DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE LAB MANUAL

AD23431 - STATISTICAL ANALYSIS AND COMPUTING

(REGULATION 2023)

RAJALAKSHMI ENGINEERING COLLEGE

Thandalam, Chennai-602015

Name: Harini S

Register No: 231801049

Year / Branch / Section: 2nd / AI&DS / AC

Semester: IV

Academic Year: 2024 - 2025

INDEX

EXP NO: 1	IMPLEMENT SIMPLE PROGRAMS IN R

Aim:

To Implement Simple Programs using R.

Algorithm:

1. Basic Arithmetic Operations

a. Finding Area of Circle

- Input: Read radius r.
- Process: Calculate the area using the formula:
  Area=π×r2\text{Area} = \pi \times r^2Area=π×r2
- Output: Print the calculated area.

2. Control Structures (if-else, for loop)

a. Check Whether the Given Year is Leap or Not

- Input: Read a year ly.
- Process:
  - If ly is divisible by 400, it's a leap year.
  - Else, if divisible by 100 (but not by 400), it's not a leap year. ○ Else, if divisible by 4, it's a leap year.
  - Otherwise, it's not a leap year.
- Output: Print whether the year is a leap year or not.

b. Reverse a Given Number • Input: Read a number num.

- Process:
  - Initialize rev = 0.
  - While num > 0:
    - Extract last digit: ld = num % 10.
    - Update rev = rev * 10 + ld.
    - Remove last digit: num = num // 10.
- Output: Print the reversed number.

c. Finding Prime Numbers for the Given Range

- Input: Read the number n (upper limit).
- Process:
  - For each number i from 1 to n, check if it's prime:
    - If divisible by any number from 2 to √i, it's not prime.
    - If no divisors found, it is prime.
- Output: Print all prime numbers from 1 to n.

3. Functions and Recursive Functions

a. Print the Fibonacci Sequence using Functions (Iterative) • Input: Read n (number of terms in the sequence).

- Process:
    - Initialize first two terms: a = 0, b = 1. ○ Print a and b.
    - Loop (n-2) times:
        - Calculate next term c = a + b.
        - Update a = b, b = c.
    - Print the sequence of n terms.

b. Print the Fibonacci Sequence using Recursive Functions

- Input: Read n (number of terms in the sequence).
- Process:
    - Define a recursive function fibo(n):
        - If n == 0, return 0 (base case).
        - If n == 1, return 1 (base case).
        - Else, return fibo(n-1) + fibo(n-2).
    - Call fibo(i) for each i from 0 to n-1 and print the sequence.

Programs:

1. Basic Arithmetic Operations

a. Finding Area of Circle

```
r=as.integer(readline(("Enter the radius: ")))
area=pi*r*r print(area)
```

Output:

```
> r=as.integer(readline(("Enter the radius: ")))
Enter the radius: 10
> area=pi*r*r
> print(area)
[1] 314.1593
```

2. Control Structure (if-else, for loop)

a. To Check Whether the Given Year is Leap or Not ly=as.integer(readline(("Enter a Number: ")))

```
if(ly%%400==0){
  print("Leap Year")
}else if(ly%%100==0){
  print("Not a Leap Year")
}else if(ly%%4==0){
  print("Leap Year")
}else{ print("Not a Leap
    Year")
}
```

Output:

```
> ly=as.integer(readline(("Enter a Number: ")))
Enter a Number: 2000
> if(ly%%400==0){
+   print("Leap Year")
+ }else if(ly%%100==0){
+   print("Not a Leap Year")
+ }else if(ly%%4==0){
+   print("Leap Year")
+ }else{
+     print("Not a Leap Year")
+ }
[1] "Leap Year"
> ly=as.integer(readline(("Enter a Number: ")))
Enter a Number: 1300
> if(ly%%400==0){
+   print("Leap Year")
+ }else if(ly%%100==0){
+   print("Not a Leap Year")
+ }else if(ly%%4==0){
+   print("Leap Year")
+ }else{
+     print("Not a Leap Year")
+ }
[1] "Not a Leap Year"
```

b. Reverse a Given Number

```
num=as.integer(readline("Enter a number: "))
rev=0
    while(num>0){
      ld=num%%10
      rev=rev*10+ld
      num=num%/%10
    }
    cat("Reversed NUmber",rev)
```

Output:

```
> num=as.integer(readline("Enter a number: "))
Enter a number: 79
> rev=0
> while(num>0){
+   ld=num%%10
+   rev=rev*10+ld
+   num=num%/%10
+ }
> cat("Reversed NUmber",rev)
Reversed NUmber 97
```

c. Finding Prime Numbers for the Given Range

```
prime<-function(n){ if(n<=1){
      return (FALSE)} for
    (i    in    2:sqrt(n)){
    if(n%%i==0){
```

```
      return (FALSE)
    }
  }
  return (TRUE)
}

n=as.integer(readline("Enter a number: "))
for (i in 1:n){
  if(prime(i)){
    print(i)
  }
}
```

Output:

```
> prime<-function(n){
+   if(n<=1){
+     return (FALSE)}
+   for (i in 2:sqrt(n)){
+     if(n%%i==0){
+       return (FALSE)
+     }
+   }
+   return (TRUE)
+ }
> n=as.integer(readline("Enter a number: "))
Enter a number: 10
> for (i in 1:n){
+   if(prime(i)){
+     print(i)
+   }
+ }
[1] 3
[1] 5
[1] 7
> 
```

3. **Functions and Recursive Functions**
   a. **Print the Fibonacci Sequence using Functions**

```
fibonacci_iterative <- function(n) { fib_series <-
numeric(n)
  fib_series[1] <- 0 if (n > 1)
  fib_series[2] <- 1

  for (i in 3:n) {
    fib_series[i] <- fib_series[i-1] + fib_series[i-2]
  }

  return(fib_series)
}

n <- as.integer(readline("How many terms? "))
print(fibonacci_iterative(n))
```

Output:

```
> fibonacci_iterative <- function(n) {
+   fib_series <- numeric(n)
+   fib_series[1] <- 0
+   if (n > 1) fib_series[2] <- 1
+
+   for (i in 3:n) {
+     fib_series[i] <- fib_series[i-1] + fib_series[i-2]
+   }
+
+   return(fib_series)
+ }
> n <- as.integer(readline("How many terms? "))
How many terms? 10
> print(fibonacci_iterative(n))
 [1]  0  1  1  2  3  5  8 13 21 34
>
```

b. **Print the Fibonacci Sequence using Recursive Functions** fibonacci_recursive <- function(n) {

```
if (n == 1) {
  return(0)
} else if (n == 2) {
  return(1)
} else { return(fibonacci_recursive(n-1) + fibonacci_recursive(n-2)) }
}
```

```
n <- as.integer(readline("How many terms? "))
fib_series <- sapply(1:n, fibonacci_recursive)
print(fib_series)
```

Output:

```
> fibonacci_recursive <- function(n) {
+   if (n == 1) {
+     return(0)
+   } else if (n == 2) {
+     return(1)
+   } else {
+     return(fibonacci_recursive(n-1) + fibonacci_recursive(n-2))
+   }
+ }
> n <- as.integer(readline("How many terms? "))
How many terms? 10
> fib_series <- sapply(1:n, fibonacci_recursive)
> print(fib_series)
 [1]  0  1  1  2  3  5  8 13 21 34
```

Result:

The Simple Program using R is Successfully Implemented.

EXP NO: 2	PERFORM DATA PREPROCESSING IN R

Aim:

To Perform Preprocessing of data using R.

Algorithm:

1. **Loading Data / Cleaning the Data:** ○ Create emp_df2 with columns: emp_id, age, dept, salary, experience.
2. **Storing / Uploading Data to Excel Sheet:**
   - Create a workbook wb, add a worksheet "Employee Data Preprocessing", and save emp_df2 to emp_df2.xlsx.
3. **Cleaning the Data:**
   - Replace missing age and salary with their respective mean values.
   - Convert dept to numeric.
4. **Scaling the Data:**
   - Scale the age, salary, and experience columns using z-score and update emp_df2.
5. **Splitting the Data into Train and Test:**
   - Set seed, split data into 80% train and 20% test (dataTrain, dataTest).
6. **Correlation Matrix**:
   - Compute the correlation matrix for the scaled features (age, salary, experience) to examine relationships between them.

Programs:

```
library(openxlsx)

emp_df2<-data.frame(
  emp_id=1:10,
  age=c(25,30,35,NA,55,65,NA,25,85,78),
  dept=c("AI&DS","IT","AI&ML","CSE","PHY","FT","BIOTECH","CSBS","CIVIL","MECH"),
  salary=c(50000,85100,52802,144510,552410,520000,445100,5552410,524160,NA),
  experience=c(2,5,8,14,4,6,3,2,4,5)
)

wb<-createWorkbook() addWorksheet(wb,"Employee

Data Preprocessing")

writeData(wb,"Employee Data Preprocessing",emp_df2)

saveWorkbook(wb,"C:\\Users\\karthick.S\\OneDrive\\Documents\\231801079-
```

4\\SAC\\emp_df2.xlsx",overwrite = TRUE) emp_df2$age[is.na(emp_df2$age)]<-

floor(mean(emp_df2$age,na.rm = TRUE)) emp_df2$salary[is.na(emp_df2$salary)]<-

floor(mean(emp_df2$salary,na.rm = TRUE)) emp_df2$dept<-as.numeric(as.factor(emp_df2$dept))

emp_df_scaled<-scale(emp_df2[,c("age","salary","experience")]) emp_df2<-
data.frame(emp_df2[,c("emp_id","dept")],emp_df_scaled)

correlation_matrix <- cor(emp_df2[, c("age", "salary", "experience")]) print("Correlation
Matrix:")
print(correlation_matrix)

set.seed(42)
trainIndex<-sample(1:nrow(emp_df2),0.8*nrow(emp_df2))
dataTrain<-emp_df2[trainIndex,] dataTest<-emp_df2[-
trainIndex,]

print(dataTrain)
print(dataTest)

Output:

```
> print("First Few Row of Dataset")
[1] "First Few Row of Dataset"
> head(emp_df2)
  emp_id age  dept salary experience
1      1  25 AI&DS  50000          2
2      2  30    IT  85100          5
3      3  35 AI&ML  52802          8
4      4  NA   CSE 144510         14
5      5  55   PHY 552410          4
6      6  65    FT 520000          6
```

```
> print("Correlation Matrix:")
[1] "Correlation Matrix:"
> print(correlation_matrix)
                 age     salary experience
age        1.0000000 -0.2680396  0.1080326
salary    -0.2680396  1.0000000 -0.3644421
experience 0.1080326 -0.3644421  1.0000000
```

```
> print(dataTrain)
   emp_id dept         age        salary  experience
1       1    1 -1.14775744 -4.991315e-01 -0.92681355
5       5   10  0.25194675 -1.972629e-01 -0.36510837
10     10    9  1.32505330 -1.802523e-07 -0.08425578
8       8    5 -1.14775744  2.806943e+00 -0.92681355
2       2    8 -0.91447341 -4.780420e-01 -0.08425578
4       4    6 -0.02799408 -4.423460e-01  2.44341753
6       6    7  0.71851482 -2.167362e-01  0.19659681
9       9    4  1.65165095 -2.142367e-01 -0.36510837
>
> print(dataTest)
  emp_id dept         age     salary experience
3      3    2 -0.68118937 -0.4974480   0.758302
7      7    3 -0.02799408 -0.2617392  -0.645961
```

Result:

Thus, Preprocessing data is cleaned, transformed and formatted dataset ready for analysis or modelling.

EXP NO: 3	PERFORM STATISTICAL ANALYSIS FOR A GIVEN DATASET

Aim:

To Perform Statistical Analysis for Given Dataset.

Algorithm:

1. Loading Libraries:

- Load the necessary libraries: dplyr, summarytools, psych.

2. Loading Data:

- Create a dataset data with columns Age and Salary.

3. Statistical Analysis:

- Mean: Calculate the mean of Age.
- Median: Calculate the median of Age.
- Mode: Calculate the mode of Age using the table function.
- Variance: Calculate the variance of Age.
- Standard Deviation: Calculate the standard deviation of Age.
- Correlation: Calculate the correlation between Age and Salary.

4. Descriptive Statistics:

- Use the summary() function to generate summary statistics for the dataset.

5. Quantile Analysis:

- Calculate the quantiles for both Age and Salary.

6. Interquartile Range (IQR):

- Calculate the IQR for both Age and Salary.

7. Hypothesis Testing (T-Test):

- Perform a one-sample t-test on Salary with a hypothesized mean of 70,000.

8. Visualization:

- Boxplot: Create a boxplot for Age and Salary to visualize their distributions.

9. Detailed Descriptive Statistics:

- Use describe() from the psych package to get detailed statistics for Age and Salary.
- Use descr() from the summarytools package for detailed descriptive statistics.

Program:

```
library(dplyr)
library(summarytools)
library(psych)

data <- data.frame(Age = c(25, 30, 28, 35, 40, 45, 50, 32, 38, 42),
                Salary = c(50000, 60000, 55000, 75000, 80000, 85000, 90000, 65000, 78000,
```

```
82000))
cat("Dataset:\n")
print(data)

mean_age <- mean(data$Age)
median_age <- median(data$Age)
mode_age <- as.numeric(names(sort(table(data$Age), decreasing = TRUE))[1])

var_age <- var(data$Age)
sd_age <- sd(data$Age)

corr <- cor(data$Age, data$Salary)

cat("\nStatistical Analysis Results:\n")
print(mean_age)
print(median_age)
print(mode_age)
print(var_age)
print(sd_age)
print(corr)

data_summary <- summary(data)
print(data_summary)

quantile_age <- quantile(data$Age)
quantile_salary <- quantile(data$Salary)

IQR_age <- IQR(data$Age)
IQR_salary <- IQR(data$Salary)

cat("Quantile Age", quantile_age) cat("\nQuantile
Salary", quantile_salary)

cat("\nIQR Age", IQR_age) cat("\nIQR
Salary", IQR_salary)

t_test_result <- t.test(data$Salary, mu = 70000)
print(t_test_result)

boxplot(data$Age, main = "Boxplot of Age", ylab = "Age", col = "lightblue")
boxplot(data$Salary, main = "Boxplot of Salary", ylab = "Salary", col = "lightgreen")

cat("\nDescribe Method From Describe of psych")
descr_stats <- describe(data[, c("Age", "Salary")])
print("Detailed Descriptive Statistics:")
print(descr_stats)
```

cat("\nDescribe Method From Descr of SummaryTools")
print(descr(data))

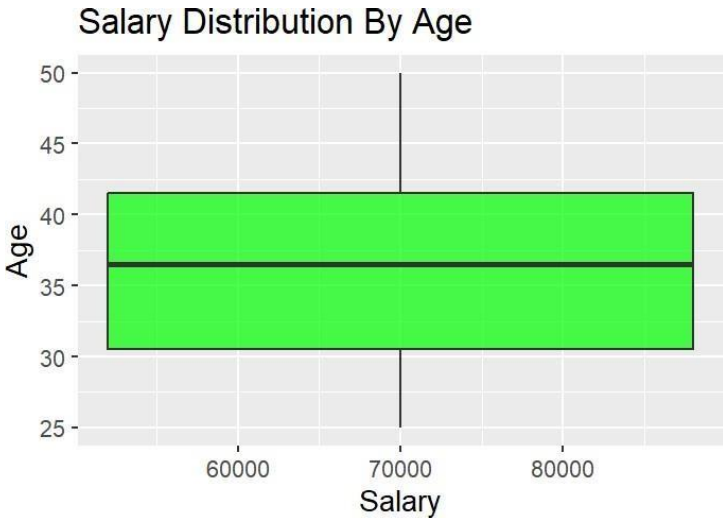
Output:

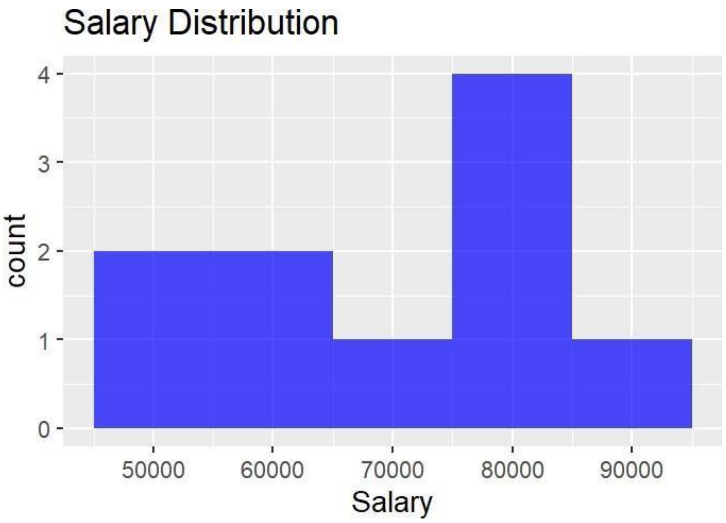
```
> cat("Dataset:\n")
Dataset:
> print(data)
   Age Salary
1   25  50000
2   30  60000
3   28  55000
4   35  75000
5   40  80000
6   45  85000
7   50  90000
8   32  65000
9   38  78000
10  42  82000
>
```

```
> cat("\nStatistical Analysis Results:\n")

Statistical Analysis Results:
> print(mean_age)
[1] 36.5
> print(median_age)
[1] 36.5
> print(mode_age)
[1] 25
> print(var_age)
[1] 63.16667
> print(sd_age)
[1] 7.947746
> print(corr)
[1] 0.9735205
```

```
> print(data_summary)
      Age            Salary
 Min.   :25.0   Min.   :50000
 1st Qu.:30.5   1st Qu.:61250
 Median :36.5   Median :76500
 Mean   :36.5   Mean   :72000
 3rd Qu.:41.5   3rd Qu.:81500
 Max.   :50.0   Max.   :90000
```

```
> cat("Quantile Age\n", quantile_age)
Quantile Age
 25 30.5 36.5 41.5 50> cat("Quantile Salary\n", quantile_salary)
Quantile Salary
 50000 61250 76500 81500 90000>
> cat("IQR Age\n", IQR_age)
IQR Age
 11> cat("IQR Salary\n", IQR_salary)
IQR Salary
 20250>
> print(t_test_result)

        One Sample t-test

data:  data$Salary
t = 0.46457, df = 9, p-value = 0.6533
alternative hypothesis: true mean is not equal to 70000
95 percent confidence interval:
 62261.33 81738.67
sample estimates:
mean of x
    72000

> cat("\nDescribe Method From Describe of psych")

Describe Method From Describe of psych> descr_stats <- describe(data[,
ge", "Salary")])
> print("Detailed Descriptive Statistics:")
[1] "Detailed Descriptive Statistics:"
> print(descr_stats)
       vars  n    mean       sd  median  trimmed     mad   min   max
Age       1 10    36.5     7.95    36.5    36.25     8.9    25    50
Salary    2 10 72000.0 13613.72 76500.0 72500.00 14826.0 50000 90000
       range  skew kurtosis      se
Age       25  0.16    -1.39    2.51
Salary 40000 -0.31    -1.57 4305.04

> cat("\nDescribe Method From Descr of SummaryTools")

Describe Method From Descr of SummaryTools> print(descr(data))
Descriptive Statistics
data
N: 10

                       Age     Salary
----------------- -------- ----------
             Mean    36.50   72000.00
          Std.Dev     7.95   13613.72
              Min    25.00   50000.00
               Q1    30.00   60000.00
           Median    36.50   76500.00
               Q3    42.00   82000.00
              Max    50.00   90000.00
              MAD     8.90   14826.00
              IQR    11.00   20250.00
               CV     0.22       0.19
         Skewness     0.16      -0.31
      SE.Skewness     0.69       0.69
         Kurtosis    -1.39      -1.57
          N.Valid    10.00      10.00
                N    10.00      10.00
        Pct.Valid   100.00     100.00
```





Result:

Thus, Statistical Analysis for a Given Dataset using is Analysed and Scaled.

EXP NO: 4	IMPLEMENT DECISION TREE ALGORITHM IN R

Aim:

Implement a Decision Tree Classification on the Given Dataset.

Procedure:

1. Load Required Libraries

- Load the necessary libraries:
    - rpart for building decision tree models. ○ rpart.plot for visualizing decision trees.
    - caret for data splitting and model evaluation.

Code:

```
library(rpart)
library(rpart.plot)
library(caret)
```

2. Load the Dataset

- Load the Iris dataset (built-in in R).
- Display the first few rows to understand the data structure.

Code:

```
data("iris")
print("First Few Rows of Dataset")
head(iris)
```

3. Split the Data into Training and Testing Sets

- Set a seed for reproducibility.
- Use createDataPartition to split the data into:
    - 80% training set ○ 20% testing set

Code:

```
set.seed(123)
train_index <- createDataPartition(iris$Species, p = 0.8, list = FALSE)
train_data <- iris[train_index, ] test_data <- iris[-train_index, ]
```

4. Train a Decision Tree Model

- Build a decision tree classifier using rpart, predicting Species based on the features.

Code:

```
tree_model <- rpart(Species ~ ., data = train_data, method = "class")
print(tree_model)
```

5. Visualize the Decision Tree

- Plot the trained decision tree using rpart.plot with enhanced formatting.

Code:

```
rpart.plot(tree_model, main = "Decision Tree
        for Iris Dataset", type = 3, extra = 101,
        under = TRUE, tweak = 1.2, box.palette
        = "RdBu")
```

6. Make Predictions on Test Data

- Use the trained model to predict the species on the test dataset. **Code:**

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

7. Evaluate Model Performance

- Create a confusion matrix to compare predicted vs actual labels.
- Print evaluation metrics like accuracy, sensitivity, specificity, etc. **Code:**

```
conf_mat <- confusionMatrix(pred, test_data$Species)
print(conf_mat)
```

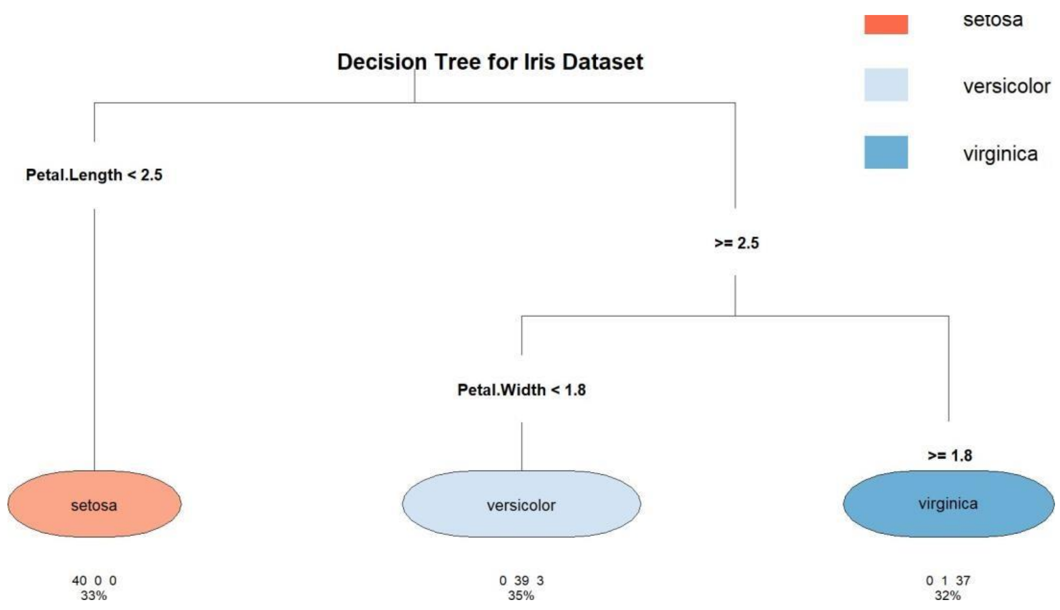
Output:

```
> print("First Few Row of Dataset")
[1] "First Few Row of Dataset"
> head(iris)
  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
```

```
> print(tree_model)
n= 120

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 120 80 setosa (0.33333333 0.33333333 0.33333333)
  2) Petal.Length< 2.45 40  0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Length>=2.45 80 40 versicolor (0.00000000 0.50000000 0.50000000)
    6) Petal.Width< 1.75 42  3 versicolor (0.00000000 0.92857143 0.07142857) *
    7) Petal.Width>=1.75 38  1 virginica (0.00000000 0.02631579 0.97368421) *
```

```
> print(conf_mat)
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         2
  virginica       0          0         8

Overall Statistics

               Accuracy : 0.9333
                 95% CI : (0.7793, 0.9918)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : 8.747e-12

                  Kappa : 0.9

 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            1.0000           0.8000
Specificity                 1.0000            0.9000           1.0000
Pos Pred Value              1.0000            0.8333           1.0000
Neg Pred Value              1.0000            1.0000           0.9091
Prevalence                  0.3333            0.3333           0.3333
Detection Rate              0.3333            0.3333           0.2667
Detection Prevalence        0.3333            0.4000           0.2667
Balanced Accuracy           1.0000            0.9500           0.9000
```



Result:

The Decision Tree is Implemented Successfully.

EXP NO: 5	IMPLEMENT K-NEAREST NEIGHBOR ALGORITHM IN R

Aim:

Implement a KNN Classification on the Given Dataset.

Procedure:

1. Load Required Libraries

- Load the necessary libraries:
  - class for KNN model. ○ ggplot2 for plotting. ○ GGally for advanced plots (pairwise plots).
  - caret for data partitioning and evaluation.

Code:

```
library(class)
library(ggplot2)
library(GGally)
library(caret)
```

2. Load the Dataset

- Load the Iris dataset.
- Display the first few rows to understand the structure.

Code:

```
data("iris")
print("First Few Rows of Dataset")
head(iris)
```

3. Define a Normalize Function

- Create a custom function to normalize (scale between 0 and 1) the numerical feature columns.

Code:

```
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x))) }
```

4. Normalize the Feature Columns

- Apply the normalization function to the first four feature columns.
- Add back the Species column separately.

Code:

```
iris_norm <- as.data.frame(lapply(iris[1:4], normalize))
iris_norm$Species <- iris$Species
```

5. Split the Data into Training and Testing Sets

- Set a random seed for reproducibility.
- Use createDataPartition to split:
  - 80% for training
  - 20% for testing

Code:

```
set.seed(123)
train_index <- createDataPartition(iris$Species, p = 0.8, list = FALSE)
train_data <- iris_norm[train_index, ] test_data <- iris_norm[-train_index, ]
```

6. Extract Training and Test Labels

- Separate the labels (Species) from the feature data for both train and test sets.

Code:

```
train_labels <- train_data$Species
test_labels <- test_data$Species
```

7. Train the KNN Model

- Train the K-Nearest Neighbors model using:
  - Normalized feature columns
  - k = 5 neighbors.

Code:

```
knn_model <- knn(train = train_data[, 1:4], test = test_data[, 1:4], cl = train_labels, k = 5)
print(knn_model)
```

8. Visualize the Data

- Create visualizations to understand feature distributions:
  - Scatter plot of Sepal Length vs Sepal Width.
  - Pairwise plots (all feature combinations).

Code:

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point() + labs(title = "Scatter Plot of Sepal Dimensions", x = "Sepal Length", y = "Sepal
Width") +
  theme_minimal()
ggpairs(iris, aes(color = Species)) +
  theme_minimal()
```

9. Evaluate Model Performance

- Generate a confusion matrix comparing predictions and true labels.
- Print classification results including accuracy, sensitivity, and specificity.

Code:

```
conf_mat <- confusionMatrix(knn_model, test_labels)
print(conf_mat)
```

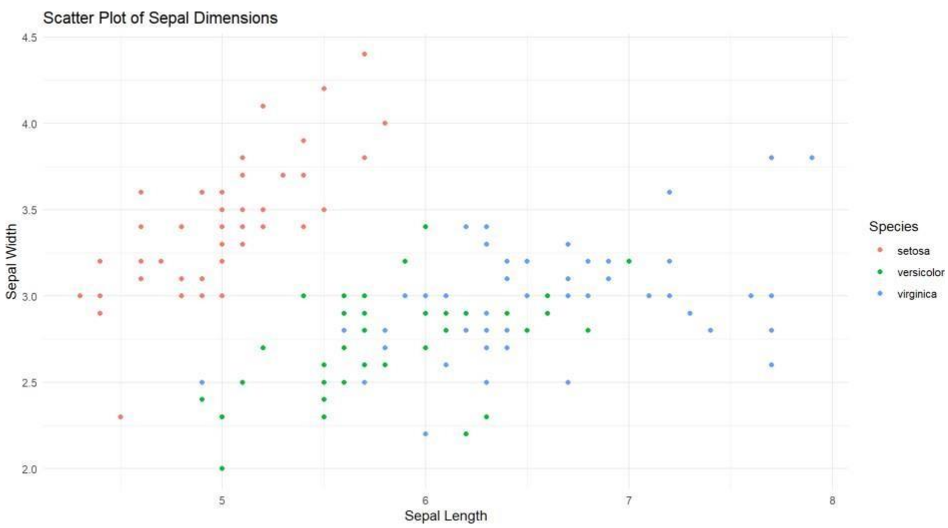
Output:

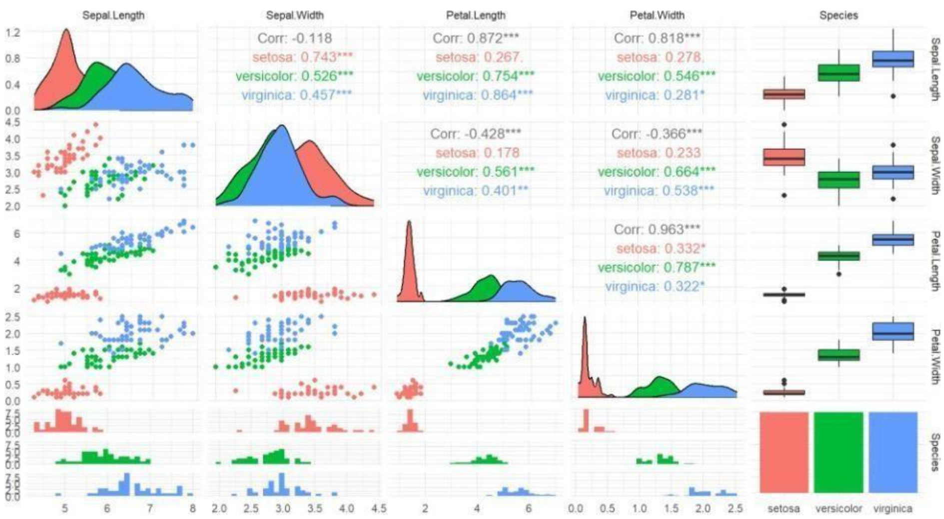
```
> print("First Few Row of Dataset")
[1] "First Few Row of Dataset"
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         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
>
```

```
> print(knn_model)
 [1] setosa     setosa     setosa     setosa     setosa
 [6] setosa     setosa     setosa     setosa     setosa
[11] versicolor versicolor versicolor versicolor versicolor
[16] versicolor versicolor versicolor versicolor versicolor
[21] virginica  virginica  virginica  virginica  virginica
[26] virginica  virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
```

```
> print(conf_mat)
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         0
  virginica       0          0        10

Overall Statistics

               Accuracy : 1
                 95% CI : (0.8843, 1)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : 4.857e-15

                  Kappa : 1

 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: setosa Class: versicolor
Sensitivity                 1.0000            1.0000
Specificity                 1.0000            1.0000
Pos Pred Value              1.0000            1.0000
Neg Pred Value              1.0000            1.0000
Prevalence                  0.3333            0.3333
Detection Rate              0.3333            0.3333
Detection Prevalence        0.3333            0.3333
Balanced Accuracy           1.0000            1.0000
                     Class: virginica
Sensitivity                    1.0000
Specificity                    1.0000
Pos Pred Value                 1.0000
Neg Pred Value                 1.0000
Prevalence                     0.3333
Detection Rate                 0.3333
Detection Prevalence           0.3333
Balanced Accuracy              1.0000
```





Result:

The KNN Classification is Successfully Implemented.

EXP NO: 6	IMPLEMENT NAIVE BAYESIAN CLASSIFIER IN R

Aim:

Implement a Naïve Bayes Classification on the Given Dataset.

Procedure:

1. Load Required Libraries

- Load the necessary libraries: ○ e1071 for the Naive Bayes model. ○ ggplot2 for visualization.
  - caret for data partitioning and evaluation.

Code:

```
library(e1071)
library(ggplot2)
library(caret)
```

2. Load the Dataset

- Load the Iris dataset.
- Display the first few rows for a quick overview.

Code:

```
data("iris")
print("First Few Rows of Dataset")
head(iris)
```

3. Split the Data into Training and Testing Sets • Set a random seed to ensure reproducibility.

- Split the data into: ○ 80% for training ○ 20% for testing **Code:**

```
set.seed(123)
train_index <- createDataPartition(iris$Species, p = 0.8, list = FALSE)
train_data <- iris[train_index, ] test_data <- iris[-train_index, ]
```

4. Extract Training and Test Labels

- Assign the Species column as the labels for training and testing.

Code:

```
train_labels <- train_data$Species
test_labels <- test_data$Species
```

5. Train the Naive Bayes Model

- Train the Naive Bayes classifier using the training data. **Code:**

```
nb_model <- naiveBayes(Species ~ ., data = train_data)
print(nb_model)
```

6. Visualize the Data

- Create a scatter plot of Sepal Length vs Sepal Width colored by species. **Code:**
  ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point() + labs(title = "Scatter Plot of Sepal Dimensions", x = "Sepal Length", y = "Sepal Width") +
  theme_minimal()

7. Make Predictions on the Test Data

- Predict the species for the test dataset using the trained model.

Code:

```
pred <- predict(nb_model, test_data)
```

8. Evaluate Model Performance

- Generate a confusion matrix to compare the predicted labels and true labels.
- Print evaluation metrics like accuracy, sensitivity, and specificity.

Code:

```
conf_mat <- confusionMatrix(pred, test_labels)
print(conf_mat)
```

Output:

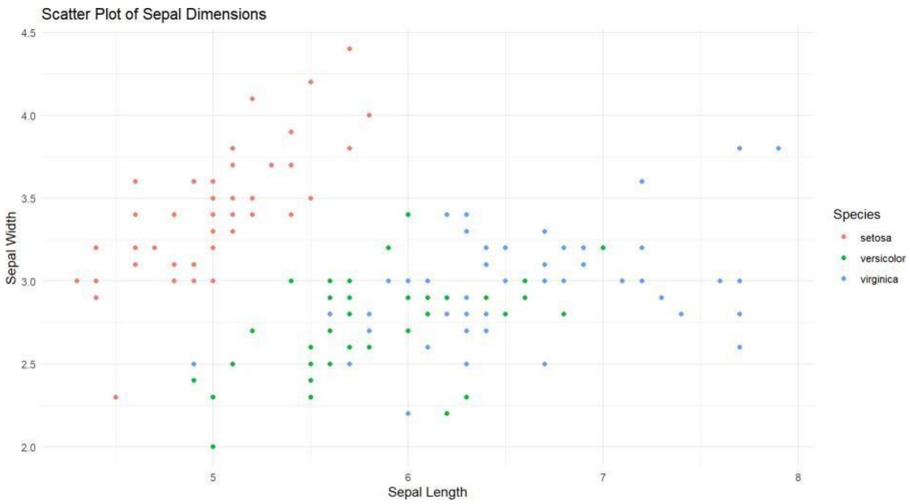
```
> print("First Few Rows of Dataset")
[1] "First Few Rows of Dataset"
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          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
```

```
> print(nb_model)

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
    setosa versicolor  virginica
 0.3333333  0.3333333  0.3333333

Conditional probabilities:
            Sepal.Length
Y              [,1]      [,2]
  setosa     4.9800 0.3567661
  versicolor 5.9400 0.4903165
  virginica  6.6375 0.6949221

            Sepal.Width
Y              [,1]      [,2]
  setosa     3.3700 0.3450752
  versicolor 2.7700 0.3267556
  virginica  3.0125 0.3123012
```

```
            Petal.Length
Y              [,1]      [,2]
  setosa     1.4650 0.1717930
  versicolor 4.2325 0.4676112
  virginica  5.6225 0.5775667

            Petal.Width
Y              [,1]      [,2]
  setosa     0.2400 0.0928191
  versicolor 1.3275 0.2087662
  virginica  2.0700 0.2662176
```

```
> print(conf_mat)
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         2
  virginica       0          0         8

Overall Statistics

               Accuracy : 0.9333
                 95% CI : (0.7793, 0.9918)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : 8.747e-12

                  Kappa : 0.9

 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: setosa Class: versicolor
Sensitivity                 1.0000            1.0000
Specificity                 1.0000            0.9000
Pos Pred Value              1.0000            0.8333
Neg Pred Value              1.0000            1.0000
Prevalence                  0.3333            0.3333
Detection Rate              0.3333            0.3333
Detection Prevalence        0.3333            0.4000
Balanced Accuracy           1.0000            0.9500
                     Class: virginica
Sensitivity                    0.8000
Specificity                    1.0000
Pos Pred Value                 1.0000
Neg Pred Value                 0.9091
Prevalence                     0.3333
Detection Rate                 0.2667
Detection Prevalence           0.2667
Balanced Accuracy              0.9000
```



Result:

The Naïve Bayes Classification is Successfully Implemented.

EXP NO: 7	IMPLEMENT LINEAR REGRESSION IN R

Aim:

Implement a Linear Regression on the Given Dataset.

Procedure:

1. Load Required Libraries

- Load the necessary libraries:
  - ggplot2 for visualization.
  - caret for splitting the data and evaluating the model.

Code:

```
library(ggplot2)
library(caret)
```

2. Load the Dataset

- Load the Headbrain dataset from a CSV file.
- Display the first few rows to inspect the data.

Code: df <-

```
read.csv("C:/Users/karthick.S/OneDrive/Documents/231801079-
4/SAC/headbrain.csv") print("First
Few Rows of Dataset") head(df)
```

3. Split the Data into Training and Testing Sets

- Set a random seed for reproducibility.
- Split the data into:
  - 70% for training
  - 30% for testing

Code:

```
set.seed(123)
index <- createDataPartition(df$Brain.Weight.grams., p = 0.7, list = FALSE)
train <- df[index, ] test <- df[-index, ]
```

4. Train the Linear Regression Model

- Train a linear regression model to predict Brain.Weight.grams. based on Head.Size.cm.3..

Code:

```
print("Linear Regression Model")
model <- lm(Brain.Weight.grams. ~ Head.Size.cm.3., data = train)
print(model)
```

5. Make Predictions on the Test Data

- Use the trained model to predict brain weight values for the test dataset.

Code:

```
pred <- predict(model, newdata = test)
```

6. Evaluate Model Performance

- Use postResample to calculate evaluation metrics:
    - RMSE (Root Mean Squared Error) ○
R-squared (Coefficient of Determination)
    - MAE (Mean Absolute Error) **Code:**

```
evaluation <- postResample(pred, test$Brain.Weight.grams.)
cat("RMSE:", evaluation["RMSE"], "\n") cat("R-squared:",
evaluation["Rsquared"], "\n") cat("MAE:",
evaluation["MAE"], "\n")
```

7. Visualize the Data

- Plot the scatter points of the original data.
- Overlay the regression line based on the model's predictions.

Code:

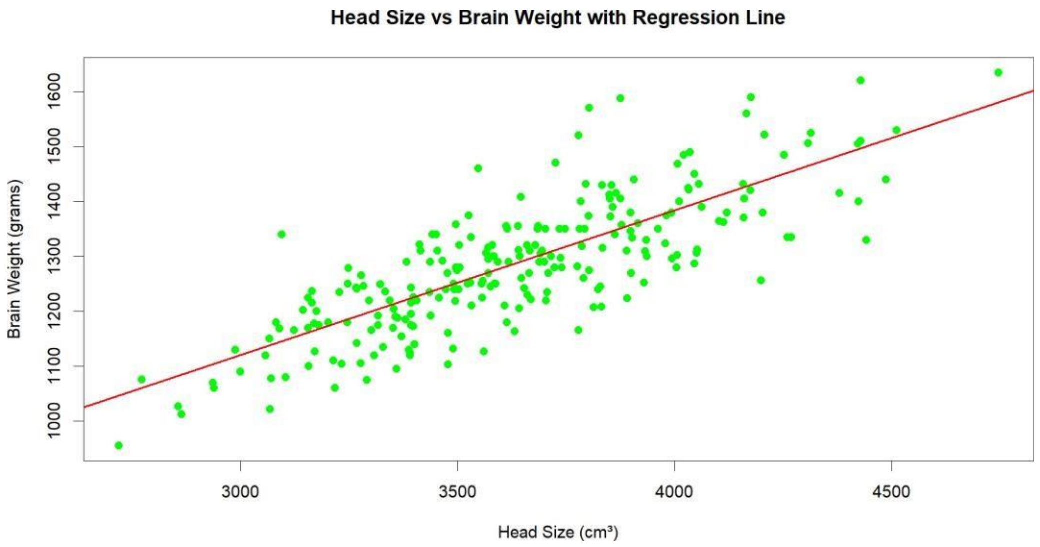
```
x_vals <- seq(min(df$Head.Size.cm.3.) - 100, max(df$Head.Size.cm.3.) + 100,
length.out = 1000)
pred_line <- data.frame(Head.Size.cm.3. = x_vals)
pred_line$Brain.Weight.grams. <- predict(model, newdata = pred_line)

plot(df$Head.Size.cm.3., df$Brain.Weight.grams.,
    col = "green", pch = 19, xlab = "Head Size
    (cm³)", ylab = "Brain Weight (grams)",
    main = "Head Size vs Brain Weight with Regression Line")

lines(pred_line$Head.Size.cm.3., pred_line$Brain.Weight.grams., col = "red", lwd =
2)
```

Output:

```
> print("First Few Rows of Dataset")
[1] "First Few Rows of Dataset"
> head(df)
  Gender Age.Range Head.Size.cm.3. Brain.Weight.grams.
1      1         1            4512                1530
2      1         1            3738                1297
3      1         1            4261                1335
4      1         1            3777                1282
5      1         1            4177                1590
6      1         1            3585                1300
```

```
> print("Linear Regression Model")
[1] "Linear Regression Model"
> model <- lm(Brain.Weight.grams. ~ Head.Size.cm.3., data = train)
> print(model)

Call:
lm(formula = Brain.Weight.grams. ~ Head.Size.cm.3., data = train)

Coefficients:
    (Intercept)  Head.Size.cm.3.
       329.6963           0.2635
```

```
> cat("RMSE:", evaluation["RMSE"], "\n")
RMSE: 75.80924
> cat("R-squared:", evaluation["Rsquared"], "\n")
R-squared: 0.5921795
> cat("MAE:", evaluation["MAE"], "\n")
MAE: 59.80292
```



Result:

The Linear Regression is Successfully Implemented.

EXP NO: 8	IMPLEMENT K-MEANS CLUSTERING ALGORITHM IN R

Aim:

Implement a Kmeans Clustering on the Given Dataset.

Procedure:

Procedure for Performing and Evaluating K-means Clustering in R

1. Load Required Libraries • Load the necessary libraries:

- ggplot2 for plotting. ○ cluster for silhouette analysis.
- factoextra for easy visualization of clustering.

Code:

```
library(ggplot2)
library(cluster)
library(factoextra)
```

2. Load the Dataset

- Load the Iris dataset.
- Remove the Species column to focus only on the numeric features for clustering.

Code:

```
data(iris) iris_data
<- iris[, -5]
head(iris_data)
```

3. Determine the Optimal Number of Clusters Using Elbow Method

- Use the Within-Cluster Sum of Squares (WSS) method to decide how many clusters are appropriate.

Code:

```
fviz_nbclust(iris_data, kmeans, method = "wss") +
  ggtitle("Elbow Method for Optimal K")
```

4. Apply K-means Clustering with 3 Clusters

- Set a random seed for reproducibility.
- Apply K-means clustering specifying 3 clusters (since Iris has 3 species).

Code:

```
set.seed(123)
kmeans_model <- kmeans(iris_data, centers = 3, nstart = 25)
```

5. Print Cluster Centers and Cluster Assignments

- View the center points of the clusters and how the data points were assigned.

Code:

print(kmeans_model$centers)
print(kmeans_model$cluster)

6. Visualize the Clusters

- Visualize the clustering result using a scatter plot with convex hulls around clusters.
  Code:
  fviz_cluster(kmeans_model, data = iris_data, geom = "point", ellipse.type = "convex") + ggtitle("K-means Clustering on Iris Dataset")

7. Evaluate the Clustering (Silhouette Analysis)

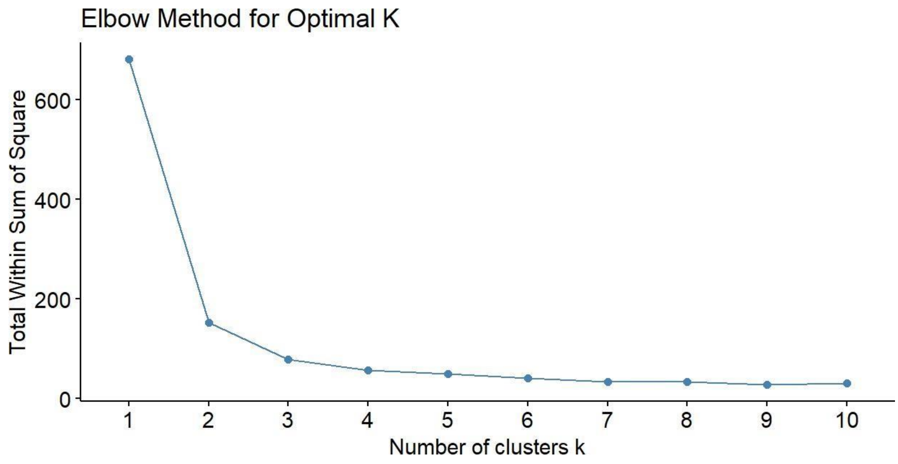
- Perform silhouette analysis to assess the quality of the clustering. **Code:**
  silhouette_score <- silhouette(kmeans_model$cluster, dist(iris_data))
  fviz_silhouette(silhouette_score)

Output:

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

```
> print(kmeans_model$centers)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053
> print(kmeans_model$cluster)
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [31] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2
 [61] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2
 [91] 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 3 3 2 2 3 3 3 3 2
[121] 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2
```

```
> fviz_silhouette(silhouette_score)
  cluster size ave.sil.width
1       1   50          0.80
2       2   62          0.42
3       3   38          0.45
```



Result:

The Kmeans is Successfully Implemented.