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:

```
Console   Terminal ×   Jobs ×
R 4.1.1 · ~/
> num1 = as.integer(readline(prompt = "Enter A value: "))
Enter A value: 10
> 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
+ }
[1] "2 is Prime.."
[1] "3 is Prime.."
[1] "5 is Prime.."
[1] "7 is Prime.."
>
```

RESULT:

Thus, our program has been successfully saved and executed.

2. 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:

```
Console   Terminal ×   Jobs ×
R 4.1.1 · ~/
> NUM1 = as.integer(readline(prompt = "Enter No: "))
Enter No: 100
> print(num)
[1] 20
>
> 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
+ }
[1] "PERFECT NUMBER IS 6"
[1] "PERFECT NUMBER IS 28"
>
```

RESULT:

Thus, our program has been successfully saved and executed.

3. 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:

Console Terminal Jobs

R 4.1.1 · ~/

```
> NUM1 = as.integer(readline(prompt = "Enter Limit: "))
Enter Limit: 1000
> 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
+ }
[1] "ARMSTRONG NUMBER IS 153"
[1] "ARMSTRONG NUMBER IS 370"
```

RESULT:

Thus, our program has been successfully saved and executed.

4. 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:

```
Console   Terminal ×   Jobs ×
R 4.1.1 · ~/
> n = as.integer(readline(prompt = "Enter number of terms in Fibonacci sequence: "))
Enter number of terms in Fibonacci sequence: 10
>  a = 0
> b = 1
> print("Fibonacci sequence:")
[1] "Fibonacci sequence:"
> for (i in 1:n) {
+   print(a)
+   temp = a + b
+   a = b
+   b = temp
+ }
[1] 0
[1] 1
[1] 1
[1] 2
[1] 3
[1] 5
[1] 8
[1] 13
[1] 21
[1] 34
> 
```

RESULT:

Thus, our program has been successfully saved and executed.

5. 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:

```
> NUM1 = as.integer(readline(prompt = "Enter Limit: "))
Enter Limit: 6
> 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
+ }
[1] "PRIME NUMBER IS 2"
[1] "PRIME NUMBER IS 3"
[1] "PRIME NUMBER IS 5"
> print(paste("PRODUCT OF PRIMES UPTO", NUM1, "IS", product))
[1] "PRODUCT OF PRIMES UPTO 6 IS 30"
> 
```

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.

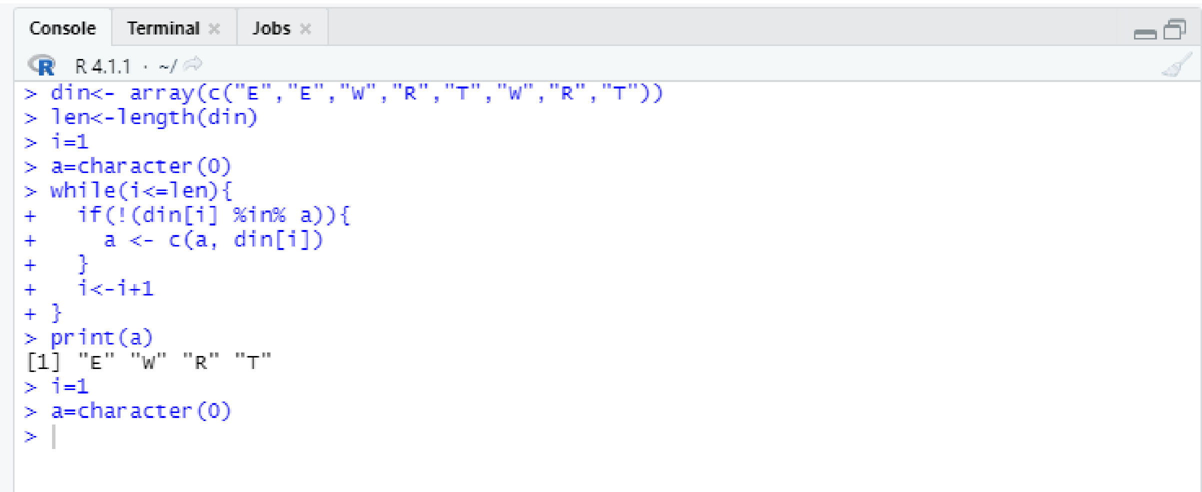
Step 10: Print the vector `a` which now contains only the unique elements.

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:

Console Terminal Jobs

R 4.1.1 · ~/

```
> 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")
Matrix 1:
> print(matrix1)
     [,1] [,2] [,3] [,4]
[1,]    8   10    8    3
[2,]    7    9    6    5
[3,]    7    8    1    7
[4,]    1    3    4    2
>
> cat("\nMatrix 2:\n")

Matrix 2:
> print(matrix2)
     [,1] [,2] [,3] [,4]
[1,]    8   10    9    9
[2,]    8    6    3    4
[3,]    4    2    5   10
[4,]    8    1    5    3
>
> cat("\nAddition:\n")

Addition:
> print(matrix_addition(matrix1, matrix2))
     [,1] [,2] [,3] [,4]
[1,]   16   20   17   12
[2,]   15   15    9    9
[3,]   11   10    6   17
[4,]    9    4    9    5
>
> cat("\nSubtraction:\n")

Subtraction:
> print(matrix_subtraction(matrix1, matrix2))
     [,1] [,2] [,3] [,4]
[1,]    0    0   -1   -6
[2,]   -1    3    3    1
[3,]    3    6   -4   -3
[4,]   -7    2   -1   -1
>
> cat("\nMultiplication:\n")
```

```
Multiplication:
> print(matrix_multiplication(matrix1, matrix2))
     [,1] [,2] [,3] [,4]
[1,]  200  159  157  201
[2,]  192  141  145  174
[3,]  180  127  127  126
[4,]   64   38   48   67
>
> cat("\nDivision:\n")

Division:
> print(matrix_division(matrix1, matrix2))
      [,1] [,2]      [,3]      [,4]
[1,] 1.000  1.0 0.8888889 0.3333333
[2,] 0.875  1.5 2.0000000 1.2500000
[3,] 1.750  4.0 0.2000000 0.7000000
[4,] 0.125  3.0 0.8000000 0.6666667
>
```

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:

```
> setwd("D:/R lab")
> getwd()
[1] "D:/R lab"
> data=read.csv("Details.csv")
> print(data)
   ID     NAME GENDER AGE
1   1     Hari   male  21
2   2 punitha   girl  20
3   3   dhamo   male  20
4   4 maalini   girl  28
5   5    diva   male  30
6   6     abi   male  19
7   7 aravind   male  29
8   8 akshaya   girl  20
9   9   raani   girl  30
10 10    mani   male  45
> res_age25=subset(data,AGE>25)
> print(res_age25)
   ID     NAME GENDER AGE
4   4 maalini   girl  28
5   5    diva   male  30
7   7 aravind   male  29
9   9   raani   girl  30
10 10    mani   male  45
> res_agebetween=subset(data,AGE>10 & AGE<40)
> print(res_agebetween)
  ID     NAME GENDER AGE
1  1     Hari   male  21
2  2 punitha   girl  20
3  3   dhamo   male  20
4  4 maalini   girl  28
5  5    diva   male  30
6  6     abi   male  19
7  7 aravind   male  29
8  8 akshaya   girl  20
9  9   raani   girl  30
>
> res_agegender=subset(data,AGE>25 & GENDER == "girl")
> print(res_agegender)
  ID     NAME GENDER AGE
4  4 maalini   girl  28
9  9   raani   girl  30
>
> res_agegender=subset(data,AGE>20 & AGE<40 & GENDER == "male")
> print(res_agegender)
  ID     NAME GENDER AGE
1  1     Hari   male  21
5  5    diva   male  30
7  7 aravind   male  29
>
> res_a=subset(data,grepl( "^a",NAME,ignore.case=TRUE))
> print(res_a)
  ID     NAME GENDER AGE
6  6     abi   male  19
7  7 aravind   male  29
8  8 akshaya   girl  20
```

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:

```
> getwd()
[1] "D:/R lab"
> setwd("D:/R lab")
> getwd()
[1] "D:/R lab"
> data <- read.csv("gender.csv")
> print(data)
   Emp_id       Name Age Gender Basic Salary  DA
1       1 ThamilMani  44   Male   200  20000 220
2       2       Hari  22   Male   300  22000 346
3       3     Mukesh  80   Male   234  18000 124
4       4      Dhamo  21   Male   523  30000 872
5       5  Divakaran  30   Male   721  35000 313
6       6      Kavin  21   Male   332  28000 301
7       7       Suji  42 Female   342  18000 349
8       8       Nivi  19 Female   445  22000 621
9       9     Kaviya  25 Female   134  23000 917
10     10   Harshini  46 Female   343  19000 719
> retval <- subset(data,data$Gender=="Female")
> print(retval)
   Emp_id     Name Age Gender Basic Salary  DA
7       7     Suji  42 Female   342  18000 349
8       8     Nivi  19 Female   445  22000 621
9       9   Kaviya  25 Female   134  23000 917
10     10 Harshini  46 Female   343  19000 719
>
> retval <- subset(data,data$Gender=="Male")
> print(retval)
  Emp_id       Name Age Gender Basic Salary  DA
1      1 ThamilMani  44   Male   200  20000 220
2      2       Hari  22   Male   300  22000 346
3      3     Mukesh  80   Male   234  18000 124
4      4      Dhamo  21   Male   523  30000 872
5      5  Divakaran  30   Male   721  35000 313
6      6      Kavin  21   Male   332  28000 301

> retval <- subset(data,data$Age>40 & data$Gender=="Male")
> print(retval)
  Emp_id       Name Age Gender Basic Salary  DA
1      1 ThamilMani  44   Male   200  20000 220
3      3     Mukesh  80   Male   234  18000 124
>
> retval <- subset(data,data$Salary>600)
> print(retval)
   Emp_id       Name Age Gender Basic Salary  DA
1       1 ThamilMani  44   Male   200  20000 220
2       2       Hari  22   Male   300  22000 346
3       3     Mukesh  80   Male   234  18000 124
4       4      Dhamo  21   Male   523  30000 872
5       5  Divakaran  30   Male   721  35000 313
6       6      Kavin  21   Male   332  28000 301
7       7       Suji  42 Female   342  18000 349
8       8       Nivi  19 Female   445  22000 621
9       9     Kaviya  25 Female   134  23000 917
10     10   Harshini  46 Female   343  19000 719
>
> retval <- subset(data,data$Salary>600 & data$Basic>300 & data$DA<200)
> print(retval)
[1] Emp_id Name   Age    Gender Basic  Salary DA
<0 rows> (or 0-length row.names)
>
> retval <- data$Salary - (data$Basic + data$DA)
> print(retval)
 [1] 19580 21354 17642 28605 33966 27367 17309 20934 21949 17938
>
```

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:

```
> data <- fromJSON("gender.json")
> print(data)
   Emp_id      Name Age Gender Basic Salary  DA
1       1 ThamilMani  44   Male   200  20000 220
2       2       Hari  22   Male   300  22000 346
3       3     Mukesh  80   Male   234  18000 124
4       4      Dhamo  21   Male   523  30000 872
5       5  Divakaran  30   Male   721  35000 313
6       6      Kavin  21   Male   332  28000 301
7       7       Suji  42 Female   342  18000 349
8       8       Nivi  19 Female   445  22000 621
9       9     Kaviya  25 Female   134  23000 917
10     10   Harshini  46 Female   343  19000 719
> retval <- subset(data,data$Gender=="Female")
> print(retval)
   Emp_id     Name Age Gender Basic Salary  DA
7       7     Suji  42 Female   342  18000 349
8       8     Nivi  19 Female   445  22000 621
9       9   Kaviya  25 Female   134  23000 917
10     10 Harshini  46 Female   343  19000 719
>
> retval <- subset(data,data$Gender=="Male")
> print(retval)
  Emp_id       Name Age Gender Basic Salary  DA
1      1 ThamilMani  44   Male   200  20000 220
2      2       Hari  22   Male   300  22000 346
3      3     Mukesh  80   Male   234  18000 124
4      4      Dhamo  21   Male   523  30000 872
5      5  Divakaran  30   Male   721  35000 313
6      6      Kavin  21   Male   332  28000 301
>
> retval <- subset(data,data$Age>40 & data$Gender=="Male")
> print(retval)
  Emp_id       Name Age Gender Basic Salary  DA
1      1 ThamilMani  44   Male   200  20000 220
3      3     Mukesh  80   Male   234  18000 124
>
> retval <- subset(data,data$Salary>600)
> print(retval)
   Emp_id       Name Age Gender Basic Salary  DA
1       1 ThamilMani  44   Male   200  20000 220
2       2       Hari  22   Male   300  22000 346
3       3     Mukesh  80   Male   234  18000 124
4       4      Dhamo  21   Male   523  30000 872
5       5  Divakaran  30   Male   721  35000 313
6       6      Kavin  21   Male   332  28000 301
7       7       Suji  42 Female   342  18000 349
8       8       Nivi  19 Female   445  22000 621
9       9     Kaviya  25 Female   134  23000 917
10     10   Harshini  46 Female   343  19000 719
>
> retval <- subset(data,data$Salary>600 & data$Basic>300 & data$DA<200)
> print(retval)
[1] Emp_id Name   Age    Gender Basic  Salary DA
<0 rows> (or 0-length row.names)
>
> retval <- data$Salary - (data$Basic + data$DA)
> print(retval)
 [1] 19580 21354 17642 28605 33966 27367 17309 20934 21949 17938
>
```

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:

```
Console   Terminal ×   Jobs ×
R 4.1.1 · C:/Users/MCA-008/Desktop/PRACTIAL/
> print(data)
   Emp_id       Name Age Gender Basic Salary  DA
1       1 ThamilMani  44   Male   200  20000 220
2       2       Hari  22   Male   300  22000 346
3       3     Mukesh  80   Male   234  18000 124
4       4      Dhamo  21   Male   523  30000 872
5       5  Divakaran  30   Male   721  35000 313
6       6      Kavin  21   Male   332  28000 301
7       7       Suji  42 Female   342  18000 349
8       8       Nivi  19 Female   445  22000 621
9       9     Kaviya  25 Female   134  23000 917
10     10   Harshini  46 Female   343  19000 719
>
> # Step 4b: Convert relevant columns to numeric (they might be factors or characters)
> data$Salary <- as.numeric(as.character(data$Salary))
> data$Basic  <- as.numeric(as.character(data$Basic))
> data$DA     <- as.numeric(as.character(data$DA))
> data$Age    <- as.numeric(as.character(data$Age)) # Also convert Age if needed
>
> # Step 5: Filter by Gender = Female
> female_data <- subset(data, Gender == "Female")
> print(female_data)
   Emp_id     Name Age Gender Basic Salary  DA
7       7     Suji  42 Female   342  18000 349
8       8     Nivi  19 Female   445  22000 621
9       9   Kaviya  25 Female   134  23000 917
10     10 Harshini  46 Female   343  19000 719
>
> # Step 6: Filter by Gender = Male
> male_data <- subset(data, Gender == "Male")
> print(male_data)
  Emp_id       Name Age Gender Basic Salary  DA
1      1 ThamilMani  44   Male   200  20000 220
2      2       Hari  22   Male   300  22000 346
3      3     Mukesh  80   Male   234  18000 124
4      4      Dhamo  21   Male   523  30000 872
5      5  Divakaran  30   Male   721  35000 313
6      6      Kavin  21   Male   332  28000 301
```

```
>
> # Step 7: Filter Age > 40 and Gender = Male
> age_gender_filter <- subset(data, Age > 40 & Gender == "Male")
> print(age_gender_filter)
  Emp_id       Name Age Gender Basic Salary  DA
1      1 ThamilMani  44   Male   200  20000 220
3      3     Mukesh  80   Male   234  18000 124
>
> # Step 8: Filter Salary > 600
> salary_filter <- subset(data, Salary > 600)
> print(salary_filter)
   Emp_id       Name Age Gender Basic Salary  DA
1       1 ThamilMani  44   Male   200  20000 220
2       2       Hari  22   Male   300  22000 346
3       3     Mukesh  80   Male   234  18000 124
4       4      Dhamo  21   Male   523  30000 872
5       5  Divakaran  30   Male   721  35000 313
6       6      Kavin  21   Male   332  28000 301
7       7       Suji  42 Female   342  18000 349
8       8       Nivi  19 Female   445  22000 621
9       9     Kaviya  25 Female   134  23000 917
10     10   Harshini  46 Female   343  19000 719
>
> # Step 8b: Filter Salary > 600 & Basic > 300 & DA < 200
> complex_filter <- subset(data, Salary > 600 & Basic > 300 & DA < 200)
> print(complex_filter)
[1] Emp_id Name    Age     Gender Basic   Salary DA
<0 rows> (or 0-length row.names)
>
> # Step 9: Calculate difference between Salary and (Basic + DA)
> data$Difference <- data$Salary - (data$Basic + data$DA)
> print(data$Difference)
 [1] 19580 21354 17642 28605 33966 27367 17309 20934 21949 17938
>
```

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:

```
> # Read the Excel file and assign to a proper variable name
> gender_data <- read_excel("gender.xlsx")
> print(gender_data)
# A tibble: 10 x 7
   Emp_id Name         Age Gender Basic Salary    DA
    <dbl> <chr>      <dbl> <chr>  <dbl>  <dbl> <dbl>
 1      1 ThamilMani    44 Male     200  20000   220
 2      2 Hari          22 Male     300  22000   346
 3      3 Mukesh        80 Male     234  18000   124
 4      4 Dhamo         21 Male     523  30000   872
 5      5 Divakaran     30 Male     721  35000   313
 6      6 Kavin         21 Male     332  28000   301
 7      7 Suji          42 Female   342  18000   349
 8      8 Nivi          19 Female   445  22000   621
 9      9 Kaviya        25 Female   134  23000   917
10     10 Harshini      46 Female   343  19000   719
```

```
> # 1. All Females
> retval <- subset(gender_data, Gender == "Female")
> print(retval)
# A tibble: 4 x 7
  Emp_id Name       Age Gender Basic Salary    DA
   <dbl> <chr>    <dbl> <chr>  <dbl>  <dbl> <dbl>
1      7 Suji        42 Female   342  18000   349
2      8 Nivi        19 Female   445  22000   621
3      9 Kaviya      25 Female   134  23000   917
4     10 Harshini    46 Female   343  19000   719
```

```
# 3. Males older than 40
retval <- subset(gender_data, Age > 40 & Gender == "Male")
print(retval)
A tibble: 2 x 7
Emp_id Name         Age Gender Basic Salary    DA
 <dbl> <chr>      <dbl> <chr>  <dbl>  <dbl> <dbl>
     1 ThamilMani    44 Male     200  20000   220
     3 Mukesh        80 Male     234  18000   124
```

```
 4. Salary > 600
etval <- subset(gender_data, Salary > 600)
rint(retval)
 tibble: 10 x 7
Emp_id Name          Age Gender Basic Salary    DA
 <dbl> <chr>       <dbl> <chr>  <dbl>  <dbl> <dbl>
     1 ThamilMani     44 Male     200  20000   220
     2 Hari           22 Male     300  22000   346
     3 Mukesh         80 Male     234  18000   124
     4 Dhamo          21 Male     523  30000   872
     5 Divakaran      30 Male     721  35000   313
     6 Kavin          21 Male     332  28000   301
     7 Suji           42 Female   342  18000   349
     8 Nivi           19 Female   445  22000   621
     9 Kaviya         25 Female   134  23000   917
    10 Harshini       46 Female   343  19000   719
```

```
> # 5. Salary > 600 & Basic > 300 & DA < 200
> retval <- subset(gender_data, Salary > 600 & Basic > 300 & DA < 200)
> print(retval)
# A tibble: 0 x 7
# i 7 variables: Emp_id <dbl>, Name <chr>, Age <dbl>, Gender <chr>, Basic <dbl>,
#   Salary <dbl>, DA <dbl>
> # 6. Salary - (Basic + DA)
> retval <- gender_data$Salary - (gender_data$Basic + gender_data$DA)
> print(retval)
 [1] 19580 21354 17642 28605 33966 27367 17309 20934 21949 17938
>
```

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:

```
> json_text <- toJSON(data, pretty = TRUE, auto_unbox = TRUE)
> write(json_text, file = "gender.json")
> data <- fromJSON("gender.json")
> print(data)
  Emp_id      Name Age Gender Basic Salary  DA
1      1 ThamilMani  44   Male   200  20000 220
2      2       Hari  22   Male   300  22000 346
3      3     Mukesh  80   Male   234  18000 124
4      4      Dhamo  21   Male   523  30000 872
5      5  Divakaran  30   Male   721  35000 313
6      6      Kavin  21   Male   332  28000 301
7      7       Suji  42 Female   342  18000 349
8      8       Nivi  19 Female   445  22000 621
9      9     Kaviya  25 Female   134  23000 917
```

```
> retval <- subset(data, Gender == "Female")
> print(retval)
   Emp_id     Name Age Gender Basic Salary  DA
7       7     Suji  42 Female   342  18000 349
8       8     Nivi  19 Female   445  22000 621
9       9   Kaviya  25 Female   134  23000 917
10     10 Harshini  46 Female   343  19000 719
> retval <- subset(data, Gender == "Male")
> print(retval)
  Emp_id       Name Age Gender Basic Salary  DA
1      1 ThamilMani  44   Male   200  20000 220
2      2       Hari  22   Male   300  22000 346
3      3     Mukesh  80   Male   234  18000 124
4      4      Dhamo  21   Male   523  30000 872
5      5  Divakaran  30   Male   721  35000 313
6      6      Kavin  21   Male   332  28000 301
```

```
> retval <- subset(data, Age > 40 & Gender == "Male")
> print(retval)
  Emp_id       Name Age Gender Basic Salary  DA
1      1 ThamilMani  44   Male   200  20000 220
3      3     Mukesh  80   Male   234  18000 124
```

```
> retval <- subset(data, Salary > 600)
> print(retval)
   Emp_id       Name Age Gender Basic Salary  DA
1       1 ThamilMani  44   Male   200  20000 220
2       2       Hari  22   Male   300  22000 346
3       3     Mukesh  80   Male   234  18000 124
4       4      Dhamo  21   Male   523  30000 872
5       5  Divakaran  30   Male   721  35000 313
6       6      Kavin  21   Male   332  28000 301
7       7       Suji  42 Female   342  18000 349
8       8       Nivi  19 Female   445  22000 621
9       9     Kaviya  25 Female   134  23000 917
10     10   Harshini  46 Female   343  19000 719
```

```
retval <- subset(data, Salary > 600 & Basic > 300 & DA < 200)
print(retval)
1] Emp_id Name   Age    Gender Basic  Salary DA
0 rows> (or 0-length row.names)
retval <- data$Salary - (data$Basic + data$DA)
print(retval)
[1] 19580 21354 17642 28605 33966 27367 17309 20934 21949 17938
```

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.

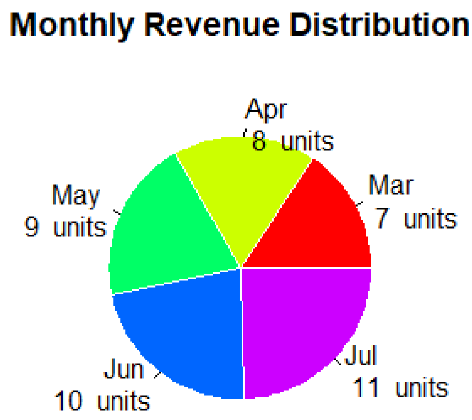
Step 10: End the process.

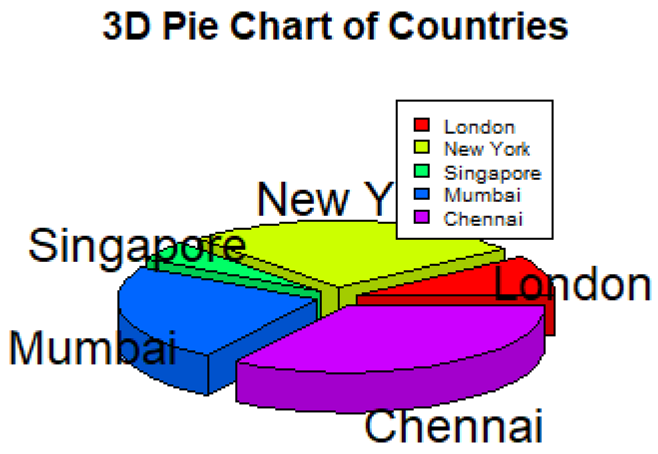
Program:

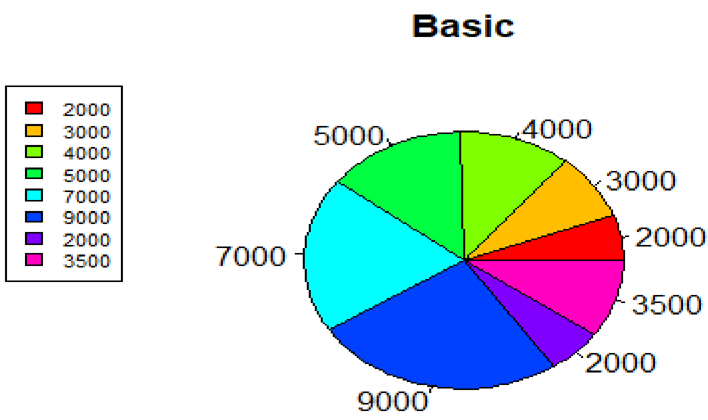
```
months <- c("Mar", "Apr", "May", "Jun", "Jul")
revenue <- c(7, 8, 9, 10, 11)
pie(revenue,
    labels = paste(months, "\n", revenue, " units"),
    main = "Monthly Revenue Distribution",
    col = rainbow(length(months)),
    border = "white")
 x <- c(21, 62, 10, 53, 76)
 labels <- c("London", "New 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)))
```

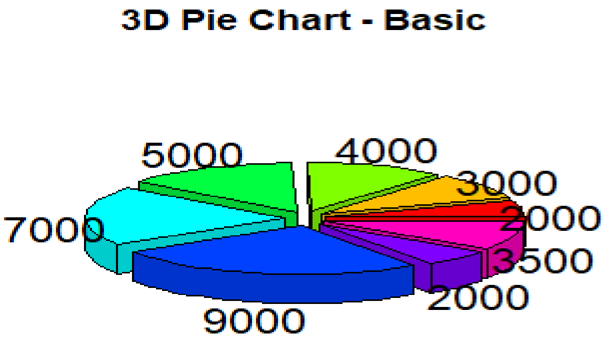
```
legend("topleft",
       legend = v,
       cex = 0.7,
       fill = rainbow(length(v)))
library(plotrix)
pie3D(v,
      labels = lbl,
      explode = 0.1,
      main = "3D Pie Chart - Basic",
      col = rainbow(length(v)))

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)))
```

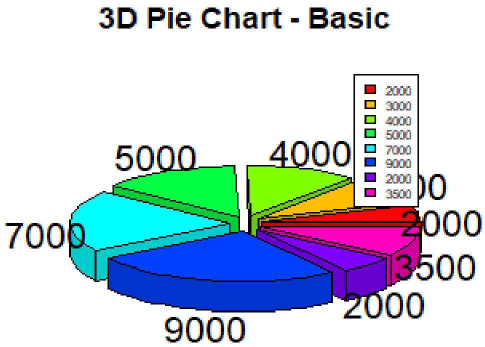
OUTPUT:

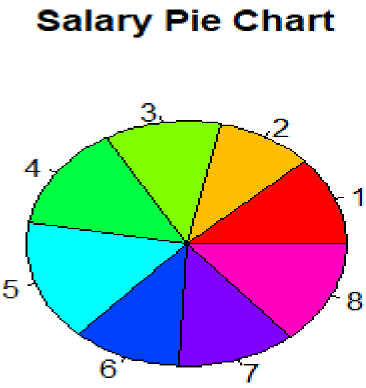

Monthly Revenue Distribution
Apr
8 units
Mar
7 units
May
9 units
Jun
10 units
Jul
11 units

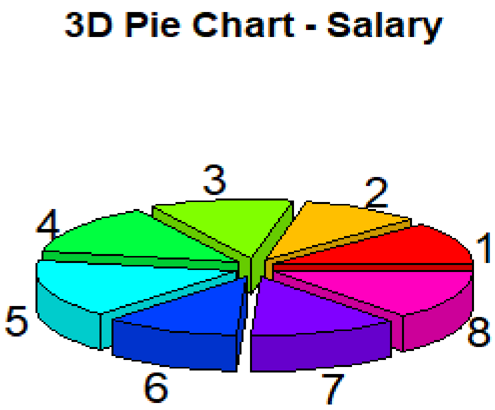

3D Pie Chart of Countries
London
New York
Singapore
Mumbai
Chennai
New Y
Singapore
London
Mumbai
Chennai


Basic
2000
3000
4000
5000
7000
9000
2000
3500
5000
4000
3000
2000
7000
3500
2000
9000









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

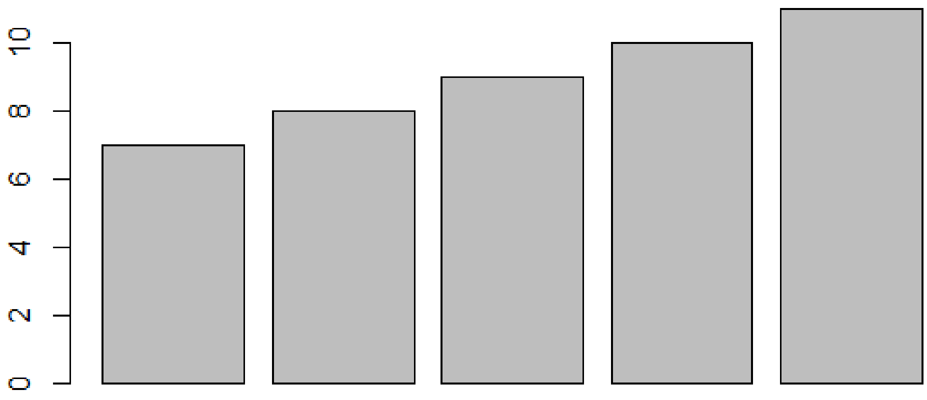
Step 6: Execute the program to display the bar chart.

Step 7: End the program.

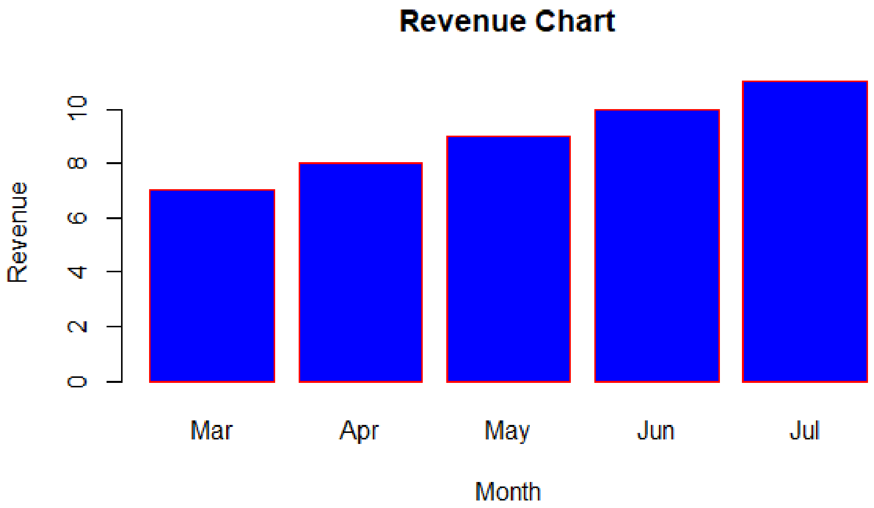
Program :

```
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)
```

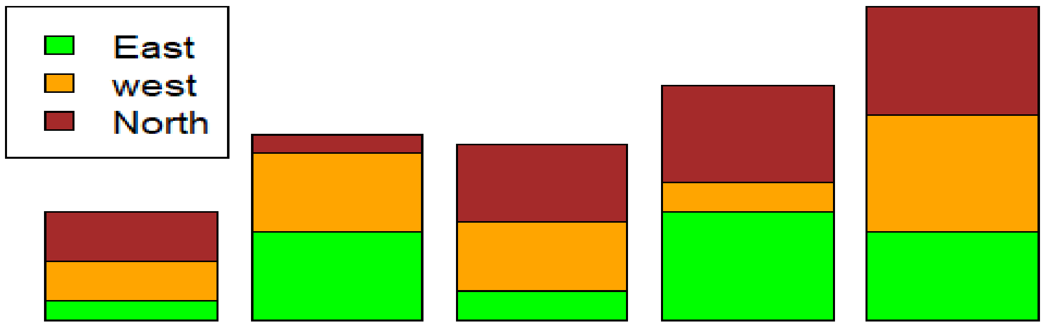
OUTPUT:

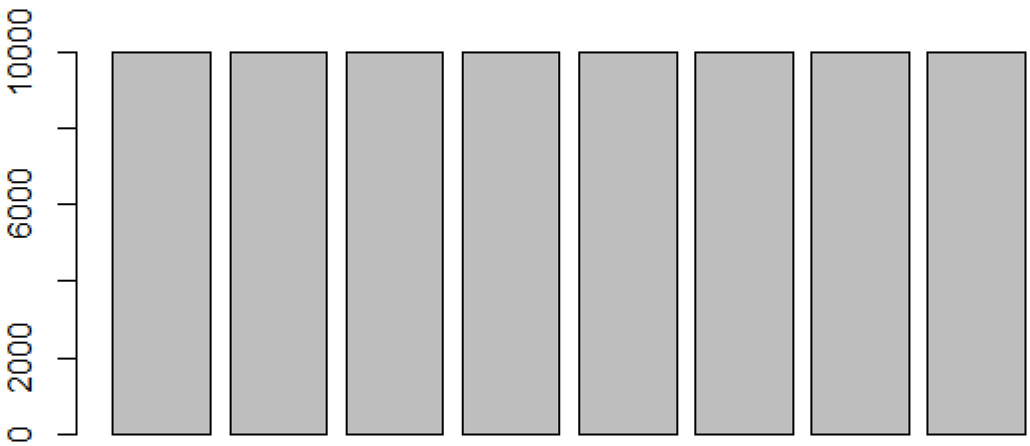


0
2
4
6
8
10



Revenue Chart
Revenue
0
2
4
6
8
10
Mar
Apr
May
Jun
Jul
Month





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.

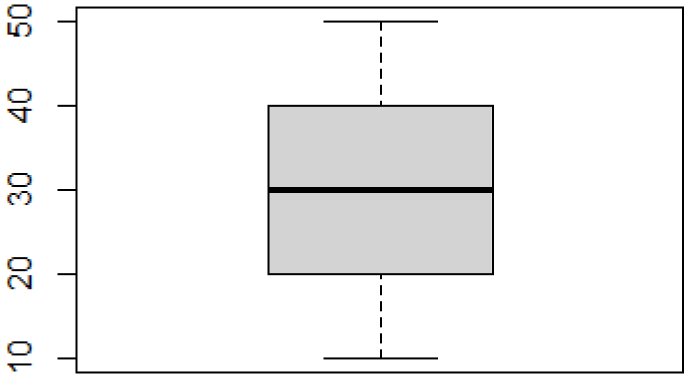
Step 8: End the process.

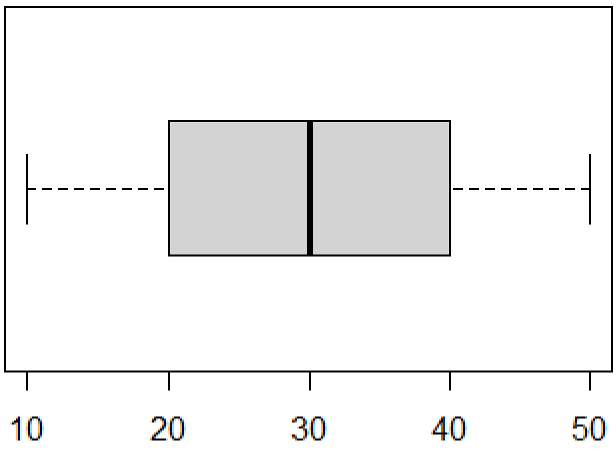
Program:

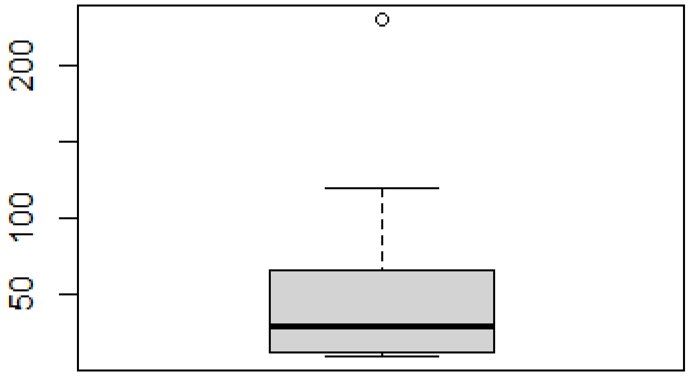
```
x <- c(10,20,30,40,50)
summary(x)
boxplot(x,horizontal = FALSE)
boxplot(x,horizontal = TRUE)
x<- c(10,11,14,15,120,12,34,54,65,24,67,230)
boxplot(x,horizontal = FALSE)
print(mean(x))
mtcars
input <- mtcars[, c("mpg", "cyl")]
print(input)
boxplot(mpg~cyl,data=mtcars,xlab = "Number of Cylinders ",ylab = "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:

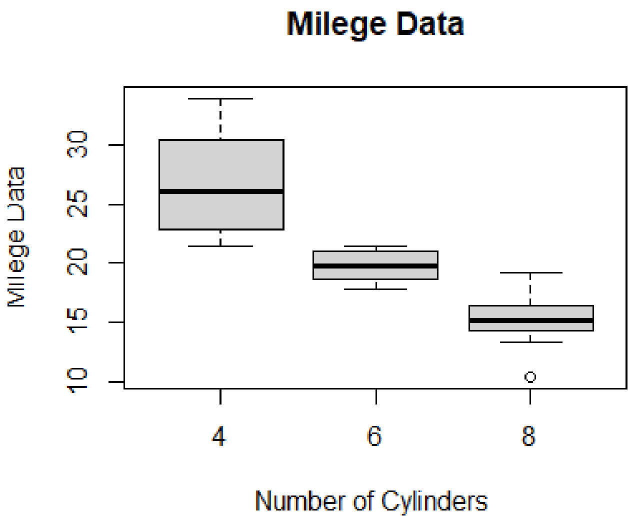


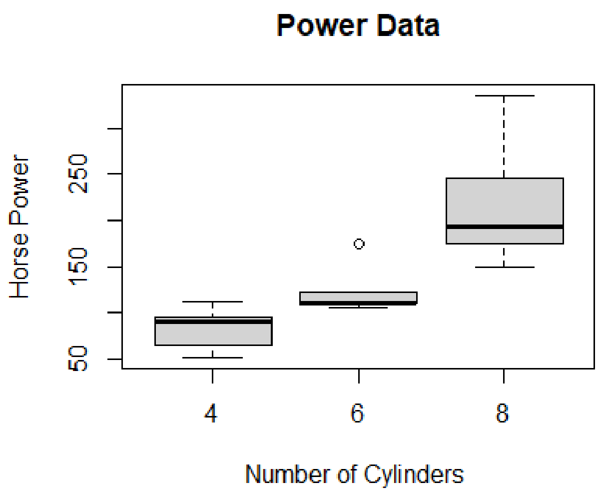


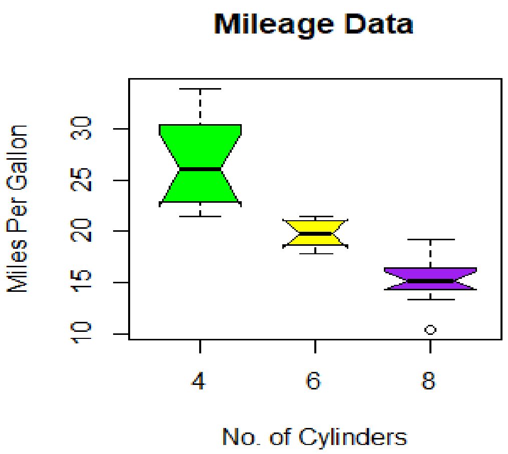



```
> x <- c(10,20,30,40,50)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
     10      20      30      30      40      50 
> boxplot(x,horizontal = FALSE)
> boxplot(x,horizontal = FALSE)
> boxplot(x,horizontal = TRUE)
> x<- c(10,11,14,15,120,12,34,54,65,24,67,230)
> boxplot(x,horizontal = FALSE)
> print(mean(x))
[1] 54.66667
> 
```

```
> input <- mtcars[, c("mpg", "cyl")]
> print(input)
                     mpg cyl
Mazda RX4           21.0   6
Mazda RX4 Wag       21.0   6
Datsun 710          22.8   4
Hornet 4 Drive      21.4   6
Hornet Sportabout   18.7   8
Valiant             18.1   6
Duster 360          14.3   8
Merc 240D           24.4   4
Merc 230            22.8   4
Merc 280            19.2   6
Merc 280C           17.8   6
Merc 450SE          16.4   8
Merc 450SL          17.3   8
Merc 450SLC         15.2   8
Cadillac Fleetwood  10.4   8
Lincoln Continental 10.4   8
Chrysler Imperial   14.7   8
Fiat 128            32.4   4
Honda Civic         30.4   4
Toyota Corolla      33.9   4
Toyota Corona       21.5   4
Dodge Challenger    15.5   8
AMC Javelin         15.2   8
Camaro Z28          13.3   8
Pontiac Firebird    19.2   8
```

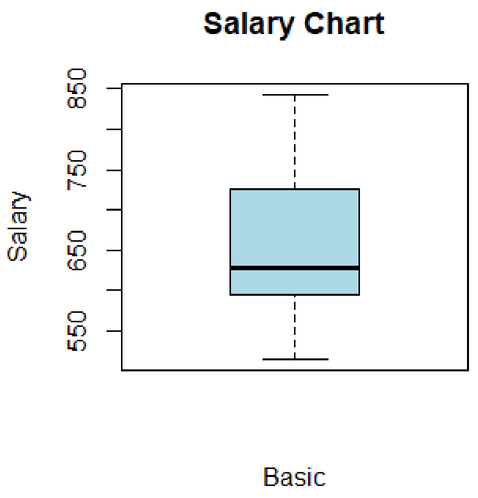

Milege Data
Milege Data
10
15
20
25
30
4
6
8
Number of Cylinders


Power Data
Horse Power
50
150
250
4
6
8
Number of Cylinders


Mileage Data
Miles Per Gallon
10
15
20
25
30
4
6
8
No. of Cylinders

```
> setwd("D:/24PCA014/Practical/Box Plot")
> # Read CSV
> df <- read.csv("combined.csv")
> print(df)
  Unnamed..0 ID     Name ID.1 Salary  STARTDATE       DEPT Basic
1          1  1     Rick    1 623.30   1/1/2012         IT 10000
2          2  2      Dan    2 515.20  9/23/2013 Operations 10000
3          3  3 Michelle    3 611.00 11/15/2014         IT 10000
4          4  4     Ryan    4 729.00  5/11/2014         HR 10000
5          5  5     Gary    5 843.25  3/27/2015    Finance 10000
6          6  6     Nina    6 578.00  5/21/2013         IT 10000
7          7  7    Simon    7 632.80  7/30/2013 Operations 10000
8          8  8     Guru    8 722.50  6/17/2014    Finance 10000
> 
```

```
> # Subset only Salary and Basic
> v <- df[, c("Salary", "Basic")]
> print(v)
  Salary Basic
1 623.30 10000
2 515.20 10000
3 611.00 10000
4 729.00 10000
5 843.25 10000
6 578.00 10000
7 632.80 10000
8 722.50 10000
```



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.

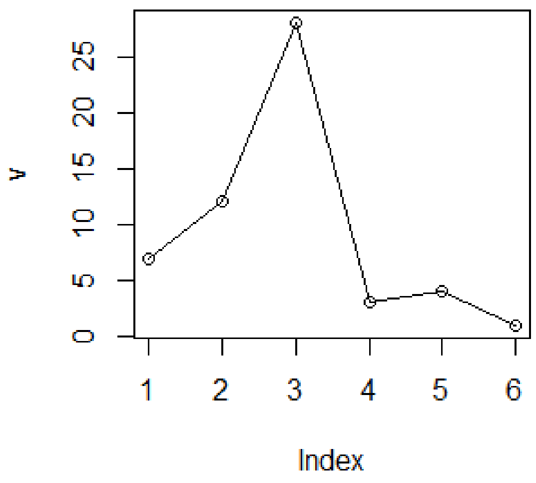
Step 8: End the program.

Program:

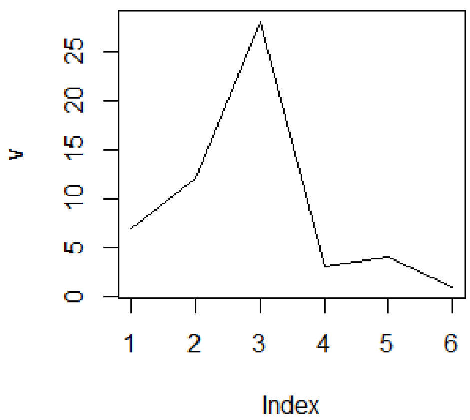
```
v<-c(7,12,28,3,4,1)
print(v)
plot(v,type="o")
plot(v,tyle="l")
plot(c)
plot(v,type="o",col="red",xlab="Month",ylab="Rainfall",main="Rainfall Chart")

#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)
```

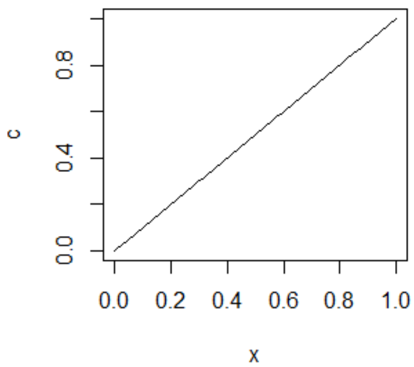
OUTPUT:



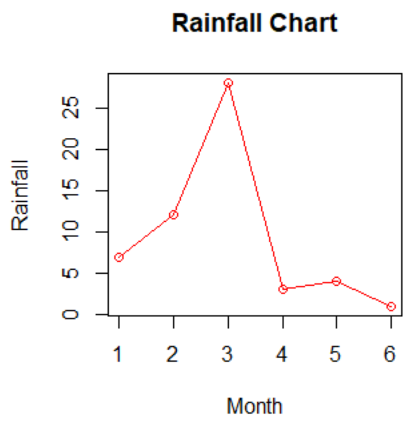
v
0
5
10
15
20
25
1
2
3
4
5
6
Index



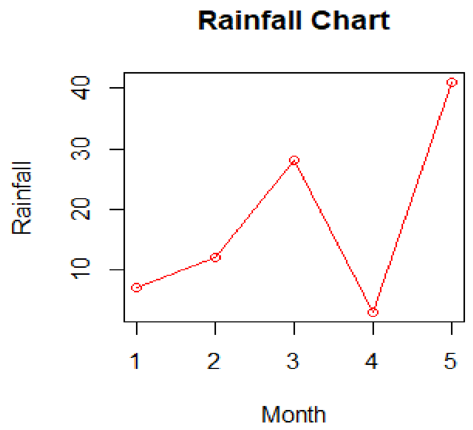
v
0
5
10
15
20
25
1
2
3
4
5
6
Index



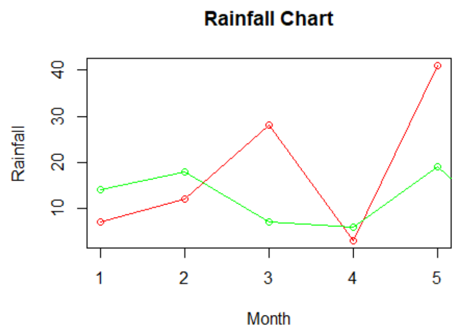
c
0.0
0.4
0.8
0.0
0.2
0.4
0.6
0.8
1.0
x



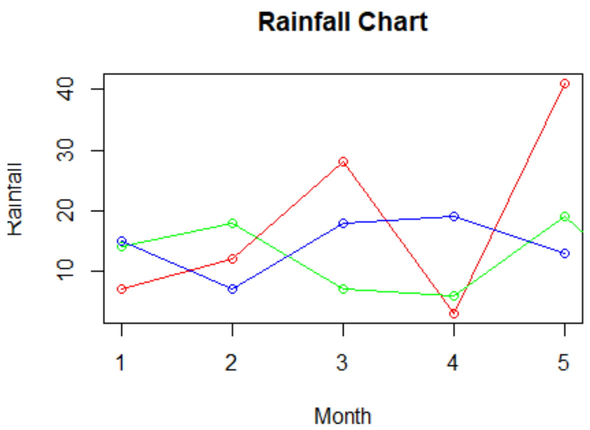
Rainfall Chart
Rainfall
0
5
10
15
20
25
1
2
3
4
5
6
Month



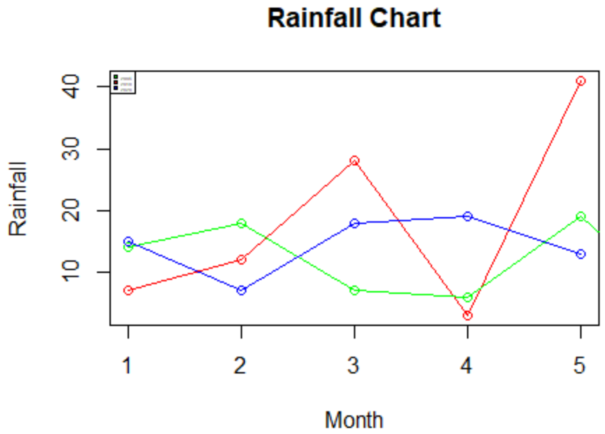
Rainfall Chart
Rainfall
10
20
30
40
1
2
3
4
5
Month



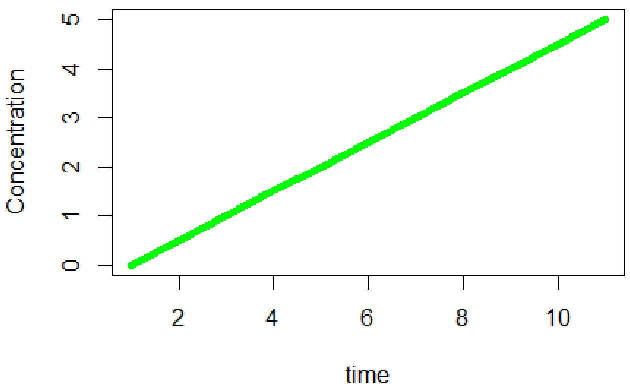
Rainfall Chart
Rainfall
10
20
30
40
1
2
3
4
5
Month

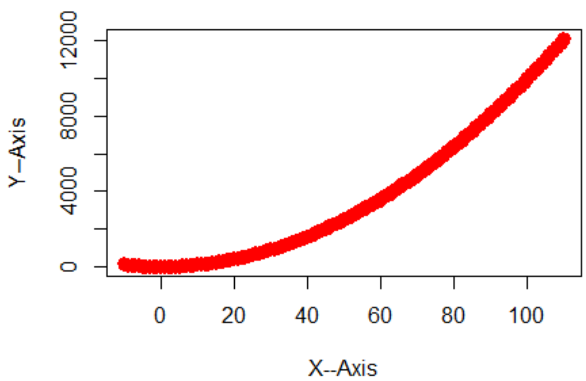


Rainfall Chart
Rainfall
10
20
30
40
1
2
3
4
5
Month

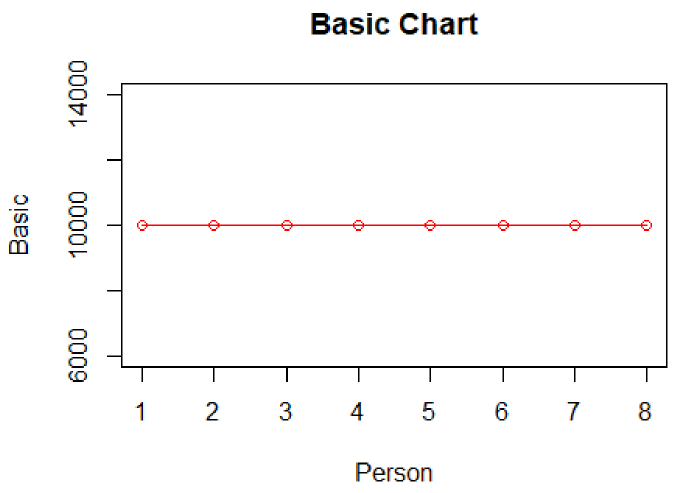


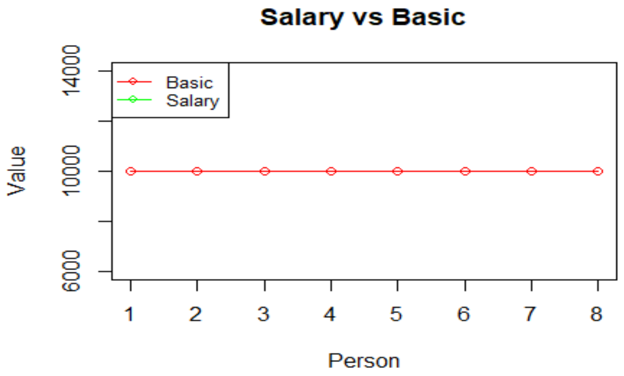

```
> t<-0:10
> z=exp(t/2)
> print(t)
 [1]  0  1  2  3  4  5  6  7  8  9 10
> print(z)
 [1]   1.000000   1.648721   2.718282   4.481689   7.389056  12.182494  20.085537  33.115452
 [9]  54.598150  90.017131 148.413159
> plot(t/2,type="l",col="green",lwd=5,xlab="time",ylab="Concentration")
> 
```





```
> df <- read.csv("combined.csv")
> print(df)
  Unnamed..0 ID     Name ID.1 Salary  STARTDATE       DEPT Basic
1          1  1     Rick    1 623.30   1/1/2012         IT 10000
2          2  2      Dan    2 515.20  9/23/2013 Operations 10000
3          3  3 Michelle    3 611.00 11/15/2014         IT 10000
4          4  4     Ryan    4 729.00  5/11/2014         HR 10000
5          5  5     Gary    5 843.25  3/27/2015    Finance 10000
6          6  6     Nina    6 578.00  5/21/2013         IT 10000
7          7  7    Simon    7 632.80  7/30/2013 Operations 10000
8          8  8     Guru    8 722.50  6/17/2014    Finance 10000
> 
```





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.

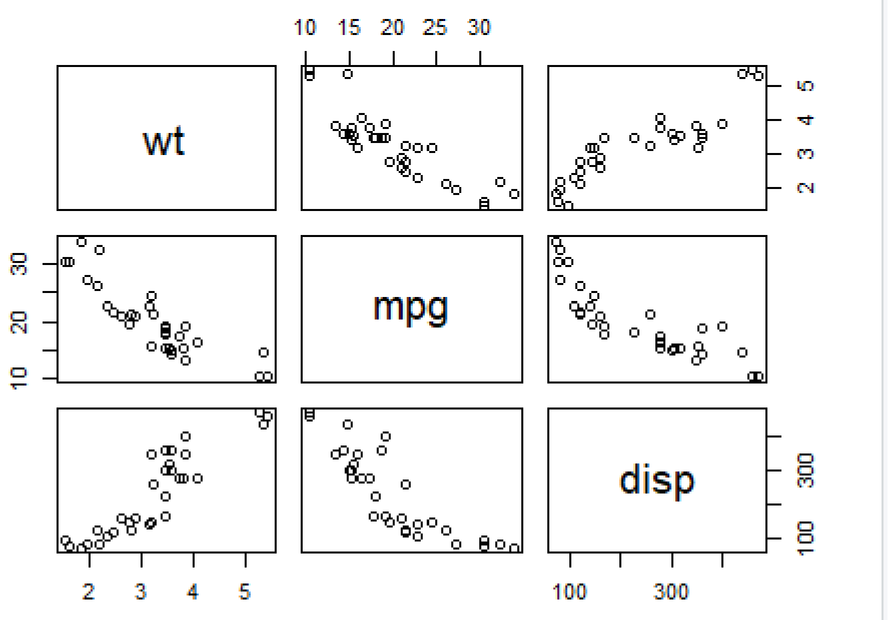
Step 6: Use the pairs() function again to create scatter plot matrices for the CSV data.

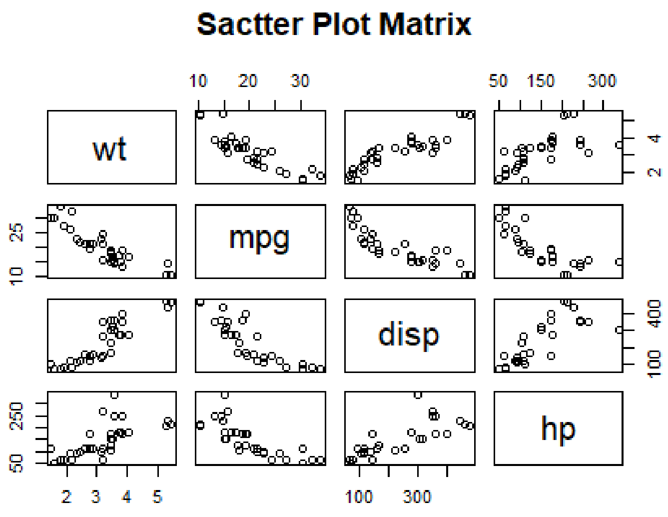
Step 7: End the program.

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")
```

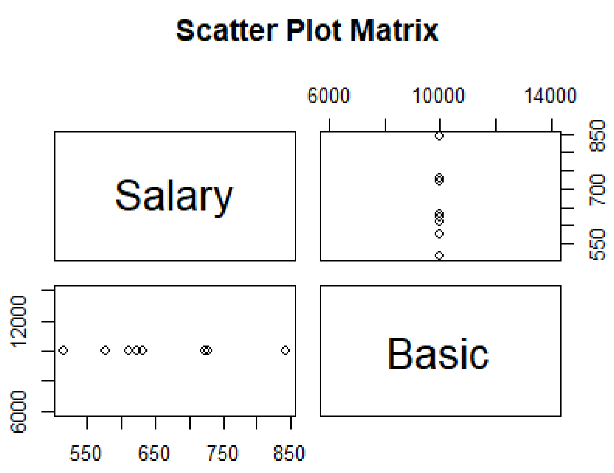
OUTPUT:






```
> setwd("D:/24PCA014/Practical/Scatter Plot")
> df<-read.csv("combined.csv")
> print(df)
  Unnamed..0 ID     Name ID.1 Salary  STARTDATE       DEPT Basic
1          1  1     Rick    1 623.30   1/1/2012         IT 10000
2          2  2      Dan    2 515.20  9/23/2013 Operations 10000
3          3  3 Michelle    3 611.00 11/15/2014         IT 10000
4          4  4     Ryan    4 729.00  5/11/2014         HR 10000
5          5  5     Gary    5 843.25  3/27/2015    Finance 10000
6          6  6     Nina    6 578.00  5/21/2013         IT 10000
7          7  7    Simon    7 632.80  7/30/2013 Operations 10000
8          8  8     Guru    8 722.50  6/17/2014    Finance 10000
> 
```

```
> v<-df[,c("Salary","Basic")]
> print(v)
  Salary Basic
1 623.30 10000
2 515.20 10000
3 611.00 10000
4 729.00 10000
5 843.25 10000
6 578.00 10000
7 632.80 10000
8 722.50 10000
```



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:

```
> data.frame <- fetch(ItemMaster)
> print(data.frame)
   ItemCode               ItemName ItemRate
1    MAT001      Plywood Sheet 18mm  1450.50
2    MAT002    Mild Steel Rod 12mm   980.00
3    MAT003        Cement Bag 50kg   370.25
4    MAT004      Iron Nail Pack 1kg   120.50
5    MAT005        Wooden Plank 6ft   860.75
6    MAT006            TMT Bar 10mm  1125.30
7    MAT007 Laminated Board 8x4ft  1620.00
8    MAT008 Galvanized Iron Sheet  1350.40
9    MAT009     Hardwood Beam 4x4in  2450.00
10   MAT010        Binding Wire 1kg   210.75
>
```

```
> data.frame <- fetch(Transaction)
> print(data.frame)
   UID ItemCode               ItemName ItemQty
1    1   MAT001      Plywood Sheet 18mm    5.00
2    2   MAT002    Mild Steel Rod 12mm   12.00
3    3   MAT003        Cement Bag 50kg   20.00
4    4   MAT004      Iron Nail Pack 1kg   15.50
5    5   MAT005        Wooden Plank 6ft    7.00
6    6   MAT006            TMT Bar 10mm    9.00
7    7   MAT007 Laminated Board 8x4ft    4.00
8    8   MAT008 Galvanized Iron Sheet    6.00
9    9   MAT009     Hardwood Beam 4x4in    3.00
10  10   MAT010        Binding Wire 1kg   10.00
11  11   MAT001      Plywood Sheet 18mm    2.50
12  12   MAT002    Mild Steel Rod 12mm    8.00
13  13   MAT003        Cement Bag 50kg   25.00
14  14   MAT004      Iron Nail Pack 1kg   18.00
15  15   MAT005        Wooden Plank 6ft    5.00
16  16   MAT006            TMT Bar 10mm    6.75
17  17   MAT007 Laminated Board 8x4ft    2.00
18  18   MAT008 Galvanized Iron Sheet    3.00
19  19   MAT009     Hardwood Beam 4x4in    1.00
20  20   MAT010        Binding Wire 1kg    7.50
>
```

```
> data.frame <- fetch(TransactionAmount)
> print(data.frame)
   UID ItemCode               ItemName ItemQty ItemRate NetAmount
1    1   MAT001     Plywood Sheet 18mm    5.00  1450.50  7252.500
2   11   MAT001     Plywood Sheet 18mm    2.50  1450.50  3626.250
3    2   MAT002    Mild Steel Rod 12mm   12.00   980.00 11760.000
4   12   MAT002    Mild Steel Rod 12mm    8.00   980.00  7840.000
5    3   MAT003        Cement Bag 50kg   20.00   370.25  7405.000
6   13   MAT003        Cement Bag 50kg   25.00   370.25  9256.250
7    4   MAT004     Iron Nail Pack 1kg   15.50   120.50  1867.750
8   14   MAT004     Iron Nail Pack 1kg   18.00   120.50  2169.000
9    5   MAT005       Wooden Plank 6ft    7.00   860.75  6025.250
10  15   MAT005       Wooden Plank 6ft    5.00   860.75  4303.750
11   6   MAT006           TMT Bar 10mm    9.00  1125.30 10127.700
12  16   MAT006           TMT Bar 10mm    6.75  1125.30  7595.775
13   7   MAT007 Laminated Board 8x4ft    4.00  1620.00  6480.000
14  17   MAT007 Laminated Board 8x4ft    2.00  1620.00  3240.000
15   8   MAT008 Galvanized Iron Sheet    6.00  1350.40  8102.400
16  18   MAT008 Galvanized Iron Sheet    3.00  1350.40  4051.200
17   9   MAT009    Hardwood Beam 4x4in    3.00  2450.00  7350.000
18  19   MAT009    Hardwood Beam 4x4in    1.00  2450.00  2450.000
19  10   MAT010       Binding Wire 1kg   10.00   210.75  2107.500
20  20   MAT010       Binding Wire 1kg    7.50   210.75  1580.625
>
> data.frame <- fetch(ItemAmount)
> print(data.frame)
   ItemCode               ItemName NetAmount
1    MAT001     Plywood Sheet 18mm 10878.750
2    MAT002    Mild Steel Rod 12mm 19600.000
3    MAT003        Cement Bag 50kg 16661.250
4    MAT004     Iron Nail Pack 1kg  4036.750
5    MAT005       Wooden Plank 6ft 10329.000
6    MAT006           TMT Bar 10mm 17723.475
7    MAT007 Laminated Board 8x4ft  9720.000
8    MAT008 Galvanized Iron Sheet 12153.600
9    MAT009    Hardwood Beam 4x4in  9800.000
10   MAT010       Binding Wire 1kg  3688.125
>
```

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:

```
> data.frame <- fetch(DepartmentAvg)
> print(data.frame)
      mark         DepartmentName
1  82.3333          Biotechnology
2  80.6667              Chemistry
3  68.3333      Civil Engineering
4  83.0000       Computer Science
5  90.0000 Electrical Engineering
6  86.6667            Electronics
7  88.0000 Information Technology
8  83.6667            Mathematics
9  78.3333 Mechanical Engineering
10 85.0000                Physics
>
```

```
> data.frame <- fetch(TotalMarkSubject)
> print(data.frame)
           SubjectName Mark StudentName         DepartmentName
1  Genetic Engineering   72       Kavin          Biotechnology
2       Linear Algebra   90       Kavin          Biotechnology
3      Web Development   85       Kavin          Biotechnology
4      Data Structures   88     Sathish              Chemistry
5    Organic Chemistry   78     Sathish              Chemistry
6       Thermodynamics   76     Sathish              Chemistry
7    Analog Electronics  70 Dhamodharan      Civil Engineering
8  Genetic Engineering   68 Dhamodharan      Civil Engineering
9  Structural Analysis   67 Dhamodharan      Civil Engineering
10    Circuit Analysis   79 Thamil Mani       Computer Science
11     Data Structures   88 Thamil Mani       Computer Science
12      Thermodynamics   82 Thamil Mani       Computer Science
13  Analog Electronics   91    Sri Hari Electrical Engineering
14    Circuit Analysis   91    Sri Hari Electrical Engineering
15 Structural Analysis   88    Sri Hari Electrical Engineering
16  Analog Electronics   85   Divakaran            Electronics
17 Genetic Engineering   90   Divakaran            Electronics
18     Web Development   85   Divakaran            Electronics
19      Linear Algebra   92      Naveen Information Technology
20   Quantum Mechanics   78      Naveen Information Technology
21     Web Development   94      Naveen Information Technology
22      Linear Algebra   89      Pranav            Mathematics
23   Organic Chemistry   82      Pranav            Mathematics
24   Quantum Mechanics   80      Pranav            Mathematics
25    Circuit Analysis   75      Mukesh Mechanical Engineering
26 Structural Analysis   84      Mukesh Mechanical Engineering
27      Thermodynamics   76      Mukesh Mechanical Engineering
28     Data Structures   88       Rahul                Physics
29   Organic Chemistry   77       Rahul                Physics
30   Quantum Mechanics   90       Rahul                Physics
> 
```

```
> data.frame <- fetch(ScoreSubject)
> print(data.frame)
           SubjectName HighestScore LowestScore
1   Analog Electronics           91          70
2     Circuit Analysis           91          75
3      Data Structures           88          88
4  Genetic Engineering           90          68
5       Linear Algebra           92          89
6    Organic Chemistry           82          77
7    Quantum Mechanics           90          78
8  Structural Analysis           88          67
9       Thermodynamics           82          76
10     Web Development           94          85
> 
```

```
Console   Terminal   Jobs
R 4.1.1 · ~/
> data.frame <- fetch(Transactions)
> print(data.frame)
   UID StudentID SubjectID DeptID years Semester mark TestName
1    1         1         1      1  2024        I   88  Midterm
2    2         2         2      2  2024       II   76    Final
3    3         3         3      3  2025      III   91     Quiz
4    4         4         4      4  2023       IV   67    Final
5    5         5         5      5  2025        V   85  Midterm
6    6         6         6      6  2024       VI   72     Quiz
7    7         7         7      7  2023      VII   94    Final
8    8         8         8      8  2025     VIII   89  Midterm
9    9         9         9      9  2024      III   90    Final
10  10        10        10     10  2023       II   78  Midterm
11  11         1         2      1  2024       II   82    Final
12  12         2         3      2  2024      III   75     Quiz
13  13         3         4      3  2025       IV   88  Midterm
14  14         4         5      4  2023        V   70     Quiz
15  15         5         6      5  2025       VI   90    Final
16  16         6         7      6  2024      VII   85  Midterm
17  17         7         8      7  2023     VIII   92     Quiz
18  18         8         9      8  2025      III   80    Final
19  19         9        10      9  2024       IV   77  Midterm
20  20        10         1     10  2023        V   88    Final
21  21         1         3      1  2024      III   79     Quiz
22  22         2         4      2  2024       IV   84  Midterm
23  23         3         5      3  2025        V   91    Final
24  24         4         6      4  2023       VI   68     Quiz
25  25         5         7      5  2025      VII   85  Midterm
26  26         6         8      6  2024     VIII   90    Final
27  27         7         9      7  2023      III   78     Quiz
28  28         8        10      8  2025       IV   82  Midterm
29  29         9         1      9  2024        V   88    Final
30  30        10         2     10  2023       VI   76     Quiz
> 
```

```
Console  Terminal ×  Jobs ×
R 4.1.1 · ~/
> data.frame <- fetch(subject)
> print(data.frame)
   SubjectID          SubjectName years Semester DeptID
1          1      Data Structures  2024        I      1
2          2       Thermodynamics  2024       II      2
3          3     Circuit Analysis  2025      III      3
4          4  Structural Analysis  2023       IV      4
5          5   Analog Electronics  2025        V      5
6          6  Genetic Engineering  2024       VI      6
7          7      Web Development  2023      VII      7
8          8       Linear Algebra  2025     VIII      8
9          9    Quantum Mechanics  2024      III      9
10        10    Organic Chemistry  2023       II     10
>
```

```
Console  Terminal ×  Jobs ×
R 4.1.1 · ~/
> data.frame <- fetch(Department)
> print(data.frame)
   DeptID           DepartmentName
1       1         Computer Science
2       2   Mechanical Engineering
3       3   Electrical Engineering
4       4        Civil Engineering
5       5              Electronics
6       6            Biotechnology
7       7   Information Technology
8       8              Mathematics
9       9                  Physics
10     10                Chemistry
>
```

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).

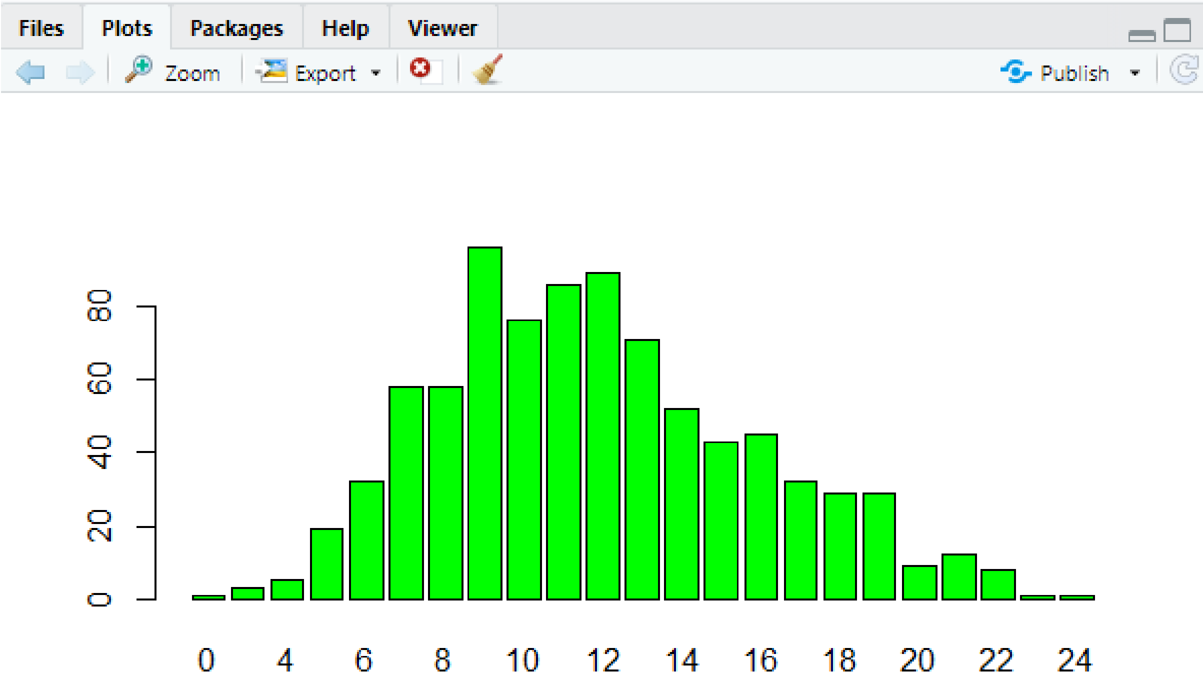
Step 10: Stop the program

Program :

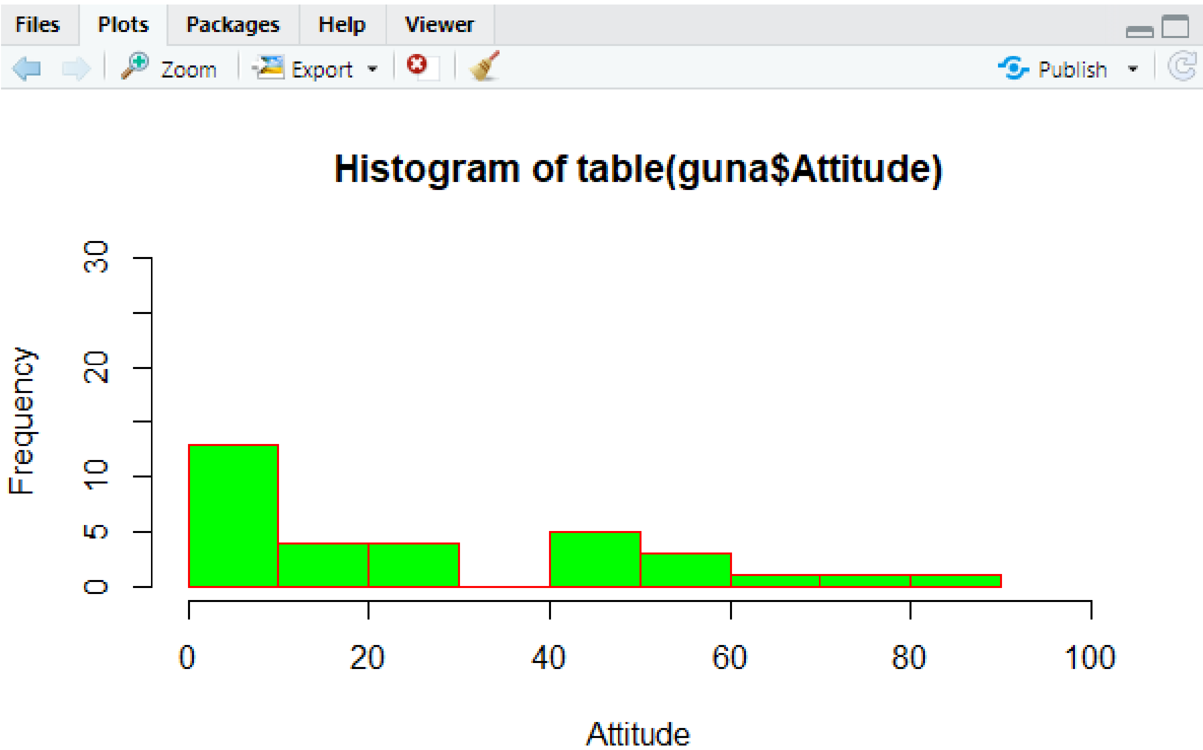
```
getwd()
setwd("C:/Users/MCA-007/Documents/R prog")
guna = read.csv("guna.csv")
head(guna)
summary(guna)
hist(table(guna$Apptitude),xlab = "Aptitude",col = "Green",border = "red",xlim = c(0,100),ylim =
c(0,30),breaks = 5)
#bar plot
barplot(table(guna$gunas),col = "green")
barplot(table(guna$Attitude),col = "green")
barplot(table(guna$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:

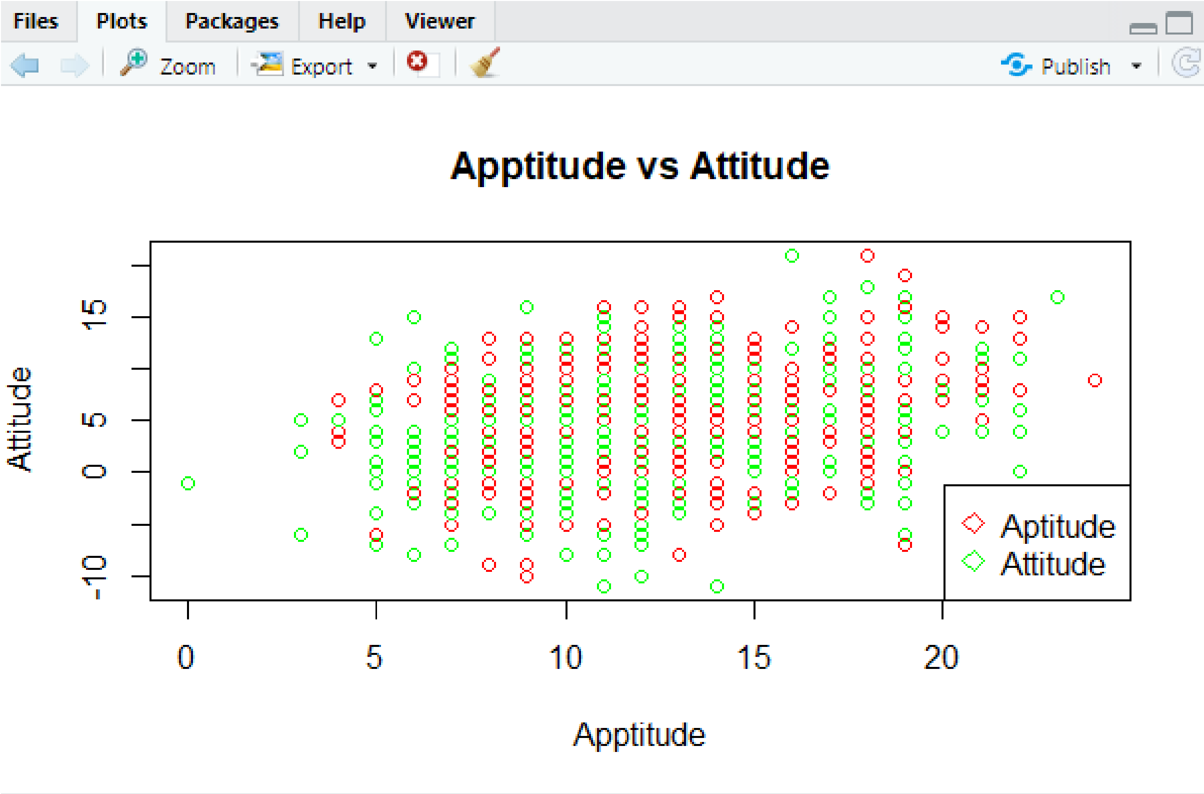
Bar Plot



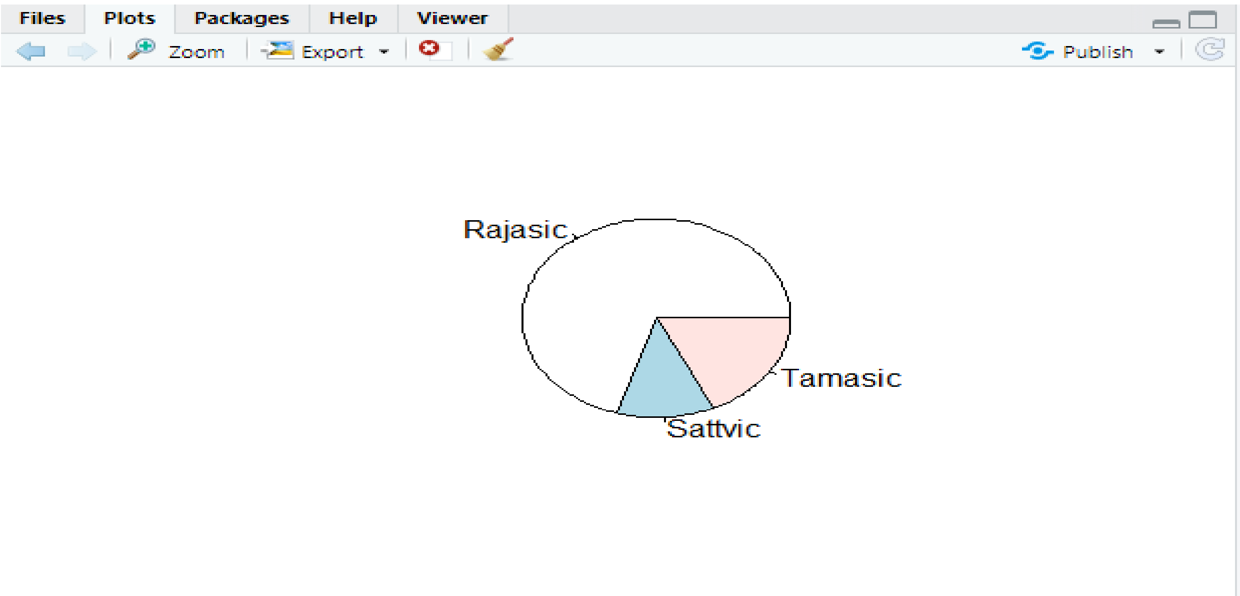
Box Plot



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

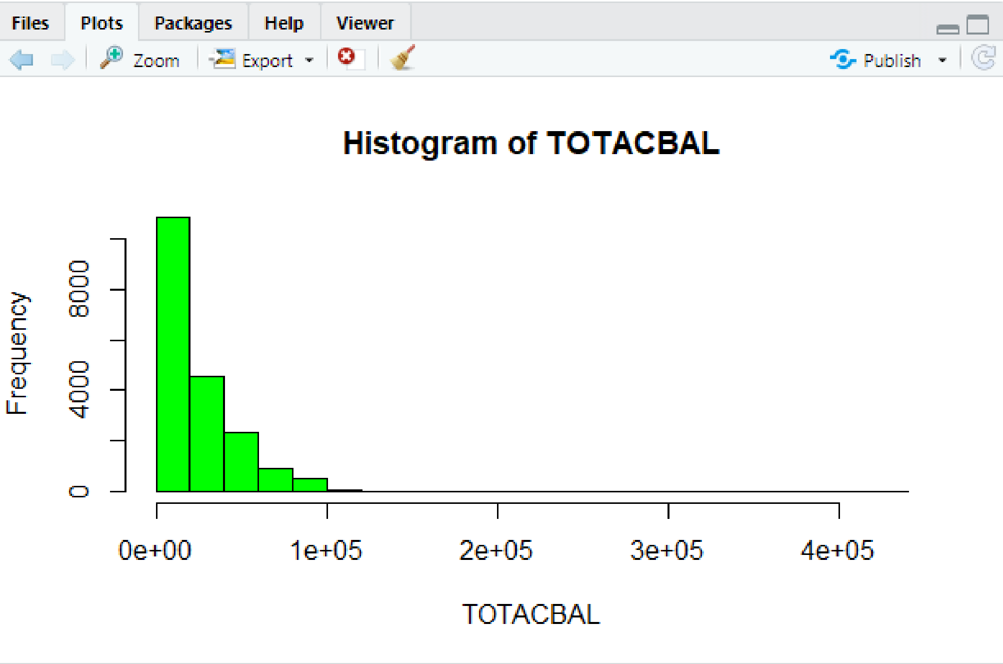
Program :

```
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)
```

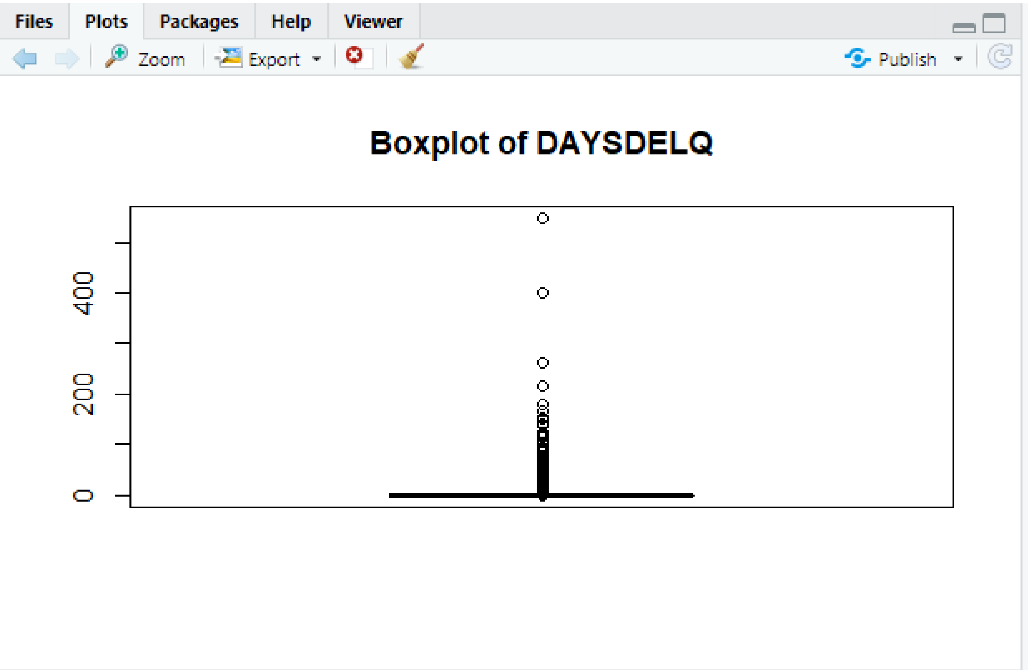
```
d <- as.data.frame(table(credit$DEFAULT))
pie(d$Freq, labels=paste(d$Var1, ":", d$Freq),
    col=c("green","red"), main="Pie Chart of Default Status")
nums <- credit[, sapply(credit, is.numeric)]
cor_matrix <- cor(nums, use="complete.obs")
```

OUTPUT:

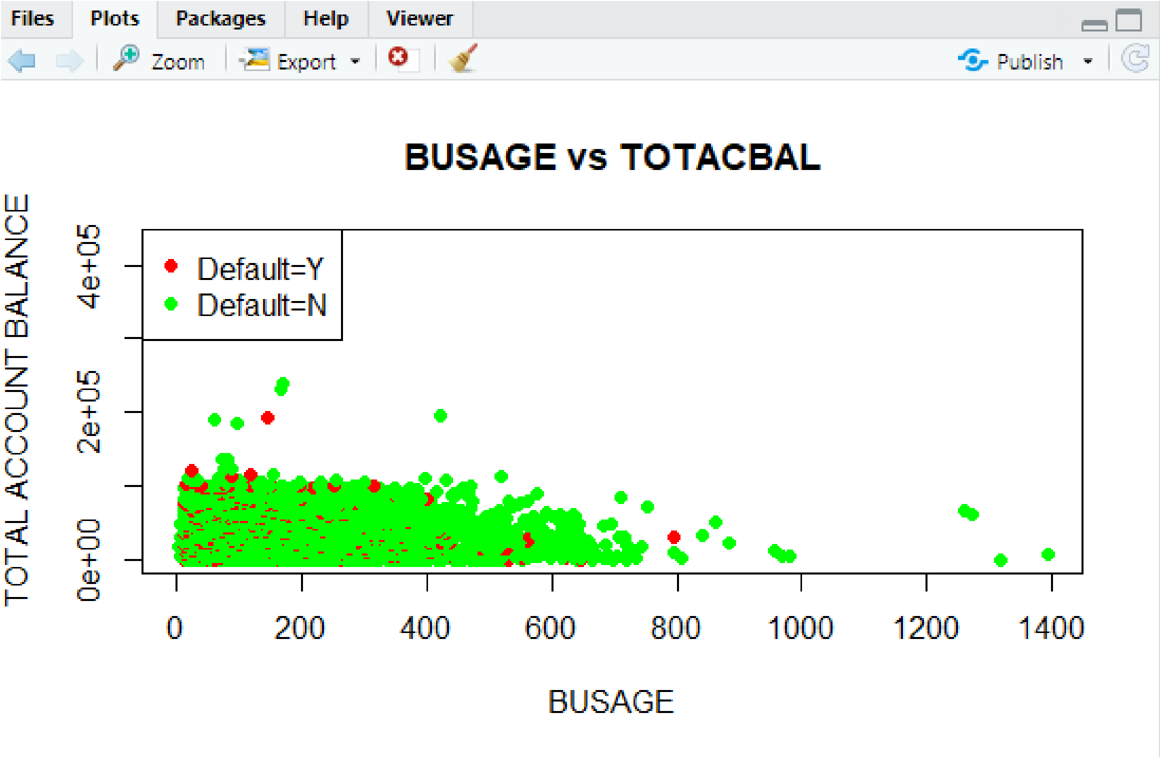
Bar Plot



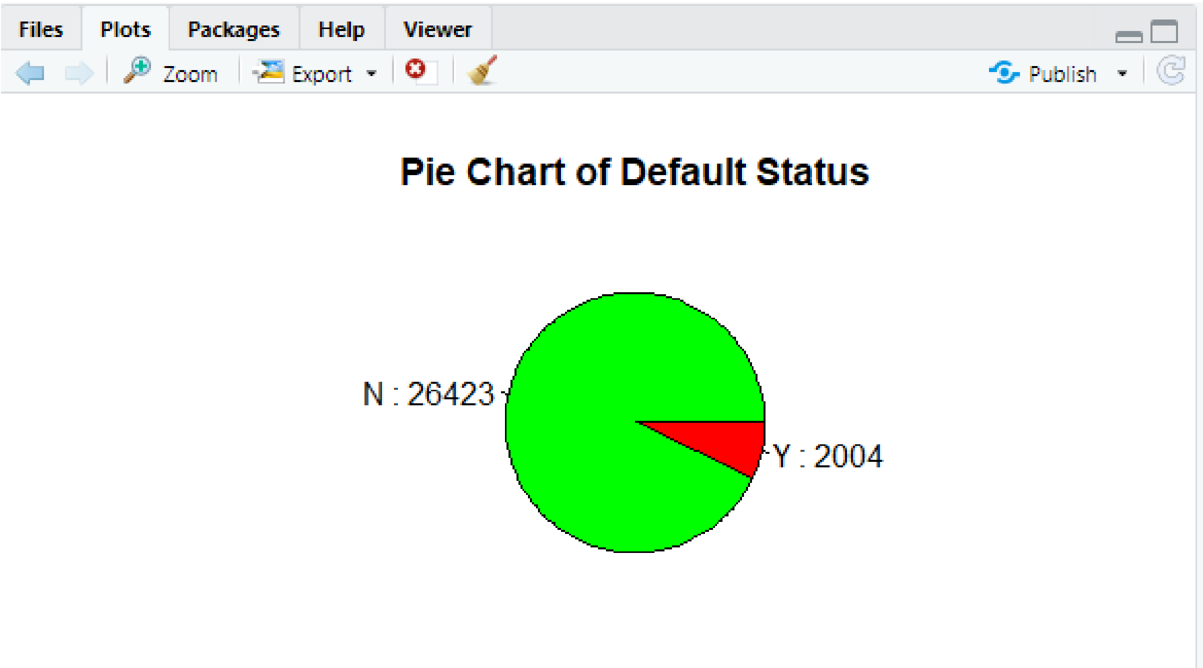
Box Plot



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.

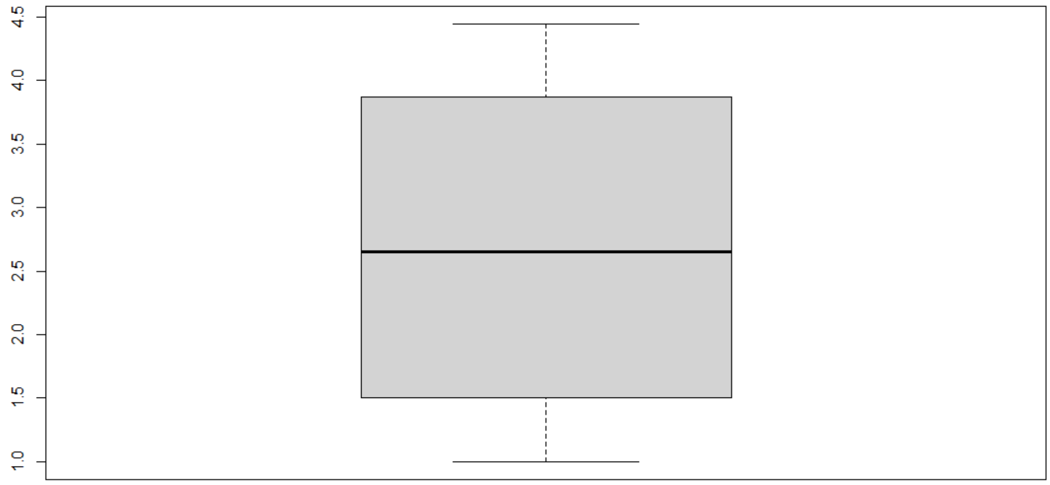
Step 11: stop the process

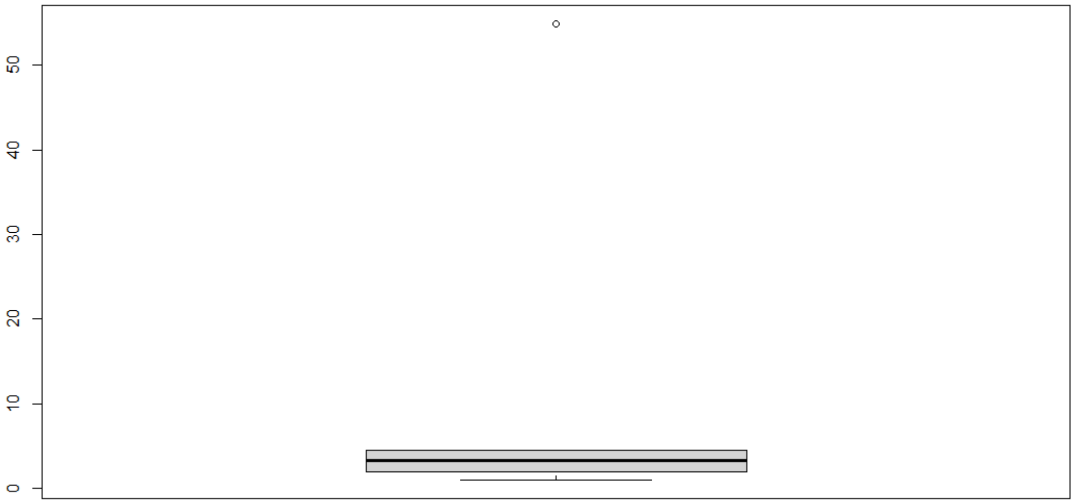
Program:

```
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:

$$(n \times \sum xy) - (\sum x \times \sum y)$$

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:

$$\sum(xi - \bar{x})(yi - \bar{y})$$

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 the correlation

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

```
> #a
> # Input data
> x <- c(10, 20, 30, 40, 50)
> y <- c(15, 25, 35, 45, 55)
>
> # Length of data
> n <- length(x)
>
> # Apply formula
> 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))
>
> # Pearson correlation
> r <- numerator / denominator
> print(paste("Pearson Correlation (Formula 1):", r))
[1] "Pearson Correlation (Formula 1): 1"
```

Formula 2

```
> #b
> # Input data
> x <- c(10, 20, 30, 40, 50)
> y <- c(15, 25, 35, 45, 55)
>
> # Means
> mean_x <- mean(x)
> mean_y <- mean(y)
>
> # Numerator: sum of product of deviations
> 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))
>
> # Correlation
> r2 <- numerator / denominator
> print(paste("Correlation using Mean (Formula 2):", r2))
[1] "Correlation using Mean (Formula 2): 1"
```

Built in function in R

```
> #c
> # Input data
> x <- c(10, 20, 30, 40, 50)
> y <- c(15, 25, 35, 45, 55)
>
> # Built-in correlation function
> r_builtin <- cor(x, y, method = "pearson")
> print(paste("Built-in cor() function:", r_builtin))
[1] "Built-in cor() function: 1"
```

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: $\sum x$, $\sum y$, $\sum x^2$, $\sum y^2$, and $\sum xy$.

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

$$b = n\sum xy - \sum x \sum y \;/\; n\sum x2 - (\sum x)2$$

Step 6: Form the linear regression equation: y=a+bxy = a + bxy=a+bx, where $a = \bar{y} - b\bar{x}$

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

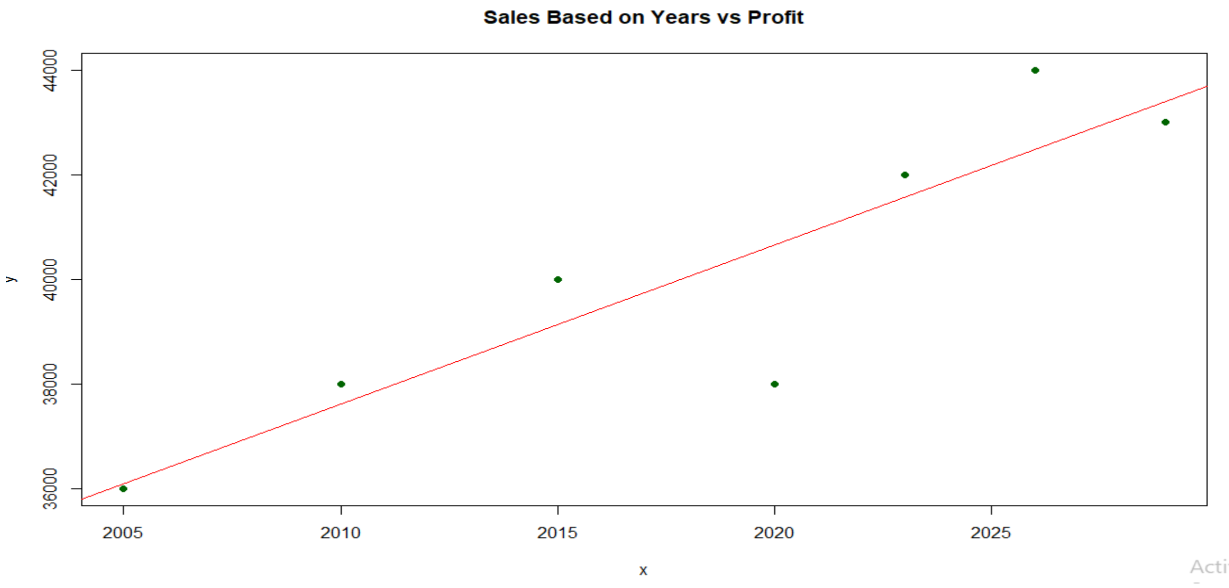
Program:

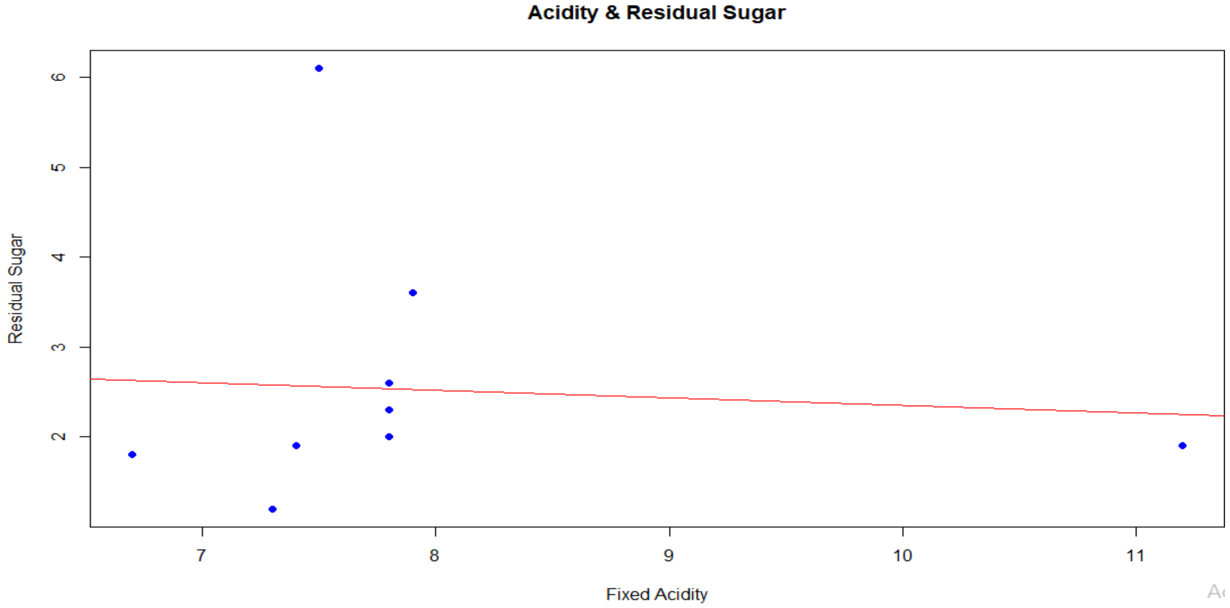
```
x <- c(1, 2, 3, 4, 5, 6, 7)

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)
```

```r
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:

A) 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

```
R 4.1.1 · ~/
> 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))
   mpg disp  hp    wt
1 21.0  160 110 2.620
2 21.0  160 110 2.875
3 22.8  108  93 2.320
4 21.4  258 110 3.215
5 18.7  360 175 3.440
6 18.1  225 105 3.460
> model <- lm(mpg~disp+hp+wt,data = input)
> #print(summary(model))
> cat("### the coefficient value ###","\n")
### the cofficient value ###
> a <- coef(model)[1]
> print(a)
(Intercept)
   31.58781
> xdisp <- coef(model)[2]
> xhp <- coef(model)[3]
> xwt <- coef(model)[4]
> print(xdisp)
       disp
0.005906507
> print(xhp)
        hp
-0.0516646
> print(xwt)
       wt
-1.951904
> #print(summary(model))
> paste("y~",a,"+",xdisp,"*x1","+",xhp,"*x2",xwt,"*x3")
[1] "y~ 31.587811106649 + 0.00590650682358486 *x1 + -0.0516646012658113 *x2 -1.95190359964048 *x3"
> disp=221; hp=102; wt=2.91
> a1 <- data.frame(disp,hp,wt)
> result <- predict(model,a1)
> print(result)
       1
21.94332
> |
```

B)Data Provided through build in table in R

```
Console  Terminal ×  Jobs ×
R 4.1.1 · ~/R/R Programming/26. Multiple Linear Regression/
> #mtcars
> #print(mtcars)
> input<-mtcars[,c("mpg","disp","hp","wt")]
> print(head(input))
                   mpg disp  hp    wt
Mazda RX4         21.0  160 110 2.620
Mazda RX4 Wag     21.0  160 110 2.875
Datsun 710        22.8  108  93 2.320
Hornet 4 Drive    21.4  258 110 3.215
Hornet Sportabout 18.7  360 175 3.440
Valiant           18.1  225 105 3.460
> input<-mtcars[,c("mpg","disp","hp","wt")]
> model<-lm(mpg~disp+hp+wt,data = input)
> #print(summary(model))
> cat("### the cofficient value ###","\n")
### the cofficient value ###
> a<-coef(model)[1]
> print(a)
(Intercept)
   37.10551
> xdisp<-coef(model)[2]
> xhp<-coef(model)[3]
> xwt<-coef(model)[4]
> print(xdisp)
         disp
-0.0009370091
> print(xhp)
         hp
-0.03115655
> print(xwt)
       wt
-3.800891
> #print(summary(model))
> paste("y~",a,"+",xdisp,"*x1","+",xhp,"*x2",xwt,"*x3")
[1] "y~ 37.1055052690318 + -0.000937009081489667 *x1 + -0.0311565508299456 *x2 -3.80089058263761 *x3"
> disp=221
> hp=102
> wt=2.91
> a1<-data.frame(disp,hp,wt)
> result<-predict(model,a1)
> print(result)
       1
22.65987
> |
```

C)Data Provided through CSV File:

```
Console   Terminal ×   Jobs ×
R 4.1.1 · ~/R/R Programming/26. Multiple Linear Regression/
> setwd("C:/Users/MCA/Documents/R/R Programming/26. Multiple Linear Regression")
> data <- read.csv("car_data.csv")
> print(head(data))
   mpg disp  hp    wt
1 21.0  160 110 2.620
2 21.0  160 110 2.875
3 22.8  108  93 2.320
4 21.4  258 110 3.215
5 18.7  360 175 3.440
6 18.1  225 105 3.460
> input <- data[,c("mpg","disp","hp","wt")]
> print(head(input))
   mpg disp  hp    wt
1 21.0  160 110 2.620
2 21.0  160 110 2.875
3 22.8  108  93 2.320
4 21.4  258 110 3.215
5 18.7  360 175 3.440
6 18.1  225 105 3.460
> model <- lm(mpg~disp+hp+wt,data = input)
> #(summary(model))
> cat("### the cofficient value ###","\n")
### the cofficient value ###
> a <- coef(model)[1]
> print(a)
(Intercept)
   31.58781
> xdisp <- coef(model)[2]
> xhp <- coef(model)[3]
> xwt <- coef(model)[4]
> print(xdisp)
        disp
0.005906507
> print(xhp)
         hp
-0.0516646
> print(xwt)
        wt
-1.951904
> #print(summary(model))
> paste("y~",a,"+",xdisp,"*x1","+",xhp,"*x2",xwt,"*x3")
[1] "y~ 31.587811106649 + 0.00590650682358486 *x1 + -0.0516646012658113 *x2 -1.95190359964048 *x3"
> disp=221; hp=102; wt=2.91
> a1 <- data.frame(disp,hp,wt)
> result <- predict(model,a1)
> print(result)
       1
21.94332
>
```

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:

$$logit(p)=ln(p/1-p)=\beta0+\beta1X$$

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)
```

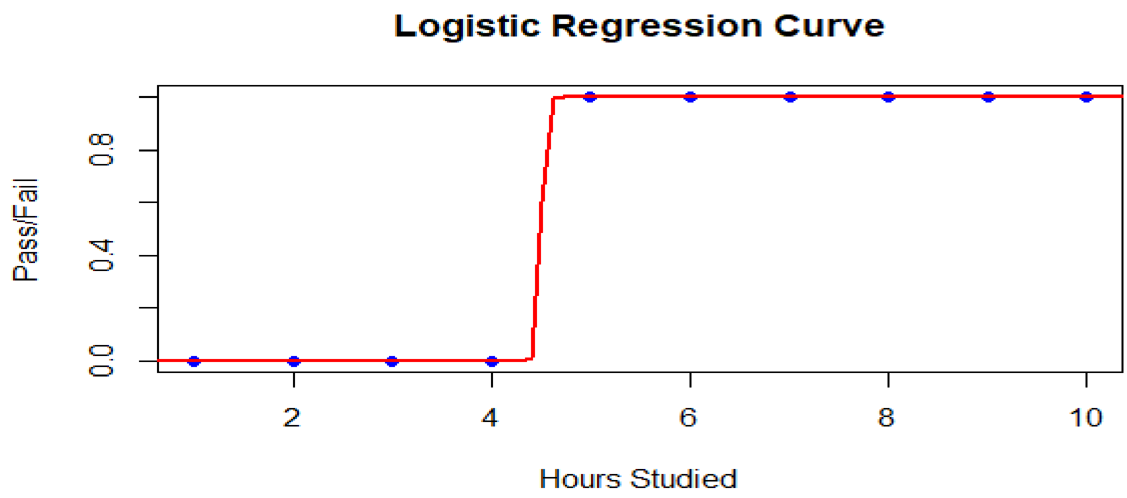
C)Data Provided through CSV File:

```
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:

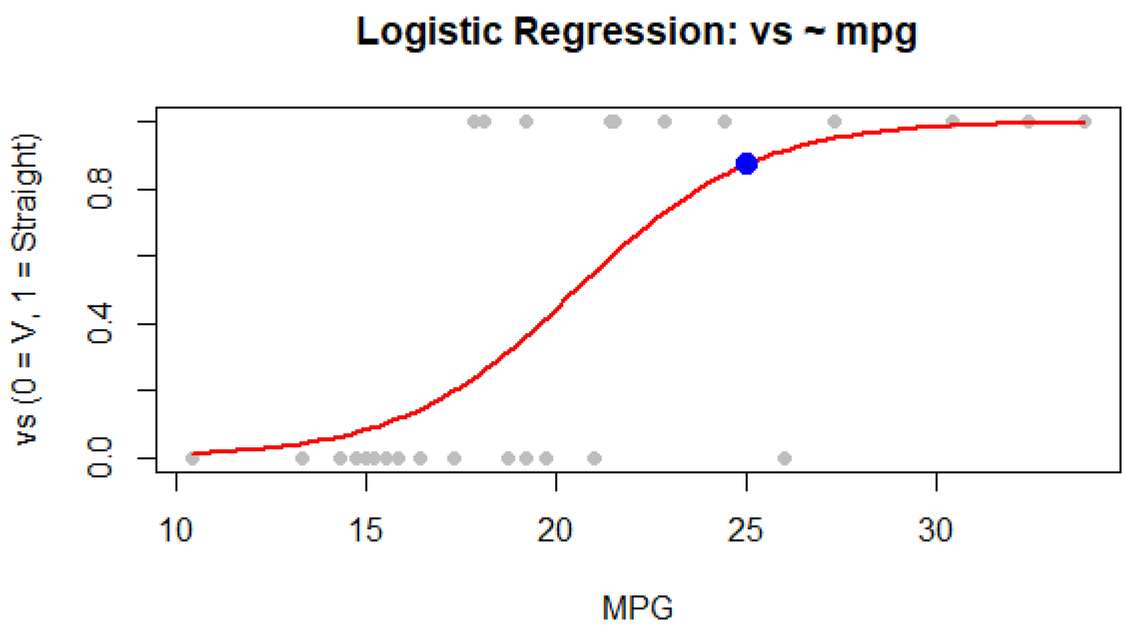
A) Data Provided Through Vector

```
> # Model summary
> summary(model)

Call:
glm(formula = pass_fail ~ hours_studied, family = "binomial",
    data = student_data)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-2.042e-05  -2.110e-08   2.110e-08   2.110e-08   2.105e-05

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)     -200.37  265802.23  -0.001    0.999
hours_studied     44.52   58511.58   0.001    0.999

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1.3460e+01  on 9  degrees of freedom
Residual deviance: 8.6042e-10  on 8  degrees of freedom
AIC: 4

Number of Fisher Scoring iterations: 25

> # 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")
Probability of passing if studied 6 hours: 1
>
```



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")
Prob at mpg 25: 0.873
>
```



C) Data Provided through CSV File

```
> # Model summary
> summary(model)

Call:
glm(formula = pass_fail ~ hours_studied, family = "binomial",
    data = data)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-2.042e-05  -2.110e-08   2.110e-08   2.110e-08   2.105e-05

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)     -200.37  265802.23  -0.001    0.999
hours_studied     44.52   58511.58   0.001    0.999

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1.3460e+01  on 9  degrees of freedom
Residual deviance: 8.6042e-10  on 8  degrees of freedom
AIC: 4

Number of Fisher Scoring iterations: 25

> # 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))
Probability of passing if studied 6 hours: 1
> 
```

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 through Vectors

b) Data provided through built in tables in R

c) Data provided through CSV 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:

$$y=a\times e\hat{}(b\times x)$$

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:

A) 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

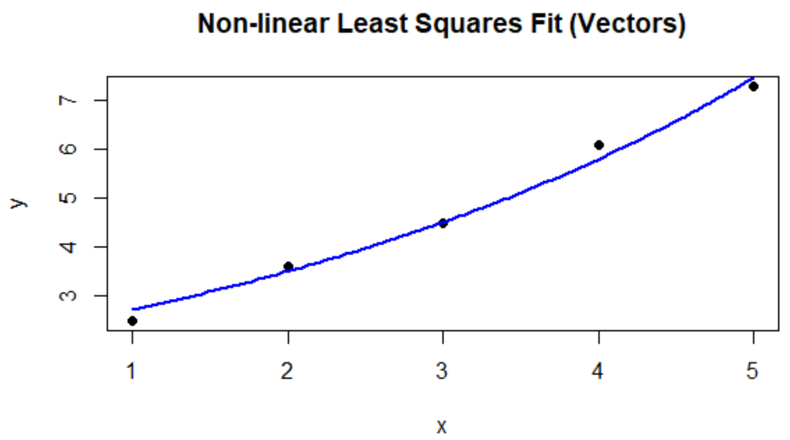
```
data <- pressure
# Inspect the data
head(data)
# Fit non-linear model
# Model: pressure = a * exp(b * temperature)
nls_model <- nls(pressure ~ a * exp(b * temperature), data = data, start = list(a = 1, b = 0.01))
# Summary of the model
summary(nls_model)
# Plot original data points
plot(data$temperature, data$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:

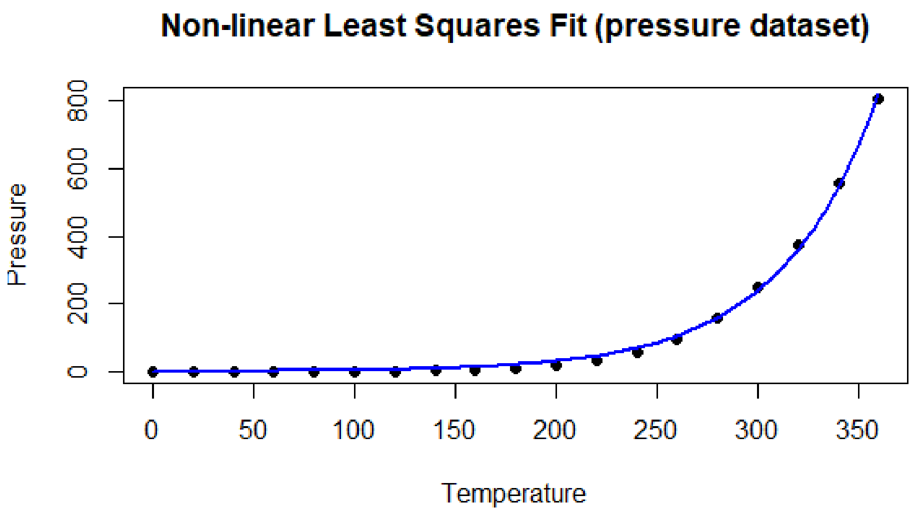
A) Data provided through vector



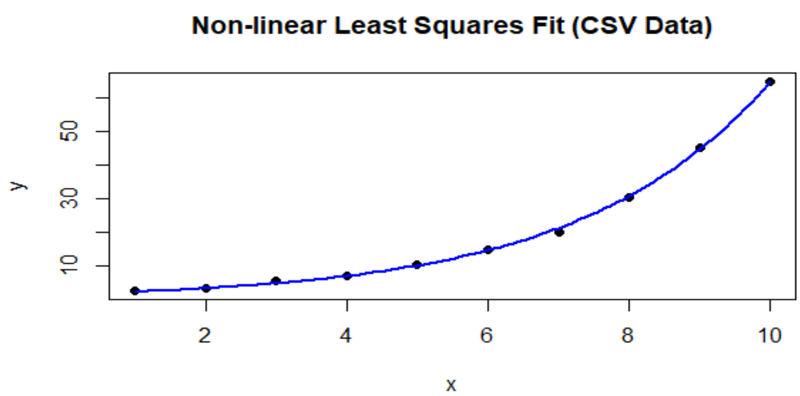
B) Data Provided through build in table in R

```
> # Summary of the model
> summary(nls_model)

Formula: pressure ~ a * exp(b * temperature)

Parameters:
  Estimate Std. Error t value Pr(>|t|)
a 0.507554   0.066385   7.646 6.73e-07 ***
b 0.020520   0.000379  54.142  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 9.839 on 17 degrees of freedom

Number of iterations to convergence: 9
Achieved convergence tolerance: 1.559e-06
```

```
> # Summary of the model
> summary(nls_model)

Formula: y ~ a * exp(b * x)

Parameters:
  Estimate Std. Error t value Pr(>|t|)
a 1.596017   0.076747    20.8 3.00e-08 ***
b 0.370541   0.005234    70.8 1.76e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6083 on 8 degrees of freedom

Number of iterations to convergence: 8
Achieved convergence tolerance: 1.99e-07
```



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))
```

<u>OUTPUT :</u>

a) **Using Formula :**

```
> x<-4
> y<-dbinom(x,10,0.5)
> print(y)
[1] 0.2050781
>
>
> #dbinom
> x<-0
> y0<-dbinom(x,4,0.5)
> print(y0)
[1] 0.0625
>
> x<-1
> y1<-dbinom(x,4,0.5)
> print(y1)
[1] 0.25
>
> x<-2
> y2<-dbinom(x,4,0.5)
> print(y2)
[1] 0.375
>
> x<-3
> y3<-dbinom(x,4,0.5)
> print(y3)
[1] 0.25
>
> x<-4
> y4<-dbinom(x,4,0.5)
> print(y4)
[1] 0.0625
>
> y = y0+y1+y2+y3+y4
> print(y)
[1] 1
>
```

b) Using dbinorm,pbinorm,qbinorm,rbinorm

```
> #pbinom
>
> x1 <- 2
> y1 <- pbinom(x1,4,0.5)
> print(y1)
[1] 0.6875
>
> x2 <- 3
> y2 <- pbinom(x2,4,0.5)
> print(y2)
[1] 0.9375
>
> y = y2-y1
> print(y)
[1] 0.25
>
> #qbinom
>
> x <- qbinom(0.375,4,0.5)
> print(x)
[1] 2
>
> #rbinom
>
> x<- rbinom(8,150,0.4)
> print(x)
[1] 71 65 64 50 62 49 65 63
>
> 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))
[1] 0.9274445
>
> print(pbinom(4,12,0.2))
[1] 0.9274445
>
```

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 :

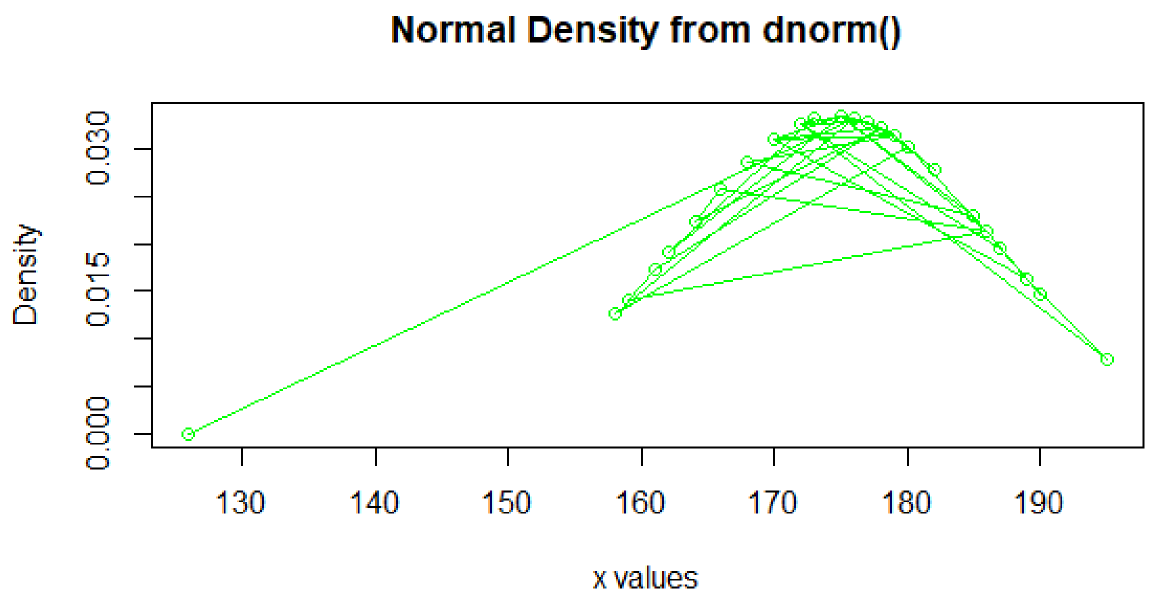
a) Using formula

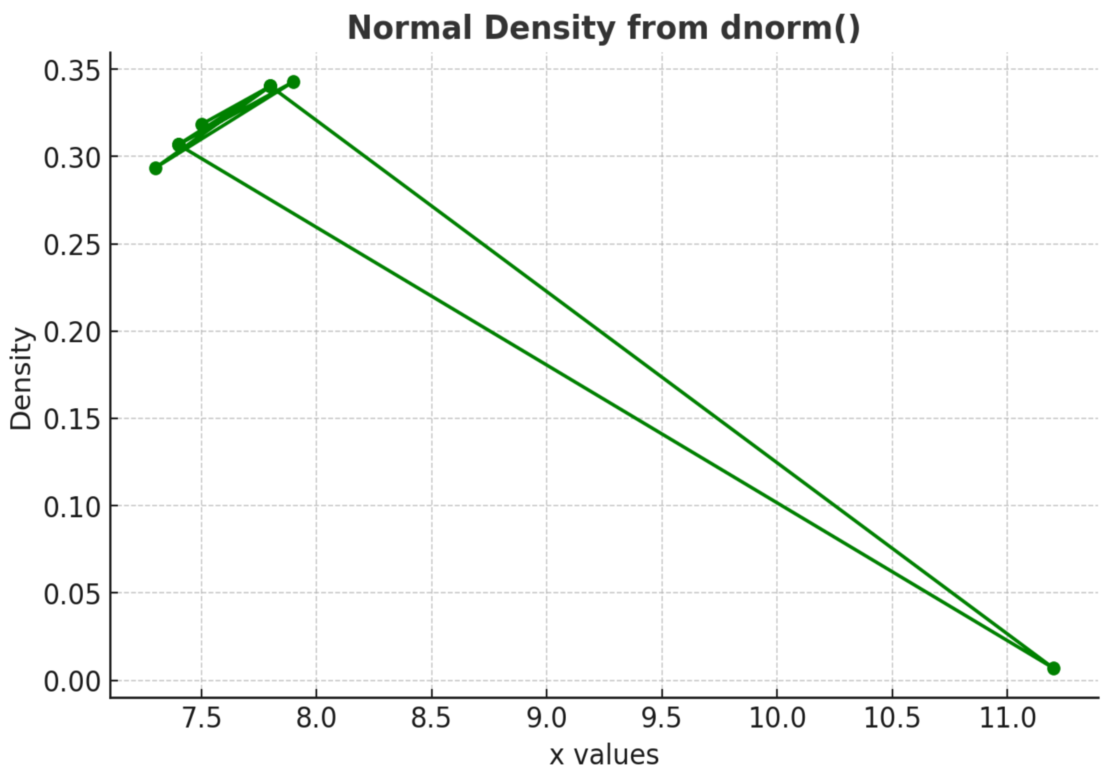
```
x <- c(126, 176, 158, 180, 186, 166, 166, 164, 178, 170,
       189, 195, 172, 187, 190, 186, 185, 168, 179, 178,
       182, 179, 170, 175, 186, 159, 161, 178, 175, 185,
       175, 162, 173, 172, 177, 175, 172, 177, 180)
mean_x <- mean(x)
sd_x <- sd(x)
print(paste("Mean:", mean_x))
print(paste("Standard 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

```
R 4.1.1 · ~/
> #ppois
>
> a = ppois(16,lambda = 12,lower.tail = TRUE)
> b = ppois(16,lambda = 12,lower.tail = FALSE)
> print(a+b)
[1] 1
>
> #dpois
> n=3000
> p=0.001
> r=6
> lambda = n*p
> b<-exp(-lambda)*lambda^6/factorial(6)
> print(b)
[1] 0.05040941
>
> dpois(6,lambda)
[1] 0.05040941
>
```

b)Using dpois,ppois,qpois

```
R 4.1.1 · ~/
> 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)
[1] "6.14421235332821e-06   7.37305482399385e-05   0.000442383289439631   0.0017695331577
5852   0.00530859947327558   0.0127406387358614  "
> print(k1+k2+k3+k4+k5+k6)
[1] 0.02034103
>
>
> ppois(3,lambda,lower.tail = TRUE)
[1] 0.002291791
> ppois(3,lambda,lower.tail = FALSE)
[1] 0.9977082
>
> lambda <- 12
> samples <- rpois(10, lambda)
> print(samples)
 [1] 10 21 12 11 15 10  9 13 10  6
>
```

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:

```
Console   Terminal ×   Background Jobs ×
R 4.1.1 · ~/
> print(paste("Sum of x1:", sum_x1, "Sum of x2:", sum_x2, "Sum of x3:", sum_x3))
[1] "Sum of x1: 50 Sum of x2: 40 Sum of x3: 60"
> print(paste("Sum of squares x1:", sum_sq_x1, "Sum of squares x2:", sum_sq_x2, "Sum
f squares x3:", sum_sq_x3))
[1] "Sum of squares x1: 530 Sum of squares x2: 336 Sum of squares x3: 734"
> sum_all <- sum_x1 + sum_x2 + sum_x3
> print(paste("Total sum:", sum_all))
[1] "Total sum: 150"
> cat("SS Treatment:", ss_treatment, "\n")
SS Treatment: 40
> cat("SS Error:", ss_error, "\n")
SS Error: 60
> cat("SS Total:", ss_total, "\n")
SS Total: 100
> cat("Degrees of Freedom (Treatment):", df_treatment, "\n")
Degrees of Freedom (Treatment): 2
> cat("Degrees of Freedom (Error):", df_error, "\n")
Degrees of Freedom (Error): 12
> cat("Mean Square Treatment:", ms_treatment, "\n")
Mean Square Treatment: 20
> cat("Mean Square Error:", ms_error, "\n")
Mean Square Error: 5
> cat("F value:", F_value, "\n")
F value: 4
```

```
Console   Terminal ×   Background Jobs ×
R 4.1.1 · ~/
> summary(anova_result)
            Df Sum Sq Mean Sq F value Pr(>F)  
groups       2     40      20       4 0.0467 *
Residuals   12     60       5                 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> 
```

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 :

```
Mean: 44.44444444444444
Median: 50.0
Mode: 50  (Count: 3 )
Correlation Coefficient: 0.9999999999999999
```

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 :

```
RangeIndex: 36 entries, 0 to 35
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Rank             36 non-null     int64
 1   Title            36 non-null     object
 2   Genre            36 non-null     object
 3   Description      36 non-null     object
 4   Director         36 non-null     object
 5   Actors           36 non-null     object
 6   Year             36 non-null     int64
 7   Runtime          36 non-null     int64
 8   Rating           36 non-null     float64
 9   Votes            36 non-null     int64
 10  Revenue_million  36 non-null     float64
 11  Metascore        36 non-null     float64
 12  Rating_category  36 non-null     object
```

```
Genre
Action,Adventure,Sci-Fi       5
Animation,Adventure,Comedy    4
Adventure,Drama,Sci-Fi        2
Drama                         2
Biography,Drama               2
Comedy,Drama,Music            1
Action,Adventure,Fantasy      1
Action,Crime,Drama            1
Action,Adventure,Comedy       1
Drama,Mystery,Sci-Fi          1
Adventure,Drama,War           1
Action,Sci-Fi                 1
Crime,Drama,Mystery           1
Biography,Comedy,Crime        1
Adventure,Drama,Fantasy       1
Action,Thriller               1
Adventure,Drama,Thriller      1
Crime,Drama,Thriller          1
Mystery,Thriller              1
Drama,Western                 1
Name: count, dtype: int64
```

```
Rank                                                           7
Title                                                 La La Land
Genre                                         Comedy,Drama,Music
Description    A jazz pianist falls for an aspiring actress i...
Director                                          Damien Chazelle
Actors         Ryan Gosling, Emma Stone, Rosemarie DeWitt, J....
Year                                                        2016
Runtime                                                      128
Rating                                                       8.3
Votes                                                     258682
Revenue_million                                           151.06
Metascore                                                   93.0
Rating_category                                             Good
Name: 1, dtype: object
```

	Rank	Title	Genre	Description	Director	Actors	Year	Runtime	Rating	Votes	Revenue_million
0	1	Guardians of the Galaxy	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014	121	8.1	757074	333.13
1	7	La La Land	Comedy,Drama,Music	A jazz pianist falls for an aspiring actress i...	Damien Chazelle	Ryan Gosling, Emma Stone, Rosemarie DeWitt, J....	2016	128	8.3	258682	151.06
4	37	Interstellar	Adventure,Drama,Sci-Fi	A team of explorers travel through a wormhole ...	Christopher Nolan	Matthew McConaughey, Anne Hathaway, Jessica Ch...	2014	169	8.6	1047747	187.99
5	51	Star Wars: Episode VII - The Force	Action,Adventure,Fantasy	Three decades after the defeat of the Galactic	J.J. Abrams	Daisy Ridley, John Boyega, Oscar Isaac,	2015	136	8.1	661608	936.63
6	55	The Dark Knight	Action,Crime,Drama	When the menace known as the Joker wreaks havo...	Christopher Nolan	Christian Bale, Heath Ledger, Aaron Eckhart,Mi...	2008	152	9.0	1791916	533.32
7	68	Mad Max: Fury Road	Action,Adventure,Sci-Fi	A woman rebels against a tyrannical ruler in p...	George Miller	Tom Hardy, Charlize Theron, Nicholas Hoult, Zo...	2015	120	8.1	632842	153.63
8	75	Zootopia	Animation,Adventure,Comedy	In a city of anthropomorphic animals, a rookie...	Byron Howard	Ginnifer Goodwin, Jason Bateman, Idris Elba, J...	2016	108	8.1	296853	341.26
9	77	The Avengers	Action,Sci-Fi	Earth's mightiest heroes must come together an...	Joss Whedon	Robert Downey Jr., Chris Evans, Scarlett Johan...	2012	143	8.1	1045588	623.28
10	78	Inglourious Basterds	Adventure,Drama,War	In Nazi-occupied France during World War II, a...	Quentin Tarantino	Brad Pitt, Diane Kruger, Eli Roth,Mélanie Laurent	2009	153	8.3	959065	120.52
11	81	Inception	Action,Adventure,Sci-Fi	A thief, who steals corporate secrets through ...	Christopher Nolan	Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen...	2010	148	8.8	1583625	292.57

				...		Ellen...					
12	83	The Wolf of Wall Street	Biography,Comedy,Crime	Based on the true story of Jordan Belfort, fro...	Martin Scorsese	Leonardo DiCaprio, Jonah Hill, Margot Robbie,M...	2013	180	8.2	865134	116.87
13	84	Gone Girl	Crime,Drama,Mystery	With his wife's disappearance having become th...	David Fincher	Ben Affleck, Rosamund Pike, Neil Patrick Harri...	2014	149	8.1	636243	167.74
14	93	The Help	Drama	An aspiring author during the civil rights mov...	Tate Taylor	Emma Stone, Viola Davis, Octavia Spencer, Bryc...	2011	146	8.1	342429	169.71
15	100	The Departed	Crime,Drama,Thriller	An undercover cop and a mole in the police att...	Martin Scorsese	Leonardo DiCaprio, Matt Damon, Jack Nicholson,...	2006	151	8.5	937414	132.37
17	115	Harry Potter and the Deathly Hallows: Part 2	Adventure,Drama,Fantasy	Harry, Ron and Hermione search for Voldemort's...	David Yates	Daniel Radcliffe, Emma Watson, Rupert Grint, M...	2011	130	8.1	590595	380.96
18	125	The Dark Knight Rises	Action,Thriller	Eight years after the Joker's reign of anarchy...	Christopher Nolan	Christian Bale, Tom Hardy, Anne Hathaway,Gary ...	2012	164	8.5	1222645	448.13
20	139	Shutter Island	Mystery,Thriller	In 1954, a U.S. marshal investigates the disap...	Martin Scorsese	Leonardo DiCaprio, Emily Mortimer, Mark Ruffal...	2010	138	8.1	855604	127.97
22	145	Django Unchained	Drama,Western	With the help of a German bounty hunter , a fr...	Quentin Tarantino	Jamie Foxx, Christoph Waltz, Leonardo DiCaprio...	2012	165	8.4	1039115	162.80
25	242	Inside Out	Animation,Adventure,Comedy	After young Riley is uprooted from her Midwest...	Pete Docter	Amy Poehler, Bill Hader, Lewis Black, Mindy Ka...	2015	95	8.2	416689	356.45
28	428	The Bourne Ultimatum	Action,Mystery,Thriller	Jason Bourne dodges a ruthless CIA official an...	Paul Greengrass	Matt Damon, Edgar Ramírez, Joan Allen, Julia S...	2007	115	8.1	525700	227.14
31	500	Up	Animation,Adventure,Comedy	Seventy-eight year old Carl Fredricksen	Pete Docter	Edward Asner, Jordan Nagai, John	2009	96	8.3	722203	292.98
34	689	Toy Story 3	Animation,Adventure,Comedy	The toys are mistakenly delivered to a day-car...	Lee Unkrich	Tom Hanks, Tim Allen, Joan Cusack, Ned Beatty	2010	103	8.3	586669	414.98
35	773	How to Train Your Dragon	Animation,Action,Adventure	A hapless young Viking who aspires to hunt dra...	Dean DeBlois	Jay Baruchel, Gerard Butler,Christopher Mintz-...	2010	98	8.1	523893	217.39

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.

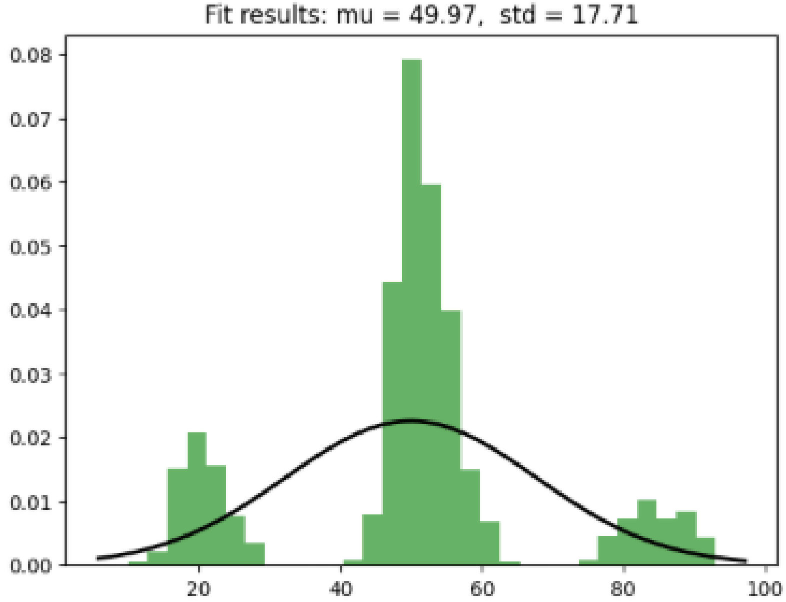
Step 11: End the Program.

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:

```
Columns in the dataset:
Index(['Sno', 'Regno', 'Sattvic', 'Rajasic', 'Tamasic', 'Number', 'Letter',
       'Verbal ', 'Logical', 'Non Verbal', 'analytical', 'Attitude',
       'Apptitude', 'gunas', 'Mentalability', 'height', 'weight', 'BMI',
       'Type', 'Att.Value', 'PhyValue', 'Total', 'Gender', 'Unnamed: 23'],
      dtype='object')
0    47
1    57
2    24
3    25
4    59
Name: Total, dtype: int64
```




```
Shapiro-Wilk Test: Statistics=0.882, p=0.000
Sample does not look Gaussian (reject H0)
```

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:

```
Group Means: [86.6, 84.4, 92.4]
Grand Mean: 87.80
SS Between: 170.80
SS Within: 341.60
DF Between: 2
DF Within: 12
MS Between: 85.40
MS Within: 28.47
F-Statistic: 3.00
```

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 :

```
Press 1 for n, p, c value or 2 for lambda value: 1
Enter the value of n: 4
Enter the value of success p: 4
Enter the value of r: 4
i: 0
P(0) = 1.1253517471925912e-07
i: 1
P(1) = 1.8005627955081459e-06
i: 2
P(2) = 1.4404502364065167e-05
i: 3
P(3) = 7.682401260834756e-05
i: 4
P(4) = 0.00030729605043339025
```

LAMBDA VALUE :

```
Press 1 for n, p, c value or 2 for lambda value: 2
Enter the value of lambda: 5
Enter the value of r: 5
i: 0
P(0) = 0.006737946999085467
i: 1
P(1) = 0.03368973499542734
i: 2
P(2) = 0.08422433748856833
i: 3
P(3) = 0.14037389581428056
i: 4
P(4) = 0.1754673697678507
i: 5
P(5) = 0.1754673697678507
```

Result :

Thus, our program has been successfully saved and executed.

38. 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

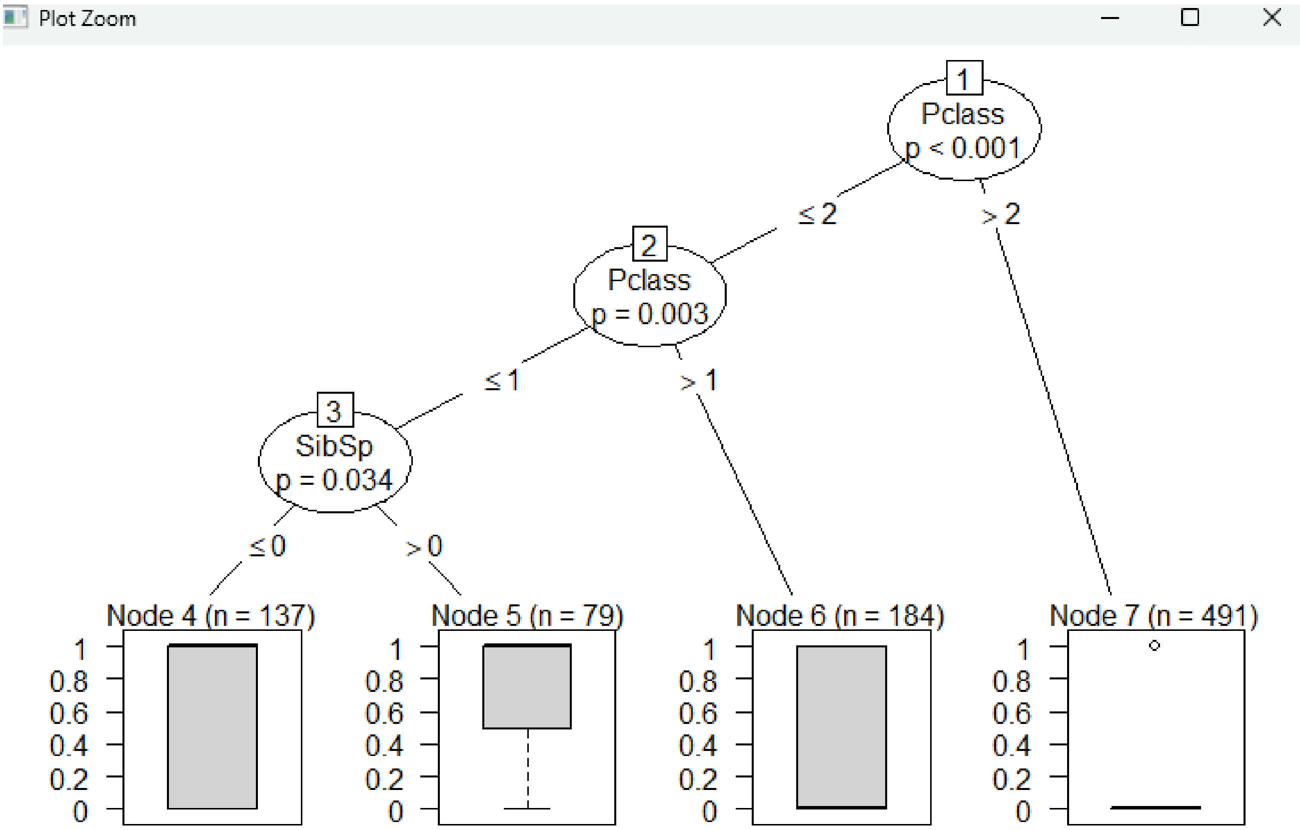
Program:

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:

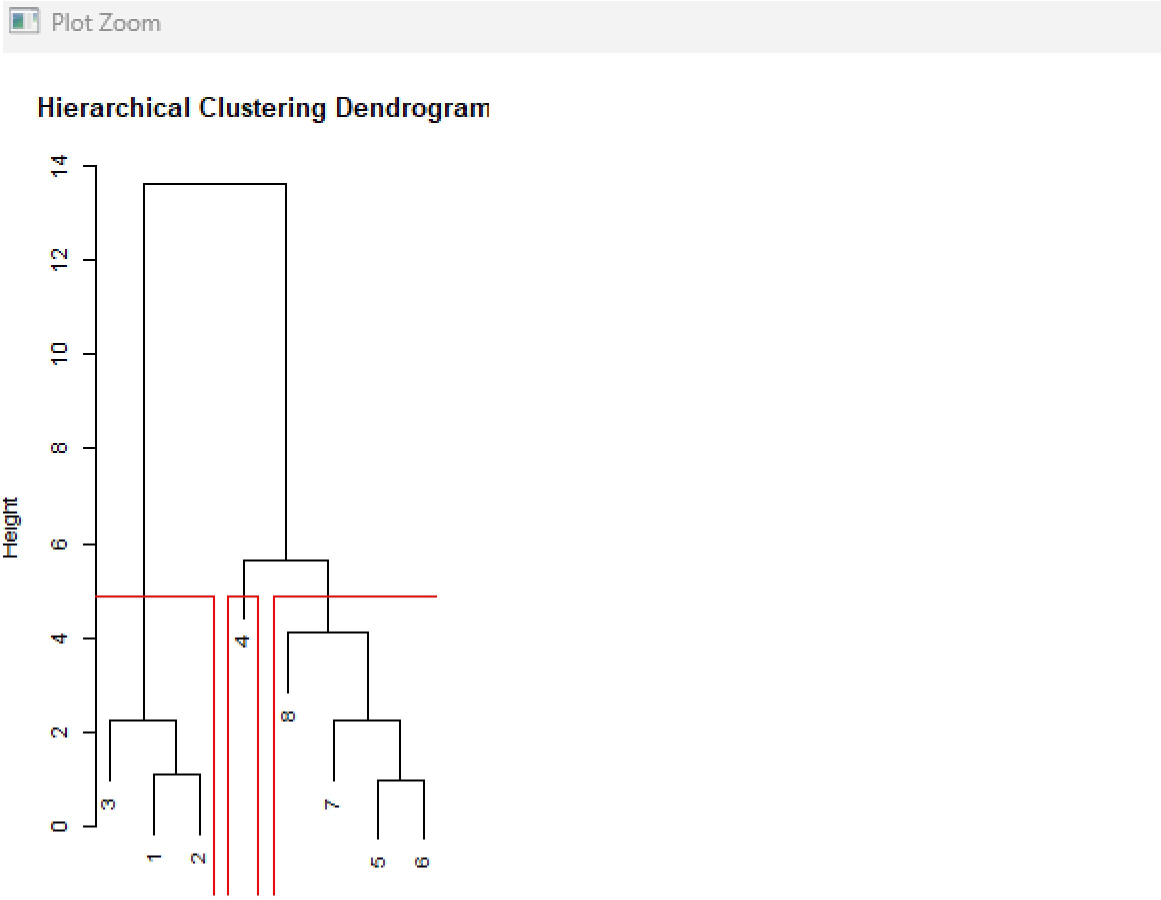
```
Console 2/A

Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.12.0 -- An enhanced Interactive Python.
P(X ≤ 16) = 0.8987089925601621
P(X > 16) = 0.10129100743983793
Manual formula P(X=6): 0.05040940672246225
scipy P(X=6): 0.05040940672246224
Probabilities: 0.049787068367863944 , 0.14936120510359185 , 0.22404180765538775 ,
0.22404180765538775
Sum of first four probabilities: 0.6472318887822313
```

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.

39. 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:

```
Chi-Square Goodness of Fit Result
Chi-Square Statistic: 1.0351054789574108
P-Value: 0.9596848665674865
```

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:

```
Contingency Table:
       ProductA  ProductB
18-25        20        25
26-35        30        30
36-45        25        20

Chi-Square Test Result:
Chi-Square Statistic: 1.1111111111111112
P-Value: 0.5737534207374329
Degrees of Freedom: 2
Expected Frequencies Table:
 [[22.5 22.5]
 [30.  30. ]
 [22.5 22.5]]
```

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:

```
[Workspace loaded from ~/.RData]

> observed <- c(10, 8, 9, 10, 2, 11)
> expected <- rep(1/6, 6)
> print(chisq.test(x = observed, p = expected))

        Chi-squared test for given probabilities

data:  observed
X-squared = 6.4, df = 5, p-value = 0.2692

>
> data <- matrix(c(10, 20, 30,
+                  6,  9,  17),
+                nrow = 2, byrow = TRUE)
> print(chisq.test(data))

        Pearson's Chi-squared test

data:  data
X-squared = 0.27157, df = 2, p-value = 0.873
```

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:

a) Moving average.

b) Auto correlation & auto correlation.

c) ARIMA for forecast.

d) 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.

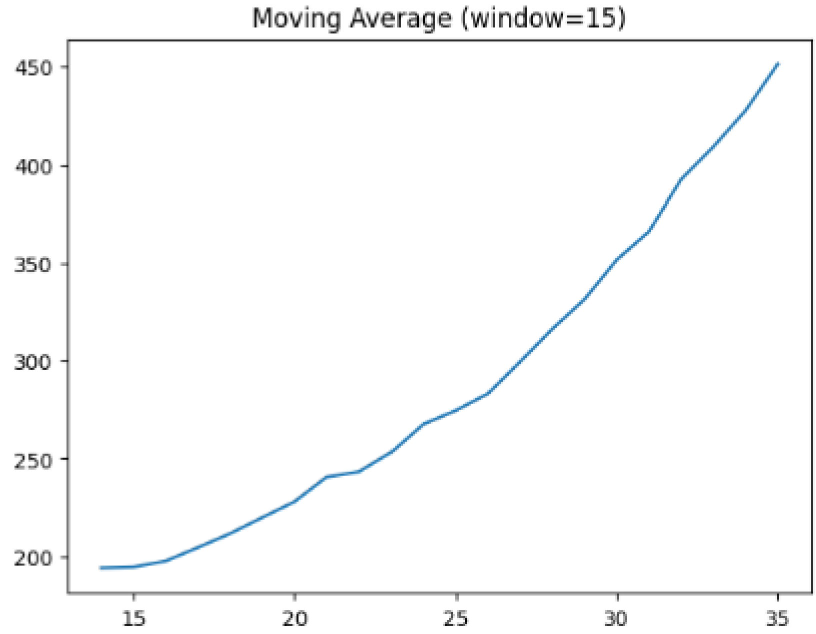
```
import pandas as pd
import numpy as np
```

```
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)
```

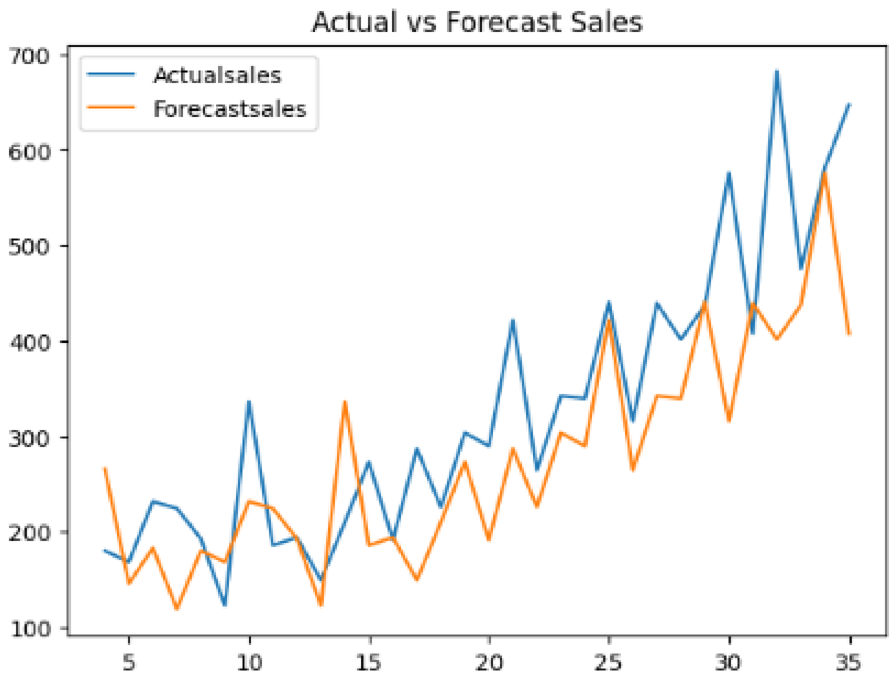
OUTPUT:

a)Moving average

```
min    119.300000
25%    192.450000
50%    280.150000
75%    411.100000
max    682.000000
0             NaN
1             NaN
2             NaN
3             NaN
4             NaN
5             NaN
6             NaN
7             NaN
8             NaN
9             NaN
10            NaN
11            NaN
12            NaN
13            NaN
14     194.093333
15     194.580000
16     197.613333
17     204.540000
18     211.653333
19     219.873333
20     227.966667
21     240.620000
22     243.286667
23     253.253333
24     267.706667
25     274.633333
26     283.300000
27     299.633333
28     316.420000
29     331.573333
30     351.720000
31     366.133333
32     392.466667
33     409.086667
34     427.600000
35     451.400000
Name: Sales, dtype: float64
```

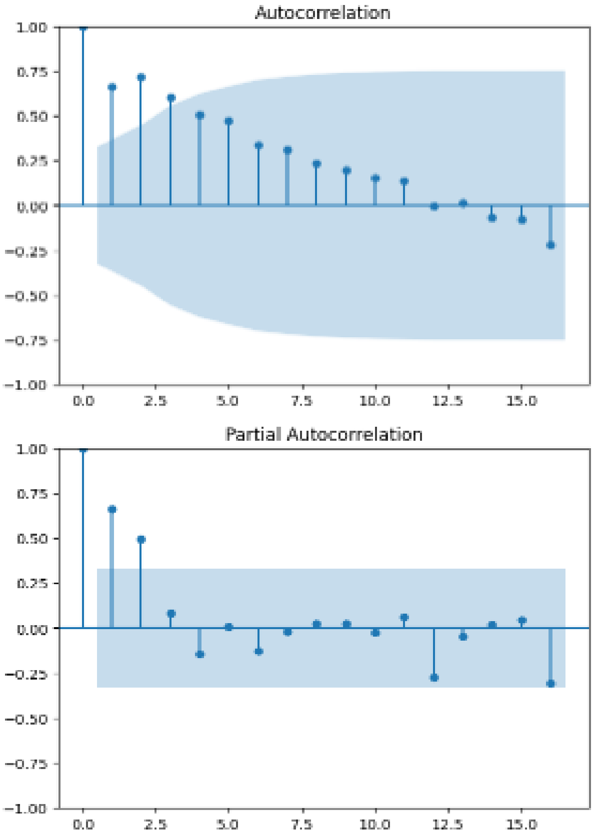


	Actualsales	Forecastsales
4	180.3	266.0
5	168.5	145.9
6	231.8	183.1
7	224.5	119.3
8	192.8	180.3

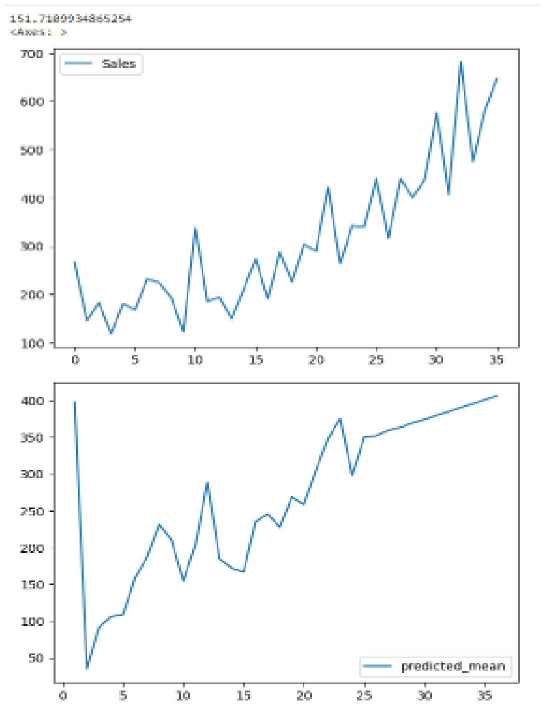



```
10493.880000000001
102.43964076469616
```

b) Auto correlation & auto correlation.



c) ARIMA for forecast



d) ARIMA for least mean square error

```
(0, 0, 0) 244.12895458248818
(0, 0, 1) 207.77606158468043
(0, 0, 2) 164.26094164913746
(0, 1, 0) 142.84038771872738
(0, 1, 1) 111.92072823359469
(0, 1, 2) 74.26671006438406
(0, 2, 0) 267.1262997160708
(0, 2, 1) 143.9480931688411
(0, 2, 2) 80.95727405704706
(1, 0, 0) 160.16927227867984
(1, 0, 1) 118.94890778082826
(1, 0, 2) 82.81202246098819
(1, 1, 0) 94.65254263717607
(1, 1, 1) 95.20661793002864
(1, 1, 2) 97.55119982558419
(1, 2, 0) 142.14461139098654
(1, 2, 1) 91.9479493440583
(1, 2, 2) 70.71925452885783
(2, 0, 0) 106.97429425062762
(2, 0, 1) 102.92545911218878
(2, 0, 2) 104.6214799743644
(2, 1, 0) 91.44069876153556
(2, 1, 1) 95.3215151829625
(2, 1, 2) 89.16197846427362
(2, 2, 0) 103.13198019580632
(2, 2, 1) 82.24839319863278
(2, 2, 2) 85.76325577803566
```

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

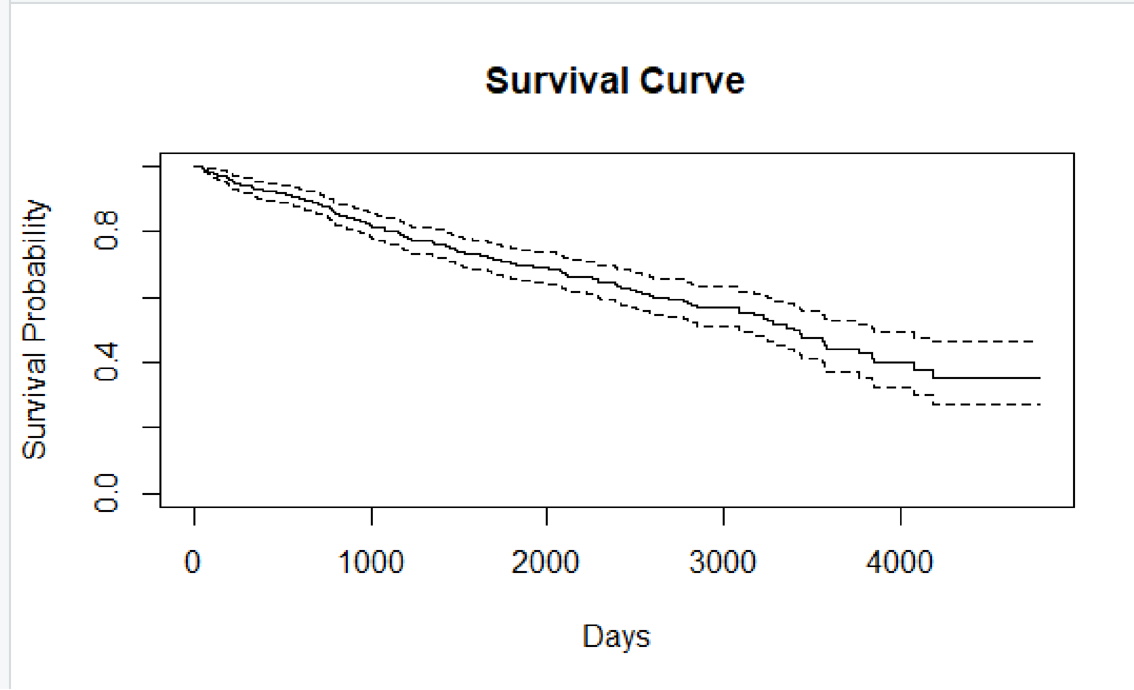
```
!pip install lifelines
from lifelines import KaplanMeierFitter
durations=[5,6,6,2.5,4,4]
event_observation=[1,0,0,1,1,1]
kmf = KaplanMeierFitter()
kmf.fit(durations,event_observation,label='kaplan Meter Esitmate')
kmf.plot(ci_show=True)
```

```
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)
```

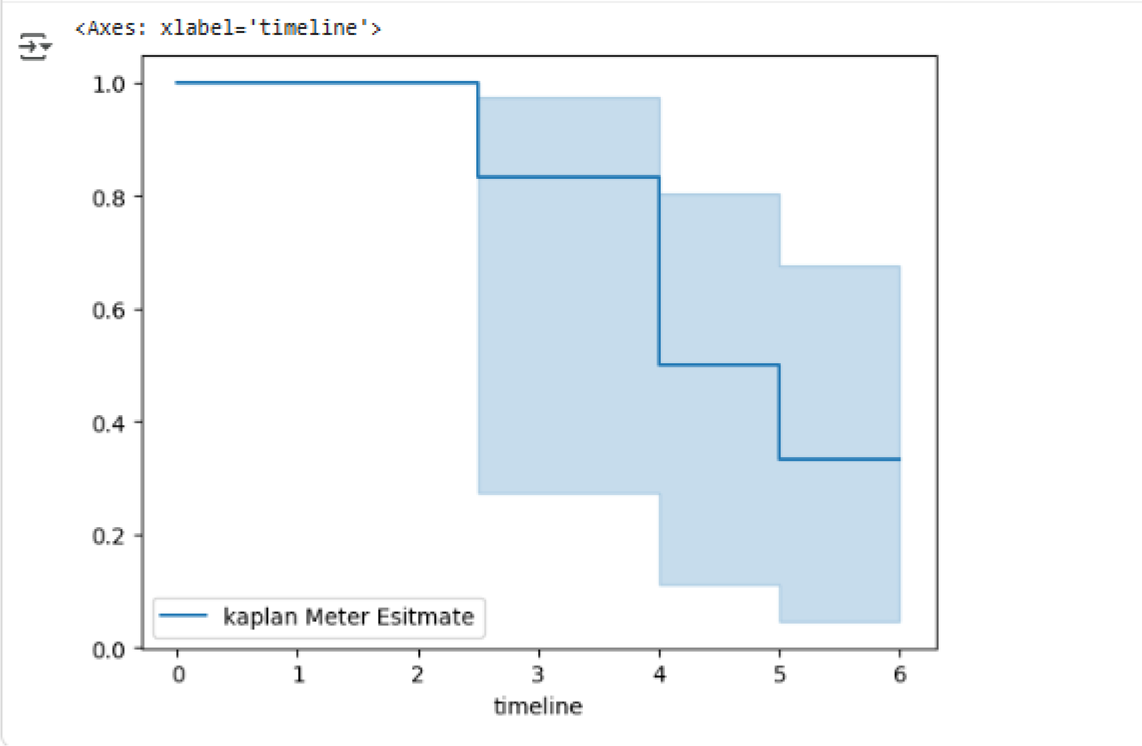
OUTPUT:

a) Vector Data

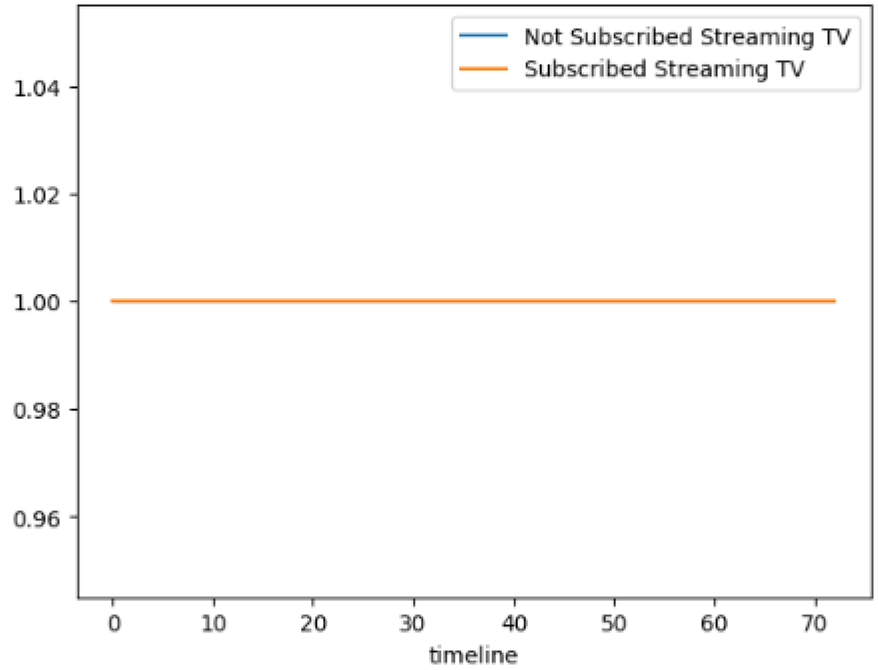
```
> library(survival)
>
> data(pbc)
> head(pbc)
  id time status trt      age sex ascites hepato spiders edema bili chol albumin
1  1  400      2   1 58.76523   f       1      1       1   1.0 14.5  261    2.60
2  2 4500      0   1 56.44627   f       0      1       1   0.0  1.1  302    4.14
3  3 1012      2   1 70.07255   m       0      0       0   0.5  1.4  176    3.48
4  4 1925      2   1 54.74059   f       0      1       1   0.5  1.8  244    2.54
5  5 1504      1   2 38.10541   f       0      1       1   0.0  3.4  279    3.53
6  6 2503      2   2 66.25873   f       0      1       0   0.0  0.8  248    3.98
  copper alk.phos    ast trig platelet protime stage
1    156   1718.0 137.95  172      190    12.2     4
2     54   7394.8 113.52   88      221    10.6     3
3    210    516.0  96.10   55      151    12.0     4
4     64   6121.8  60.63   92      183    10.3     4
5    143    671.0 113.15   72      136    10.9     3
6     50    944.0  93.00   63       NA    11.0     3
>
> fit <- survfit(Surv(pbc$time, pbc$status == 2) ~ 1)
>
> plot(fit, xlab = "Days", ylab = "Survival Probability", main = "Survival Curve")
>
> summary(fit, times = 0)
Call: survfit(formula = Surv(pbc$time, pbc$status == 2) ~ 1)

 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    0    418       0        1       0            1            1
> summary(fit, times = 3000)
Call: survfit(formula = Surv(pbc$time, pbc$status == 2) ~ 1)

 time n.risk n.event survival std.err lower 95% CI upper 95% CI
 3000     76     143    0.569  0.0303        0.512        0.632
>
```



b) Data From CSV



<Axes: xlabel='timeline'>



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.

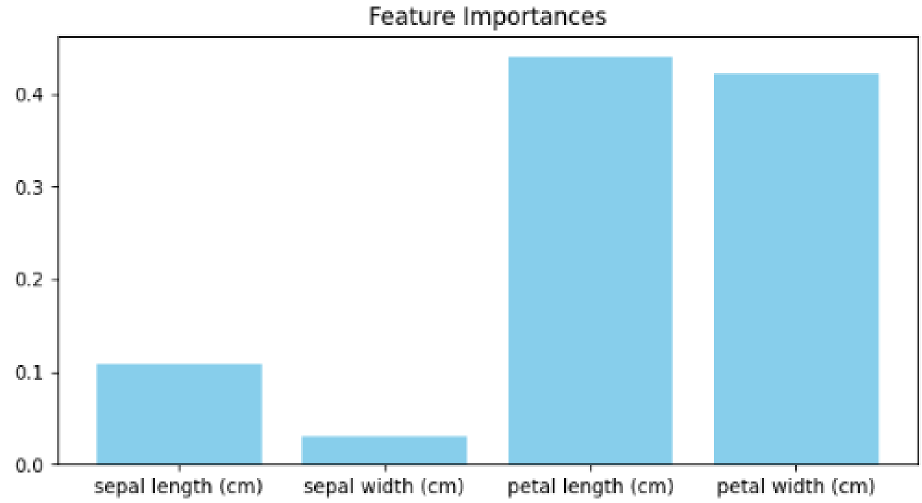
Step 13: Stop the process.

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:

```
Accuracy: 1.00
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```



Result:

Thus, our program has been successfully saved and executed.