Ex No 1: Write a program to perform numerical operations (MAX, MIN, AVG, SUM, SQRT, ROUND) using R.

AIM:

To write a program to perform numerical operations (MAX, MIN, AVG, SUM, SQRT, ROUND).

ALGORITHM:

- ❖ Use the read.csv() function to read the data from the CSV file and stores it in a variable called csv_data.
- ❖ Print the contents (age column) of the csv_data variable, displaying the data from the CSV file.
- ❖ Calculate the minimum and maximum values of the 'Age' column in the csv_data data-frame using the min() and max() functions.
- ❖ Calculate the average (mean) of the 'Age' column in the csv_data data frame using the mean() function.
- ❖ Calculate the square root of each value in the 'Age' column using the sqrt() function.
- ❖ Apply the round() function to the avg variable to round the average age.
- ❖ Calculate the sum of all values in the 'Age' column using the sum() function.

CODE:

```
 # Import the data using read.csv()
csv_file_path <- "C:\\Users\\HARI RAGHAV\\Downloads\\MMM.csv"
csv_data <- read.csv(csv_file_path)
print(csv_data$Age)
# Compute the min&max value
print(min(csv_data$Age))
print(max(csv_data$Age))
# Compute the Average
avg = mean(csv_data$Age)
print(avg)
# Compute the Squareroot
print(sqrt(csv_data$Age))
# Apply round function for the Average Value
round(avg)
# Compute the Sum
print(sum(csv_data$Age))
```

OUTPUT:

```
20 22 100 100
Min & Max: 20 & 100
Avg: 60.5
Sqrt: 4.472136 ,4.690416,10.000000,10.000000,
Round: 60
Sum: 242
```

RESULT:

Thus, a program to perform numerical operations has been successfully executed using R.

Ex No 2: Implement a program for statistical operations such as Mean, Median, Mode and Standard deviation.

AIM:

To write a program for performing statistical operations such as mean median mode.

ALGORITHM:

- ❖ Use the read.csv() function to read the data from the CSV file and stores it in a variable called csv_data.
- ❖ Print the contents (age column) of the csv_data variable, displaying the data from the CSV file.
- ❖ Calculate the mean (average) value of the 'Age' column in the csv_data dataframe using the mean() function.
- ❖ Calculate the median value of the 'Age' column in the csv_data dataframe using the median() function.
- ❖ The function uses the table() function to create a frequency table and identifies the mode.
- ❖ Calculate the standard deviation of the 'Age' column in the csv_data dataframe using the sd() function.
- ❖ Print the results.

CODE:

```
# Import the data using read.csv()
csv_file_path <- "C:\\Users\\HARI RAGHAV\\Downloads\\MMM.csv"
csv_data <- read.csv(csv_file_path)
print(csv_data$Age)
# Compute the mean value
mean = mean(csv_data$Age)
print(mean)
# Compute the median value
median = median(csv_data$Age)
print(median)
# Compute the mode value
mode = function()
{
return(sort(-table(csv_data$Age))[1])
}
mode()
# Compute the Standard Deviation
print(sd(csv_data$Age))
```

OUTPUT:

```
Age: 20 22 100 100
Mean: 60.5
Median: 61
Mode: 100, -2
Standard Deviation:  45.61798
```

RESULT:

Thus, a program to perform statistical operations has been successfully executed using R.

Ex No 3: Write a Program to Read and Write operations on different types of Files (csv, xls, txt etc).

AIM:

To write a program to read and write operations on different types of Files (csv, xls, txt etc).

ALGORITHM:

- Reading CSV File: Use the `read.csv()` function to read the CSV file specified by `csv_file_path`.
- Writing CSV File: Use the `write.csv()` function to write the `csv_data` data frame to `csv_output_file`.
- Reading Excel File: Use the `read_excel()` function from the `readxl` library to read the Excel file specified by `excel_file_path`.
- Writing Excel File: Use the `write_xlsx()` function to write the data frame to `excel_output_file`.
- Reading TXT file: Use the `read.delim()` function to read the tab-delimited text file specified by `txt_file_path`.
- Writing TXT file:Use the `write.table()` function to write `data_txt` to `txt_output_file`, using `*` as the separator.

CODE:

```
#Reading CSV File

csv_file_path <- "C:\\Users\\HARI RAGHAV\\Downloads\\HelloEX1a.csv"

csv_data <- read.csv(csv_file_path)

print(csv_data)
```

```
#Writing CSV File

csv_data<-data.frame(Name = c("sdfsafasdca","vva","bffbb"),Age=c(14,15,16),City=c("ddd1","aaa1","bbb1"))

csv_output_file <- "C:\\Users\\HARI RAGHAV\\Downloads\\HelloEX1b.csv"

write.csv(csv_data, csv_output_file,row.names = FALSE)

print(csv_data)

#Reading Excel File

library(readxl)

Data1<-read_excel("C:\\Users\\HARI RAGHAV\\Downloads\\HelloEX1c.xlsx")

print(Data1)

#Writing Excel File

write_xlsx(data.frame(Name=c("ggg"),Age=c(23),City=("df")),"C:\\Users\\HARI RAGHAV\\Downloads\\HelloEX1cii.xlsx")

Data2<-read_excel("C:\\Users\\HARI RAGHAV\\Downloads\\HelloEX1cii.xlsx")

print(Data2)

# Reading TXT file

myData = read.delim("C:\\Users\\HARI RAGHAV\\Downloads\\HelloEX1d.txt", header = FALSE)

print(myData)
```

```
#Writing TXT file

data_txt="Welcome to R programing!!"

data=write.table(data_txt,file = "C:\\Users\\HARI RAGHAV\\Downloads\\HelloEX1d.txt", sep = "*")

print(data)
```

OUTPUT:

```
1 Hari         20 MDU

2 Gopi         22 MDU

# A tibble: 1 × 3

  Name    Age City

  <chr> <dbl> <chr>

1 ggg      23 df

2 1*Welcome to R programing!!
```

RESULT:

Thus, a program to perform Read & write operations has been successfully executed using R.

Ex No 4: Implement data pre-processing operations:

❖ **Handling Missing data**

❖ **Min-Max normalization**

AIM:

To implement data pre-processing operations:

❖ Handling Missing data

❖ Min-Max normalization

ALGORITHM:

❖ **Handling Missing data:**

➢ Import dplyr Library.

➢ Create a Data Frame.

➢ Identify Columns with Missing Values.

➢ Calculate Means and Medians of Columns with Missing Values.

➢ Impute Missing Values with Medians.

➢ Print the Resulting Data Frame.

❖ **Min-Max Normalization:**

➢ Import the `caret` library.

➢ Create a Dataset (`data`).

➢ Define a Custom Min-Max Scaling Function (`minMax`).

- Normalize Data Using the Custom Function (`normalisedMydata`).
- Print the First Few Rows of Normalized Data (`head(normalisedMydata)`).
- Check Summary Statistics After Normalization (`summary(normalisedMydata)`).

<u>CODE:</u>

<u>Handling Missing data:</u>

```
library(dplyr)

dataframe <- data.frame( Name = c("Bhuwanesh", "Anil", "Jai",
"Naveen"),

   Physics = c(98, 87, 91, 94),

   Chemistry = c(NA, 84, 93, 87),

   Mathematics = c(91, 86, NA, NA) )

dataframe

listMissingColumns <- colnames(dataframe)[ apply(dataframe, 2,
anyNA)]

meanMissing <- apply(dataframe[,colnames(dataframe) %in%
listMissingColumns],  2, mean, na.rm =  TRUE)

medianMissing <- apply(dataframe[,colnames(dataframe) %in%
listMissingColumns], 2, median, na.rm =  TRUE)

newDataFrameMedian <- dataframe %>% mutate(

   Chemistry = ifelse(is.na(Chemistry), medianMissing[1], Chemistry),
   Mathematics =  ifelse(is.na(Mathematics),
medianMissing[2],Mathematics))

print(newDataFrameMedian)
```

Min-Max Normalization:

```
library(caret)

data = data.frame(var1 = c(120,345,145,122,596,285,211),

                  var2 = c(10,15,45,22,53,28,12),

                  var3 = c(-34,0.05,0.15,0.12,-6,0.85,0.11))

minMax <- function(x) {

(x - min(x)) / (max(x) - min(x)) }

#normalise data using custom function

normalisedMydata <- as.data.frame(lapply(data, minMax))

head(normalisedMydata)

summary(normalisedMydata)
```

OUTPUT:

Handling Missing data:

```
 Name          Physics    Chemistry    Mathematics
 Bhuwanesh       98           NA          91
 Anil            87           84          86
 Jai             91           93          NA
 Naveen          94           87          NA
```

```
Name          Physics  Chemistry  Mathematics
Bhuwanesh      98          87         91.0
Anil           87          84         86.0
Jai            91          93         88.5
Naveen         94          87         88.5
```

MIN-MAX Normalization:

```
         var1      var2      var3
1 0.000000000 0.0000000 0.0000000
2 0.472689076 0.1162791 0.9770445
3 0.052521008 0.8139535 0.9799139
4 0.004201681 0.2790698 0.9790531
5 1.000000000 1.0000000 0.8034433
6 0.346638655 0.4186047 1.0000000
```

```
      var1              var2              var3
 Min.   :0.00000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.02836   1st Qu.:0.0814   1st Qu.:0.8902
 Median :0.19118   Median :0.2791   Median :0.9788
 Mean   :0.29532   Mean   :0.3821   Mean   :0.8169
 3rd Qu.:0.40966   3rd Qu.:0.6163   3rd Qu.:0.9795
 Max.   :1.00000   Max.   :1.0000   Max.   :1.0000
```

RESULT:

Thus, a program to implement data pre-processing operations has been successfully executed using R.

Ex No 5: Write a Program to implement Principal Component Analysis for House dataset

AIM:

To implement Principal Component Analysis for House dataset.

ALGORITHM:

- Generate a sample dataset with features and house prices.
- Load necessary R libraries: dplyr, caret, and pcaMethods.
- Prepare and separate the data into features (X) and target variable (y).
- Perform PCA on the features and choose to retain 3 principal components.
- Train a linear regression model on the PCA-transformed data.
- Create a new dataset with similar features and different values.
- Apply the same PCA transformation to the new data.
- Use the model to predict house prices for the new dataset.

CODE:

```
# Load necessary libraries

library(dplyr)

# Set a random seed for reproducibility

set.seed(123)

# Create a sample dataset with 100 observations

n <- 100

# Generate random values for features

bedrooms <- sample(1:5, n, replace = TRUE)
```

```
bathrooms <- sample(1:3, n, replace = TRUE)

square_feet <- rnorm(n, mean = 1500, sd = 500)

garage <- sample(0:1, n, replace = TRUE)

year_built <- sample(1950:2020, n, replace = TRUE)

# Generate house prices based on the features (with some random noise)

house_prices <- 50000 + 25000 * bedrooms + 30000 * bathrooms +

  200 * square_feet + 15000 * garage - 10 * (2023 - year_built) +
rnorm(n, mean = 0, sd = 50000)

# Create the sample dataset

sample_data <- data.frame(

  Bedrooms = bedrooms,

  Bathrooms = bathrooms,

  SquareFeet = square_feet,

  Garage = garage,

  YearBuilt = year_built,

  HousePrice = house_prices

)

# Display the first few rows of the dataset

head(sample_data)

# Load necessary libraries

install.packages("caret")
```

```
library(caret)

install.packages("pcaMethods")

library(pcaMethods)

# Prepare your data

X <- sample_data[, -6]  # Features (exclude the target variable)

y <- sample_data$HousePrice  # Target variable

# Perform PCA

pca_result <- prcomp(X, center = TRUE, scale = TRUE)

# Examine explained variance

summary(pca_result)

cumsum(pca_result$sdev^2) / sum(pca_result$sdev^2)

# Select the number of principal components (e.g., 3 components)

n_components <- 3

# Ensure that 'X_pca' is available before converting to a data frame

X_pca <- pca_result$x

# convert 'X_pca' to a data frame

X_pca_df <- as.data.frame(X_pca)

# Train a linear regression model

model <- lm(y ~ ., data = X_pca_df)

# Display the summary of the regression model

summary(model)
```

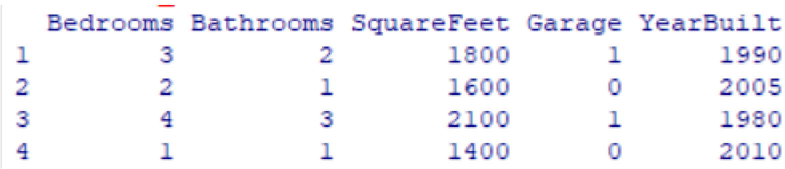
```
# Create a new dataset with the same feature columns
new_data <- data.frame(
  Bedrooms = c(3, 2, 4, 1),        # Example values for Bedrooms
  Bathrooms = c(2, 1, 3, 1),       # Example values for Bathrooms
  SquareFeet = c(1800, 1600, 2100, 1400),  # Example values for
SquareFeet
  Garage = c(1, 0, 1, 0),          # Example values for Garage
  YearBuilt = c(1990, 2005, 1980, 2010)  # Example values for
YearBuilt
)
# Load necessary libraries (if not already loaded)
library(caret)
# Assuming you've already created and loaded the new_data dataset
# Prepare your new data
X_new <- new_data[, -6]  # Features (exclude the target variable)
print(X_new)
# Use the same PCA transformation as the original dataset
X_new_pca <- predict(pca_result, newdata = X_new)
# Convert X_new_pca to a data frame
X_new_pca_df <- as.data.frame(X_new_pca)
# Predict house prices using the trained linear regression model
```

predicted_prices <- predict(model, newdata = X_new_pca_df)

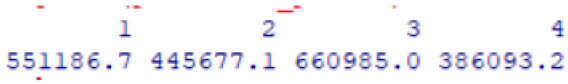
# The 'predicted_prices' variable now contains the predicted house prices for the new dataset.

print(predicted_prices)

OUTPUT:



```
  Bedrooms Bathrooms SquareFeet Garage YearBuilt
1        3         2       1800      1      1990
2        2         1       1600      0      2005
3        4         3       2100      1      1980
4        1         1       1400      0      2010
```



```
        1         2         3         4 
551186.7 445677.1 660985.0 386093.2 
```

RESULT:

Thus, a program to implement Principal Component Analysis for House dataset has been successfully executed using R.

Ex No 6: Implement simple linear regression program to predict the future values and analyze the goodness of fit.

AIM:

To Implement simple linear regression program to predict the future values and analyze the goodness of fit.

ALGORITHM:

- ❖ Data Collection: Create vectors for height and weight to store height and weight data.
- ❖ Linear Regression Model: Build a linear regression model relation using the lm function to predict weight from height.
- ❖ Create a data frame newData with a specific height value (170) for prediction.
- ❖ Predict weight for the new data point using the predict function and print the predicted weight value.
- ❖ Data Visualization: Create a scatter plot of height and weight using plot, with labeled axes and visualize the result.

CODE:

```
height <- c (150, 160, 140, 155, 148, 177, 167, 126, 149, 131)

weight <- c (60, 70, 55, 55, 58, 80, 75, 45, 50, 44)

relation <- lm(weight ~ height)

print (relation)

print (summary(relation))

newData <- data.frame(height = 170)

print (newData)
```

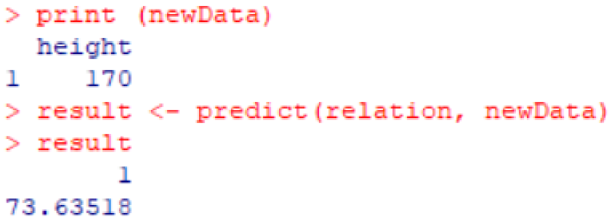
result <- predict(relation, newData)

result

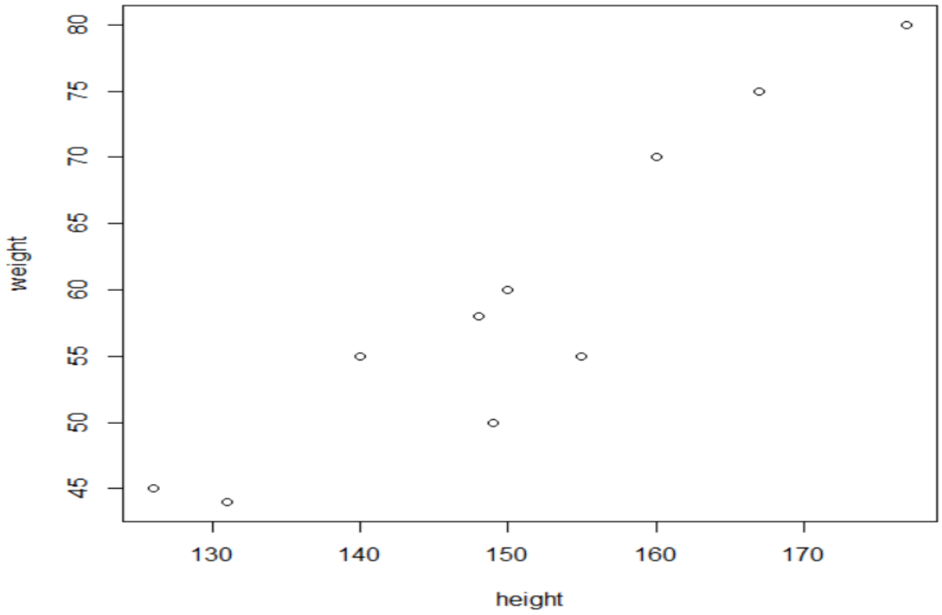
plot(weight, height)

plot(height, weight)

OUTPUT:



```
> print (newData)
  height
1    170
> result <- predict(relation, newData)
> result
       1
73.63518
```



RESULT:

Thus, a program to implement linear regression program to predict the future values and analyze the goodness of fit has been successfully executed using R.

Ex No 7: Write a Program to implement Simple Naïve Bayes classification algorithm for predicting the weather forecast

AIM:

To implement Simple Naive Bayes classification algorithm for predicting the weather forecast

ALGORITHM:

- ❖ Load weather data from a CSV file.
- ❖ Convert the "weather" column to categories (e.g., rainy, sunny).
- ❖ Split the data into training and testing sets.
- ❖ Build a Naive Bayes model to predict weather based on features like precipitation, temperature, and wind using the training data.
- ❖ Use the model to predict the weather on the test data and store the predictions.
- ❖ Prepare a new data frame with sample values for weather-related features.
- ❖ Apply the trained model to predict weather conditions for the new data.
- ❖ Print the predicted weather conditions for each observation in the new data.

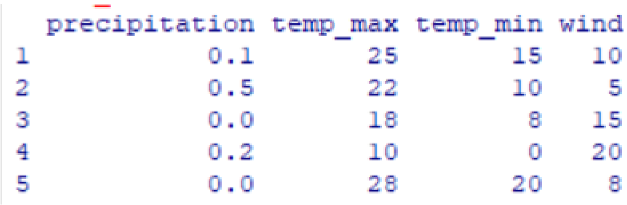
CODE:

```
install.packages("el071")

library(e1071)

install.packages("caTools")

library(caTools)
```

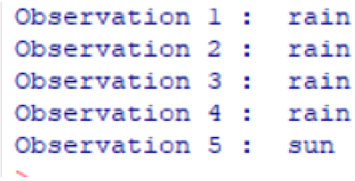
```
# Load your dataset

data <- read.csv("C:\\Users\\HARI RAGHAV\\Downloads\\seattle-weather.csv")

data$weather <- as.factor(data$weather)

set.seed(123) # For reproducibility

split <- sample.split(data$weather, SplitRatio = 0.7)

train_data <- subset(data, split == TRUE)

test_data <- subset(data, split == FALSE)

model <- naiveBayes(weather ~ precipitation + temp_max + temp_min + wind, data = train_data)

predictions <- predict(model, newdata = test_data)

# Create a new data frame with sample data

new_data <- data.frame(

  precipitation = c(0.1, 0.5, 0.0, 0.2, 0.0),  # Example precipitation values

  temp_max = c(25, 22, 18, 10, 28),            # Example max temperature values

  temp_min = c(15, 10, 8, 0, 20),              # Example min temperature values

  wind = c(10, 5, 15, 20, 8)                   # Example wind values

)

# Use the trained model to make predictions on the new data

predictions <- predict(model, newdata = new_data)
```

```
# Print the predicted weather conditions

for (i in 1:length(predictions))

{

        cat("Observation", i, ": ",
levels(predictions)[as.integer(predictions[i])], "\n")

}
```

OUTPUT:



```
  precipitation temp_max temp_min wind
1           0.1       25       15   10
2           0.5       22       10    5
3           0.0       18        8   15
4           0.2       10        0   20
5           0.0       28       20    8
```



```
Observation 1 :  rain
Observation 2 :  rain
Observation 3 :  rain
Observation 4 :  rain
Observation 5 :  sun
```

RESULT:

Thus, a program to implement Simple Naive Bayes classification algorithm for predicting the weather forecast has been successfully executed using R.

Ex No 8: Write a Program to implement K-Means clustering operation and visualize for iris dataset

AIM:

To Write a Program to implement K-Means clustering operation and visualize for iris dataset.

ALGORITHM:

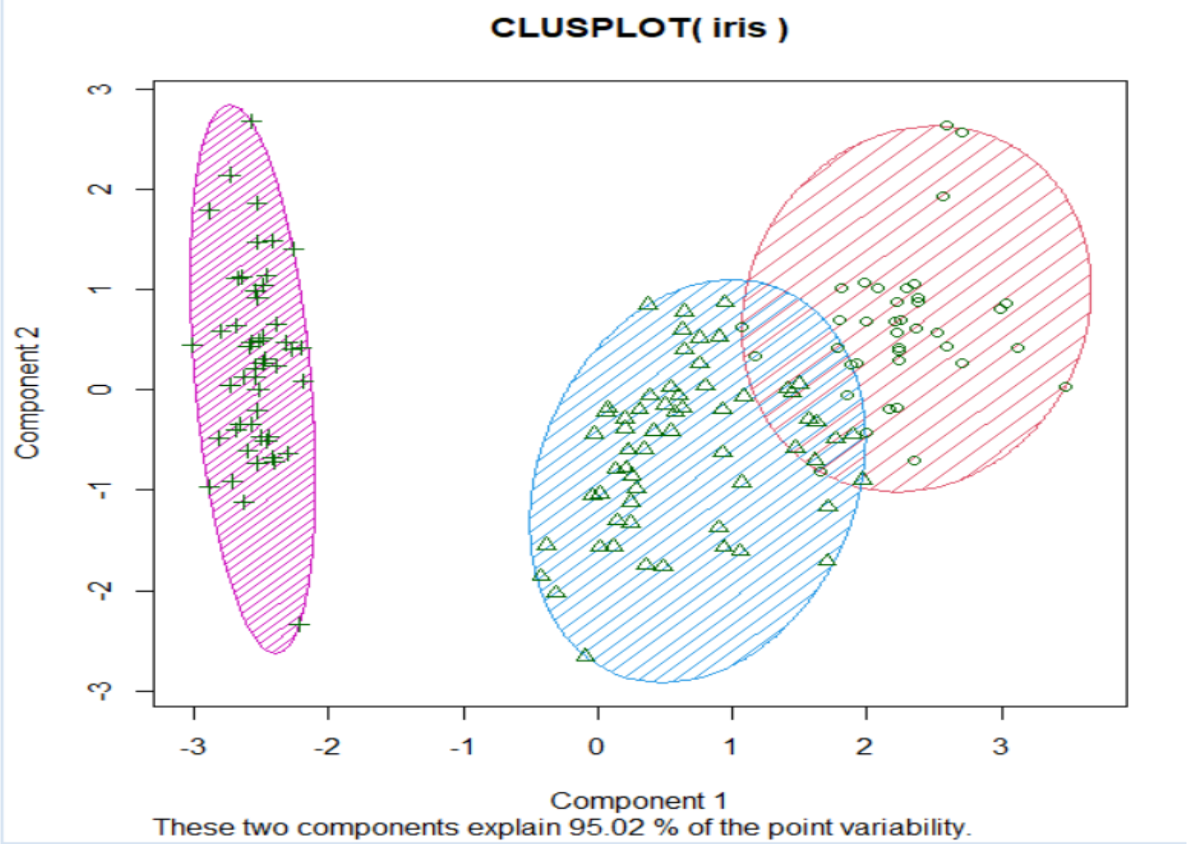
- ❖ Import the necessary libraries, `ggplot2` and `cluster`, for data visualization and clustering, respectively.
- ❖ Data Preparation: Create a dataframe `df` and assign the Iris dataset to it.
- ❖ Data Visualization: Create a scatter plot using `ggplot2`, where `Petal.Length` is on the x-axis, `Petal.Width` is on the y-axis.
- ❖ K-means Clustering: Perform k-means clustering on the numeric columns (1:4) of the Iris dataset with three centers (`center=3`).
- ❖ Cluster Label Assignment: Add cluster labels to the original dataframe `df` based on the clustering results.
- ❖ Create a scatter plot with cluster labels using `ggplot2` & Apply a minimal theme to the plot using `theme_minimal()`.
- ❖ Cluster Analysis and Visualization: Compute and display a contingency table (cross-tabulation) between the cluster assignments and the actual species
- ❖ Visualize the clusters using a cluster plot with `clusplot` from the `cluster` library.
- ❖ Performing elbow Method for Optimal Cluster Number Selection

CODE:

```
library(ggplot2)
df <- iris
head(iris)
ggplot(df, aes(Petal.Length, Petal.Width)) +
geom_point(aes(col=Species), size=4)
set.seed(101)
irisCluster <- kmeans(df[,1:4], center=3, nstart=20)
irisCluster
# Add cluster labels to the original dataframe
df$Cluster <- as.factor(irisCluster$cluster)
# Create a scatter plot with cluster labels
ggplot(df, aes(Petal.Length, Petal.Width, color = Cluster)) +
  geom_point(size = 4) +
  scale_color_manual(values = c("red", "blue", "green")) +  # Customize cluster colors
  labs(color = "Cluster") +
  theme_minimal()
table(irisCluster$cluster, df$Species)
library(cluster)
clusplot(iris, irisCluster$cluster, color=T, shade=T, labels=0, lines=0)
```

```
tot.withinss <- vector(mode="character", length=10)

for (i in 1:10){

  irisCluster <- kmeans(df[,1:4], center=i, nstart=20)

  tot.withinss[i] <- irisCluster$tot.withinss

}

plot(1:10, tot.withinss, type="b", pch=19)
```

OUTPUT:



RESULT:

Thus, a program to implement K-Means clustering operation and visualize for iris dataset has been successfully executed using R.

Ex No 9: Write a Program to diagnose any disease using KNN classification and plot the results

AIM:

To diagnose any disease using KNN classification and plot the results in R.

ALGORITHM:

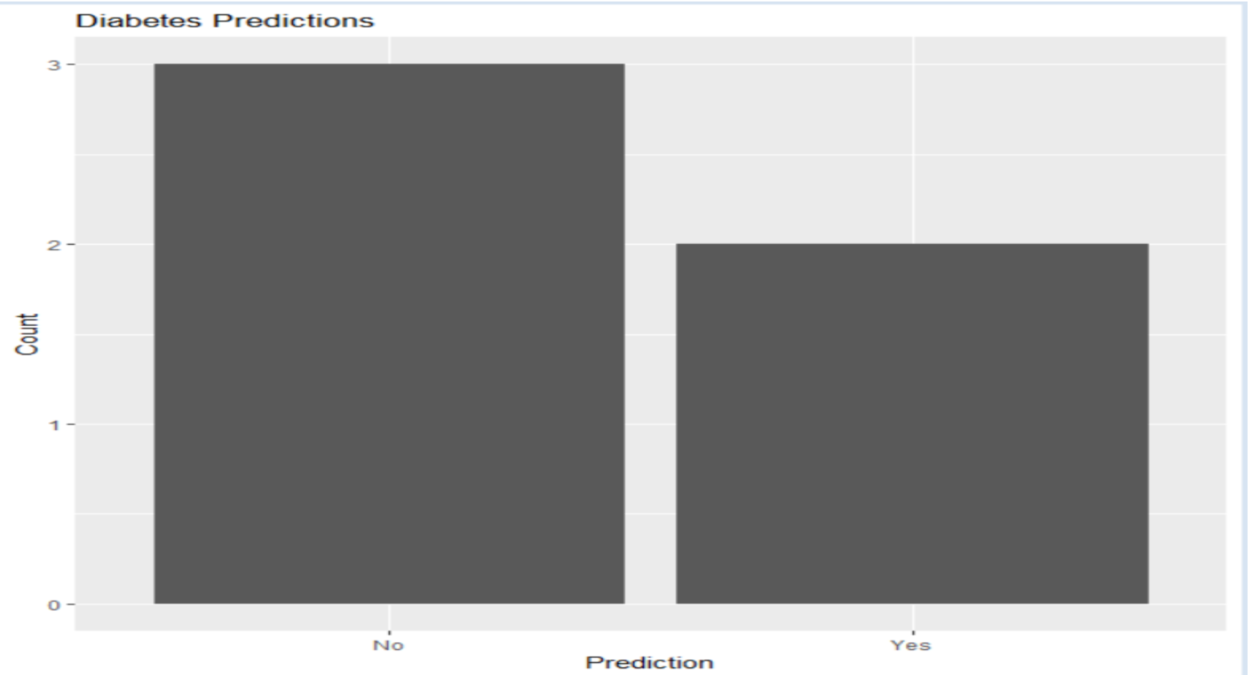
- ❖ Load the "class," "caret," "e1071," and "ggplot2" libraries for various functions and data visualization.
- ❖ Read a dataset (assumed to be in CSV format) named "diabetes.csv" into a variable called "data."
- ❖ Split the dataset into training and testing sets
- ❖ Set the number of neighbors (K) to 5.
- ❖ Build KNN Model: Use the "train" function from the "caret" package to build a KNN classification model & Specify "Outcome" as the target variable, and use all other columns as features for prediction.
- ❖ Create a new data frame named "new_data" containing information for five hypothetical individuals.
- ❖ Apply the previously trained KNN model to the "new_data" to make predictions for each of the five hypothetical individuals.
- ❖ Convert the numeric predictions into "Yes" or "No" labels based on a threshold of 0.5.
- ❖ Create a bar chart using "ggplot2" to visualize the count of "Yes" and "No" predictions.

CODE:

```
# Load necessary libraries
library(class)
# Load the dataset
data <- read.csv("E:\\Semester Notes\\7th sem\\Data Analytics Lab\\diabetes.csv")
# Split the dataset into training and testing sets
set.seed(123) # for reproducibility
sample_indices <- sample(nrow(data), size = 0.7 * nrow(data))
train_data <- data[sample_indices, ]
test_data <- data[-sample_indices, ]
# Define the number of neighbors (K) - you can tune this value
K <- 5
library(caret)
library(e1071)
# Assuming 'trainData' is your training dataset
knn_model <- train(Outcome ~ ., data = train_data, method = "knn",
trControl = trainControl(method = "cv", number = 5))
# Create a new data frame with the same column names
new_data <- data.frame(
Pregnancies = c(2, 5, 1, 7, 0),
Glucose = c(120, 160, 90, 200, 80),
BloodPressure = c(70, 80, 60, 90, 70),
SkinThickness = c(25, 30, 20, 35, 10),
Insulin = c(80, 120, 40, 200, 0),
BMI = c(27.5, 32.1, 25.0, 28.6, 21.4),
DiabetesPedigreeFunction = c(0.245, 0.625, 0.186, 0.956, 0.402),
Age = c(30, 45, 28, 54, 23)
)
# Print the new data frame
print(new_data)
# Make sure 'new_data' is properly formatted with the same column names as your training data
# Exclude the 'Outcome' column since you're predicting it
new_data <- new_data[, names(train_data) != "Outcome"]
# Step 2: Apply the KNN model to the new dataset
new_predictions <- predict(knn_model, newdata = new_data)
# Step 3: Convert numeric predictions to "Yes" or "No" based on the threshold
threshold <- 0.5
diabetes_predictions <- ifelse(new_predictions >= threshold, "Yes", "No")
# Print the predictions
```

```
print(diabetes_predictions)

install.packages("ggplot2")

library(ggplot2)

predictions_df <- data.frame(Prediction = diabetes_predictions)

# Create a bar chart

ggplot(predictions_df, aes(x = Prediction)) +

  geom_bar() +

  labs(title = "Diabetes Predictions", x = "Prediction", y = "Count")
```

OUTPUT:



RESULT:

Thus, a program to diagnose any disease using KNN classification and plotting the results has been successfully executed using R.

Ex No 10: Create the following visualization plots for the movie recommendations system

1. Bar, Pie, Box and scatter plot.

2. Find the outliers using plot

AIM:

To visualize bar, pie, box, scatter plot and to find the outliers using plots for a movie recommendation system.

ALGORITHM:

- ❖ Data Preparation:
  - ➢ Create a data frame named `data` containing movie information.
  - ➢ Preprocess the `tagline` column by converting text to lowercase, removing punctuation, numbers, stopwords, and extra whitespace.
  - ➢ Create a Document-Term Matrix (DTM) from the preprocessed text.
- ❖ Define a Function to Get Similarity Scores:
  - ➢ Create a function called `get_single_movie_recommendation` that takes a target movie name as input.
  - ➢ Find the row index of the target movie in the dataset.
  - ➢ Calculate cosine similarity between the target movie and all other movies based on the DTM.
  - ➢ Create a data frame with movie titles and similarity scores.
  - ➢ Sort the data frame by similarity and exclude the target movie.
  - ➢ Get the top recommended movie based on similarity and return its title.

- Data Visualization: Install and load necessary R packages, including `tm`, `proxy`, and `ggplot2`.
- Box Plot Visualization: Create a box plot to visualize the distribution of similarity scores for all movies.
- Scatter Plot Visualization: Create a scatter plot to show cosine similarity scores between the target movie and other movies.
- Bar Chart Visualization: Create a bar chart with reordered movie titles based on similarity scores.
- Pie Chart Visualization for Recommendations:
    - Get similarity scores for all movies using the `get_single_movie_recommendation` function.
    - Create a pie chart to display recommended movies based on similarity to the target movie.
- Outlier Detection and Visualization:
    - Get similarity scores for all movies relative to a target movie using the **get_single_movie_recommendation** function.
    - Create a box plot to visualize the distribution of similarity scores.
    - Customize the plot to make outliers transparent.
    - Identify the movie with the maximum similarity score and highlight it as an outlier.
    - This is done by adding a red dot to the box plot at the position of the maximum similarity score.

CODE:

```
data <- data.frame(

movie_id = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),

title = c(
"The Shawshank Redemption", "The Godfather", "Pulp Fiction", "The Dark Knight", "Schindler's List",
"Forrest Gump", "The Matrix", "Inception", "Fight Club", "The Silence of the Lambs"
),
tagline = c(
"Fear can hold you prisoner. Hope can set you free.", "An offer you can't refuse.","Just because you are a character doesn't mean you have character.", "Why So Serious?",
"The list is an absolute good. The list is life.", "Life is like a box of chocolates...",
"Welcome to the Real World.", "Your mind is the scene of the crime.", "Mischief. Mayhem. Soap.",
"To enter the mind of a killer, she must challenge the mind of a madman."
),
vote_average = c(9.3, 9.2, 8.9, 9.0, 8.9, 8.8, 8.7, 8.8, 8.9, 8.6),
vote_count = c(13878, 10121, 19354, 24352, 8340, 18847, 17286, 24178, 18123, 16341)
)
# Preprocess text-based columns
install.packages("tm")
library(tm)
corpus <- Corpus(VectorSource(data$tagline))
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removeWords, stopwords("en"))
corpus <- tm_map(corpus, stripWhitespace)
dtm <- DocumentTermMatrix(corpus)

# Function to get similarity scores for all movies (excluding the target movie)
get_similarity_scores <- function(target_movie_name) {
# Find the row index corresponding to the target movie by name
target_movie_index <- which(data$title == target_movie_name)
if (length(target_movie_index) == 0) {
return(NULL)
}
else {
# Calculate cosine similarity between the target movie and all other movies
target_movie_vector <- as.matrix(dtm)[target_movie_index, , drop = FALSE]
similarity_scores <- proxy::simil(as.matrix(dtm), target_movie_vector, method = "cosine")
# Ensure similarity_scores has the same length as the number of movies

similarity_scores <- rep(similarity_scores, nrow(data))

# Create a data frame with movie titles and similarity scores

similarity_df <- data.frame(

movie_title = data$title,

similarity = similarity_scores

)

# Sort by similarity and exclude the target movie

similarity_df <- similarity_df[order(-similarity_df$similarity), ]

similarity_df <- similarity_df[similarity_df$movie_title != target_movie_name, ]

# Get the top recommended movie

top_recommended_movie <- head(similarity_df, 1)

# Return the title of the top recommended movie

return(top_recommended_movie$movie_title)

}

}

# Get a recommended movie for a specific movie ("Forrest Gump" in this example)

install.packages("proxy")

target_movie_name <- "Forrest Gump"
```

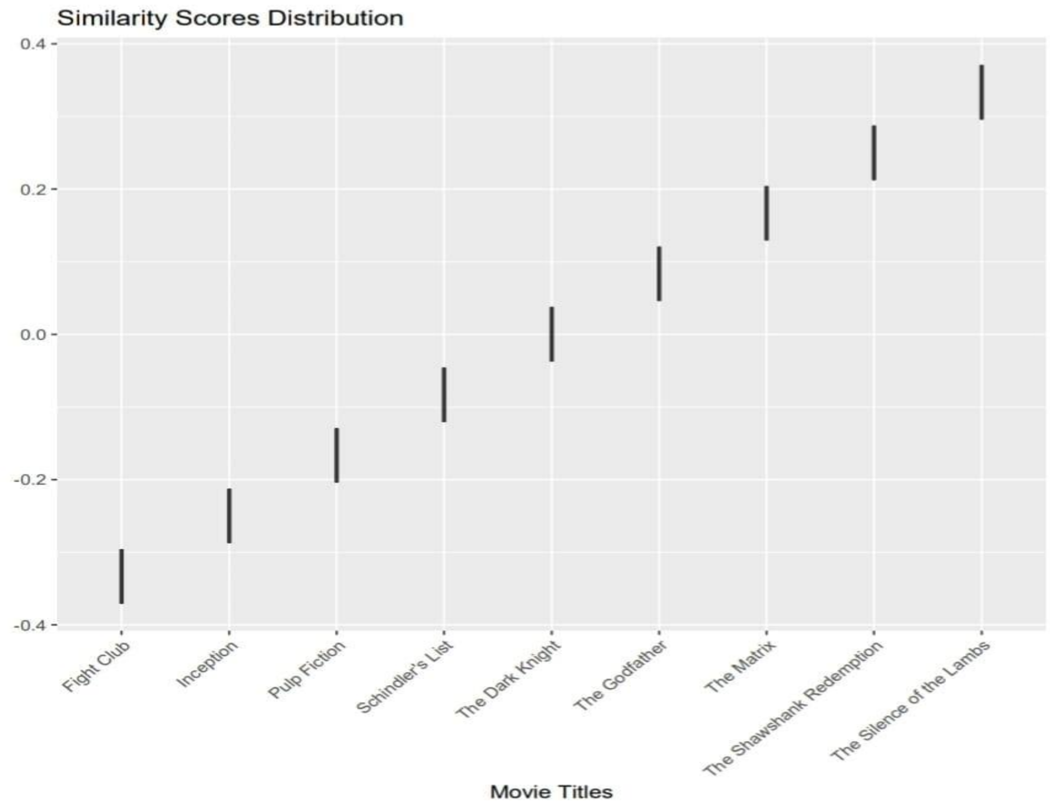
```
recommended_movie <- get_single_movie_recommendation(target_movie_name)
recommended_movie
# Filter the similarity scores for recommended movies only
recommended_movies <- similarity_df[similarity_df$movie_title == recommended_movie, ]
install.packages("ggplot2")
library(ggplot2)
# Create a data frame with movie titles and similarity scores
similarity_df <- data.frame(
movie_title = data$title,
similarity = similarity_scores
)
# Create a data frame for the box plot
boxplot_data <- data.frame(
Movie_Title = similarity_df$movie_title,
Similarity_Score = similarity_df$similarity
)
# Create a box plot with custom x-axis labels and no y-axis label
ggplot(boxplot_data, aes(x = Movie_Title)) +
geom_boxplot() +
labs(title = "Similarity Scores Distribution", x = "Movie Titles") +

theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Create a scatter plot

ggplot(similarity_df, aes(x = movie_title, y = similarity)) +

geom_point(size = 3) +

labs(

title = paste("Movie Similarity to", target_movie_name),

x = "Movie Title",

y = "Cosine Similarity"

) +

theme(axis.text.x = element_text(angle = 90, hjust = 1)) +

coord_flip()

# Load the ggplot2 library

library(ggplot2)

# Calculate cosine similarity between the target movie and all other movies

target_movie_vector <- as.matrix(dtm)[which(data$title == target_movie_name), , drop = FALSE]

similarity_scores <- proxy::simil(as.matrix(dtm), target_movie_vector, method = "cosine")

# Sort the data frame by similarity score in descending order

similarity_df <- similarity_df[order(-similarity_df$similarity), ]
```

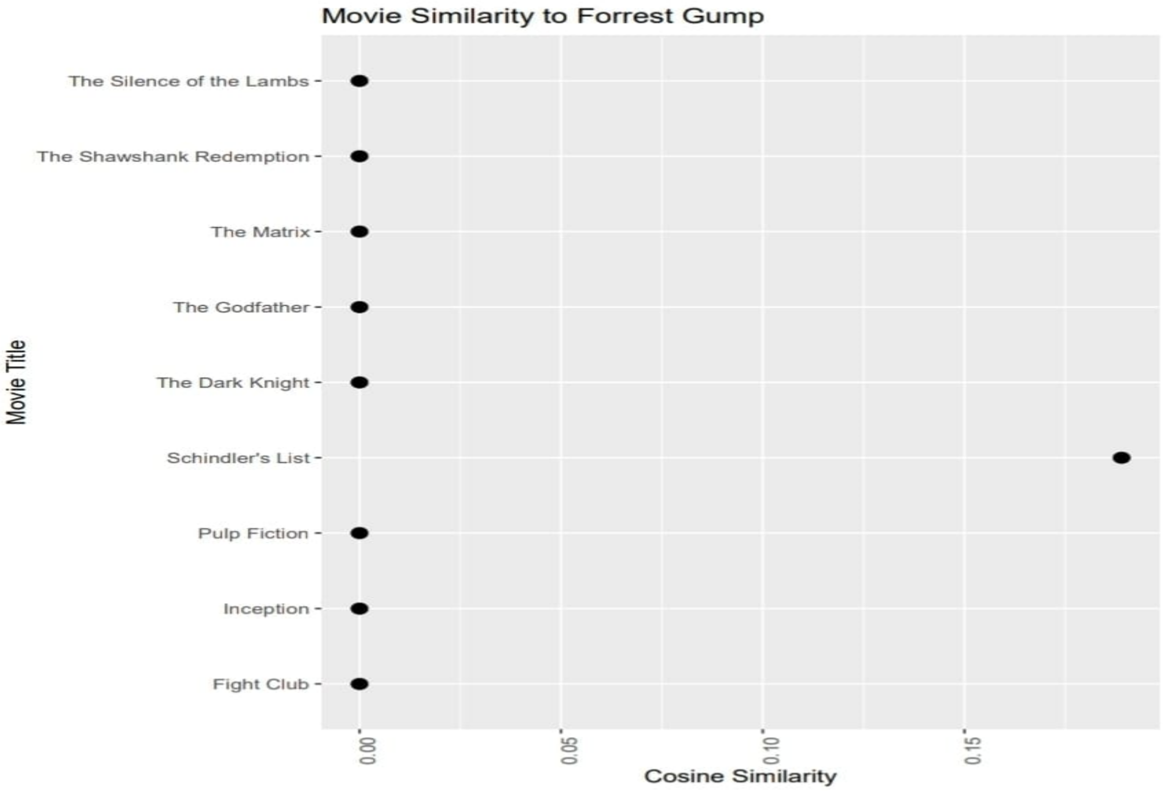
```r
# Exclude the target movie
similarity_df <- similarity_df[similarity_df$movie_title != target_movie_name, ]
# Create a bar chart
ggplot(similarity_df, aes(x = reorder(movie_title, -similarity), y = similarity)) +
geom_bar(stat = "identity", fill = "blue") +
labs(
title = paste("Movie Similarity to", target_movie_name),
x = "Movie Title",
y = "Cosine Similarity"
) +
theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
coord_flip()
# Load the ggplot2 library
library(ggplot2)
# Get similarity scores for all movies
similarity_scores_df <- get_similarity_scores(target_movie_name)
if (!is.null(similarity_scores_df)) {
# Create a pie chart
ggplot(similarity_scores_df, aes(x = "", y = similarity, fill = movie_title)) +
geom_bar(stat = "identity") +
coord_polar(theta = "y") +
labs(
title = paste("Recommended Movies based on Similarity to", target_movie_name),
x = NULL,
y = NULL,
fill = "Movie Title"
) +
theme_void() +
theme(legend.position = "bottom")
}
else {
# Handle the case when the target movie is not found
print("Target movie not found in the dataset.")
}
# Load the ggplot2 library
library(ggplot2)
# Function to get similarity scores for all movies (excluding the target movie)
get_similarity_scores <- function(target_movie_name) {
```

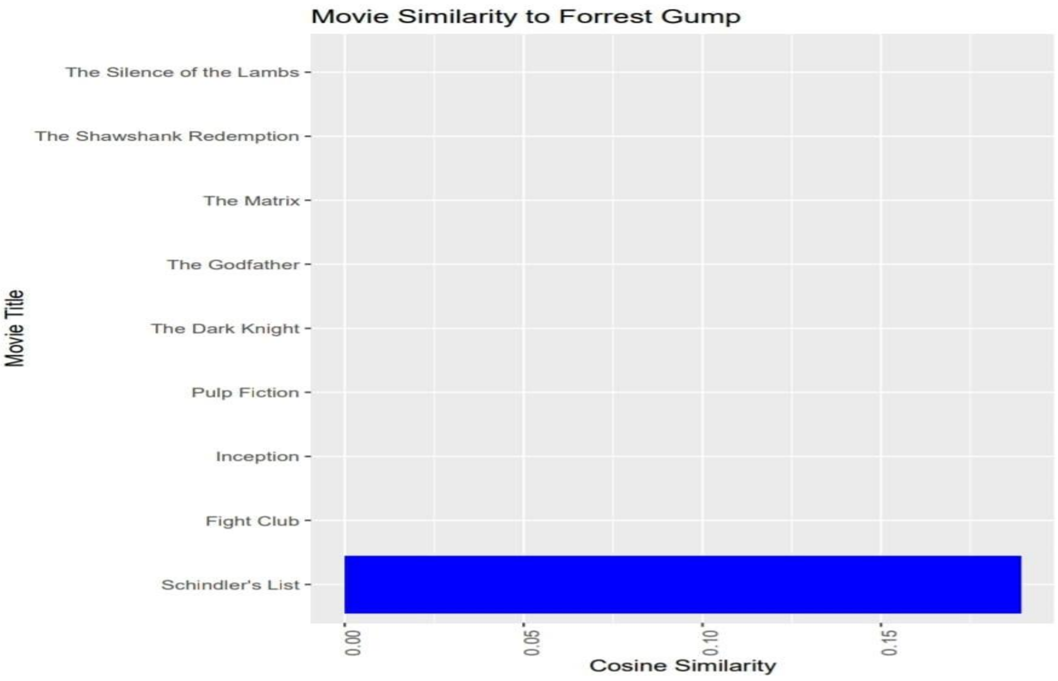
```
# Find the row index corresponding to the target movie by name
target_movie_index <- which(data$title == target_movie_name)
if (length(target_movie_index) == 0) {
return(NULL)
} else {
# Calculate cosine similarity between the target movie and all other movies
target_movie_vector <- as.matrix(dtm)[target_movie_index, , drop = FALSE]
similarity_scores <- proxy::simil(as.matrix(dtm), target_movie_vector, method = "cosine")
# Ensure similarity_scores has the same length as the number of movies
similarity_scores <- rep(similarity_scores, nrow(data))
# Create a data frame with movie titles and similarity scores
similarity_df <- data.frame(
movie_title = data$title,
similarity = similarity_scores
)
# Sort by similarity and exclude the target movie
similarity_df <- similarity_df[order(-similarity_df$similarity), ]
similarity_df <- similarity_df[similarity_df$movie_title != target_movie_name, ]
return(similarity_df)
}
}
# Get similarity scores for all movies
similarity_scores_df <- get_similarity_scores(target_movie_name)
if (!is.null(similarity_scores_df)) {
# Create a box plot and highlight the outlier with a red dot
ggplot(similarity_scores_df, aes(x = reorder(movie_title, -similarity), y = similarity)) +
geom_boxplot(outlier.colour = "transparent") + # Make outliers transparent in the box plot
geom_point(data = similarity_scores_df[which(similarity_scores_df$similarity == max(similarity_scores_df$similarity)), ],
aes(x = reorder(movie_title, -similarity), y = similarity),
color = "red", size = 3) + # Highlight the maximum similarity as an outlier in red
labs(
title = paste("Similarity Scores Distribution to", target_movie_name),
x = "Movie Title",
y = "Cosine Similarity"
) +
theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
coord_flip()
}
else
{
# Handle the case when the target movie is not found
print("Target movie not found in the dataset.")
}
```

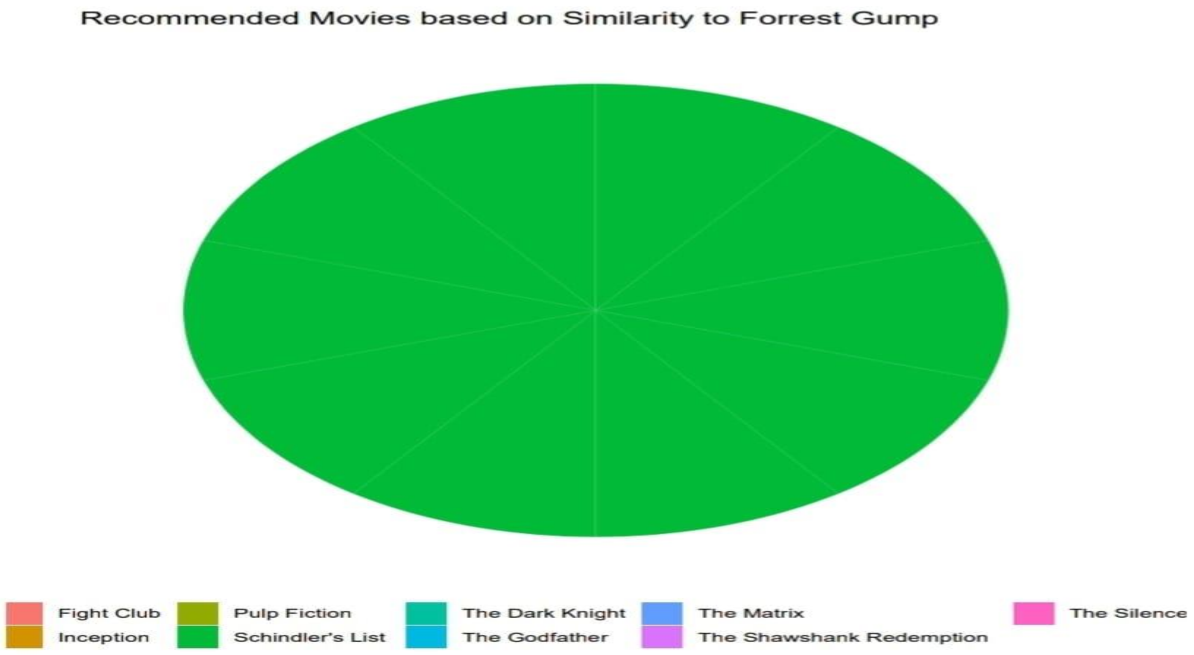
OUTPUT:

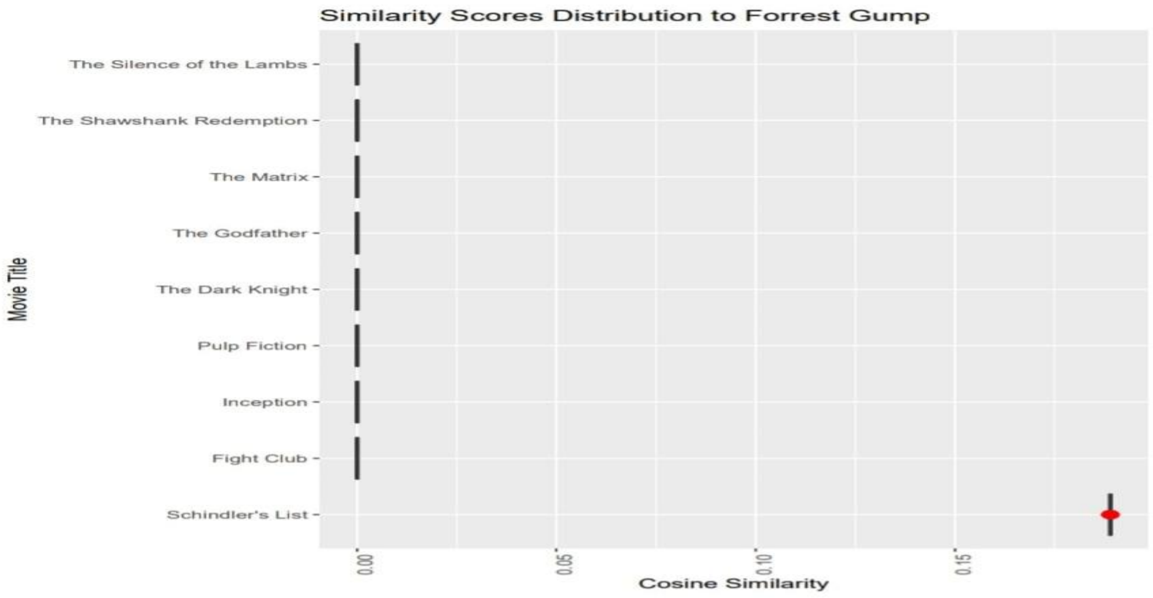
Schindler's List




Movie Similarity to Forrest Gump
Movie Title
The Silence of the Lambs
The Shawshank Redemption
The Matrix
The Godfather
The Dark Knight
Schindler's List
Pulp Fiction
Inception
Fight Club
0.00
0.05
0.10
0.15
Cosine Similarity


Movie Similarity to Forrest Gump
Movie Title
The Silence of the Lambs
The Shawshank Redemption
The Matrix
The Godfather
The Dark Knight
Pulp Fiction
Inception
Fight Club
Schindler's List
0.00
0.05
0.10
0.15
Cosine Similarity





RESULT:

Thus, a program to visualize bar, pie, box, scatter plot and to find the outliers using plots for a movie recommendation system has been successfully executed using R.