List of Programs:

1. Download and install R-Programming environment and install basic packages using install. Packages () command in R.
2. Learn all the basics of R-Programming (Data types ,Variables , Operators ,Control statements
3. Write a R program to create a two-dimensional 5x3 array of sequence of even integers greater than 50.
4. a)Create a Scatter plot from CSV in R

   b) Create a Json Files
5. Implement data frames in R. Write a program to join columns and rows in a data frame using c bind () and r bind () in R.
6. a)Write a R program to find factorial of a number using recursion

   b)Write a R program to mean, variance, standard deviation for the given probability distribution
7. Implement Apriori algorithm to extract association rule of datamining.
8. Implement k-means clustering technique.
9. Implement Classification algorithm.
10. Create pie charts and bar charts using R.

Introduction to R programming:

R is a programming language and free software developed by Ross Ihaka and Robert Gentleman in 1993. R possesses an extensive catalog of statistical and graphical methods. It includes machine learning algorithms, linear regression, time series, statistical inference to name a few. Most of the R libraries are written in R, but for heavy computational tasks, C, C++ and Fortran codes are preferred. R is not only entrusted by academic, but many large companies also use R programming language, including Uber, Google, Airbnb, Facebook and so on.

Data analysis with R is done in a series of steps; programming, transforming, discovering, modeling and communicate the results.

Program: R is a clear and accessible programming tool

Transform: R is made up of a collection of libraries designed specifically for data science

Discover: Investigate the data, refine your hypothesis and analyze them

Model: R provides a wide array of tools to capture the right model for your data

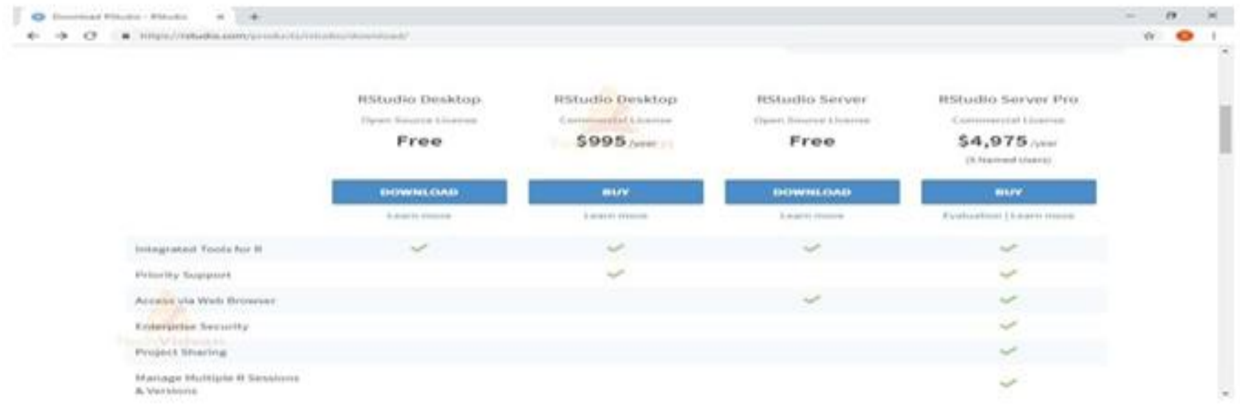
Communicate: Integrate codes, graphs, and outputs to a report with R Markdown or build Shiny apps to share with the world

What is R used for?

- Statistical inference
- Data analysis
- Machine learning algorithm

Installation of R-Studio on windows:

Step – 1: With R-base installed, let’s move on to installing RStudio. To begin, go to download RStudio and click on the download button for RStudio desktop.

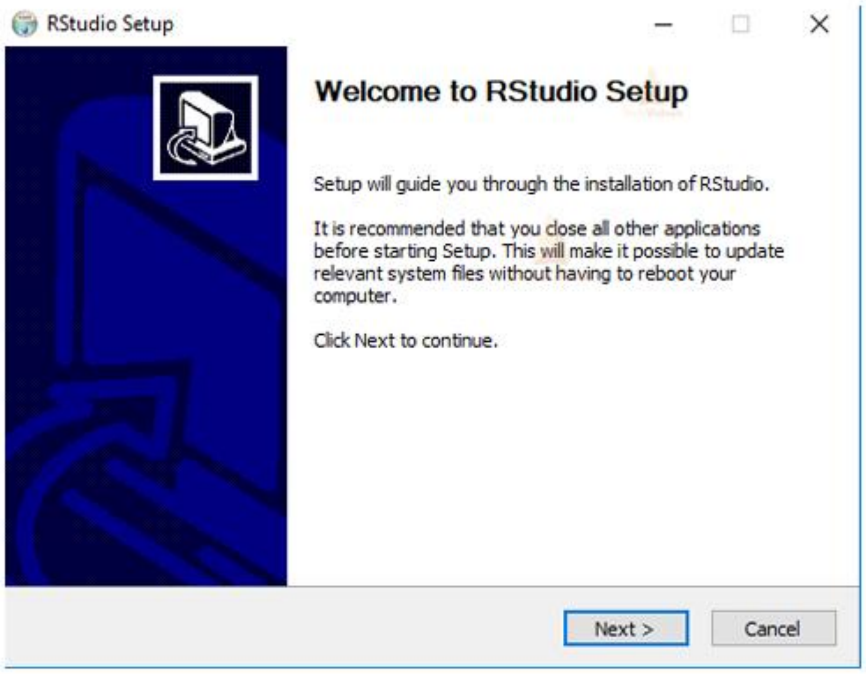


	RStudio Desktop	RStudio Desktop	RStudio Server	RStudio Server Pro
	Open Source License	Commercial License	Open Source License	Commercial License
	Free	$995 /year	Free	$4,975 /year (5 Named Users)
	DOWNLOAD	BUY	DOWNLOAD	BUY
	Learn more	Learn more	Learn more	Evaluation | Learn more
Integrated Tools for R	✓	✓	✓	✓
Priority Support		✓		✓
Access via Web Browser			✓	✓
Enterprise Security				✓
Project Sharing				✓
Manage Multiple R Sessions & Versions				✓

Step – 2: Click on the link for the windows version of RStudio and save the .exe file.

Step – 3: Run the .exe and follow the installation instructions.

3. Click Next on the welcome window.


RStudio Setup

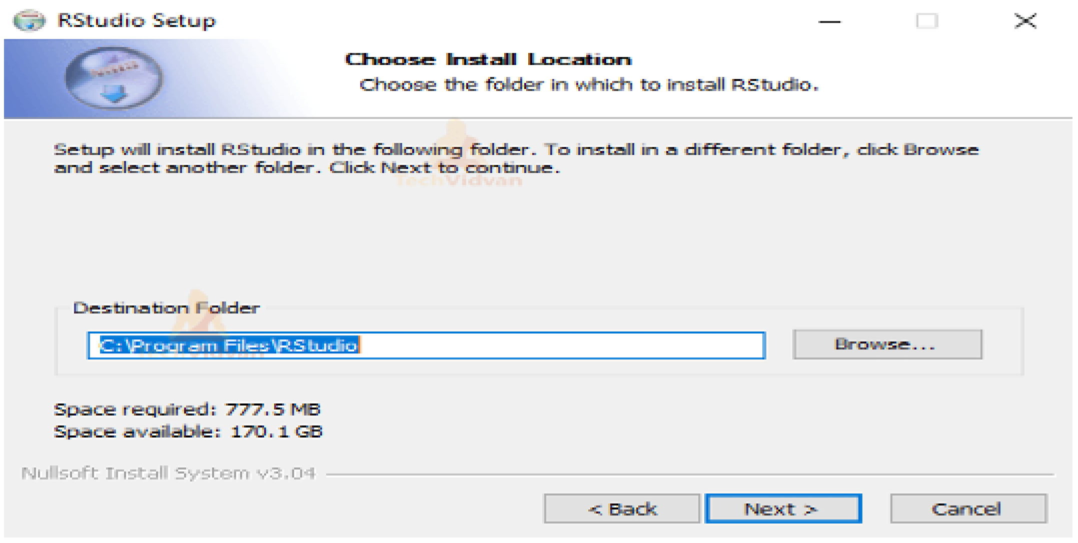
Welcome to RStudio Setup

Setup will guide you through the installation of RStudio.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Enter/browse the path to the installation folder and click Next to proceed.


RStudio Setup

Choose Install Location

Choose the folder in which to install RStudio.

Setup will install RStudio in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

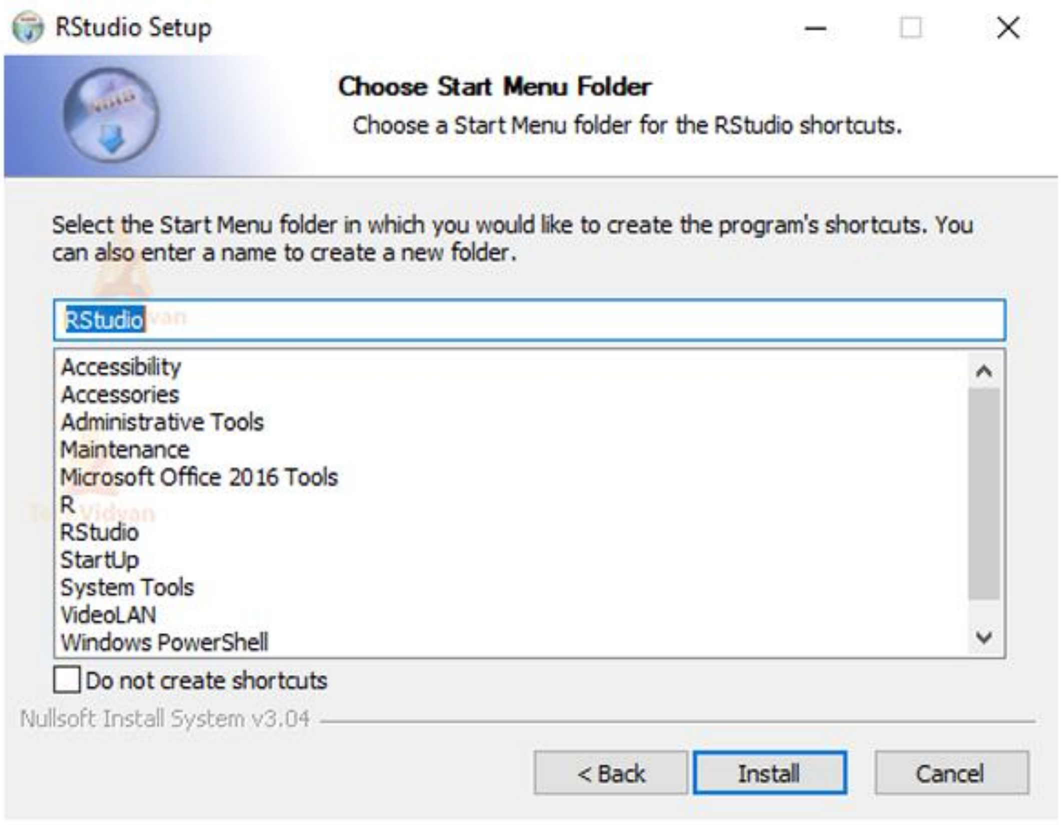
Destination Folder

C:\Program Files\RStudio

Space required: 777.5 MB
Space available: 170.1 GB

Nullsoft Install System v3.04

Select the folder for the start menu shortcut or click on do not create shortcuts and then click Next.


RStudio Setup

Choose Start Menu Folder

Choose a Start Menu folder for the RStudio shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

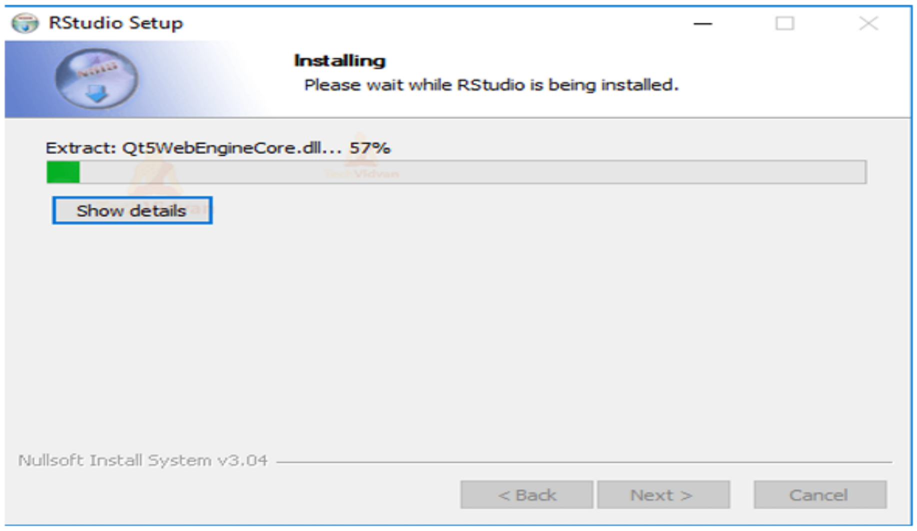
RStudio

Accessibility
Accessories
Administrative Tools
Maintenance
Microsoft Office 2016 Tools
R
RStudio
StartUp
System Tools
VideoLAN
Windows PowerShell

☐ Do not create shortcuts

Nullsoft Install System v3.04

Wait for the installation process to complete.


RStudio Setup

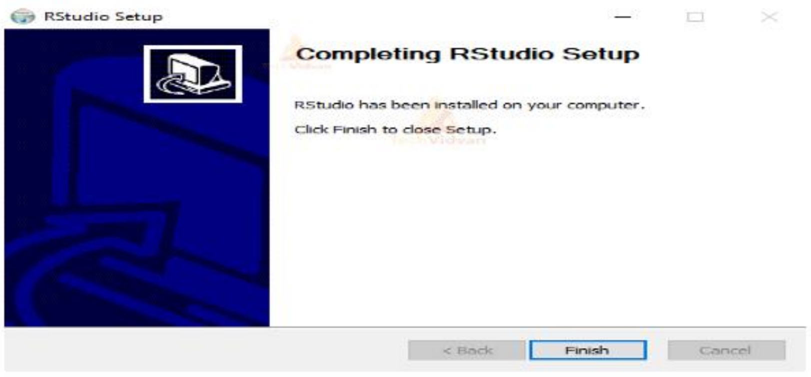
Installing

Please wait while RStudio is being installed.

Extract: Qt5WebEngineCore.dll... 57%

Nullsoft Install System v3.04

Click Finish to end the installation.



RStudio has been installed on your computer.

Click Finish to close Setup.

Install the R Packages:-

In RStudio, if you require a particular library, then you can go through the following instructions:

First, run R Studio.

After clicking on the packages tab, click on install. The following dialog box will appear.

In the Install Packages dialog, write the package name you want to install under the Packages field and then click install. This will install the package you searched for or give you a list of matching packages based on your package text.

2.LEARN ALL THE BASICS OF R PROGRAMMING (Data Types, variables, operators, control statements)

Aim

The aim of the code is to demonstrate the use of basic data structures in R programming, such as vectors, lists, data frames, and matrices.

Algorithm

1. **Create Vectors**:
    - Create a numeric vector int_vector containing integers.
    - Create a character vector char_vector containing strings (fruit names).
2. **Create a List**:
    - Create a list my_list which contains the previously created vectors (int_vector and char_vector), a numeric value (3.14), and a boolean value (TRUE).
3. **Access Elements in the List**:
    - Access the second element of the list (the character vector char_vector) and print it.
4. **Create a Data Frame**:
    - Create a data frame data containing three columns: Name, Age, and Score.
    - Populate the columns with appropriate data (names, ages, and scores).
5. **Access Columns in the Data Frame**:
    - Access the Name column of the data frame using the $ operator and print it.
6. **Create a Matrix**:
    - Create a matrix my_matrix with 2 rows and 3 columns, containing the numbers 1 to 6.
7. **Access Elements in the Matrix**:
    - Access the element located at row 1, column 2 in the matrix and print it.
8. **Operations on Data Structures**:
    - **Vector Operation**: Calculate the sum of the integer vector int_vector and print the result.
    - **Matrix Operation**: Compute the transpose of the matrix my_matrix and print the transposed matrix.

9. **Add a New Column to the Data Frame**:
    - Add a new column City to the data data frame, assigning each row a city name.
    - Print the updated data frame to show the new column.

A.PROGRAM:

```
int_vector<-c(1,2,3,4,5)
print("Integer vector:")
print(int_vector)
char_vector<-c("apple","banana","Cherry")
print('character vector:')
print(char_vector)
my_list<-list(int_vector,char_vector,3.14,TRUE)
print("list:")
print(my_list)
print("Second element of the list(char_vector):")
print(my_list[[2]])
data<-data.frame(
  Name=c("john","mary","steve"),
  Age=c(23,25,30),
  Score=c(88.5,91.3,85.0)
)
print("Data Frame:")
print(data)
print("Accessing the 'Name'column of the data frame:")
print(data$Name)
my_matrix<-matrix(1:6,nrow=2,ncol=3)
print("matrix:")
print(my_matrix)
print("Element at row1,column2 of the matrix:")
print(my_matrix[1,2])
sum_vector<-sum(int_vector)
print("sum of the integer vector:")
print(sum_vector)
matrix_transpose<-t(my_matrix)
print("Transpose of matrix:")
print(matrix_transpose)
data$city<-c("New york","Los angeles","chicago")
print("updated data frame with 'city' column:")
print(data)
```

R-OPERATORS:

```
A<-matrix(c(1,2,5,7,2,4,6,8),nrow=4,ncol=2)
B<-matrix(c(9,10,11,12,13,14,15,16),nrow=2,ncol=4)
cat("matrix A:\n")
print(A)
cat("\n matrix B:\n")
print(B)
cat("\n matrix Multiplication of A and B(%*%):\n")
result_matrix_multiply<-A%*%B
print(result_matrix_multiply)
df<-data.frame(
  ID=c(1,2,3,4,5),
  Age=c(25,30,22,35,28),
  Salary=c(50000,60000,45000,70000,52000),
  Department=c("HR","IT","Finance","IT","HR")
```

```
)
cat("\n Data Frame:\n")
print(df)
filtered_data<-df[df$Age>=30&df$salary>=50000,]
cat("\n filtered Data(Age>=30 and salaRY>=55000):\n")
print(filtered_data)
is_high_salary<-df$salary>55000
is_in_IT<-df$Department=="IT"
cat("\n Salary>55000(high salary):\n")
print(is_high_salary)
cat("\n Department is IT:\n")
print(is_in_IT)
cat("\n High salary AND in IT Department:\n")
print(is_high_salary&is_in_IT)
cat("\n High salary OR in IT Department:\n")
print(is_high_salary|is_in_IT)
vector1<-c(5,10,15,20)
vector2<-c(2,4,6,8)
cat("\n Element-wise Addition of vectors:\n")
print(vector1+vector2)
cat("\n Element-wise multiplication of vectors:\n")
print(vector1*vector2)
cat("\n Advanced sequence generation and modulus operations:\n")
for (i in 1:15){
  seq_val<-i*3
  modulus_val<-seq_val%%4
  cat("multiple of 3:",seq_val,",modulus with 4:",modulus_val,"\n")
```

```
}
x<-5
y<-10
z<-20
cat("\n Assignment with nested Expressions:\n")
result<-(x<-x+2)*(y<-y-3)+(z<-z/2)
cat("final result with complex assignment and expression:",result,"\n")
fruits<-c("apple","banana","cherry","data")
search_fruits<-c("banana","fig","data")
cat("\n check if search fruits are in the fruit list(%in%):\n")
print(search_fruits %in% fruits)
na_vector<-c(1,NA,3,4,NA)
cat("\n Relational operators with NA-safe comparisons:\n")
cat("Is NA equal to 3:",NA==3,"\n")
cat("Is NA not equal to 3:",NA!=3,"\n")
cat("Is NA less than 3:",NA<3,"\n")
cat("\n checking NA values using is.na():\n")
print(is.na(na_vector))
matrix_result<-(A*2)> 10
cat("\n complex matrix Element-wise operations with relational operations:\n")
print(matrix_result)
```

CONTROL-STATEMENT:

```
x<-10
if(x>5){
  print("x is greater than 5")
}else{
  print("x is less than or equal to 5")
}
y<-7
if(y>10){
  print("y is greater than 10")
}else if(y==7){
  print("y is equal to 7")
}else{
  print("y is less than 7")
}
for(i in 1:5){
  print(paste("current value of i:",i))
}
count<-1
while(count<=3){
  print(paste("current count is:",count))
  count<-count+1
}
day<-3
result<-switch(day,
               "sunday"="weekend",
               "monday"="weekday",
               "tuesday"="weekday",
               "wednesday"="weekday",
               "thursday"="weekday",
               "friday"="weekday",
               "saturday"="weekend",
               "Invalid day")
print(paste("The day is:",result))
```

OUTPUT:

```
 [1] "Integer vector:"
 [1] 1 2 3 4 5
 [1] "character vector:"
 [1] "apple"  "banana" "Cherry"
 [1] "list:"
 [[1]]
[1] 1 2 3 4 5
[[2]]
[1] "apple"  "banana" "Cherry"
[[3]]
[1] 3.14
[[4]]
[1] TRUE
 [1] "Second element of the list(char_vector):"
 [1] "apple"  "banana" "Cherry"
 [1] "Data Frame:"
   Name Age Score
1  john  23  88.5
2  mary  25  91.3
3 steve  30  85.0
 [1] "Accessing the 'Name'column of the data frame:"
 [1] "john"  "mary"  "steve"
 [1] "matrix:"
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
 [1] "Element at row1,column2 of the matrix:"
 [1] 3
 [1] "sum of the integer vector:"
 [1] 15
 [1] "Transpose of matrix:"
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
 [1] "updated data frame with 'city' column:"
   Name Age Score        city
1  john  23  88.5    New york
2  mary  25  91.3 Los angeles
```

```
3 steve  30  85.0     chicago
matrix A:
     [,1] [,2]
[1,]    1    2
[2,]    2    4
[3,]    5    6
[4,]    7    8
 matrix B:
     [,1] [,2] [,3] [,4]
[1,]    9   11   13   15
[2,]   10   12   14   16
 matrix Multiplication of A and B(%*%):
     [,1] [,2] [,3] [,4]
[1,]   29   35   41   47
[2,]   58   70   82   94
[3,]  105  127  149  171
[4,]  143  173  203  233

Data Frame:
  ID Age Salary Department
1  1  25  50000         HR
2  2  30  60000         IT
3  3  22  45000    Finance
4  4  35  70000         IT
5  5  28  52000         HR
filtered Data(Age>=30 and salaRY>=55000):
 [1] ID         Age        Salary     Department
<0 rows> (or 0-length row.names)
Salary>55000(high salary):
logical(0)
 Department is IT:
 [1] FALSE  TRUE FALSE  TRUE FALSE
 High salary AND in IT Department:
logical(0)
 High salary OR in IT Department:
logical(0)
 Element-wise Addition of vectors:
 [1]  7 14 21 28
 Element-wise multiplication of vectors:
 [1]  10  40  90 160
 Advanced sequence generation and modulus operations:
multiple of 3: 3 ,modulus with 4: 3
multiple of 3: 6 ,modulus with 4: 2
multiple of 3: 9 ,modulus with 4: 1
multiple of 3: 12 ,modulus with 4: 0
multiple of 3: 15 ,modulus with 4: 3
multiple of 3: 18 ,modulus with 4: 2
multiple of 3: 21 ,modulus with 4: 1
multiple of 3: 24 ,modulus with 4: 0
multiple of 3: 27 ,modulus with 4: 3
multiple of 3: 30 ,modulus with 4: 2
multiple of 3: 33 ,modulus with 4: 1
multiple of 3: 36 ,modulus with 4: 0
multiple of 3: 39 ,modulus with 4: 3
multiple of 3: 42 ,modulus with 4: 2
multiple of 3: 45 ,modulus with 4: 1
 Assignment with nested Expressions:
final result with complex assignment and expression: 59
 check if search fruits are in the fruit list(%in%):
 [1]  TRUE FALSE  TRUE
 Relational operators with NA-safe comparisons:
Is NA equal to 3: NA
Is NA not equal to 3: NA
Is NA less than 3: NA
 checking NA values using is.na():
 [1] FALSE  TRUE FALSE FALSE  TRUE
 complex matrix Element-wise operations with relational operations:
      [,1]  [,2]
[1,] FALSE FALSE
[2,] FALSE FALSE
[3,] FALSE  TRUE
[4,]  TRUE  TRUE
 [1] "x is greater than 5"
 [1] "y is equal to 7"
 [1] "current value of i: 1"
[1] "current value of i: 2"
[1] "current value of i: 3"
[1] "current value of i: 4"
[1] "current value of i: 5"
```

```
 [1] "current count is: 1"
[1] "current count is: 2"
[1] "current count is: 3"
 [1] "The day is: weekday"
```

B.PROGRAM

```
num_var<-42
char_var<-"Hello"
log_var<-TRUE
vec_var<-c(1,2,3,4,5)
cat("Numeric variable:",num_var,"\n")
cat("Character variable:",char_var,"\n")
cat("Logical variable:",log_var,"\n")
cat("Vector variable:",vec_var,"\n\n")
x<-10
y<-5
cat("Addition :",x+y,"\n")
cat("Greater than comparison:",x>y,"\n")
cat("Logical AND(x>5 AND y>5):",x>5 && y>5,"\n\n")
if(x>y)
{
  cat("x is greater than y \n")
} else {
  cat("x is not greater than y \n")
}
cat("\n using for loop:\n")
for(i in vec_var)
{
  cat("value:",i,"\n")
}
cat("\n using while loop:\n")
i<-1
while(i<=2)
{
  cat("Iteration:",i,"\n")
  i<-i+1
}
```

OUTPUT:

```
Numeric variable: 42
Character variable: Hello
Logical variable: TRUE
Vector variable: 1 2 3 4 5
Addition : 15
Greater than comparison: TRUE
Logical AND(x>5 AND y>5): FALSE
x is greater than y
value: 1
value: 2
value: 3
value: 4
value: 5
Iteration: 1
Iteration: 2
```

3.To Create a two-dimensional 5x3 array of sequence of even integers greater than 50

Aim :

To create a R program to generate and display a two-dimensional array (5x3) filled with even integ ers, each of which is greater than 50.

Algorithm:

1. Initialize an empty 5x3 array.
2. Start with the first even integer greater than 50 (which is 52).
3. Iterate and place the next even integers (increment by 2) in the array until the array is filled.

A.PROGRAM

```
a<-array(seq(from=50,length.out=15,by=2),c(5,3))
print("content of the array")
print("5*3 array of sequence of even integer greater than 50:")
print(a)
```

OUTPUT:

```
[1] "content of the array"
 [1] "5*3 array of sequence of even integer greater than 50:"
     [,1] [,2] [,3]
[1,]   50   60   70
[2,]   52   62   72
[3,]   54   64   74
[4,]   56   66   76
[5,]   58   68   78
```

B.PROGRAM:

```
x<-array(seq(from=100,length.out=16,by=13),c(4,4))
print("Content of the array")
print("4*4 array of sequence of even integer greater than 100:")
print(x)
```

OUTPUT:

```
[1] "Content of the array"
 [1] "4*4 array of sequence of even integer greater than 100:"
     [,1] [,2] [,3] [,4]
[1,]  100  152  204  256
[2,]  113  165  217  269
[3,]  126  178  230  282
[4,]  139  191  243  295
```

4. Scatter Plot from CSV in R

Aim:

To create a scatter plot using data from a CSV file in R programming. The program will read the data from a CSV file, extract the relevant variables for plotting, and generate a scatter plot to visually represent the relationship between two numerical variables.

Algorithm:

1. Load Required Libraries: Use necessary libraries such as ggplot2 for visualization and readr or utils to read the CSV file.
2. Read the CSV File: Import the data from a CSV file into a data frame.
3. Extract Data: Identify and extract the two columns of numerical data that you want to plot as the X and Y axes.
4. Plot the Data: Use the ggplot() function or base plotting methods in R to generate a scatter plot.
5. Customize the Plot: Add labels, titles, and customize the appearance of the plot.
6. Display the Plot: Display the scatter plot to the user.

4A1.PROGRAM:

```
data<-read.csv("D:/sample R/record.csv")
print(data)
print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
max_sal<-max(data$SALARY)
print(max_sal)
details<-subset(data,DEPARTMENT=="IT")
print(details)
details<-subset(data,DEPARTMENT=="HR"&SALARY>600)
print(details)
details1<-subset(data,as.Date(START_DATE)>as.Date("02/06/2024"))
print(details1)
details<-subset(data,as.Date(START_DATE)>as.Date("25/06/2023"))
write.csv(details,"output.csv")
new_details<-read.csv("output.csv")
print(new_details)
plot(x=data$YEAR.OF.EXPERIENCE,
    y=data$SALARY,
    xlim=c(0,20),
    xlab="NAME",
    ylab="SALARY",
    col="red",
  main="Sactter plot")
```

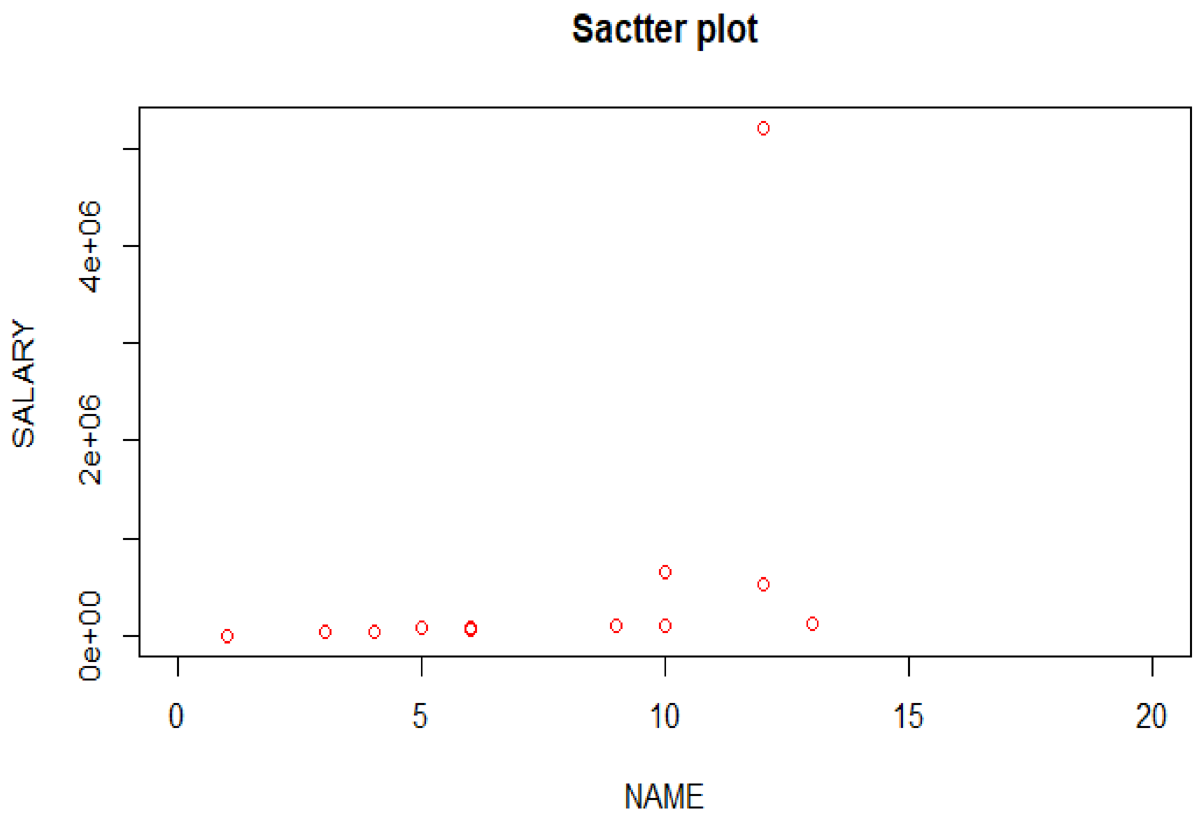
<u>Data set:</u>

ID	NAME	SALARY	START_DATE	DEPARTMENT	YEAR OF EXPERIENCE
1	AARAV	80000	25-11-2023	IT	5
2	ANANYA	45000	10/5/2023	OPERATOR	3
3	ISHAAN	650000	7/8/2024	HR	10
4	PRIYA	56000	1/9/2024	MANAGER	6
5	RAHUL	90000	11/11/2024	DOCTOR	5
6	SAANVI	50000	5/8/2023	SI	4
7	ARJUN	85000	12/12/2023	PROGRAMMER	5
8	DIYA	45000	4/7/2024	ASST.PROFESSOR	4
9	VISHAL	100000	17-08-2024	DIRECTOR	9
10	ADITI	520000	15-04-2023	DEAN	12
11	RAVI	120000	2/6/2024	IT	13
12	NEHA	45000	3/9/2023	OPERATOR	3
13	KARAN	650000	14-11-2023	HR	10
14	SIMRAN	56000	8/2/2023	MANAGER	6
15	RITIKA	90000	6/8/2024	DOCTOR	5
16	SIVANI	50000	8/12/2023	SI	4
17	ARUN	85000	21-06-2024	PROGRAMMER	6
18	SIVARAJ	4500	25-11-2023	ASST.PROFESSOR	1
19	RAJU	100000	10/8/2023	DIRECTOR	10
20	ARJUN	5200000	7/8/2024	DEAN	12

OUTPUT:

```
   ID    NAME     SALARY     START_DATE     DEPARTMENT      YEAR.OF.EXPERIENCE
1    1     AARAV      80000            25-11-2023              IT                                5
2    2    ANANYA    45000           10/5/2023          OPERATOR                          3
3   3     ISHAAN      650000          7/8/2024                HR                              10
4   4      PRIYA       56000           1/9/2024           MANAGER                          6
5   5      RAHUL      90000           11/11/2024          DOCTOR                        5
6   6     SAANVI     50000            5/8/2023                 SI                              4
7   7     ARJUN       85000           12/12/2023        PROGRAMMER                  5
8   8      DIYA        45000             4/7/2024        ASST.PROFESSOR              4
9   9     VISHAL     100000           17-08-2024           DIRECTOR                    9
10 10     ADITI       520000          15-04-2023             DEAN                        12
11 11      RAVI        120000           2/6/2024                IT                          13
12 12    NEHA         45000           3/9/2023           OPERATOR                      3
13 13    KARAN       650000          14-11-2023             HR                          10
14 14     SIMRAN      56000           8/2/2023           MANAGER                    6
15 15     RITIKA       90000           6/8/2024             DOCTOR                    5
16 16     SIVANI       50000            8/12/2023               SI                      4
17 17    ARUN           85000            21-06-2024     PROGRAMMER                6
18 18    SIVARAJ       4500              25-11-2023    ASST.PROFESSOR              1
19 19    RAJU           100000            10/8/2023          DIRECTOR                  10
20 20   ARJUN         5200000          7/8/2024              DEAN                       12

[1] TRUE

[1] 6

[1] 20

[1] 5200000

   ID  NAME     SALARY     START_DATE     DEPARTMENT  YEAR.OF.EXPERIENCE
1   1   AARAV   80000            25-11-2023               IT                          5
11 11  RAVI     120000            2/6/2024                  IT                        13
   ID   NAME     SALARY     START_DATE     DEPARTMENT   YEAR.OF.EXPERIENCE
3   3   ISHAAN  650000           7/8/2024                  HR                     10
13 13  KARAN  650000          14-11-2023              HR                     10
   ID     NAME    SALARY    START_DATE    DEPARTMENT    YEAR.OF.EXPERIENCE
1   1     AARAV    80000            25-11-2023              IT                           5
```

```
9   9     VISHAL    100000         17-08-2024       DIRECTOR               9
10 10    ADITI      520000          15-04-2023          DEAN                  12
13 13     KARAN   650000          14-11-2023           HR                     10
17 17     ARUN       85000          21-06-2024       PROGRAMMER          6
18 18    SIVARAJ   4500             25-11-2023     ASST.PROFESSOR        1
  X ID    NAME    SALARY    START_DATE    DEPARTMENT     YEAR.OF.EXPERIENCE
1    1    AARAV      80000        25-11-2023              IT                         5
2   18    SIVARAJ     4500        25-11-2023      ASST.PROFESSOR         1
```



4 1) B) PROGRAM:

PROGRAM:

```
data<-read.csv("D:/sample R/students_data.csv")

print(data)

file_path<-"D:/sample R/students_data.csv"

students_data<-read.csv("D:/sample R/students_data.csv")

head(students_data)

str(students_data)

summary(students_data)
```

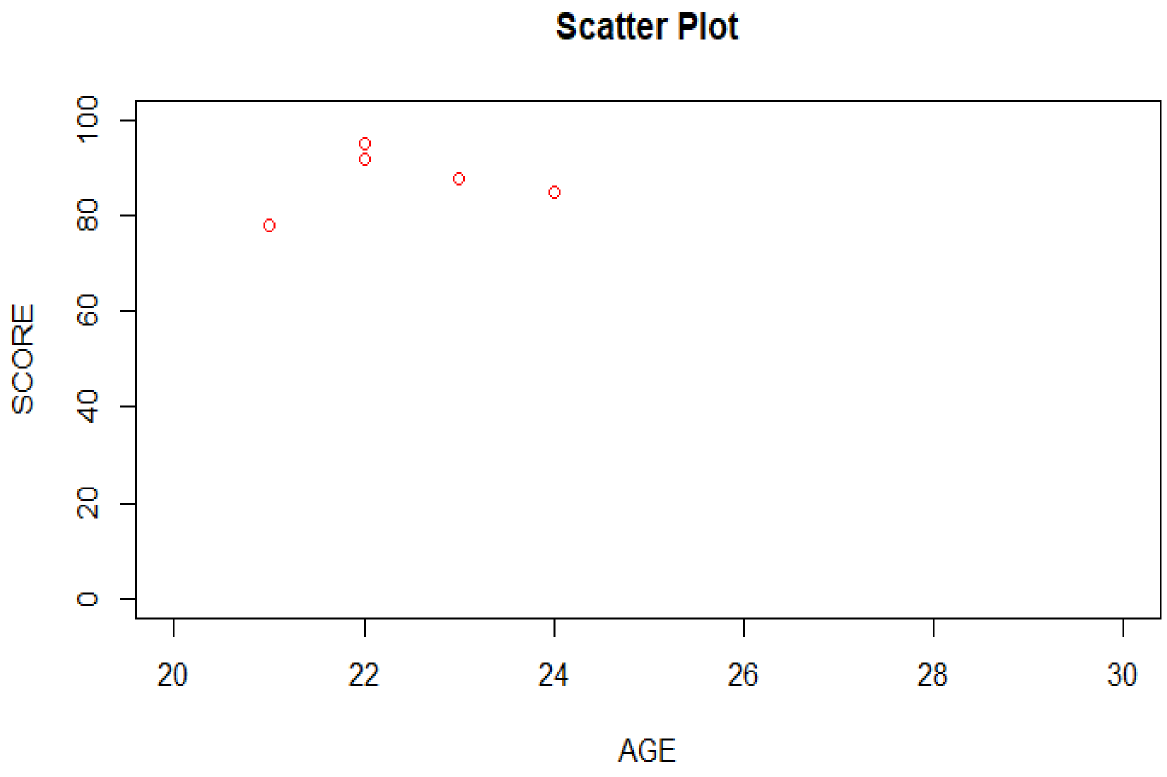
```
NAME    AGE      SCORE
Alice          23         88
Bob            22         95
Charlie        21         78
David          24         85
Eve            22         92
```

OUTPUT:

```
    NAME    AGE    SCORE
1     Alice      23        88
2      Bob       22         95
3    Charlie      21        78
4     David       24         85
5       Eve       22        92
```

```
     NAME        AGE         SCORE
1     Alice            23              88
2      Bob             22              95
3    Charlie           21             78
4      David           24             85
5       Eve             22             92
```

```
'data.frame':  5 obs. of  3 variables:
 $ NAME :  chr  "Alice"  "Bob"  "Charlie"  "David"     ...
 $ AGE  :     int       23       22          21           24         22
 $ SCORE:  int       88       95          78           85         92

     NAME               AGE                SCORE
 Length:5             Min.   :21.0        Min.   :78.0
 Class :character   1st Qu.:22.0       1st Qu.:85.0
 Mode  :character   Median :22.0      Median :88.0
                      Mean   :22.4        Mean   :87.6
                      3rd Qu.:23.0        3rd Qu.:92.0
                      Max.   :24.0        Max.   :95.0
```



4 b. JSON FILE

Aim:

To demonstrate how to handle JSON (JavaScript Object Notation) data in R programming. JSON is a widely used data format for representing structured data, commonly used in APIs and web applications. In this program, we will explore how to read, write, and manipulate JSON data using R.

Algorithm:

1. **Install and Load the JSON Package**:
   - First, ensure that the jsonlite package is installed and loaded.
2. **Reading JSON Data**:
   - Use the fromJSON() function to parse JSON data from a string or a file into an R object (e.g., a list, data frame).
3. **Writing JSON Data**:
   - Use the toJSON() function to convert an R object into JSON format, and optionally write it to a file.
4. **Manipulating JSON Data**:
   - Once the JSON data is read into an R object (typically a list or data frame), it can be manipulated as needed (e.g., adding/removing elements, modifying values).
5. **Saving JSON Data**:
   - After making modifications, the R object can be converted back into JSON format and saved to a file.

JSON CODE:

```
{
"ID":["1","2","3","4","5"],
"NAME":["sivaranjani","snegha","kavya","lakshmi","soniya"],
"SALARY":["722.5","815.2","1611","2829","843.25"],
"STARTDATE":["6/17/2014","1/1/2012","11/15/2024","9/23/2022","10/12/2023"],
"DEPT":["IT","IT","HR","OPERATION","FINANCE"]
```

}

A.PROGRAM:

```
Library(“rjson”)
Result<-fromJSON(file=”D:/sample R/j2.json”)
Print(result)
```

OUTPUT:

```
ID
[1] “1","2","3","4","5”
NAME
[1] "sivaranjani","snegha","kavya","lakshmi","soniya"
SALARY
[1] "722.5","815.2","1611","2829","843.25"
STARTDATE
[1] "6/17/2014","1/1/2012","11/15/2024","9/23/2022","10/12/2023"
DEPT
[1] "IT","IT","HR","OPERATION","FINANCE"
```

B.PROGRAM

JSON CODE:

```
{
"ID":["1","2","3","4","5"],
"NAME":["sivaranjani","snegha","kavya","lakshmi","soniya"],
"SALARY":["722.5","815.2","1611","2829","843.25"],
"STARTDATE":["6/17/2014","1/1/2012","11/15/2024","9/23/2022","10/12/2023"],
"DEPT":["IT","IT","HR","OPERATION","FINANCE"]
}
```

CODE:

```
library(rjson)
json_data <- fromJSON(file="D:/sample R/j2.json")
print(json_data)
if ("name" %in% names(json_data)) {
  print(paste("Name: ", json_data$name))
} else {
  print("Name not found in the JSON data.")
}
if ("address" %in% names(json_data)) {
  address <- json_data$address
  print(paste("City: ", address$city))
  print(paste("Zip Code: ", address$zip_code))
} else {
  print("Address not found in the JSON data.")
}
if ("items" %in% names(json_data)) {
  items <- json_data$items
  for (item in items) {
    print(paste("Item Name: ", item$name))
    print(paste("Item Price: ", item$price))
  }
} else {
  print("Items array not found in the JSON data.")
}
```

OUTPUT:

```
$ID
[1] "1" "2" "3" "4" "5"

$NAME
[1] "sivaranjani" "snegha"      "kavya"       "lakshmi"     "soniya"

$SALARY
[1] "722.5"  "815.2"  "1611"   "2829"   "843.25"

$STARTDATE
[1] "6/17/2014"  "1/1/2012"   "11/15/2024" "9/23/2022"  "10/12/2023"

$DEPT
[1] "IT"        "IT"        "HR"        "OPERATION" "FINANCE"

[1] "Name not found in the JSON data."
[1] "Address not found in the JSON data."
[1] "Items array not found in the JSON data."
```

5.Data Frame using CBIND() and RBIND() in R

Aim:

To demonstrate how to use the cbind() and rbind() functions in R to create and manipulate data frames. These functions allow for column-wise (cbind()) and row-wise (rbind()) binding of data, which is useful for combining multiple datasets into a single data frame.

Definitions:

- **cbind()**: This function is used to combine objects by columns. It binds two or more vectors, matrices, or data frames side by side.
- **rbind()**: This function is used to combine objects by rows. It stacks two or more vectors, matrices, or data frames on top of each other.

Key Concepts:

- **Column Binding (cbind())**: When you combine data by columns, the number of rows must be the same across all objects.
- **Row Binding (rbind())**: When you combine data by rows, the number of columns must be the same across all objects.

Algorithm:

1. **Create Data**: Start by creating individual vectors or data frames that you want to combine.
2. **Column Binding (cbind())**: Use cbind() to combine the data objects by columns.
3. **Row Binding (rbind())**: Use rbind() to combine the data objects by rows.
4. **Create a Data Frame**: After combining the data using either cbind() or rbind(), ensure the data is organized into a data frame.
5. **Display the Result**: Print or view the resulting data frame to verify the changes.

<u>A.PROGRAM:</u>

```
Name <- c("shubham Rastogi","nishka jain","Gunjan garg","sumit chaudhary")
Address <- c("Moradabad","etah","sambhal","Khurja")
Marks <- c(255,355,455,655)
info <- cbind(Name, Address, Marks)
print(info)
new.stuinfo <- data.frame(
  Name = c("Deepmala","Arun"),
  Address = c("khurja","moradabad"),
  Marks = c("755","855"),
  stringsAsFactors = FALSE
)
 cat("### The Second data frame \n")
 print(new.stuinfo)
 all.info<- rbind(info,new.stuinfo)
```

```
cat("### The Combined data frame \n")
print(all.info)
```

OUTPUT:

```
              Name                        Address              Marks
[1,]   "shubham Rastogi"       "Moradabad"           "255"
[2,]   "nishka jain"                  "etah"                  "355"
[3,]   "Gunjan garg"             "sambhal"                "455"
[4,]   "sumit chaudhary"        "Khurja"                 "655"
### The Second data frame
          Name                           Address                  Marks
1       Deepmala                      khurja                     755
2       Arun                        moradabad                   855
### The Combined data frame
                  Name                         Address              Marks
1          shubham Rastogi              Moradabad            255
2            nishka jain                      etah                     355
3           Gunjan garg                  sambhal              455
4          sumit chaudhary              Khurja                 655
5            Deepmala                      khurja               755
6               Arun                        moradabad            855
```

B.PROGRAM:

```
df1<-data.frame(
  Name=c("priya","mano","Ram"),
  Age=c(35,26,30)
)
df2<-data.frame(
  Gender=c("Female","male","male"),
  occupation=c("programmer","doctor","engineer")
)
df_combined_columns<-cbind(df1,df2)
print("Data frame after column-wise binding:")
print(df_combined_columns)
df3<-data.frame(
  Name=c("Raja","kavitha"),
  Age=c(40,45)
)
df4<-data.frame(
  Gender=c("male","female"),
  occupation=c("Lawyer","Scientist")
)
df_combined_rows<-rbind(df_combined_columns,cbind(df3,df4))
print("Data frame after row-wise binding:")
print(df_combined_rows)
```

OUTPUT:

```
[1] "Data frame after column-wise binding:"

       Name     Age      Gender      occupation
1      priya      35        Female     programmer
2       mano      26         male           doctor
3        Ram      30          male          engineer
[1] "Data frame after row-wise binding:"
       Name       Age     Gender      occupation
1       priya        35       Female      programmer
2       mano         26        male            doctor
3        Ram          30         male         engineer
4        Raja         40        male          Lawyer
5       kavitha        45        female        Scientist
```

6A1.Find the factorial of a number using recursion.

Aim:

The aim of this program is to find the factorial of a given number using recursion in R programming. The factorial of a number nnn (denoted as n!n!n!) is the product of all positive integers less than or equal to nnn. Recursion is a programming technique where a function calls itself to solve smaller instances of the same problem.

Factorial Definition:

- Factorial of 0: 0!=10! = 10!=1
- Factorial of a positive integer nnn: n!=n×(n−1)×(n−2)×...×1n! = n \times (n-1) \times (n-2) \times ... \times 1n!=n×(n−1)×(n−2)×...×1

For example:

- 5!=5×4×3×2×1=1205! = 5 \times 4 \times 3 \times 2 \times 1 = 1205!=5×4×3×2×1=120
- 3!=3×2×1=63! = 3 \times 2 \times 1 = 63!=3×2×1=6

Key Concept:

Recursion works by breaking down the problem into smaller subproblems. The recursive function for factorial can be defined as:

- Base Case: n=0n = 0n=0, the factorial is 1.
- Recursive Case: n!=n×(n−1)!n! = n \times (n-1)!n!=n×(n−1)!

Algorithm:

1. Input: Read the number nnn for which the factorial needs to be calculated.
2. Base Case Check: If n=0n = 0n=0, return 1 (because 0!=10! = 10!=1).
3. Recursive Case: If n>0n > 0n>0, recursively call the factorial function for n−1n-1n−1 and multiply the result by nnn.
4. Output: Return the computed factorial of nnn.

A.PROGRAM:

```
recur_fact<-function(num){
  if(num<=1){
    return(1)
  }else{
    return(num*recur_fact(num-1))
  }
}
print(paste("The factorial of 5 is",recur_fact(5)))
findfactorial<-function(n){
  factorial<-1
  if((n==0)|(n==1))
    factorial<-1
  else{
    for(i in 1:n)
      factorial<-factorial*i
  }
  return(factorial)
}
print(paste(findfactorial(0)))
print(paste(findfactorial(3)))
num = as.integer(readline(prompt="Enter a number:"))
factorial = 1
if (num < 0){
  print("Sorry; factorial does not exist for negative numbers")
} else if (num == 0) {
  print("The factorial of 0 is 1")
} else {
  for (i in 1:num) {
    factorial = factorial * i
  }
  print(paste("The factorial of", num, "is", factorial))
}
```

OUTPUT:

```
[1] "The factorial of 5 is 120"
 [1] "1"
[1] "6"
 Enter a number:8
“The factorial of 8 is 40320
```

B.PROGRAM:

```
factorial<-function(n){
  if(n==0){
    return(1)
  }else{
    return(n*factorial(n-1))
  }
}
number<-7
result<-factorial(number)
print(paste("Factorial of",number,"is",result))
```

output:

```
[1] "Factorial of 7 is 5040"
```

6B . Mean,Variance,Standard deviation for the given probability distribution

Aim:

The aim of this program is to calculate the **mean**, **variance**, and **standard deviation** for a given probability distribution in **R programming**. **Mean (Expected Value)**: The average value of a random variable, weighted by its probabilities.

- **Variance**: Measures the spread of the values in the probability distribution, indicating how much the values deviate from the mean.
- **Standard Deviation**: The square root of the variance, which provides a measure of the dispersion of the distribution in the same units as the data.

Formulae:

1. **Mean (Expected Value)**:
   E(X)=∑i=1nxi·piE(X) = \sum_{i=1}^{n} x_i \cdot p_iE(X)=∑i=1nxi·pi
   Where xix_ixi is the value, and pip_ipi is the probability of xix_ixi.
2. **Variance**:
   Var(X)=∑i=1npi·(xi−E(X))2\text{Var}(X) = \sum_{i=1}^{n} p_i \cdot (x_i - E(X))^2Var(X)=∑i=1npi·(xi−E(X))2
   Where xix_ixi is the value, pip_ipi is the probability, and E(X)E(X)E(X) is the mean.
3. **Standard Deviation**:
   σ(X)=Var(X)\sigma(X) = \sqrt{\text{Var}(X)}σ(X)=Var(X)

Algorithm:

1. **Input**:
   - A set of values x1,x2,...,xnx_1, x_2, ..., x_nx1,x2,...,xn representing the outcomes of a random variable.
   - Corresponding probabilities p1,p2,...,pnp_1, p_2, ..., p_np1,p2,...,pn representing the probability of each outcome.
2. **Calculate the Mean**:
   - Compute the mean (expected value) of the distribution using the formula:
     E(X)=∑i=1nxi·piE(X) = \sum_{i=1}^{n} x_i \cdot p_iE(X)=i=1∑nxi·pi

3. **Calculate the Variance**:
    - Compute the variance using the formula: Var(X)=∑i=1npi·(xi−E(X))2\text{Var}(X) = \sum_{i=1}^{n} p_i \cdot (x_i - E(X))^2Var(X)=i=1∑npi·(xi−E(X))2
4. **Calculate the Standard Deviation**:
    - Compute the standard deviation by taking the square root of the variance: σ(X)=Var(X)\sigma(X) = \sqrt{\text{Var}(X)}σ(X)=Var(X)
5. **Output**: Return the values for the mean, variance, and standard deviation.

A.PROGRAM:

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

mean.result=mean(x)

print(mean.result)

median.result=median(x)

print(median.result)

variance.result=var(x)

print(variance.result)

sd.result=sqrt(var(x))

print(sd.result)
```

output:

```
[1] 2.8
 [1] 2.5
 [1] 2.484211
 [1] 1.576138
```

B.PROGRAM:

```
outcomes <- c(1, 2, 3)

probabilities <- c(0.2, 0.5, 0.3)

mean_value <- sum(outcomes * probabilities)

cat("Mean (E(X)):", mean_value, "\n")

variance_value <- sum((outcomes - mean_value)^2 * probabilities)

cat("Variance (Var(X)):", variance_value, "\n")

std_dev_value <- sqrt(variance_value)

cat("Standard Deviation (SD):", std_dev_value, "\n")
```

output:

```
Mean (E(X)): 2.1
Variance (Var(X)): 0.49
Standard Deviation (SD): 0.7
```

7.Apriori Algorithm to extract Association rule of datamining

Aim:

To implement the Apriori algorithm in R programming to extract association rules from a given transactional dataset. The Apriori algorithm is a popular method in data mining that is used for discovering frequent itemsets and generating association rules from large datasets.

Algorithm:

1. **Input**:
   - A transactional dataset with items bought by customers.
   - Minimum support threshold and confidence threshold.
2. **Generate Frequent Itemsets**:
   - Start with individual items and count their frequency.
   - Find itemsets that meet the minimum support threshold.
   - Use these frequent itemsets to generate candidate itemsets of length 2, then 3, and so on.
   - Continue until no more frequent itemsets are found.
3. **Generate Association Rules**:
   - For each frequent itemset, generate rules.
   - Calculate the confidence of each rule.
   - Keep the rules that meet the minimum confidence threshold.
4. **Output**:
   - A list of association rules with support, confidence, and lift values.

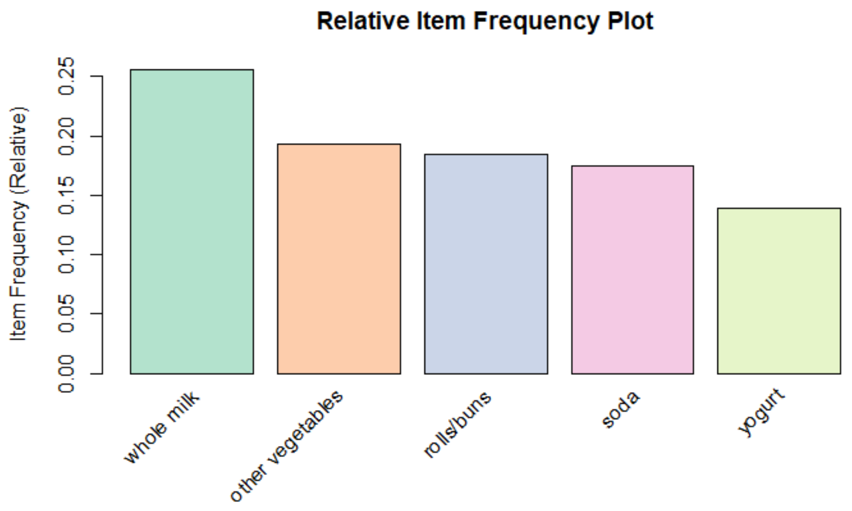
A.PROGRAM:

```
library(arules)

library(arulesViz)

library(RColorBrewer)

data("Groceries")
```

```
rules <- apriori(Groceries,
        parameter = list(supp = 0.01, conf = 0.2))
inspect(rules[1:10])
arules::itemFrequencyPlot(Groceries, topN = 5,
            col = brewer.pal(8, 'Pastel2'),
            main = 'Relative Item Frequency Plot',
            type = "relative",
            ylab = "Item Frequency (Relative)")
```

OUTPUT

```
     lhs                    rhs              support     confidence coverage     lift      count
[1]   {}                 => {whole milk}      0.255516    0.25551601.0000000     1.0000     513
[2]  {hard cheese}   => {whole milk}      0.010066     0.4107884   0.024504     1.60768     99
[3]  {butter milk}    => {othervegetable} 0.01037      0.3709091  0.0279613     1.91691     102
[4]  {butter milk}    => {whole milk}      0.011591    0.4145455  0.0279613     1.62238     114
[5]   {ham}              => {whole milk}      0.011489    0.4414062  0.026029      1.7275      113
[6]  {sliced cheese} => {whole milk}      0.010777    0.4398340  0.0245043     1.7213      106
[7]  {oil}                => {whole milk}      0.011286   0.402173  0.02806304     1.5739      111
[8]  {onions}           => {other vegetable} 0.014234    0.459016 0.03101169     2.3722     140
[9]  {onions}           => {whole milk}       0.012099    0.39016  0.03101169     1.5269     119
[10]  {berries}          => {yogurt}           0.010574    0.31804  0.03324860     2.2798     104
```



B.PROGRAM:

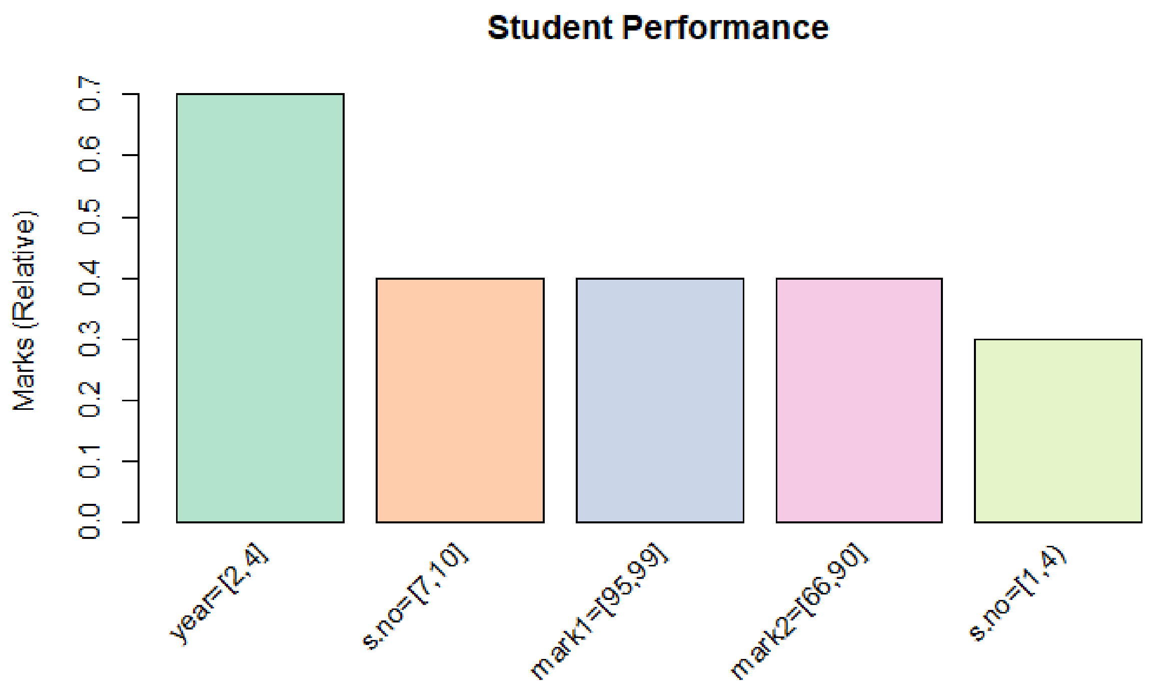
```
library(arules)
library(arulesViz)
library(RColorBrewer)
stud_details <- read.csv("D:/sample R/stud_details.csv")
stud_details_trans <- as(stud_details, "transactions")
rules <- apriori(stud_details_trans, parameter = list(supp = 0.01, conf = 0.2))
inspect(rules[1:10])
itemFrequencyPlot(stud_details_trans, topN = 5,
          col = brewer.pal(8, 'Pastel2'),
          main = 'Student Performance',
          type = "relative",
          ylab = "Marks (Relative)")
```

Stud_details:

s.no	s.name	mark1	mark2	dept	year
1	priya	60	56	mca	1
2	mano	67	66	bca	3
3	ram	68	67	mba	2
4	raja	58	78	bba	1
5	ravi	79	65	bcom	2
6	malathi	78	45	mcom	3
7	snegha	96	55	msc	2
8	ashwin	95	90	bsc	2
9	kavya	98	61	ba	1
10	siva	99	62	be	4

OUTPUT:

```
     lhs          rhs                      support       confidence   coverage       lift      count
[1]  {}  => {s.no=[4,7)}                0.3              0.3           1            1        3
[2]  {}  => {mark1=[58,6 8)}         0.3             0.3            1            1        3
[3]  {}  => {mark2=[45,61)}          0.3             0.3            1            1        3
[4]  {}  => {year=[1,2)}               0.3              0.3           1           1        3
[5]  {}  => {mark2=[61,66) }         0.3             0.3            1            1        3
[6]  {}  => {mark1=[68,95)}          0.3             0.3            1            1        3
[7]  {}  => {s.no=[1,4)}              0.3              0.3            1            1        3
[8]  {}  => {mark1=[95,99]}          0.4             0.4            1            1        4
[9]  {}  => {mark2=[66,90]}          0.4              0.4           1            1        4
[10] {}  => {s.no=[7,10]}             0.4              0.4           1           1         4
```



8.K-Means Clustering Technique

<u>Aim:</u>

To implement the K-Means clustering technique in R programming. K-Means is an unsupervised machine learning algorithm used to partition a dataset into K distinct, non-overlapping subsets (clusters).

Algorithm:

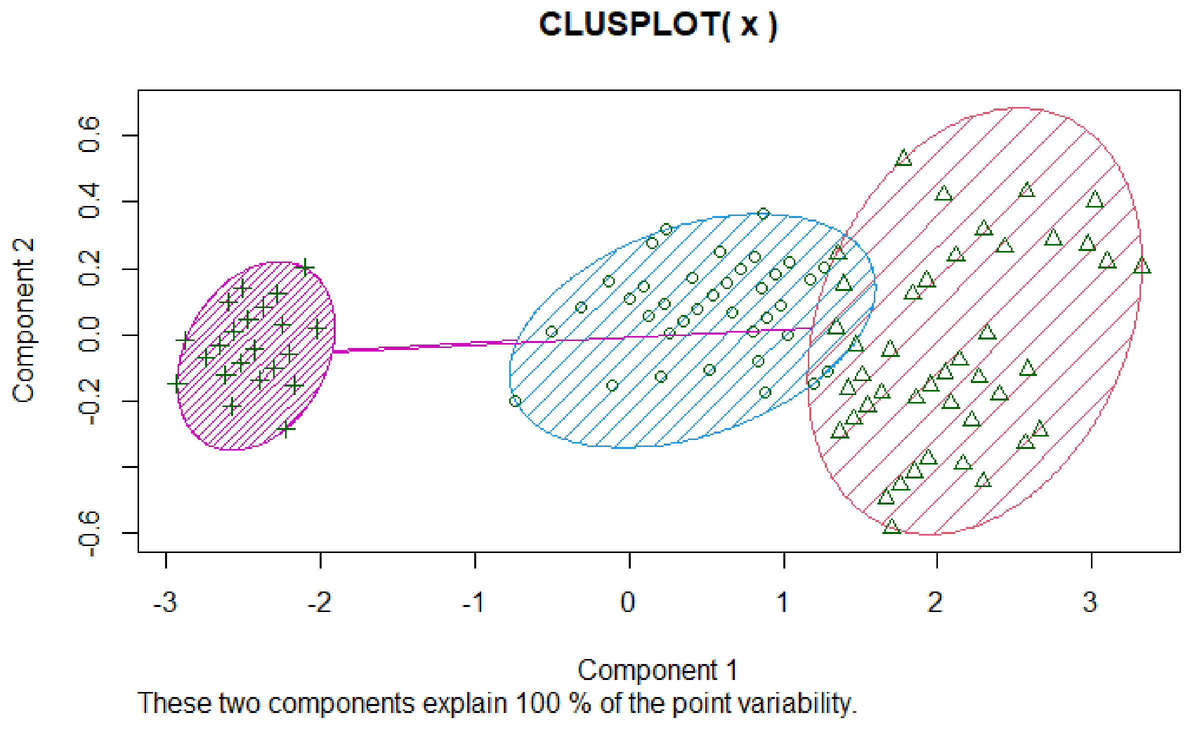
1. **Input**:
   - A dataset with features (numerical values).
   - The number of clusters, KKK.
2. **Initialization**:
   - Select KKK initial centroids randomly from the data points.
3. **Cluster Assignment**:
   - For each data point, calculate the distance to each of the KKK centroids.
   - Assign the data point to the cluster with the nearest centroid.
4. **Centroid Update**:
   - After all points are assigned, recalculate the centroids as the mean of the data points in each cluster.
5. **Repeat**:
   - Repeat the cluster assignment and centroid update steps until the centroids no longer change or a maximum number of iterations is reached.
6. **Output**:
   - Final clusters with their corresponding data points.
   - The final centroids.

A.PROGRAM:

```
data("iris")
head(iris)
x=iris[,3:4]
head(x)
model=kmeans(x,3)
library(cluster)
clusplot(x,model$cluster)
clusplot(x,model$cluster,color=T,shade=T)
```

OUTPUT:

```
  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width  Species
1      5.1          3.5          1.4           0.2       setosa
2      4.9          3.0          1.4           0.2       setosa
3      4.7          3.2          1.3           0.2       setosa
4      4.6          3.1          1.5           0.2       setosa
5      5.0          3.6          1.4           0.2       setosa
6      5.4          3.9          1.7           0.4       setosa

  Petal.Length  Petal.Width
1      1.4          0.2
2      1.4          0.2
3      1.3          0.2
4      1.5          0.2
5      1.4          0.2
6      1.7          0.4
```

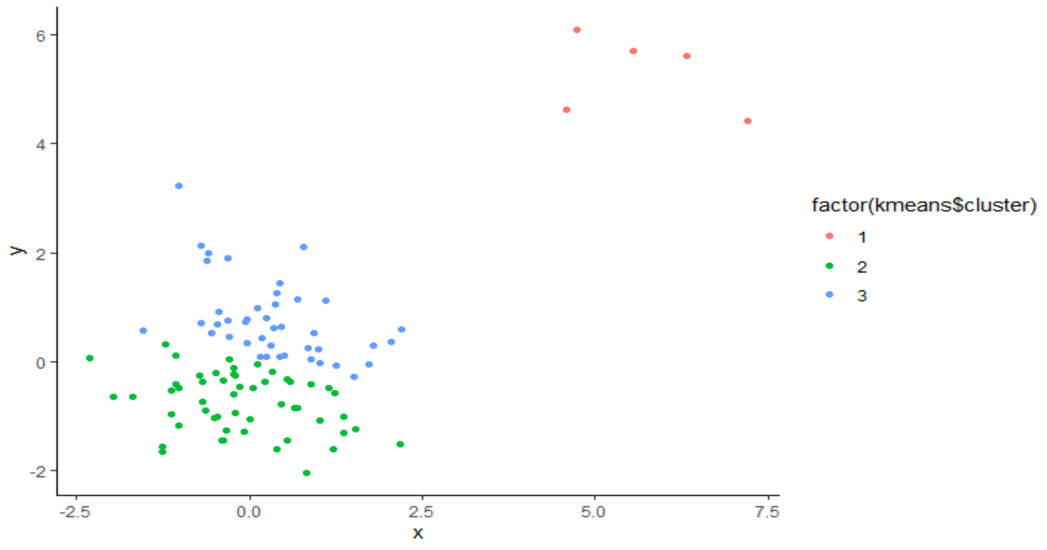

CLUSPLOT(x)
Component 2
0.6
0.4
0.2
0.0
-0.2
-0.6
-3
-2
-1
0
1
2
3
Component 1
These two components explain 100 % of the point variability.

8B.PROGRAM:

```
library(ggplot2)

set.seed(123)

data<-data.frame(x=rnorm(100,mean=0,sd=1),

              y=rnorm(100,mean=0,sd=1))

data$x[1:5]<-rnorm(20,mean=5,sd=1)

data$y[1:5]<-rnorm(20,mean=5,sd=1)

set.seed(123)

kmeans<-kmeans(data,centers=3)

print(kmeans$cluster)

ggplot(data,aes(x=x,y=y,color=factor(kmeans$cluster)))+geom_point()+theme_classic()
```

OUTPUT:

```
  [1] 1 1 1 1 1 3 2 2 2 3 2 3 2 2 3 3 3 2 2 2 2 2 2 2 2 3 2 3 3 2 3 3 3 3 2 2 3 2 3 3 2 3 2 2 2 2 2 2
 [48] 3 3 2 3 3 3 2 2 3 3 2 3 2 3 2 2 3 2 3 3 2 3 3 2 2 3 3 2 2 2 2 3 2 2 3 2 2 2 2 3 3 3 2 3 2 3 2
 [95] 2 3 3 2 2 2
```



9.CLASSIFICATION ALGORITHM

Aim:

The aim of this program is to implement a Classification algorithm in R programming to predict the class labels of a dataset based on input features.

Algorithm:

1. **Input**: A dataset with features and class labels.
2. **Preprocessing**:
   - Handle missing values.
   - Normalize or scale features if required.
   - Split the dataset into training and testing sets.
3. **Choose a Classification Algorithm**:
   - In this case, we will use a **Decision Tree** algorithm as an example.
4. **Train the Model**: Use the training data to train the model.
5. **Test the Model**: Evaluate the model on the test data using performance metrics.
6. **Output**: The model's classification results (predictions) and performance metrics.

A.PROGRAM:

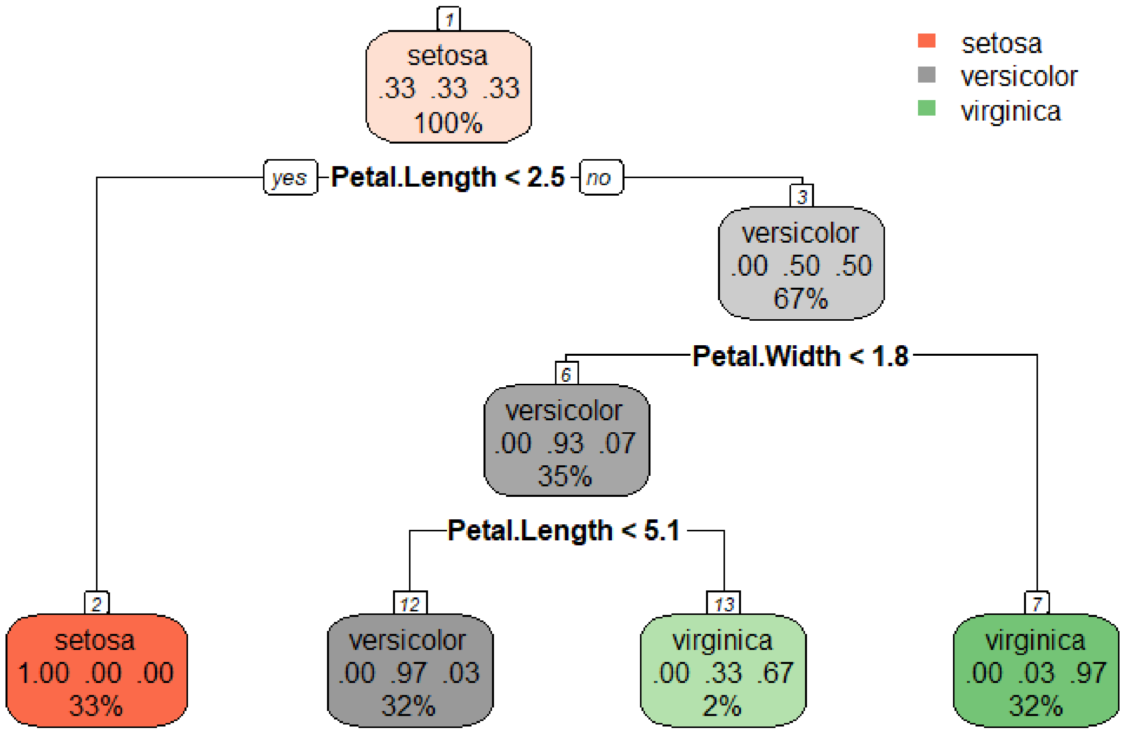
```
library(rpart)

library(rpart.plot)

library(caret)

data(iris)

set.seed(123)

train_index<-createDataPartition(iris$Species,p=0.8,list=FALSE)

train_data<-iris[train_index,]

test_data<-iris[-train_index,]
```

```
tree_model<-rpart(Species ~ .,
                data=train_data,
                method="class",
                control=rpart.control(minsplit=10,cp=0.01))
rpart.plot(tree_model,box.palette="auto",nn=TRUE)
predictions<-predict(tree_model,test_data,type="class")
confusionMatrix(predictions,test_data$Species)
```

OUTPUT:

```
Confusion Matrix and Statistics

          Reference
Prediction   setosa  versicolor   virginica
  setosa       10        0             0
 versicolor     0       10             1
  virginica     0        0             9

Overall Statistics

               Accuracy         : 0.9667
                 95% CI         : (0.8278, 0.9992)
    No Information Rate  : 0.3333
    P-Value [Acc > NIR]  : 2.963e-13

                            Kappa : 0.95

 Mcnemar's Test P-Value : NA

Statistics by Class:

                      Class: setosa    Class: versicolor    Class: virginica
Sensitivity              1.0000           1.0000              0.9000
Specificity              1.0000           0.9500              1.0000
Pos Pred Value           1.0000           0.9091              1.0000
Neg Pred Value           1.0000           1.0000              0.9524
```

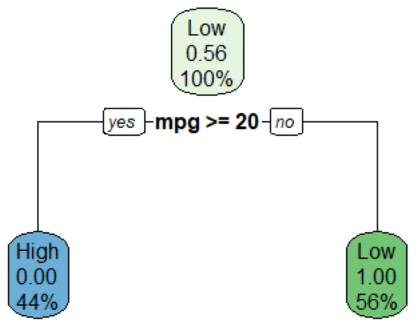
```
Prevalence             0.3333      0.3333          0.3333
Detection Rate         0.3333      0.3333          0.3000
Detection Prevalence   0.3333      0.3667          0.3000
Balanced Accuracy      1.0000      0.9750          0.9500
```



9B.PROGRAM:

```
library(rpart)

library(rpart.plot)

library(caret)

data(mtcars)

mtcars$mpg_category <- ifelse(mtcars$mpg > 20, "High", "Low")

set.seed(123)

train_data <- mtcars[createDataPartition(mtcars$mpg_category, p=0.8, list=FALSE), ]

test_data <- mtcars[-createDataPartition(mtcars$mpg_category, p=0.8, list=FALSE), ]

tree_model <- rpart(mpg_category ~ ., data=train_data)

rpart.plot(tree_model)

predictions <- predict(tree_model, test_data, type="class")

confusionMatrix(predictions, test_data$mpg_category)
```

OUTPUT:



10.Pie Charts and Bar Charts Using R

Aim:

The aim of this program is to demonstrate how to create Pie Charts and Bar Charts in R programming for data visualization. These charts are used to represent categorical data in a way that is easy to interpret.

Algorithm:

1. Pie Chart Algorithm:

1. **Input**: A categorical dataset with counts or percentages for each category.
2. **Preprocessing**:
   - Convert the data into a vector of counts or percentages.
3. **Generate Pie Chart**:
   - Use the pie() function in R to create the pie chart.
   - Specify the categories and corresponding values.
4. **Output**: A pie chart displaying the distribution of categories.

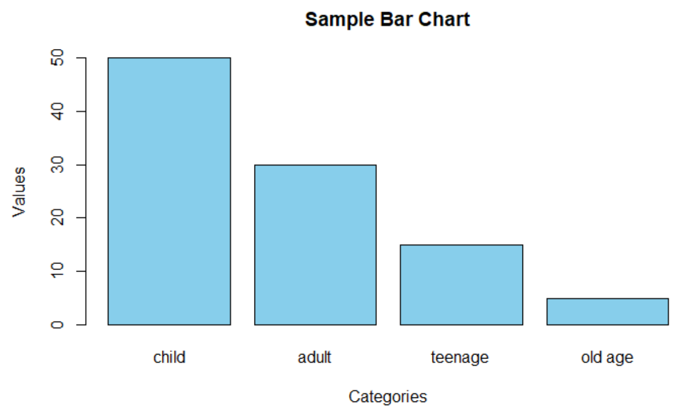
2. Bar Chart Algorithm:

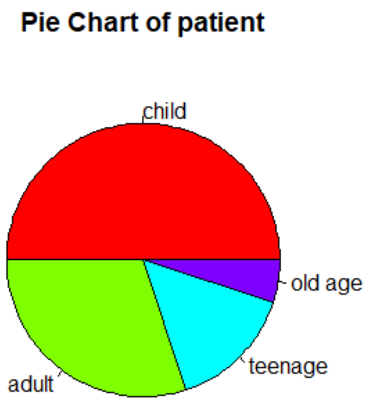
1. **Input**: A categorical dataset with counts or values for each category.
2. **Preprocessing**:
   - Convert the data into a vector of counts or values.
3. **Generate Bar Chart**:
   - Use the barplot() function in R to create the bar chart.
   - Specify the categories and their corresponding values.
4. **Output**: A bar chart displaying the comparison of categories.

<u>A.PROGRAM</u>

```
patient <- c(50, 30, 15, 5)
count <- c("child", "adult", "teenage", "old age")
pie(patient, labels = count, col = rainbow(length(patient)),
   main = "Pie Chart of patient")
patient <- c(50, 30, 15, 5)
count <- c("child", "adult", "teenage", "old age")
barplot(patient, names.arg = count, col = "skyblue",
        main = "Sample Bar Chart",
        xlab = "Categories", ylab = "Values")
```

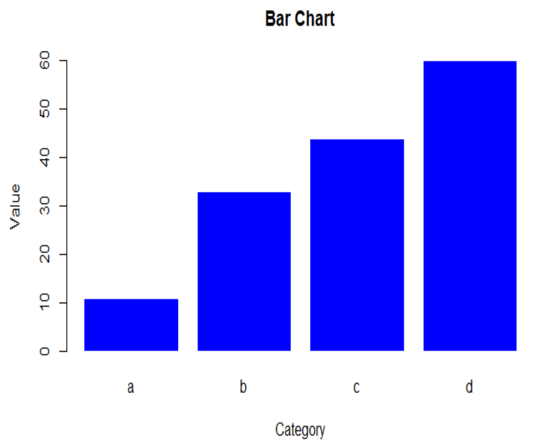
OUTPUT:

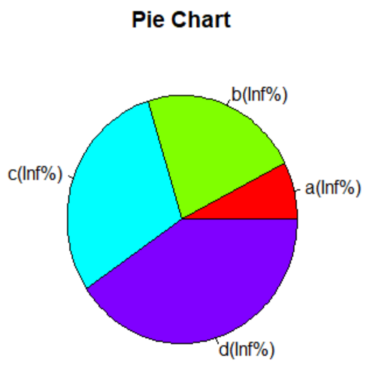




B.PROGRAM

```
data <- read.csv("D:/DV/Book1.csv")
print(data)
barplot(data$value,
        names.arg = data$category,
        col = "blue",
        main = "Bar Chart",
        xlab = "Category",
        ylab = "Value",
        border = "white")
percentages <- round(data$value / sum(data$Value) * 100, 1)
pie(data$value,
    labels = paste(data$category, "(", percentages, "%)", sep = ""),
    col = rainbow(length(data$value)),
    main = "Pie Chart")
```


Bar Chart
Value
a
b
c
d
Category


Pie Chart
b(Inf%)
c(Inf%)
a(Inf%)
d(Inf%)