**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 | Write a program to find the uniqueness of data such as E,F,W,B,X,R,T not using factor function Array should be used. |
| 7 | Write a program to apply Mathematical Operations on Matrix of 4x4 using function concept ( + , - , \* , / ) . |
| 8 | Write a program to apply 3 Vectors of Name , Age , Gender . It should be converted as Data Frame provide atleast 10 datasets. Filter data as follows:-  a) Age > 25  b) Age in between 10 , 40.   c) Age < 25 & Gender = Female.   d) Age in between 10 and 45 & gender = male. e) Name Starting with Character "A" . |
| 9 | Write a program to handle CSV file having: emp\_id , name , age , gender , salary , DA Provide Atleast 20 datasets:-   a) Genderwise.   b) Age > 40 & gender = male.   c) salary > 600 for different genders.   d) find out the different between salary & Basic + DA.   e) salary > 600 & Basic > 300 & DA < 200. Mentioned in the CSV file. |
| 10 | Write a program to handle JSON file having: emp\_id , name , age , gender , salary , DA Provide Atleast 20 datasets:-   a) Genderwise.   b) Age > 40 & gender = male.   c) salary > 600 for different genders.   d) find out the different between salary & Basic + DA.   e) salary > 600 & Basic > 300 & DA < 200. Mentioned in the JSON file. |
| 11 | Write a Program with xml file having emp\_id , name , age , gender , salary , basic , DA provide atleast 20 datasets:-   a) Genderwise.   b) Age > 40 & gender = male.  c) salary > 600 for different genders.   d) find out the different between salary & basic + DA .   e) salary > 600 & Basic > 300 & DA < 200. |
| 12 | Write a Program with xlsx file having empid , name , age , gender , salary , basic , DA provide atleast 20 dataset:-   a) Genderwise.   b) Age > 40 & gender = male.  c) salary > 600 for different genders.   d) find out the different between salary & basic + DA .   e) salary > 600 & Basic > 300 & DA < 200. |
| 13 | Write a Program having xml file as empid , name , age , gender , & Json file as empid , basic , DA , salary , merge xml file & Json file as CSV:-  a) Genderwise.   b) Age > 40 & gender = male.   c) salary > 600 for different genders.   d) find out the different between salary & basic + DA .  e) salary > 600 & Basic > 300 & DA < 200. |
| 14 | Write a Program to Develop a Pie chart. |
| 15 | Write a program to Develop a Bar chart. |
| 16 | Write a program to Develop a Box plot. |
| 17 | Write a program to Develop a Line chart. |
| 18 | Write a program to Develop a Scatter plot. |
| 19 | Write a program to create a Database Inventory with the following tables:  Item Master:- Item code , Item Name , Item Rate .  Item Transaction:- Item code , Item Name , Item Quantity. Using R language write following:-  a) List all Item from Item Master.   b) List all Item from Item Transaction.  c) List all Item from Item Transactions in such a way that amount = Item Quantiy \* Item  Rate.  d) List all Items by grouping on Item Name with Total Amount = Sum(Amount). |
| 20 | Write a Program to a student Management System with the following tables:- **Student:** Student Id , Student Name , Department , Year , Semaster.  **Department**: Dept Id , Department Name. **Subject Master**: Dept Id , Subject Id , Subject Name , Year , Semaster. **Transaction**: Student Id , Subject Id , Dept Id , Year , Semaster , Marks , Test Name. a) Find the Highest Score & Lowest Score for each subject. b) Find the Total Secured in each subject by each student in each  department. c) Subject Average of each class. d) Performance of students as on Average for Each Department. |
| 21 | Write a program to data analysis for Guna.xlsx data with visualization of graph. |
| 22 | Write a program to data analysis for Creditlimit.csv |
| 23 | Write a program to find Mean , Median , Mode , SD , Variance. |
| 24 | Write a program to find correlation  Correlation:  a) Pearsons Correlation Coefficient(formula 1).   b) Correlation Coefficient using Mean( formula 2).   c) Bulid in function in R - Correlation. |
| 25 | Write a program to find Linear Regression Equation:   a) Y on X.   b) X on Y.   c) Build in Function in R using X and Y.   d) Build in Function in R using CSV file. |
| 26 | Write a program to find Multiple Linear Regression:   a) Data Provided through Vector.   b) Data Provided through build in table in R.   c) Data Provided through CSV file. |
| 27 | Write a program to find Logistic Regression:   a) Data Provided through Vector.   b) Data Provided through build in table in R.   c) Data Provided through CSV file. |
| 28 | Write a program to find Non - Linear Least Squares:  a) Data Provided through Vector.   b) Data Provided through build in table in R.   c) Data Provided through CSV file. |
| 29 | Write a program to find Bionomial Distribution:   a) Using Formula.   b) Using Dnorm , Pnorm , Rnorm , Qnorm. |
| 30 | Write a program to find Normal Distribution:   a)Using Formula.   b) Using Dnorm , Pnorm , Rnorm , Qnorm. |
| 31 | Write a program to find Poission Distribution:   a) Using Formula.   b) Dpois , Ppois , Qpois. |
| 32 | Write a program for Analysis of Variable using R:   a) Without Using Built in Function.   b) Using Built in Functions. |
| 33 | Write a program to find Features of Numpy , Mean , Median , Mode and Correlation Coefficient using Numpy of Python. |
| 34 | Write a program for Data Analysis using Pandas of Python having Imdb , Movie\_data. |
| 35 | Write a program for Normal Distribution Analysis of any CSV file using Python. |
| 36 | Write a program for Analysis of Variance using Python. |
| 37 | Write a program for Poisson Distribution Using R and Python |
| 38 | Write a program for  a) Decision Tree Using Python  b) Decision Tree Using R  c) Hierachical Cluster Using R |
| 39 | Write a program for  a) Chi - square Test for single vector  b) Chi - square Test for two-dimensional vector  c) Chi - square Test Using R |
| 40 | Write a program for Time Series Analaysis :  a) Moving Average  b) Auto Correlation & Partial Auto Correlation  c) ARIMA For Forecast  d) Find (p,d,q) for fitting suitable ARIMA For least Mean square Error |
| 41 | Write a program for Survival Analysis Using R and Survival Analysis Using Python  a) vector Data  b) Data From CSV |
| 42 | Write a Program for Random Forest using Python |

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) {

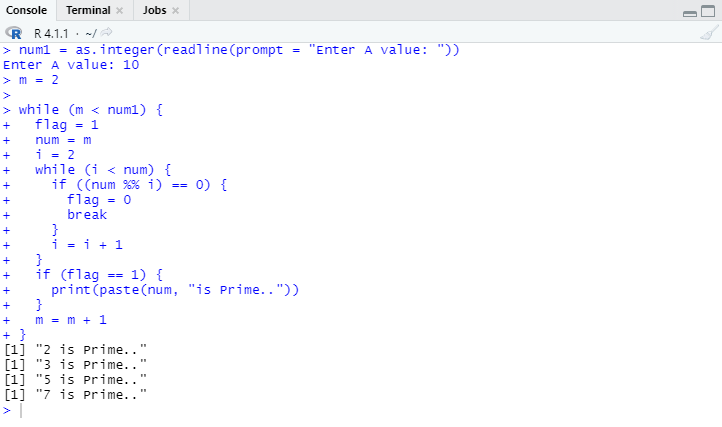
print(paste(num, "is Prime.."))

}

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: "))

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) {

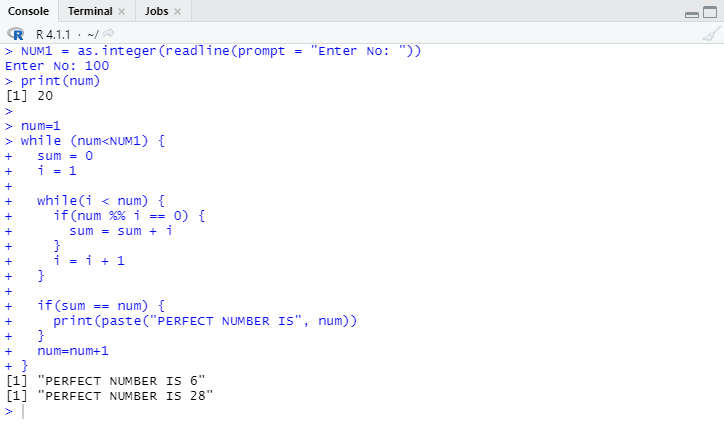
print(paste("PERFECT NUMBER IS", num))

}

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) {

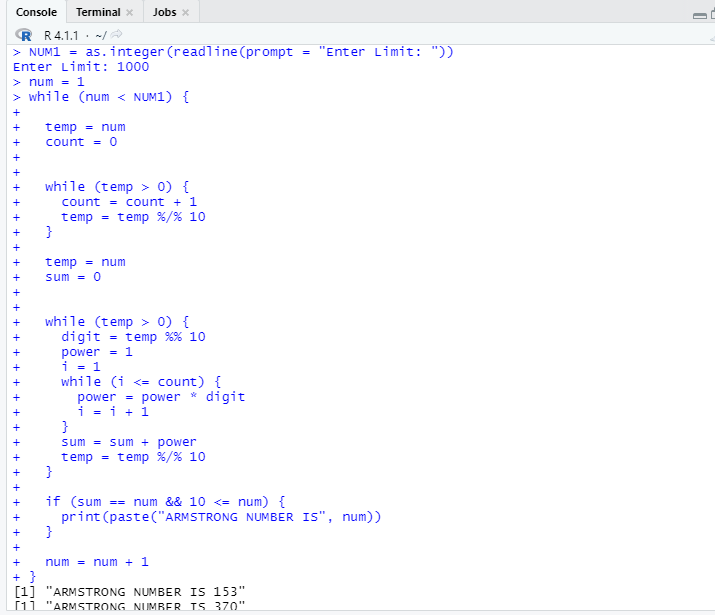
print(paste("ARMSTRONG NUMBER IS", num))

}

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)

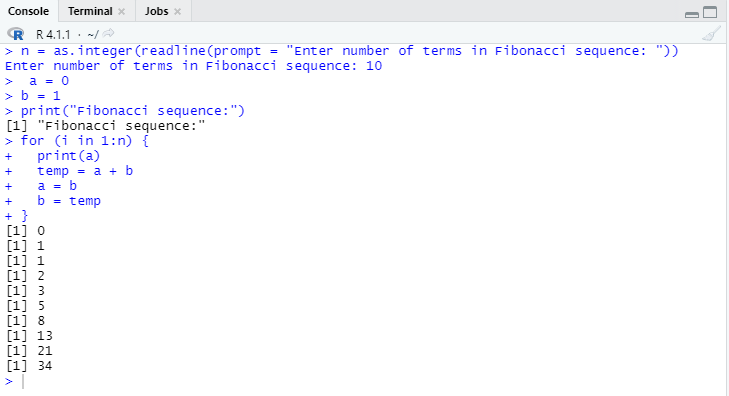
temp = a + b

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))

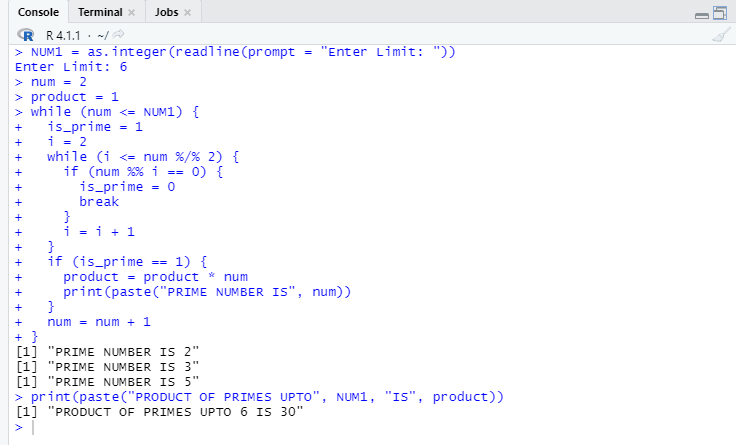
}

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])

}

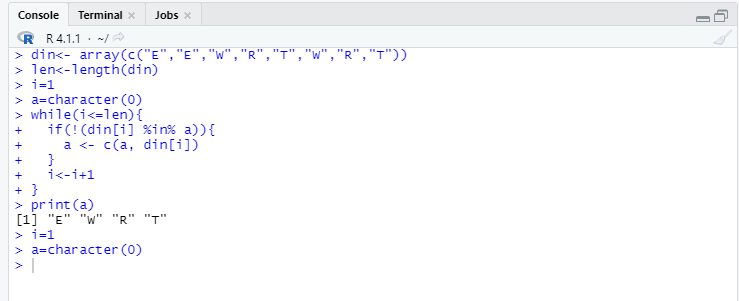
i<-i+1

}

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")

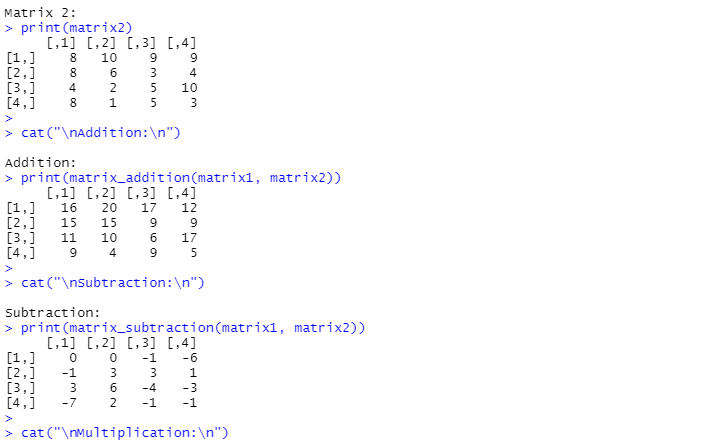
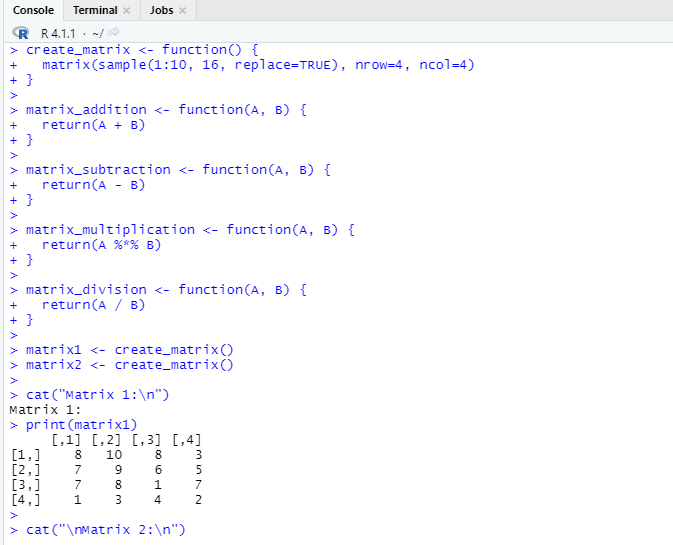
print(matrix\_subtraction(matrix1, matrix2))

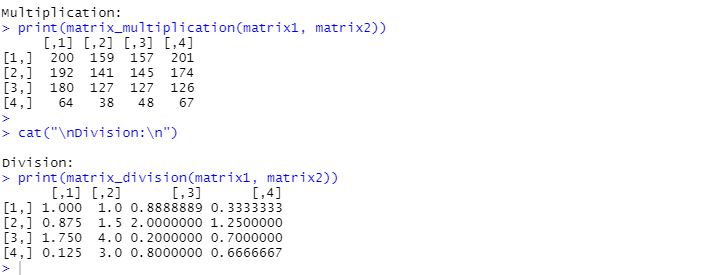
cat("\nMultiplication:\n")

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)

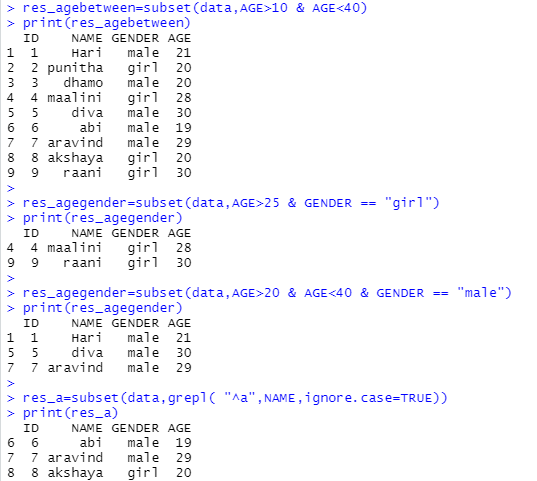
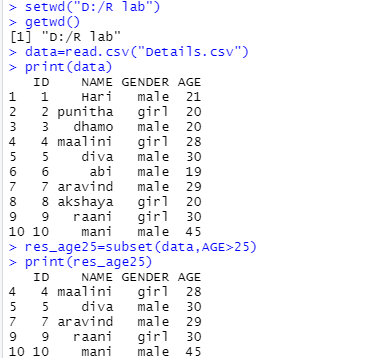
res\_agegender=subset(data,AGE>20 & AGE<40 & GENDER == "male")

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.

**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.

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

print(retval)

**Step 10:** Print the result.

**Step 11:** Stop the process.

**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)

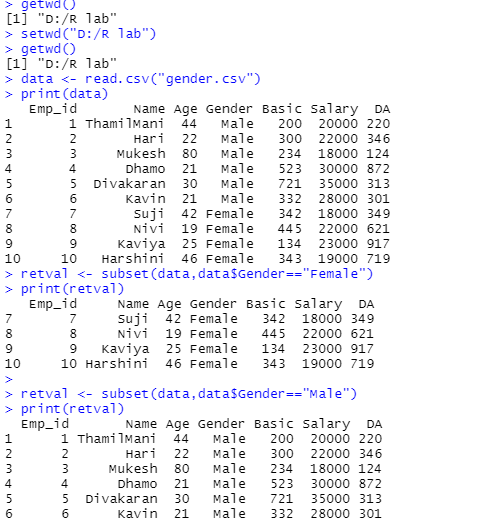
retval <- subset(data,data$Salary>600 & data$Basic>300 & data$DA<200)

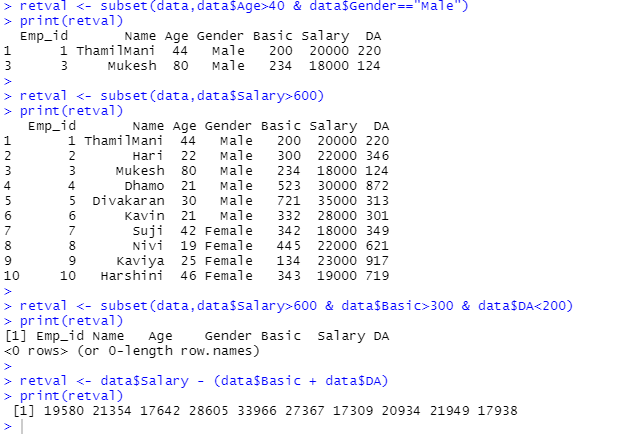
print(retval)

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** : Print the result.

**Step 10 :** End the process.

**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)

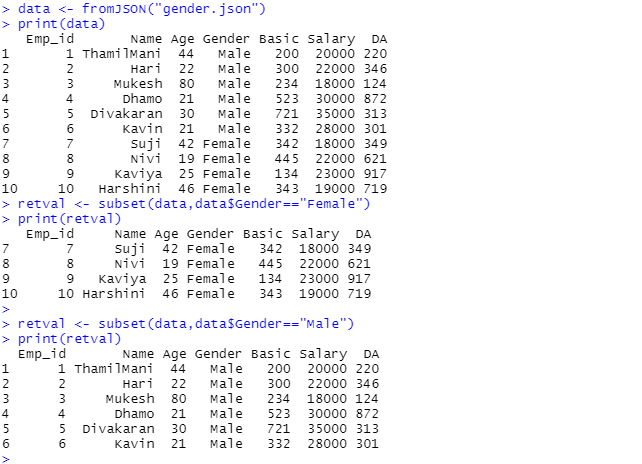
retval <- subset(data,data$Salary>600 & data$Basic>300 & data$DA<200)

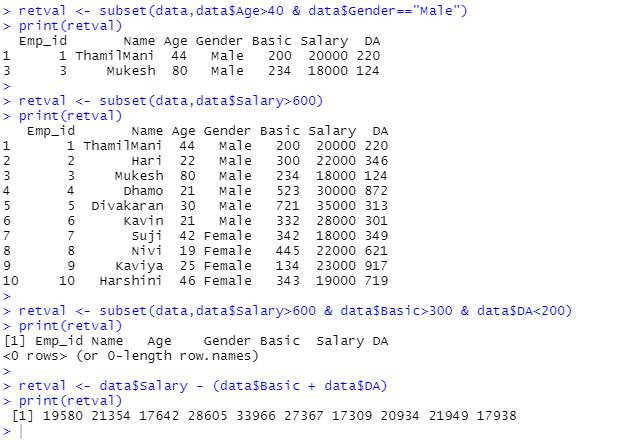
print(retval)

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

print(retval)

**OUTPUT:**





**RESULT:**

Thus, our program has been successfully saved and executed.

**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)

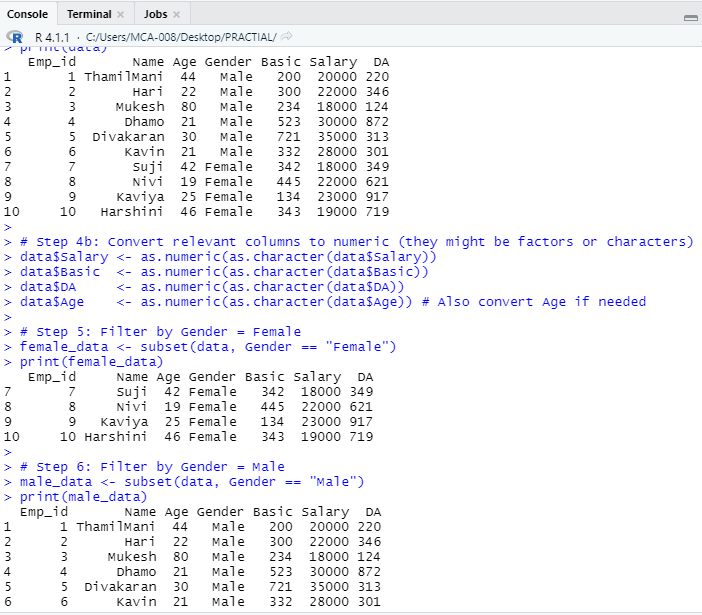
retval <- subset(data,data$Salary>600 & data$Basic>300 & data$DA<200)

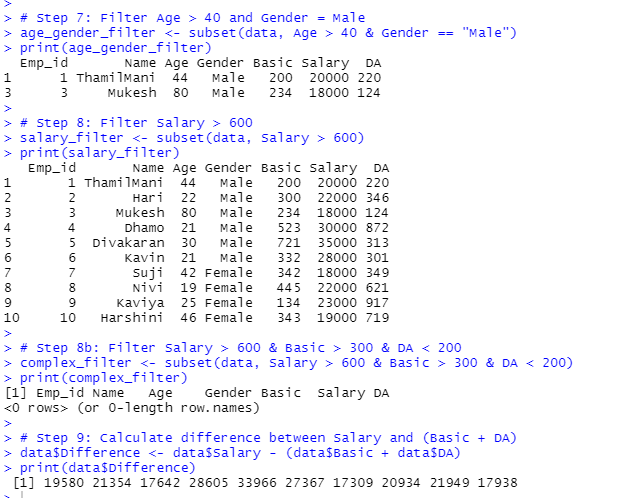
print (retval)

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

print (retval)

**OUTPUT:**

****

****

**RESULT:**

Thus, our program has been successfully saved and executed.

**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)

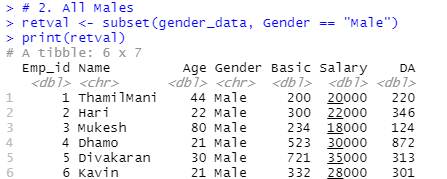
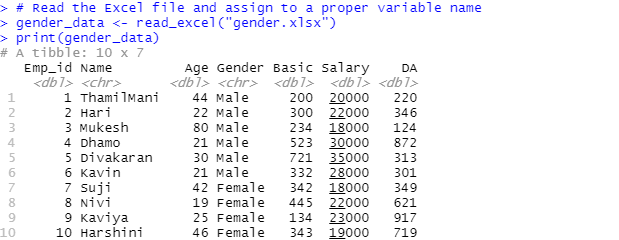
retval <- subset(data,data$Salary>600 & data$Basic>300 & data$DA<200)

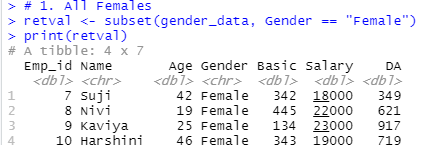
print(retval)

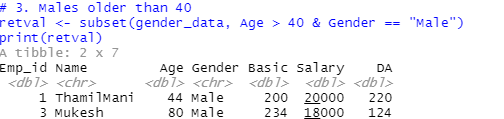
retval <- data$Salary - (data$Basic + data$DA)

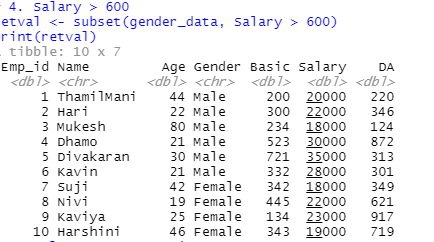
print(retval)

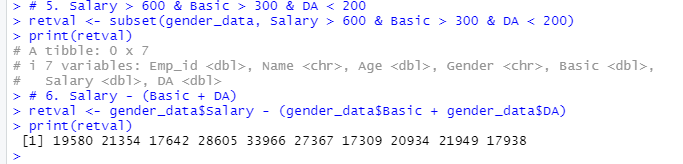
**OUTPUT:**











**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)

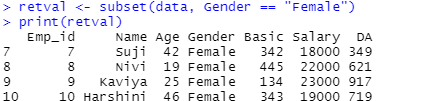
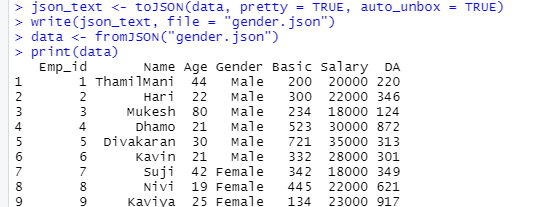
retval <- subset(data, Salary > 600 & Basic > 300 & DA < 200)

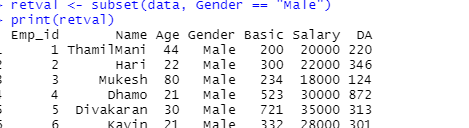
print(retval)

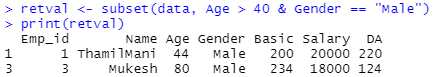
retval <- data$Salary - (data$Basic + data$DA)

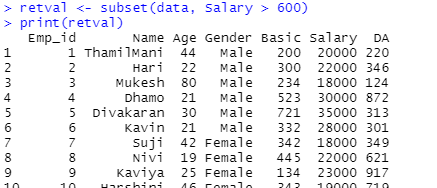
print(retval)

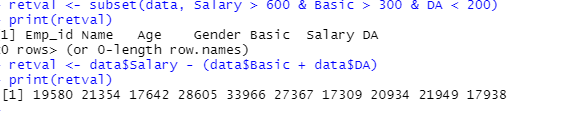
**OUTPUT:**

****









**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,

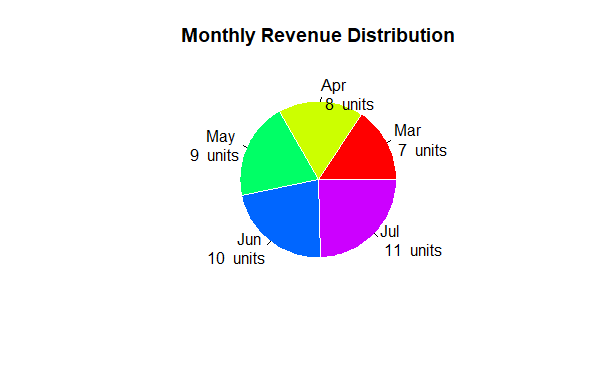
labels = lbl,

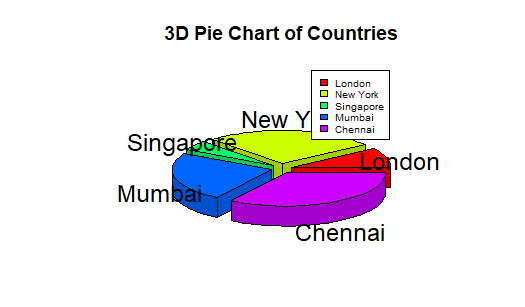
explode = 0.1,

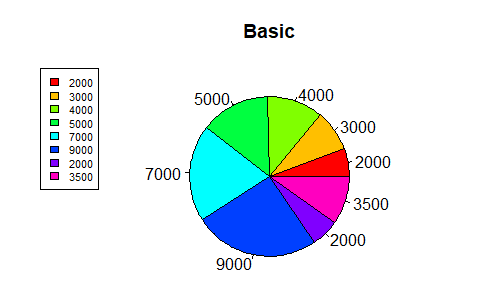
main = "3D Pie Chart - Salary",

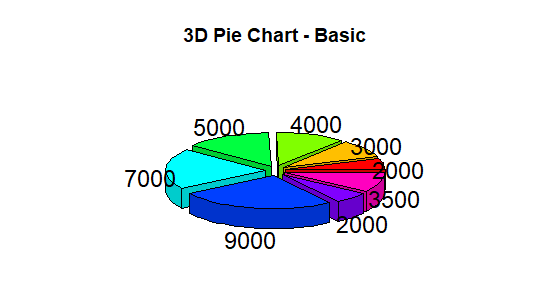
col = rainbow(length(v)))

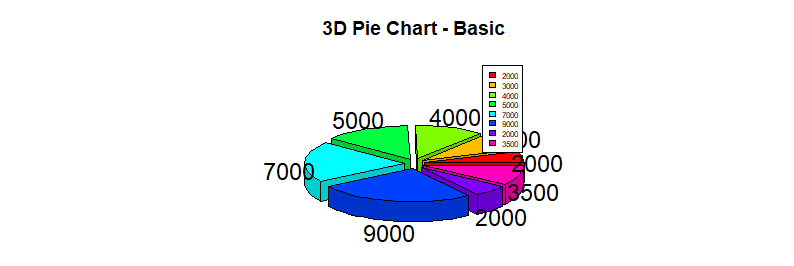
**OUTPUT:**

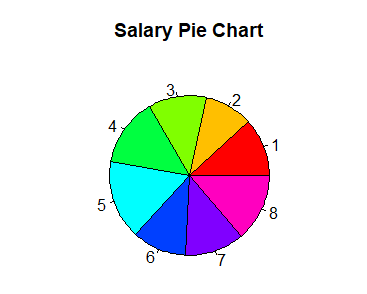


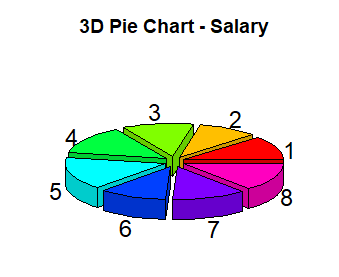












**RESULT:**

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")

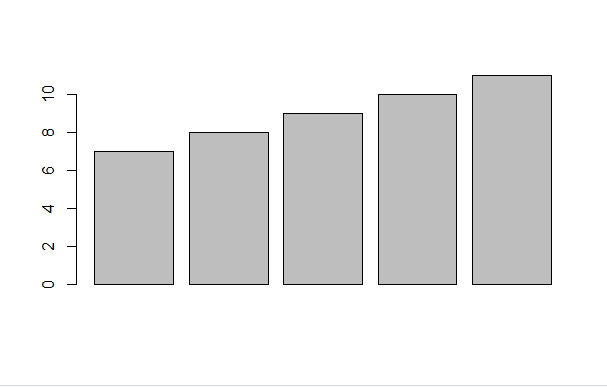
print(df)

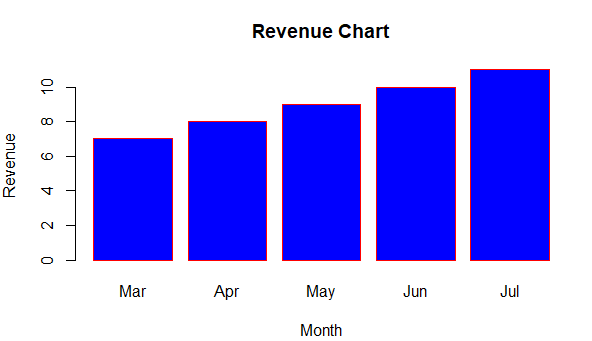
v<-df[,c("Basic")]

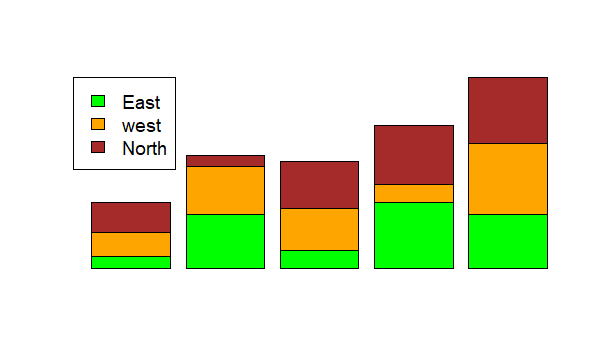
print(v)

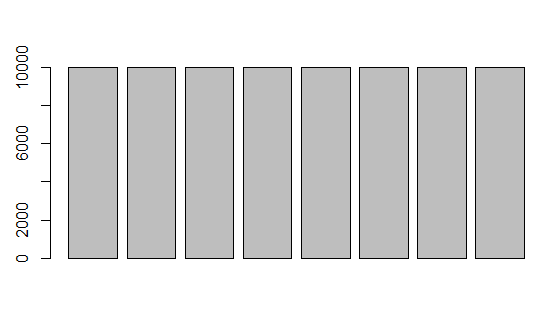
barplot(v)

**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:** End 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,

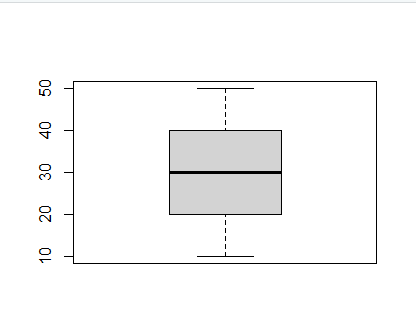
data = v,

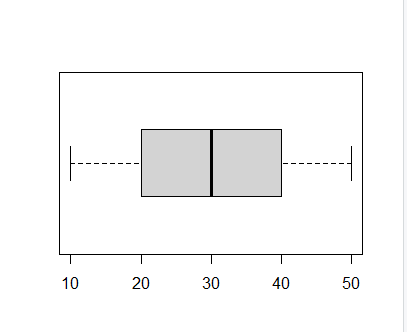
xlab = "Basic",

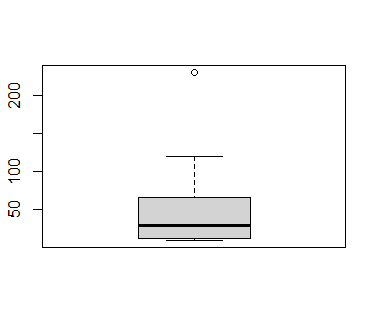
ylab = "Salary",

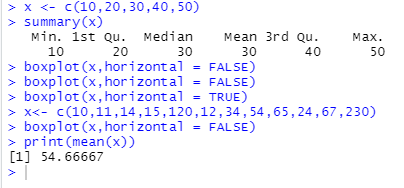
main = "Salary Chart",

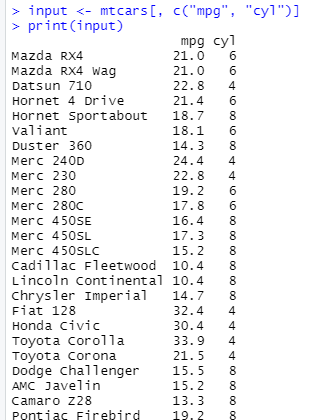
col = "lightblue")

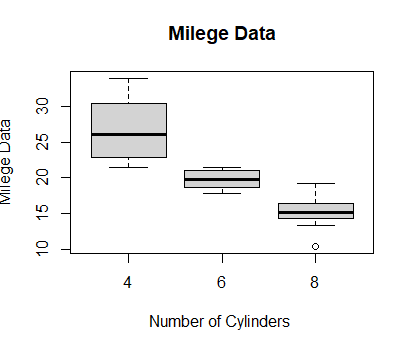
**OUTPUT:  
**

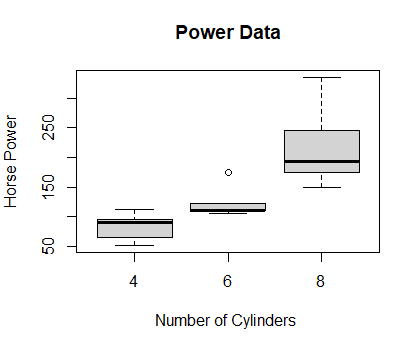
****

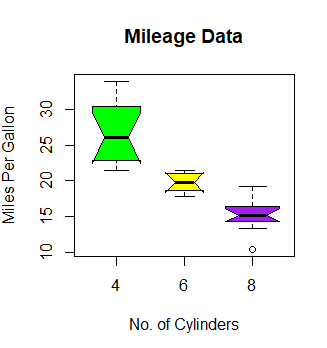
****

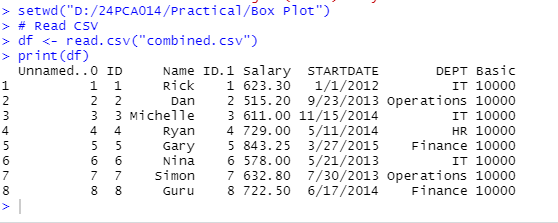
****

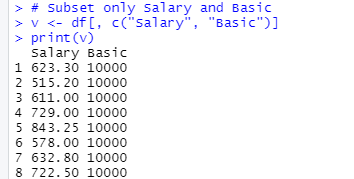
****

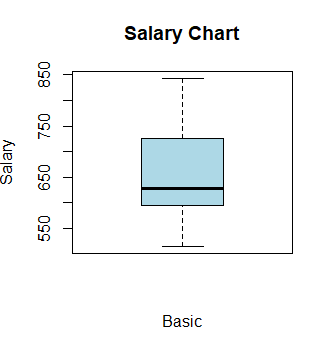
****

****

****

****

****

****

**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")

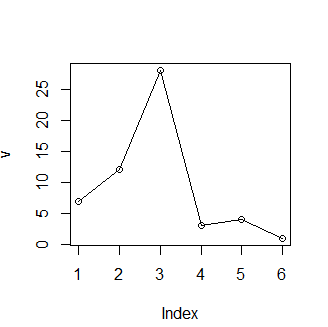
# Add legend

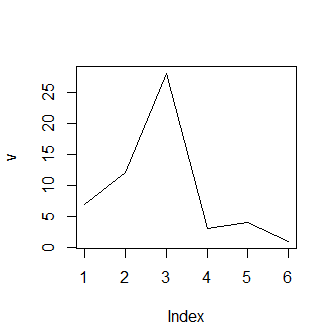
legend("topleft", c("Basic", "Salary"),

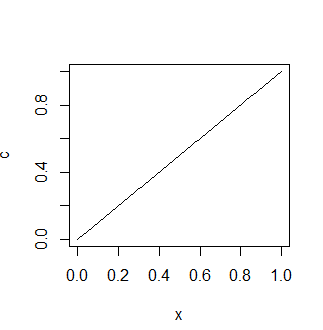
col=c("red","green"),

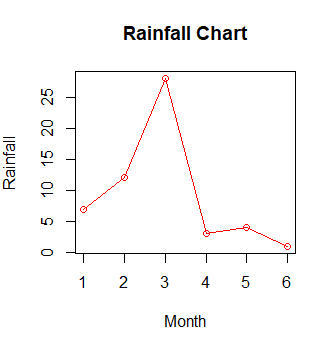
lty=1, pch=1, cex=0.8)

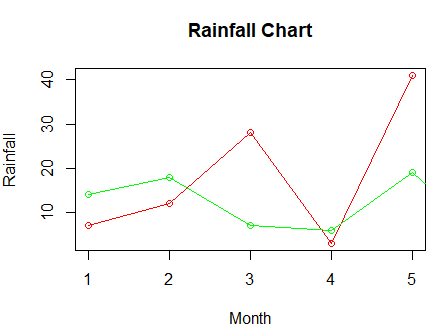
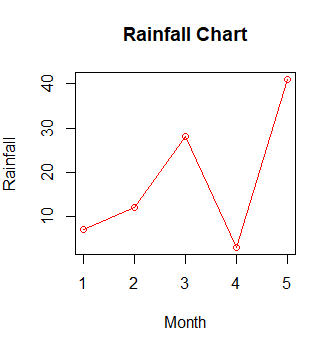
**OUTPUT:**

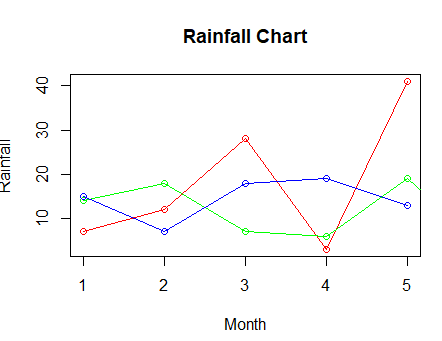
****

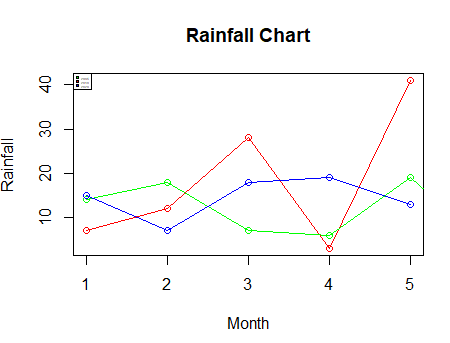
****

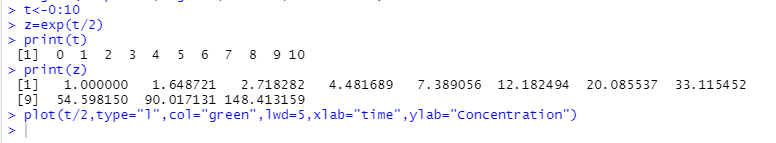
****

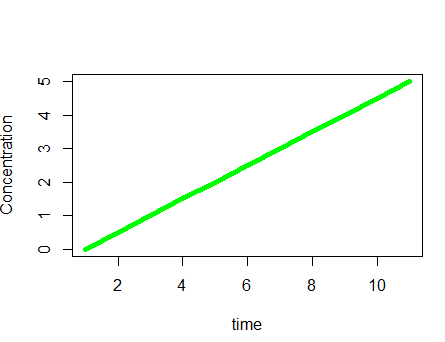
****

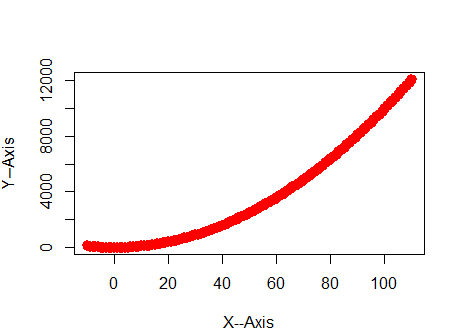
****

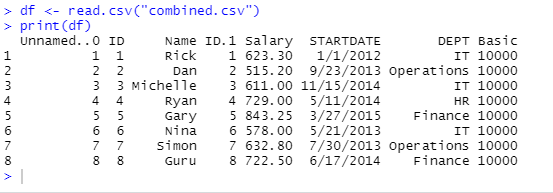
****

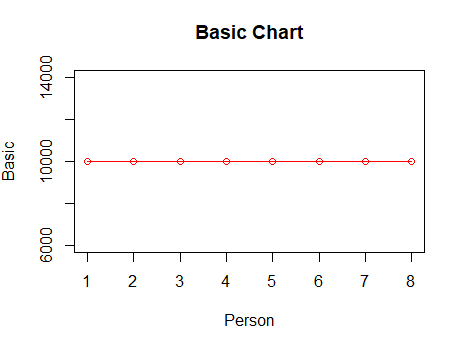
****

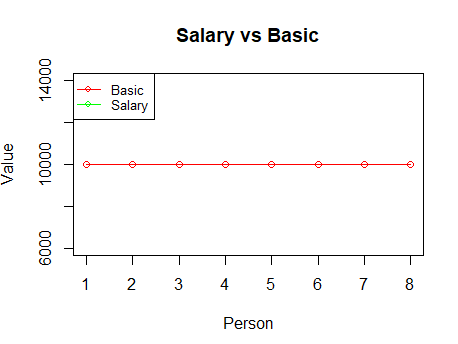
****

****

****

****

****

****

**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")

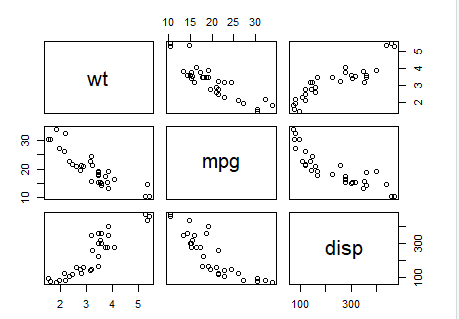
print(df)

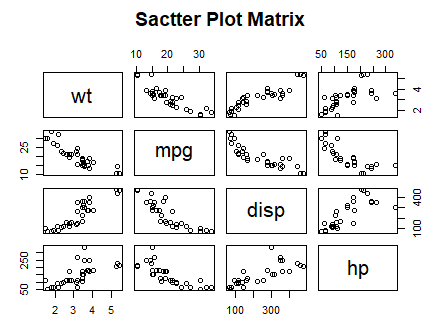
v<-df[,c("Salary","Basic")]

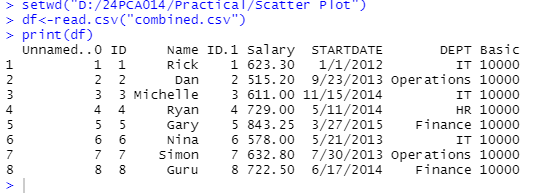
print(v)

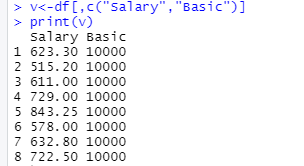
pairs(~Salary+Basic,data = v,main="Scatter Plot Matrix")

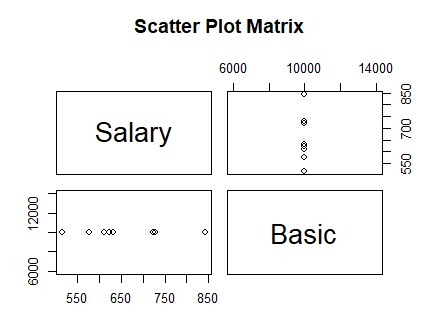
**OUTPUT:**

****

****

****

****

****

**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

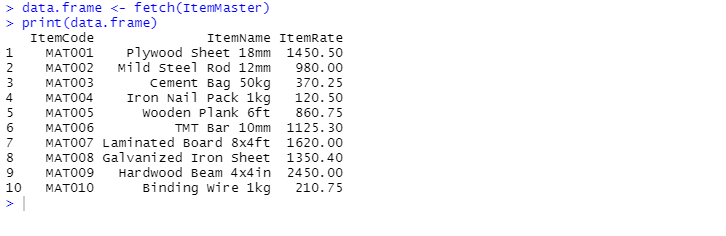
group by I.ItemCode,I.ItemName

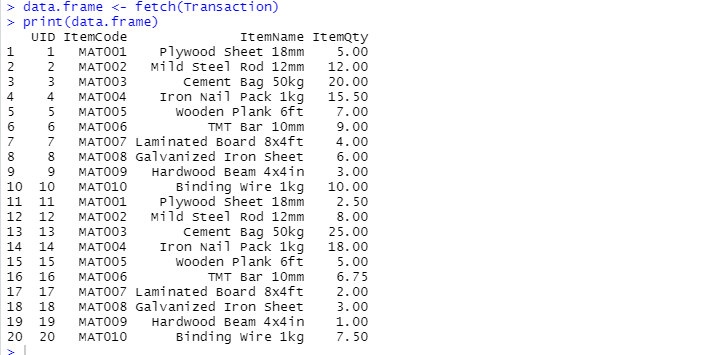
")

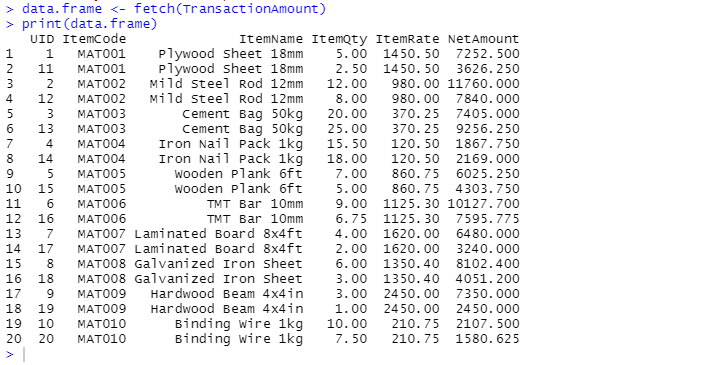
data.frame <- fetch(ItemAmount)

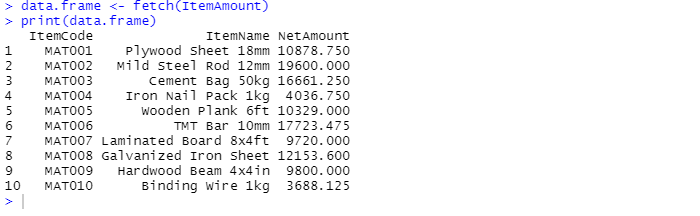
print(data.frame)

**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

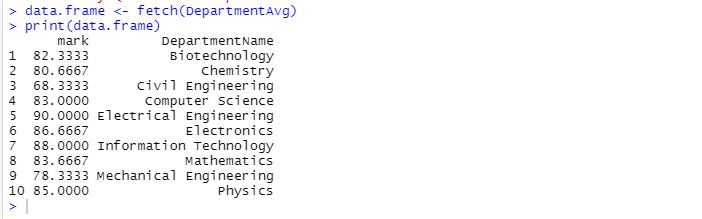
group by D.DepartmentName

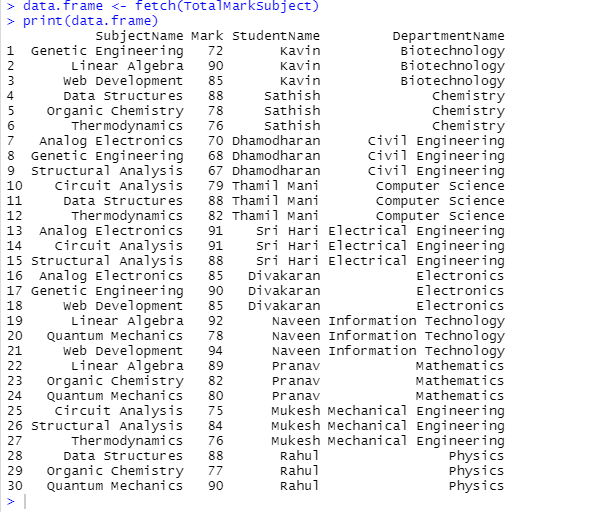
order by D.DepartmentName;")

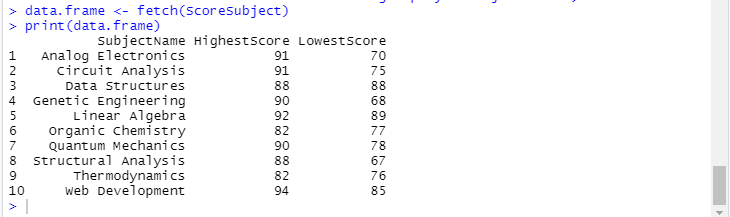
data.frame <- fetch(DepartmentAvg)

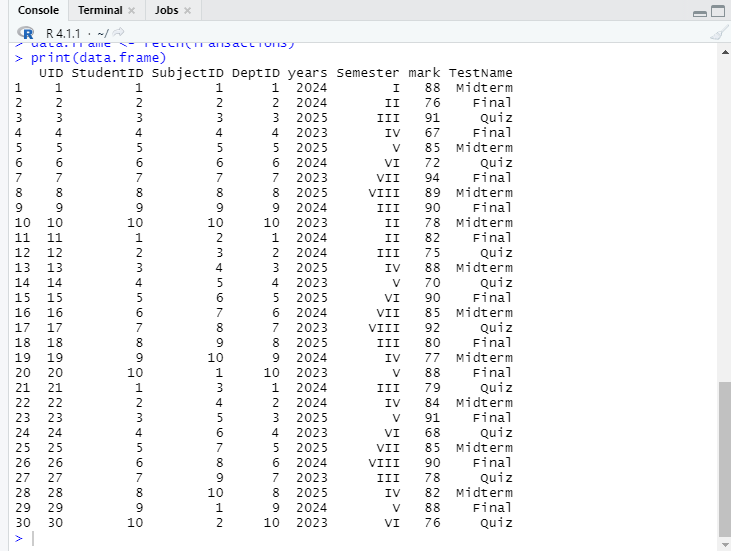
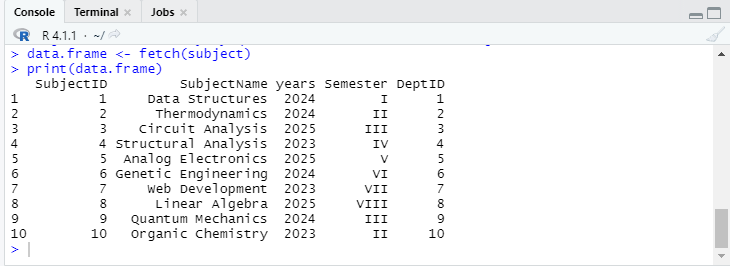
print(data.frame)

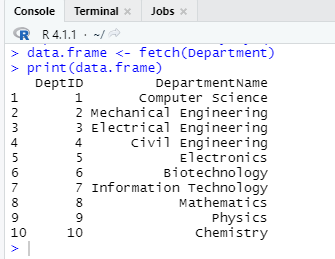
**OUTPUT:**









**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))

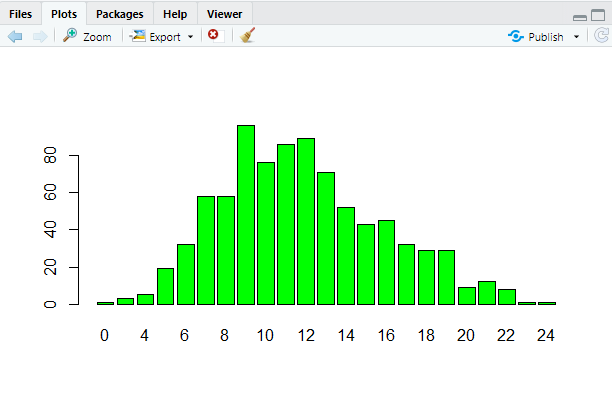
print(d)

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

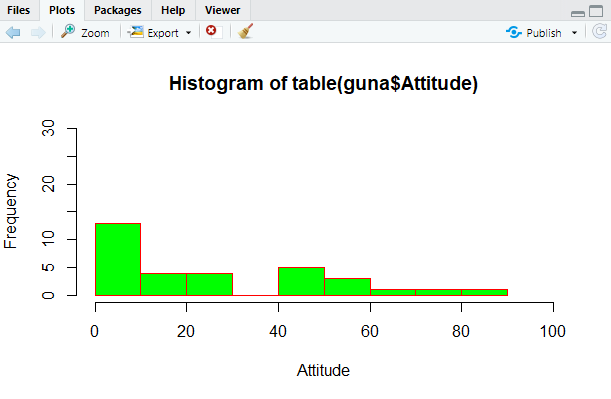
pie(d$Freq,d$Var1)

**OUTPUT:**

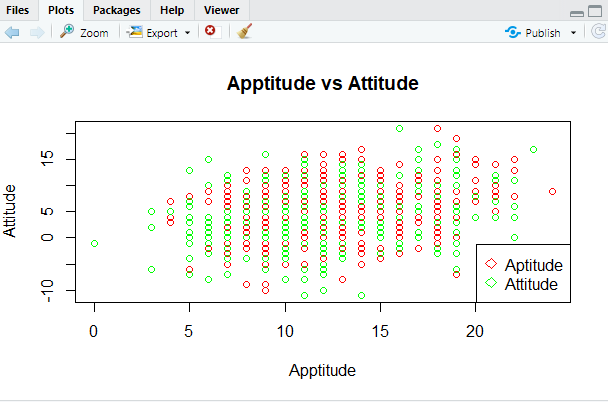
**Bar Plot**



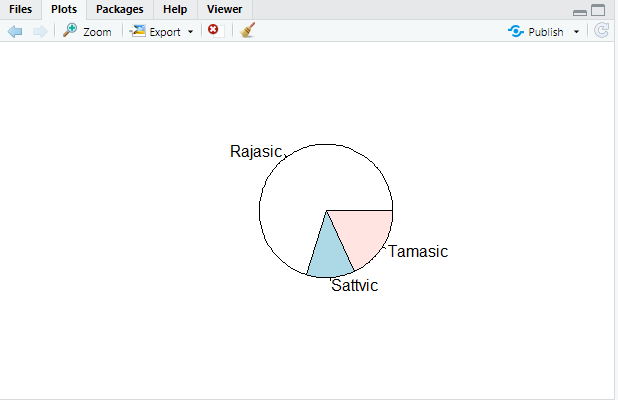
**Box Plot**



**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),

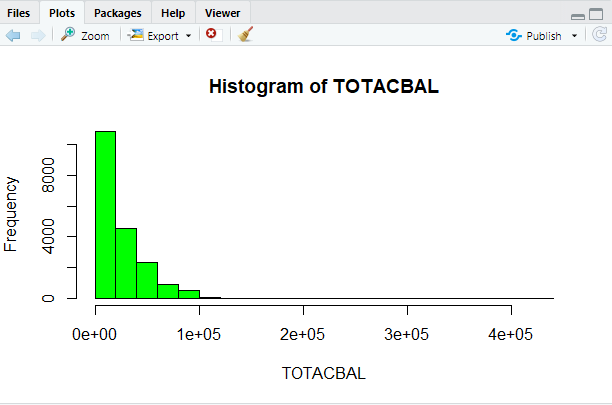
col=c("green","red"), main="Pie Chart of Default Status")

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

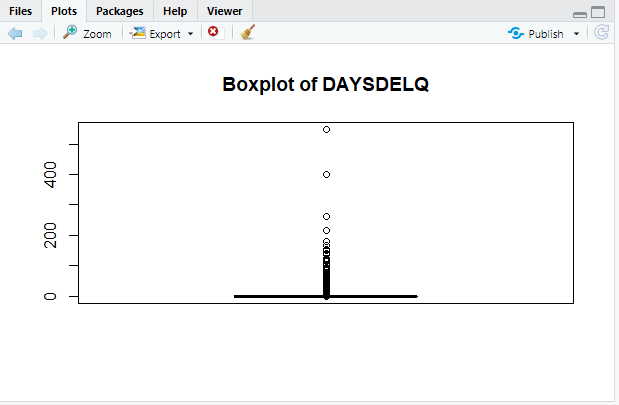
cor\_matrix <- cor(nums, use="complete.obs")

**OUTPUT:**

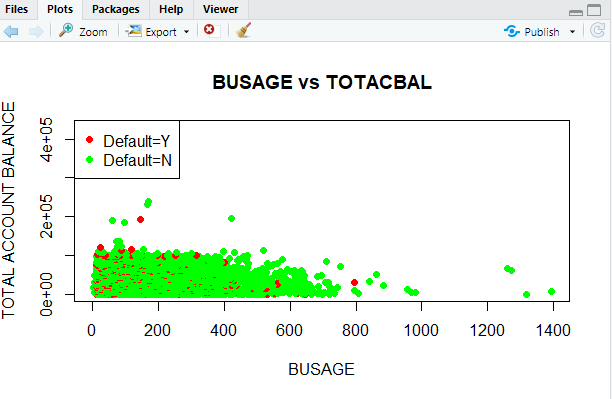
**Bar Plot**



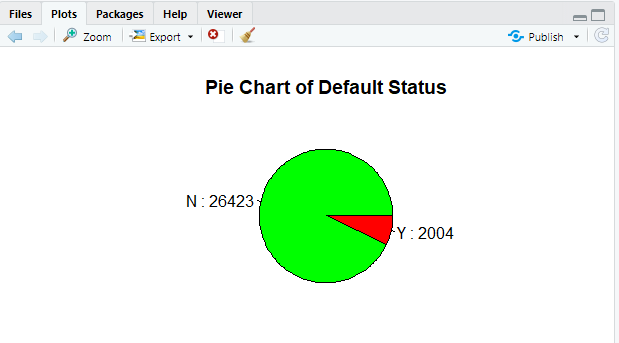
**Box Plot**



**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)

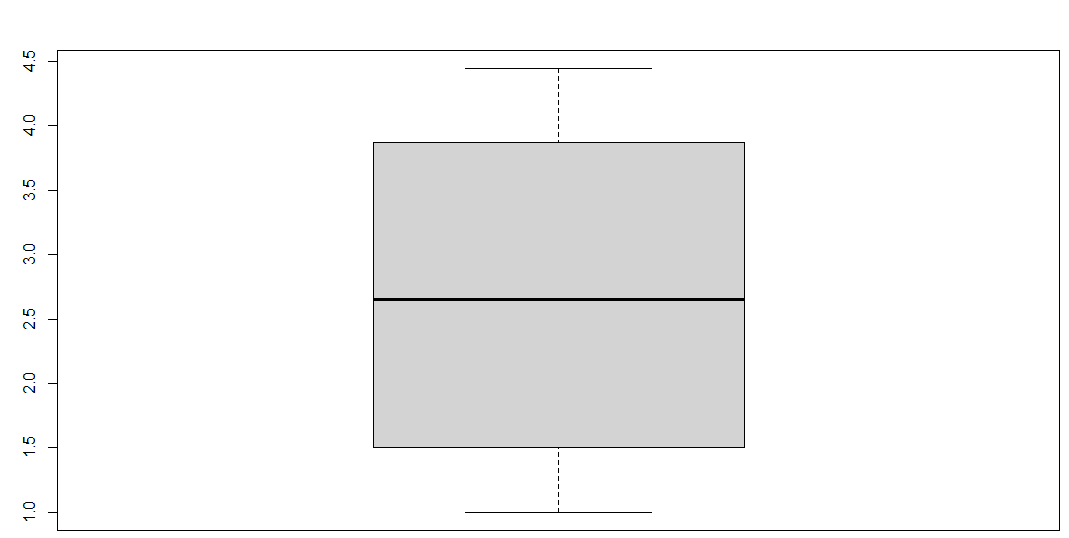
sd.result<-sd(x)

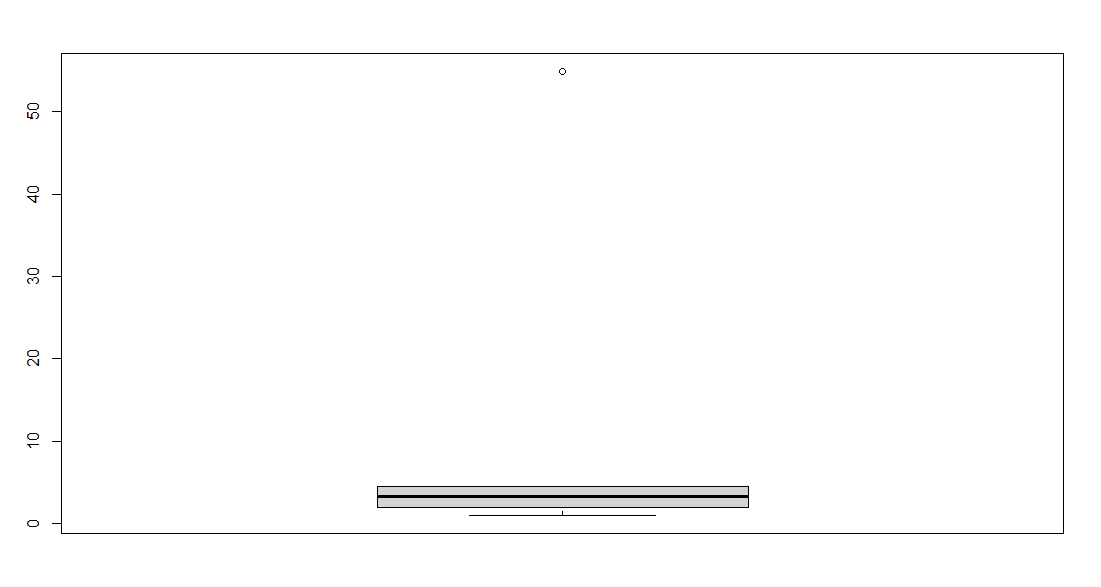
print(sd.result)

var.result<-var(x)

print(var.result)

**OUTPUT:**

****

****

**RESULT:**

This, our program has been successfully saved and executed.

**24. Correlation**

**AIM:**

To write an R program calculate the Pearson Correlation Coefficient between two variables using two formula methods and the built-in cor() function in R.

A).Pearsons Correlation Coefficient (Formula 1)

B).Correlation Coefficient using mean (Formula 2)

C).Built in function in R – Correlation

**Algorithm :**

**Step 1:** Start the Process and Create two numeric vectors x and y representing the variables.  
**Step 2:** Calculate the length n of the vectors using length().  
**Step 3:** Compute the numerator for Pearson correlation using formula:

(n×∑xy)−(∑x×∑y)

**Step 4:** Compute the denominator using formula:

**Step 5:** Calculate Pearson correlation coefficient as numerator divided by denominator.  
**Step 6:** Calculate the means of x and y using mean().  
**Step 7:** Calculate numerator as sum of product of deviations:

∑(xi​−xˉ)(yi​−yˉ​)

**Step 8:** Calculate denominator as square root of product of sums of squared deviations

for x and y. **Step 9:** Calculate correlation coefficient as numerator divided by denominator using the

mean-based formula  
**Step 10:** Use the built-in R function cor(x, y, method = "pearson") to calculate and print

thecorrelation

**Step 11:** stop the process

**Program:**

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

y <- c(15, 25, 35, 45, 55)

n <- length(x)

numerator <- (n \* sum(x \* y)) - (sum(x) \* sum(y))

denominator <- sqrt((n \* sum(x^2) - sum(x)^2) \* (n \* sum(y^2) - sum(y)^2))

r <- numerator / denominator

print(paste("Pearson Correlation (Formula 1):", r))

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

y <- c(15, 25, 35, 45, 55)

mean\_x <- mean(x)

mean\_y <- mean(y)

numerator <- sum((x - mean\_x) \* (y - mean\_y))

denominator <- sqrt(sum((x # Denominator: sqrt of sum of squares

- mean\_x)^2) \* sum((y - mean\_y)^2))

r2 <- numerator / denominator

print(paste("Correlation using Mean (Formula 2):", r2))

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

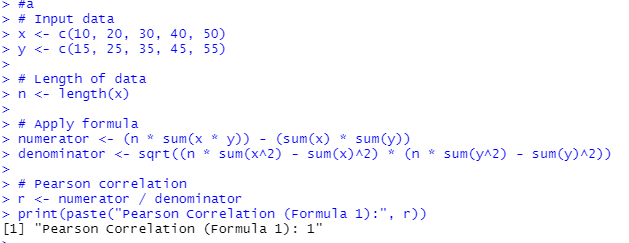
y <- c(15, 25, 35, 45, 55)

r\_builtin <- cor(x, y, method = "pearson")

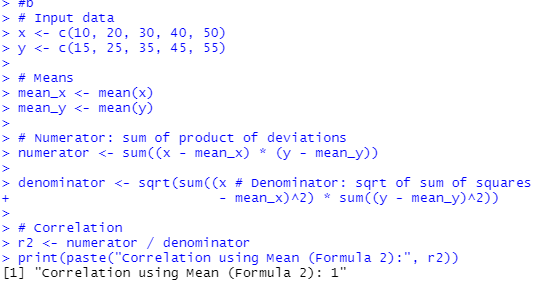
print(paste("Built-in cor() function:", r\_builtin))

**OUTPUT:**

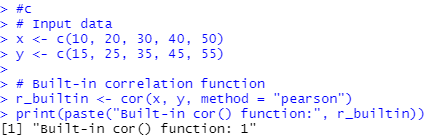
**Formula 1**

****

**Formula 2**

****

**Built in function in R**

****

**RESULT:**

This, our program has been successfully saved and executed

**25. Linear Regression Equation**

**Aim :**

To write an R program implement simple linear regression in R to analyze the relationship

between two variables and use the regression model for prediction and interpretation.

A).Y on X

B).X on Y

C).Built in Function in R using X and Y as vectors

D).Built in Function in R using CSV file

**Algorithm :**

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

**Step 2:** Define the independent variable x and dependent variable y as numeric vectors.

**Step 3:** Calculate the mean of x and y using mean() function.

**Step 4:** Compute required sums: ∑x, ∑y, ∑x², ∑y², and ∑xy.

**Step 5:** Calculate the regression coefficient (slope) using the formula:

b = n∑xy−∑x∑y​ / n∑x2−(∑x)2

**Step 6:** Form the linear regression equation: y=a+bxy = a + bxy=a+bx, where a=yˉ​−bxˉ

**Step 7:** Use lm(y ~ x) to fit the linear regression model in R.

**Step 8:** Use predict() function to estimate the value of y for new x.

**Step 9:** Plot the original data using plot() and add regression line using abline().

**Step 10:** Read data from a CSV file using read.csv() and repeat steps 2–9.

**Step 11:** Stop the process

**Program:**

x <- c(1, 2, 3, 4, 5, 6, 7)

y <- c(10, 20, 30, 40, 50, 60, 70)

model <- lm(y ~ x)

new\_data <- data.frame(x = 8)

print(predict(model, new\_data)) # Prediction when x = 8

plot(x, y, main = "If X Increases then Y also Increases", col = "blue", pch = 19)

abline(model, col = "red")

x <- c(1, 2, 3, 4, 5, 6, 7)

y <- c(10, 20, 30, 40, 50, 60, 70)

n <- length(x)

x\_bar <- mean(x)

y\_bar <- mean(y)

xs <- x \* x

print(xs)

Ex <- sum(x)

Ey <- sum(y)

Exs <- sum(xs)

ys <- y \* y

Eys <- sum(ys)

xy <- x \* y

Exy <- sum(xy)

numerator <- (n \* Exy - Ex \* Ey)

denominator <- (n \* Exs - Ex^2)

byx <- numerator / denominator

print(paste("byx =", byx))

lhs <- y - y\_bar

rhs <- byx \* (x - x\_bar)

print("lhs (y - ȳ):")

print(lhs)

print("rhs (byx \* (x - x̄)):")

print(rhs)

x\_bar <- mean(x)

y\_bar <- mean(y)

xs <- x \* x

ys <- y \* y

xy <- x \* y

Ex <- sum(x)

Ey <- sum(y)

Exy <- sum(xy)

Eys <- sum(ys)

numerator <- (n \* Exy - Ex \* Ey)

denominator <- (n \* Eys - Ey^2)

bxy <- numerator / denominator

print(paste("bxy =", bxy))

lhs <- x - x\_bar

rhs <- bxy \* (y - y\_bar)

print("lhs (x - x̄):")

print(lhs)

print("rhs (bxy \* (y - ȳ)):")

print(rhs)

x <- c(2005, 2010, 2015, 2020, 2023, 2026, 2029)

y <- c(36000, 38000, 40000, 38000, 42000, 44000, 43000)

model <- lm(y ~ x)

future\_data <- data.frame(x = 2025)

prediction <- predict(model, newdata = future\_data)

print(paste("Predicted sales for year 2025:", prediction))

plot(x, y, main = "Sales Based on Years vs Profit", col = "darkgreen", pch = 19)

abline(model, col = "red")

setwd("D:\\csv")

data <- read.csv("wine\_data.csv")

x <- data$fixed.acidity

y <- data$residual.sugar

relation <- lm(y ~ x)

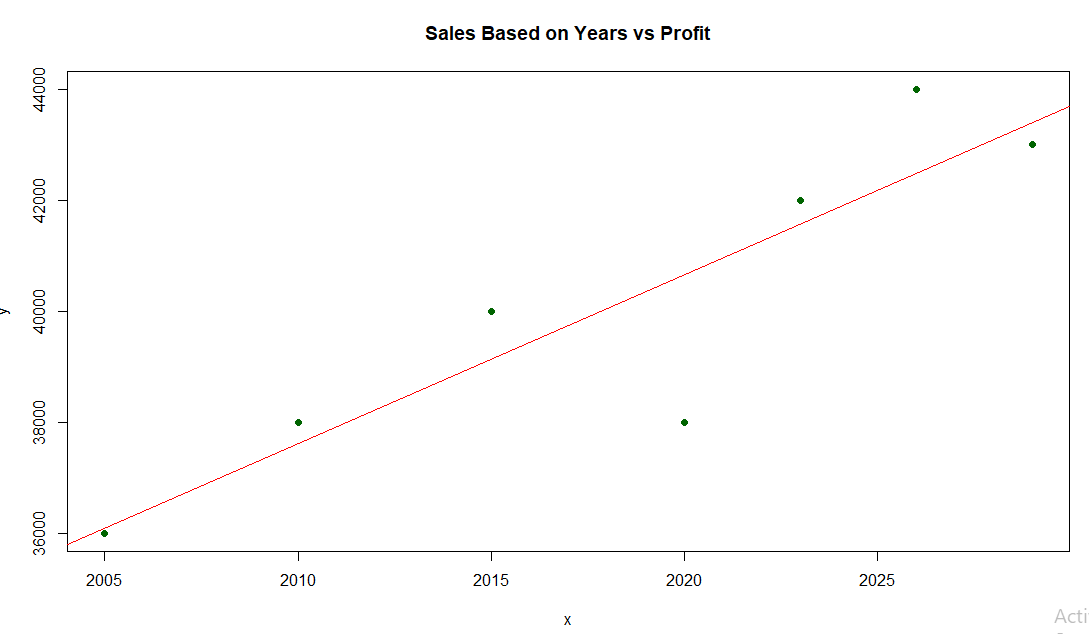
print(summary(relation))

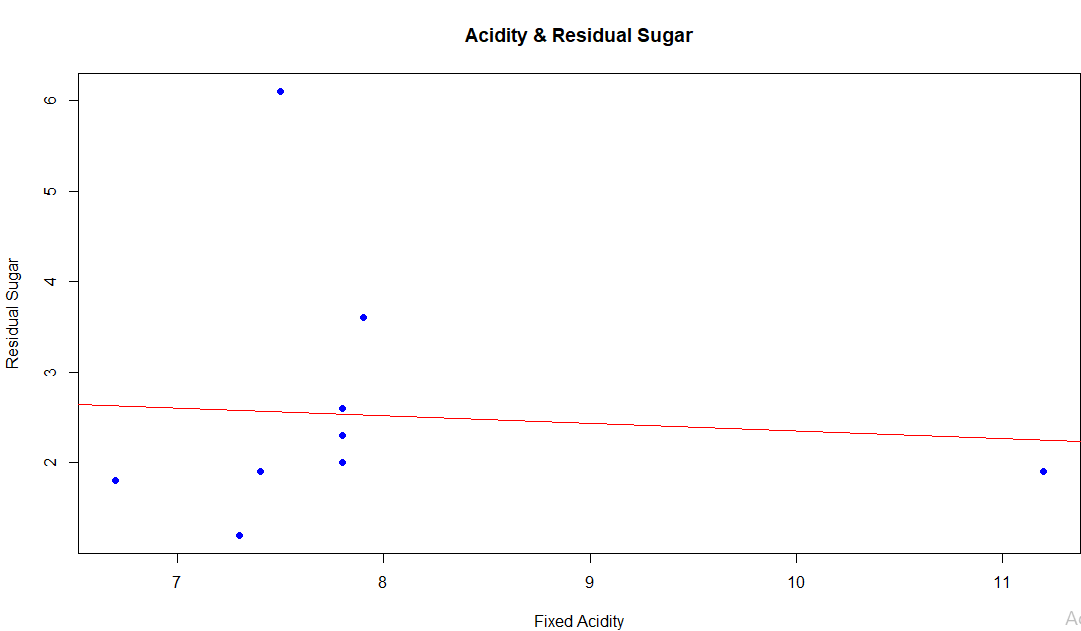
plot(x, y, col = "blue", main = "Acidity & Residual Sugar",

xlab = "Fixed Acidity", ylab = "Residual Sugar", pch = 20, cex = 1.5)

abline(relation, col = "red")

**OUTPUT:**





**RESULT:**

This, our program has been successfully saved and executed.

**26. Multiple Linear Regression**

**Aim:**

To write an R program to perform Multiple Linear Regression to predict a dependent variable using three independent variables, with data provided through:

a) Data Provided through Vector,  
b) Data Provided through build in table in R  
c) Data Provided through CSV file

**Algorithm:**

**Step1 :** Start the Process.

**Step2 :** Define or load the dataset (vectors, built-in, or CSV).

**Step3 :** Extract relevant columns (mpg, disp, hp, wt) into a data frame.

**Step4 :** Fit the regression model: lm(mpg ~ disp + hp + wt, data).

**Step5 :** Retrieve model coefficients (intercept and slopes).

**Step6 :**  Display the regression equation using the coefficients.

**Step7 :** Create new input data frame with predictor values for prediction.

**Step8 :** Use the model to predict mpg for new input data.

**Step9 :** Print the result.

**Step10 :** Stop the process.

**Program:**

1. **Data Provided Through Vector**

mpg <- c(21,21,22.8,21.4,18.7,18.1,14.3,24.4,22.8,19.2)

disp <- c(160,160,108,258,360,225,360,146.7,140.8,167.6)

hp <- c(110,110,93,110,175,105,245,62,95,123)

wt <- c(2.62,2.875,2.32,3.215,3.44,3.46,3.57,3.19,3.15,3.44)

input <- data.frame(mpg,disp,hp,wt)

print(head(input))

model <- lm(mpg~disp+hp+wt,data = input)

print(summary(model))

cat("### the cofficient value ###","\n")

a <- coef(model)[1]

print(a)

xdisp <- coef(model)[2]

xhp <- coef(model)[3]

xwt <- coef(model)[4]

print(xdisp)

print(xhp)

print(xwt)

print(summary(model))

paste("y~",a,"+",xdisp,"\*x1","+",xhp,"\*x2",xwt,"\*x3")

disp=221; hp=102; wt=2.91

a1 <- data.frame(disp,hp,wt)

result <- predict(model,a1)

print(result)

**B) Data Provided through build in table in R**

mtcars

print(mtcars)

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

print(head(input))

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

model<-lm(mpg~disp+hp+wt,data = input)

print(summary(model))

cat("### the cofficient value ###","\n")

a<-coef(model)[1]

print(a)

xdisp<-coef(model)[2]

xhp<-coef(model)[3]

xwt<-coef(model)[4]

print(xdisp)

print(xhp)

print(xwt)

print(summary(model))

paste("y~",a,"+",xdisp,"\*x1","+",xhp,"\*x2",xwt,"\*x3")

disp=221

hp=102

wt=2.91

a1<-data.frame(disp,hp,wt)

result<-predict(model,a1)

print(result)

**C) Data Provided through CSV File:**

setwd("C:/Users/MCA/Documents/R/R Programming/26. Multiple Linear Regression")

data <- read.csv("car\_data.csv")

print(head(data))

input <- data[,c("mpg","disp","hp","wt")]

print(head(input))

model <- lm(mpg~disp+hp+wt,data = input)

(summary(model))

cat("### the cofficient value ###","\n")

a <- coef(model)[1]

print(a)

xdisp <- coef(model)[2]

xhp <- coef(model)[3]

xwt <- coef(model)[4]

print(xdisp)

print(xhp)

print(xwt)

print(summary(model))

paste("y~",a,"+",xdisp,"\*x1","+",xhp,"\*x2",xwt,"\*x3")

disp=221; hp=102; wt=2.91

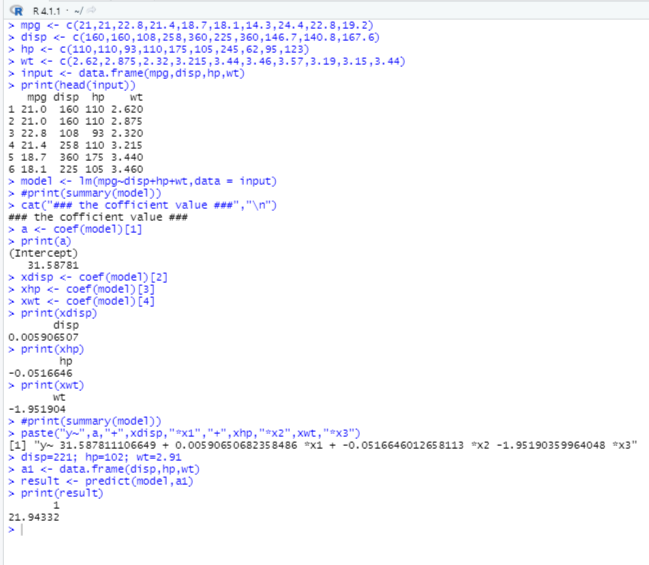
a1 <- data.frame(disp,hp,wt)

result <- predict(model,a1)

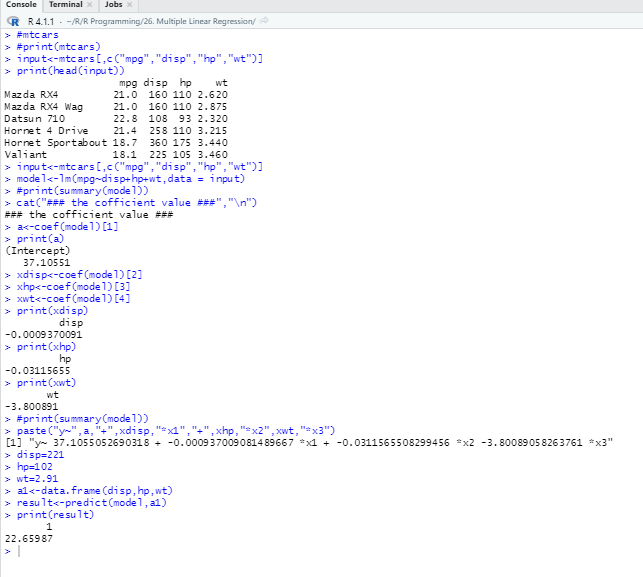
print(result)

**Output:**

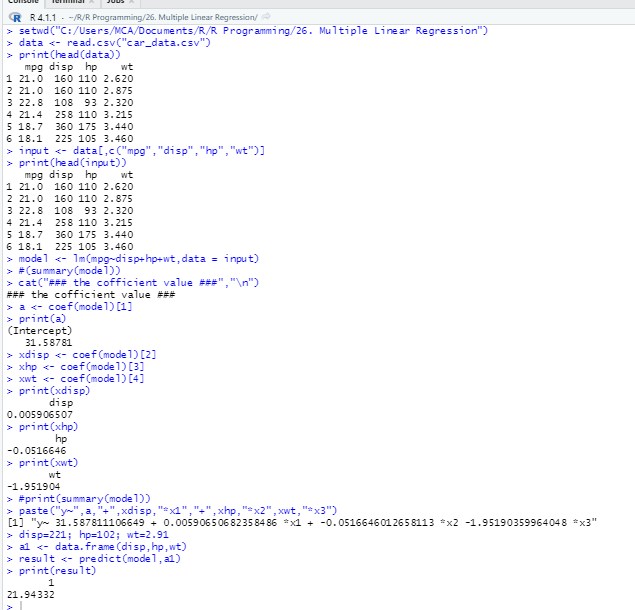
**A)Data Provided Through Vector**

****

**B)Data Provided through build in table in R**

****

**C)Data Provided through CSV File:**

****

**Result:**

Thus, our program has been successfully saved and executed.

**27.Logistic Regression**

**Aim:**

To write an R program to perform Logistic Regression analysis using data provided through vectors, data built within tables, and data imported from CSV files.

a) Data Provided through Vector  
b) Data Provided through build in table in R.  
c) Data Provided through CSV file

**Algorithm:**

**Step 1:** Start the program.

**Step 2:** Open RStudio and write the code using vectors, built-in datasets, or CSV files.

**Step 3:** Prepare the dataset with one dependent variable (0/1) and one or more independent variables.

**Step 4:** Apply the logistic regression formula:

logit(p)=ln(p/1−p​)=β0​+β1​X

**Step 5:** Build the logistic regression model using glm() with family="binomial".

**Step 6:** Check the model output using summary(model).

**Step 7:** Predict the probability for new values using predict(model, newdata, type="response") and draw the curve with plot() and curve().

**Step 8:** Stop the program

**Program:**

**A)Data Provided Through Vector**

# Data

hours\_studied <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

pass\_fail <- c(0, 0, 0, 0, 1, 1, 1, 1, 1, 1)

# Data frame

student\_data <- data.frame(hours\_studied, pass\_fail)

# Logistic regression model

model <- glm(pass\_fail ~ hours\_studied, data = student\_data, family = "binomial")

# Model summary

summary(model)

# Predict for 6 hours

new\_data <- data.frame(hours\_studied = 6)

predicted\_prob <- predict(model, newdata = new\_data, type = "response")

cat("Probability of passing if studied 6 hours:", round(predicted\_prob, 3), "\n")

# Plot

plot(hours\_studied, pass\_fail, pch = 19, col = "blue",

xlab = "Hours Studied", ylab = "Pass/Fail",

main = "Logistic Regression Curve")

# Add logistic regression curve

curve(predict(model, data.frame(hours\_studied = x), type = "response"),

from = 0, to = 11, add = TRUE, col = "red", lwd = 2)

**B)Data Provided through build in table in R**

model <- glm(vs ~ mpg, data = mtcars, family = "binomial")

p <- predict(model, data.frame(mpg = 25), type = "response")

cat("Prob at mpg 25:", round(p, 3), "\n")

plot(mtcars$mpg, mtcars$vs, pch = 19, col = "gray",

xlab = "MPG", ylab = "vs (0 = V, 1 = Straight)",

main = "Logistic Regression: vs ~ mpg")

curve(predict(model, data.frame(mpg = x), type = "response"),

from = min(mtcars$mpg), to = max(mtcars$mpg),

add = TRUE, col = "red", lwd = 2)

points(25, p, col = "blue", pch = 19, cex = 1.5)

**C)Data Provided through CSV File:**

setwd("C:/Users/MCA-017/Downloads/MCA/MCA/LogisticR")

# Load data from CSV

data <- read.csv("student\_logistic.csv.xlsx")

# Build logistic regression model

model <- glm(pass\_fail ~ hours\_studied, data = data, family = "binomial")

# Model summary

summary(model)

# Predict for 6 hours studied

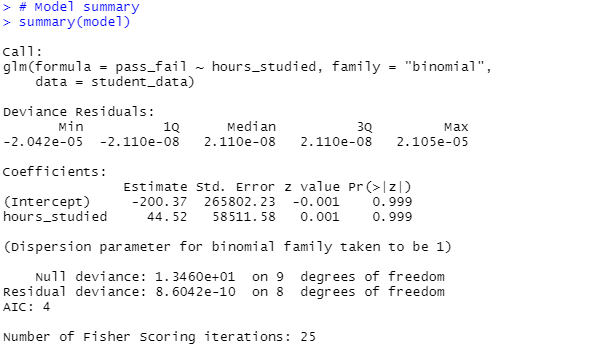
new\_data <- data.frame(hours\_studied = 6)

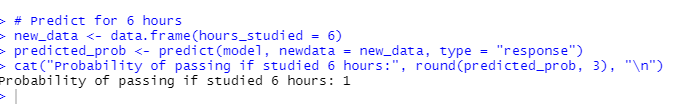
predicted\_prob <- predict(model, newdata = new\_data, type = "response")

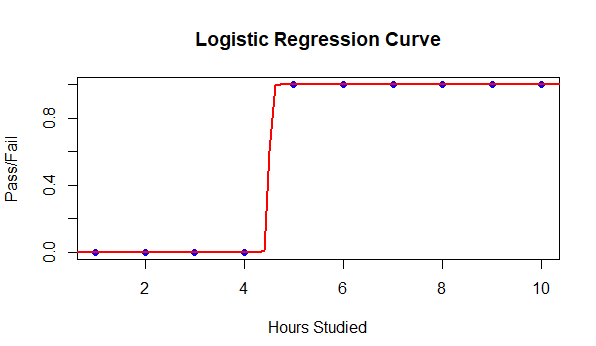
cat("Probability of passing if studied 6 hours:", round(predicted\_prob, 3))

**Output:**

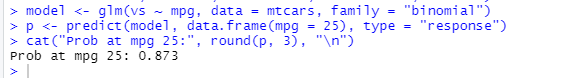
**A) Data Provided Through Vector**

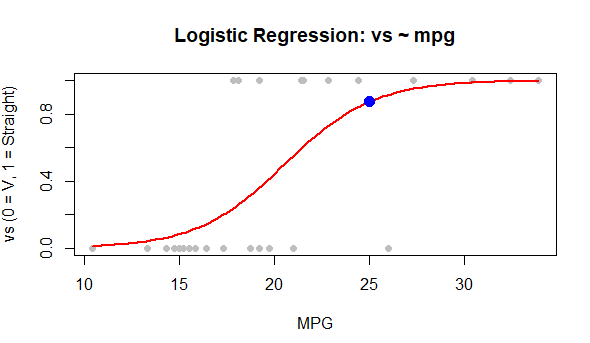




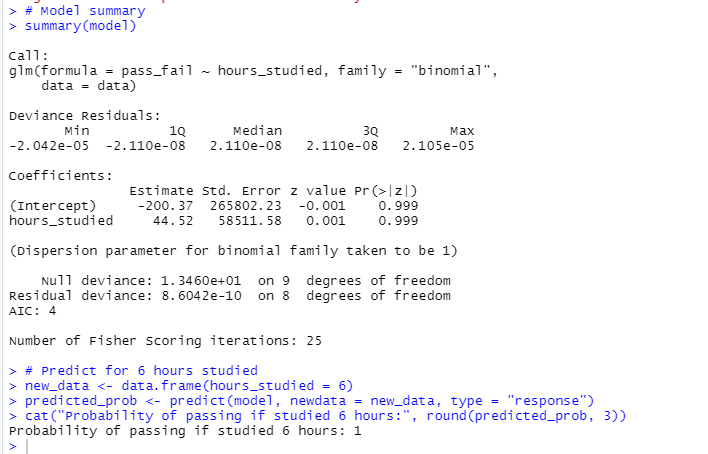


**B) Data Provided through build in table in R**





**C) Data Provided through CSV File**

****

**RESULT:**

This, our program has been successfully saved and executed.

**28.Non-Linear Least Squares**

**Aim:**

To write an R program to perform Non-Linear Least Squares analysis using data provided through vectors, data built within tables, and data imported from CSV files.

a) Data provided throughVectors  
b) Data provided throughbuilt in tables in R  
c) Data provided throughCSV files

**Algorithm:**

**Step 1:** Start the program.

**Step 2:** Open RStudio and write the code using vectors, built-in datasets, or CSV files.

**Step 3:** Prepare the dataset with independent variable (x) and dependent variable (y).

**Step 4:** Apply the non-linear least squares formula:

y=a×e^(b×x)

**Step 5:** Build the NLS model using nls() function with a given start value for parameters.

**Step 6:** Display the model results using summary(nls\_model).

**Step 7:** Plot the original data points using plot() and then add the fitted curve using lines().

**Step 8:** End the program.

**Program:**

1. **Data provided through vector**

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

y <- c(2.5, 3.6, 4.5, 6.1, 7.3)

# Fit non-linear model

nls\_model <- nls(y ~ a \* exp(b \* x), start = list(a = 1, b = 0.1))

# Plot original data points

plot(x, y, main = "Non-linear Least Squares Fit (Vectors)", xlab = "x", ylab = "y", pch = 19)

# Create predicted values from the model

x\_pred <- seq(min(x), max(x), length.out = 100)

y\_pred <- predict(nls\_model, newdata = data.frame(x = x\_pred))

# Add fitted curve

lines(x\_pred, y\_pred, col = "blue", lwd = 2)

**B) Data Provided through build in table in R**

data <- pressure

# Inspect the data

head(data)

# Fit non-linear model

# Model: pressure = a \* exp(b \* temperature)

nls\_model <- nls(pressure ~ a \* exp(b \* temperature), data = data, start = list(a = 1, b = 0.01))

# Summary of the model

summary(nls\_model)

# Plot original data points

plot(data$temperature, data$pressure, main = "Non-linear Least Squares Fit (pressure dataset)",

xlab = "Temperature", ylab = "Pressure", pch = 19)

# Create sequence for smooth curve

temp\_seq <- seq(min(data$temperature), max(data$temperature), length.out = 100)

# Predict pressure values from the model

predicted\_pressure <- predict(nls\_model, newdata = data.frame(temperature = temp\_seq))

# Add fitted curve to the plot

lines(temp\_seq, predicted\_pressure, col = "blue", lwd = 2)

**C)Data Provided through CSV File**

setwd("C:/Users/MCA-017/Downloads/MCA/MCA/NonLinear")

# Read the CSV file

data\_csv <- read.csv("sample\_data.csv")

# Fit non-linear model: y = a \* exp(b \* x)

nls\_model <- nls(y ~ a \* exp(b \* x), data = data\_csv, start = list(a = 1, b = 0.2))

# Summary of the model

summary(nls\_model)

# Plot original data points

plot(data\_csv$x, data\_csv$y, main = "Non-linear Least Squares Fit (CSV Data)", xlab = "x", ylab = "y", pch = 19)

# Create smooth sequence of x values for prediction

x\_pred <- seq(min(data\_csv$x), max(data\_csv$x), length.out = 100)

# Predict y values using the fitted model

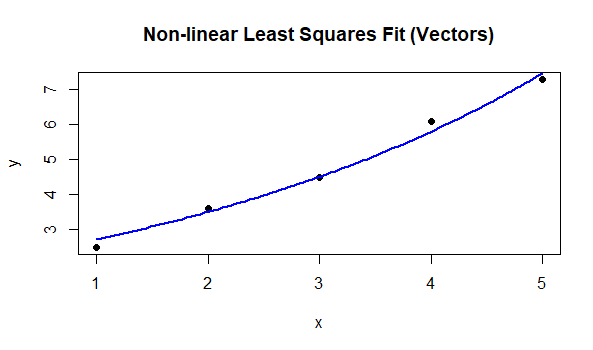
y\_pred <- predict(nls\_model, newdata = data.frame(x = x\_pred))

# Add fitted curve to plot

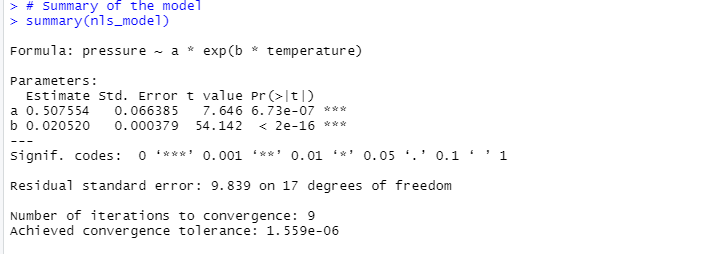
lines(x\_pred, y\_pred, col = "blue", lwd = 2)

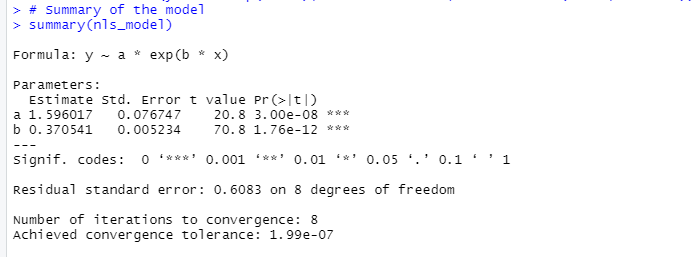
**Output:**

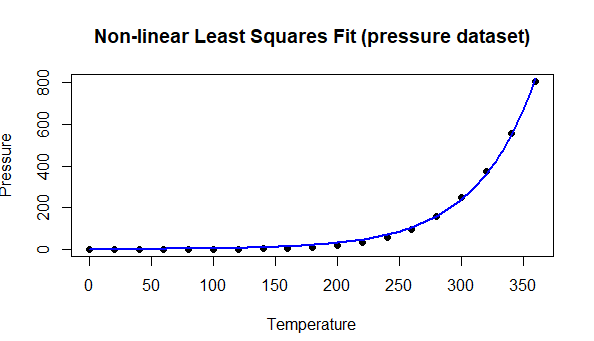
**A) Data provided through vector**

****

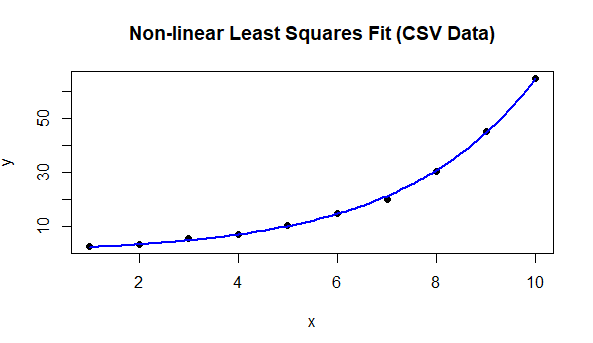
**B) Data Provided through build in table in R**

****

****

****

**C) Data Provided through CSV File**



**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

a) Using Formula

b) 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)+

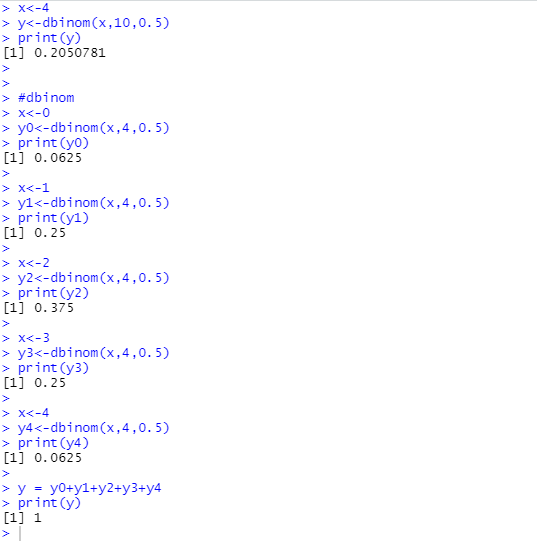
dbinom(3,size = 12,prob = 0.2)+

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

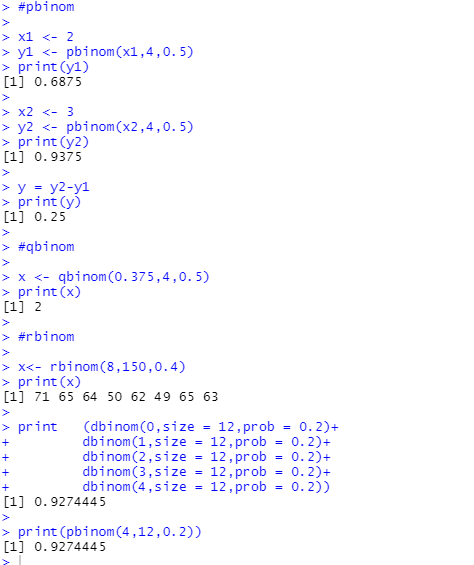
print(pbinom(4,12,0.2))

**OUTPUT :**

1. **Using Formula :**



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



**RESULT:**

Thus, the program has been successfully saved and executed.

**30. Normal Distribution**

**Aim:**

To write an R program to generate and visualize a normal distribution, and calculate its probability density and cumulative distribution values.

a) Using formula

b) Using dnorm,pnorm,qnorm,rnorm

**Algorithm:**

**Step 1:** Start and define the aim of studying Normal Distribution using R.

**Step 2:** Import or generate dataset and check for missing values/outliers.

**Step 3:** Compute descriptive statistics (mean, median, standard deviation and variance).

**Step 4:** Visualize data using histogram, density plot, and Q–Q plot.

**Step 5:** Perform normality test (e.g., Shapiro–Wilk) to check if data follows Normal distribution.

**Step 6:** Use Normal distribution functions (dnorm, pnorm, qnorm, rnorm) to calculate probabilities, densities, quantiles, and simulate values.

**Step 7:** Interpret the results and conclude whether the data fits the Normal distribution.

**Step 8:**  End the program

**Program :**

**a)** **Using formula**

x <- c(126, 176, 158, 180, 186, 166, 166, 164, 178, 170,

189, 195, 172, 187, 190, 186, 185, 168, 179, 178,

182, 179, 170, 175, 186, 159, 161, 178, 175, 185,

175, 162, 173, 172, 177, 175, 172, 177, 180)

mean\_x <- mean(x)

sd\_x <- sd(x)

print(paste("Mean:", mean\_x))

print(paste("Standard Deviation:", sd\_x))

b <- (x - mean\_x) \* (x - mean\_x) / (2 \* sd\_x \* sd\_x

k1 <- exp(-b)

k2 <- 1 / (sqrt(2 \* pi) \* sd\_x)

k3 <- k2 \* k1

plot(x, type = "o", col = "blue", main = "Original Data Plot", ylab = "Values", xlab = "Index")

plot(x, k3, type = "o", col = "red", main = "Normal Distribution", ylab = "Density", xlab = "x values")

y <- dnorm(x, mean = mean\_x, sd = sd\_x)

plot(x, y, type = "o", col = "green", main = "Normal Density from dnorm()", ylab = "Density", xlab = "x values")

**b)Normal Distribution by Using dnorm,pnorm,qnorm,rnorm**

getwd()

setwd("C:/Users/MCA/Documents")

getwd()

wine <- read.csv("winequality-red.csv", sep = ";")

print(wine)

x <- wine$fixed.acidity

print(head(x, 10))

print(mean(x))

print(sd(x))

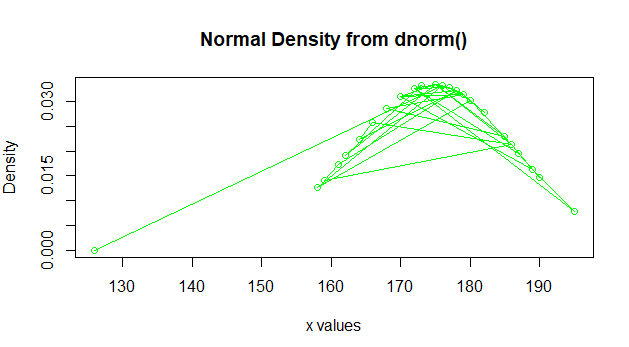
y <- dnorm(x, mean = mean(x), sd = sd(x))

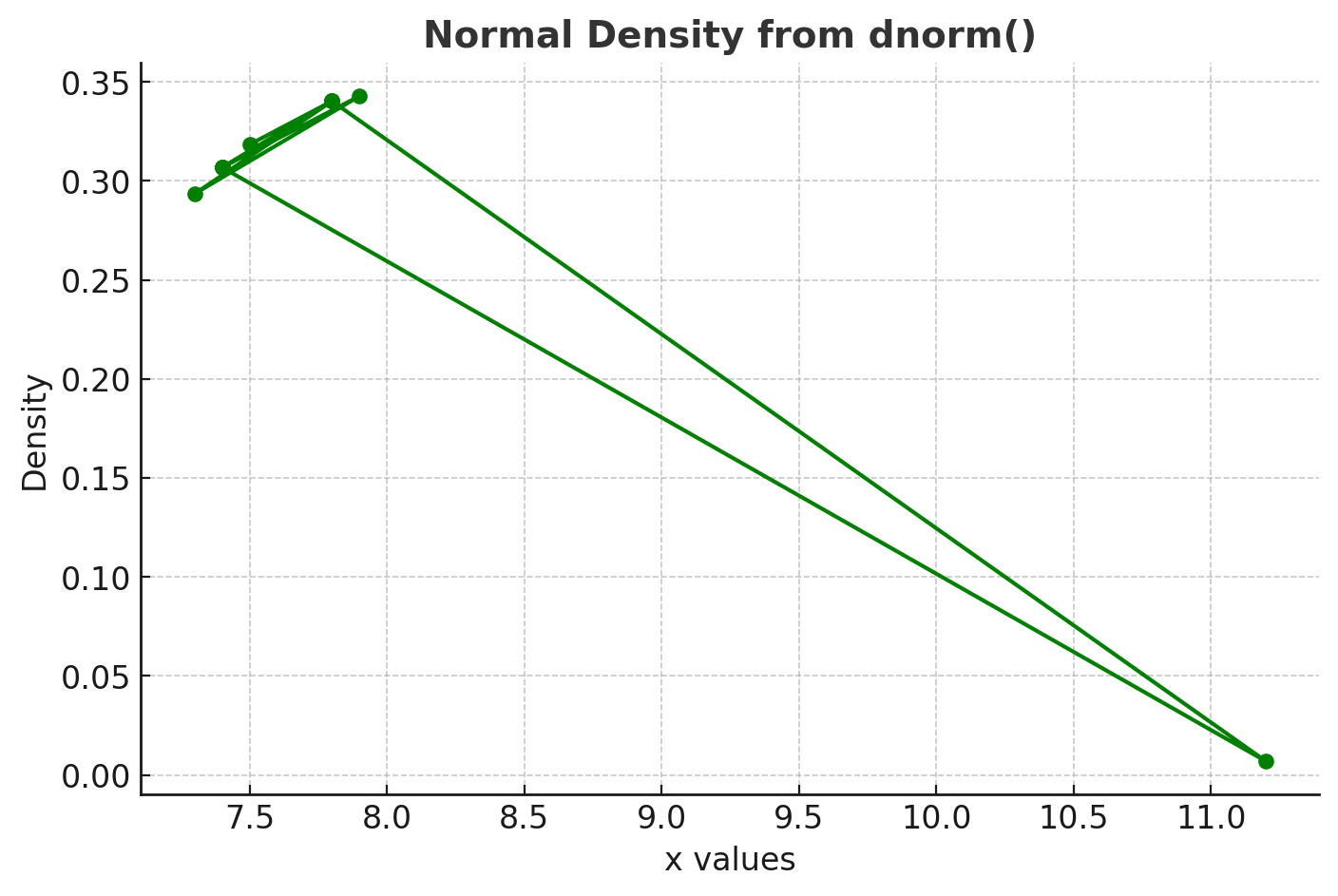
plot(x, y, type = "l", col = "darkred",

main = "Normal Distribution of Wine$Alcohol",

xlab = "Alcohol", ylab = "Density")

**OUTPUT:**

****

****

**RESULT:**

Thus, our program has been successfully saved and executed.

**31.** **Poisson Distribution**

**Aim :**

To write an R program to perform Poisson distribution calculations using **ppois**, **dpois**, and **rpois** functions for probabilities.

**Algorithm :**

**Step 1 :** Start the process to perform Poisson distribution calculations using ppois, dpois, and rpois functions.

**Step 2 :** Calculate cumulative probability using ppois() for given values with both tail options.

**Step 3 :** Calculate exact Poisson probability for a specific event count using the Poisson formula.

**Step 4 :** Determine probabilities for multiple event counts using Poisson probability function.

**Step 5 :** Generate random samples from the Poisson distribution for simulation purposes.

**Step 6 :** Compare probabilities and samples to verify they follow Poisson distribution characteristics.

**Step 7 :** End of the Program

**Program :**

#ppois

a = ppois(16,lambda = 12,lower.tail = TRUE)

b = ppois(16,lambda = 12,lower.tail = FALSE)

print(a+b)

#dpois

n=3000

p=0.001

r=6

lambda = n\*p

b<-exp(-lambda)\*lambda^6/factorial(6)

print(b)

dpois(6,lambda)

k1<-dpois(0,lambda)

k2<-dpois(1,lambda)

k3<-dpois(2,lambda)

k4<-dpois(3,lambda)

k5<-dpois(4,lambda)

k6<-dpois(5,lambda)

c<-paste(k1," ",k2," ",k3," ",k4," ",k5," ",k6," ")

print(c)

print(k1+k2+k3+k4+k5+k6)

ppois(3,lambda,lower.tail = TRUE)

ppois(3,lambda,lower.tail = FALSE)

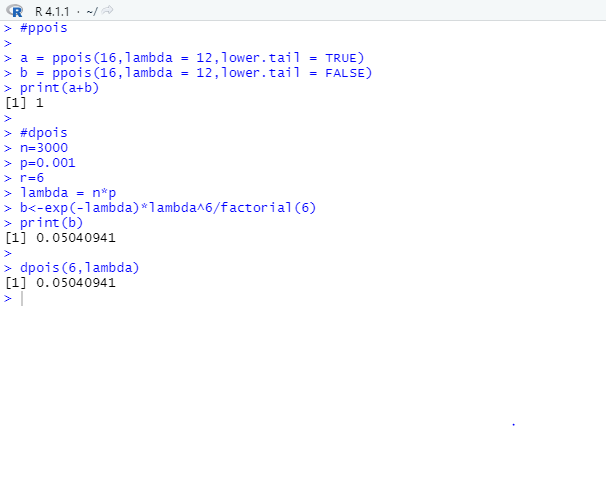
lambda <- 12

samples <- rpois(10, lambda)

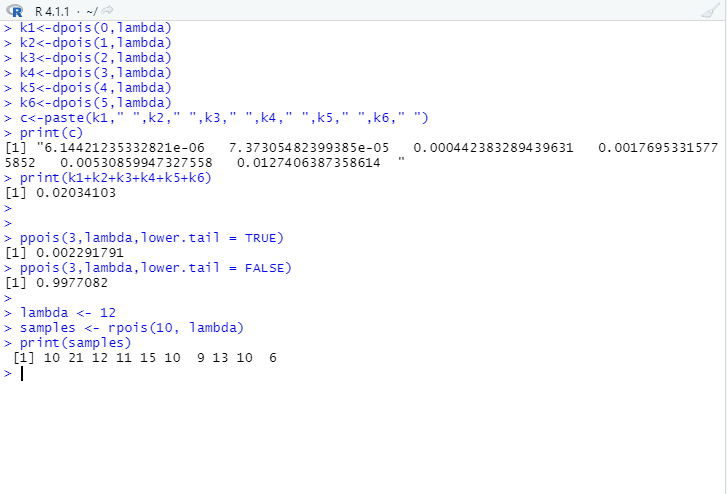
print(samples)

**Output :**

**a)Using Formula**



**b)Using dpois,ppois,qpois**



**RESULT:**

This, our program has been successfully saved and executed.

**32. Analysis of Variance using R**

**Aim:**

To write an R program to perform and interpret Analysis of Variance, in order to compare means across multiple groups and assess the significance of differences in a data science context

a) Without using Built in function

b) Using Built in function

**Algorithm:**

**Step 1 :** Start the Program

**Step 2:** To understand the concept of Analysis of Variance (ANOVA) in statistical data analysis.

**Step 3:** To manually compute sum of squares, degrees of freedom, and F-ratio.

**Step 4:** To implement ANOVA manually without using built-in functions in R.

**Step 5:** To calculate sum of squares, degrees of freedom, mean squares, and F-statistic manually.

**Step 6:** To learn how ANOVA is derived from basic mathematical formulas.

**Step 7:** To perform ANOVA using built-in R functions like aov() and summary().

**Step 8:** To validate the manual results by comparing with built-in function output.

**Step 9:** To apply ANOVA as a statistical technique for data science and decision-making tasks.

**Step 10:** End the program.

**Program:**

**a)Without using Built in function**

x1 <- c(8, 10, 7, 14, 11)

x2 <- c(7, 5, 10, 9, 9)

x3 <- c(12, 9, 13, 12, 14)

sum\_x1 <- sum(x1)

sum\_x2 <- sum(x2)

sum\_x3 <- sum(x3)

sum\_sq\_x1 <- sum(x1^2)

sum\_sq\_x2 <- sum(x2^2)

sum\_sq\_x3 <- sum(x3^2)

print(paste("Sum of x1:", sum\_x1, "Sum of x2:", sum\_x2, "Sum of x3:", sum\_x3))

print(paste("Sum of squares x1:", sum\_sq\_x1, "Sum of squares x2:", sum\_sq\_x2, "Sum of squares x3:", sum\_sq\_x3))

sum\_all <- sum\_x1 + sum\_x2 + sum\_x3

print(paste("Total sum:", sum\_all))

n1 <- length(x1)

n2 <- length(x2)

n3 <- length(x3)

N <- n1 + n2 + n3

ss\_treatment <- (sum\_x1^2 / n1) + (sum\_x2^2 / n2) + (sum\_x3^2 / n3) - (sum\_all^2 / N)

ss\_total <- sum\_sq\_x1 + sum\_sq\_x2 + sum\_sq\_x3 - (sum\_all^2 / N)

ss\_error <- ss\_total - ss\_treatment

df\_treatment <- 3 - 1

df\_error <- N - 3

ms\_treatment <- ss\_treatment / df\_treatment

ms\_error <- ss\_error / df\_error

F\_value <- ms\_treatment / ms\_error

cat("SS Treatment:", ss\_treatment, "\n")

cat("SS Error:", ss\_error, "\n")

cat("SS Total:", ss\_total, "\n")

cat("Degrees of Freedom (Treatment):", df\_treatment, "\n")

cat("Degrees of Freedom (Error):", df\_error, "\n")

cat("Mean Square Treatment:", ms\_treatment, "\n")

cat("Mean Square Error:", ms\_error, "\n")

cat("F value:", F\_value, "\n")

**b)Using Built in function**

x1 <- c(8, 10, 7, 14, 11)

x2 <- c(7, 5, 10, 9, 9)

x3 <- c(12, 9, 13, 12, 14)

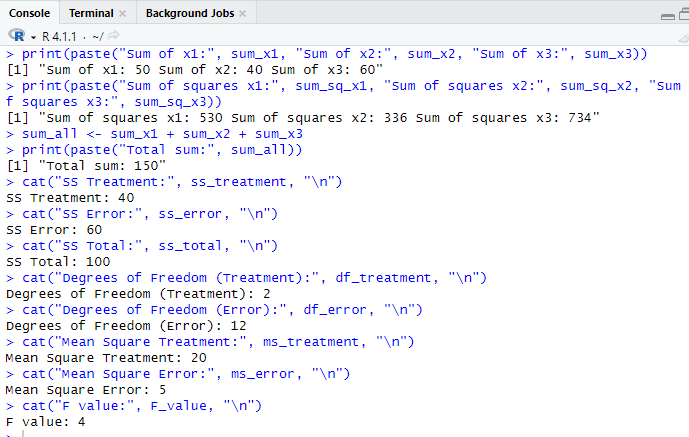
values <- c(x1, x2, x3)

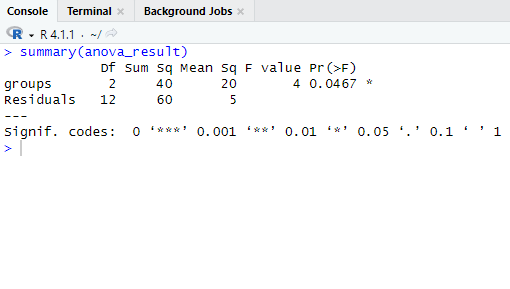
groups <- factor(rep(c("x1", "x2", "x3"), each = 5))

anova\_result <- aov(values ~ groups)

summary(anova\_result)

**OUTPUT:**

****

****

**RESULT:**

Thus, our program has been successfully saved and executed.

**33. Features of Numpy ,Mean,Median,mode and correlation coefficient using Numpy of Python.**

**Aim :**

To write a Python program using **NumPy** to demonstrate the features of NumPy and to compute **Mean, Median, Mode, and Correlation Coefficient** of given data.

**Algorithm :**

**Step 1 :** Start the process.

**Step 2 :** Import the necessary libraries. Import numpy as np for numerical operations

and stats from scipy for mode calculation.

**Step 3** :Create a NumPy array with sample numeric data.

**Step 4** :Compute the Mean of the array using np.mean().

**Step 5** : Compute the Median of the array using np.median().

**Step 6** : Compute the Mode of the array using stats.mode().

**Step 7** :Define two numeric arrays x and y for correlation analysis.

**Step 8** :Compute the Correlation Coefficient using np.corrcoef(x, y).

**Step 9** :Display the results of Mean, Median, Mode, and Correlation Coefficient.

**Step 10** :Stop the program.

**Program :**

import numpy as np

from scipy import stats

data = np.array([10, 20, 20, 40, 50, 50, 50, 70, 90])

mean\_val = np.mean(data)

print("Mean:", mean\_val)

median\_val = np.median(data)

print("Median:", median\_val)

mode\_val = stats.mode(data, keepdims=True)

print("Mode:", mode\_val.mode[0], " (Count:", mode\_val.count[0], ")")

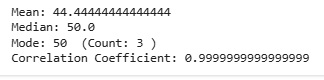
x = np.array([1, 2, 3, 4, 5])

y = np.array([2, 4, 6, 8, 10])

corr\_matrix = np.corrcoef(x, y)

print("Correlation Coefficient:", corr\_matrix[0, 1])

**Output :**

****

**Result :**

Thus, our program has been successfully saved and executed.

**34. Data Analysis using pandas of python having imdb\_movie\_data.**

**Aim :**

To write a program to analyze the IMDb movie dataset using Python's pandas library.

**Algorithm :**

**Step 1** **:** Start the process.

**Step 2 :** **Import necessary libraries,** Import pandas for data handling. Import pandas

library as pd.

**Step 3 :** Load the dataset using pd.read\_csv() and store it in movies\_df.

**Step 4 :** Display top rows using head() and bottom rows using tail().

**Step 5 :** Display dataset structure using shape and columns.

**Step 6 :** Find mean revenue using.

**Step 7 :** Filter movies , Released after 2005 and before 2010.With rating ≥ 8.0 With

revenue less than 25th quantile.

**Step 8 :** Display top rows of updated DataFrame.

**Step 9 :** Select rows using .loc[] and .iloc[] and end the program.

**Step 10 :** Stop the program.

**Program :**

# Mount Google Drive (for Google Colab)

from google.colab import drive

drive.mount('/content/drive')

# Import pandas

import pandas as pd

# Read CSV file

movies\_df = pd.read\_csv('/content/sample\_data/movies/Filtered\_Movies.csv')

# Display top rows

movies\_df.head(50) # Top 10 ordered by Ascending

movies\_df.tail(10) # Top 10 ordered by Descending

movies\_df.info() # Table presentation

movies\_df.shape # Number of rows & columns

movies\_df.columns # Column names

# Rename columns

movies\_df.rename(columns={

'Runtime (Minutes)': 'Runtime',

'Revenue (Millions)': 'Revenue\_million'

}, inplace=True)

# Handle missing data

movies\_df.columns

movies\_df.isnull()

movies\_df.isnull().sum()

movies\_df.dropna(inplace=True)

movies\_df.isnull().sum()

movies\_df.shape

# Calculate mean revenue and fill missing values

revenue = movies\_df['Revenue\_million']

revenue\_mean = revenue.mean()

print(revenue\_mean)

revenue.fillna(revenue\_mean, inplace=True)

revenue.head()

# Describe dataset

movies\_df.describe()

movies\_df['Genre'].describe()

print(movies\_df['Genre'].value\_counts().head(20))

movies\_df['Genre'].value\_counts().head(20)

# Subset of columns

subset = movies\_df[['Genre', 'Rating']]

subset.head()

# Strip whitespace from title column (replace 'Title' with exact column name if different)

movies\_df['Title'] = movies\_df['Title'].str.strip()

# Check if 'Prometheus' exists in titles and set index accordingly

if 'Prometheus' in movies\_df['Title'].values:

movies\_df = movies\_df.set\_index('Title')

prom = movies\_df.loc['Prometheus']

print(prom)

else:

print("Movie 'Prometheus' not found in the dataset.")

# Using iloc (by index)

prom = movies\_df.iloc[1]

print(prom)

# Filtering using multiple conditions

quartile = movies\_df['Revenue\_million'].quantile(0.25)

filtered = movies\_df[

(movies\_df['Year'] > 2005) &

(movies\_df['Year'] < 2010) &

(movies\_df['Rating'] > 8.0) &

(movies\_df['Revenue\_million'] < quartile)]

# Apply custom function on ratings

def rating\_function(x):

if x > 8.0:

return "Good"

else:

if (x >= 7.0) & (x < 8.0):

return "Better"

else:

return "bad"

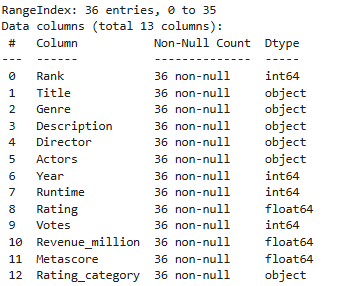
movies\_df['Rating\_category'] = movies\_df['Rating'].apply(rating\_function)

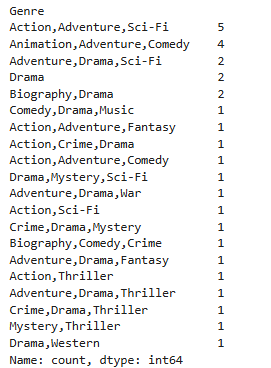
movies\_df.head()

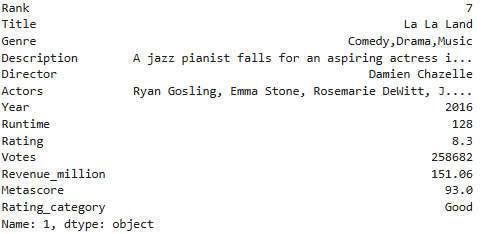
# Filter by Rating category and Revenue

movies\_df[(movies\_df['Rating\_category'] == 'Good') & (movies\_df['Revenue\_million'] > 100)]

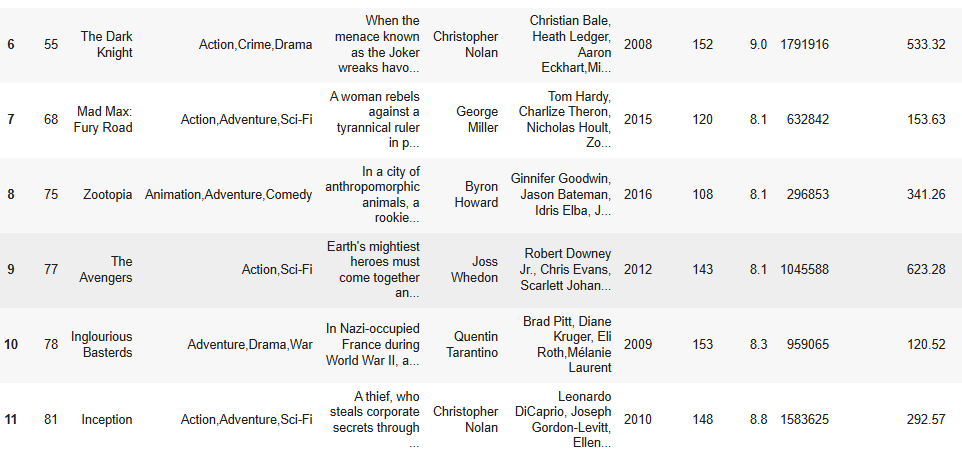
**Output :**

****

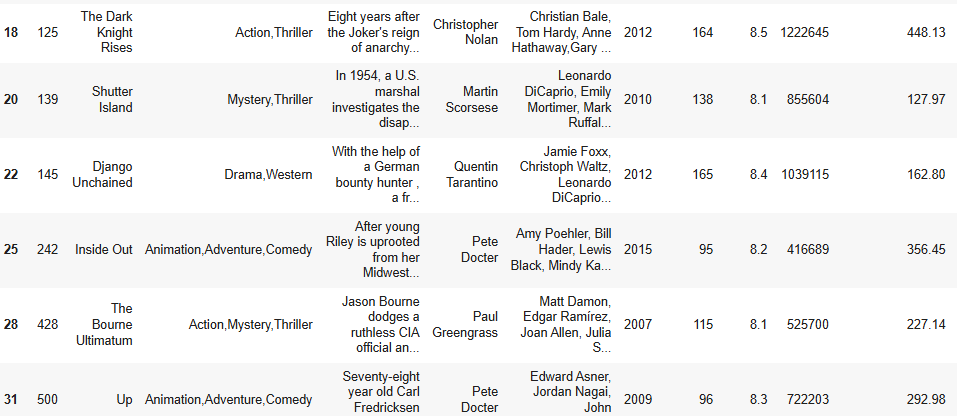
****

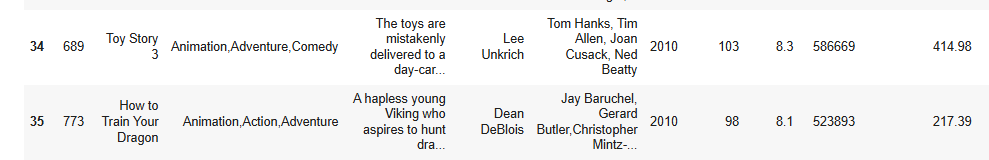
****

****

****

****

****

****

**Result :**

Thus, our program has been successfully saved and executed.

**35. Normal Distribution Analysis of any CSV file using Python.**

**Aim:**

To write a Python program that reads a CSV file, analyzes the distribution of a selected numeric column, fits a normal distribution to the data, plots the histogram with the normal curve, and performs a statistical test to check for normality.

**Algorithm:**

**Step 1:** Start the program.

**Step** **2:** Import required libraries (pandas, numpy, matplotlib, scipy.stats).

**Step** **3:** Read the CSV file into a DataFrame.

**Step** **4:** Select the desired numeric column from the DataFrame.

**Step** **5:** Remove any missing values from the selected column.

**Step** **6:** Plot a histogram of the column with density normalization.

**Step** **7:** Fit a normal distribution to the data (calculate mean and std).

**Step 8:** Plot the normal distribution curve over the histogram.

**Step** **9:** Perform the Shapiro-Wilk test to check for normality.

**Step** **10:** Print the Result.

**Step 11:** End the Program.

**Program:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import norm, shapiro

filepath = '/content/NormalDistribution/guna.csv'

df = pd.read\_csv(filepath , encoding= 'ISO-8859-1')

print("Columns in the dataset:")

print(df.columns)

column = 'Total'

print(df[column].head())

data = df[column].dropna()

plt.hist(data, bins=30, density=True, alpha=0.6, color='g')

mu, std = norm.fit(data)

xmin, xmax = plt.xlim()

x = np.linspace(xmin, xmax, 100)

p = norm.pdf(x, mu, std)

plt.plot(x, p, 'k', linewidth=2)

title = f"Fit results: mu = {mu:.2f},  std = {std:.2f}"

plt.title(title)

plt.show()

stat, p\_value = shapiro(data)

print(f'Shapiro-Wilk Test: Statistics={stat:.3f}, p={p\_value:.3f}')

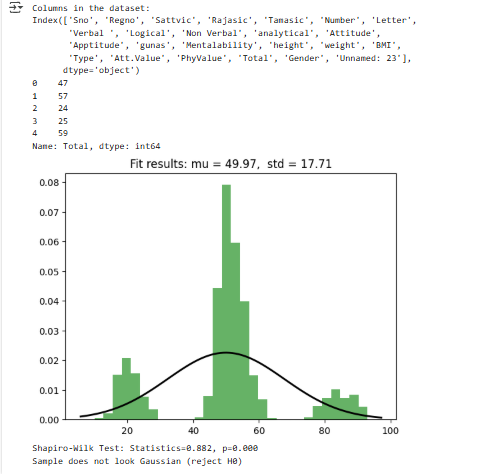
if p\_value > 0.05:

    print('Sample looks Gaussian (fail to reject H0)')

else:

    print('Sample does not look Gaussian (reject H0)')

**Output:**

****

**Result:**

Thus, our program has been successfully saved and executed.

**36. Analysis of Variance using python**

**Aim:**

To perform a **ANOVA** to determine if there are statistically significant differences between the means of three independent groups.

**Algorithm:**

**Step 1:** Start the Process.

**Step** **2:** Define the input groups containing numerical data.

**Step** **3:** Calculate the mean of each group.

**Step** **4:** Flatten all groups into a single list to get all values.

**Step** **5:** Calculate the grand mean of all combined values.

**Step** **6:** Compute the sum of squares between groups (SSB).

**Step** **7:** Compute the sum of squares within groups (SSW).

**Step** **8:** Calculate degrees of freedom for between and within groups.

**Step** **9:** Calculate mean squares for between (MSB) and within (MSW).

**Step** **10:** Compute the F-statistic using MSB divided by MSW.

**Step 11:** Print the result.

**Step 12:** Stop the process.

**Program:**

group1 = [85, 90, 88, 75, 95]

group2 = [78, 82, 84, 88, 90]

group3 = [92, 94, 89, 96, 91]

groups = [group1, group2, group3]

group\_means = [sum(g) / len(g) for g in groups]

all\_values = sum(groups, [])

grand\_mean = sum(all\_values) / len(all\_values)

ss\_between = sum(len(g) \* (group\_mean - grand\_mean) \*\* 2 for g, group\_mean in zip(groups, group\_means))

ss\_within = sum(sum((x - group\_mean) \*\* 2 for x in g) for g, group\_mean in zip(groups, group\_means))

df\_between = len(groups) - 1

df\_within = len(all\_values) - len(groups)

ms\_between = ss\_between / df\_between

ms\_within = ss\_within / df\_within

f\_statistic = ms\_between / ms\_within

print(f"Group Means: {group\_means}")

print(f"Grand Mean: {grand\_mean:.2f}")

print(f"SS Between: {ss\_between:.2f}")

print(f"SS Within: {ss\_within:.2f}")

print(f"DF Between: {df\_between}")

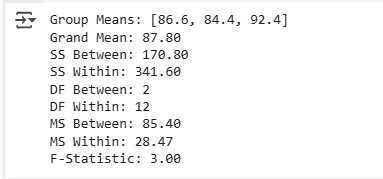
print(f"DF Within: {df\_within}")

print(f"MS Between: {ms\_between:.2f}")

print(f"MS Within: {ms\_within:.2f}")

print(f"F-Statistic: {f\_statistic:.2f}")

**Output:**

****

**Result:**

Thus, our program has been successfully saved and executed.

**37. Poisson Distribution Using R and Python**

**Aim :**

To write a program to implement the Poisson Distribution using R and Python, and visualize the results.

**Algorithm :**

**Step 1 :** Start the process.

**Step 2 :** Import required libraries, Python: math, matplotlib.pyplotR: No external library

needed for basic Poisson (dpois, barplot)

**Step 3 :** Define a factorial function (only for Python if not using built-in factorial).

**Step 4**  **:** Ask the user to choose one of the two input methods:Method 1: Input values for n

(number of trials) and p (probability of success)Method 2: Directly input the value of

λ (lambda)

**Step 5 :** Ask for the number of x values (r) to evaluate (range: 0 to r)

**Step 6**  **:** For each integer x from 0 to r, Calculate the Poisson probability using the formula

**Step 7**  **:** Store x and corresponding P(x) values in lists or vectors.

**Step 8**  **:** Plot the Poisson distribution using a bar graph to visualize the probability

distribution.

**Step 9 :** Display or return the probability table and the plot.

**Step 10**  **:** Stop the program.

**Program :**

import math

import matplotlib.pyplot as plt

def fact(n):

if n == 0:

return 1

elif n == 1:

return 1

else:

return n \* fact(n - 1)

x = int(input("Press 1 for n, p, c value or 2 for lambda value: "))

if x == 1:

n = int(input("Enter the value of n: "))

p = float(input("Enter the value of success p: "))

r = int(input("Enter the value of r: "))

lambda1 = n \* p

else:

lambda1 = float(input("Enter the value of lambda: "))

r = int(input("Enter the value of r: "))

# Compute Poisson Distribution

x = []

t = []

for i in range(0, r + 1):

t.append(i)

x.append(i)

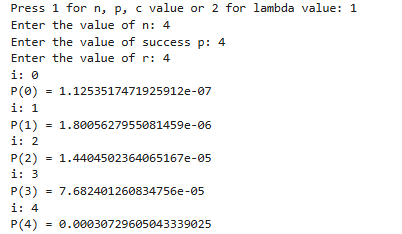
print(f"i: {i}")

x[i] = math.exp(-lambda1) \* pow(lambda1, i) / fact(i)

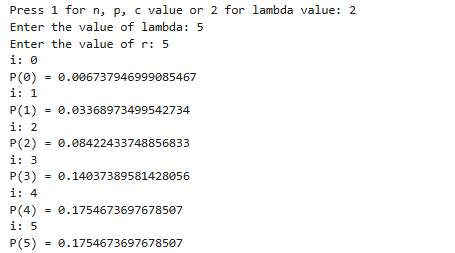
print(f"P({i}) = {x[i]}")

**Output :**

**N , P , Value :**



**LAMBDA VALUE :**



**Result :**

Thus, our program has been successfully saved and executed.

1. **Decision Tree using R and Python**

**Aim :**

To write a program in R and Python to calculate Decision Tree.

A) Decision Tree Using Python

B) Decision Tree Using R

C) Hierachical Cluster Using R

**Algorithm :**

**Step 1:** Start the process of implementing machine learning techniques using Python and R.

**Step 2:** Import the required libraries In Python: pandas, sklearn.tree, matplotlib.

**Step 3:** Load the dataset (e.g., Titanic dataset for Decision Tree, numeric dataset for clustering).

**Step 4:** Preprocess the dataset by handling missing values, encoding categorical variables (Python: map(), R: factor()), and standardizing features for clustering.

**Step 5:** For Decision Tree in Python: Select input features and target variable, train the model using DecisionTreeClassifier(), and visualize the tree.

**Step 6:** Show the trained decision tree in Python and R.Show feature importance in Python..

**Step 7:** End the program

**Program:**

**a)Using R:**

install.packages("party")

library(party)

print(head(readingSkills))

print(readingSkills)

library(party)

input.dat <- readingSkills[c(1:150),]

output.tree <- ctree(

nativeSpeaker ~ age + shoeSize + score,

data = input.dat)

plot(output.tree)

output.tree <- ctree(

nativeSpeaker ~ age + score,

data = input.dat)

plot(output.tree)

output.tree <- ctree(

nativeSpeaker ~ shoeSize + score,

data = input.dat)

plot(output.tree)

library(party)

traindata <- read.table('C:/Users/MCA-010/Desktop/Piere/train.csv', sep=",", header= TRUE)

head(traindata)

output.tree <- ctree(

Survived~Pclass,

data = traindata)

plot(output.tree)

output.tree <- ctree(

Survived~Pclass+Parch,

data = traindata)

plot(output.tree)

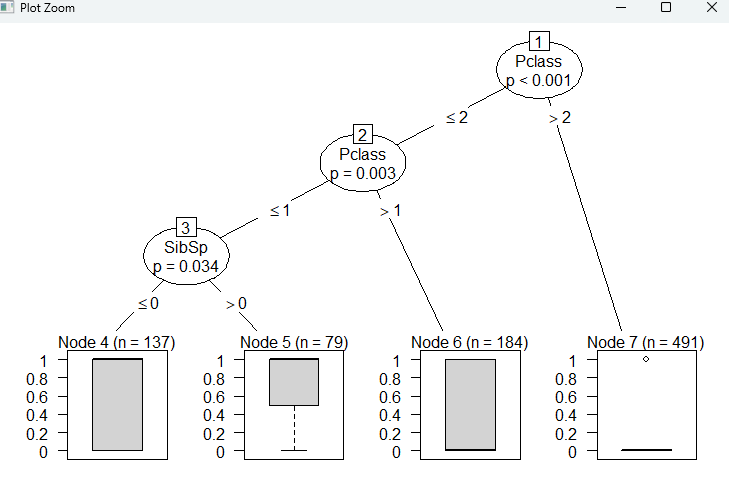
output.tree <- ctree(

Survived~Pclass+SibSp,

data = traindata)

plot(output.tree)

**OUTPUT:**



**Program:**

**b)Using Python:**

import math

from scipy.stats import poisson

p1 = poisson.cdf(16, mu=12)

print("P(X ≤ 16) =", p1)

p2 = poisson.sf(16, mu=12)

print("P(X > 16) =", p2)

lamda1 = 3000 \* 0.001

k = math.exp(-lamda1) \* lamda1\*\*6 / math.factorial(6)

print("Manual formula P(X=6):", k)

k\_scipy = poisson.pmf(6, mu=lamda1)

print("scipy P(X=6):", k\_scipy)

k1 = poisson.pmf(0, mu=lamda1)

k2 = poisson.pmf(1, mu=lamda1)

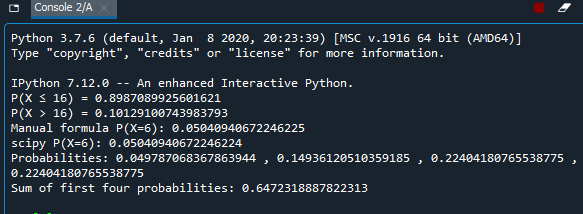
k3 = poisson.pmf(2, mu=lamda1)

k4 = poisson.pmf(3, mu=lamda1)

print("Probabilities:", k1, ",", k2, ",", k3, ",", k4)

print("Sum of first four probabilities:", k1 + k2 + k3 + k4)

**OUTPUT:**



**Program:**

**c)Hierarchical Cluster using R:**

par(mfrow=c(1,3))

x<-cbind(c(-1.4806,1.5772,-0.9567,-0.92,-1.9976,-0.2723,-0.3153),c(-0.6283,-0.1065,0.428,-0.7777,-1.2939,-0.7796,0.012))

plot(x, pch = as.character(1:nrow(x)), asp = 1)

library(cluster)

mc1 <- mutualCluster(x, plot=TRUE)

dist(x)

hc <- hclust(dist(x))

plot(hc)

install.packages("cluster")

library(cluster)

data <- data.frame(

X = c(1, 2, 3, 5, 8, 8, 9, 10),

Y = c(1, 1.5, 2, 8, 10, 11, 12, 8))

print(data)

plot(data, pch = as.character(1:nrow(data)), asp = 1)

dist\_matrix <- dist(data, method = "euclidean")

hc <- hclust(dist\_matrix, method = "complete") # complete linkage

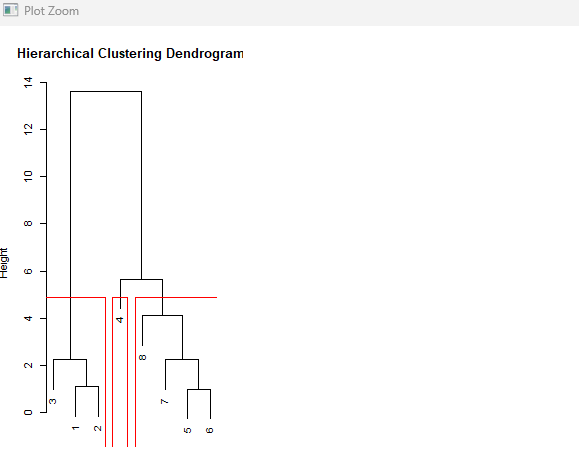
plot(hc, main = "Hierarchical Clustering Dendrogram", xlab = "", sub = "", cex = 0.9)

rect.hclust(hc, k = 3, border = "red")

clusters <- cutree(hc, k = 3)

print(clusters)

**OUTPUT:**



**RESULT:**

Thus, our program has been successfully saved and executed.

1. **Chi – Square Test Using Python and R**

**Aim :**

To write a program in R and Python to perform Chi –Square Test.

A) Chi - square Test for single vector

B) Chi - square Test for two-dimensional vector

C) Chi - square Test Using R

**Algorithm :**

**Step 1:** Start the process of implementing Chi-Square statistical tests using Python and R.

**Step 2:** Import the required libraries. In Python: numpy, scipy.stats.

In R: built-in function chisq.test()..

**Step 3:** Load or define the dataset.

**Step 4:** Preprocess the data by ensuring observed and expected frequencies are correctly specified.

**Step 5:** Display results including Chi-Square statistic, degrees of freedom, and p-value. Compare the p-value with the chosen significance level (α) to accept or reject the null hypothesis.

**Step 6:** End the program

**Program:**

**a)Chi - square Test for single vector:**

import numpy as np

from scipy.stats import chi2\_contingency

observed = [10, 8, 9, 10, 2, 11]

expected = [0.5]\*6

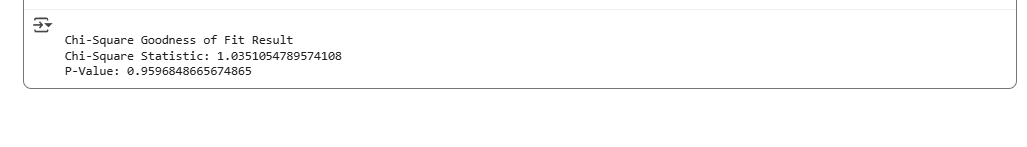
chi2\_stat, p\_value = chi2\_contingency([observed, expected])[:2]

print('\nChi-Square Goodness of Fit Result')

print("Chi-Square Statistic:", chi2\_stat)

print("P-Value:", p\_value)

**OUTPUT:**

****

**Program:**

**b) Chi - square Test for two-dimensional vector:**

import pandas as pd

from scipy.stats import chi2\_contingency

data = {'ProductA': [20, 30, 25],

'ProductB': [25, 30, 20]}

df = pd.DataFrame(data, index=['18-25', '26-35', '36-45'])

print('Contingency Table:')

print(df)

chi2\_stat, p\_value, dof, expected = chi2\_contingency(df)

print('\nChi-Square Test Result:')

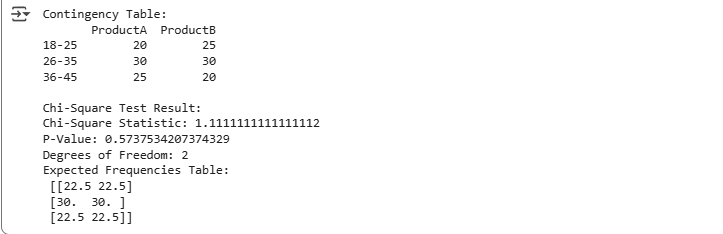
print("Chi-Square Statistic:", chi2\_stat)

print("P-Value:", p\_value)

print("Degrees of Freedom:", dof)

print("Expected Frequencies Table:\n", expected)

**OUTPUT:**



**Program:**

**c)Chi - square Test Using R:**

observed <- c(10, 8, 9, 10, 2, 11)

expected <- rep(1/6, 6)

print(chisq.test(x = observed, p = expected))

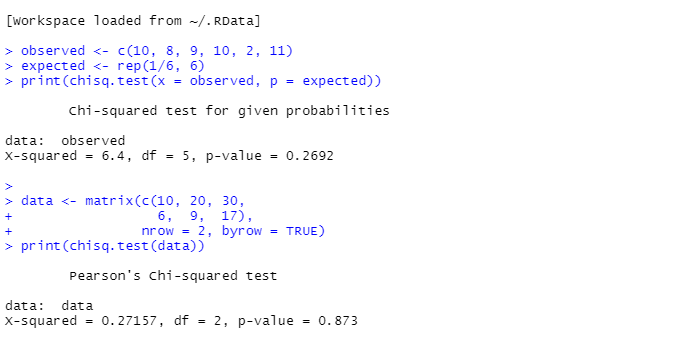
data <- matrix(c(10, 20, 30,

6, 9, 17),

nrow = 2, byrow = TRUE)

print(chisq.test(data)

**OUTPUT:**



**RESULT:**

This, our program has been successfully saved and executed.

**40. Times series analysis**

**Aim:**

To write the Python program for the **time series analysis** on the shampoo sales data by applying:

1. Moving average.
2. Auto correlation & auto correlation.
3. ARIMA for forecast.
4. find(p,d,q) for fitting suitable ARIMA for least mean square error.

**Algorithm:**

**Step1 :** Start the process.

**Step2 :** Import necessary libraries (pandas, numpy, ARIMA, mean\_squared\_error).

**Step3 :** Split shampoo sales data into training (first 25 rows) and testing (next 11 rows).

**Step4 :** Define ranges for ARIMA parameters p, d, q (0 to 2).

**Step5 :** For each (p, d, q) combination, initialize empty predictions and set history to training data.

**Step6 :** For each test point, fit ARIMA on history, forecast next value, append prediction, and update history with actual value.

**Step7 :** Calculate RMSE between test actual values and predictions.

**Step8 :** Print the current (p, d, q) and its RMSE.

**Step9 :** Select the ARIMA order with the lowest RMSE as the best model.

**Step10 :** Print the result.

**Step11 :** Stop the process.

**Program:**

**a) Moving average.**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import mean\_squared\_error

from google.colab import drive

drive.mount('/content/drive')

shampoo = pd.read\_csv('/content/drive/My Drive/shampoo.csv')

print(shampoo.size)

print(shampoo.describe())

shampoo\_ma = shampoo['Sales'].rolling(window=15).mean()

print(shampoo\_ma)

shampoo\_ma.plot()

plt.title("Moving Average (window=15)")

plt.show()

shampoo\_base = pd.concat([shampoo['Sales'], shampoo['Sales'].shift(4)], axis=1)

shampoo\_base.columns = ['Actualsales', 'Forecastsales']

shampoo\_base.dropna(inplace=True)

print(shampoo\_base.head())

shampoo\_base.dropna(inplace=True)

shampoo\_base.plot()

plt.title("Actual vs Forecast Sales")

plt.show()

mse\_shampoo\_error = mean\_squared\_error(shampoo\_base.Actualsales, shampoo\_base.Forecastsales)

print(mse\_shampoo\_error)

rmse = np.sqrt(mse\_shampoo\_error)

print(rmse)

**b) Auto correlation & auto correlation.**

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

import matplotlib.pyplot as plt

plot\_acf(shampoo['Sales'])

plt.show()

plot\_pacf(shampoo['Sales'])

plt.show()

**c) ARIMA for forecast.**

from statsmodels.tsa.arima.model import ARIMA

import pandas as pd

import numpy as np

from sklearn.metrics import mean\_squared\_error

shampoo\_train = shampoo['Sales'][:25]

shampoo\_test = shampoo['Sales'][25:36]

shampoo\_model = ARIMA(shampoo\_train, order=(1,2,1))

shampoo\_model\_fit = shampoo\_model.fit()

shampoo\_model\_fit.summary()

shampoo\_forecast = shampoo\_model\_fit.forecast(steps=11)

print(np.sqrt(mean\_squared\_error(shampoo\_test, shampoo\_forecast)))

df = pd.DataFrame(shampoo\_model\_fit.predict(start=1, end=36))

df1 = pd.DataFrame(shampoo)

df1.plot()

df.plot()

df2 = pd.concat([df1, df], axis=1)

df2.plot()

**d) ARIMA for least mean square error.**

import pandas as pd

import numpy as np

from statsmodels.tsa.arima.model import ARIMA

from sklearn.metrics import mean\_squared\_error

train, test = shampoo[0:25], shampoo[25:36]

for p in range(0, 3):

for d in range(0, 3):

for q in range(0, 3):

order1 = (p, d, q)

predictions = []

history = list(train['Sales'])

for t in range(len(test)):

model = ARIMA(history, order=order1)

model\_fit = model.fit()

pred\_y = model\_fit.forecast()[0]

predictions.append(pred\_y)

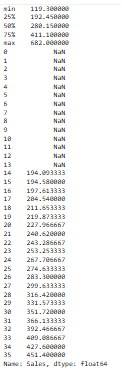
history.append(test['Sales'].iloc[t])

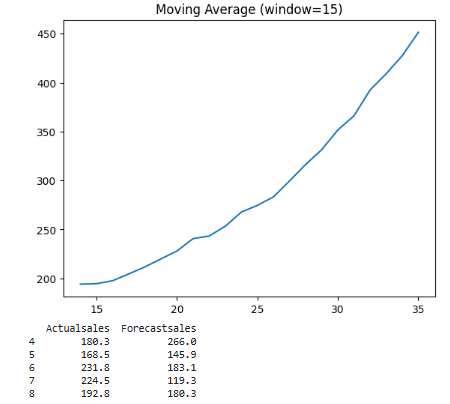
error = np.sqrt(mean\_squared\_error(test['Sales'], predictions))

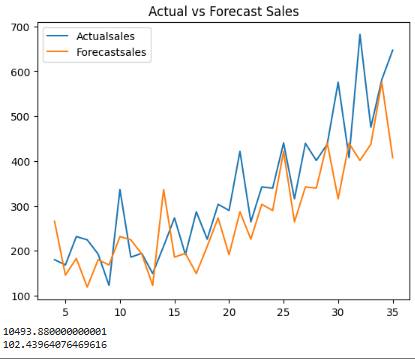
print(order1, error)

**OUTPUT:**

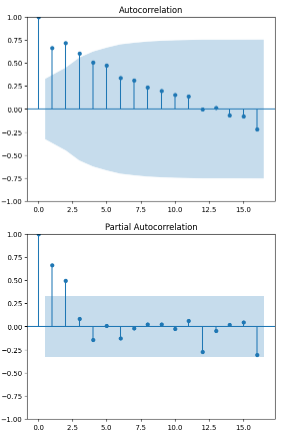
**a)Moving average**



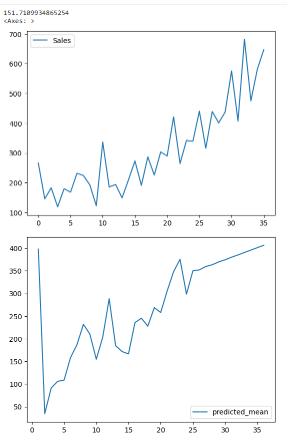


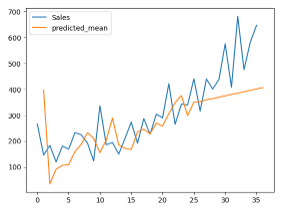


**b) Auto correlation & auto correlation.**

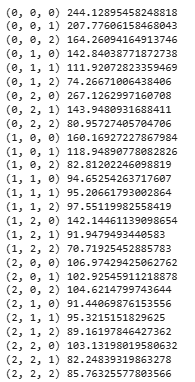


**c) ARIMA for forecast**





**d) ARIMA for least mean square error**



**Result:**

This, our program has been successfully saved and executed.

**41. SURVIVAL ANALYSIS**

**Aim:**

To write a program using R and Python to perform survival analysis using the Kaplan-Meier estimator:

a) vector Data

b) Data From CSV

**Algorithm:**

**Step 1:** Start the process

**Step 2:** Install necessary libraries and packages

**R code:** install.packages("survival")

**Python code:** pip install lifelines

**Step 3:** Load the dataset

**R code:** data(pbc)

**Python code:** df = pd.read\_csv('/path/to/your/dataset.csv')

**Step 4:** Define survival object with time and status

**R code:** fit <- survfit(Surv(pbc$time, pbc$status == 2) ~ 1)

**Python code:** T= df["tenure"] E = df["Churn"].apply(lambda x: 1 if x == "yes" else 0)

**Step 5:** Initialize Kaplan-Meier Estimator

**R code:** survfit()

**Python code:** kmf = KaplanMeierFitter()

**Step 6:** Fit the model to data  
 **R code:** fit <- survfit(Surv(pbc$time, pbc$status == 2) ~ 1)

**Python code:** kmf.fit(T, E, label='Kaplan-Meier Estimate')

**Step 7:** Plot the survival curve  
 **R code:** plot(fit, xlab = "Days", ylab = "Survival Probability", main = "Survival Curve")

**Python code:** kmf.plot(ci\_show=True)

**Step 8:** Provide summary of survival at time = 0

**R code:** summary(fit, times = 0)

**Python code:** kmf.median\_

**Step 9:** Print the result.

**Step 10:** Stop the process

**Program:**

**a) Vector Data**

install.packages("survival")

library(survival)

data(pbc)

head(pbc)

fit <- survfit(Surv(pbc$time, pbc$status == 2) ~ 1)

plot(fit, xlab = "Days", ylab = "Survival Probability", main = "Survival Curve")

summary(fit, times = 0)

summary(fit, times = 3000)

**b)** **Data From CSV**

!pip install lifelines

from lifelines import KaplanMeierFitter

durations=[5,6,6,2.5,4,4]

event\_observation=[1,0,0,1,1,1]

kmf = KaplanMeierFitter()

kmf.fit(durations,event\_observation,label='kaplan Meter Esitmate')

kmf.plot(ci\_show=True)

import pandas as pd

from lifelines import KaplanMeierFitter

df = pd.read\_csv('/content/dataset/Churn (2).csv')

T = df["tenure"]

E = df["Churn"].apply(lambda x:1 if x=="yes" else 0)

Kmf = KaplanMeierFitter()

Groups = df[ 'StreamingTV' ]

i1 = (groups == "No")

i2 = (groups == "Yes")

kmf1 = KaplanMeierFitter()

kmf2 = KaplanMeierFitter()

kmf1.fit(T[i1], E[i1], label="Not Subscribed Streaming TV")

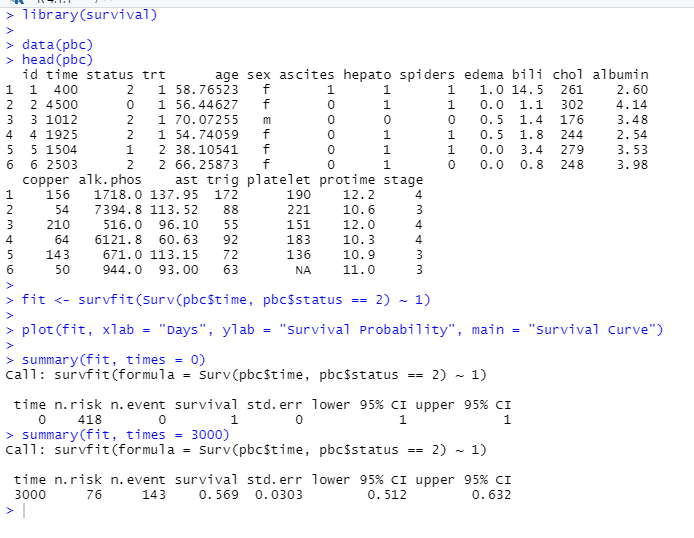
ax = kmf1.plot()

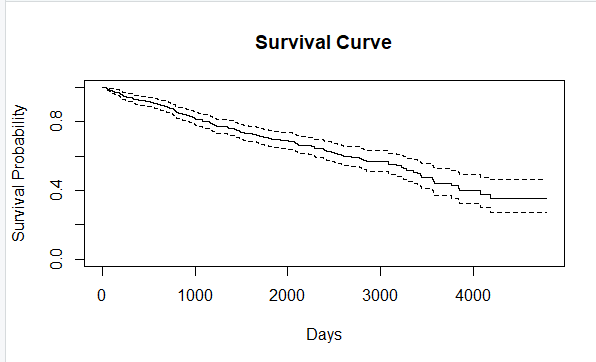
kmf2.fit(T[i2], E[i2], label="Subscribed Streaming TV")

kmf2.plot(ax=ax)

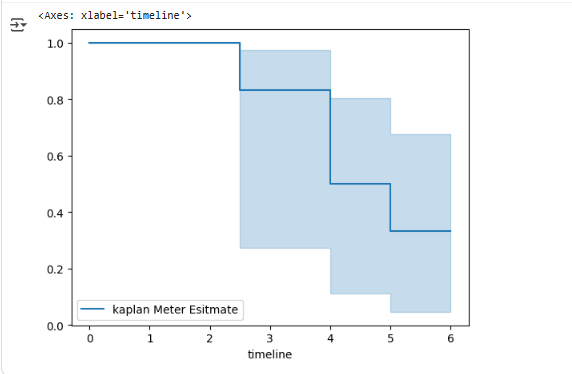
**OUTPUT:**

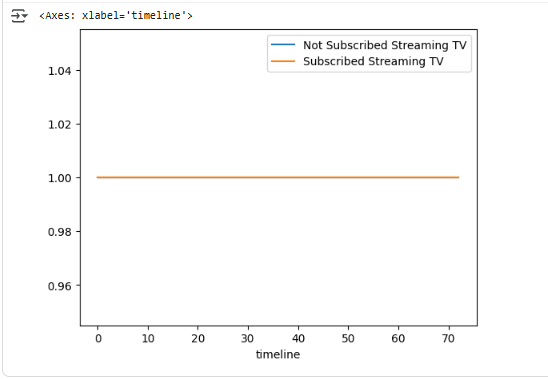
**a) Vector Data**

****

****

**b)** **Data From CSV**

****

****

**RESULT:**

This, our program has been successfully saved and executed.

**42. Random Forest using Python**

**Aim:**

To build and evaluate a Random Forest Classifier to classify iris flower species using the Iris dataset and visualize the importance of each feature.

**Algorithm:**

**Step 1:** Start the process.

**Step 2:** Import necessary libraries (numpy, pandas, matplotlib, sklearn modules).

**Step 3:** Load the Iris dataset using load\_iris() from sklearn.datasets.

**Step 4:** Extract features (x) and target labels (y) from the dataset.

**Step 5:** Split the dataset into training and testing sets using train\_test\_split (80% train, 20% test).

**Step 6:** Initialize the Random Forest Classifier with 100 trees and a fixed random state.

**Step 7:** Train the model using the training data (x\_train, y\_train).

**Step 8:** Predict the target values for the test data using the trained model.

**Step 9:** Evaluate the model's performance using accuracy\_score and classification\_report.

**Step 10:** Extract feature importances from the trained Random Forest model.

**Step 11:** Plot the feature importances as a bar chart using matplotlib.

**Step 12:** Print the result.

**Step 13:** Stop the process.

**Program:**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

iris = load\_iris()

x = iris.data

y = iris.target

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

from sklearn.ensemble import RandomForestClassifier

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(x\_train, y\_train)

from sklearn.metrics import accuracy\_score, classification\_report

y\_pred = rf\_model.predict(x\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

print(classification\_report(y\_test, y\_pred))

import matplotlib.pyplot as plt

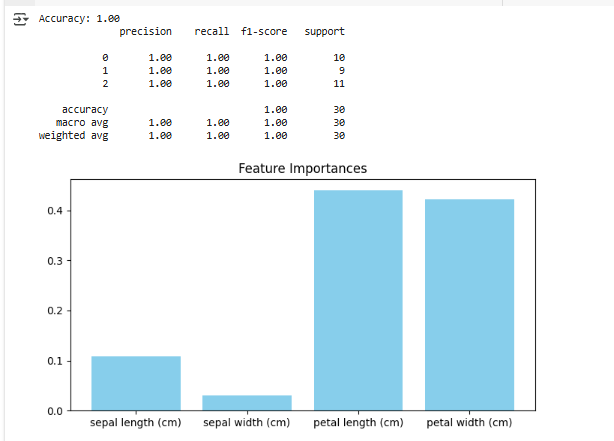
importances = rf\_model.feature\_importances\_

plt.figure(figsize=(8,4))

plt.bar(iris.feature\_names, importances, color='skyblue')

plt.title()

**Output:**

****

**Result:**

Thus, our program has been successfully saved and executed.