Course Title	**–**	**Practical VI: R Programming Lab**
Course Code	**–**	**20YP6**
Semester	**–**	**III**

Exercises

1. To get the input from user and perform numerical operations (MAX, MIN, AVG, SUM, SQRT, ROUND) using in R.
2. To perform data import/export (.CSV, .XLS, .TXT) operations using data frames in R.
3. To get the input matrix from user and perform Matrix addition, subtraction, multiplication, inverse transpose and division operations using vector concept in R.
4. To perform statistical operations (Mean, Median, Mode and Standard deviation) using R.
5. To perform data pre-processing operations: i) Handling Missing data ii) MinMax normalization.
6. To perform dimensionality reduction operation using PCA for Houses Data Set.
7. To perform Simple Linear Regression with R.
8. To perform K-Means clustering operation and visualize for iris data set.
9. Write R script to diagnose any disease using KNN classification and plot the results.
10. To perform market basket analysis using Association Rules (Apriori).

Ex no: 01 Numerical Operations

Aim:

To get the input from user and perform numerical operations (MAX, MIN, AVG, SUM, SQRT, ROUND) using in R.

Program:

```
while (TRUE) {
 cat("MENU - NUMERICAL OPERATIONS\n")
 cat("1. Find Maximum\n")
 cat("2. Find Minimum\n")
 cat("3. Calculate Average\n")
 cat("4. Calculate Sum\n")
 cat("5. Calculate Square Root\n")
 cat("6. Round Numbers\n")
 cat("7. Exit\n")

 choice <- as.integer(readline(prompt = "Enter your choice (1/2/3/4/5/6/7): "))

 if (!(choice %in% c(1:7))) {
  cat("Invalid choice. Please enter 1, 2, 3, 4, 5, 6, or 7.\n\n")
  next
 } else if (choice %in% c(1:6)) {
  input <- readline(prompt = "Enter a list of numbers separated by commas: ")
  input_vector <- as.numeric(unlist(strsplit(input, ",")))
 }

 switch(choice,

     "1" = {cat("Maximum:", max(input_vector), "\n\n")},

     "2" = {cat("Minimum:", min(input_vector), "\n\n")},

     "3" = {cat("Average:", mean(input_vector), "\n\n")},

     "4" = {cat("Sum:", sum(input_vector), "\n\n")},

     "5" = {cat("Square root of values:", sqrt(input_vector), "\n\n")},

     "6" = {cat("Rounded values:", round(input_vector), "\n\n")},

     "7" = {
      cat("Exiting the program.\n\n")
      break
     }
 )
}
```

Output:

```
MENU - NUMERICAL OPERATIONS
1. Find Maximum
2. Find Minimum
3. Calculate Average
4. Calculate Sum
5. Calculate Square Root
6. Round Numbers
7. Exit
Enter your choice (1/2/3/4/5/6/7): 1
Enter a list of numbers separated by commas: 10, 20, 30, 40
Maximum: 40

MENU - NUMERICAL OPERATIONS
1. Find Maximum
2. Find Minimum
3. Calculate Average
4. Calculate Sum
5. Calculate Square Root
6. Round Numbers
7. Exit
Enter your choice (1/2/3/4/5/6/7): 2
Enter a list of numbers separated by commas: 10, 20, 0, -40
Minimum: -40

MENU - NUMERICAL OPERATIONS
1. Find Maximum
2. Find Minimum
3. Calculate Average
4. Calculate Sum
5. Calculate Square Root
6. Round Numbers
7. Exit
Enter your choice (1/2/3/4/5/6/7): 3
Enter a list of numbers separated by commas: 20.5, 20.5, 20.5
Average: 20.5

MENU - NUMERICAL OPERATIONS
1. Find Maximum
2. Find Minimum
3. Calculate Average
4. Calculate Sum
5. Calculate Square Root
6. Round Numbers
7. Exit
Enter your choice (1/2/3/4/5/6/7): 4
Enter a list of numbers separated by commas: 1.1, 2.33, -0.8, -2.2, 0, -80
Sum: -79.57

MENU - NUMERICAL OPERATIONS
1. Find Maximum
2. Find Minimum
3. Calculate Average
4. Calculate Sum
5. Calculate Square Root
6. Round Numbers
7. Exit
Enter your choice (1/2/3/4/5/6/7): 5
```

```
Enter a list of numbers separated by commas: 11, 4, 69, 420
Square root of values: 3.316625 2 8.306624 20.4939

MENU - NUMERICAL OPERATIONS
1. Find Maximum
2. Find Minimum
3. Calculate Average
4. Calculate Sum
5. Calculate Square Root
6. Round Numbers
7. Exit
Enter your choice (1/2/3/4/5/6/7): 6
Enter a list of numbers separated by commas: 0, 78.78, 91.2, 81.5, 73.8
Rounded values: 0 79 91 82 74

MENU - NUMERICAL OPERATIONS
1. Find Maximum
2. Find Minimum
3. Calculate Average
4. Calculate Sum
5. Calculate Square Root
6. Round Numbers
7. Exit
Enter your choice (1/2/3/4/5/6/7): 7
Exiting the program.
```

Ex no: 02 Data Import / Export

Aim:

To perform data import/export (.CSV, .XLS, .TXT) operations using data frames in R.

Program:

```
library(tidyverse)
library(xlsx)

# Import data from CSV file
df1 <- read_csv("Assets\\EX2-Import.csv")
summary(df1)
cat("\n")

# Import data from Excel file
df2 <- read.xlsx("Assets\\EX2-Import.xlsx", sheetIndex = 1)
summary(df2)
cat("\n")

# Import data from TXT file
df3 <- read.table("Assets\\EX2-Import.txt", sep = ",", header = TRUE)
summary(df3)
cat("\n")

# Export data to CSV file
write_csv(df1, "Assets\\EX2-Export.csv")

# Export data to Excel file
write.xlsx(df2, "Assets\\EX2-Export.xlsx", row.names = FALSE)

# Export data to TXT file
write.table(df3, "Assets\\EX2-Export.txt", sep = ",", row.names = FALSE)
```

Output:

EX2-Import.txt

EX2-Import.txt - Notepad

File Edit Format View Help

```
Product,Price,Stock
Phone,20000,10
TV,38000,3
Tablet,28000,6
Earphones,1200,25
Powerbank,1450,18
Laptop,89000,5
```

EX2-Export.txt

EX2-Export.txt - Notepad

File Edit Format View Help

```
"Product","Price","Stock"
"Phone",20000,10
"TV",38000,3
"Tablet",28000,6
"Earphones",1200,25
"Powerbank",1450,18
"Laptop",89000,5
```

EX2-Import.xlsx

	A	B	C
1	Product	Price	Stock
2	Phone	20000	10
3	TV	38000	3
4	Tablet	28000	6
5	Earphones	1200	25
6	Powerbank	1450	18
7	Laptop	89000	5

EX2-Export.xlsx

	A	B	C
1	Product	Price	Stock
2	Phone	20000	10
3	TV	38000	3
4	Tablet	28000	6
5	Earphones	1200	25
6	Powerbanl	1450	18
7	Laptop	89000	5

EX2-Import.csv

	A	B	C
1	Product	Price	Stock
2	Phone	20000	10
3	TV	38000	3
4	Tablet	28000	6
5	Earphones	1200	25
6	Powerbanl	1450	18
7	Laptop	89000	5

EX2-Export.csv

	A	B	C
1	Product	Price	Stock
2	Phone	20000	10
3	TV	38000	3
4	Tablet	28000	6
5	Earphones	1200	25
6	Powerbanl	1450	18
7	Laptop	89000	5

Ex no: 03 Matrix Operations Using Vector Concept

Aim:

To get the input matrix from user and perform Matrix addition, subtraction, multiplication, inverse transpose and division operations using vector concept in R.

Program:

```
print("MATRIX OPERATIONS")
m1 <- matrix(0, nrow = 2, ncol = 2)
m2 <- matrix(0, nrow = 2, ncol = 2)
cat("\n")

for (i in 1:2) {
 for (j in 1:2) {
  cat("Enter matrix 1 [", i, ",", j, "]: \n")
  m1[i, j] = as.integer(readline())
 }
}
cat("\n")


for (k in 1:2) {
 for (l in 1:2) {
  cat("Enter matrix 2 [", k, ",", l, "]: \n")
  m2[k, l] = as.integer(readline())
 }
}
cat("\n")

add <- m1 + m2
sub <- m1 - m2
mul <- m1 %*% m2
div_m1 <- m1 / m2
div_m2 <- m2 / m1
trans_m1 <- t(m1)
trans_m2 <- t(m2)
inv_m1 <- solve(m1)
inv_m2 <- solve(m2)

while (TRUE) {
 cat("1. Addition (M1 + M2)\n")
 cat("2. Subtraction (M1 - M2)\n")
 cat("3. Multiplication (M1 * M2)\n")
 cat("4. Division (M1 / M2)\n")
 cat("5. Division (M2 / M1)\n")
 cat("6. Transpose M1\n")
 cat("7. Transpose M2\n")
 cat("8. Inverse M1\n")
 cat("9. Inverse M2\n")
 cat("10. Exit\n")
```

```
  ch <- as.integer(readline(prompt = "Enter Your Choice: "))
  switch(ch,
      "1" = {print(add)},
      "2" = {print(sub)},
      "3" = {print(mul)},
      "4" = {print(div_m1)},
      "5" = {print(div_m2)},
      "6" = {print(trans_m1)},
      "7" = {print(trans_m2)},
      "8" = {print(inv_m1)},
      "9" = {print(inv_m2)},
      "10" = {
       cat("Exiting the program.\n\n")
       break
      }
  )
  cat("\n")
}
```

Output:

```
[1] "MATRIX OPERATIONS"

Enter matrix 1 [ 1 , 1 ]:
1
Enter matrix 1 [ 1 , 2 ]:
2
Enter matrix 1 [ 2 , 1 ]:
3
Enter matrix 1 [ 2 , 2 ]:
4

Enter matrix 2 [ 1 , 1 ]:
10
Enter matrix 2 [ 1 , 2 ]:
20
Enter matrix 2 [ 2 , 1 ]:
30
Enter matrix 2 [ 2 , 2 ]:
40

1. Addition (M1 + M2)
2. Subtraction (M1 - M2)
3. Multiplication (M1 * M2)
4. Division (M1 / M2)
5. Division (M2 / M1)
6. Transpose M1
7. Transpose M2
8. Inverse M1
9. Inverse M2
10. Exit
Enter Your Choice: 1
```

```
     [,1] [,2]
[1,]   11   22
[2,]   33   44

1. Addition (M1 + M2)
2. Subtraction (M1 - M2)
3. Multiplication (M1 * M2)
4. Division (M1 / M2)
5. Division (M2 / M1)
6. Transpose M1
7. Transpose M2
8. Inverse M1
9. Inverse M2
10. Exit
Enter Your Choice: 2
     [,1] [,2]
[1,]   -9  -18
[2,]  -27  -36

1. Addition (M1 + M2)
2. Subtraction (M1 - M2)
3. Multiplication (M1 * M2)
4. Division (M1 / M2)
5. Division (M2 / M1)
6. Transpose M1
7. Transpose M2
8. Inverse M1
9. Inverse M2
10. Exit
Enter Your Choice: 3
     [,1] [,2]
[1,]   70  100
[2,]  150  220

1. Addition (M1 + M2)
2. Subtraction (M1 - M2)
3. Multiplication (M1 * M2)
4. Division (M1 / M2)
5. Division (M2 / M1)
6. Transpose M1
7. Transpose M2
8. Inverse M1
9. Inverse M2
10. Exit
Enter Your Choice: 4
     [,1] [,2]
[1,]  0.1  0.1
[2,]  0.1  0.1

1. Addition (M1 + M2)
2. Subtraction (M1 - M2)
3. Multiplication (M1 * M2)
4. Division (M1 / M2)
5. Division (M2 / M1)
6. Transpose M1
7. Transpose M2
8. Inverse M1
9. Inverse M2
10. Exit
```

```
Enter Your Choice: 5
     [,1] [,2]
[1,]   10   10
[2,]   10   10

1. Addition (M1 + M2)
2. Subtraction (M1 - M2)
3. Multiplication (M1 * M2)
4. Division (M1 / M2)
5. Division (M2 / M1)
6. Transpose M1
7. Transpose M2
8. Inverse M1
9. Inverse M2
10. Exit
Enter Your Choice: 6
     [,1] [,2]
[1,]    1    3
[2,]    2    4

1. Addition (M1 + M2)
2. Subtraction (M1 - M2)
3. Multiplication (M1 * M2)
4. Division (M1 / M2)
5. Division (M2 / M1)
6. Transpose M1
7. Transpose M2
8. Inverse M1
9. Inverse M2
10. Exit
Enter Your Choice: 7
     [,1] [,2]
[1,]   10   30
[2,]   20   40

1. Addition (M1 + M2)
2. Subtraction (M1 - M2)
3. Multiplication (M1 * M2)
4. Division (M1 / M2)
5. Division (M2 / M1)
6. Transpose M1
7. Transpose M2
8. Inverse M1
9. Inverse M2
10. Exit
Enter Your Choice: 8
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5

1. Addition (M1 + M2)
2. Subtraction (M1 - M2)
3. Multiplication (M1 * M2)
4. Division (M1 / M2)
5. Division (M2 / M1)
6. Transpose M1
7. Transpose M2
8. Inverse M1
9. Inverse M2
```

```
10. Exit
Enter Your Choice: 9
      [,1]  [,2]
[1,] -0.20  0.10
[2,]  0.15 -0.05


1. Addition (M1 + M2)
2. Subtraction (M1 - M2)
3. Multiplication (M1 * M2)
4. Division (M1 / M2)
5. Division (M2 / M1)
6. Transpose M1
7. Transpose M2
8. Inverse M1
9. Inverse M2
10. Exit
Enter Your Choice: 10
Exiting the program.
```

Ex no: 04 Statistical Operations

Aim:

To perform statistical operations (Mean, Median, Mode and Standard deviation) using R.

Program:

```
# Read input vector from the user
cat("Enter a vector of numbers separated by spaces (e.g., 1 2 3 4 5): ")
data_vector <- scan()

while (TRUE) {
 cat("Enter your choice:\n")
 cat("1. Calculate Mean\n")
 cat("2. Calculate Median\n")
 cat("3. Calculate Mode\n")
 cat("4. Calculate Standard Deviation\n")
 cat("5. Exit\n")

 choice <- as.integer(readline(prompt = "Enter your choice (1/2/3/4/5): "))

 if (choice == 5) {
  cat("Exiting the program.\n")
  break
 }

 switch(choice,
     "1" = {cat("Mean:", mean(data_vector), "\n")},

     "2" = {
      sort(data_vector)
      cat("Median:", median(data_vector), "\n")
     },

     "3" = {
      mode_value <- as.numeric(names(sort(table(data_vector), decreasing = TRUE)[1]))
      if (!is.na(mode_value)) {
       cat("Mode: ", mode_value, "\n")
      } else {
       cat("Mode: No unique mode exists.\n")
      }
     },

     "4" = {cat("Standard Deviation:", sd(data_vector), "\n")}
 )
}
```

Output:

```
Enter a vector of numbers separated by spaces (e.g., 1 2 3 4 5):
1: 40 20 20 10 -10 5 5 5 5
10:
Read 9 items
Enter your choice:
1. Calculate Mean
2. Calculate Median
3. Calculate Mode
4. Calculate Standard Deviation
5. Exit
Enter your choice (1/2/3/4/5): 1
Mean: 11.11111
Enter your choice:
1. Calculate Mean
2. Calculate Median
3. Calculate Mode
4. Calculate Standard Deviation
5. Exit
Enter your choice (1/2/3/4/5): 2
Median: 5
Enter your choice:
1. Calculate Mean
2. Calculate Median
3. Calculate Mode
4. Calculate Standard Deviation
5. Exit
Enter your choice (1/2/3/4/5): 3
Mode:  5
Enter your choice:
1. Calculate Mean
2. Calculate Median
3. Calculate Mode
4. Calculate Standard Deviation
5. Exit
Enter your choice (1/2/3/4/5): 4
Standard Deviation: 14.09295
Enter your choice:
1. Calculate Mean
2. Calculate Median
3. Calculate Mode
4. Calculate Standard Deviation
5. Exit
Enter your choice (1/2/3/4/5): 5
Exiting the program.
```

Ex no: 05 Data Pre-Processing Operations

Aim:

To perform data pre-processing operations: i) Handling missing data ii) MinMax normalization.

(i) Handling missing Data

Program:

```
library(zoo)

# Create a synthetic dataset with missing values
data <- data.frame(
  ID = 1:10,
  Age = c(25, 28, NA, 32, 30, 35, NA, 40, 42, 45),
  Height = c(170, NA, 165, 178, 175, NA, 180, NA, 185, 190),
  Weight = c(70, 75, 68, NA, 85, 90, 78, 88, NA, 92)
)

cat("Original Dataset:\n")
print(data)

cat("\nData Preprocessing:\n")

# 1. Removing Rows with Missing Values
cat("1. Removing Rows with Missing Values:\n")
cleaned_data_1 <- na.omit(data)
print(cleaned_data_1)
cat("\n")

# 2. Filling Missing Values with a Specific Value (e.g., 0)
cat("2. Filling Missing Values with 0:\n")
cleaned_data_2 <- data
cleaned_data_2[is.na(cleaned_data_2)] <- 0
print(cleaned_data_2)
cat("\n")

# 3. Filling Missing Values with Mean of the Column
cat("3. Filling Missing Values with Mean of the Column:\n")
cleaned_data_3 <- data

# Loop through each column
for (col in colnames(cleaned_data_3)) {
  # Find the mean of the column, excluding missing values
  col_mean <- mean(cleaned_data_3[, col], na.rm = TRUE)

  # Identify missing values in the column and replace them with the mean
  missing_values <- is.na(cleaned_data_3[, col])
  cleaned_data_3[missing_values, col] <- col_mean
}
print(cleaned_data_3)
cat("\n")
```

```
# 4. Interpolation (Linear)
cat("4. Interpolation (Linear):\n")
cleaned_data_4 <- na.approx(data)
print(cleaned_data_4)
cat("\n")
```

Output:

```
Original Dataset:
   ID Age Height Weight
1   1  25    170     70
2   2  28     NA     75
3   3  NA    165     68
4   4  32    178     NA
5   5  30    175     85
6   6  35     NA     90
7   7  NA    180     78
8   8  40     NA     88
9   9  42    185     NA
10 10  45    190     92

Data Preprocessing:
1. Removing Rows with Missing Values:
   ID Age Height Weight
1   1  25    170     70
5   5  30    175     85
10 10  45    190     92

2. Filling Missing Values with 0:
   ID Age Height Weight
1   1  25    170     70
2   2  28      0     75
3   3   0    165     68
4   4  32    178      0
5   5  30    175     85
6   6  35      0     90
7   7   0    180     78
8   8  40      0     88
9   9  42    185      0
10 10  45    190     92

3. Filling Missing Values with Mean of the Column:
   ID    Age   Height Weight
1   1 25.000 170.0000  70.00
2   2 28.000 177.5714  75.00
3   3 34.625 165.0000  68.00
4   4 32.000 178.0000  80.75
5   5 30.000 175.0000  85.00
6   6 35.000 177.5714  90.00
7   7 34.625 180.0000  78.00
8   8 40.000 177.5714  88.00
9   9 42.000 185.0000  80.75
10 10 45.000 190.0000  92.00
```

```
4. Interpolation (Linear):
      ID  Age Height Weight
 [1,]  1 25.0  170.0   70.0
 [2,]  2 28.0  167.5   75.0
 [3,]  3 30.0  165.0   68.0
 [4,]  4 32.0  178.0   76.5
 [5,]  5 30.0  175.0   85.0
 [6,]  6 35.0  177.5   90.0
 [7,]  7 37.5  180.0   78.0
 [8,]  8 40.0  182.5   88.0
 [9,]  9 42.0  185.0   90.0
[10,] 10 45.0  190.0   92.0
```

(ii) MinMax normalization

Program:

```
# Create a synthetic dataset
data <- data.frame(
 ID = 1:10,
 Age = c(25, 28, 32, 30, 35, 40, 42, 45, 50, 55),
 Height = c(150, 160, 170, 175, 180, 185, 190, 160, 170, 155),
 Weight = c(50, 55, 60, 65, 70, 75, 80, 85, 90, 95)
)

# Display the original dataset
cat("Original Dataset:\n")
print(data)

# Min-max normalization function
minmax_normalize <- function(x) {
 min_val <- min(x, na.rm = TRUE)
 max_val <- max(x, na.rm = TRUE)
 return((x - min_val) / (max_val - min_val))
}

# Normalize the numeric columns (Age, Height, Weight)
normalized_data <- data
numeric_columns <- c("Age", "Height", "Weight")
for (col in numeric_columns) {
 normalized_data[, col] <- minmax_normalize(data[, col])
}

# Display the normalized dataset
cat("\nMin-Max Normalized Dataset:\n")
print(normalized_data)
```

Output:

```
Original Dataset:
   ID Age Height Weight
1   1  25    150     50
2   2  28    160     55
3   3  32    170     60
4   4  30    175     65
5   5  35    180     70
6   6  40    185     75
7   7  42    190     80
8   8  45    160     85
9   9  50    170     90
10 10  55    155     95

Min-Max Normalized Dataset:
   ID       Age Height    Weight
1   1 0.0000000  0.000 0.0000000
2   2 0.1000000  0.250 0.1111111
3   3 0.2333333  0.500 0.2222222
4   4 0.1666667  0.625 0.3333333
5   5 0.3333333  0.750 0.4444444
6   6 0.5000000  0.875 0.5555556
7   7 0.5666667  1.000 0.6666667
8   8 0.6666667  0.250 0.7777778
9   9 0.8333333  0.500 0.8888889
10 10 1.0000000  0.125 1.0000000
```

Ex no: 06 Dimensionality Reduction Using PCA

Aim:

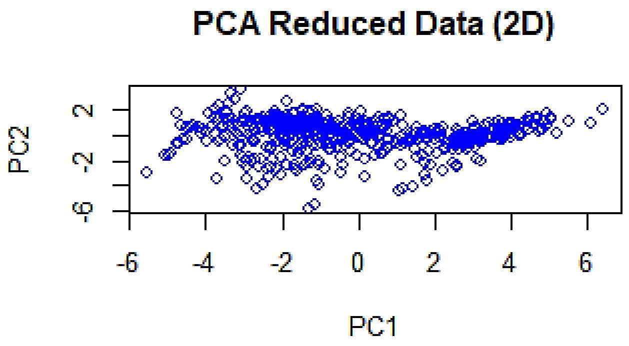
To perform dimensionality reduction operation using PCA for Houses Data Set.

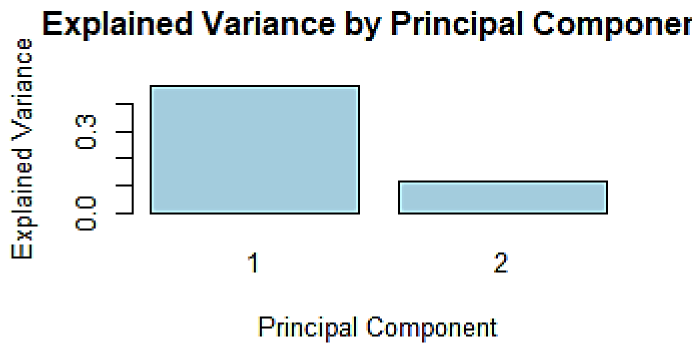
Program:

```
library(MASS)
data("Boston")

# Standardize the data
standardized_data <- scale(Boston)

# Calculate the covariance matrix
cov_matrix <- cov(standardized_data)

# Calculate eigenvalues and eigenvectors
eigen_result <- eigen(cov_matrix)
eigenvalues <- eigen_result$values
eigenvectors <- eigen_result$vectors

# Set the number of components to retain (e.g., 2 for visualization)
num_components <- 2

# Project data onto the selected principal components
reduced_data <- standardized_data %*% eigenvectors[, 1:num_components]

# Calculate explained variance
explained_variance <- sum(eigenvalues[1:num_components]) / sum(eigenvalues)

# Print explained variance
cat("Explained Variance:", explained_variance, "\n")

# Create a 2x2 grid of plots
par(mfrow = c(2, 2))

# Plot 1: Scatterplot of PC1 vs. PC2
plot(reduced_data[, 1], reduced_data[, 2], xlab = "PC1", ylab = "PC2",
   main = "PCA Reduced Data (2D)", col = "blue")

# Plot 2: Explained Variance Bar Plot
barplot(eigenvalues[1:num_components] / sum(eigenvalues),
     names.arg = 1:num_components,
     xlab = "Principal Component", ylab = "Explained Variance",
     main = "Explained Variance by Principal Component", col = "lightblue")
```

Output:

Explained Variance: 0.5853944





Ex no: 07 Linear Regression

Aim:

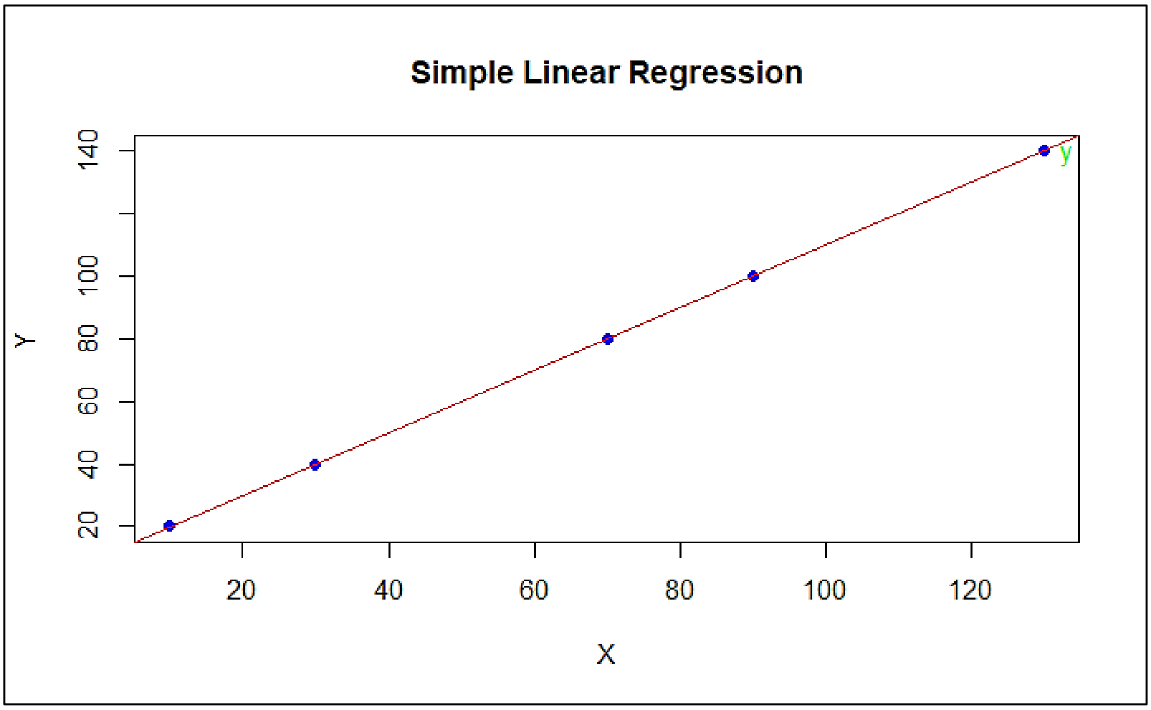
To perform Simple Linear Regression with R.

Program:

```
# Simple Linear Regression with User Input

# Prompt the user for input
cat("Enter the number of data points: ")
num_points <- as.numeric(readline())

# Initialize vectors to store input data
x <- numeric(num_points)
y <- numeric(num_points)

# Get input data from the user
for (i in 1:num_points) {
  cat("Enter x value for data point ", i, ": ")
  x[i] <- as.numeric(readline())

  cat("Enter y value for data point ", i, ": ")
  y[i] <- as.numeric(readline())
}

# Perform simple linear regression
model <- lm(y ~ x)

# Get slope and intercept from the model
slope <- coef(model)[2]
intercept <- coef(model)[1]

# Print slope and intercept
cat("\nSlope: ", slope, "\n")
cat("Intercept: ", intercept, "\n")

# Plot the data points and the regression line
plot(x, y, main="Simple Linear Regression", xlab="X", ylab="Y", pch=16, col="blue")
abline(model, col="red")

# Add the regression line equation to the plot
equation <- paste("y =", round(intercept, 2), "+", round(slope, 2), "x")
text(max(x), max(y), equation, pos=4, col="green")
```

Output:

```
Enter the number of data points:
5
Enter x value for data point  1 :
10
Enter y value for data point  1 :
20
Enter x value for data point  2 :
30
Enter y value for data point  2 :
40
Enter x value for data point  3 :
70
Enter y value for data point  3 :
80
Enter x value for data point  4 :
90
Enter y value for data point  4 :
100
Enter x value for data point  5 :
130
Enter y value for data point  5 :
140

Slope:   1
Intercept:   10
```



Ex no: 08 Clustering Operation

Aim:

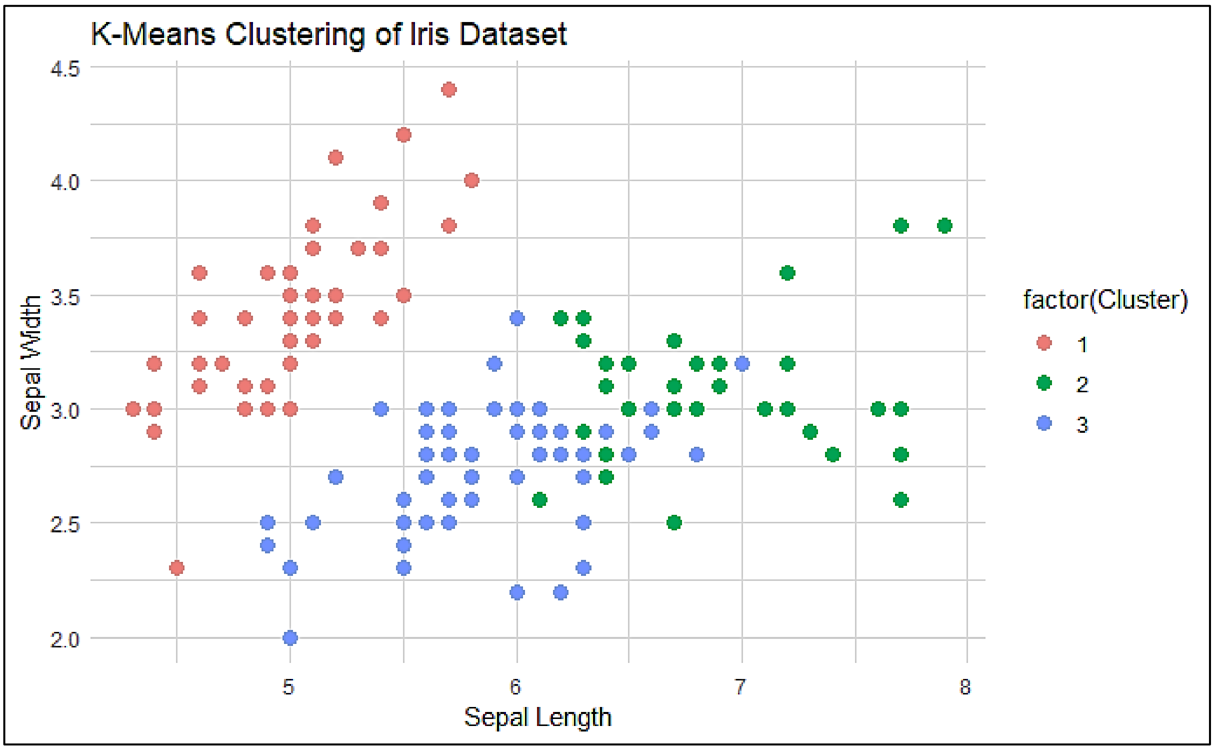
To perform K-Means clustering operation and visualize for iris data set.

Program:

```
# K-Means Clustering on Iris Dataset

# Load the Iris dataset
data(iris)

# Selecting columns for clustering (excluding species column)
iris_features <- iris[, 1:4]

# Prompt the user for the number of clusters
k <- 3

# Perform K-Means clustering
kmeans_model <- kmeans(iris_features, centers = k)

# Print the cluster centers
cat("\nCluster Centers:\n")
print(kmeans_model$centers)

# Add cluster assignments to the original dataset
iris_clustered <- cbind(iris, Cluster = kmeans_model$cluster)

# Visualize the clusters
library(ggplot2)

# Scatter plot of Sepal Length vs Sepal Width with colors representing clusters
cluster_plot = ggplot(iris_clustered, aes(x = Sepal.Length, y = Sepal.Width, color = factor(Cluster)))
+
  geom_point(size = 3) +
  ggtitle("K-Means Clustering of Iris Dataset") +
  xlab("Sepal Length") +
  ylab("Sepal Width") +
  theme_minimal()

print(cluster_plot)
```

Output:

```
Cluster Centers:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     6.850000    3.073684     5.742105    2.071053
2     5.006000    3.428000     1.462000    0.246000
3     5.901613    2.748387     4.393548    1.433871
```



Ex no: 09 Diagnose Any Disease Using KNN Classification

Aim:

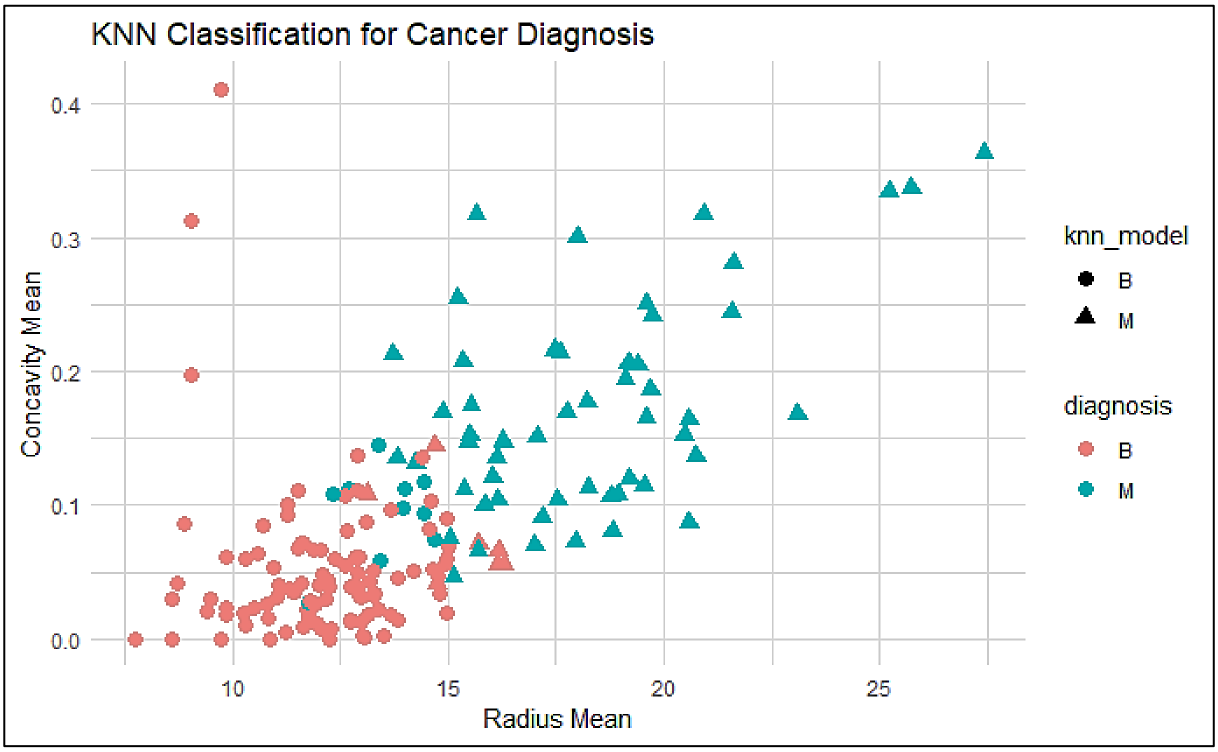
To write an R script to diagnose any disease using KNN classification and plot the results.

Program:

```
# Load necessary libraries
library(class)
library(ggplot2)
library(caret)

# Load the dataset
cancer_data <- read.csv("Assets\\EX9-CancerData.csv")

# Display the structure of the dataset
# str(cancer_data)

# Split the dataset into training and testing sets
set.seed(123)
train_indices <- createDataPartition(cancer_data$diagnosis, p = 0.7, list = FALSE)
train_data <- cancer_data[train_indices, ]
test_data <- cancer_data[-train_indices, ]

# Train the KNN model
k_value <- 3
knn_model <- knn(train = train_data[, c('radius_mean', 'concavity_mean', 'symmetry_mean')],
         test = test_data[, c('radius_mean', 'concavity_mean', 'symmetry_mean')],
         cl = train_data$diagnosis, k = k_value)

# Evaluate the model
conf_matrix <- table(Actual = test_data$diagnosis, Predicted = knn_model)
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)

cat("Confusion Matrix:\n")
print(conf_matrix)

cat("\nAccuracy:", round(accuracy, 2), "\n")

# Plot the results
class_plot = ggplot(test_data, aes(x = radius_mean, y = concavity_mean, color = diagnosis, shape =
knn_model)) +
 geom_point(size = 3) +
 ggtitle("KNN Classification for Cancer Diagnosis") +
 xlab("Radius Mean") +
 ylab("Concavity Mean") +
 theme_minimal()

print(class_plot)
```

Output:

```
Confusion Matrix:
      Predicted
Actual   B   M
     B 100   7
     M  10  53

Accuracy: 0.9
```



Ex no: 10 Market Basket Analysis Using Association Rules (Apriori)

Aim:

To perform market basket analysis using Association Rules (Apriori).

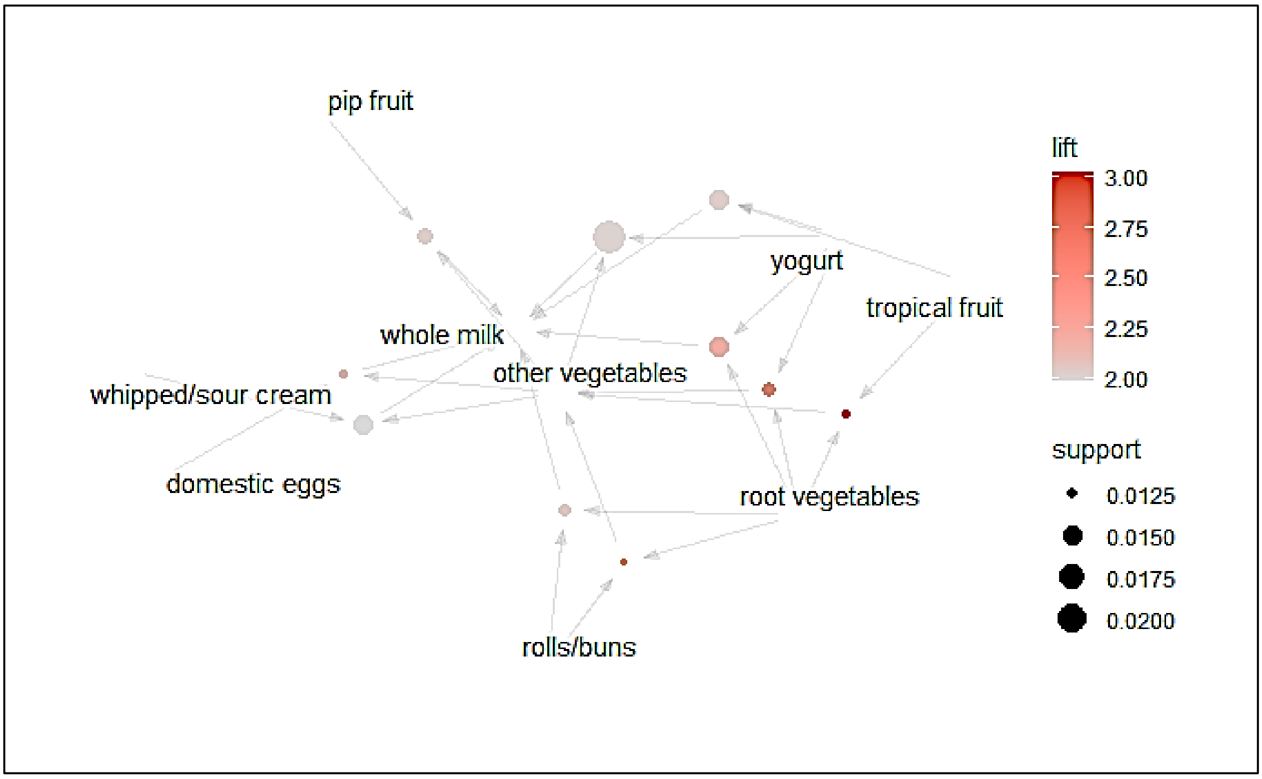
Program:

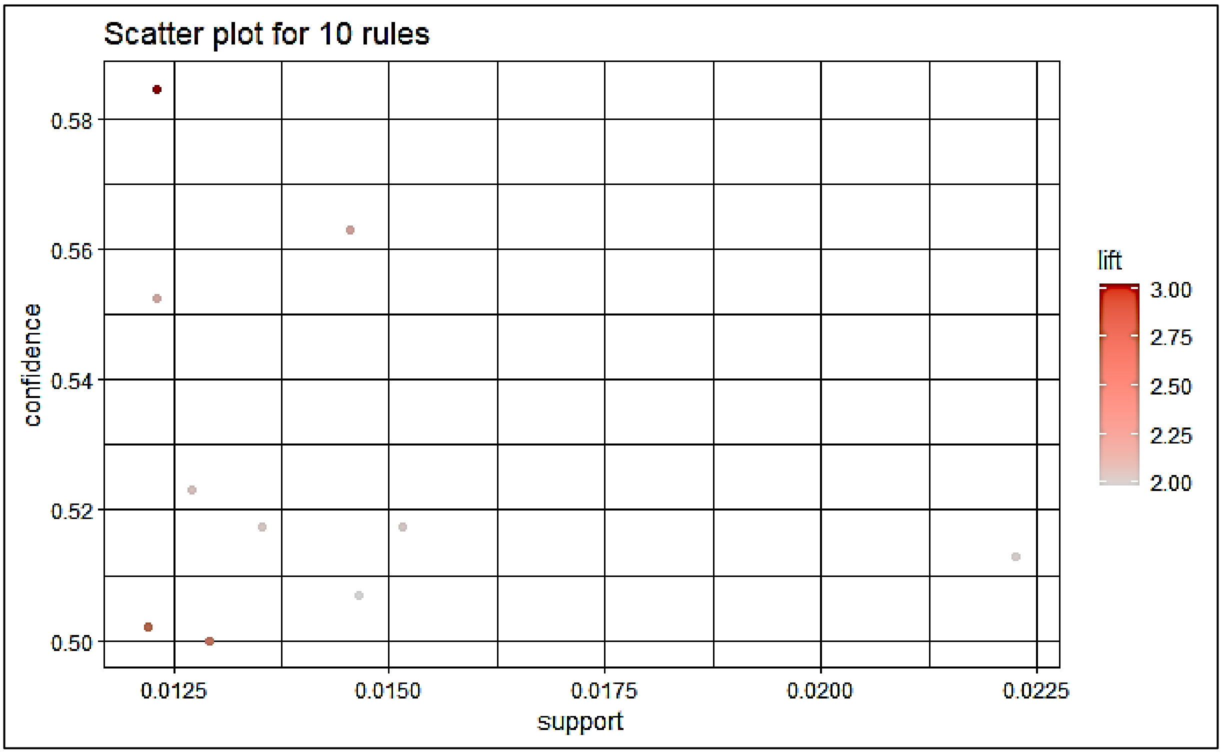
```
# Load necessary libraries
library(arules)
library(arulesViz)

# Load the "Groceries" dataset
data("Groceries")

# Set parameters for the Apriori algorithm
min_support <- 0.001
min_confidence <- 0.5

# Perform Apriori algorithm
rules <- apriori(Groceries, parameter = list(support = min_support, confidence = min_confidence))

# Sort rules by support and confidence
rules <- sort(rules, by = "support", decreasing = TRUE)
rules <- head(rules, 10)

# Display the mined top 10 rules
cat("Top 10 Association Rules:\n")
inspect(rules)

# Plotting the top 10 rules using a graph
cat("\nGraph and Scatterplot of Top 10 Association Rules:\n")
result_1 = plot(rules, method = "graph")
result_2 = plot(rules, method = "scatterplot")

print(result_1)
print(result_2)
```

Output:

```
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
maxlen target  ext
        0.5    0.1    1 none FALSE            TRUE       5   0.001      1
10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE
```

```
Absolute minimum support count: 9

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [157 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 done [0.01s].
writing ... [5668 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
Top 10 Association Rules:
     lhs                                      rhs                support    confidence
[1]  {other vegetables, yogurt}            => {whole milk}       0.02226741 0.5128806
[2]  {tropical fruit, yogurt}              => {whole milk}       0.01514997 0.5173611
[3]  {other vegetables, whipped/sour cream} => {whole milk}      0.01464159 0.5070423
[4]  {root vegetables, yogurt}             => {whole milk}       0.01453991 0.5629921
[5]  {pip fruit, other vegetables}         => {whole milk}       0.01352313 0.5175097
[6]  {root vegetables, yogurt}             => {other vegetables} 0.01291307 0.5000000
[7]  {root vegetables, rolls/buns}         => {whole milk}       0.01270971 0.5230126
[8]  {other vegetables, domestic eggs}     => {whole milk}       0.01230300 0.5525114
[9]  {tropical fruit, root vegetables}     => {other vegetables} 0.01230300 0.5845411
[10] {root vegetables, rolls/buns}         => {other vegetables} 0.01220132 0.5020921
     coverage   lift     count
[1]  0.04341637 2.007235 219
[2]  0.02928317 2.024770 149
[3]  0.02887646 1.984385 144
[4]  0.02582613 2.203354 143
[5]  0.02613116 2.025351 133
[6]  0.02582613 2.584078 127
[7]  0.02430097 2.046888 125
[8]  0.02226741 2.162336 121
[9]  0.02104728 3.020999 121
[10] 0.02430097 2.594890 120

Graph and Scatterplot of Top 10 Association Rules:
```




Scatter plot for 10 rules
confidence
0.58
0.56
0.54
0.52
0.50
support
0.0125
0.0150
0.0175
0.0200
0.0225
lift
3.00
2.75
2.50
2.25
2.00