignore.case=TRUE))

print(res\_a)

**OUTPUT:**





**RESULT:**

Thus, our program has been successfully saved and executed.

**9.Import CSV File Into DataFrame to Filter the data**

**Aim :**

To write a R program with CSV file having empid,name,age,gender, salary,basic, DA.Provide atleast 20 datasets.

Read the following file and filter the data as follows:

a. Genderwise

b Age>40 and gender=male

c Salary >600 for different genders mentioned in the csv file

d. find out the difference between salary and Basic+DA

e. Salary >600 and Basic >300 and DA < 200

**Algorithm :**

**Step 1:** Start the process to identify Armstrong numbers less than a specified number.

**Step 2:** Set the folder where your file gender.csv is saved.

setwd("D:/R lab")

**Step 3** Read the data from the CSV file

data <- read.csv("gender.csv")

**Step 4:** Find and print only rows where Gender is "Female".

retval <- subset(data, data$Gender == "Female")

print(retval)

**Step 5:** Find and print only rows where Gender is "Male".

retval <- subset(data, data$Gender == "Male")

print(retval)

**Step 6:** Filter records for male employees older than 40.

retval <- subset(data, data$Age > 40 & data$Gender == "Male")

print(retval)

**Step 7:** Print rows where Salary is more than 600.

retval <- subset(data, data$Salary > 600)

print(retval)

### Step 8: Show employees with all these:

Salary > 600

Basic > 300

DA < 200

retval <- subset(data, data$Salary > 600 & data$Basic > 300 & data$DA < 200)

print(retval)

**Step 9:** Subtract (Basic + DA) from Salary and print the result.

retval <- data$Salary - (data$Basic + data$DA)

print(retval)

**Program :**

getwd()

setwd("D:/R lab")

getwd()

data <- read.csv("gender.csv")

print(data)

retval <- subset(data,data$Gender=="Female")

print(retval)

retval <- subset(data,data$Gender=="Male")

print(retval)

retval <- subset(data,data$Age>40 & data$Gender=="Male")

print(retval)

retval <- subset(data,data$Salary>600)

print(retval)

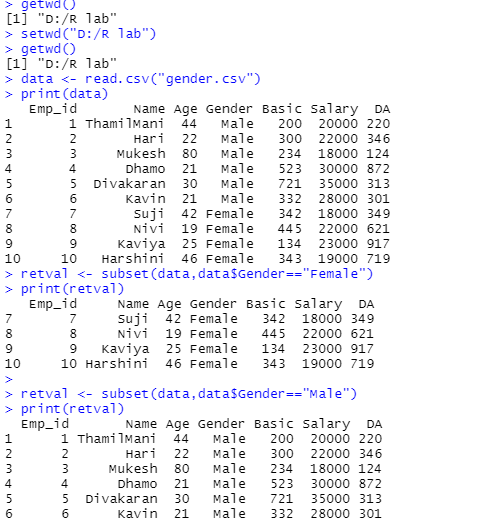
retval <- subset(data,data$Salary>600 & data$Basic>300 & data$DA<200)

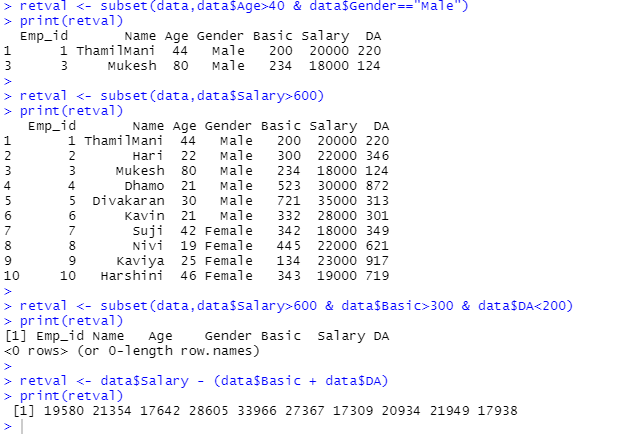
print(retval)

retval <- data$Salary - (data$Basic + data$DA)

print(retval)

**OUTPUT:**





**RESULT:**

Thus, our program has been successfully saved and executed

**10. JSON FILE HANDLING USING DATASETS.**

**Aim :**

To write a R Program to read employee data from a JSON file and perform various filters based on gender, age, salary, and calculate differences between salary and the sum of Basic and DA.

**Algorithm :**

**Step 1 :** start the process

**Step 2:** Set the working directory to the location of your JSON file using setwd().

**Step 3:** Load the jsonlite library to handle JSON file reading.

**Step 4:** Read the JSON file (gender.json) using fromJSON() and store it in a variable (e.g., data).

**Step 5 :** Display the full dataset using print (data) to verify successful import.

**Step 6 :** Filter records by Gender using subset () — one for "Male" and another for "Female".

**Step 7 : Filter records where Age > 40 and Gender == "Male"`** using:

**Step 8** : Filter records where Salary > 600, and also combine conditions like Salary > 600 & Basic > 300 & DA < 200

**Step 9** : Calculate and print the difference between Salary and (Basic + DA) using:

**Step 10 :** End the program

**Program :**

getwd()

setwd("E:/Practial")

getwd()

library(jsonlite)

#Tools -> Install Packages

data <- fromJSON("gender.json")

print(data)

retval <- subset(data,data$Gender=="Female")

print(retval)

retval <- subset(data,data$Gender=="Male")

print(retval)

retval <- subset(data,data$Age>40 & data$Gender=="Male")

print(retval)

retval <- subset(data,data$Salary>600)

print(retval)

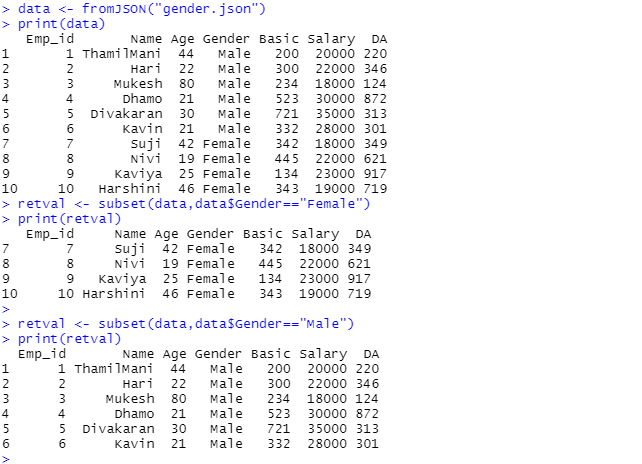
retval <- subset(data,data$Salary>600 & data$Basic>300 & data$DA<200)

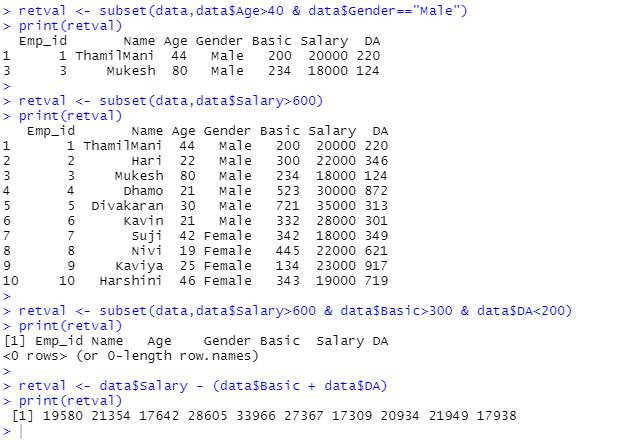
print(retval)

retval <- data$Salary - (data$Basic + data$DA)

print(retval)

**OUTPUT:**





**RESULT:**

Thus the given program was verified and executed successfully.

**11. XML FILE HANDLING USING DATASETS.**

**Aim :**

To write a R Program read employee data from an XML file and perform filtering based on gender, age, salary, and compute the difference between salary and the sum of Basic and DA.

**Algorithm :**

**Step 1 :** Start the Program

**Step 2:** Set the working directory to the location of your XML file using setwd().

**Step 3:** Load the XML library to handle XML file reading..

**Step 4:** Read the XML file (e.g., gender.xml) using xmlToDataFrame() and store it in a variable (e.g., data).

**Step 5** Display the full dataset using print(data) to verify successful import.

**Step 6:** Filter records by Gender using subset() — one for "Male" and another for "Female".

**Step 7:** Filter records where Age > 40 and Gender == "Male" using subset():

**Step 8**: Filter records where Salary > 600, and also combine conditions like Salary > 600 & Basic > 300 & DA < 200.

**Step 9**: Calculate and print the difference between Salary and (Basic + DA) using a new column

**Step 10:** End the program

**Program :**

library(xml2)

getwd()

setwd("E:/Practial ")

getwd()

install.packages("xml2")

install.packages("XML")

#Tools -> Install Packages

library(XML)

data\_xml <- xmlParse("gender.xml")

data <- xmlToDataFrame(data\_xml)

print (data)

retval <- subset(data,data$Gender=="Female")

print (retval)

retval <- subset(data,data$Gender=="Male")

print (retval)

retval <- subset(data,data$Age>40 & data$Gender=="Male")

print (retval)

retval <- subset(data,data$Salary>600)

print (retval)

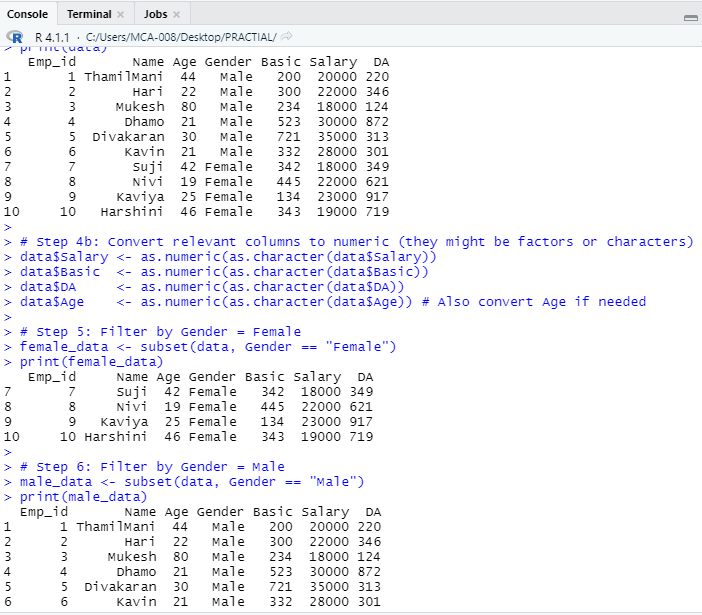
retval <- subset(data,data$Salary>600 & data$Basic>300 & data$DA<200)

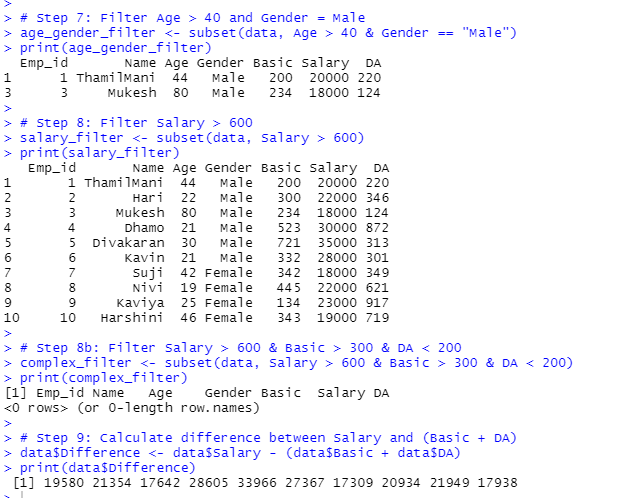
print (retval)

retval <- data$Salary - (data$Basic + data$DA)

print (retval)

**OUTPUT:**

****

****

**RESULT:**

Thus the given program was verified and executed successfully

**12. Import XLSX File Into DataFrame to Filter the data**

**Aim**

To write a R program with XLSX file having empid,name,age,gender, salary,basic, DA.Provide atleast 20 datasets.

Read the following file and filter the data as follows:

a. Genderwise

b Age>40 and gender=male

c Salary >600 for different genders mentioned in the xlsx file

d. find out the difference between salary and Basic+DA

e. Salary >600 and Basic >300 and DA < 200

**Algorithm**

**Step 1:** Start the process to read and analyze Excel data in R.

**Step 2:** Set and check the working directory using setwd() and getwd().

**Step 3:** Install and load the readxl package (installation is required only once).

**Step 4:** Read the Excel file using read\_excel() and assign it to a variable (e.g., gender\_data).

**Step 5:** Print the dataset to verify the data has been read correctly.

**Step 6**: Use the subset() function to extract and print:

All records where Gender = "Female".

All records where Gender = "Male".

All male employees older than 40 years.

All employees with Salary greater than 600.

All employees with Salary > 600, Basic > 300, and DA < 200.

**Step 7:** Calculate and print the difference between Salary and the sum of (Basic + DA).

**Step 8:** End the program.

**Program :**

getwd()

setwd("D:/ThamilMani/Learning-Programming-/R Programming/12. XLSX File Handling")

getwd()

install.packages("readxl")

library(readxl)

#Tools -> Install Packages

data <- read\_excel("gender.xlsx")

print(data)

retval <- subset(data,data$Gender=="Female")

print(retval)

retval <- subset(data,data$Gender=="Male")

print(retval)

retval <- subset(data,data$Age>40 & data$Gender=="Male")

print(retval)

retval <- subset(data,data$Salary>600)

print(retval)

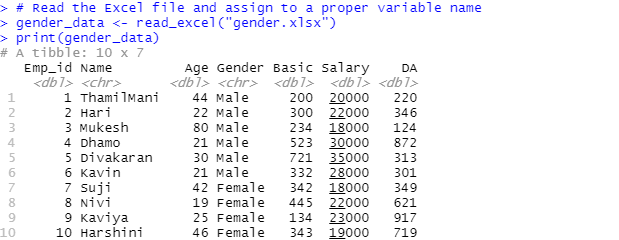
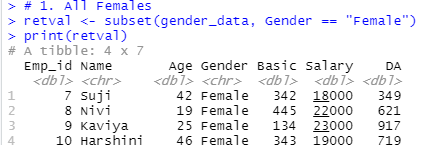
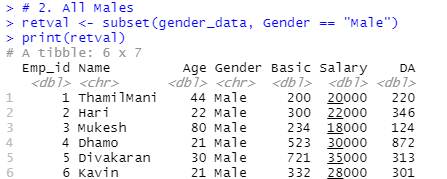
retval <- subset(data,data$Salary>600 & data$Basic>300 & data$DA<200)

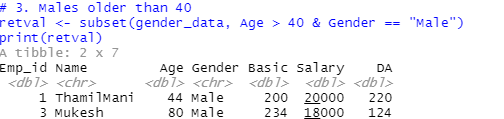
print(retval)

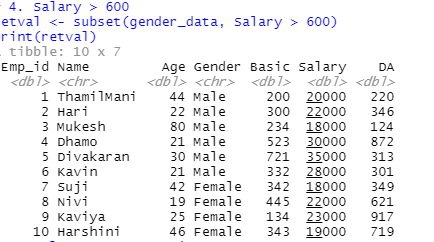
retval <- data$Salary - (data$Basic + data$DA)

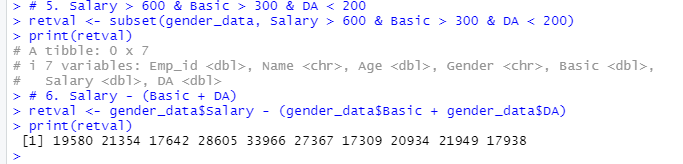
print(retval)

**OUTPUT:**









**RESULT**

Thus, our program has been successfully saved and executed.

**13. Merge XML and Json into DataFrame to Filter the data**

**Aim**

To write a R program with read employee data from an XLSX file and a JSON file, merge them into a single DataFrame, and apply various filters to analyze employee details based on gender, age, and salary-related conditions.

Read the following file and filter the data as follows:

a. Genderwise

b Age>40 and gender=male

c Salary >600 for different genders mentioned in the xlsx file

d. find out the difference between salary and Basic+DA

e. Salary >600 and Basic >300 and DA < 200

**Algorithm**

**Step 1:** Start the process to handle XML and JSON file conversion in R.

**Step 2:** Open RStudio and write the program using required packages (xml2, jsonlite).

**Step 3:** Read the XML file (e.g., gender.xml) using read\_xml().

**Step 4:** Extract all records from the XML using xml\_find\_all().

**Step 5:** Convert the extracted XML nodes into a data frame with proper column names and values.

**Step 6:** Convert necessary columns (Emp\_id, Age, Basic, Salary, DA) into integer type for further processing.

**Step 7:** Convert the data frame into JSON format using toJSON() and save it into a JSON file (e.g., gender.json).

**Step 8:** Read the JSON file back into R using fromJSON().

**Step 9:** Perform filtering and subsetting operations (e.g., Female employees, Male employees, Salary > 600, etc.).

**Step 10:** Perform calculations such as salary difference (Salary – (Basic + DA)).

**Step 11:** Print the results.

**Step 12:** End the program.

**Program :**

getwd()

setwd("D:/ThamilMani/Learning-Programming-/R Programming/13. XML File To JSON File")

getwd()

install.packages("xml2")

install.packages("jsonlite")

library(xml2)

library(jsonlite)

doc <- read\_xml("gender.xml")

records <- xml\_find\_all(doc, ".//Record")

data <- as.data.frame(

t(sapply(records, function(node) {

setNames(xml\_text(xml\_children(node)), xml\_name(xml\_children(node)))

})),

stringsAsFactors = FALSE

)

data$Emp\_id <- as.integer(data$Emp\_id)

data$Age <- as.integer(data$Age)

data$Basic <- as.integer(data$Basic)

data$Salary <- as.integer(data$Salary)

data$DA <- as.integer(data$DA)

json\_text <- toJSON(data, pretty = TRUE, auto\_unbox = TRUE)

write(json\_text, file = "gender.json")

data <- fromJSON("gender.json")

print(data)

retval <- subset(data, Gender == "Female")

print(retval)

retval <- subset(data, Gender == "Male")

print(retval)

retval <- subset(data, Age > 40 & Gender == "Male")

print(retval)

retval <- subset(data, Salary > 600)

print(retval)

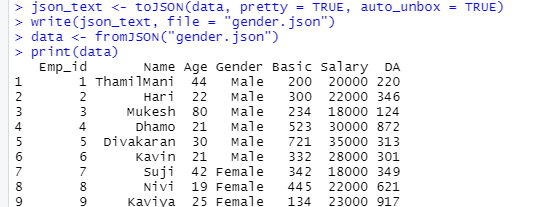
retval <- subset(data, Salary > 600 & Basic > 300 & DA < 200)

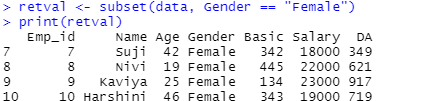
print(retval)

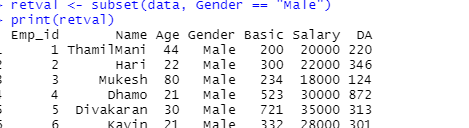
retval <- data$Salary - (data$Basic + data$DA)

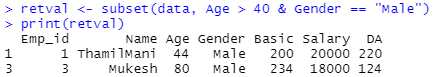
print(retval)

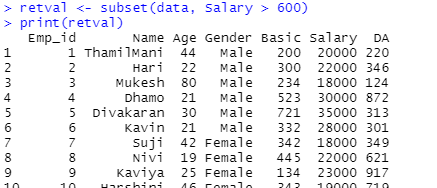
**OUTPUT:**

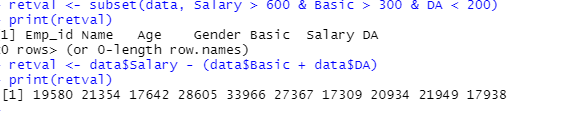
****











**RESULT**

Thus, our program has been successfully saved and executed.

**14. Generate PieChart**

**AIM:**

To write a R Program create and display a bar chart showing the monthly revenue distribution.

**ALGORITHM:**

**Step 1:** Start the process to visualize monthly revenue using a bar chart.

**Step 2:** Open RStudio and write the program using vectors and the barplot() function.

**Step 3:** Create a vector months to store the month names (e.g., months <- c("Mar", "Apr", "May", "Jun", "Jul")).

**Step 4:** Create another vector revenue to store revenue values corresponding to each month (e.g., revenue <- c(7, 8, 9, 10, 11)).

**Step 5:** Use the barplot() function to display the revenue values as vertical bars.

**Step 6:** Add labels for the x-axis (xlab = "Month") and y-axis (ylab = "Revenue").

**Step 7:** Add the main title (main = "Revenue Chart") to the bar chart.

**Step 8:** Enhance the chart with colors (e.g., col = "blue") and bar borders (e.g., border = "red").

**Step 9:** Run the program and view the bar chart output.

**Step 10:** End the process.

**PROGRAM:**

months <- c("Mar", "Apr", "May", "Jun", "Jul")

revenue <- c(7, 8, 9, 10, 11)

pie(revenue,

labels = paste(months, "\n", revenue, " units"),

main = "Monthly Revenue Distribution",

col = rainbow(length(months)),

border = "white")

x <- c(21, 62, 10, 53, 76)

labels <- c("London", "New York", "Singapore", "Mumbai", "Chennai")

library(plotrix)

pie3D(x,

labels = labels,

explode = 0.1,

main = "3D Pie Chart of Countries")

legend("topright",

labels,

cex = 0.6,

fill = rainbow(length(x)))

#Work with CSV Files

setwd("D:/24PCA014/Practical/Pie chart")

df <- read.csv("Combined.csv")

print(df)

v <- df[, c("Basic")]

lbl <- v # using values as labels

pie(v,labels = lbl,main="Basic",col=rainbow(length(v)))

legend("topleft",

legend = v,

cex = 0.7,

fill = rainbow(length(v)))

pie(v,

labels = lbl,

main = "Basic Pie Chart",

col = rainbow(length(v)))

legend("topleft",

legend = v,

cex = 0.7,

fill = rainbow(length(v)))

library(plotrix)

pie3D(v,

labels = lbl,

explode = 0.1,

main = "3D Pie Chart - Basic",

col = rainbow(length(v)))

legend("topright",

legend = lbl,

cex = 0.5,

fill = rainbow(length(v)))

v <- df[, c("Salary")]

print(v)

lbl <- c("1","2","3","4","5","6","7","8")

pie(v,

labels = lbl,

main = "Salary Pie Chart",

col = rainbow(length(v)))

pie3D(v,

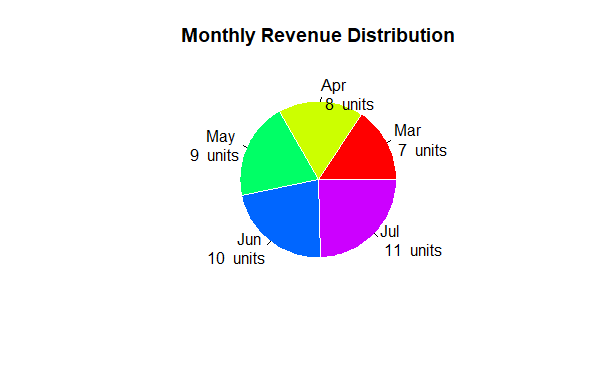
labels = lbl,

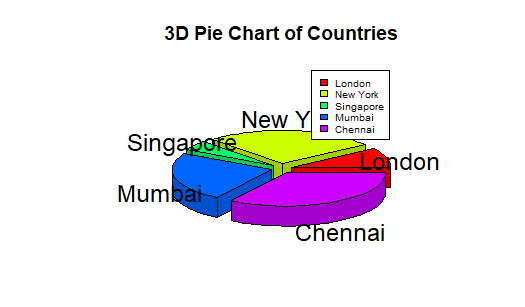
explode = 0.1,

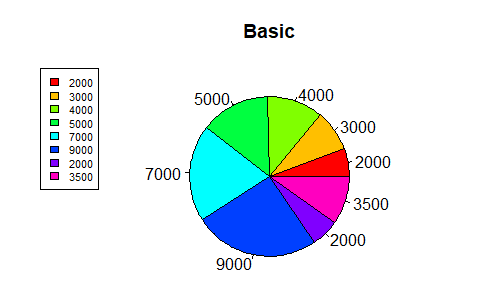
main = "3D Pie Chart - Salary",

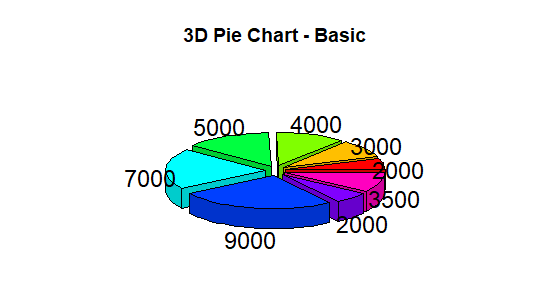
col = rainbow(length(v)))

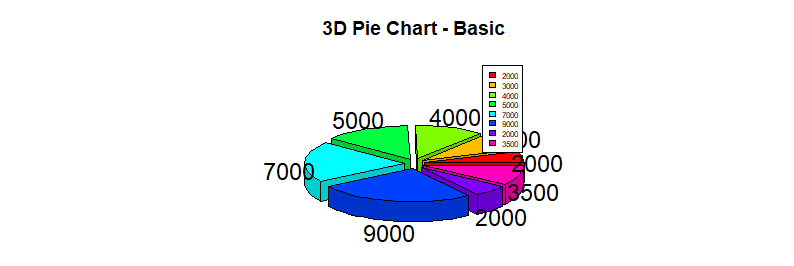
**OUTPUT:**

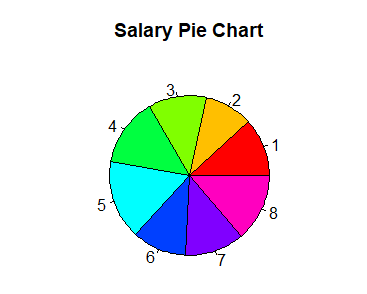


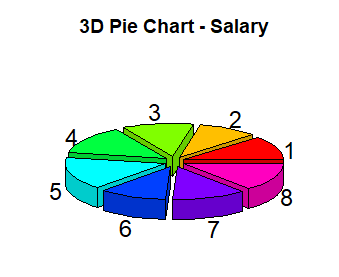












**RESULT**

Thus, our program has been successfully saved and executed.

**15. Generate  BarChart**

**Aim:**

To write a R Program to create a Bar Chart in R for displaying revenue values of different months.

**Algorithm:**

**Step 1:** Start the process to create a bar chart using R.

**Step 2:** Open RStudio and write the program.

**Step 3:** Create a numeric vector containing the revenue values.

**Step 4:** Create another vector containing the month names as labels.

**Step 5:** Use the barplot() function with the following arguments:

height → revenue values

names.arg → months

xlab → label for x-axis

ylab → label for y-axis

main → title of the chart

col → bar color

border → border color

**Step 6:** Execute the program to display the bar chart.

**Step 7:** End the program.

**Program :**

h<-c(7,8,9,10,11)

#png(file="bar")

barplot(h)

months <- c("Mar", "Apr", "May", "Jun", "Jul")

revenue <- c(7, 8, 9, 10, 11)

barplot(revenue,

names.arg = months,

xlab = "Month",

ylab = "Revenue",

col = "blue",

main = "Revenue Chart",

border = "red")

#group and stacked Bar chart

colors<-c("green","orange","brown")

months<-c("Mar","Apr","Jun","Jul")

regions<-c("East","west","North")

values<-matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11),nrow=3,ncol=5,byrow=TRUE)

barplot(values,main="Total Forecats",names.arg=months,xlab="months",ylab="Forecast",col=colors)

legend("topleft",regions,cex=1.2,fill=colors)

#CSV File

setwd("D:/24PCA014/Practical/Barplot")

df<-read.csv("Combined.csv")

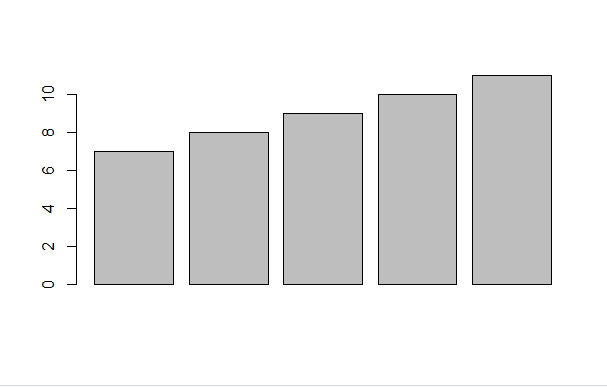
print(df)

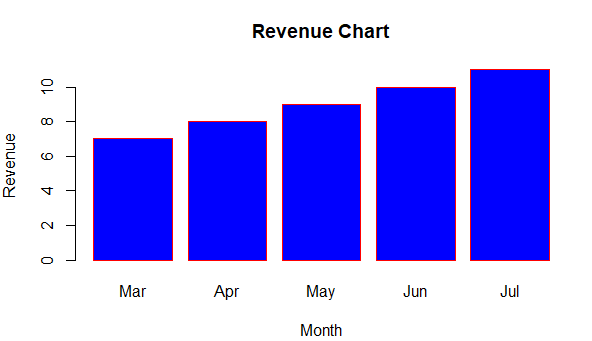
v<-df[,c("Basic")]

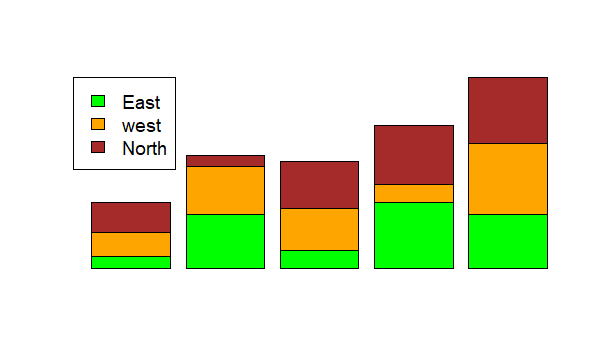
print(v)

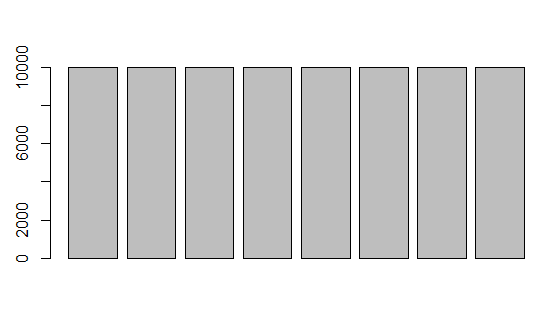
barplot(v)

**OUTPUT:**









**RESULT**

Thus, our program has been successfully saved and executed.

**16. Box Plot**

**AIM:**

To write a R Program to create and analyze boxplots in R using numeric vectors, built-in datasets, and CSV files.

**ALGORITHM:**

**Step 1:** Start the process to visualize the distribution of data using boxplots.  
**Step 2:** Open RStudio and write the program using vectors, built-in datasets, and CSV files.  
**Step 3:** Create numeric vectors and apply the summary() function to calculate basic statistics such as min, max, median, and quartiles.  
**Step 4:** Generate boxplots for the numeric vectors (both vertical and horizontal) to study spread and outliers.  
**Step 5:** Use the mtcars dataset to plot boxplots of mpg and hp grouped by the number of cylinders.  
**Step 6:** Import a CSV file, extract the required columns (e.g., Salary and Basic), and create boxplots for analyzing the relationship between variables.  
**Step 7:** Customize the boxplots with labels, titles, colors, notches, and widths, and interpret the results for meaningful insights.

**Step 8:** Stop the process

**Program:**

x <- c(10,20,30,40,50)

summary(x)

boxplot(x,horizontal = FALSE)

boxplot(x,horizontal = TRUE)

x<- c(10,11,14,15,120,12,34,54,65,24,67,230)

boxplot(x,horizontal = FALSE)

print(mean(x))

mtcars

input <- mtcars[, c("mpg", "cyl")]

print(input)

boxplot(mpg~cyl,data=mtcars,xlab = "Number of Cylinders ",ylab = "Milege Data",main = "Milege Data")

boxplot(hp~cyl,data=mtcars,xlab = "Number of Cylinders ",ylab = "Horse Power",main = "Power Data")

boxplot(mpg ~ cyl,

data = mtcars,

xlab = "No. of Cylinders",

ylab = "Miles Per Gallon",

main = "Mileage Data",

notch = TRUE,

varwidth = TRUE,

col = c("green", "yellow", "purple"),

names = c("4", "6", "8"))

setwd("D:/24PCA014/Practical/Box Plot")

df <- read.csv("combined.csv")

print(df)

# Subset only Salary and Basic

v <- df[, c("Salary", "Basic")]

print(v)

# Boxplot with correct case

boxplot(Salary ~ Basic,

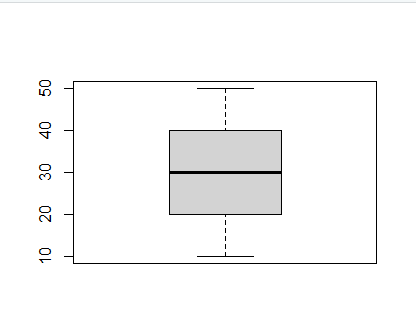
data = v,

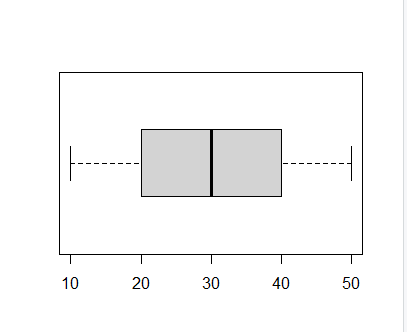
xlab = "Basic",

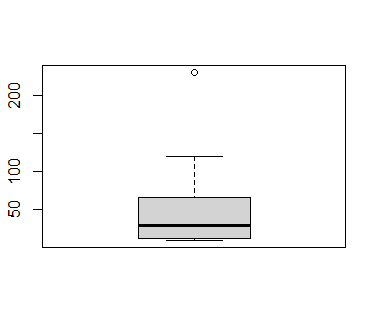
ylab = "Salary",

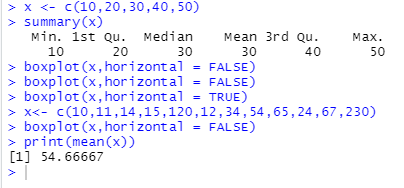
main = "Salary Chart",

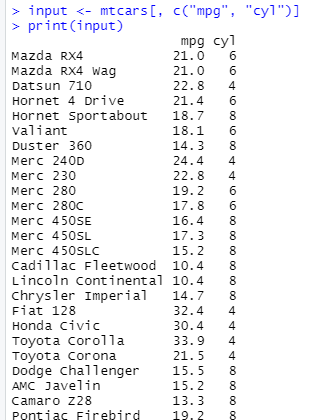
col = "lightblue")

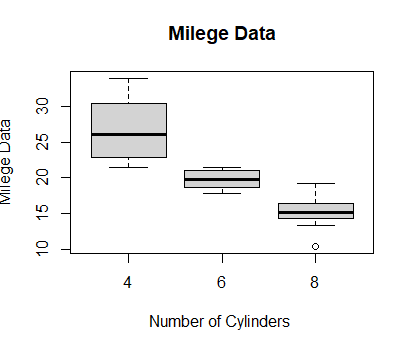
**OUTPUT:  
**

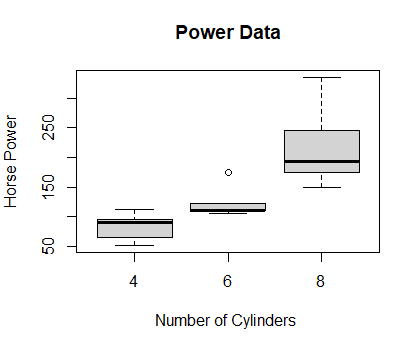
****

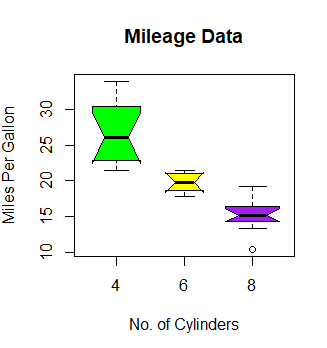
****

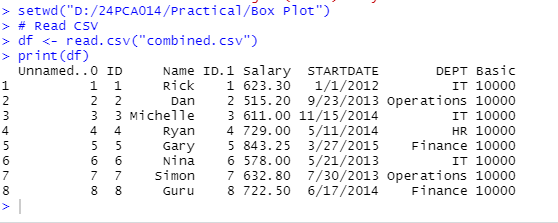
****

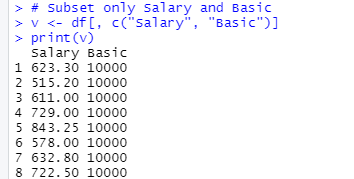
****

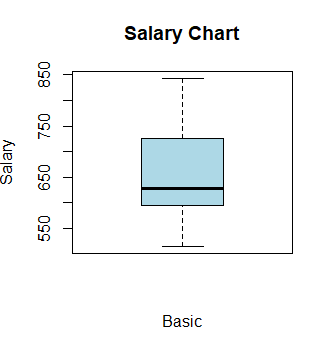
****

****

****

****

****

****

**RESULT:**

Thus, our program has been successfully saved and executed.

**17. Line Chart**

**AIM:**

To write a R Program to read data from a CSV file and plot line charts in R for visualizing and comparing **Salary** and **Basic** values.

**ALGORITHM:**

**Step 1:** Start the process to plot a line chart.  
**Step 2:** Open RStudio and set the working directory to the folder containing the CSV file.  
**Step 3:** Read the CSV file into a data frame using read.csv().  
**Step 4:** Extract the required columns (Salary and Basic) from the data frame.  
**Step 5:** Plot a line chart for **Basic** values using the plot() function.  
**Step 6:** Plot a line chart for **Salary** values using the plot() function.  
**Step 7:** Plot both **Salary** and **Basic** values in the same graph using plot() and lines() functions, and add a legend.  
**Step 8:** End the program.

**PROGRAM:**

v<-c(7,12,28,3,4,1)

print(v)

plot(v,type="o")

plot(v,tyle="l")

plot(c)

plot(v,type="o",col="red",xlab="Month",ylab="Rainfall",main="Rainfall Chart")

#multiple Lines

v<-c(7,12,28,3,41)

t<-c(14,18,7,6,19,3)

b<-c(15,7,18,19,13)

plot(v,type="o",col="red",xlab="Month",ylab = "Rainfall",main = "Rainfall Chart")

lines(t,type="o",col="green")

lines(b,type="o",col="blue")

colors<-c("green","red","blue")

regions<-c("2005","2010","2020")

legend("topleft",regions,cex=0.2,fill=colors)

t<-0:10

z=exp(t/2)

print(t)

print(z)

plot(t/2,type="l",col="green",lwd=5,xlab="time",ylab="Concentration")

x=-10:110

y=x\*x

plot(x,y,type="o",col="red",lwd=5,xlab="X--Axis",ylab = "Y--Axis")

# Set working directory

setwd("D:/24PCA014/Practical/Line Chart")

df <- read.csv("combined.csv")

print(df)

# Extract Basic and Salary

basic <- df$Basic

salary <- df$Salary

# Plot Basic

plot(basic, type="o", col="red",

xlab="Person", ylab="Basic",

main="Basic Chart")

# Plot Salary

plot(salary, type="o", col="blue",

xlab="Person", ylab="Salary",

main="Salary Chart")

# Plot both Basic and Salary together

plot(basic, type="o", col="red",

xlab="Person", ylab="Value",

main="Salary vs Basic")

lines(salary, type="o", col="green")

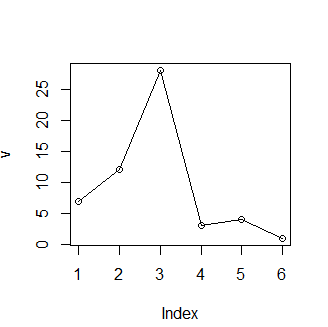
# Add legend

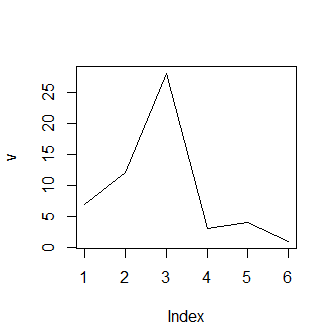
legend("topleft", c("Basic", "Salary"),

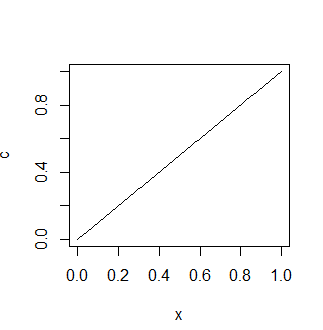
col=c("red","green"),

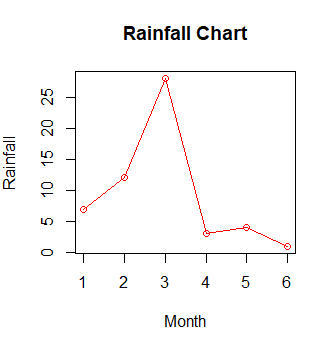
lty=1, pch=1, cex=0.8)

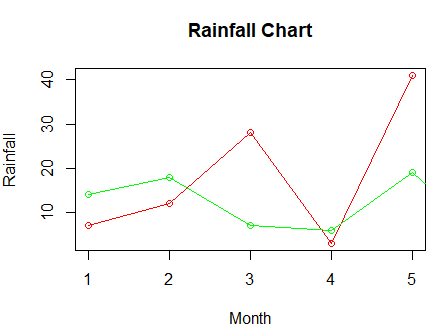
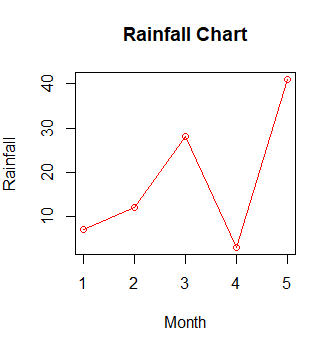
**OUTPUT:**

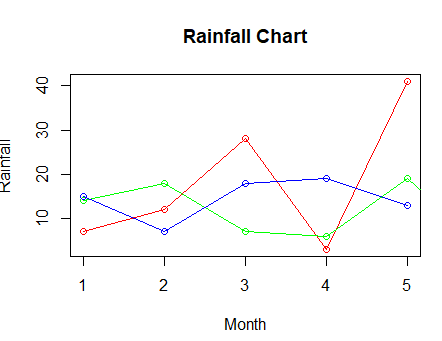
****

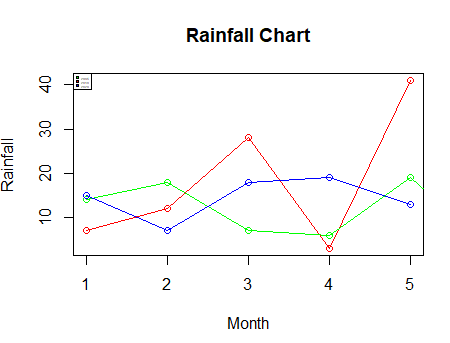
****

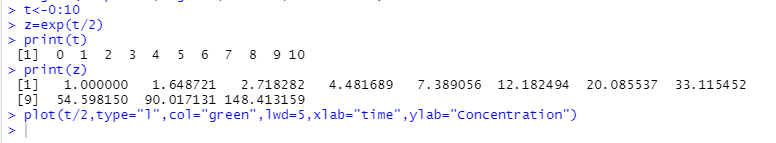
****

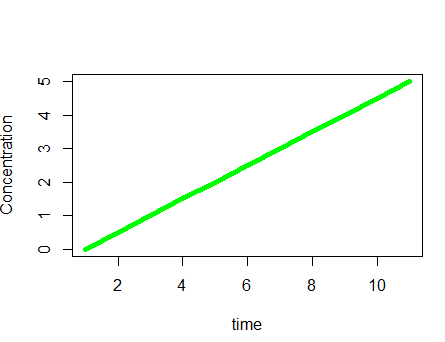
****

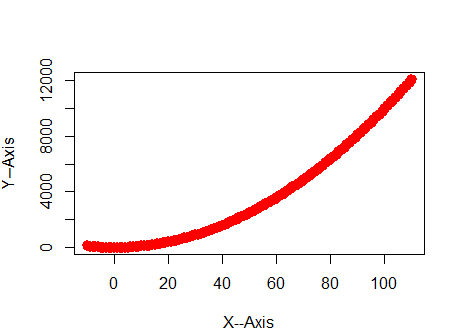
****

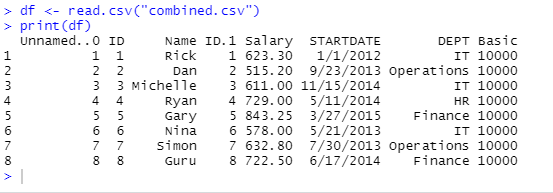
****

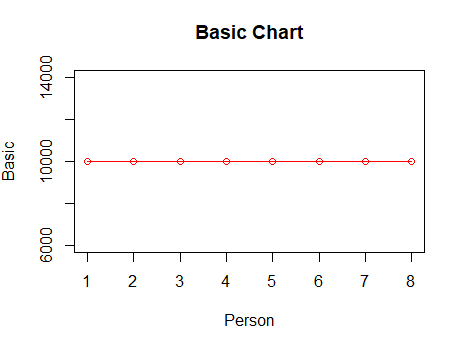
****

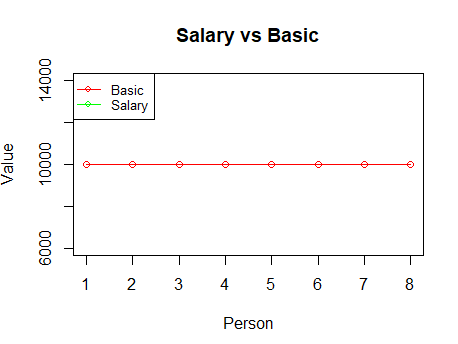
****

****

****

****

****

****

**RESULT:**

Thus, our program has been successfully saved and executed.

**18.Scatter Plot**

**AIM:**

To write a R Program visualize the relationship between multiple variables using scatter plots and scatter plot matrices.

**ALGORITHM:**

**Step 1:** Start the process to create scatter plots in R.  
**Step 2:** Open RStudio and load the built-in dataset (mtcars) into a variable.  
**Step 3:** Select the required columns (e.g., wt, mpg, disp, hp) and use the pairs() function to generate scatter plot matrices.  
**Step 4:** Set the working directory and read external data from a CSV file using read.csv().  
**Step 5:** Extract the required columns (e.g., Salary and Basic) into a new data frame.  
**Step 6:** Use the pairs() function again to create scatter plot matrices for the CSV data.  
**Step 7:** End the program.

**PROGRAM:**

mtcars

input<-mtcars[,c("wt","mpg","disp","cyl")]

pairs(~wt+mpg+disp,data=mtcars,mian="SactterPlot Matrix")

pairs(~wt+mpg+disp+hp,data=mtcars,main="Sactter Plot Matrix")

#work with CSV File

setwd("D:/24PCA014/Practical/Scatter Plot")

df<-read.csv("combined.csv")

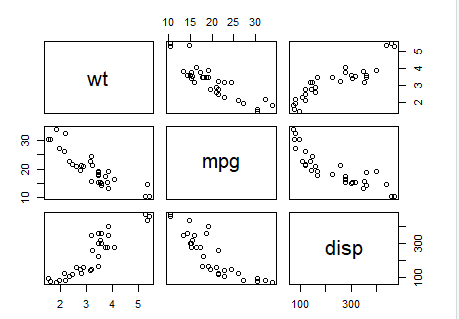
print(df)

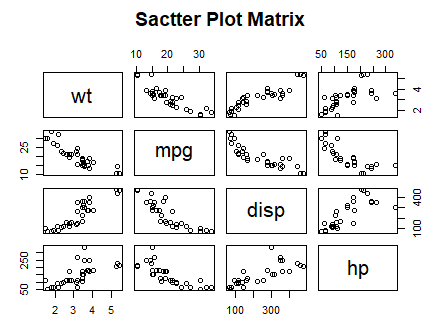
v<-df[,c("Salary","Basic")]

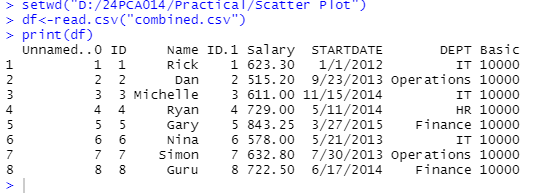
print(v)

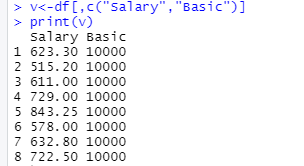
pairs(~Salary+Basic,data = v,main="Scatter Plot Matrix")

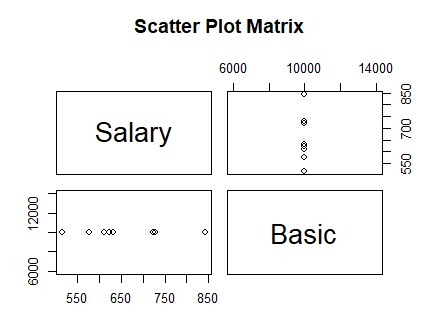
**OUTPUT:**

****

****

****

****

****

**RESULT:**

Thus, our program has been successfully saved and executed.

**19.Inventory Database Program**

**Aim :** To write an R program to connect with a MySQL database using RMySQL and DBI packages, retrieve data from tables, perform join operations, and calculate transaction-wise and item-wise net amounts.

**Algorithm :  
Step 1:** Start the process to connect R with MySQL and perform data analysis.  
**Step 2:** Open RStudio and load the required libraries RMySQL and DBI.  
**Step 3:** Define the MySQL driver using dbDriver("MySQL").  
**Step 4:** Establish a connection with the database test1 by providing username, password, database name, and host.  
**Step 5:** Retrieve and display all tables from the database using dbListTables().  
**Step 6:** Execute a query "SELECT \* FROM Items" using dbSendQuery().  
Fetch the results into a data frame.Print the Items table.  
**Step 7:** Execute a query "SELECT \* FROM Transaction" using dbSendQuery().  
Fetch the results into a data frame.Print the Transaction table.  
**Step 8:** Perform an INNER JOIN between Transaction and Items tables to calculate transaction-wise net amounts.Query: SELECT UID, ItemCode, ItemName, ItemQty, ItemRate, (ItemRate \* ItemQty) AS NetAmount.Fetch the results and print them.  
**Step 9:** Perform an INNER JOIN with GROUP BY to calculate item-wise total amounts.  
Query: SELECT ItemCode, ItemName, SUM(ItemRate \* ItemQty) AS NetAmount.  
Fetch the results and print them.  
**Step 10:** End the program.

**Program :**

#Install RMysql Package ...

library(RMySQL)

library(DBI)

drv <- dbDriver("MySQL")

mysqlconnection = dbConnect(drv,username='root',password='',dbname='test1',host='localhost')

dbListTables(mysqlconnection)

ItemMaster <- dbSendQuery(mysqlconnection,"select \* from items")

data.frame <- fetch(ItemMaster)

print(data.frame)

Transaction <- dbSendQuery(mysqlconnection,"select \* from Transaction")

data.frame <- fetch(Transaction)

print(data.frame)

TransactionAmount <- dbSendQuery(mysqlconnection,"select T.UID,I.ItemCode,I.ItemName,T.ItemQty,I.ItemRate,

(I.ItemRate \* T.ItemQty)as NetAmount

from Transaction T inner join Items I on I.ItemCode = T.ItemCode ")

data.frame <- fetch(TransactionAmount)

print(data.frame)

ItemAmount <- dbSendQuery(mysqlconnection,"select I.ItemCode,I.ItemName,

sum((I.ItemRate \* T.ItemQty))as NetAmount

from Transaction T inner join Items I on I.ItemCode = T.ItemCode

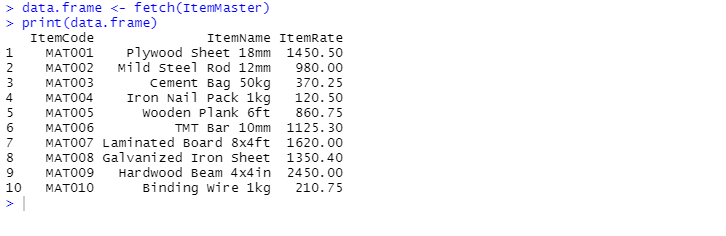
group by I.ItemCode,I.ItemName

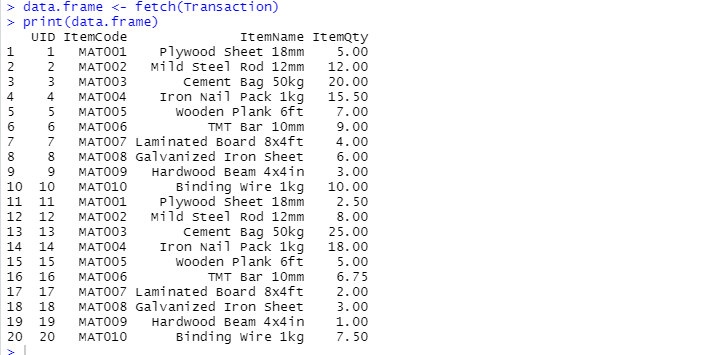
")

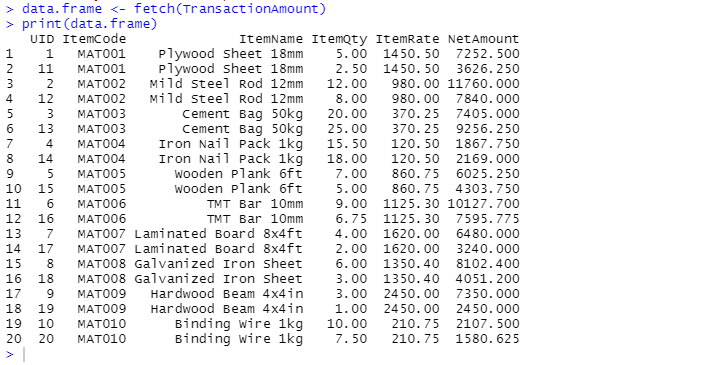
data.frame <- fetch(ItemAmount)

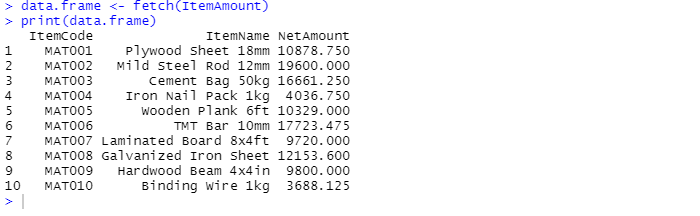
print(data.frame)

**OUTPUT:**









**RESULT:**

This, our program has been successfully saved and executed.

**20.Student Database Program**

**Aim :**

To write a R program to connect RStudio with a MySQL database using the RMySQL package and perform data retrieval and analysis

**Algorithm :**

**Step 1:** Start the process to connect RStudio and load the required database libraries.

**Step 2:** Establish a connection to the MySQL database hosted in XAMPP.

**Step 3:** List all tables available in the connected database.

**Step 4:** Retrieve and display data from the Department, Student, Subject, and Transactions tables..

**Step 5:** Find the highest and lowest marks scored in each subject..

**Step 6:** Calculate the total marks obtained by each student in each subject across all departments.

**Step 7:** Compute the average marks for each subject within every department.

**Step 8:** Calculate the overall average marks for each department.

**Step 9:** Display the results of each analysis.

**Step 10:** Close the database connection

**Step 11:** Stop the process

**PROGRAM:**

library(RMySQL)

library(DBI)

drv <- dbDriver("MySQL")

mysqlconnection = dbConnect(drv,username='root',password='',dbname='test1',host='localhost')

dbListTables(mysqlconnection)

Department <- dbSendQuery(mysqlconnection,"select \* from Department")

data.frame <- fetch(Department)

print(data.frame)

Student <- dbSendQuery(mysqlconnection,"select \* from Student")

data.frame <- fetch(Student)

print(data.frame)

subject <- dbSendQuery(mysqlconnection,"select \* from subject")

data.frame <- fetch(subject)

print(data.frame)

Transactions <- dbSendQuery(mysqlconnection,"select \* from Transactions")

data.frame <- fetch(Transactions)

print(data.frame)

# Highest & Lowest Score for each Subject

ScoreSubject <- dbSendQuery(mysqlconnection,"SELECT S.SubjectName,MAX(T.mark)as HighestScore,

MIN(T.mark)as LowestScore FROM transactions T

inner join subject S on S.SubjectID = T.SubjectID

group by S.SubjectName ")

data.frame <- fetch(ScoreSubject)

print(data.frame)

# Total Secured in each subject by each student in each department

TotalMarkSubject <- dbSendQuery(mysqlconnection,"SELECT S.SubjectName,sum(T.mark)as Mark,STD.StudentName,D.DepartmentName

FROM transactions T

inner join subject S on S.SubjectID = T.SubjectID

inner join student std on std.StudentID = T.StudentID

inner join department d on d.DeptID = T.DeptID

group by S.SubjectName,STD.StudentName,D.DepartmentName

order by D.DepartmentName,S.SubjectName;")

data.frame <- fetch(TotalMarkSubject)

print(data.frame)

# Subject average of each department

SubjectAvg <- dbSendQuery(mysqlconnection,"SELECT AVG(T.mark)as mark,S.SubjectName,D.DepartmentName

FROM transactions T

inner join subject S on S.SubjectID = T.SubjectID

inner join department d on d.DeptID = T.DeptID

group by D.DepartmentName,S.SubjectName

order by D.DepartmentName;")

data.frame <- fetch(SubjectAvg)

print(data.frame)

# average of department

DepartmentAvg <- dbSendQuery(mysqlconnection,"SELECT AVG(T.mark)as mark,D.DepartmentName

FROM transactions T

inner join department d on d.DeptID = T.DeptID

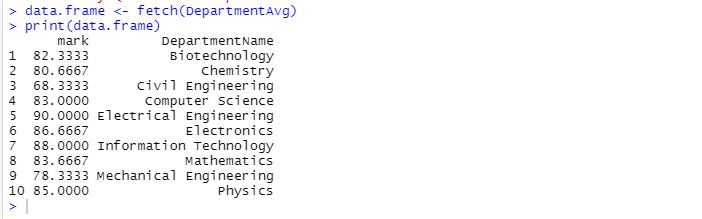
group by D.DepartmentName

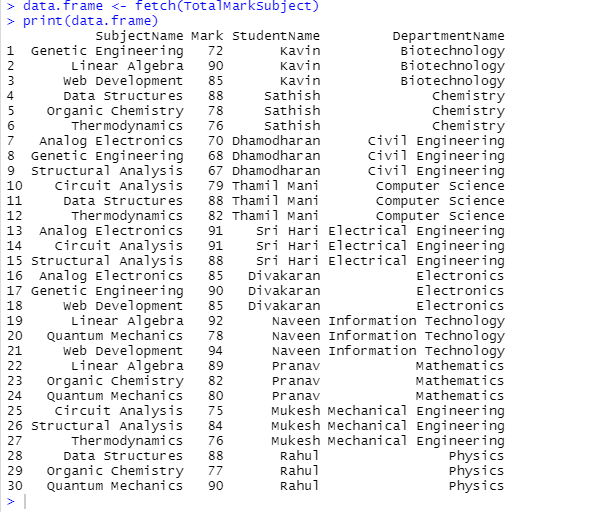
order by D.DepartmentName;")

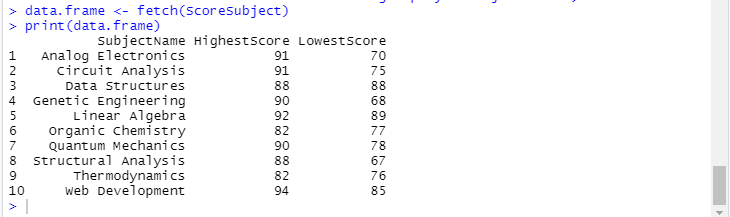
data.frame <- fetch(DepartmentAvg)

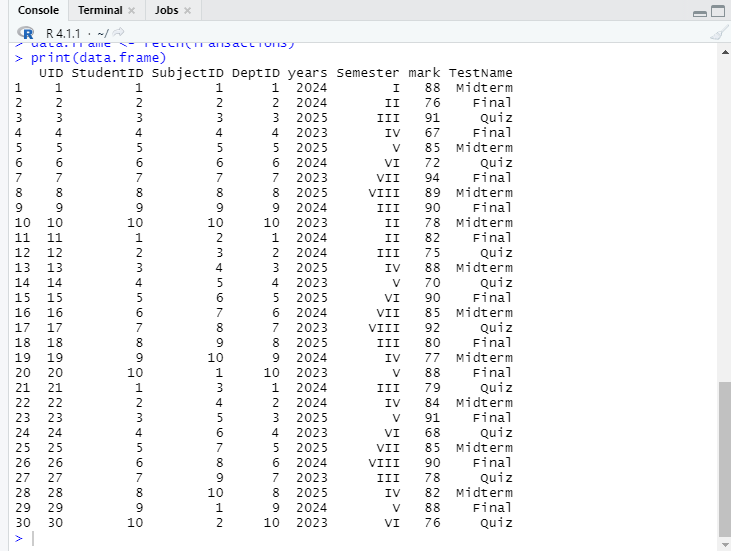
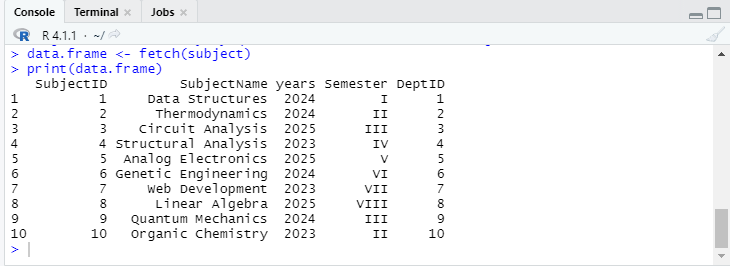
print(data.frame)

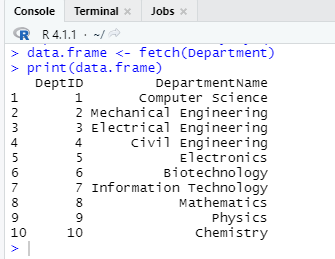
**OUTPUT:**









**RESULT:**

This program has been successfully saved and executed.

**21. Data Analytics for Guna.xlsx data with visualization of graph**

**Aim :**

To write an R Program perform data analysis and visualization in **R** using a Guna dataset

**Algorithm :**

**Step 1:** Start the process to find prime numbers below a given number.

**Step 2:** Use setwd() to specify the folder containing the dataset, Confirm the path using getwd()

**Step 3:** Use read.csv() to load the CSV file into a dataframe (here, guna).

**Step 4:** Use head() to display the first few rows. Use summary() to get descriptive statistics of all columns.

**Step 5:** Plot histograms for numeric attributes (e.g., Apptitude, Attitude) to visualize frequency distribution. Use hist() with parameters like breaks, col, border.

**Step 6:** Generate bar plots for categorical data (e.g., Gender, gunas) using barplot(table(...)).

**Step 7:** Use boxplot() to show the spread, quartiles, and outliers of numeric variables (Apptitude, Attitude).

**Step 8:** Use plot() to represent the relationship between two numeric variables (Apptitude vs Attitude).

**Step 9:** Create a frequency table using table().Convert to data.frame and use pie() to visualize category proportions (e.g., gunas).

**Step 10:** Stop the program

**Program :**

getwd()

setwd("C:/Users/MCA-007/Documents/R prog")

guna = read.csv("guna.csv")

head(guna)

summary(guna)

hist(table(guna$Apptitude),xlab = "Aptitude",col = "Green",border = "red",xlim = c(0,100),ylim = c(0,30),breaks = 5)

#bar plot

barplot(table(guna$gunas),col = "green")

barplot(table(guna$Attitude),col = "green")

barplot(table(guna$Gender),col = "green")

barplot(table(guna$Apptitude),col = "green")

#box plot

boxplot(guna$Apptitude,col = c("green"))

boxplot(guna$Attitude,col = c("green"))

hist(table(guna$Apptitude),xlab = "Aptitude",col = "green",border = "red",xlim = c(0,100),ylim = c(0,30),breaks = 5)

hist(table(guna$gunas),xlab = "gunas",col = "green",border = "red",xlim = c(0,100),ylim = c(0,30),breaks = 5)

hist(table(guna$Attitude),xlab = "Attitude",col = "green",border = "red",xlim = c(0,100),ylim = c(0,30),breaks = 10)

#Scatter plot

plot(x=guna$Apptitude,y=guna$Attitude,xlab = "Apptitude",ylab = "Attitude",main = "Apptitude vs Attitude",

col=c("red","green"))

legend("bottomright",pch = 5,col = c("red","green"),legend = c("Aptitude","Attitude"))

#pie chart

d=as.data.frame(table(guna$gunas))

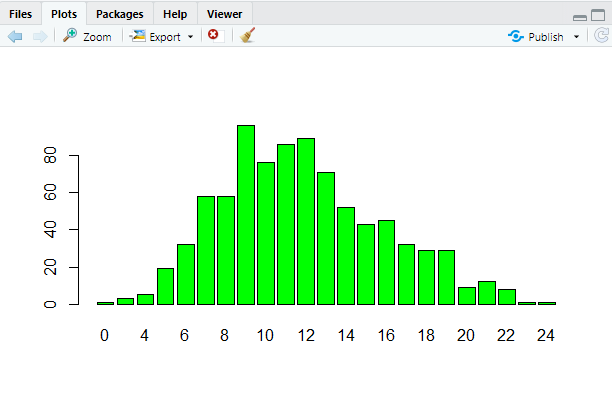
print(d)

pie(d$Freq,c("Rajasic","Sattvic","Tamasic"))

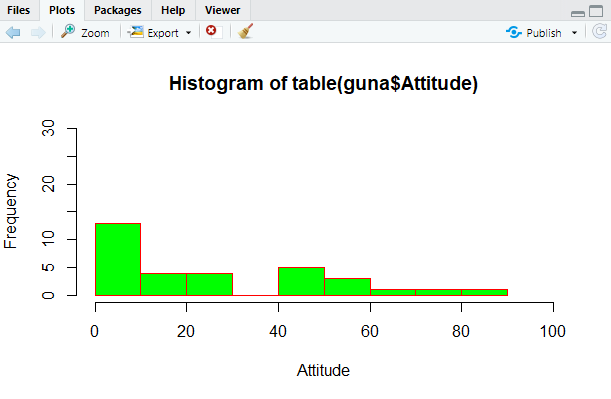
pie(d$Freq,d$Var1)

**OUTPUT:**

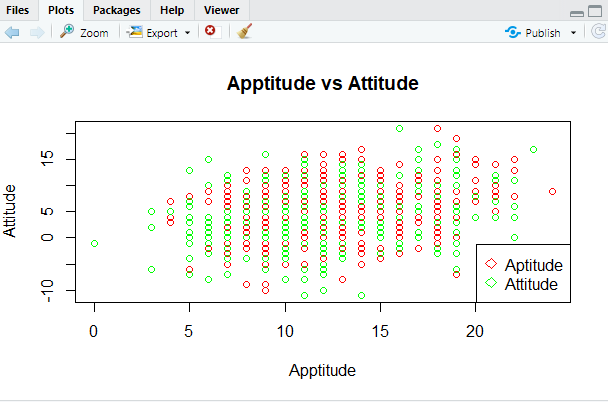
**Bar Plot**



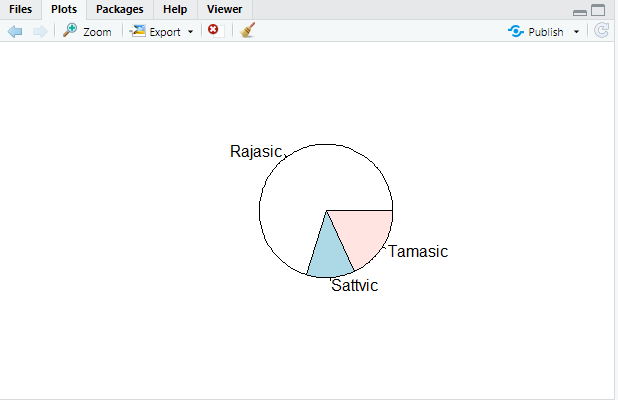
**Box Plot**



**Scatter Plot**



**Pie Chart**



**RESULT:**

Thus, our program has been successfully saved and executed.

**22. Data Analytics for creditlimit.csv with visualization of graph**

**Aim :**

To write an R Program perform data analysis and visualization using a creditlimit.csv dataset

**Algorithm :**

**Step 1:** Start the process.

**Step 2:** Use setwd() to specify the folder where the dataset is stored.

**Step 3:** Use read.csv() to read Credit\_train.csv into a dataframe called credit. **Step 4:** Use head(), summary(), and str() to check the structure and contents.Use colSums(is.na()) to find missing values.

**Step 5:** Draw histograms for numeric variables (BUSAGE, MAXLINEUTIL, TOTACBAL) to observe frequency distribution.

**Step 6:** Draw bar plots for categorical variables (BUSTYPE, DEFAULT) to view category frequencies.

**Step 7:** Create box plots for numeric variables to detect spread and outliers (BUSAGE, TOTACBAL, DAYSDELQ).

**Step 8:** Plot BUSAGE vs TOTACBAL and use colors to differentiate default (Y/N).

**Step 9:** Show the proportion of default vs non-default customers.

**Step 10:** Stop the program

**Program :**

setwd("C:/Users/MCA-007/Documents/R prog")

if (file.exists("Credit\_train.csv")) {

credit <- read.csv("Credit\_train.csv")

} else {

stop("File 'Credit\_train.csv' not found in working directory!")

}

head(credit)

summary(credit)

str(credit)

colSums(is.na(credit))

hist(na.omit(credit$BUSAGE),

main="Histogram of BUSAGE",

xlab="BUSAGE", col="skyblue", border="black")

hist(na.omit(credit$MAXLINEUTIL),

main="Histogram of MAXLINEUTIL",

xlab="MAXLINEUTIL", col="orange", border="black")

hist(na.omit(credit$TOTACBAL),

main="Histogram of TOTACBAL",

xlab="TOTACBAL", col="green", border="black")

barplot(table(credit$BUSTYPE),

main="Bar Plot of BUSTYPE", col="blue")

barplot(table(credit$DEFAULT),

main="Bar Plot of DEFAULT", col=c("green","red"))

boxplot(na.omit(credit$BUSAGE), main="Boxplot of BUSAGE", col="lightgreen")

boxplot(na.omit(credit$TOTACBAL), main="Boxplot of TOTACBAL", col="lightpink")

boxplot(na.omit(credit$DAYSDELQ), main="Boxplot of DAYSDELQ", col="lightblue")

credit$DEFAULT <- as.factor(credit$DEFAULT)

plot(credit$BUSAGE, credit$TOTACBAL,

xlab="BUSAGE", ylab="TOTAL ACCOUNT BALANCE",

main="BUSAGE vs TOTACBAL",

col=ifelse(credit$DEFAULT=="Y","red","green"),

pch=19)

legend("topleft", legend=c("Default=Y","Default=N"),

col=c("red","green"), pch=19)

d <- as.data.frame(table(credit$DEFAULT))

pie(d$Freq, labels=paste(d$Var1, ":", d$Freq),

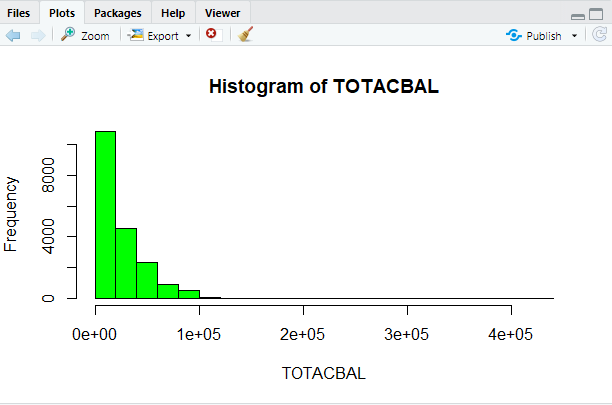
col=c("green","red"), main="Pie Chart of Default Status")

nums <- credit[, sapply(credit, is.numeric)]

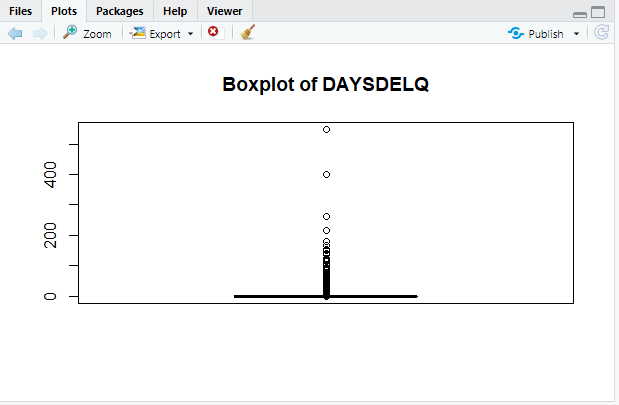
cor\_matrix <- cor(nums, use="complete.obs")

**OUTPUT:**

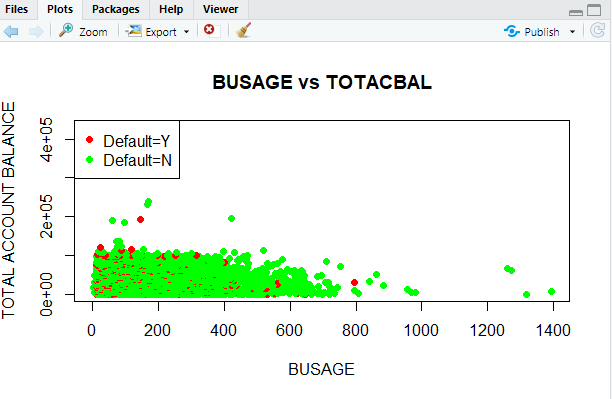
**Bar Plot**



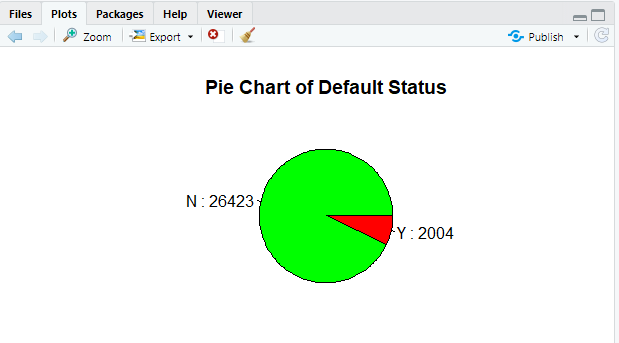
**Box Plot**



**Scatter Plot**



**Pie Chart**



**RESULT:**

Thus, our program has been successfully saved and executed.

**23. Mean, Median, Mode, Standard Deviation and Variance**

**AIM:**

To Write an R program **Mean, Median, Mode, Standard Deviation**, and **Variance** of a dataset and to handle missing values and trimmed mean where necessary.

**Algorithm :**

**Step 1:** Start the process RStudio and open a new script or R file.

**Step 2:** Use mean(x) to calculate the mean of the vector x.

**Step 3:** Use mean(x, trim = 0.3) to calculate the trimmed mean, excluding extreme values.

**Step 4:** Use mean(x, na.rm = TRUE) to calculate mean while removing NA values

**Step 5:** Use median(x) to calculate the median (middle value) of the vector

**Step 6:** Use median(x) / 3 or other expressions if you want to process the median further.

**Step 7:** Create a user-defined function getmode() calculate the mode of a numeric vector.

**Step 8:** Call getmode(v) on a vector to return the mode (most frequent value).

**Step 9:** Use sd(x) to calculate the standard deviation of the data.

**Step 10:** Use var(x) to calculate the variance of the data.

**Step 11:** stop the process

**Program:**

x<-c(12,7,3,4.2,18,2,54,-21,8,-5)

mean(x)

result.mean<-mean(x)

print(result.mean)

x<-c(-21,-5,2,3,4.2,7,8,12,18,54)

mean(x)

22.2/4

mean1<-mean(x,trim=0.3)

print(mean1)

x<-c(3,4.2,7,8)

mean(x)

x<-c(1,2,3.3,4.44567,54.898)

mean(x)

boxplot(x,horizontal=FALSE)

print(mean(x,trim=0.1))

mean1<-mean(x,trim=0.1)

x<-c(1,2,3.3,4.44567)

boxplot(x,horizontal=FALSE)

x<-c(12,7,3,4.2,18,2,54,-2,8,-5)

x<-c(21,5,2,3,4.2,7,8)

mean(x)

result.mean<-mean(x,trim=0.3)

print(result.mean)

x<-c(12,7,3,4.2,18,2,54,-21,8,-5,NA)

mean(x)

result.mean<-mean(x)

print(result.mean)

result.mean<-mean(x,na.rm=TRUE)

print(result.mean)

x<-c(12,7,3,4.2,18,2,54,-21,8,-5)

median.result<-median(x)

print(median.result)

#4.2+7=11.2/2=5.6

x<-c(-21,-5,2,3,4.2,7.89876,8,12,18,54,78)

median.result<-median(x)

print(median.result)

t=median(x)/3

median.result<-median(t,trim=0.3)

print(median.result)

x=c(2.2,4,8,6,7,9,2)

getmode<-function(v){

uniqv<-unique(v)

print(unique(v))

print(tabulate(match(v,uniqv)))

uniqv[which.max(tabulate(match(v,uniqv)))]

}

v<-c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)

unique(v)

tabulate(match(v,unique(v)))

result<-getmode(v)

print(result)

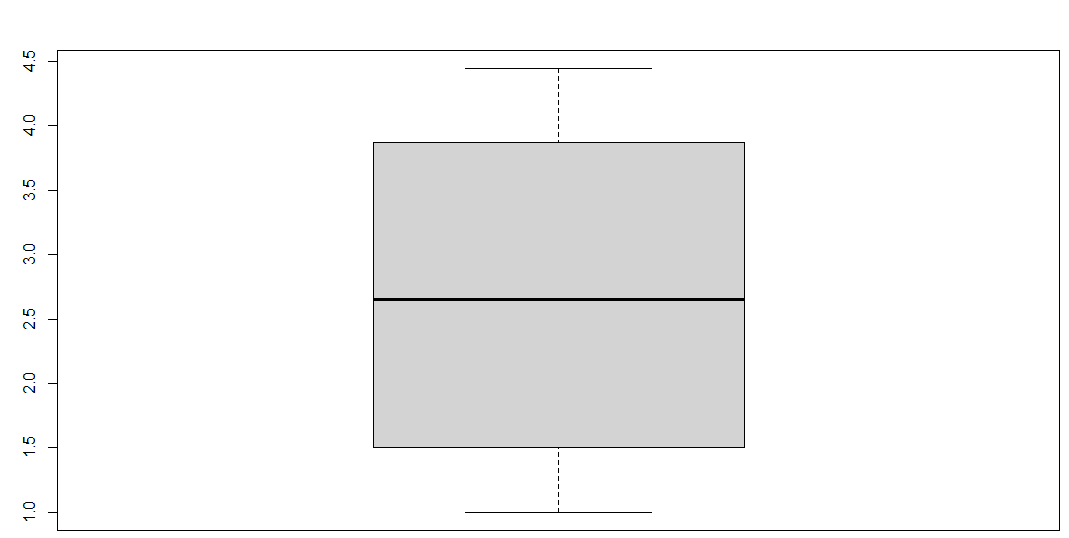
sd.result<-sd(x)

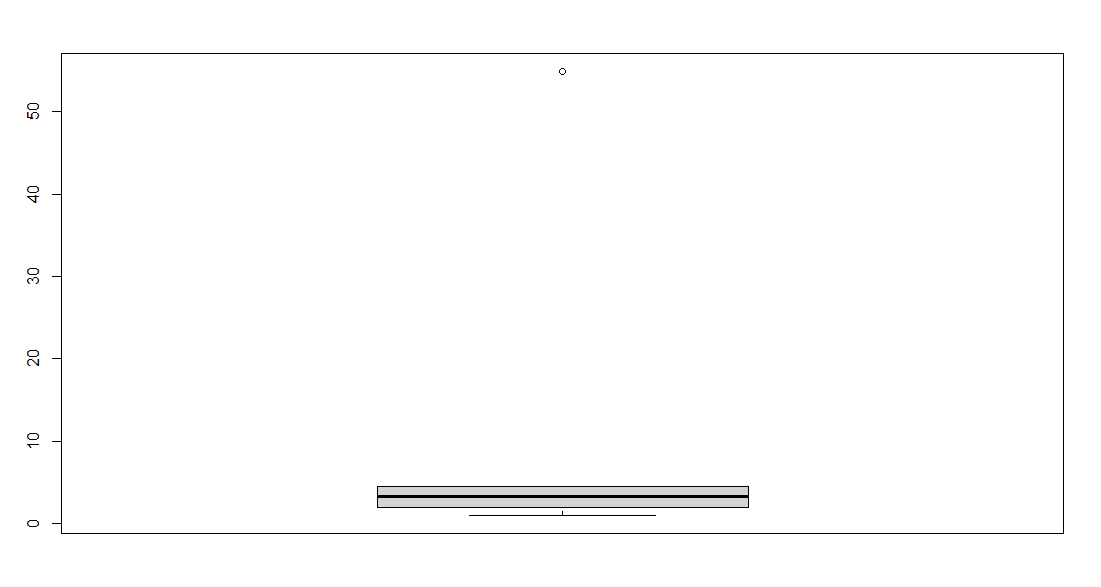
print(sd.result)

var.result<-var(x)

print(var.result)

**OUTPUT:**

****

****

**RESULT:**

This, our program has been successfully saved and executed.

**29**. **Binomial Distribution**

**Aim :**

To Write an R program to study the Binomial Distribution using the following methods

1. **Using Formula**
2. **Using dbinorm,pbinorm,qbinorm,rbinorm ,**

**Algorithm :**

**Step 1 :** Start the process to find the Binomial Distribution using the mathematical formula and built-in functions (dbinom, pbinom, qbinom, and rbinom).

**Step 2 :** Define parameters: number of trials n, success probability p, and values of x to analyze.

**Step 3 :** Use dbinom(x, n, p) to calculate and print the probability of exactly x successes.

**Step 4 :** Calculate probabilities for multiple values of x (e.g., 0 to 4) and print each.

**Step 5 :** Use pbinom(x, n, p) to compute and print cumulative probabilities up to specific values of x.

**Step 6 :** Find probability between two points by subtracting two cumulative probabilities and print the difference.

**Step 7 :** Use qbinom(p, n, p) to find the smallest x for which cumulative probability exceeds p.

**Step 8 :** Generate and print random binomial samples using rbinom() with specified size and probability.

**Step 9 :** End the program.

**Program :**

x<-4

y<-dbinom(x,10,0.5)

print(y)

#dbinom

x<-0

y0<-dbinom(x,4,0.5)

print(y0)

x<-1

y1<-dbinom(x,4,0.5)

print(y1)

x<-2

y2<-dbinom(x,4,0.5)

print(y2)

x<-3

y3<-dbinom(x,4,0.5)

print(y3)

x<-4

y4<-dbinom(x,4,0.5)

print(y4)

y = y0+y1+y2+y3+y4

print(y)

#pbinom

x1 <- 2

y1 <- pbinom(x1,4,0.5)

print(y1)

x2 <- 3

y2 <- pbinom(x2,4,0.5)

print(y2)

y = y2-y1

print(y)

#qbinom

x <- qbinom(0.375,4,0.5)

print(x)

#rbinom

x<- rbinom(8,150,0.4)

print(x)

print(dbinom(0,size = 12,prob = 0.2)+

dbinom(1,size = 12,prob = 0.2)+

dbinom(2,size = 12,prob = 0.2)+

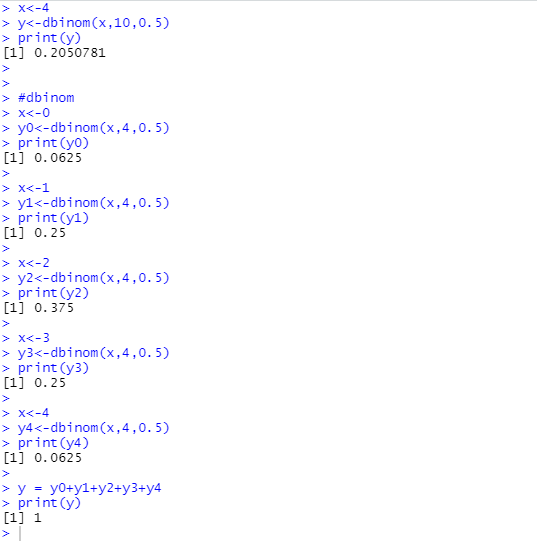
dbinom(3,size = 12,prob = 0.2)+

dbinom(4,size = 12,prob = 0.2))

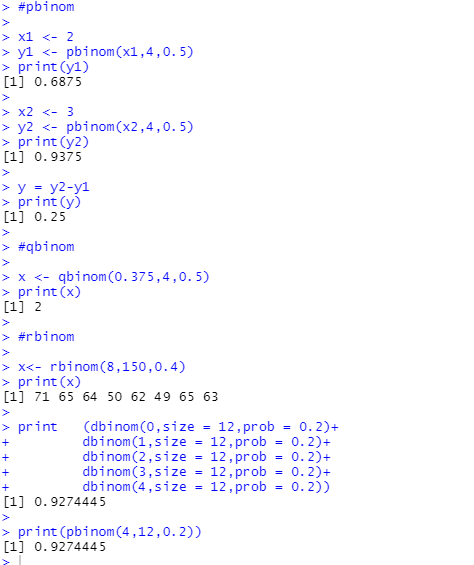
print(pbinom(4,12,0.2))

**OUTPUT :**

1. **Using Formula :**



**b) Using dbinorm,pbinorm,qbinorm,rbinorm**



**RESULT:**

Thus, the program has been successfully saved and executed.