SRI RAMAKRISHNA MISSION VIDYALAYA COLLEGE OF ARTS AND SCIENCE

(An Autonomous Institution Affiliated to Bharathiar University, Re-Accredited by NAAC with A grade)

COIMBATORE-641 020

DEPARTMENT OF MASTER OF APPLICATION

RECORD NOTE

Core Practical –I: **Data Science using R**

Course code: **20PCA3C09**

This is certified that this is a bonafide record of work done by

Name:_____ **Reg .No**.:_____

Staff-In-Charge **Head of the Department**

Submitted for the Practical Examination held at Sri Ramakrishna Mission Vidyalaya College of Arts and Science on **05/11/2023** during the year 2023.

Examiners

Internal **External**

INDEX

12	Mean, Median, Mode and Standard Deviation	
13	Input XLS file and find the Mean, Median, and Mode	
14	Correlation without using builtin function	
15	Linear Regression using dataset	
16	Multiple Linear Regression using dataset	
17	Logistic Regression	
18	Poisson Regression	
19	Non-linear Regression	
20	**Charts and Graphs**	
	a) Bar Chart	
	b) Box plot	
	c) Pie chart	
	d) Histogram	
	e) Line Plot	
21	**Distributions**	
	a) Binomial distribution	
	b) Normal distribution	
	c) Continuous distribution	
	d) Exponential distribution	
	e) Chi-squared distribution	
22	Hierarchical Clustering	

23	K- Means clustering	
24	Case study on Guna's theory	
25	Analysis of Variance (ANOVA)	
26	Wilcox on signed-rank test	
27	**Time Series analysis**	
	a) Moving average	
	b) Auto regression	
	c) ARIMA model	
	d) Time series analysis using stock data	
	e) Time series analysis using weather temperature data	
28	Decision trees on any dataset	
29	Random Forest using Titanic dataset	
30	Survival analysis	
31	Mathematical functions using Numpy	
32	Data analysis using Pandas	
33	Visual representation using Matplotlib	
34	Normal distribution to evaluate fitness of data	

1. Fibonacci sequence

Aim:

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be :1,2,3,5,8,13,21,34,55,89,... By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

Algorithm:

Step 1 : Open R studio.

Step 2: Create a new file .

Step 3: Initialize an empty numeric vector fb

Step 4: Set the first two elements of fb to 1 and 2, respectively.

Step 5: Calculate the next Fibonacci number by adding the first and second elements of fb and store it in the variable tmp.

Step 6: Enter a while loop that continues until tmp is less than 4,000,000.

Step 7:After the while loop, use indexing and the modulo operator (%%) to extract even numbers from the fb vector.

Step 8:Calculate the sum of these even numbers using the sum function.

Step 9: Save and Run the program.

Fibonacci sequence

```
fb <- numeric(0)
fb[1] <- 1
fb[2] <- 2
num <- 3
tmp <- fb[1] + fb[2]
while (tmp < 4000000) {
  fb[num] <- tmp
  tmp <- fb[num] + fb[num-1]
  num <- num + 1
}
sum(fb[fb%%2 == 0])
```

Output:

```
4613732
```

2. Largest Prime Factor

Aim:

The prime factors of 13195 are 5,7,13 and 29. What is the largest prime factor of the number 600851475143?

Algorithm:

Step 1: Start the R studio .

Step 2: Initialize Variables:

fb <- numeric(0): Initializes an empty numeric vector fb.

Step 3: Define the Factorization Function (factorize):

curr_max <- 1: Initializes the variable curr_max to 1.

Step 4: while(x %% i == 0) { ... }: Continuously divides x by i until it is no longer divisible.

Step 5 :Return the Maximum Prime Factor: return(curr_max):

Step 6: Save the program and exit.

Largest prime factor

```
factorize <- function(x){
curr_max <- 1
while(x %% 2 == 0){
curr_max <- 2
x <- x / 2
}
for(i in 3:max(3,floor(sqrt(x)))){
while(x %% i == 0){
curr_max <- i
x <- x / i
}
}
if(x > 2){
curr_max <- max(curr_max,x)
}
return(curr_max)
}
num <- 600851475143
print(factorize(num))
```

Output:

```
6857
```

3. Largest palindrome

Aim:

A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009=91×99. Find the largest palindrome made from the product of two 3-digit numbers.

Algorithm:

Step 1 : Open a new file in R studio.

Step 2 : Use ispalindrome function to characterize the program.

Step 3: Palin function to import the numeric values to find the largest palindrome.

Step 4 : Print max value of palin.

Step 5: Save the program and exit the program.

Largest palindrome

```
isPalindrome <- function(n) {
 n <- as.character(n)
 n <- unlist(strsplit(n, ""))
 len <- floor(length(n)/2)
 result <- all(n[1:len] == n[length(n):(length(n)-len+1)])
 return(result)
 }
palin <- numeric(0)
for (i in 100:999) {
palin <- c(palin, (i:999)*i)
}
max(palin[sapply(palin, isPalindrome)])
```

Output:

```
906609
```

4. 10001st Prime number

Aim:

By listing the first six prime numbers: 2,3,5,7,11, and 13, we can see that the 6th prime is 13.What is the 10001st prime number?

Algorithm:

Step 1 : Open a new file in R studio.

Step 2 : use max_num to find 10001st prime number.

Step 3: for(i in 2:floor(max_num/2)){

if(nums[i] != 0){

nums[seq.int(i*2,max_num,i)]

Step 4 : Print the output of primie number.

Step 5 : Run the program and exit.

10001st prime number

```
max_num <- 999999
nums <- 1:max_num
nums[1] <- 0
to_find <- 10001
found<-0
for(i in 2:floor(max_num/2)){
  if(nums[i] != 0){
    nums[seq.int(i*2,max_num,i)] <- 0
    found <- found + 1
    if(found == to_find){
      print(i)
      break
    }
  }
}
```

Output:

```
104743
```

5.Pythagorean triplet

Aim:

A Pythagorean triplet is a set of three natural numbers, a<b<c, for which, **$a^2+b^2=c^2$** For example, **$3^2+4^2=9+16=25=5^2$**.

There exists exactly one Pythagorean triplet for which a+b+c=1000.

Find the product abc.

Algorithm:

Step 1 : Open a new file in R studio.

Step 2 : use max_num to pythagorean triplet.

Step 3: Set three different variables to sqrt(a^2 + b^2)

Step 4 : Print the output of pythagorean triplet.

Step 5 : Run the program and exit.

Pythagorean triplet

```
a <- 0
b <- 0
found <- FALSE
while(!found & a <997) {
a <- a + 1
b <- a
while(!found & b <998) {
b <- b + 1
c <- sqrt(a^2 + b^2)
found <- (1000-a-b == c)
}
}
a*b*c
```

Output:

```
31875000
```

6. Longest Collatz sequence

Aim:

The following iterative sequence is defined for the set of positive integers:

$$n \rightarrow n/2 (n \text{ is even})$$

$$n \rightarrow 3n+1 (n \text{ is odd})$$

Using the rule above and starting with 13, we generate the following sequence:

$$13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1.$$

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.Which starting number, under one million, produces the longest chain?

NOTE: Once the chain starts the terms are allowed to go above one million

Algorithm:

Step 1 : Open a new file in R studio.

Step 2 : use max_num to Collatz Problem.

Step 3: Set the Collatz function.

Step 4 : Print the output of collatz.

Step 5: To collatz_nums <- rep(0,max_to_calc)

Step 6 : Run the program and exit.

Longest Collatz Sequence

```
collatz <- function(n){
  if(n <= max_to_calc & collatz_nums[n]!=0){ # We already calculated
    return(collatz_nums[n])
  }
  seq_length <- collatz(next_collatz(n))+1
  if(n <= max_to_calc){
    collatz_nums[n] <<- seq_length
  }
  return(seq_length)
}

next_collatz <- function(n){
  if((n%%2)==0){
    return(n/2)
  }else{
    return((n*3)+1)
  }
}

max_n <- 999999
max_to_calc <- 3999999
collatz_nums <- rep(0,max_to_calc)
collatz_nums[1] <- 1
max_seq <- 0
```

```
for(i in 2:max_n){
  if(collatz_nums[i]==0){
    collatz_nums[i] <- collatz(i)
    if(collatz_nums[i] > max_seq){
      max_seq <- collatz_nums[i]
      max_start <- i
    }
  }
}
print(max_start)
```

Output:

```
837799
```

7.Factorial digit sum

Aim:

n! means $n\times(n-1)\times\cdots\times3\times2\times1$. For example, $10!=10\times9\times\cdots\times3\times2\times1=3628800$, and the sum of the digits in the number 10! is 3+6+2+8+8+0+0=27. Find the sum of the digits in the number 100!.

Algorithm:

Step 1 : Open a new file in R studio.

Step 2 :Use mult_digit function for factorial digit sum.

Step 3: For fac.tmp

Step 4 : Print the output of the factorial digit.

Step 5 : Run the program and exit.

Factorial Digit Sum

```
mult_digit <- function(mydig1, mydig2) {
fac <- 0
for(i in 1:length(mydig2)) {
fac.tmp <- mydig1 * mydig2[length(mydig2)-i+1]
fac.tmp <- c(fac.tmp, rep(0, i-1))
fac <- c(rep(0,length(fac.tmp)-length(fac)),fac) + fac.tmp
while(!all(fac < 10)) {
carry <- c(fac %/% 10, 0)
if(carry[1]==0) carry <- carry[-1]
fac <- c(rep(0,length(carry)-length(fac)), fac%%10) + carry
}
}
return(fac)
}
mynum <- 1
for(n in 2:100) {
num.vect <- as.integer(unlist(strsplit(as.character(n), "")))
mynum <- mult_digit(mynum, num.vect)
}
sum(mynum)
```

Output:

```
648
```

8. Amicable numbers

Aim:

Let d(n) be defined as the sum of proper divisors of n(numbers less than n which divide evenly into n). If d(a)=b and d(b)=a, where a≠b, then a and b are an amicable pair and each of a and b are called amicable numbers.

For example, the proper divisors of 220 are 1,2,4,5,10,11,20,22,44,55 and 110; therefore d(220)=284. The proper divisors of 284 are 1,2,4,71 and 142; so d(284)=220. Evaluate the sum of all the amicable numbers under 10000.

Algorithm:

Step 1 : Open a new file in R studio.

Step 2 :Use a maximum 10000 amicable number.

Step 3: Use If else statement to sum the two digit numbers

Step 4 : Print the output of the amicable number.

Step 5: Run the program and exit.

Amicable Numbers

```
max <-10000
sumdiv <- function(num) {
if (num==0 | num>max)
return(0)
else {
div <- 1:num
div <- div[(num %% (1:num))==0]
result <- sum(div) - num
if (result != num)
return(result)
else
return(0)
}}
numarr1 <- 1:max
numdiv <- sapply(numarr1, sumdiv)
numarr2 <- sapply(numdiv, sumdiv)
sum(numarr1[numarr1==numarr2])
```

Output:

```
31626
```

9. Pandigital prime

Aim:

We shall say that an n-digit number is pandigital if it makes use of all the digits 1 to n exactly once. For example, 2143 is a 4-digit pandigital and is also prime.What is the largest n-digit pandigital prime that exists?

Algorithm:

Step 1 : Open a new file in R studio.

Step 2 :Use pandigital number 9 function.

Step 3: Sum the duplicated total numbers in pandigital prod

Step 4: Unlist the strsplit list of prime numbers.

Step 5 : Print the output of the factorial digit.

Step 6 : Run the program and exit.

Pandigital prime

```
pandigital.9 <- function(x)
(length(x)==9 & sum(duplicated(x))==0 & sum(x==0)==0)
pandigital.prod <- vector()
i <- 1
for (m in 2:100) {
if (m < 10)
n_start <- 1234
else
n_start <- 123
for (n in n_start:round(10000/ m)) {
digs <- as.numeric(unlist(strsplit(paste0(m, n, m * n), "")))
if (pandigital.9(digs)) {
pandigital.prod[i] <- m * n
i <- i + 1
print(paste(m, "*", n, "=", m * n))
}
}
}
answer <- sum(unique(pandigital.prod))
print(answer)
```

Output:

"4 * 1738 = 6952"

"4 * 1963 = 7852"

"12 * 483 = 5796"

"18 * 297 = 5346"

"27 * 198 = 5346"

"28 * 157 = 4396"

"39 * 186 = 7254"

"42 * 138 = 5796"

"48 * 159 = 7632”

```
sum(unique(pandigital.prod)
```

45228

10. Circular primes

Aim:

The number,197, is called a circular prime because all rotations of the digits: 197,971, and 719, are themselves prime. There are thirteen such primes below 100: 2,3,5,7,11,13,17,31,37,71,73,79, and 97.

How many circular primes are there below one million?

Algorithm:

Step 1 : Open a new file in R studio.

Step 2 : Import a library pracma

Step 3 : Print the one million circular prime numbers

Step 5 : Use containsComposite to prod the numbers.

Step 6: isprime number the numeric values of one million circular prime.

Step 7 : Run the program and exit.

Circular Prime

```
main <- function(){
pmt <- proc.time()
library(pracma)
p <- primes(1e6)
p <- p[sapply(p, containsComposite)]
p <- p[sapply(p,isCircular)]
print(paste("Number of circular primes below one million is",
length(p)+2))#plus 2 and 5
print(proc.time()-pmt)  }
containsComposite <- function(x){
x <- as.numeric(unlist(strsplit(as.character(x),"")))
if( any(x%%2==0) || any(x==5) ){
return(FALSE)}
return(TRUE)}
isCircular <- function(x){
x <- as.numeric(unlist(strsplit(as.character(x),"")))
l <- length(x)
x <- c(x,x)
for(i in 2:(l) ){
if(!isprime(as.numeric(paste(x[i:(i+l-1)],sep = "",collapse =
"")))){return(FALSE)}}
return(TRUE)}
```

Output:

```
55
```

11. Create two data tables having name, age, DoJ, DoR, empcode,salary

Aim:

To Create data files having name, age, DoJ, DoR, emp code,salary in json,xml,xls.Merge all those files as a single file into an xls file.

a) How will you merge these two tables to create a single table?
b) Print those who are receiving salaries greater than 5000.
c) Print those who are receiving salaries between 1000 and 10000.
d) Print those employees whose age is greater than 50.
e) Print those employees who have joined the company in less than one year

Algorithm:

Step 1: Open the R studio

Step 2: Create 2 tables using data.table minimum 10 records in each table containing given columns name, age, doj, dor, empcode and salary.

Step 3: Print both the tables and check the data displayed was correct.

Step 4: Merge the table without a redundant column present.

Step 5: Get the salary greater than 5000 and print from merged table

Step 6: Get the salary in between 1000 and 10000 inbetween = MergedTable[salary > 1000& salary<10000]

Step 7: Get the employees' age greater than 50 and print from merged table.

Step 8: Get the employees who have joined company in less than one year

Step 9: Save the program and run the conditions one by one to get the output.

Create two data tables having name, age, DoJ, DoR, empcode,salary

```
install.packages("jsonlite")
install.packages("xlsx")
install.packages("XML")
```

#A) How will you merge these two tables to create a single table

```
        library(XML)
        library(xlsx)
        library(jsonlite)
        getwd()
# Read data from JSON files
#data <- read_json("employee1.json")
#print(data)
        df1 <- fromJSON("employee1.json")
        df2 <- fromJSON("employee2.json")


# Print the data frame
        xml_fileF<- "employee1.xml"
        xml_fileS<- "employee2.xml"
        docF <- xmlParse(xml_fileF)
        docS <- xmlParse(xml_fileS)

# Read data from XML files
        df3 <- xmlToDataFrame(docF)
        df4 <- xmlToDataFrame(docS)


        library(xlsx)
        df5 <- read.xlsx("employees1.xlsx", sheetName = "Sheet1")
        df6 <- read.xlsx("employees2.xlsx", sheetName = "Sheet1")
        print(df1)
        print(df2)
        print(df3)
        print(df4)
        print(df5)
        print(df6)
```

```
# Concatenate data frames using rbind
    merged_data <- rbind(df1, df2, df3,df4, df5, df6)
    print(merged_data)

# Merge data from all sources into a single data frame
    merged_data <- rbind(data_jsonF, data_jsonS, data_xmlF, data_xmlS,
    data_excelF, data_excelS)
    print(merged_data)

# Save the merged data to a new Excel file
    write.xlsx(merged_data, "merged_employees2.xlsx", sheetName =
"Sheet1", row.names = FALSE)
```

Output:

```
print(df1)
  EmployeeName Age Salary Department        Doj        DoR
1  Json1Swetha  26   4900    Finance 2023-08-01 2025-08-01
2       Mahesh  28   5000         HR 2023-08-01 2025-08-01
3         Hari  25   4800         IT 2023-08-01 2025-08-01
4         Ammu  29   5200      Sales 2023-08-01 2025-08-01
5         Mani  31   5500  Marketing 2023-08-01 2025-08-01
print(df2)
  EmployeeName Age Salary Department        Doj        DoR
1  Json2Swetha  26   4000    Finance 2023-08-01 2025-08-01
2       Mahesh  28   4200         HR 2023-08-01 2025-08-01
3         Hari  25   4800         IT 2023-08-01 2025-08-01
4         Ammu  29   5200      Sales 2023-08-01 2025-08-01
5         Mani  31   5500  Marketing 2023-08-01 2025-08-01
print(df3)
  EmployeeName Age Salary Department        Doj        DoR
1   XML1Swetha  26  49000    Finance 2023-08-01 2025-08-01
2       Mahesh  28  50000         HR 2023-08-01 2025-08-01
3         Hari  25  48000         IT 2023-08-01 2025-08-01
4         Ammu  29  52000      Sales 2023-08-01 2025-08-01
5         Mani  31  55000  Marketing 2023-08-01 2025-08-01
print(df4)
  EmployeeName Age Salary Department        Doj        DoR
1   XML2Swetha  26   4900    Finance 2023-08-01 2025-08-01
2       Mahesh  28   5000         HR 2023-08-01 2025-08-01
3         Hari  25   4800         IT 2023-08-01 2025-08-01
4         Ammu  29   5200      Sales 2023-08-01 2025-08-01
5         Mani  31   5500  Marketing 2023-08-01 2025-08-01

print(merged_data)
EmployeeName Age Salary Department        Doj        DoR
1   Json1Swetha  26   4900    Finance 2023-08-01 2025-08-01
2        Mahesh  28   5000         HR 2023-08-01 2025-08-01
3          Hari  25   4800         IT 2023-08-01 2025-08-01
```

```
4          Ammu  29   5200      Sales 2023-08-01 2025-08-01
5          Mani  31   5500  Marketing 2023-08-01 2025-08-01
6   Json2Swetha  26   4000    Finance 2023-08-01 2025-08-01
7        Mahesh  28   4200         HR 2023-08-01 2025-08-01
8          Hari  25   4800         IT 2023-08-01 2025-08-01
9          Ammu  29   5200      Sales 2023-08-01 2025-08-01
10         Mani  31   5500  Marketing 2023-08-01 2025-08-01
11   XML1Swetha  26  49000    Finance 2023-08-01 2025-08-01
12       Mahesh  28  50000         HR 2023-08-01 2025-08-01
13         Hari  25  48000         IT 2023-08-01 2025-08-01
14         Ammu  29  52000      Sales 2023-08-01 2025-08-01
15         Mani  31  55000  Marketing 2023-08-01 2025-08-01
16   XML2Swetha  26   4900    Finance 2023-08-01 2025-08-01
17       Mahesh  28   5000         HR 2023-08-01 2025-08-01
18         Hari  25   4800         IT 2023-08-01 2025-08-01
19         Ammu  29   5200      Sales 2023-08-01 2025-08-01
20         Mani  31   5500  Marketing 2023-08-01 2025-08-01
```

#B) Print those who are receiving salary greater than 5000

```
high_salary_employees <- merged_data[merged_data$Salary > 5000, ]
print(high_salary_employees)
```

Output:

```
     EmployeeName Age Salary Department        Doj        DoR
4           Ammu  29   5200      Sales 2023-08-01 2025-08-01
5           Mani  31   5500  Marketing 2023-08-01 2025-08-01
9           Ammu  29   5200      Sales 2023-08-01 2025-08-01
10          Mani  31   5500  Marketing 2023-08-01 2025-08-01
12        Mahesh  28  50000         HR 2023-08-01 2025-08-01
14          Ammu  29  52000      Sales 2023-08-01 2025-08-01
15          Mani  31  55000  Marketing 2023-08-01 2025-08-01
19          Ammu  29   5200      Sales 2023-08-01 2025-08-01
20          Mani  31   5500  Marketing 2023-08-01 2025-08-01
```

#C) Print those who are receiving salary in between 1000 and 10000

```
medium_salary_employees <- merged_data[merged_data$Salary >=
1000 & merged_data$Salary <= 10000]
print(medium_salary_employees)
```

#D) Print those employees whose age is greater than 50

```
older_employees <- merged_data[merged_data$Age > 30, ]
print(older_employees)
```

Output:

```
EmployeeName     Age   Salary  Department      Doj        DoR
5          Mani  31   5500  Marketing 2023-08-01 2025-08-01
10         Mani  31   5500  Marketing 2023-08-01 2025-08-01
15         Mani  31  55000  Marketing 2023-08-01 2025-08-01
20         Mani  31   5500  Marketing 2023-08-01 2025-08-01
25         Mani  31  55000  Marketing      19574      21574
30         Mani  31   5500  Marketing      19574      21574
```

#E) Print those employees who have joined the company in less than one year

```
# Convert Date of Joining (Doj) and Date of Resignation (DoR) to Date
objects

    merged_data$Doj <- as.Date(merged_data$Doj, format="%Y-%m-%d")
    merged_data$DoR <- as.Date(merged_data$DoR,
    format="%Y-%m-%d")
    print(merged_data$Doj)

# Calculate the duration of employment in days
    merged_data$employment_duration <-
    as.numeric(difftime(merged_data$DoR, merged_data$Doj, units =
    "days"))

# Filter employees who have worked for less than one year (365 days)
    less_than_one_year_employees <-
    merged_data[merged_data$employment_duration < 365, ]
    print(less_than_one_year_employees)
```

12. Mean, Median, Mode and Standard Deviation

Aim:

To Find the mean, median, mode and Standard Deviation following list of values: 12,7,3,4.2,18,2,54,-21,8,-5

Algorithm

Step 1: Start the R language

Step 2: To Formula for find Mean is

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Step 3: Assign the values to x.

Step4: Use following function to find means mean(x)

Step 5: We apply the median function to compute the median value median(x)

Step 6: To finds mode use names(sort(-table(x)))[1]

Step 7: To find Standard deviation sqrt(var(x))

Step 8: Print the result

Step 9: Stop the program

Mean, Median, Mode and Standard Deviation

```
#Mean
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
mean(x)
result.mean <- mean(x)
print(result.mean)

mean1<-mean(x,trim=0.1)
print(mean1)
x<-c(3,4.2,7,8)
mean(x)

x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
result.mean <-  mean(x,trim = 0.3)
print(result.mean)

#Median

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)

#Mode
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)

charv <- c("o","it","the","it","it")

print(charv)
print(unique(charv))
print(tabulate(match(charv, unique(charv))))
result <- getmode(charv)
print(result)

#Standard Deviation
v <- c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)
standard_deviation <- sd(v)
print(paste("Standard Deviation: ", standard_deviation))
```

Output:

```
Mean :
        5.55
Median :
         7.89876
Mode :
          2
Standard Deviation :  1.33630620956212
```

13. Input XLS file and find the Mean, Median, and Mode

Aim:

To write a program in R from an input XLS file and find the Mean, Median, and Mode

Algorithm

Step 1: Start the R language

Step 2: To Formula for find Mean is

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Step 3: Assign the values to x.

Step4: Use following function to find means mean(x)

Step 5: We apply the median function to compute the median value median(x)

Step 6: To finds mode use names(sort(-table(x)))[1]

Step 7: To find Standard deviation sqrt(var(x))

Step 8: Print the result

Step 9: Stop the program

Input XLS file and find the Mean, Median, and Mode

```
install.packages("xlsx")
library(xlsx)

excel_file <- "input.xlsx"

data <- read.xlsx(excel_file, sheetIndex = 1)

print(data)

column_name <- "salary"

column_mean <- mean(data[[column_name]], na.rm = TRUE)
cat("Mean of", column_name, "is:", column_mean, "\n")

column_median <- median(data[[column_name]], na.rm = TRUE)
cat("Median of", column_name, "is:", column_median, "\n")

column_mode <- as.numeric(names(sort(table(data[[column_name]]),
decreasing = TRUE)[1]))
cat("Mode of", column_name, "is:", column_mode, "\n")
```

Output:

```
Mean of salary is: 656.8813
Median of salary is: 628.05
Mode of salary is: 515.2
```

14. Correlation without using builtin function

Aim:

Write a program of Correlations without using builtin function, for x=1,2,3,4,5 and y=2,4,6,8,10

Algorithm:

Step 1: To open the R-Studio software in the system. And use the Correlation formula.

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}}$$

Step 2: Enter the x and y values, then multiple the sum values sum (x*y)

Step 3: Then enter the sum of n(x*y), a(x), b(y) values, Using the formula and the sqrt values of (x1 & y1), then print the (x1 and y1).

Step 4: Enter the r value r<-(10*n-a*b)/(x1*y1) and print(r)

Step 5: Then finally execute and Save the program.

Correlation without using builtin function

```
x <- c(10, 12, 14)
y <- c(11, 13, 24)
n <- length(x)
sumx <- sum(x)
sumy <- sum(y)
sumxy <- sum(x * y)
sumxx <- sum(x*x)
sumyy <- sum(y*y)
r <- (n * sumxy - sumx * sumy) / sqrt((n * sumxx - sumx * sumx) * (n *
sumyy - sumy * sumy))
print(paste("Mathematical method Correlation coefficient (r):", r))
```

Output:

"Mathematical method Correlation coefficient (r): 0.928571428571429"

15. Linear Regression using dataset

Aim :

Write a program for Linear Regression with winequality-red Dataset.

Algorithm

Step 1: Start the R language

Step 2: Get winequality-red dataset details

Step 3: To Formula for find Linear Regressions

$$a = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$b = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$

Step4: Use following function to Linear Regression lm(dataset)

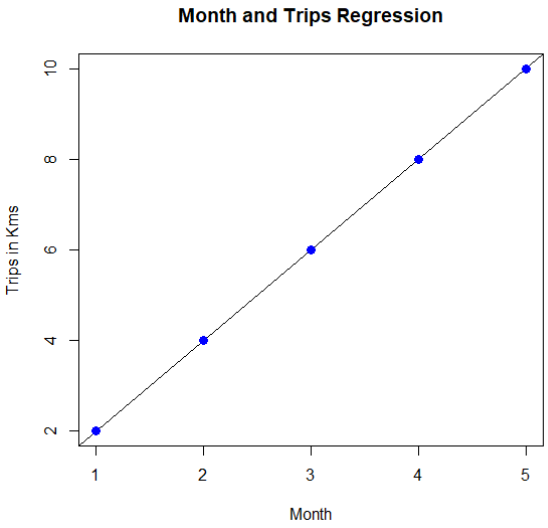
Step 5: To draw plot for Linear Regression

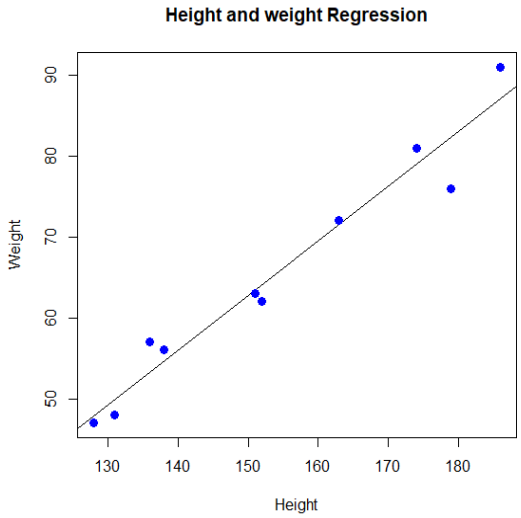
Linear Regression using dataset

```
# Apply the lm() function.
x<-c(1,2,3,4,5)
y<-c(2,4,6,8,10)

#Create Relationship Model & get the Coefficients
relation <- lm(y~x)
print(relation)
print(summary(relation))

plot(x,y,col = "blue",main = "Month and Trips Regression",
    abline(lm(y~x)),cex = 1.9,pch = 20,xlab = "Month",ylab = "Trips in Kms")

a <- data.frame(x =30)
result <-  predict(relation,a)
print(result)

relation<-lm(x~y)
print(relation)
print(summary(relation))
plot(y,x,col = "blue",main = "Month and Trips Regression",
    abline(lm(x~y)),cex = 1.9,pch = 20,xlab = "Month",ylab = "Trips in Kms")

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)
print(relation)
plot(x,y,col = "blue",main = "Height and weight Regression",
    abline(lm(y~x)),cex = 1.9,pch = 20,xlab = "Height",ylab = "Weight")

a <- data.frame(x = 170)
result <-  predict(relation,a)
print(result)
```

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)

# Plot the chart.
plot(x,y,col = "blue",main = "Height & Weight Regression",
    abline(lm(y~x)),cex = 1.9,pch = 20,xlab = "Height in Kg",ylab = "Weight in
cm")

#read data from csv files
setwd("e:/MCA/Unit iv")

data=read.csv("winequality-red.csv")
x<-data$fixed.acidity
y<-data$residual.sugar
relation <- lm(y~x)
print(relation)
plot(x,y,col = "blue",main = "Acidity & Quality",
    abline(lm(y~x)),cex = 1.9,pch = 20,xlab = "Acidity",ylab = "Quality")
```

Output:


Month and Trips Regression
Trips in Kms
Month
2
4
6
8
10
1
2
3
4
5


Height and weight Regression
Weight
Height
50
60
70
80
90
130
140
150
160
170
180

16.Multiple Linear Regression using dataset

Aim :

Write a program for Multiple Linear Regression with mtcars Dataset.

Algorithm:

Step 1: Start the R language

Step 2: Get mtcars dataset details

Step 3: To Formula for find Multiple Linear Regression

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots. + \beta_n X_n + \varepsilon$$

Step4: Use following function to Multiple Linear Regression lm(dataset)

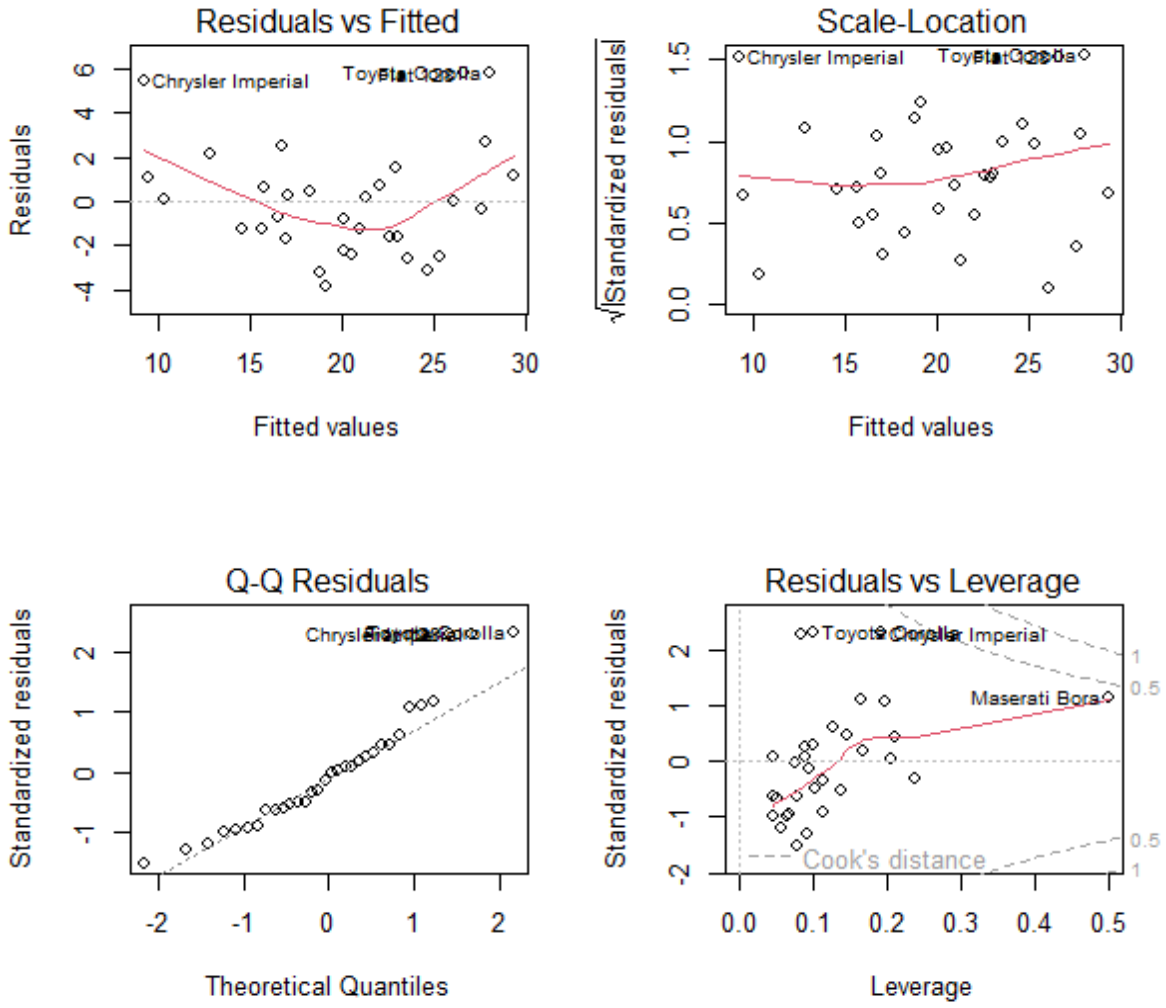
Step 5: To print the summary of Multiple Linear Regression.

Multiple Linear Regression using dataset

```
input <- mtcars[,c("mpg","disp","hp","wt")]
print(head(input))
model <- lm(mpg~disp+hp+wt, data = input)
print(model)
cat("# # # # The Coefficient Values # # # ","\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)
x1 = 221
x2 = 102
x3 = 2.91
Y = 37.15+(-0.000937)*x1+(-0.0311)*x2+(-3.8008)*x3
layout(matrix(c(1,2,3,4),2,2)) # optional 4 graphs/page
plot(model)


data=read.csv("winequality-red.csv")
x1<-data$fixed.acidity
x2<-data$volatile.acidity
x3<-data$citric.acid
x4<-data$residual.sugar
x5<-data$free.sulfur.dioxide
x6<-data$total.sulfur.dioxide
x7<-data$density
x8<-data$pH
x9<-data$sulphates
x10<-data$chlorides
x11=data$alcohol
y<-data$quality
```

```
model=lm(y~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10+x11)
print(summary(model))
paste("y=",a,"+",x1,"*x1","+",x2,"*x2","+",x3,"*x3", "+",x4,"*x4")
#6.3    0.51    0.13    2.3 0.076   29  40  0.99574 3.42    0.75    11

a1 <- data.frame (x1=6.3, x2=0.51, x3=0.13, x4=2.3, x10=0.076, x5=29,
x6=40, x7=0.99574, x8=3.42, x9=0.75, x11=11)
result <-  predict(model,a1)
```

Output:


Residuals vs Fitted
Residuals
Fitted values
Chrysler Imperial
Scale-Location
√|Standardized residuals|
Fitted values
Chrysler Imperial
Q-Q Residuals
Standardized residuals
Theoretical Quantiles
Residuals vs Leverage
Standardized residuals
Leverage
Chrysler Imperial
Maserati Bora
Cook's distance

17. Logistic Regression

Aim :

Write a program for logistic Regression with mtcars Dataset.

Algorithm

Step 1: Start the R language

Step 2: Get mtcars dataset details

Step 3: To Formula for find logistic Regression

$$P = \frac{e^{a+bX}}{1+e^{a+bX}}$$

Step4: Use following function to logistic Regression
lm(dataset),coef(model)

Step 5: To draw plot for MultipleLinear Regression

Logistic Regression

```
mtcars
# Select some columns form mtcars.
input <- mtcars[,c("am","cyl","hp","wt")]

print(head(input))
input <- mtcars[,c("am","cyl","hp","wt")]

am.data = glm(formula = am ~ cyl + hp + wt, data = input, family =
binomial)

print(summary(am.data))
```

Output:

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 19.70288    8.11637   2.428   0.0152 *
cyl          0.48760    1.07162   0.455   0.6491
hp           0.03259    0.01886   1.728   0.0840 .
wt          -9.14947    4.15332  -2.203   0.0276 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’
0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be
1)

    Null deviance: 43.2297  on 31  degrees of freedom
Residual deviance:  9.8415  on 28  degrees of freedom
AIC: 17.841

Number of Fisher Scoring iterations: 8
```

18.Poisson Regression

Aim:

To implement the concept of Poisson regression using R Language.

Algorithm:

Step1:Start the R studio

Step2: Poisson Regression involves regression models in which the response variable is in the form of counts and not fractional numbers.

Step3:The general mathematical equation for Poisson regression is

log(y) = a + b1x1 + b2x2 + bnxn.....

Step 4:Create a Poisson regression model using the glm() function.

Step 5:Create a Regression model

Step 6: The wool "type" and "tension" are taken as predictor variables in a Poisson regressions.

Step 7: Print the Result

Step 8:Stop the process.

Poisson regression

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

output <-glm(formula = breaks ~ wool+tension, data = warpbreaks,
        family = poisson)
print(summary(output))
output1 <-glm(formula = breaks ~ wool+tension, data = warpbreaks,
        family = quasipoisson(link="log"))
print(summary(output1))

mtcars
input <- mtcars[,c("am","cyl","hp","wt")]

print(head(input))
input <- mtcars[,c("am","cyl","hp","wt")]
am.data = glm(formula = am ~ cyl + hp + wt, data = input, family = poisson)
print(summary(am.data))
```

Output:

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.529843   1.081631   2.339   0.0193 *
cyl         -0.205711   0.467090  -0.440   0.6596
hp           0.011133   0.007435   1.497   0.1343
wt          -1.365985   0.674785  -2.024   0.0429 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 23.420  on 31  degrees of freedom
Residual deviance: 10.288  on 28  degrees of freedom
AIC: 44.288

Number of Fisher Scoring iterations: 5
```

19. Non-Linear regression

Aim :

Write a program for NonLinear Regression with winequality-red Dataset.

Algorithm

Step 1: Start the R language

Step 2: Get some dataset details for the program.

Step 3: To Formula for find Non Linear Regressions

$$a = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$b = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$

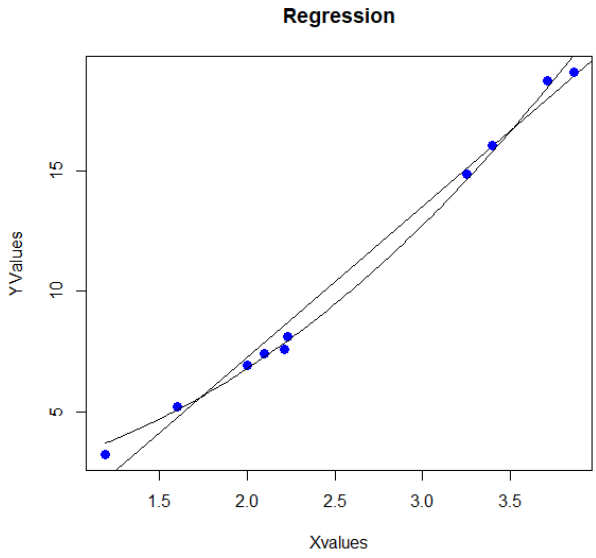
Step4: Use following function to Non Linear Regression lm(dataset)

Step 5: To draw plot for NonLinear Regression

Non-linear Regression

```
xvalues <- c(1.6,2.1,2,2.23,3.71,3.25,3.4,3.86,1.19,2.21)
yvalues <- c(5.19,7.43,6.94,8.11,18.75,14.88,16.06,19.12,3.21,7.58)
plot(xvalues,yvalues)

#linear Regression
plot(xvalues,yvalues,col = "blue",main = "Regression",
    abline (lm (yvalues~xvalues)), cex = 1.9,pch = 20,xlab = "Xvalues",ylab
= "YValues")

model <- nls(yvalues ~ b1*xvalues^2+b2,start = list(b1 = 10,b2 = 10))
print(summary(model))

new.data <- data.frame(xvalues = seq(min(xvalues),max(xvalues), len =
100))
print(new.data)
lines(new.data$xvalues,predict(model,newdata = new.data))
print(sum(resid(model)^2))

print(confint(model))
```

Output:



20.Charts and Graphs

Aim:

To create a Pie Chart, Bar Chart, Box Plot, Line Plot and Histogram in R Language.

Algorithm:

Step1:Start the R studio

Step2:Create Data for the Charts in Pie Chart, Bar Chart and Box Plot

Step3:Plot the Charts for Pie Chart, Bar Chart, Box Plot.

Step4: Create the Data for Graphs in Histogram and Lion Plot

Step5: Create a graph Histogram using the method hist and create a graph for Line Plot using the method plot .

Step6:Save the R File.

Step7: Print the Various charts and Graphs.

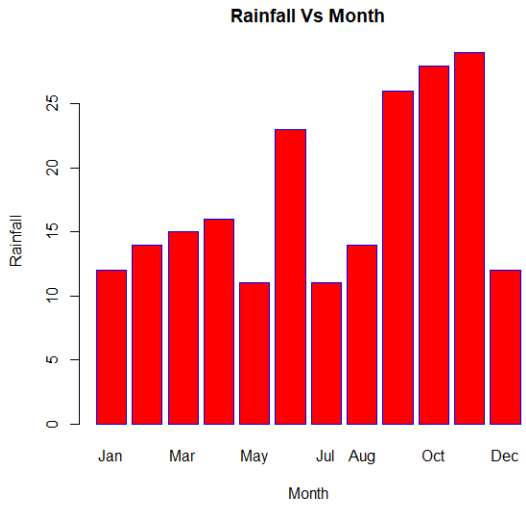
Step8:Stop the process.

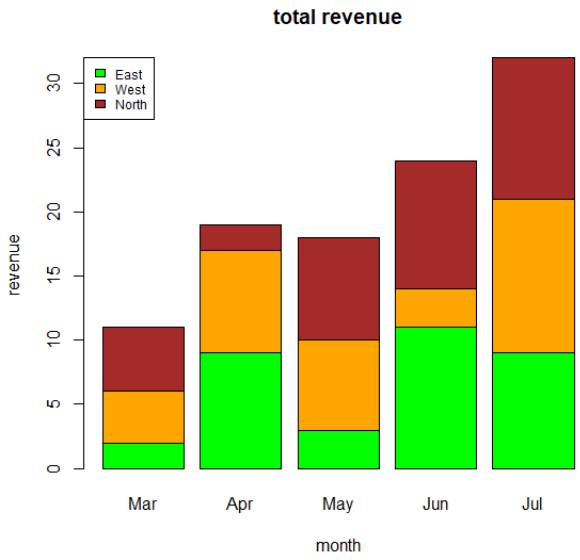
a) Bar Chart

```
H <- c(7,12,28,3,41)
barplot(H)
H <-  c(9,13,21,8,36,22,12,41,31,33,19)

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

H<-c(12,14,15,16,11,23,11,14,26,28,29,12)
M<-c("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov
","Dec")
barplot(H,names.arg = M)
barplot(H,names.arg=M,xlab="Month",ylab="Rainfall",col="red",main="Ra
infall Vs Month",border="blue")

dev.off()

colors = c("green","orange","brown")
months <- c("Mar","Apr","May","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)
print(Values)
barplot(Values, main = "total revenue", names.arg = months, xlab =
"month", ylab = "revenue", col = colors)
legend("topleft", regions, cex = 0.75, fill = colors)


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

```
df1=df["SALARY"]
print(df1)
#barplot(df1)
v<-df1[,]
print(v)
dfname1<-df["Name"]
names<-dfname1[,]
print(names)
barplot(v,names.arg=names,xlab="salary",ylab="in Rs",main="Salary
Chart",col="red",border="blue")
```

Output:

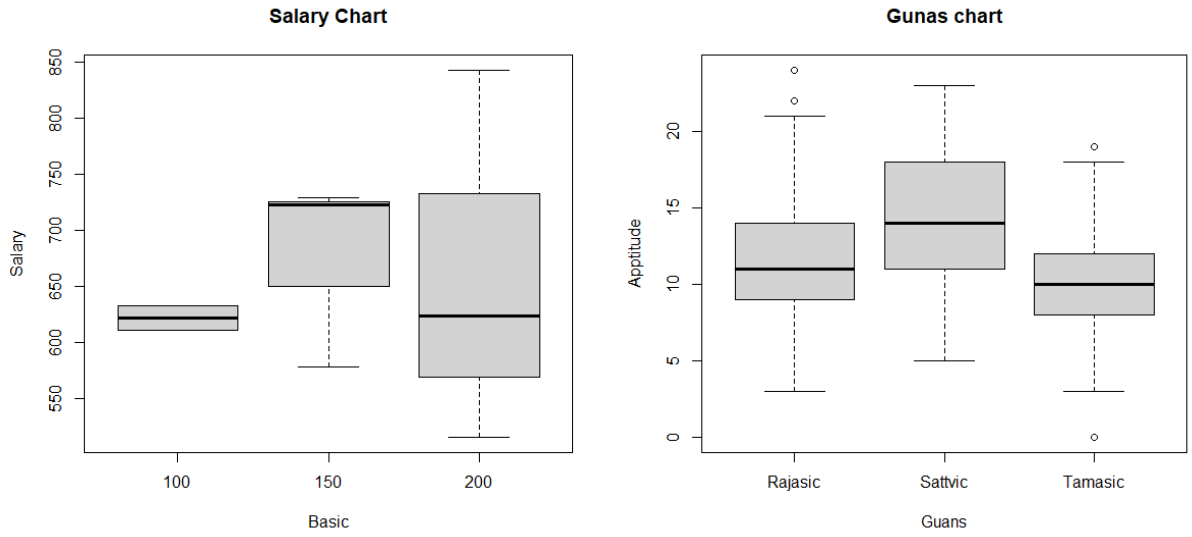


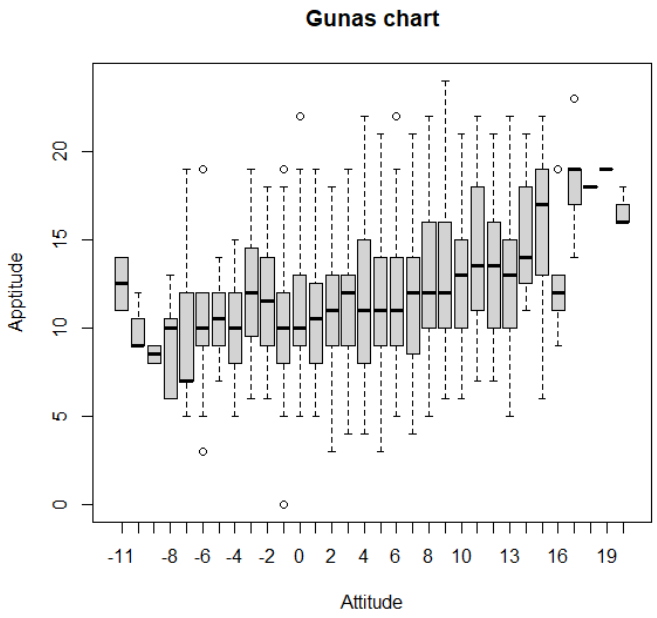


b) Box Plot

```
df<-read.csv("Combined.csv")
print(df)
v<-df[,c("SALARY","Basic")]
print(v)
boxplot(SALARY~Basic,data=v,xlab="Basic",ylab="Salary", main="Salary
Chart")
df<-read.csv("guna1.csv")
print(df)
v<-df[,c("Apptitude","gunas")]
print(v)
boxplot(Apptitude~gunas,data=v,xlab="Guans",ylab="Apptitude",
main="Gunas chart")
v<-df[,c("Apptitude","Attitude")]
print(v)
boxplot(Apptitude~Attitude,data=v,xlab="Attitude",ylab="Apptitude",
main="Gunas chart")
```

Output:



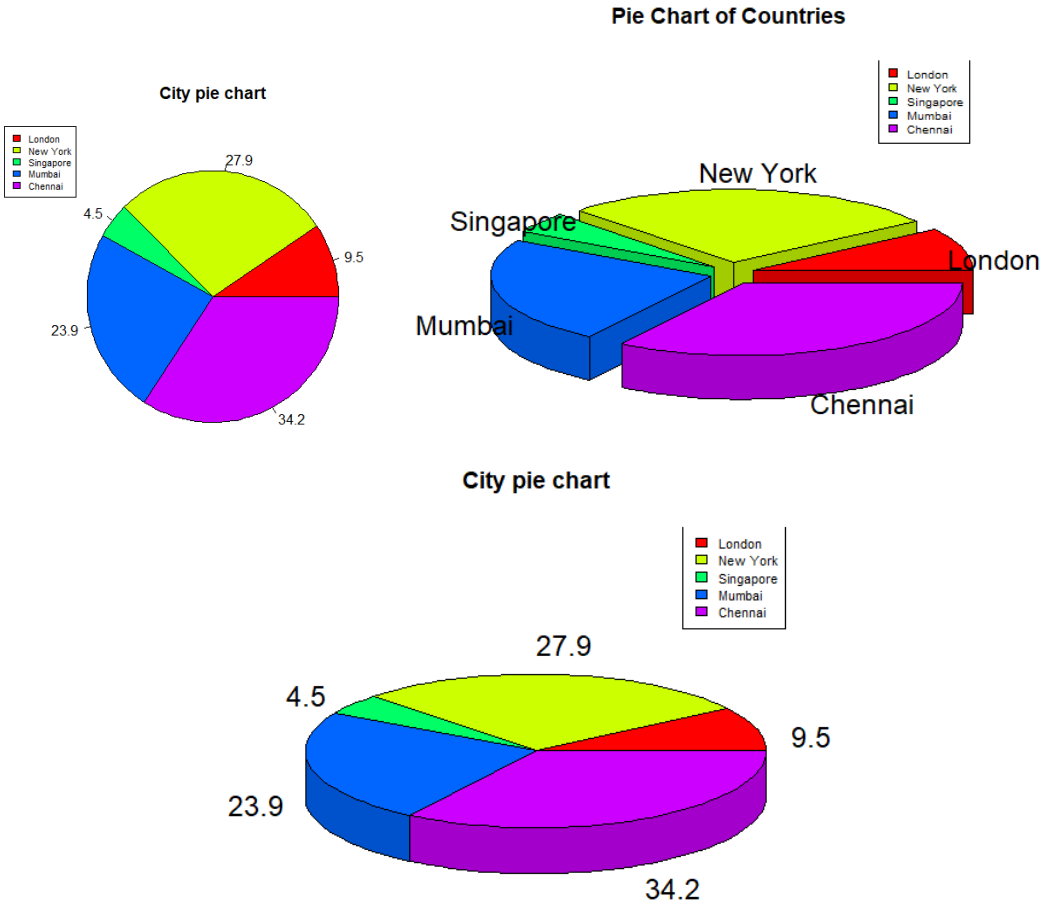


c) Pie Chart

```
x <- c(21, 62, 10, 53,76)
21/222*360

34.05+100.54+16.21+85.95+123.24
labels <- c("London", "New York", "Singapore", "Mumbai","chennai")
pie(x,labels)
pie(x, labels, main = "City pie chart", col = rainbow(length(x)))

x <- c(21, 62, 10, 53,76)
labels <-  c("London","New York","Singapore","Mumbai","Chennai")
piepercent<- round(100*x/sum(x), 1)

pie(x, labels = piepercent, main = "City pie chart",col = rainbow(length(x)))
legend("topleft", c("London","New
York","Singapore","Mumbai","Chennai"), cex = 0.7,
      fill = rainbow(length(x)))

legend("bottomright",labels,cex=1.2,fill=rainbow(length(x)))
```

```
library(plotrix)

x <-  c(21, 62, 10,53,76)
lbl <-  c("London","New York","Singapore","Mumbai","Chennai")

pie3D(x,labels = lbl,explode = 0.1, main = "Pie Chart of Countries ")
pie3D(x,labels=lbl,explode = 0.1, main = "Pie Chart of Countries ",)
legend("topright", c("London","New
York","Singapore","Mumbai","Chennai"), cex =0.6,
      fill = rainbow(length(x)))
dev.off()

x <- c(21, 62, 10, 53,76)
labels <-  c("London","New York","Singapore","Mumbai","Chennai")

piepercent<- round(100*x/sum(x), 1)
pie3D(x, labels = piepercent, main = "City pie chart",col =
rainbow(length(x)))
legend("topright", c("London","New
York","Singapore","Mumbai","Chennai"), cex = 0.7,
      fill = rainbow(length(x)))
```

Output:


City pie chart
London
New York
Singapore
Mumbai
Chennai
27.9
4.5
9.5
23.9
34.2
Pie Chart of Countries
New York
Singapore
London
Mumbai
Chennai
City pie chart
27.9
4.5
9.5
23.9
34.2

d) Histogram

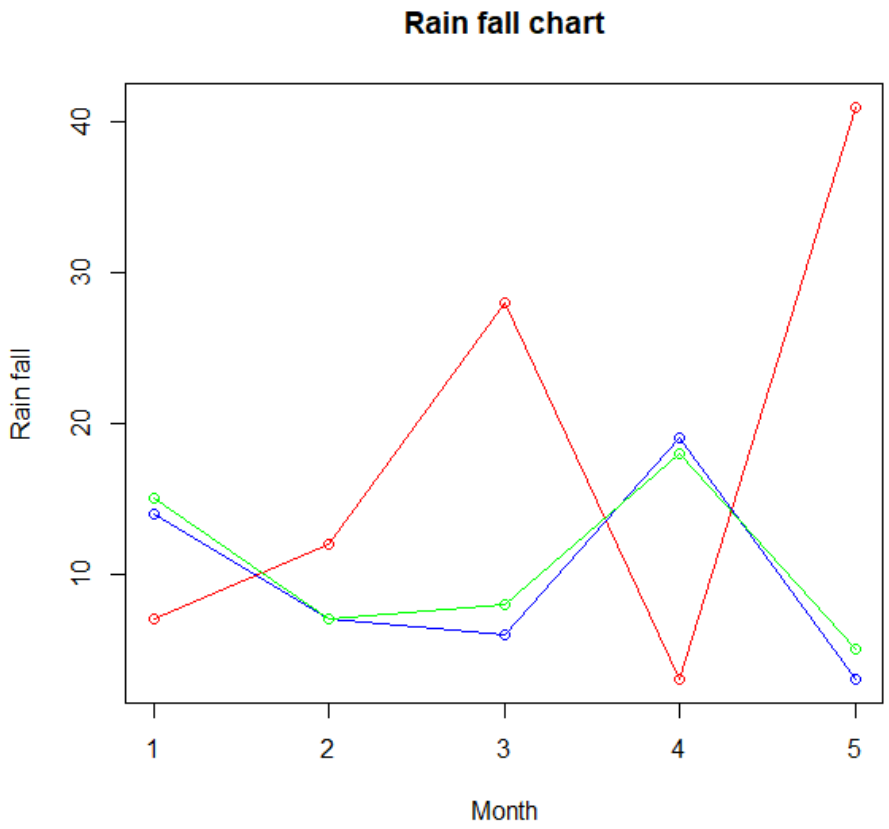
```
df <- read.csv("Combined.csv")
# Create a histogram for the "SALARY" column
hist(df$SALARY,
    main = "Salary Distribution",
    xlab = "Salary",
    ylab = "Frequency",
    col = "skyblue",
    border = "black",
    xlim = c(min(df$SALARY), max(df$SALARY)),
    breaks = 20,  # You can adjust the number of bins
    probability = TRUE)

# Optionally, you can add labels and a legend if needed
legend("topright", legend = "Salary", fill = "skyblue")
```

Output:

e) Line Plot

```
v <- c(7,12,28,3,41)
plot(v,type = "o")
v <- c(7,12,28,3,41)
plot(v,type = "o", col = "red", xlab = "Month", ylab = "Rain fall",
    main = "Rain fall chart")
v <- c(7,12,28,3,41)
t <- c(14,7,6,19,3)
x<-c(15,7,8,18,5)
plot(v,type = "o",col = "red", xlab = "Month", ylab = "Rain fall",
    main = "Rain fall chart")
lines(t, type = "o", col = "blue")
lines(x, type = "o", col = "green")
```

Output:



21. Distributions

Aim:

Write a R Program For Finding Distributions

a)Binomial Distribution

b)Normal Distribution

c)Continuous Distribution

d)Exponential Distribution

e)Chi-Squared Distribution

Algorithm:

Step1: Start the Program

Step2: Create Vector x with Sequence of number from 0-50

Step3: Find the Binomial Distributions using dbinom(),qbinom(),rbinom(),pbinom() functions

Step4: Find the Poisson Distributions using glm() function

Step5: Find the Continuous Distributions using runif() and dunif() function

Step6: Find the Exponential Distribution using dexp(),qexp(),rexp() and pexp() Function

Step7: Find the Normal Distribution using dnorm(),pnorm(),qnorm(),rnorm()

Step8: Find the Chi-Squared Distribution using Dchisq(),pchisq(),qchisq(),rchisq()

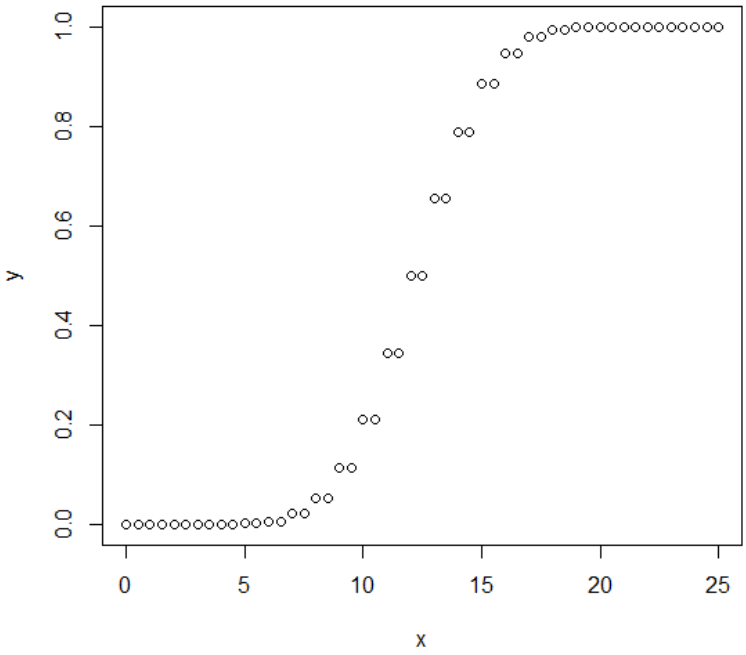
Step9: Stop the Program

a) Binomial Distribution

```
x<-4
y<-pbinom(x,4,0.5)
print(y)

x<-seq(0,4,0.1)
x<-4
y<-pbinom(x,4,0.1)
print(y)
plot(x,y)

x<-26
y<-dbinom(x,51,0.5)
print(y)

x<-seq(26,51,0.5)
y <- pbinom(x,51,0.5)

print(x)
plot(x,y)

y<-1-pbinom(x,51,0.5)
x<-seq(0,25,0.5)
print(y)
plot(x,y)

x<-seq(0,25,0.5)
y<-pbinom(x,25,0.5)
plot(x,y)

x <- seq(0,50,1)
y <- dbinom(x,50,0.5)
print(dbinom(0, size=12, prob=0.2) +
  + dbinom(1, size=12, prob=0.2) +
```

```
  + dbinom(2, size=12, prob=0.2) +
  + dbinom(3, size=12, prob=0.2) +
  + dbinom(4, size=12, prob=0.2))
print(pbinom(4,12,0.2))
```

Output:

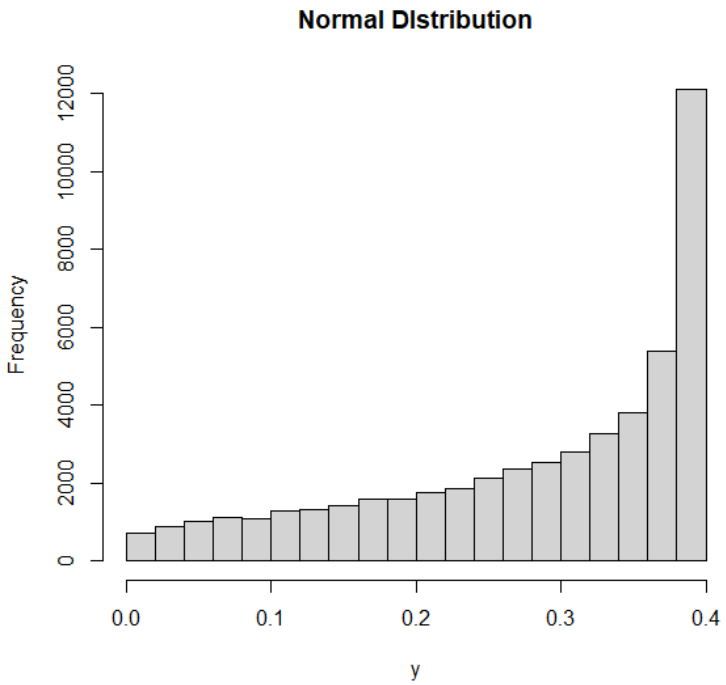


b)Normal distribution

```
x <- seq(-10, 10, by = .1)
y <- dnorm(x, mean = 2.5, sd = 0.5)
plot(x,y)

y1<-pnorm(x,mean=2.5,sd=0.5)
plot(x,y1)
y2<-qnorm(x,mean=2.5,sd=0.5)
plot(x,y2)
y3<-qnorm(x,mean(x),sd(x))
plot(x,y3)

setwd("E:/MCA/unit III")
guna<-read.csv("guna1.csv")
head(guna)
summary(guna)
x<-guna$Attitude
print(x)
print(mean(guna$Attitude))
print(sd(guna$Attitude))

y<-dnorm(guna$Attitude,mean=mean(guna$Attitude),
sd=sd(guna$Attitude))
plot(x,y)
y <- pnorm (guna$Attitude, mean=mean(guna$Attitude),
sd=sd(guna$Attitude))
plot(x,y)
y <- qnorm (guna$Attitude, mean=mean(guna$Attitude),
sd=sd(guna$Attitude))
plot(x,y)
x <- seq(-10,10,by = .2)
y <- pnorm(x, mean = 2.5, sd = 2)

plot(x,y)
```

```
x <- seq(0, 1, by = 0.02)
y <- qnorm(x, mean = 2, sd = 1)
y<-qnorm(x,mean(x),sd(x))
plot(x,y)
y<-pnorm(x,mean=2,sd=1)
plot(x,y)
y<-dnorm(x,mean=2,sd=1)
y<-dnorm(x,mean(x),sd(x))
plot(x,y)
y<-pnorm(x,mean(x),sd(x))
plot(x,y)
y<-qnorm(x,mean(x),sd(x))
plot(x,y)
x<-rnorm(50000)
y<-dnorm(x,mean(x),sd(x))
plot(x,y)
hist(y, main = "Normal DIstribution")
```

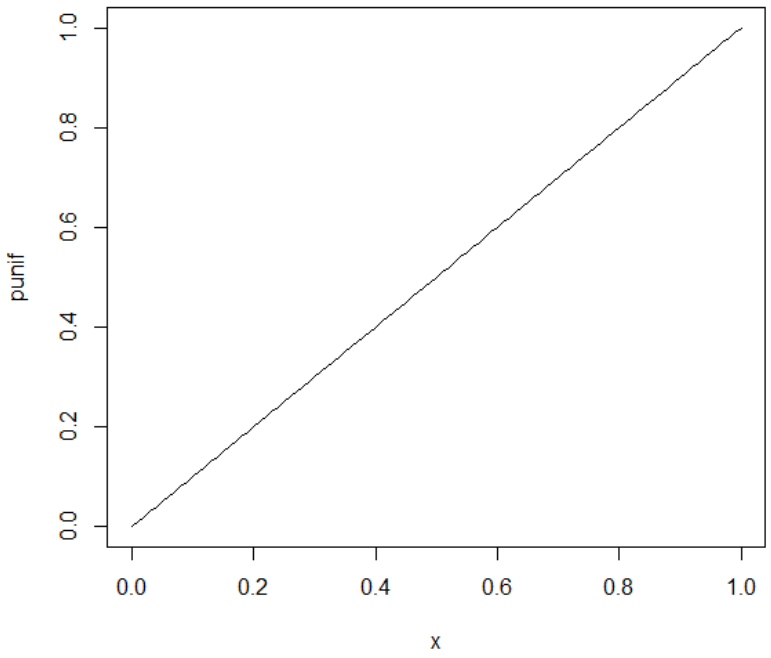
Output:



c)Continuous distribution

```
#runif()
runif(15, min=1, max=3)

#qunif()
min <- 0
max <- 40
qunif(0.2, min = min, max = max)

#dunif()
x <- 5:10
dunif(x, min = 1, max = 20)

#punif()
min <- 0
max <- 60
punif (15 , min =min , max = max)
plot(punif)
```

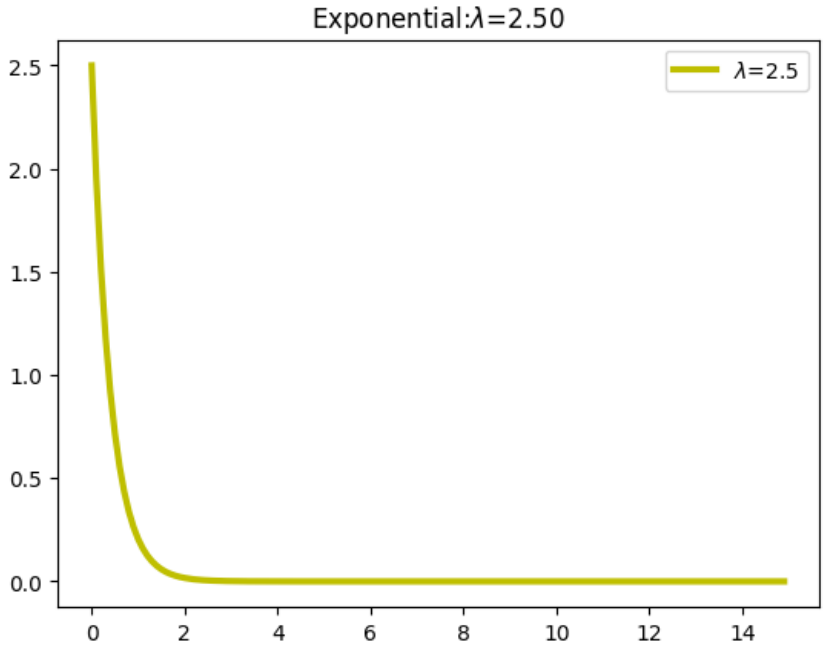
Output:



d)Exponential distribution

```
import numpy as np
from scipy.stats import beta
from matplotlib import pyplot as plt
import math
lambda1=2.5

x=np.arange(0,15,0.1)
y=lambda1 * np.exp(-lambda1*x)
plt.plot(x,y,linewidth=3,color='y',label=r'$\lambda$=%.1f' % (lambda1))
plt.title('Exponential:$\lambda$=%.2f'%lambda1)
#plt.xlabel('x')
#plt.ylabel('Probability density')

plt.legend(loc=0)
plt.show()
```

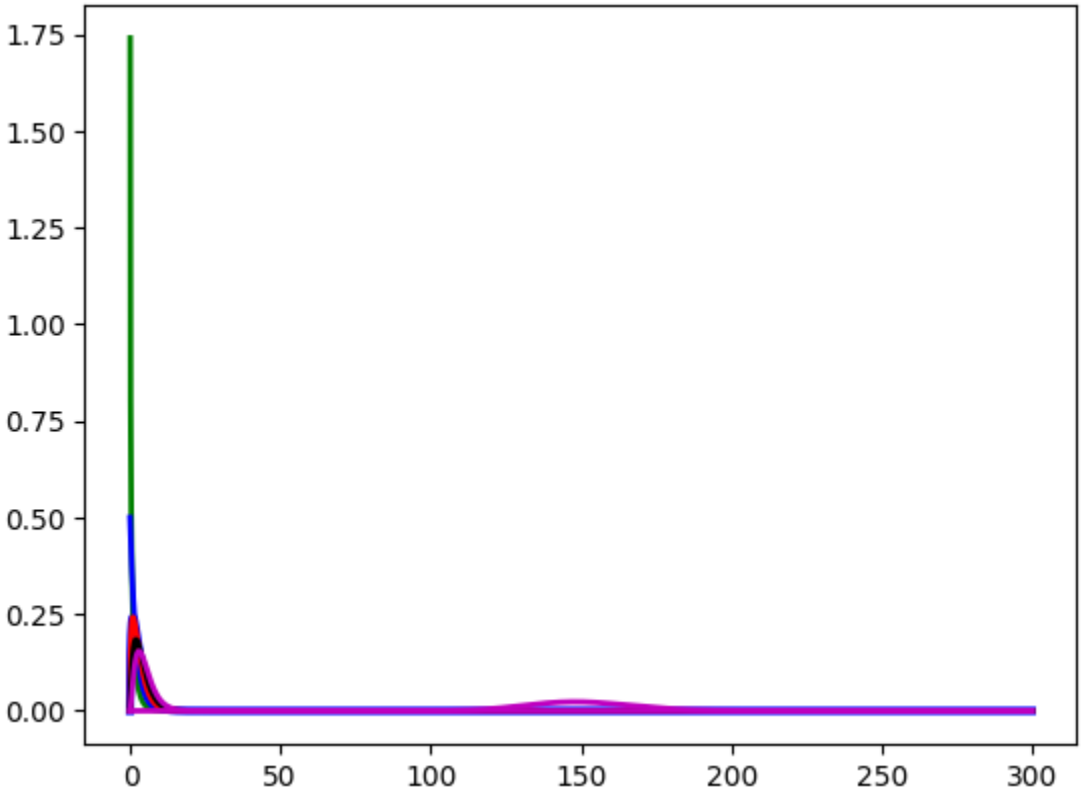
Output:



e) Chi-squared distribution

```
import numpy as np
import scipy.stats as stats
from scipy.stats import gamma
import matplotlib.pyplot as plt
import math
# Plot the theoretical density of f
x = np.arange(0, 300, .05)
df=3
y=[]
for i in range(len(x)):
   y.append(i)

y[i]=(math.exp(-x[i]/2)*(pow(x[i],(df/2)-1)))/(pow(2,df/2)*math.gamma(df/
2))
plt.plot(x,y,color='b',lw=3)
#plt.show()
plt.plot(x, stats.chi2.pdf(x, df=1), color='g', lw=2)
#plt.show()
plt.plot(x, stats.chi2.pdf(x, df=2), color='b', lw=2)
#plt.show()
plt.plot(x, stats.chi2.pdf(x, df=3), color='r', lw=2)
#plt.show()
plt.plot(x, stats.chi2.pdf(x, df=4), color='k', lw=2)
#plt.show()
plt.plot(x, stats.chi2.pdf(x, df=5), color='m', lw=2)
#plt.show()
plt.plot(x, stats.chi2.pdf(x, df=150), color='m', lw=2)
plt.show()
```

Output:



22. Hierarchical Clustering

Aim:

Write a R Program to find Hierarchical Clustering and plot graph.

Algorithm :

Step1: Start the program

Step2: Use matrix(rnorm) with hclust with scaled

Step3: Load Input Variable dataset

Step4: Find the HieraricalClustring using hclust() function

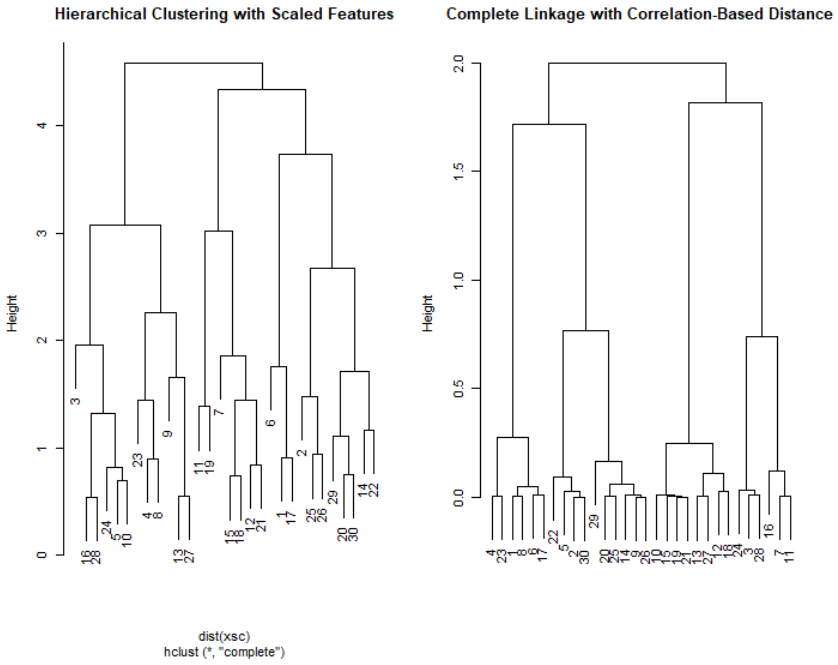
Step5: Plot Graph

Step6: Stop the program

Hierarchical Clustering

```
x = matrix(rnorm(30 * 3), ncol = 3)

hc.complete = hclust(dist(x), method = "complete")
hc.average = hclust(dist(x), method = "average")
hc.single = hclust(dist(x), method = "single")

# Set up the plotting area with 1 row and 3 columns
par(mfrow = c(1, 3))

# Plot dendrograms for different linkage methods
plot(hc.complete, main = "Complete Linkage", xlab = "", sub = "", cex = 0.9)
plot(hc.average, main = "Average Linkage", xlab = "", sub = "", cex = 0.9)
plot(hc.single, main = "Single Linkage", xlab = "", sub = "", cex = 0.9)

# Cut dendrograms to form clusters
cutree(hc.complete, 2)
cutree(hc.average, 2)
cutree(hc.single, 2)
cutree(hc.single, 4)

# Scale the features
xsc = scale(x)

# Plot hierarchical clustering with scaled features
plot(hclust(dist(xsc), method = "complete"), main = "Hierarchical Clustering with
Scaled Features")

# Create a distance matrix based on correlation and perform hierarchical clustering
dd = as.dist(1 - cor(t(x)))
plot(hclust(dd, method = "complete"), main = "Complete Linkage with
Correlation-Based Distance", xlab = "", sub = "")
```

Output:


Hierarchical Clustering with Scaled Features
Height
dist(xsc)
hclust (*, "complete")
Complete Linkage with Correlation-Based Distance
Height

23. K- Means clustering

Aim:

Write a R program to Find K-Means Clustering

Algorithm:

Step 1: Start the Program

Step 2: Load Data set.seed

Step 3: Standardize the Data

Step 4: Check Mean and SD

Step 5: Fitting the Cluster

Step 6: Cross Validate with Original Species available in data

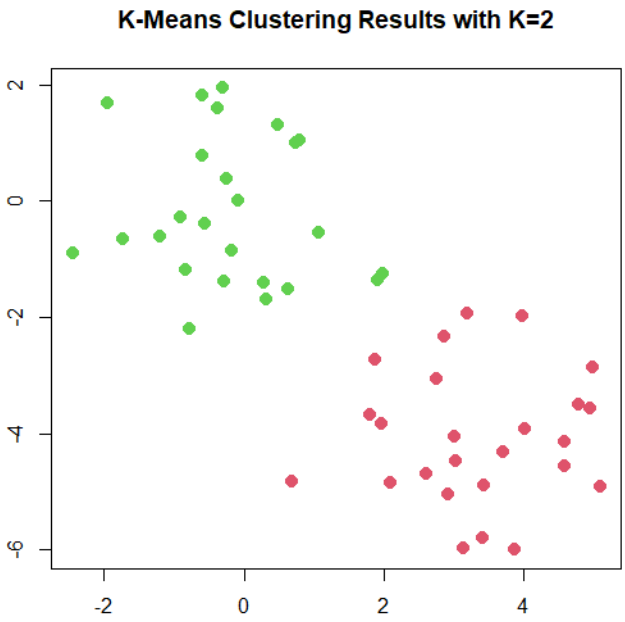
Step 7: PLot graph for K means clustering.

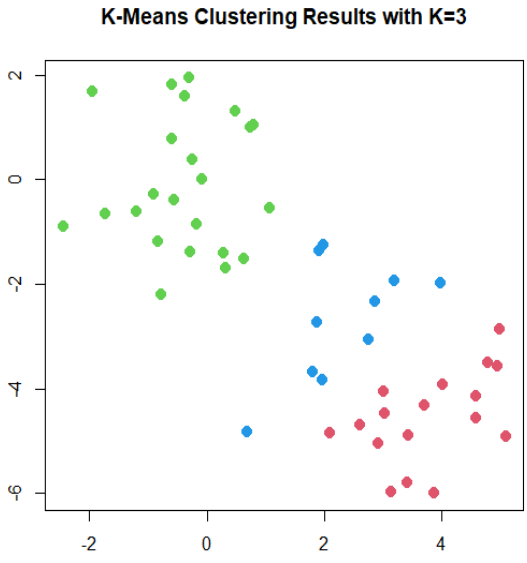
Step 8: Stop the program

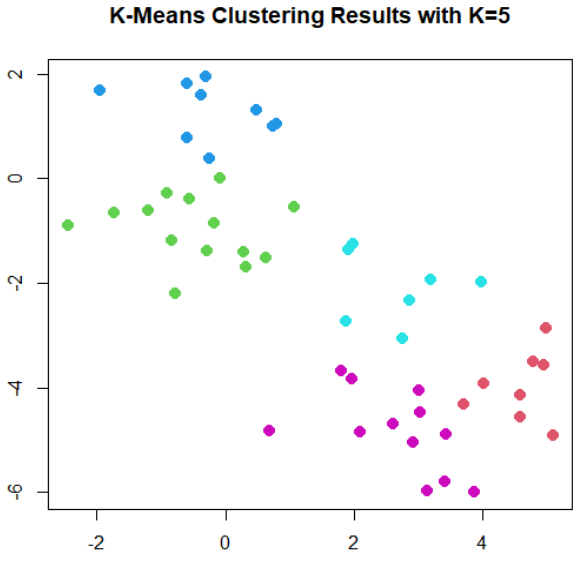
K-Means Clustering

```
set.seed(2)
x=matrix(rnorm(50*2), ncol=2)
x[1:25,1]=x[1:25,1]+3
x[1:25,2]=x[1:25,2]-4
km.out=kmeans(x,2,nstart=20)
km.out$cluster
plot(x, col=(km.out$cluster+1), main="K-Means Clustering Results with
K=2", xlab="", ylab="", pch=20, cex=2)

set.seed(4)
km.out=kmeans(x,3,nstart=20)
km.out
plot(x, col=(km.out$cluster+1), main="K-Means Clustering Results with
K=3", xlab="", ylab="", pch=20, cex=2)

set.seed(4)
km.out=kmeans(x,5,nstart=20)
km.out
plot(x, col=(km.out$cluster+1), main="K-Means Clustering Results with
K=5", xlab="", ylab="", pch=20, cex=2)
```

Output:


K-Means Clustering Results with K=2
2
0
-2
-4
-6
-2
0
2
4


K-Means Clustering Results with K=3
2
0
-2
-4
-6
-2
0
2
4


K-Means Clustering Results with K=5
2
0
-2
-4
-6
-2
0
2
4

24. Case study on Guna’s theory

Aim:

Write a R Program to Case Study on Guna's Theory

Algorithm:

Step1 : Start the program

Step2 : Read the CSV Data From Guna1.Csv file and Store it in Guna variable

Step3 : Use Comparison of Attitude and Apptitude, Calculate Mean and Standard Deviation, boxplot(),barplot(), histogram and more function to plot the csv file

Step4 : Load the library lattice

Step5 : And Perform all the Plotting Function For Case Study

Step6 : Stop the Program

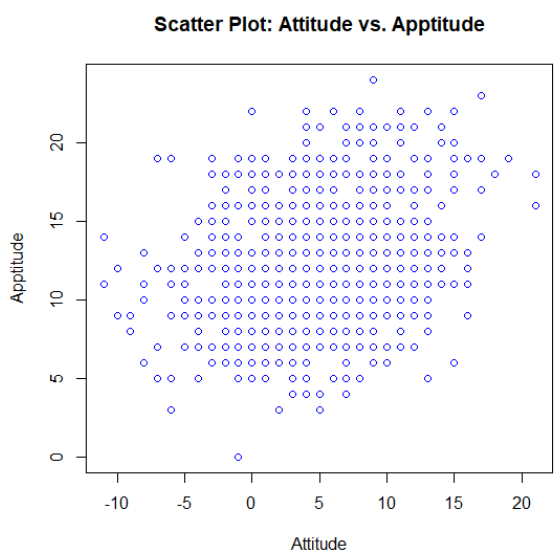
Case Study on Guna's Theory

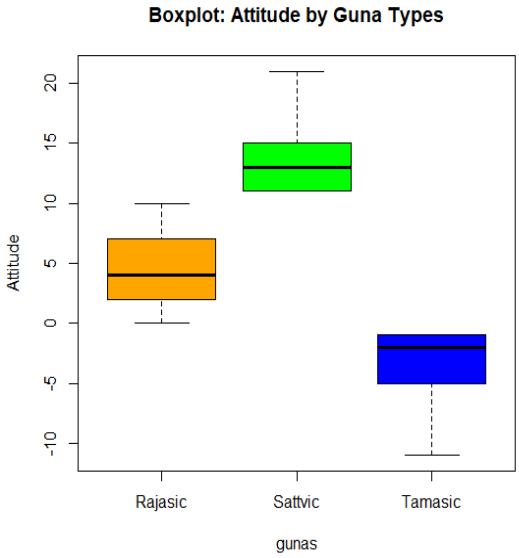
```
data <- read.csv("guna1.csv")

# Comparison of Attitude and Apptitude
attitude <- data$Attitude
apptitude <- data$Apptitude

# Calculate Mean and Standard Deviation
mean_attitude <- mean(attitude)
sd_attitude <- sd(attitude)

cat("Attitude - Mean:", mean_attitude, "SD:", sd_attitude, "\n")
cat("Apptitude - Mean:", mean_apptitude, "SD:", sd_apptitude, "\n")

# Create a Scatter Plot
plot(attitude, apptitude, main = "Scatter Plot: Attitude vs. Apptitude", xlab =
"Attitude", ylab = "Apptitude", col = "blue")

# Create a Boxplot for Attitude and Apptitude
boxplot(attitude, apptitude, names = c("Attitude", "Apptitude"), main =
"Boxplot: Attitude vs. Apptitude", col = c("orange", "green"))

# Create a Histogram for Attitude and Apptitude
par(mfrow = c(1, 2))
hist(attitude, col = "lightblue", main = "Histogram: Attitude", xlab =
"Attitude", ylab = "Frequency")
hist(apptitude, col = "lightgreen", main = "Histogram: Apptitude", xlab =
"Apptitude", ylab = "Frequency")
par(mfrow = c(1, 1))

# Correlation Matrix
cor_matrix <- cor(data[, c("Attitude", "Apptitude", "Sattvic", "Rajasic",
"Tamasic")])
print("Correlation Matrix:")
print(cor_matrix)

# Hypothesis Testing (ANOVA) for Attitude and Apptitude based on Gunas
```

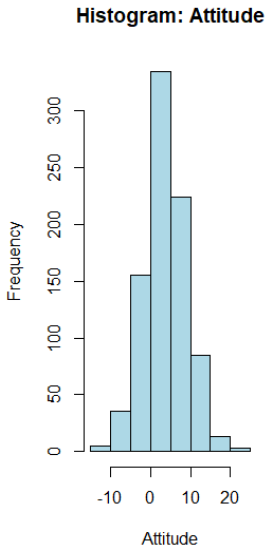
```
anova_result <- aov(Attitude ~ Sattvic + Rajasic + Tamasic + Apptitude,
data = data)
print("ANOVA Result:")
print(anova_result)

# Correlation Matrix
cor_matrix <- cor(data[, c("Attitude", "Apptitude", "Sattvic", "Rajasic",
"Tamasic")])
print("Correlation Matrix:")
print(cor_matrix)

# Pairwise Scatter Plots
pairs(data[, c("Attitude", "Apptitude", "Sattvic", "Rajasic", "Tamasic")],
main = "Pairwise Scatter Plots")

# Create a Boxplot for Attitude based on Guna Types
boxplot(Attitude ~ gunas, data = data, main = "Boxplot: Attitude by Guna
Types", col = c("orange", "green", "blue"))

# Bar Plot for Apptitude by Guna Types
barplot(data$Apptitude, beside = TRUE, names.arg = data$gunas, col =
c("orange", "green", "blue"), main = "Bar Plot: Apptitude by Guna Types",
xlab = "Guna Types", ylab = "Apptitude")
```

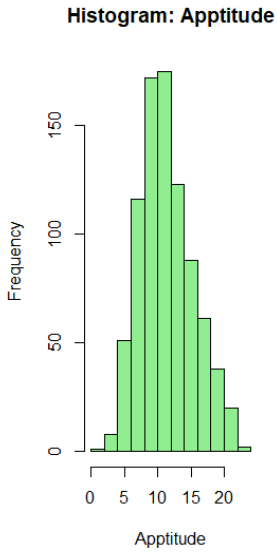
Output:

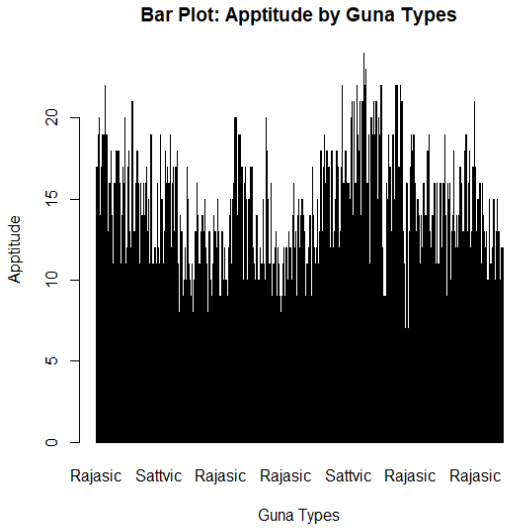
```
Attitude - Mean: 4.249123 SD: 5.256861

Apptitude - Mean: 11.8269 SD: 3.99947
```

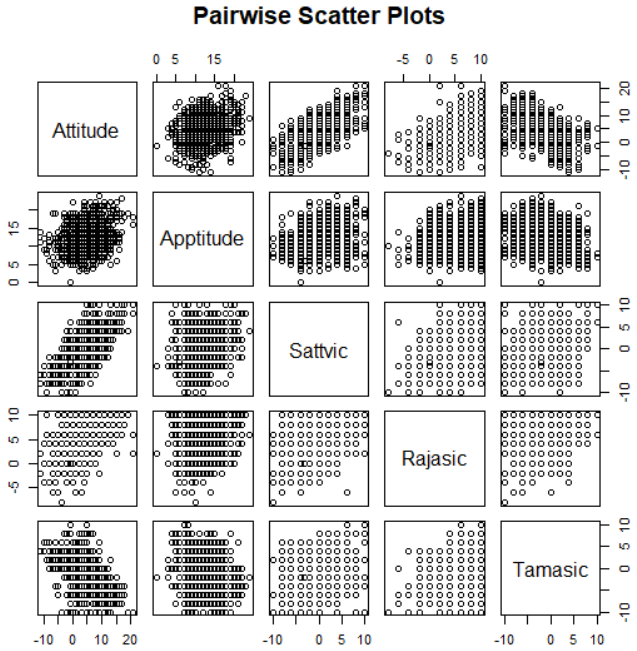













```
Residual standard error: 3.407374e-14
Estimated effects may be unbalanced
```

25. Analysis of Variance (ANOVA)

Aim:

To write a program to analyze variance using R.

Algorithm:

Step 1: Open R studio. Start the program.

Step 2: Use mtcars amd regression model to compare two data

Step 3: Type the interval estimation data.

Step 4: Set working directory.

Step 5: Process the data.aov(mpg~hp*cyl,data = input)

Step 6: Get the output.

Step 7: Save and exit the program

Analyze of variance

```
#Live Demo
input <- mtcars[,c("am","mpg","hp")]
print(head(input))

#Live Demo
# Get the dataset.
input <- mtcars

# Create the regression model.
result <- aov(mpg~hp*am,data = input)
print(summary(result))

#Live Demo
# Get the dataset.
input <- mtcars

# Create the regression model.
result <- aov(mpg~hp+am,data = input)
print(summary(result))

#Live Demo
# Get the dataset.
input <- mtcars

# Create the regression models.
result1 <- aov(mpg~hp*am,data = input)
result2 <- aov(mpg~hp+am,data = input)

# Compare the two models.
print(anova(result1,result2))


print(input)
result1 <- aov(mpg~hp*cyl,data = input)
result2 <- aov(mpg~hp+cyl,data = input)

# Compare the two models.
print(anova(result1,result2))
```

Output:

```
(summary(result))
            Df Sum Sq Mean Sq F value   Pr(>F)
hp           1  678.4   678.4   80.15 7.63e-10 ***
am           1  202.2   202.2   23.89 3.46e-05 ***
Residuals   29  245.4     8.5
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Analysis of Variance Table

```
Model 1: mpg ~ hp * am
Model 2: mpg ~ hp + am
  Res.Df    RSS Df  Sum of Sq     F Pr(>F)
1     28 245.43
2     29 245.44 -1 -0.0052515 6e-04 0.9806
```

Analysis of Variance Table

```
Model 1: mpg ~ hp * cyl
Model 2: mpg ~ hp + cyl
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     28 247.60
2     29 291.98 -1   -44.375 5.0181 0.0332 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

26. Wilcox on signed-rank test

Aim:

To write a program Wilcox on Signed-rank test in google colab.

Algorithm:

Step1: To Open the google colab in chrome.

Step 2: Use the library Dataset bloodpressure.csv

Step 3: Then data values of the wilcoxon

Step 4: Print data with using print(stats.wilcoxon(x, y))

Step 5: Then Finally execute and Save the program.

Wilcox on signed-rank test

```
df = pd.read_csv("/content/drive/MyDrive/blood_pressure1.csv")
#print(df.describe())
x=df['bp_before']
y=df['bp_after']
t=stats.wilcoxon(x,y)
print(stats.wilcoxon(x, y))
#print(df[['bp_before','bp_after']].describe())
if (t[1]>0.05):
   print("There is no difference between x and y")
else:
   print("There is significance between x and y")
```

Output:

```
WilcoxonResult(statistic=17.0, pvalue=0.0478515625) There is significance
between x and y WilcoxonResult(statistic=2234.5,
pvalue=0.0014107333565442858) There is significance between x and y
```

27. Time Series analysis

Aim:

To write a program time series analysis

Algorithm:

Step1: To Open the google colab in chrome.

Step 2: Use the library Dataset or set some input data

Step 3: To find some time series analysis of

a) Moving average
b) Auto regression
c) ARIMA model
d) Time series analysis using stock data
e) Time series analysis using weather temperature data

Step 4: For Time series analysis using stock data use stock_data.csv file and use daily weather data.csv for time series analysis using weather temperature data

Step 5: Then Finally execute and Save the program.

a). Moving Average:

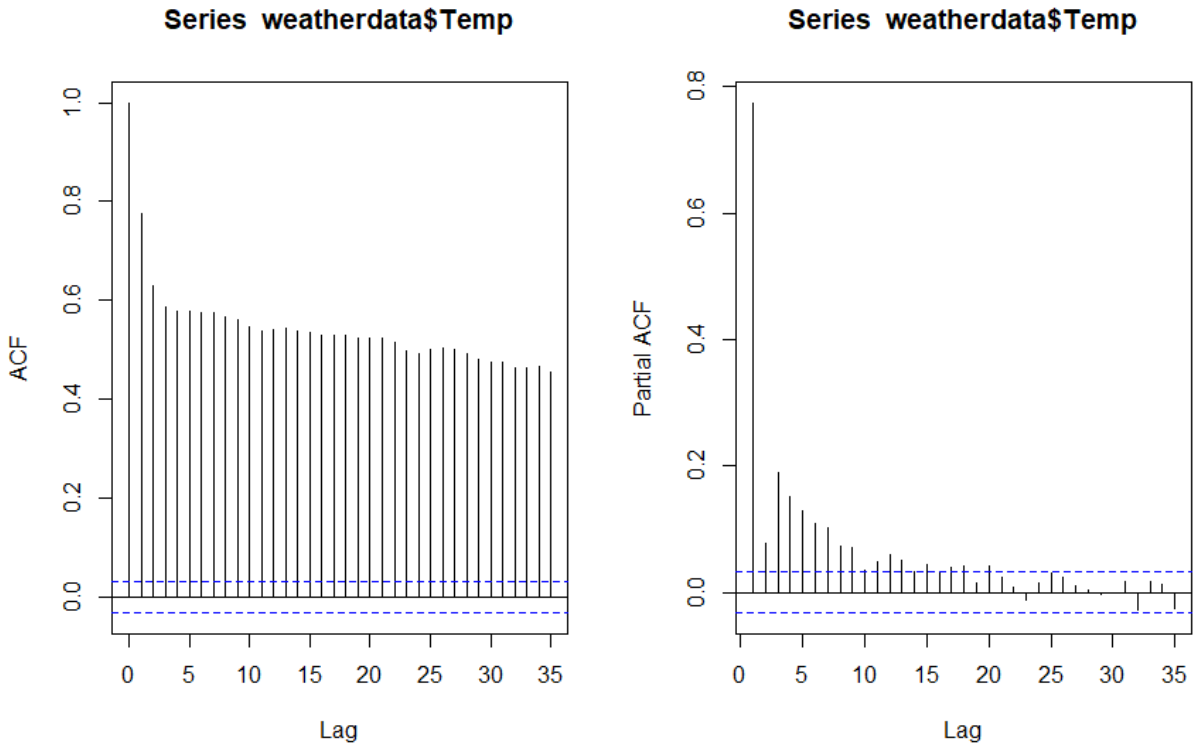
```
library(TTR)
x <- c(3,5,7,3,4,2,6,4,7,2,1,9, 1, 10, 1,12)
y=x
By <- diff(y) # y_i - B y_i
print(By)
B2y<-diff(y,2)
B3y <- diff(y, 3) # y_i - B^3 y_i
message(paste0("y is: ", paste(y, collapse = ","), "\n",
               "By is: ", paste(By, collapse = ","), "\n",
               "By2 is: ",paste(B2y, collapse = ","), "\n",
               "B3y is: ", paste(B3y, collapse = ",")))
get_autocor <- function(x, lag) {
  x.left <- x[1:(length(x) - lag)]
  x.right <- x[(1+lag):(length(x))]
  print(x.left)
  print(x.right)
  autocor <- cor(x.left, x.right)
  return(autocor)}
get_autocor(y, 1)
get_autocor(y, 2)
get_autocor(y,3)
get_autocor(y, 4)
```

#Moving Average

```
print(x)
filter(x, rep(1/3,3))
runMean(x,2)
weatherdata <- read.table('C:/Users/MCA-29/R
Practical/daily-min-temperatures.csv', sep=",", header= TRUE)
head(weatherdata)
plot(weatherdata$Temp)
plot(log(weatherdata$Temp))
```

```
par(mfrow = c(1,2))
acf(weatherdata$Temp)
pacf(weatherdata$Temp)
```

Output:

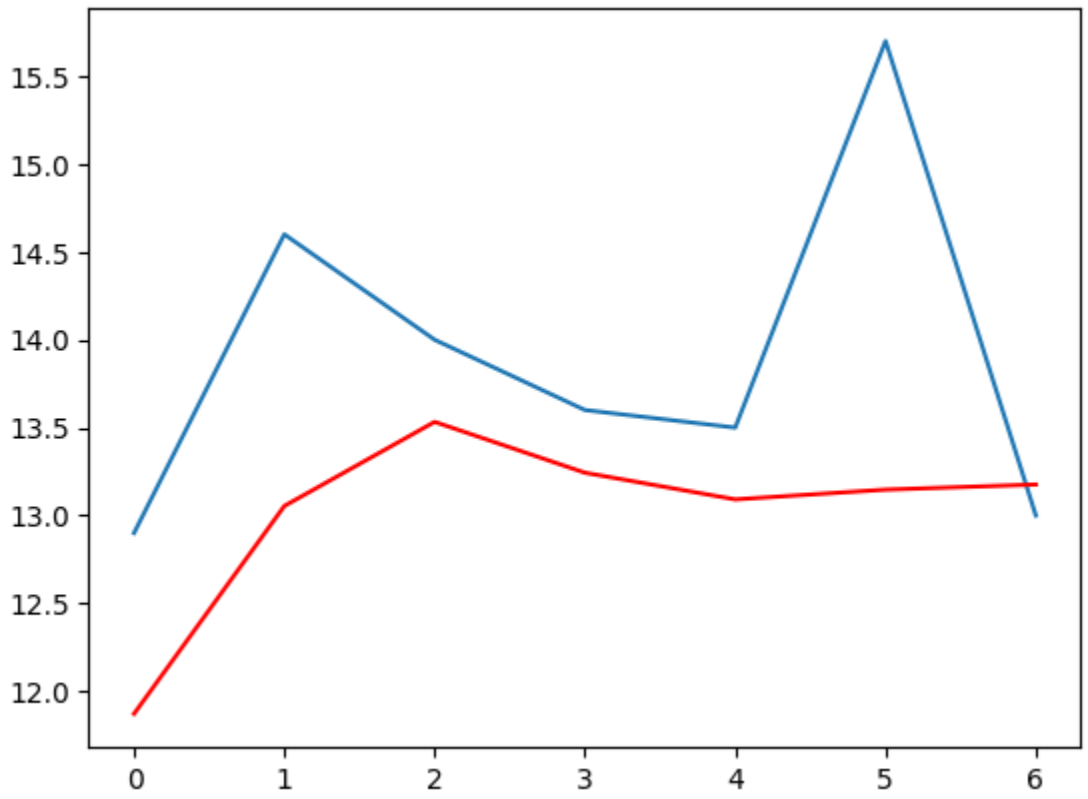


b). Auto Regression (AR):

```
from pandas import read_csv
from matplotlib import pyplot
from statsmodels.tsa.ar_model import AutoReg
from sklearn.metrics import mean_squared_error
from math import sqrt
# load dataset
#series = read_csv('daily-min-temperatures.csv', header=0, index_col=0,
parse_dates=True, squeeze=True)
# split dataset
X = series.values
train, test = X[1:len(X)-7], X[len(X)-7:]
# train autoregression
model = AutoReg(train, lags=29)
model_fit = model.fit()
print('Coefficients: %s' % model_fit.params)
# make predictions
predictions = model_fit.predict(start=len(train), end=len(train)+len(test)-1,
dynamic=False)
for i in range(len(predictions)):
  print('predicted=%f, expected=%f' % (predictions[i], test[i]))
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)
# plot results
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```

Output:

```
Coefficients: [ 5.57543506e-01  5.88595221e-01 -9.08257090e-02
4.82615092e-02
  4.00650265e-02  3.93020055e-02  2.59463738e-02  4.46675960e-02
  1.27681498e-02  3.74362239e-02 -8.11700276e-04  4.79081949e-03
  1.84731397e-02  2.68908418e-02  5.75906178e-04  2.48096415e-02
  7.40316579e-03  9.91622149e-03  3.41599123e-02 -9.11961877e-03
  2.42127561e-02  1.87870751e-02  1.21841870e-02 -1.85534575e-02
 -1.77162867e-03  1.67319894e-02  1.97615668e-02  9.83245087e-03
  6.22710723e-03 -1.37732255e-03]
predicted=11.871275, expected=12.900000
predicted=13.053794, expected=14.600000
predicted=13.532591, expected=14.000000
predicted=13.243126, expected=13.600000
predicted=13.091438, expected=13.500000
predicted=13.146989, expected=15.700000
predicted=13.176153, expected=13.000000
Test RMSE: 1.225
```

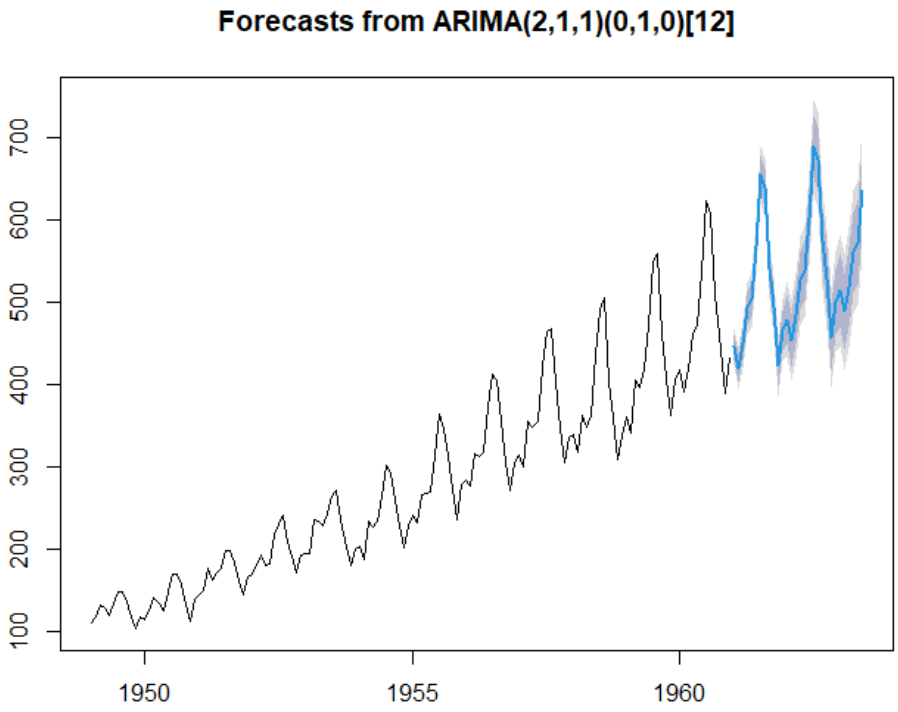


c). ARIMA Model:

```
library(ggfortify)
library(tseries)
library(forecast)

data(AirPassengers)
head(AirPassengers)
print(AirPassengers)
AP <- AirPassengers
class(AP)

AP
sum(is.na(AP))

frequency(AP)
cycle(AP)
summary(AP)

library(ggfortify)

#ARIMA Model
arimaAP <- auto.arima(AP)
arimaAP
ggtsdiag(arimaAP)

forecastAP <- forecast(arimaAP, level = c(95), h = 36)
autoplot(forecastAP)

A <- arima(AirPassengers, order = c(2,1,1),seasonal=c(0,1,0))
library(forecast)
forecast <- forecast(A, h = 30) # predict 30 years into the future
plot(forecast)
```

Output:

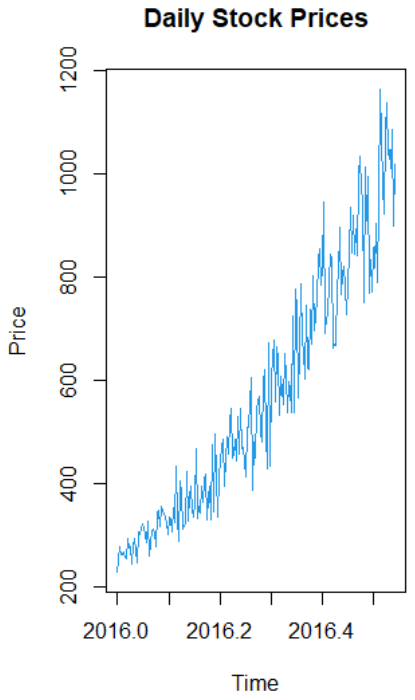
```
   Min. 1st Qu.  Median    Mean   3rd Qu.    Max.
  104.0   180.0   265.5   280.3   360.5   622.0
```

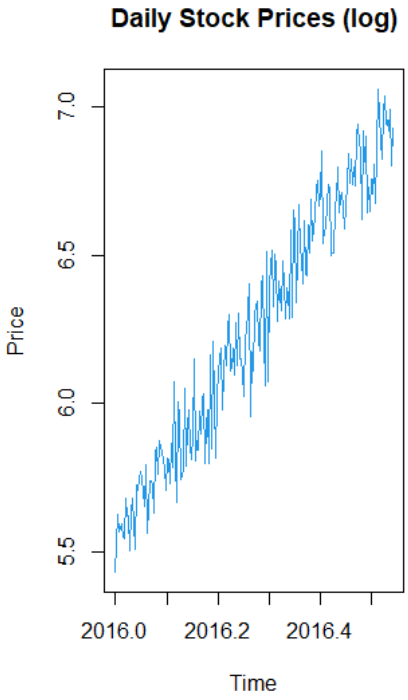

Forecasts from ARIMA(2,1,1)(0,1,0)[12]
100
200
300
400
500
600
700
1950
1955
1960

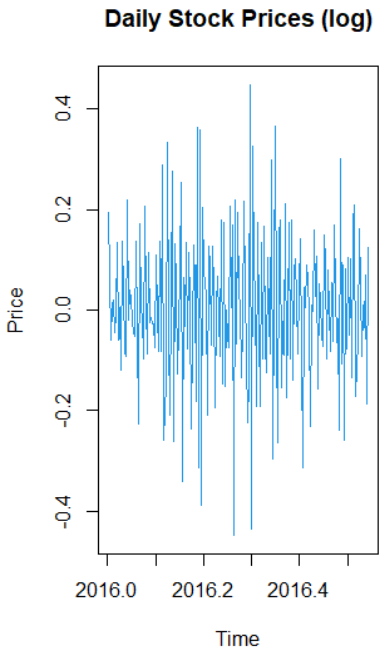
d). Time Series Analysis using Stock Data:

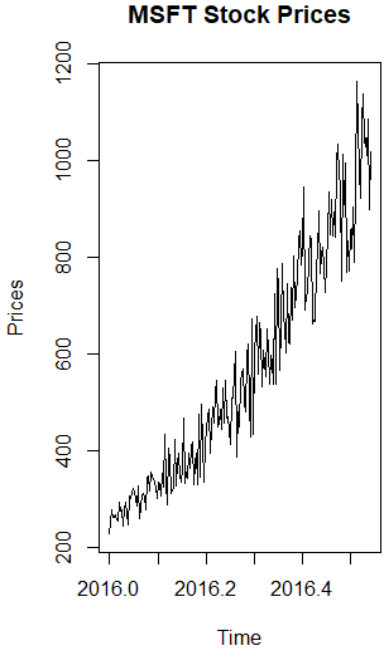
```
sp<-read.table('E:/MCA/Unit iv/stock_daily.csv', sep=",", header=
TRUE)
print(sp)
sp_ts <- ts(sp, start=2016, frequency=365)
print(sp_ts)
plot.ts(sp_ts,col=10, main="Daily Stock Prices", ylab="Price")

sp_linear<-log(sp_ts)
plot.ts(sp_linear, main="Daily Stock Prices (log)", ylab="Price",
col=4)
par(mfrow = c(1,2))
plot.ts(sp_ts,col=4, main="Daily Stock Prices", ylab="Price")
plot.ts(sp_linear, main="Daily Stock Prices (log)", ylab="Price",
col=4)

sp_linear_diff <- diff(sp_linear)
plot.ts(sp_linear_diff, main="Daily Stock Prices (log)", ylab="Price",
col=4)
par(mfrow = c(1,2))
de_earnings_diff <- diff(johndeere_earnings,lag=4)
plot.ts(johndeere_earnings, main="Earnings (Quarterly)")
plot.ts(de_earnings_diff, main="Earnings (Differenced, lag=4)")

par(mfrow = c(1,2))
plot.ts(sp_ts,main="MSFT Stock Prices",ylab="Prices")
sp_ts_lag1=diff(sp,1)
plot(sp_ts,sp_ts_lag1,main="Scatterplot (lag=1)")
cor(sp_ts[-252],sp_ts[-1])
abline(lm(sp_ts[-1] ~ sp_ts[-252]),col=4)

acf(sp_ts, lag.max=10)
cor(sp_ts[-(251:252)],sp_ts[-(1:2)])
```

Output:


Daily Stock Prices
Price
200
400
600
800
1000
1200
2016.0
2016.2
2016.4
Time


Daily Stock Prices (log)
Price
5.5
6.0
6.5
7.0
2016.0
2016.2
2016.4
Time

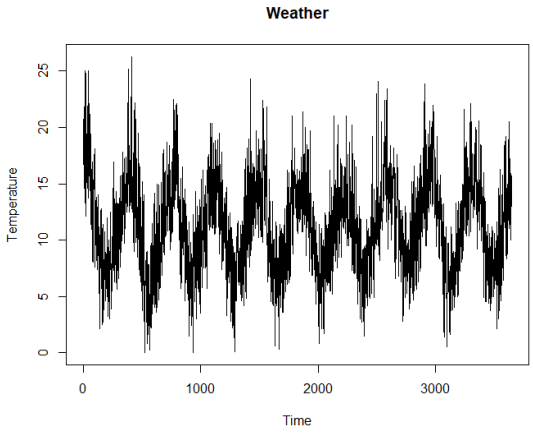

Daily Stock Prices (log)
Price
-0.4
-0.2
0.0
0.2
0.4
2016.0
2016.2
2016.4
Time

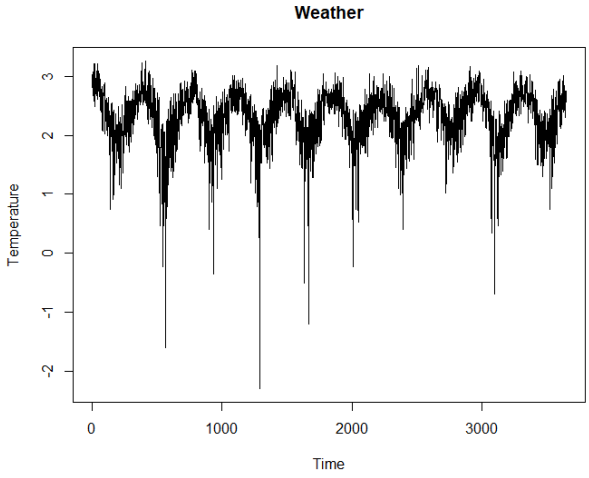

MSFT Stock Prices
Prices
200
400
600
800
1000
1200
2016.0
2016.2
2016.4
Time

e). Time Series Analysis using Weather Temperature Data:

```
weatherdata <- read.table('C:/Users/MCA-29/R
Practical/daily-min-temperatures.csv', sep=",", header= TRUE)
head(weatherdata)
plot.ts(weatherdata$Temp, main="Weather", ylab="Temperature")
plot.ts(log(weatherdata$Temp),main="Weather", ylab="Temperature")
temperature.timeseries <- ts(weatherdata$Temp)
plot.ts(temperature.timeseries)
datats=ts(weatherdata,start = c(2012,1),frequency = 12)
plot(decompose(datats))
msft_ar <- arima(temperature.timeseries, order = c(1, 2, 2))
print(msft_ar)
```

Output:





28.Decision trees on any dataset

Aim:

To write a program on Decision tree with any dataset

Algorithm:

Step1: Open R Package

Step2: install.packages("party")

Step3: The package "party" has the function ctree() which is used to create and analyze decision trees.

Step4: We will use the R in-built data set named readingSkills to create a decision tree.

Step5: Create the input data frame.

Step6: Give the chart file a name png(file = "decision_tree.png")

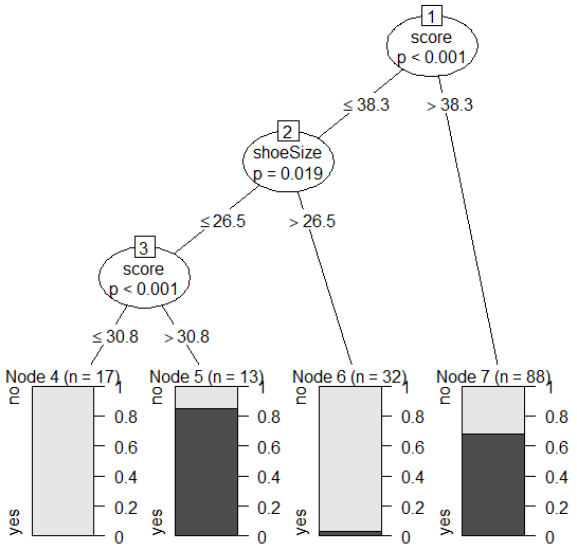
Step7: Create a tree and plot a tree

Step8: Then Finally execute and Save the program.

Decision tree with any dataset

```
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)
```

Output:



29. Random Forest using Titanic dataset

Aim:

To write a program on Random forest using Titanic dataset

Algorithm:

Step1: Open the google colab in chrome

Step2: Mount drive file named possum.csv

Step3: Load the random forest package. It will automatically load other required packages.

Step 4: Use sklearn.ensemble import RandomForestClassifier in colab

Step 5: Print the output:

Step 6: Then Finally execute and Save the program.

Random forest using Titanic dataset

```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks /possum.csv")
df.sample(5, random_state=44)
df = df.dropna()
X = df.drop(["case", "site", "Pop", "sex"], axis=1)
y = df["sex"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=44)
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=50, max_features="auto",
random_state=44)
rf_model.fit(X_train, y_train)

predictions = rf_model.predict(X_test)
print(predictions)

print(y_test)

importances=rf_model.feature_importances_
columns=X.columns
print(columns)
i=0
while(i<len(columns)):
  print(columns[i],round(importances[i]*100,2))
  i=i+1
```

Output:

```
      ['f' 'm' 'm' 'm' 'f' 'f' 'm' 'm' 'm' 'm' 'm' 'f' 'm' 'm' 'm' 'm' 'm'
'f'
 'm' 'm' 'm' 'f' 'f' 'm' 'm' 'm' 'm' 'm' 'm' 'm' 'f']
57     m
80     m
27     m
97     m
61     f
69     f
7      f
26     f
24     m
65     f
71     m
30     m
38     f
76     m
19     f
103    f
44     m
33     m
11     f
72     m
83     m
39     f
5      f
73     f
51     m
94     m
56     f
96     m
89     m
90     m
98     f
Name: sex, dtype: object
Index(['age', 'hdlngth', 'skullw', 'totlngth', 'taill', 'footlgth',
'earconch',
       'eye', 'chest', 'belly'],
      dtype='object')
age 5.24
hdlngth 14.62
skullw 8.88
totlngth 11.32
taill 7.15
footlgth 16.1
earconch 11.99
eye 11.08
chest 5.29
belly 8.34
```

30.Survival analysis

Aim:

Write a program to Survival analysis.

Algorithm:

Step 1: Open the Google colab in chrome.

Step 2: import the package of KaplanMeierFitter.

Step 3: Create library estimates to plot kmf.

Step 4: Print the first few rows.

Step 5: Create the survival object.

step 6: Plot the graph.

Step 6: Save and run the program.

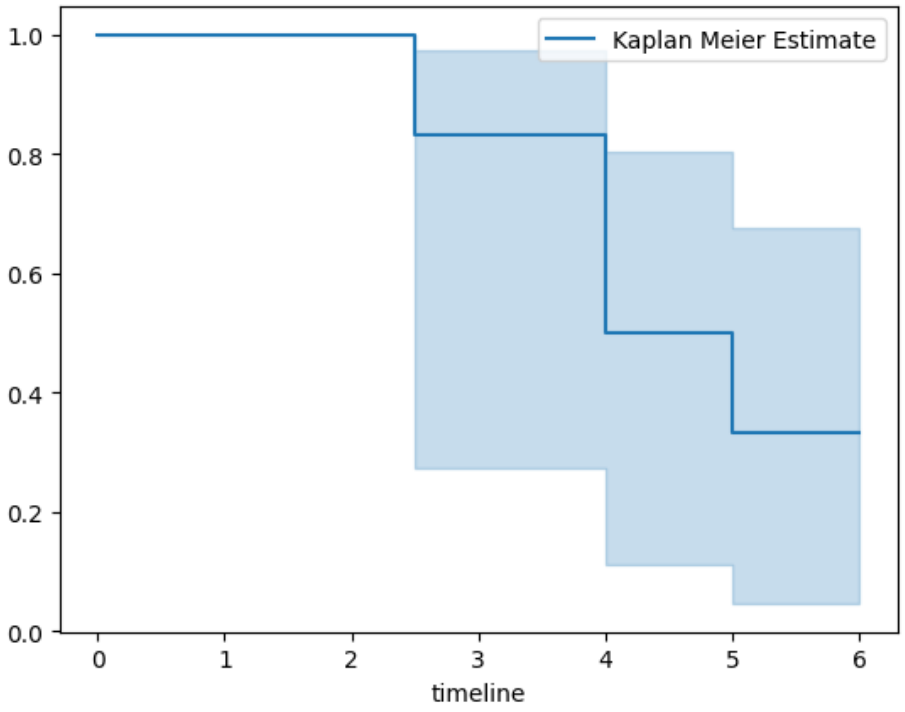
Step 7: Verify the output in the output area.

Survival analysis

```
from lifelines import KaplanMeierFitter
durations = [5,6,6,2.5,4,4]
event_observed = [1, 0, 0, 1, 1, 1]

kmf = KaplanMeierFitter()
kmf.fit(durations, event_observed,label='Kaplan Meier Estimate')

## Create an estimate
kmf.plot(ci_show=True)
```

Output:

<Axes: xlabel='timeline'>



31.Mathematical functions using Numpy

Aim:

Write mathematical functions using numpy in Colab.

Algorithm:

Step 1: Open the google colab in chrome

Step 2: Import numpy and math library

Step 3: Use the math functions like sin, cos, sinhetc..

Step 4: Declare the array and use the appropriate math functions and print the array values.

Step 5: By using numpy and math write mathematical functions one by one

Step 6: Save the program, run it one by one and get the output.

Mathematical functions using numpy

Sin()

```
import numpy as np
import math
in_array= [0, math.pi / 2, np.pi / 3, np.pi]
print("Input array:\n",in_array)
Sin_Values = np.sin(in_array)
print("\nSine values\n",Sin_Values)
```

Output:

Input array: [0, 1.5707963267948966, 1.0471975511965976, 3.141592653589793]
Sine values [0.00000000e+00 1.00000000e+00 8.66025404e-01 1.22464680e-16]

Cos()

```
import numpy as np
import math
in_array= [0, math.pi / 2, np.pi / 3, np.pi]
print("Input array:\n",in_array)
Cos_Values = np.cos(in_array)
print("\nCose values\n",Cos_Values)
```

Output

Input array: [0, 1.5707963267948966, 1.0471975511965976, 3.141592653589793] Cose values [1.000000e+00 6.123234e-17 5.000000e-01 -1.000000e+00]

sinh()

```
import numpy as np
import math
in_array= [0, math.pi / 2, np.pi / 3, np.pi]
```

```
print("Input array:\n",in_array)
Sinh_Values = np.sinh(in_array)
print("\nSinh Hyperbolic values\n",Sinh_Values)
```

output

```
Input array: [0, 1.5707963267948966, 1.0471975511965976,
3.141592653589793]
Sinh Hyperbolic values [ 0. 2.3012989 1.24936705 11.54873936]
```

Cosh()

```
import numpy as np
import math
in_array= [0, math.pi / 2, np.pi / 3, np.pi]
print("Input array:\n",in_array)
Cosh_Values = np.cosh(in_array)
print("\nCosh Hyperbolic values\n",Cosh_Values)
```

output

```
Input array: [0, 1.5707963267948966, 1.0471975511965976,
3.141592653589793] Cosh Hyperbolic values [ 1. 2.50917848 1.60028686
11.59195328]
```

Around ()

```
import numpy as np
in_array= [.5, 1.5, 2.5, 3.5, 4.5, 10.1]
print ("Input array : \n", in_array)
round_off_values = np.around(in_array)
print ("\nRounded values : \n", round_off_values)
in_array= [.53, 1.54, .71]
print ("\nInput array : \n", in_array)
round_off_values = np.around(in_array)
```

```
print ("\nRounded values : \n", round_off_values)
in_array = [.5538, 1.33354, .71445]
print ("\nInput array: \n", in_array)
round_off_values = np.around(in_array, decimals = 3)
```

Output:

```
Input array : [0.5, 1.5, 2.5, 3.5, 4.5, 10.1]
Rounded values : [ 0. 2. 2. 4. 4. 10.]
Input array : [0.53, 1.54, 0.71]
 Rounded values : [1. 2. 1.]
Input array: [0.5538, 1.33354, 0.71445]
```

Exponential ()

```
import numpy as np
in_array = [1, 3, 5]
print ("Input array: ", in_array)
out_array= np.exp(in_array)
print ("Output array : ", out_array)
import numpy as np
in_array= [1, 3, 5]
print ("Input array: ", in_array)
out_array= np.exp(in_array)
print ("Output array: ", out_array)
```

Output:

```
Input array: [1, 3, 5]
Output array : [ 2.71828183 20.08553692 148.4131591 ]
 Input array: [1, 3, 5]
Output array: [ 2.71828183 20.08553692 148.4131591 ]
```

log()

```
import numpy as np
```

```
in_array= [1, 3, 5, 2**8]
print ("Input array: ", in_array)
out_array= np.log(in_array)
print ("Output array : ", out_array)
print("np.log(28) : ", np.log(28))
```

OUTPUT

arr1 : [2, 27, 2, 21, 23] arr2 : [2, 3, 4, 5, 6] Output array: [1. 9. 0.5 4.2 3.83333333]

Complex number

```
import numpy as np
in_complx1 = 2+4j
out_complx1 = np.conj(in_complx1)
print ("Output conjugated complex number of 2+4j : ", out_complx1)
in_complx2 =5-8j
out_complx2 = np.conj(in_complx2)
print ("Output conjugated complex number of 5-8j: ", out_complx2)
```

Output

Output conjugated complex number of 2+4j : (2-4j)
Output conjugated complex number of 5-8j: (5+8j)

32.Data analysis using Pandas

Aim:

To write a program in Data analytics using Pandas.

Algorithm:

Step 1: Start the program

Step 2:Load the PANDAS package

Step 3:Import the Imbd.csv file and read the CSV File

Step 4: Run the CSV file

Step 5: To take a action data.head(), data.info(), data.shape(), data.describe(), data.dropna()

Step 6: Verify the output

Step 7: Exit the program

Data analytics using Pandas

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
movies_df = pd.read_csv("IMDB-Movie-Data.csv", index_col="Title")

movies_df.head()
movies_df.tail()
movies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine
Lives
Data columns (total 11 columns):
# Column Non-Null Count Dtype --- ------
-------------- -----

0    Rank           1000 non-null  int64
1    Genre          1000 non-null  object
2    Description    1000 non-null  object
3    Director       1000 non-null  object
4    Actors         1000 non-null  object
5    Year           1000 non-null  int64
6    Runtime        1000 non-null  int64
7    Rating         1000 non-null  float64
8    Votes          1000 non-null  int64
9    Revenue        872  non-null  float64
10   Metascore      936  non-null  float64
Dtypes:        float64(3), int64(4), object(4)
memory usage:  93.8+ KB
```

```
movies_df.shape
```

```
(1000, 11)
```

```
movies_df.columns
```

```
     Index(['Rank', 'Genre', 'Description', 'Director',
'Actors', 'Year', 'Runtime (Minutes)', 'Rating', 'Votes',
'Revenue (Millions)', 'Metascore'], dtype='object')
```

```
movies_df.rename(columns={
      'Runtime (Minutes)': 'Runtime',
      'Revenue (Millions)': 'Revenue_millions'
   }, inplace=True)
movies_df.columns
```

```
     Index(['Rank', 'Genre', 'Description', 'Director',
     'Actors', 'Year', 'Runtime', 'Rating', 'Votes',
     'Revenue_millions', 'Metascore'], dtype='object')
```

```
movies_df.isnull().sum()
```

```
     Rank 0
     Genre 0
     Description 0
     Director 0
     Actors 0
     Year 0
     Runtime 0
     Rating 0
     Votes 0
     Revenue_millions 128
     Metascore 64
     dtype: int64
```

```
revenue = movies_df['Revenue_millions']
```

```
revenue.head()
```

```
     Title Guardians of the Galaxy 333.13
     Prometheus                    126.46
     Split                         138.12
     Sing                          270.32
     Suicide Squad                 325.02
```

```
Name:          Revenue_millions,
dtype:                  float64
```

```
revenue_mean = revenue.mean()

#revenue_mean
print(movies_df['Revenue_millions'].mean())
```

```
82.95637614678898
```

```
movies_df['Genre'].describe()
```

```
count     1000
unique    207
top       Action,Adventure,Sci-Fi
freq      50
Name:     Genre,
dtype:    object
```

```
movies_df.describe()

movies_df['Genre'].value_counts().head(10)
```

```
Action,Adventure,Sci-Fi       50
Drama                         48
Comedy,Drama,Romance          35
Comedy                        32
Drama,Romance                 31
Animation,Adventure,Comedy    27
Action,Adventure,Fantasy      27
Comedy,Drama                  27
Comedy,Romance                26
Crime,Drama,Thriller          24
Name: Genre, dtype: int64
```

```
movies_df['Genre'].value_counts().head(207)
print(movies_df['Genre'].value_counts().head(50))
```

```
movies_df[movies_df['Rating'] >= 8.6]
movies_df[(movies_df['Rating']>8.6) & (movies_df['Rating']<9.0)]
movies_df[movies_df['Director'].isin(['Christopher Nolan', 'Ridley
Scott'])].head()

def rating_function(x):
    if x >= 8.0:
        return "good"
    else:
        if (x>=7.0) & (x<8.0):
            return "better"
        else:
            return "bad"
#Now we want to send the entire rating column through this function, which
is what apply() does:
movies_df["Rating_category"] =
movies_df["Rating"].apply(rating_function)

movies_df.head()
```

33. Visual representation using Matplotlib

Aim:

To write a program in Visual representations using Matplotlib.

Algorithm:

Step1:Start the program

Step2:Load the library matplotlib as pl and np in the program

Step3 Import and read the matplotlib inline package

Step4:Then use keyword “force = TRUE"

Step5:To use logical functions to calculate single line plotting , multi line plotting , logspace etc..

Step6:Run the Matplotlib program

Step7:Verify the output

Step8:Exit the program

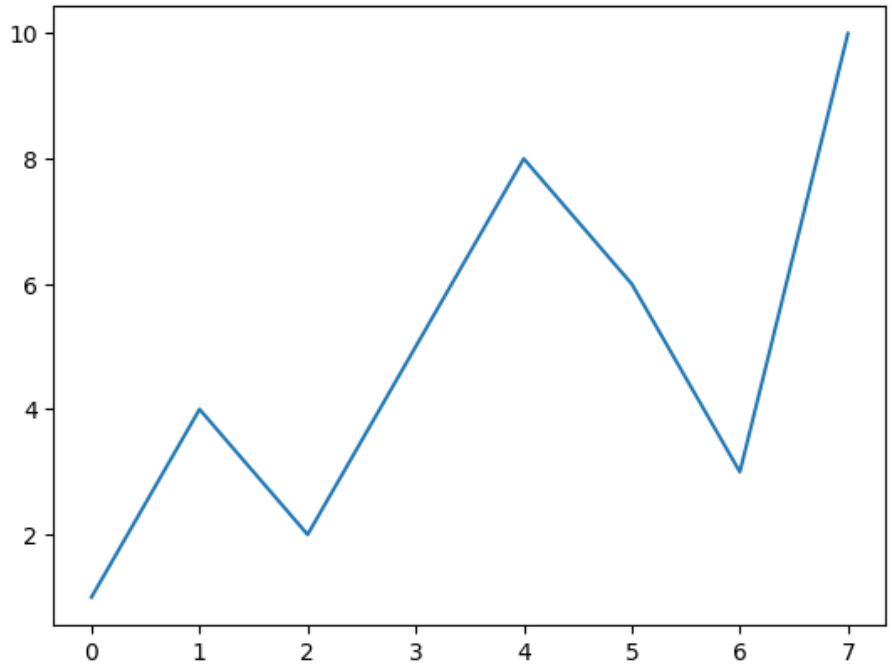
Visual representations using Matplotlib

```
matplotlib inline
import numpy as np
import matplotlib.pyplot as pl
x = [1, 4, 2, 5, 8, 6, 3, 10]
```

SingleLine plotting

```
plt.plot(x)
plt.show()
```

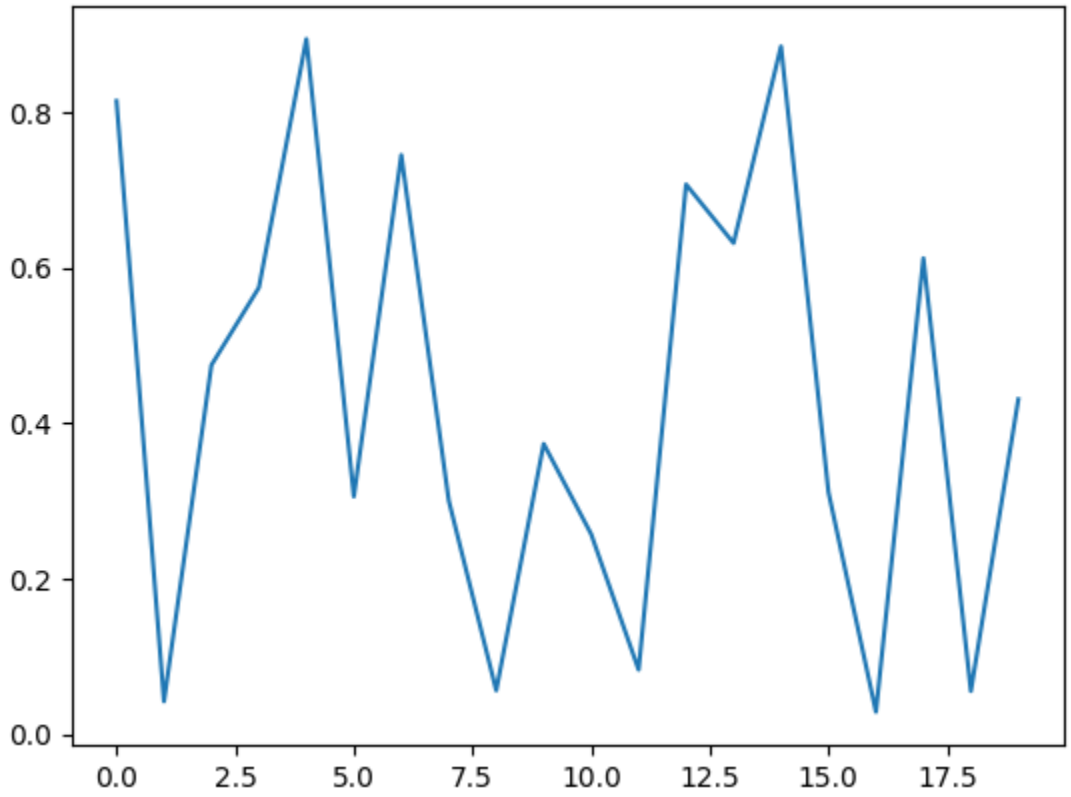
OUTPUT:



```
x  = np.random.random(20)
print(x)
plt.plot(x)
plt.show()
```

Output:

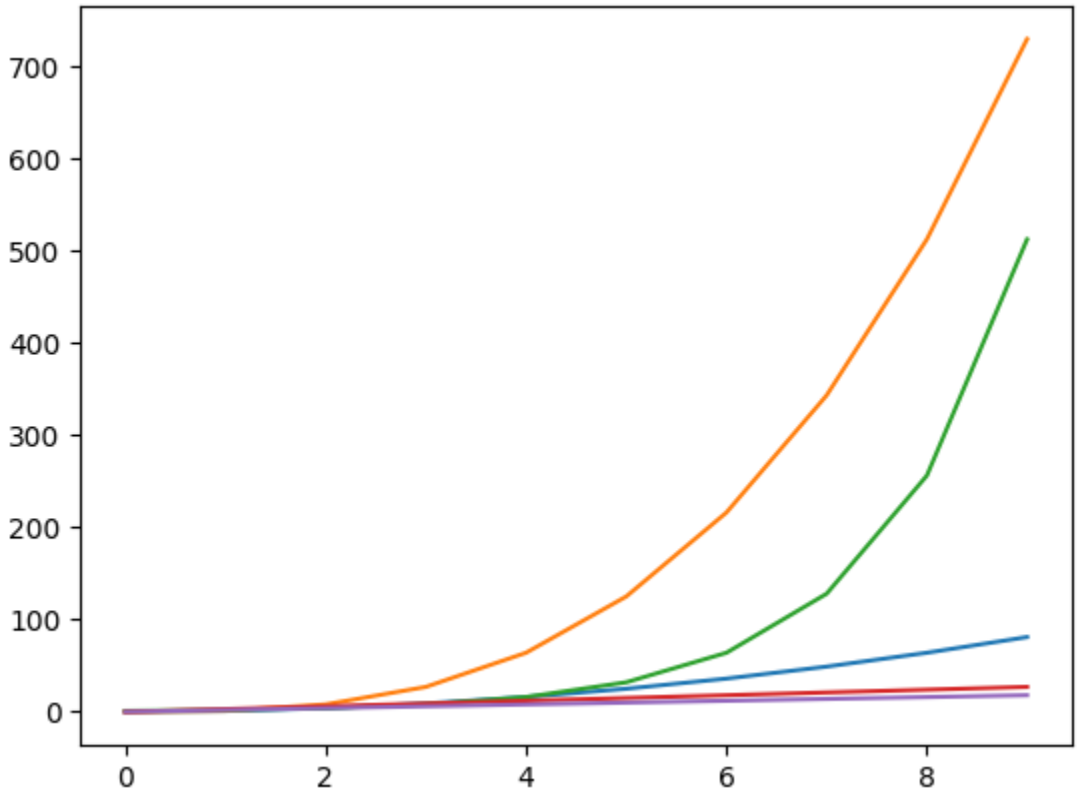
[0.81405255 0.0422034 0.4745679 0.57424733 0.89369256 0.30558337
0.74460838 0.3003304 0.0562306 0.37334593 0.25724118 0.08284745
0.70680878 0.63138551 0.88433257 0.31103243 0.02880032 0.61188153
0.05545284 0.43062125]



Multi line plotting

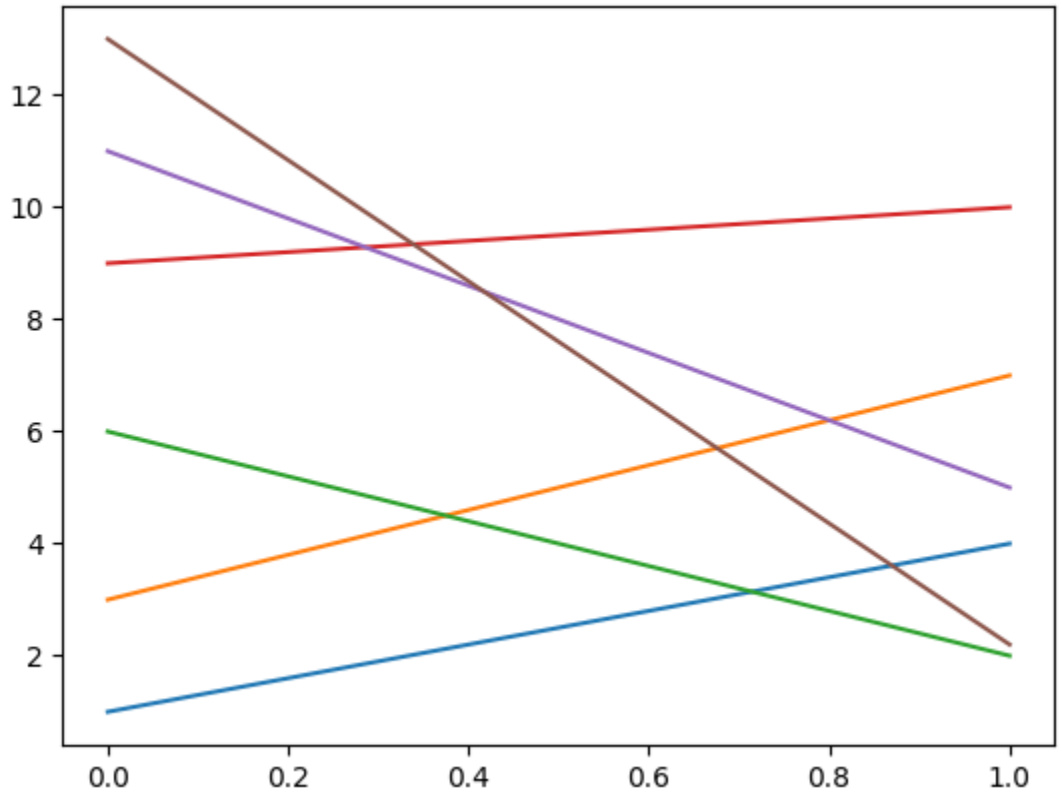
```
x = np.arange(10)
print(x)

plt.plot(x, x**2)
plt.plot(x, x**3)
plt.plot(x, 2**x)
plt.plot(x, 3*x)
plt.plot(x, x*2)
plt.show()
```

OUTPUT:



```
x = np.array([[1, 3, 6, 9, 11, 13], [4, 7, 2, 10, 5, 2.2]])
plt.plot(x)
plt.show()
```

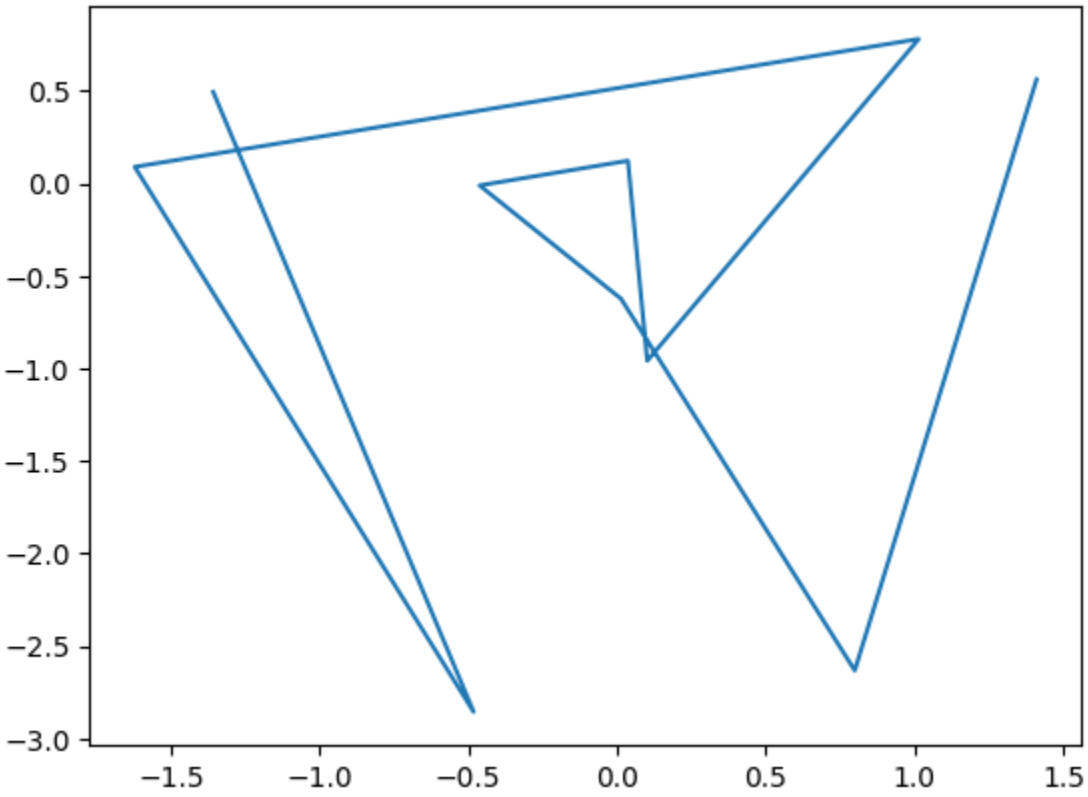
OUTPUT:



```
data = np.random.randn(2, 10)
print(data)
plt.plot(data[0], data[1])
plt.show()
```

OUTPUT:

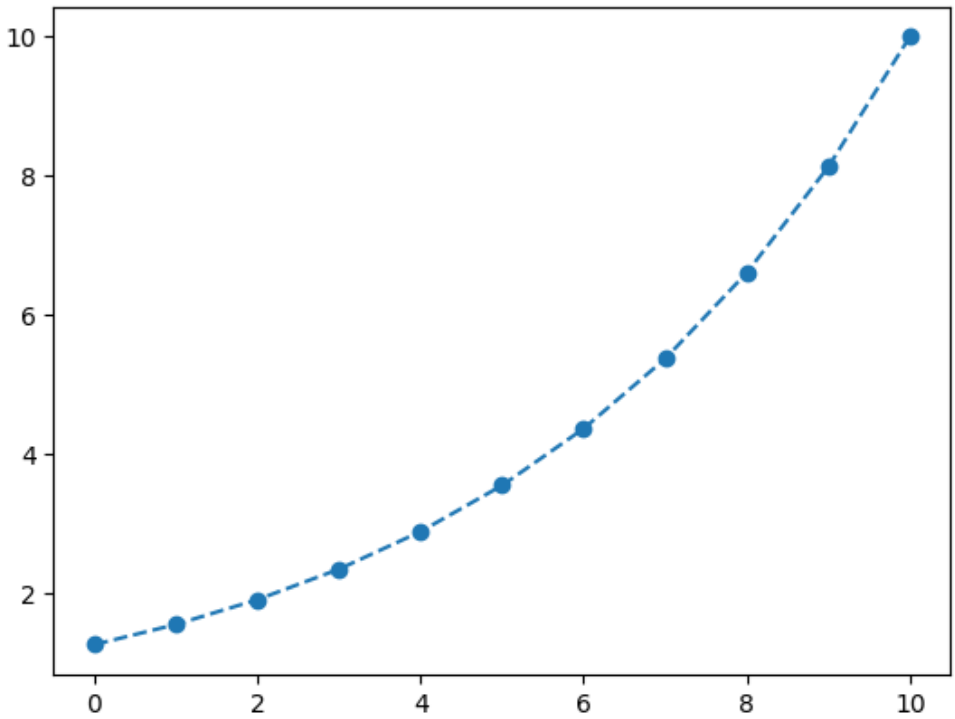
```
[[-1.35710153 -0.48159038 -1.62091397 1.01505753 0.10241209 0.037349
-0.46147589 0.01415971 0.80009676 1.41154965] [ 0.4950268 -2.85649624
0.08980429 0.78113453 -0.9584716 0.12304158 -0.01056969 -0.62303254
-2.63295351 0.56319273]]
```



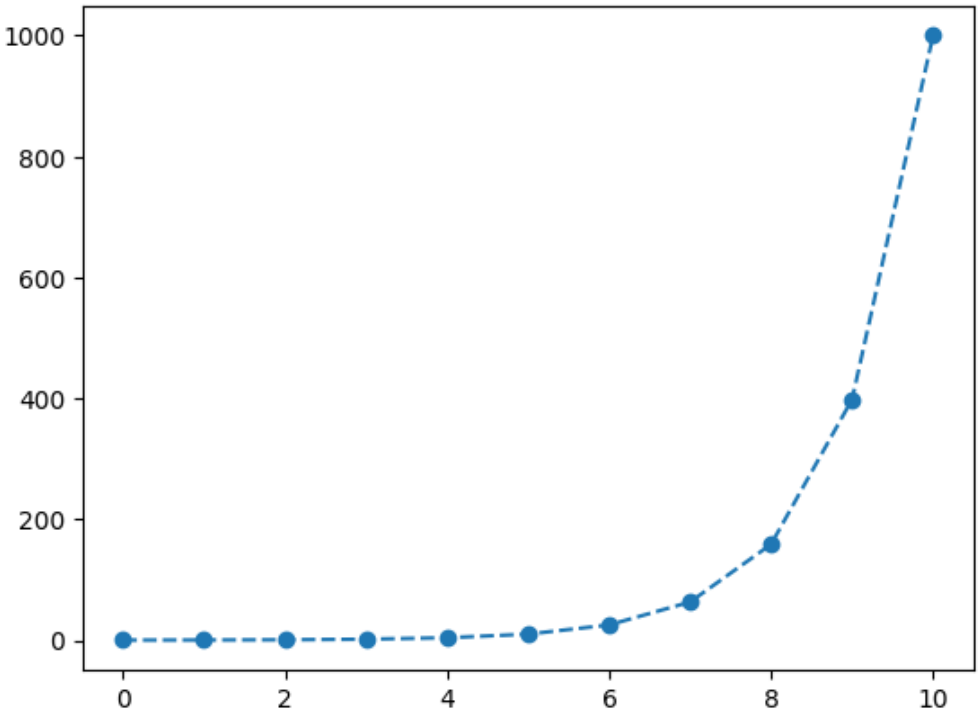
```
N = 11
x = np.linspace(0, 10, N)
print(x)
y = np.logspace(0.1, 1, N)
print(y)
plt.plot(x, y, 'o--')
plt.show()
y = np.geomspace(0.1, 1000, N)
print(y)
plt.plot(x, y, 'o--')
plt.show()
```

OUTPUT:

[0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.] [1.25892541 1.54881662 1.90546072 2.34422882 2.8840315 3.54813389 4.36515832 5.37031796 6.60693448 8.12830516]



[1.00000000e-01 2.51188643e-01 6.30957344e-01 1.58489319e+00
3.98107171e+00 1.00000000e+01 2.51188643e+01 6.30957344e+01
1.58489319e+02 3.98107171e+02 1.00000000e+03]



34. Normal distribution to evaluate fitness of data

Aim:

To create a program of normal distribution to evaluate fitness of the data in python using google colab in chrome.

Algorithm:

Step 1: Open the new google colab file.

Step 2 : Import pandas , numpy and seaborn and matplotlib package.

Step 3: Mount the drive file of winequality-red.csv file.

Step 4: Filtering numerical data, print subplot ,df.describe.

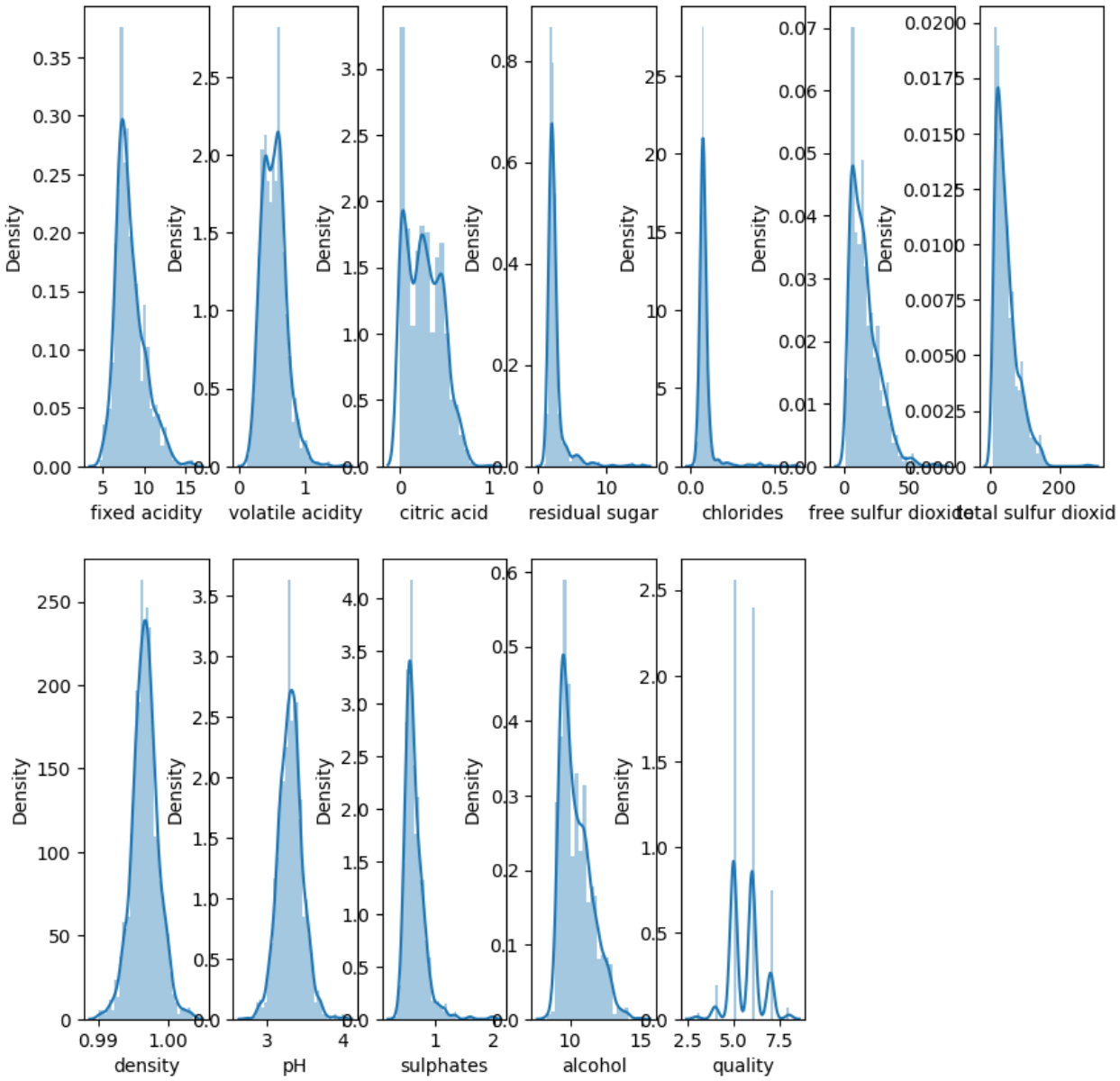
Step 5: Plot the graph

Step 6: Run the file and get output

Normal distribution to evaluate fitness of the data

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#df=pd.read_csv('mobile_train.csv')
df=pd.read_csv('/content/drive/MyDrive/winequality-red.csv')
#df = pd.read_csv('winequality-white1.csv')

df.describe()
#Filtering numerical data
df_numerics_only = df.select_dtypes(include=[np.number])
df_numerics_only
print(df_numerics_only.head())
df3=df_numerics_only.columns.values
print(len(df3))
fig=plt.figure(figsize=(10,10))
rows=2
cols=int(np.absolute(len(df3)/rows)+1)
#probability distribution plot
for i in range(len(df3)):
   plt.subplot(rows,cols,i+1)
   sns.distplot(df[df3[i]],kde=True)

plt.show()
```

Output:


Density
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxid
density
pH
sulphates
alcohol
quality