

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE LAB MANUAL**

**AD23431 - STATISTICAL ANALYSIS AND COMPUTING**

**(REGULATION 2023)**

**RAJALAKSHMI ENGINEERING COLLEGE**

**Thandalam, Chennai-602015**

Name: Dilli Ganesh

Register No: 231801033

Year / Branch / Section: 2nd / AI&DS / FA Semester: IV

Academic Year: 2024 - 2025

INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | Date | Title | Page No |
| 1. | 05/2/25 | Implement Simple Programs in R | 3 |
| 2. | 19/2/25 | Perform Data Preprocessing in R | 9 |
| 3. | 05/3/25 | Perform Statistical Analysis for a Given Dataset | 12 |
| 4. | 26/3/25 | Implement Decision Tree Algorithm in R | 17 |
| 5. | 02/4/25 | Implement K-Nearest Neighbor Algorithm in R | 21 |
| 6. | 16/4/25 | Implement Naive Bayesian Classifier in R | 26 |
| 7. | 16/4/25 | Implement Linear Regression in R | 30 |
| 8. | 23/4/25 | Implement K-means Clustering Algorithm in R | 33 |

**IMPLEMENT SIMPLE PROGRAMS IN R**

**EXP NO: 1**

# Aim:

To Implement Simple Programs using R.

**Algorithm:**

1. **Basic Arithmetic Operations**
   1. **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.

# Control Structures (if-else, for loop)

* 1. **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.

# 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.

# 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.

# Functions and Recursive Functions

* 1. **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.

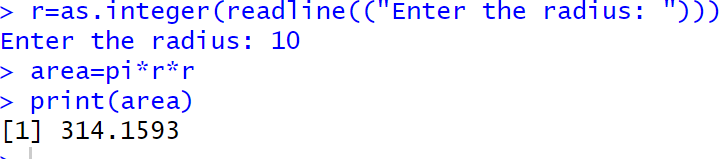
# Print the Fibonacci Sequence using Recursive Functions

* + - 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**
   1. **Finding Area of Circle** r=as.integer(readline(("Enter the radius: "))) area=pi\*r\*r

print(area)

**Output:**

1. **Control Structure (if-else, for loop)**
   1. **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")

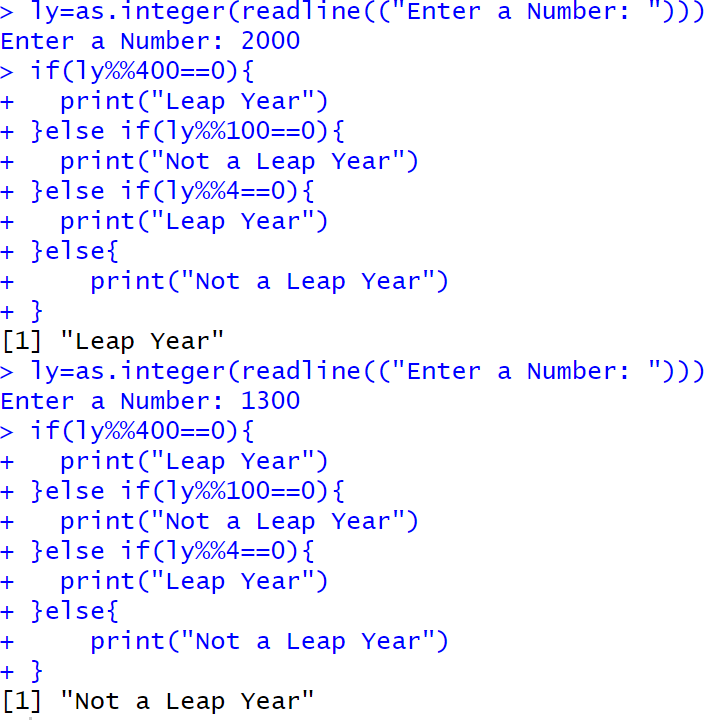
}else if(ly%%4==0){ print("Leap Year")

}else{

print("Not a Leap Year")

}

**Output:**

****

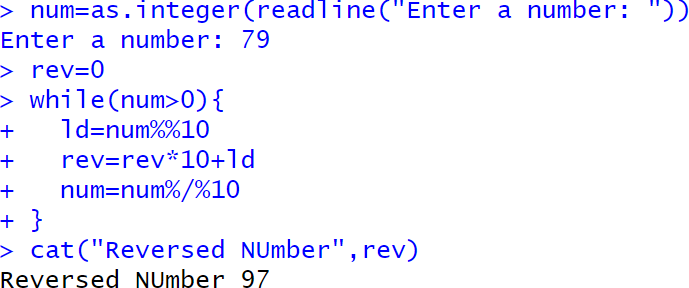
* 1. **Reverse a Given Number** num=as.integer(readline("Enter a number: ")) rev=0

while(num>0){ ld=num%%10 rev=rev\*10+ld num=num%/%10

}

cat("Reversed NUmber",rev)

**Output:**

****

* 1. **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)

}

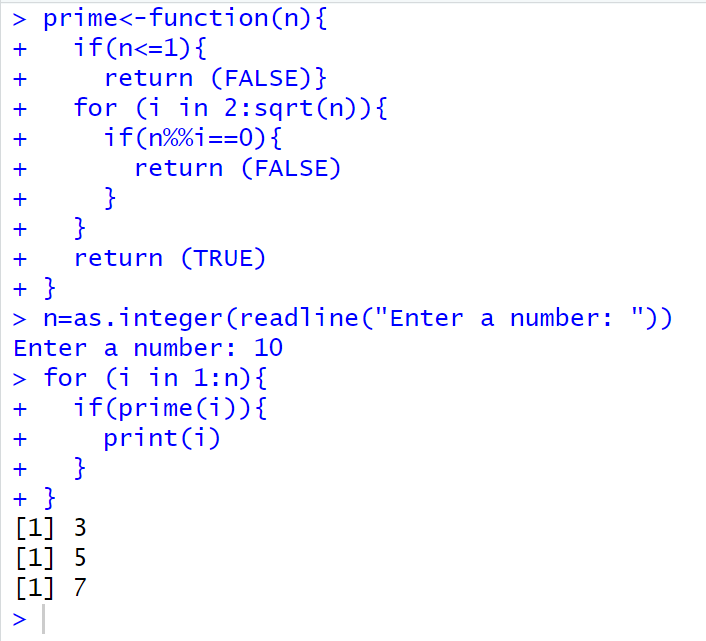
n=as.integer(readline("Enter a number: ")) for (i in 1:n){

if(prime(i)){ print(i)

}

}

**Output:**

****

1. **Functions and Recursive Functions**
   1. **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]

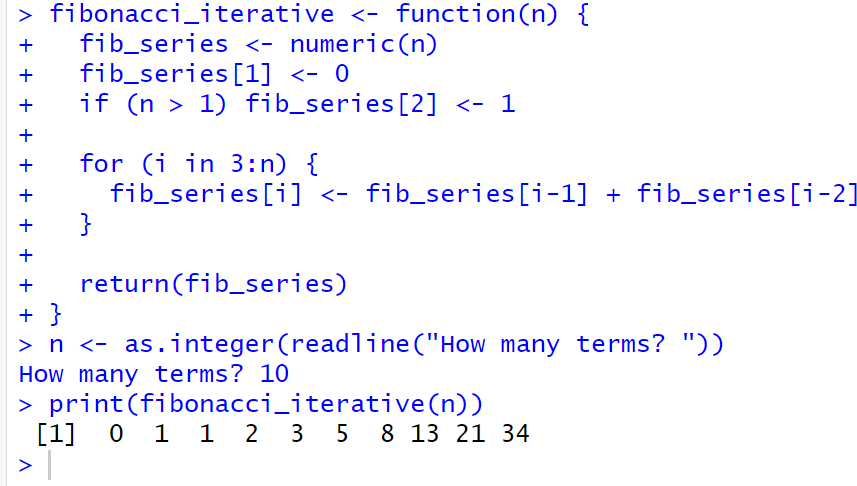
}

return(fib\_series)

}

n <- as.integer(readline("How many terms? ")) print(fibonacci\_iterative(n))

**Output:**

****

* 1. **Print the Fibonacci Sequence using Recursive Functions**

fibonacci\_recursive <- function(n) { if (n == 1) {

return(0)

} else if (n == 2) { return(1)

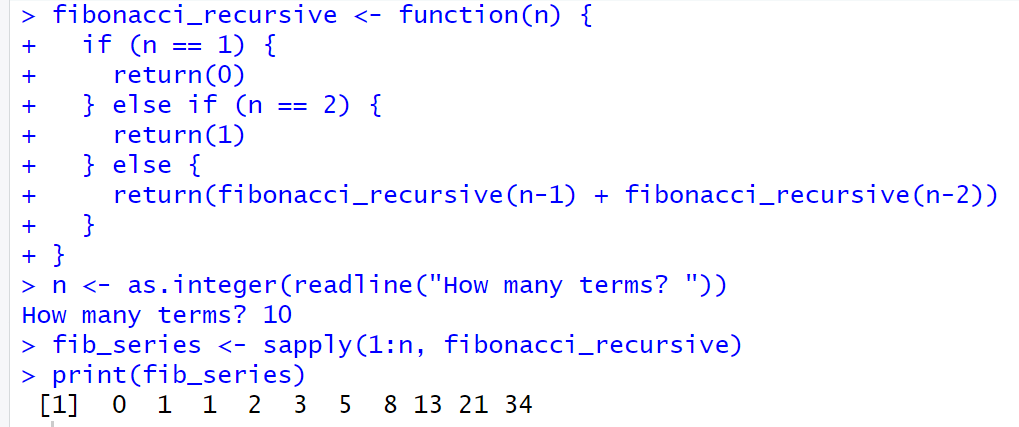
} else {

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:**

# Result:

The Simple Program using R is Successfully Implemented.

**PERFORM DATA PREPROCESSING IN R**

**EXP NO: 2**

# 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.

# Storing / Uploading Data to Excel Sheet:

* + Create a workbook wb, add a worksheet "Employee Data Preprocessing", and save emp\_df2 to emp\_df2.xlsx.

# Cleaning the Data:

* + Replace missing age and salary with their respective mean values.
  + Convert dept to numeric.

# Scaling the Data:

* + Scale the age, salary, and experience columns using z-score and update emp\_df2.

# Splitting the Data into Train and Test:

* + Set seed, split data into 80% train and 20% test (dataTrain, dataTest).

# 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)

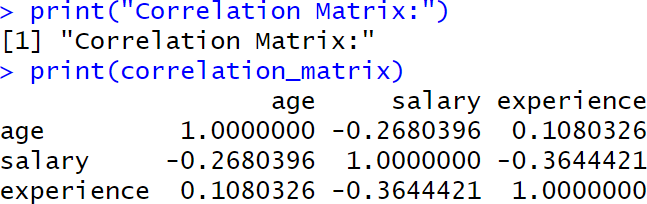
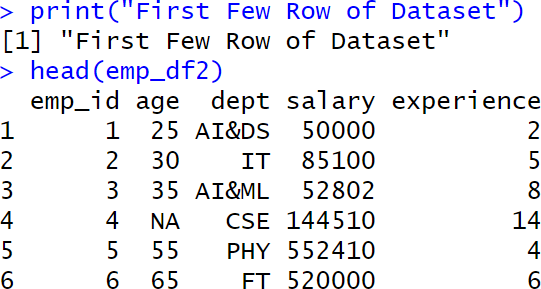
set.seed(42)

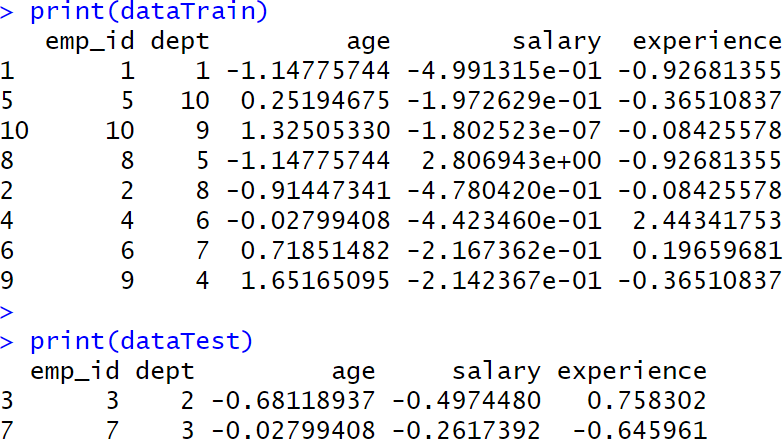
trainIndex<-sample(1:nrow(emp\_df2),0.8\*nrow(emp\_df2)) dataTrain<-emp\_df2[trainIndex,]

dataTest<-emp\_df2[-trainIndex,]

print(dataTrain) print(dataTest)

# Output:

****



**Result:**

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

**PERFORM STATISTICAL ANALYSIS FOR A GIVEN DATASET**

**EXP NO: 3**

# Aim:

To Perform Statistical Analysis for Given Dataset.

# Algorithm:

1. **Loading Libraries:**
   * Load the necessary libraries: dplyr, summarytools, psych.

# Loading Data:

* + Create a dataset data with columns Age and Salary.

# 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.

# Descriptive Statistics:

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

# Quantile Analysis:

* + Calculate the quantiles for both Age and Salary.

# Interquartile Range (IQR):

* + Calculate the IQR for both Age and Salary.

# Hypothesis Testing (T-Test):

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

# Visualization:

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

# 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)

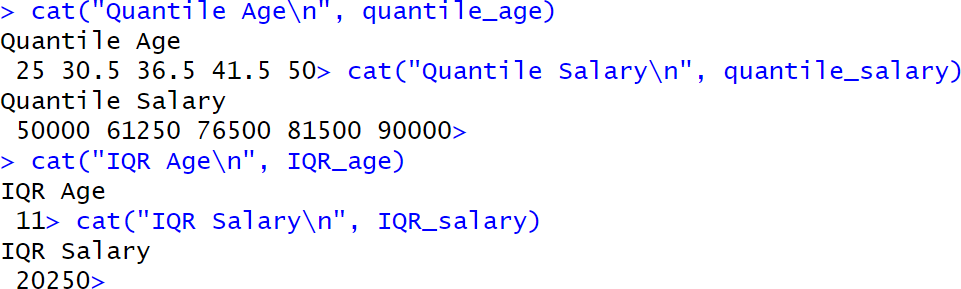
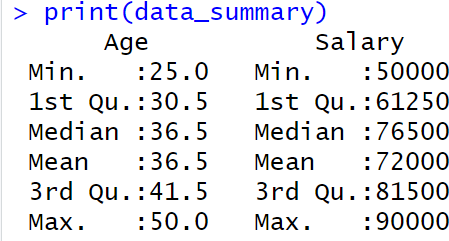
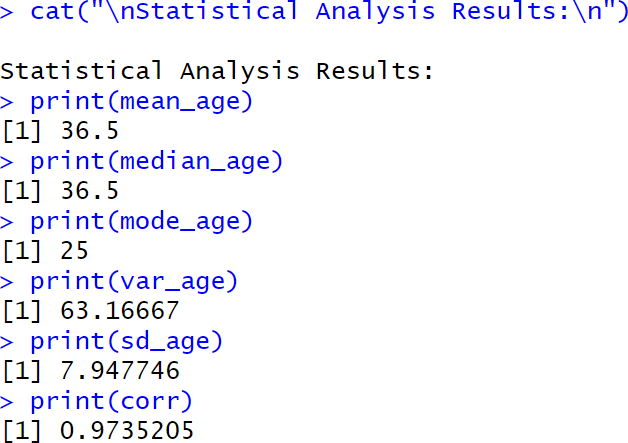
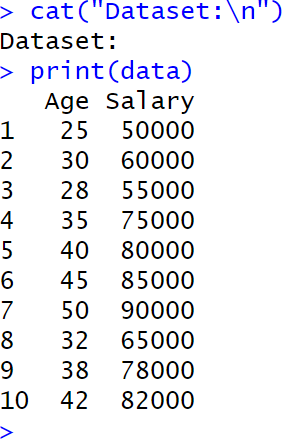
t\_test\_result <- t.test(data$Salary, mu = 70000) print(t\_test\_result)

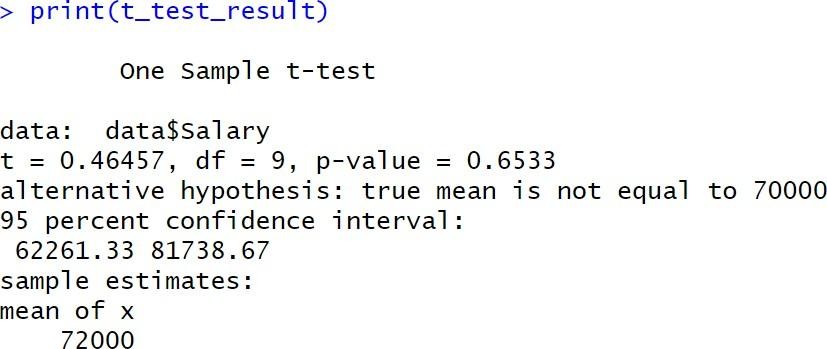
boxplot(data$Age, main = "Boxplot of Age", ylab = "Age", col = "lightblue") boxplot(data$Salary, main = "Boxplot of Salary", ylab = "Salary", col = "lightgreen")

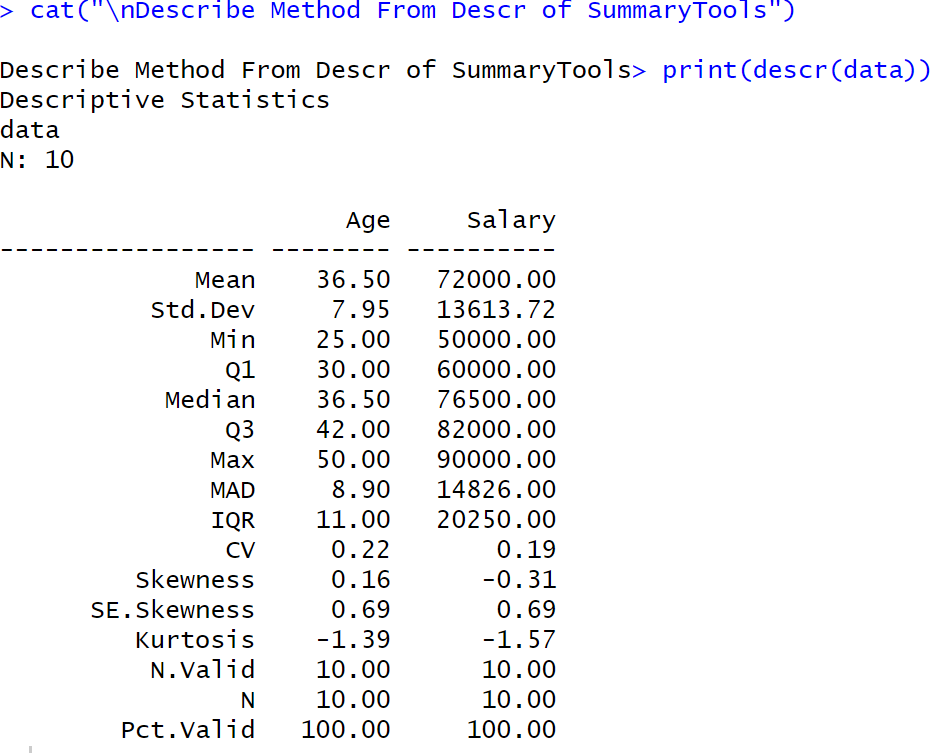
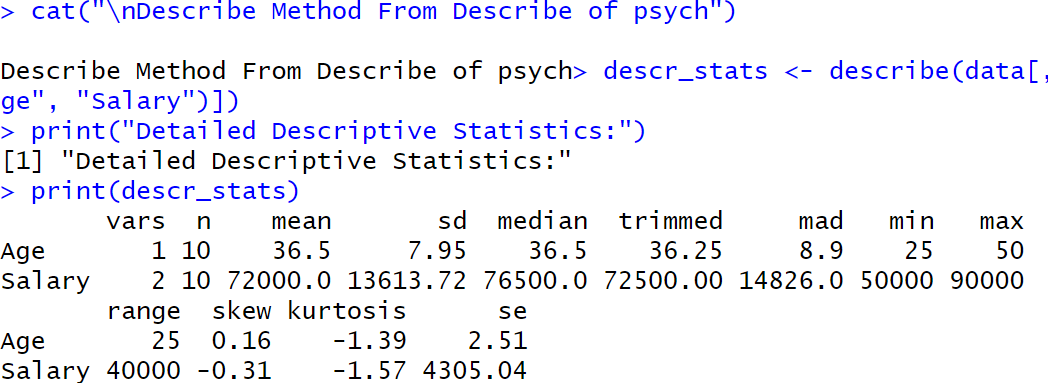
cat("\nDescribe Method From Describe of psych") descr\_stats <- describe(data[, c("Age", "Salary")]) print("Detailed Descriptive Statistics:") print(descr\_stats)

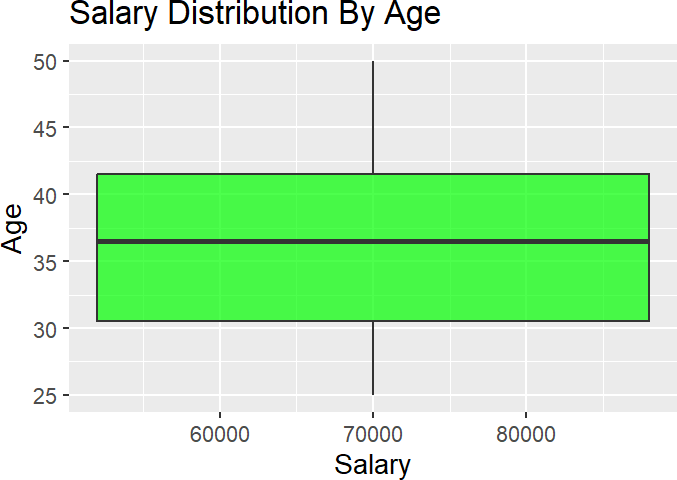
cat("\nDescribe Method From Descr of SummaryTools") print(descr(data))

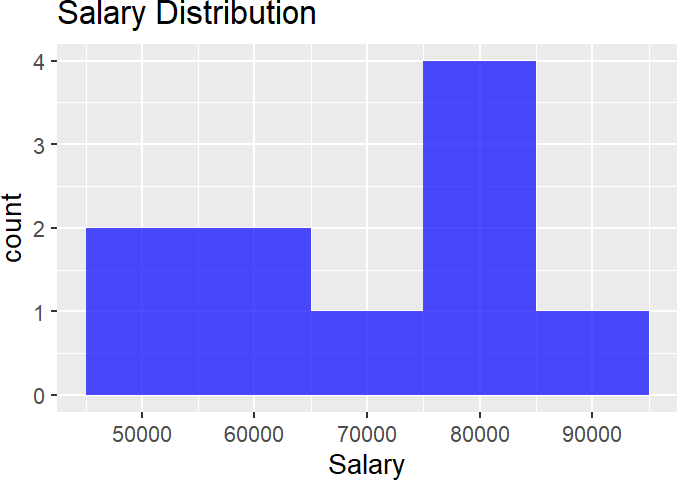
# Output:

****



****



****

**Result:**

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

**IMPLEMENT DECISION TREE ALGORITHM IN R**

**EXP NO: 4**

# 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)

# 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)

# 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, ]

# 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)

# 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")

# 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")

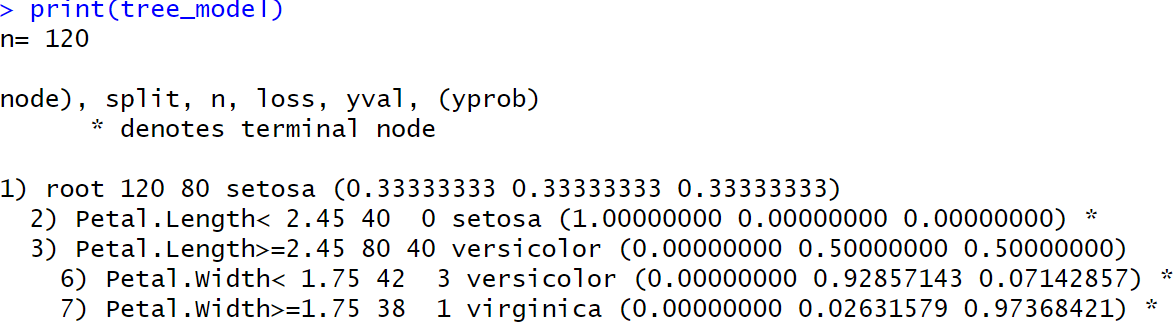
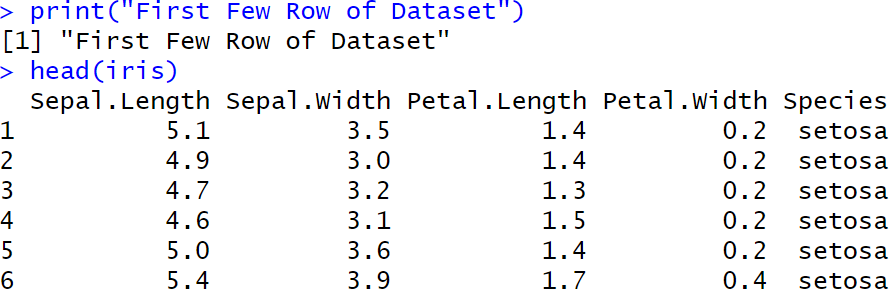
# Evaluate Model Performance

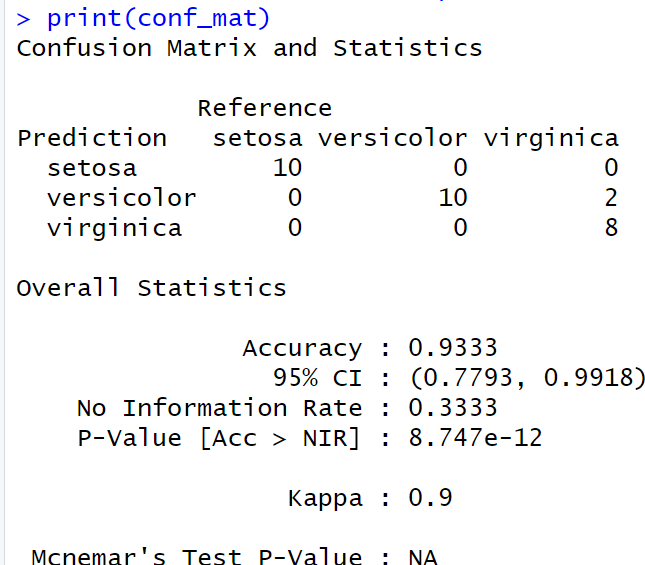
* + Create a confusion matrix to compare predicted vs actual labels.
  + Print evaluation metrics like accuracy, sensitivity, specificity, etc.

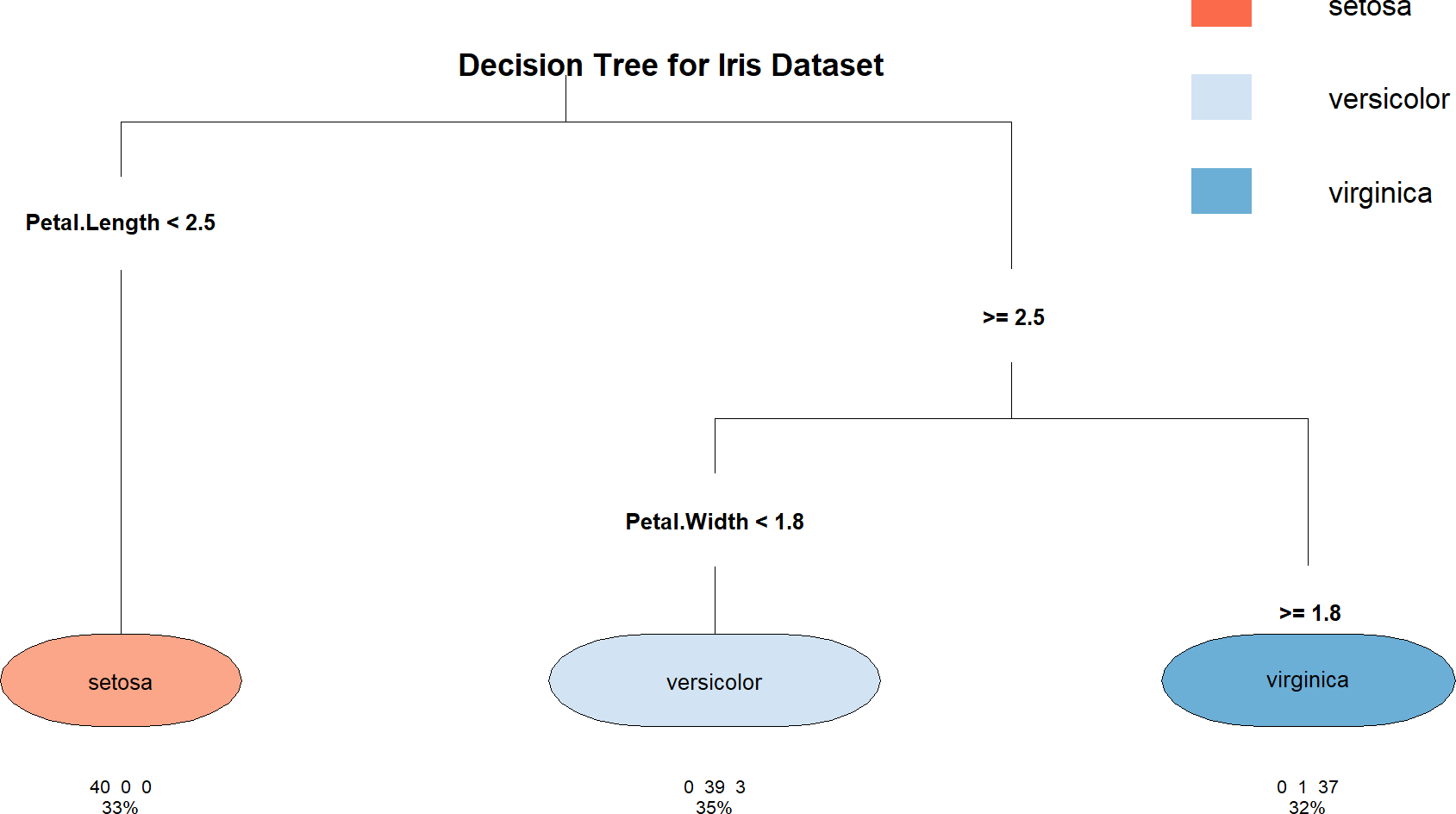
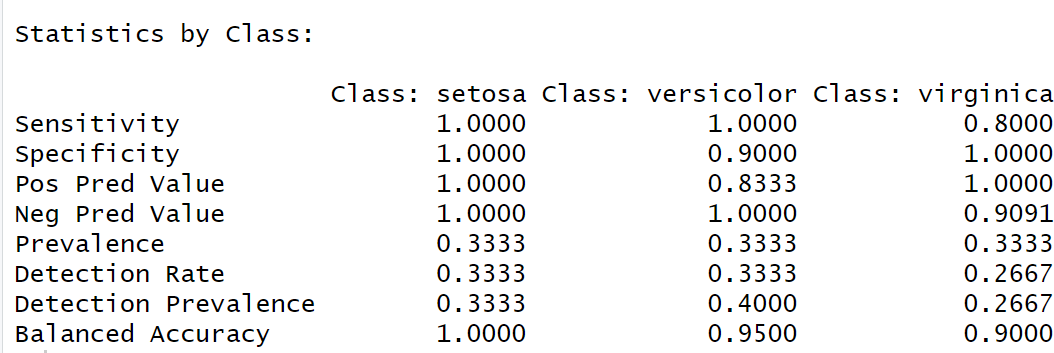
# Code:

conf\_mat <- confusionMatrix(pred, test\_data$Species) print(conf\_mat)

# Output:

****



****

**Result:**

The Decision Tree is Implemented Successfully.

**IMPLEMENT K-NEAREST NEIGHBOR ALGORITHM IN R**

**EXP NO: 5**

# 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)

# 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)

# 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)))

}

# 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

# 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, ]

# 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

# 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)

# 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()

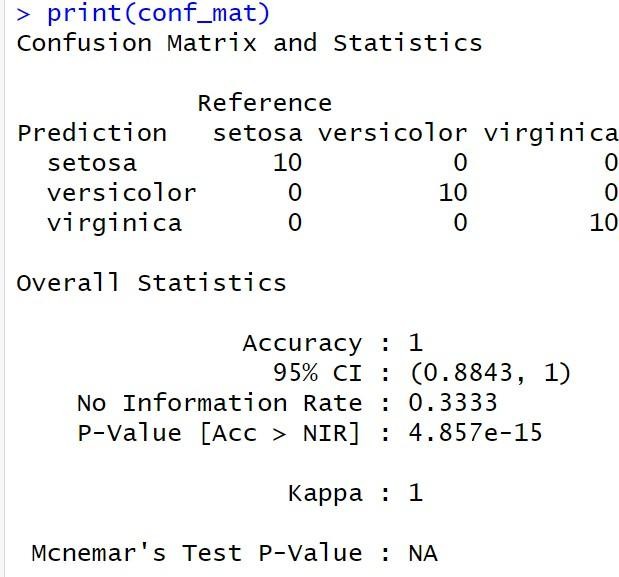
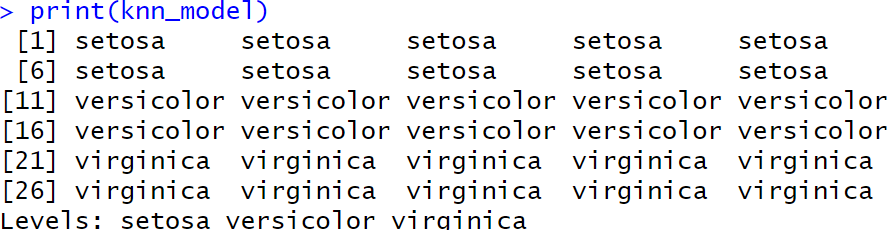
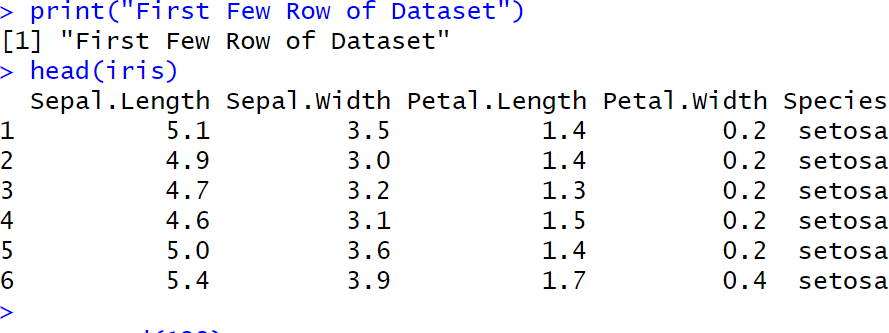
# Evaluate Model Performance

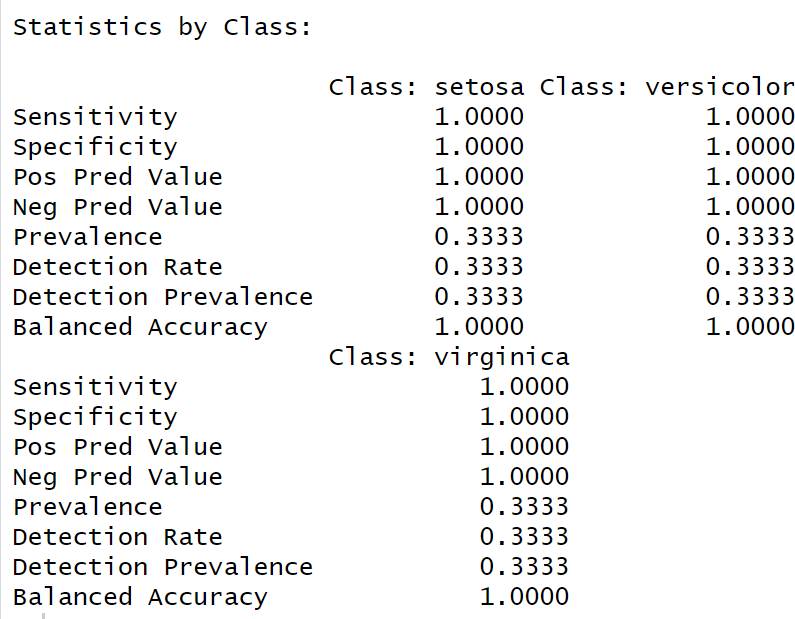
* + Generate a confusion matrix comparing predictions and true labels.
  + Print classification results including accuracy, sensitivity, and specificity.

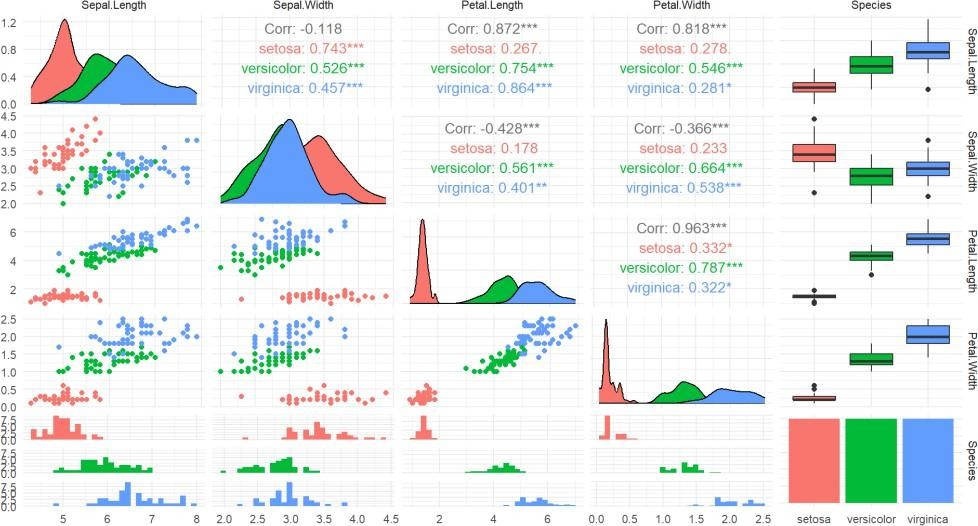
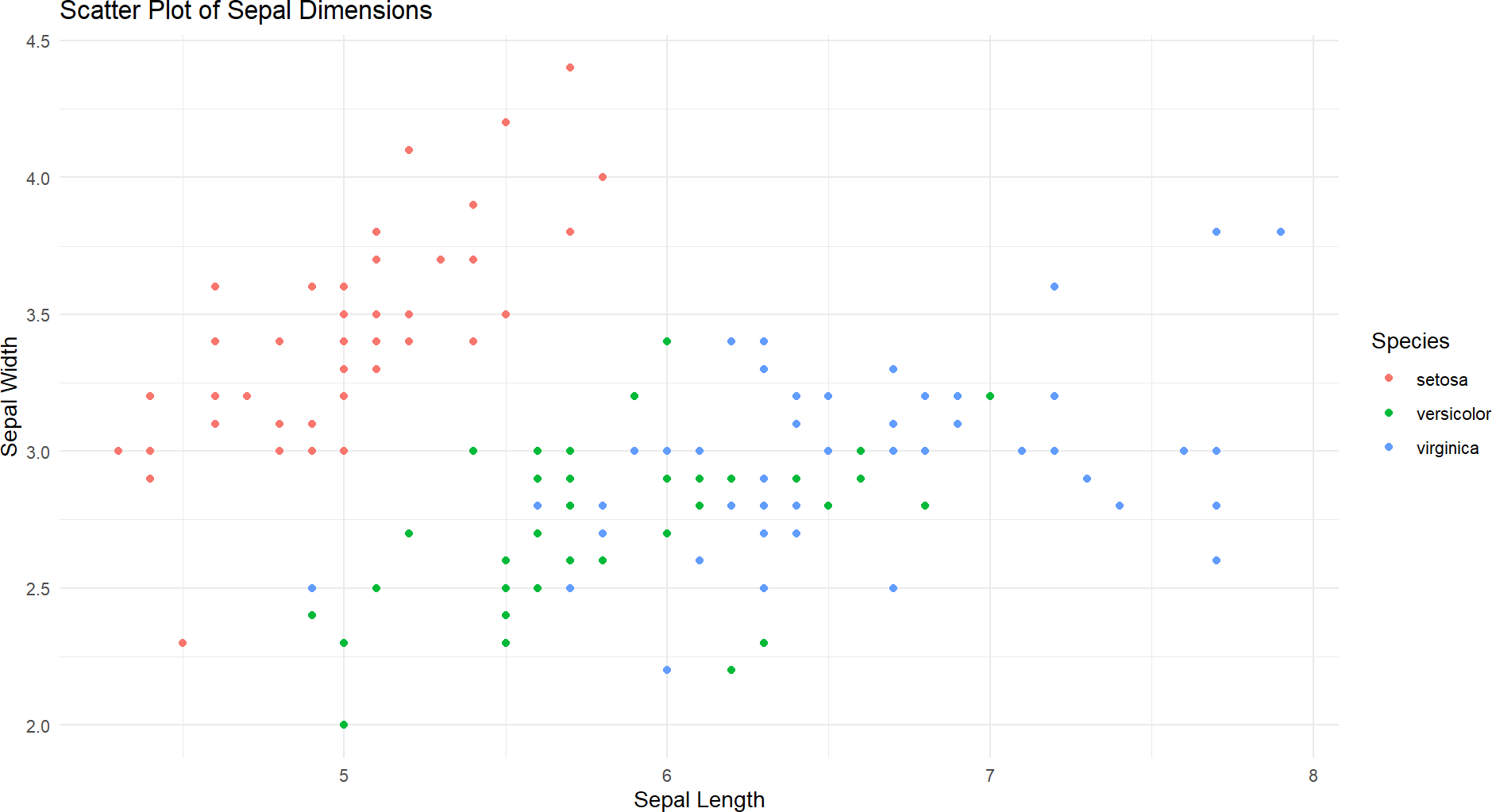
# Code:

conf\_mat <- confusionMatrix(knn\_model, test\_labels) print(conf\_mat)

# Output:

****



****

**Result:**

The KNN Classification is Successfully Implemented.

**IMPLEMENT NAIVE BAYESIAN CLASSIFIER IN R**

**EXP NO: 6**

# 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)

# 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)

# 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, ]

# 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

# 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)

# 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()

# Make Predictions on the Test Data

* + Predict the species for the test dataset using the trained model.

# Code:

pred <- predict(nb\_model, test\_data)

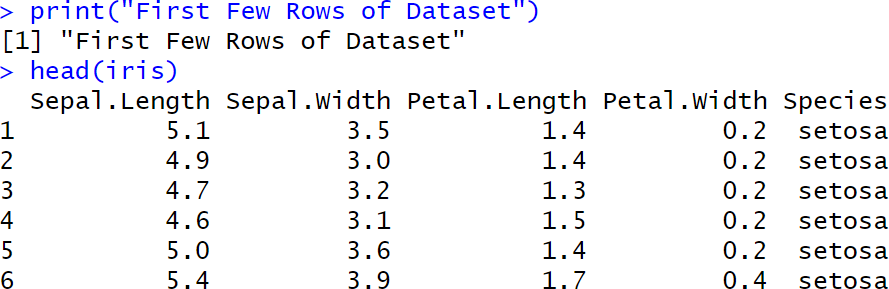
# Evaluate Model Performance

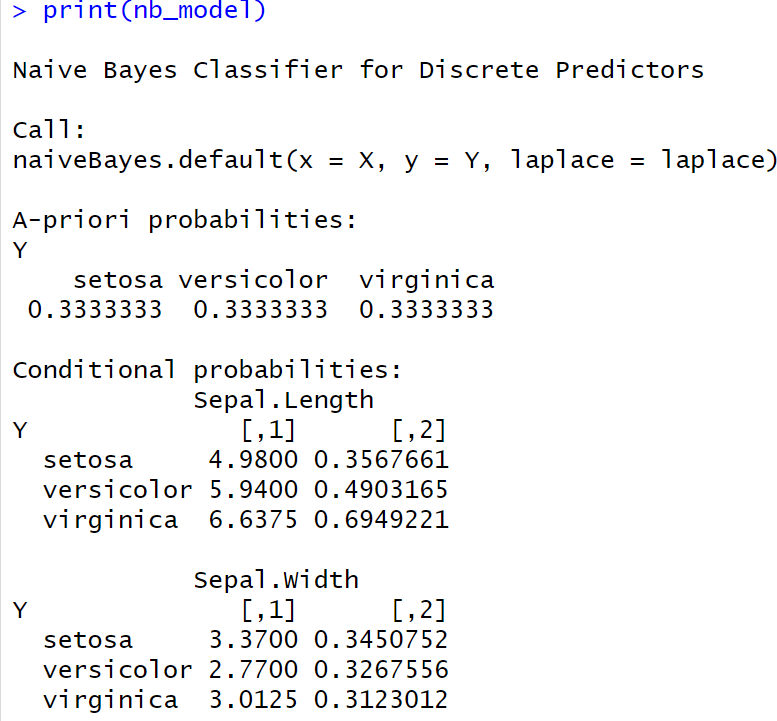
* + Generate a confusion matrix to compare the predicted labels and true labels.
  + Print evaluation metrics like accuracy, sensitivity, and specificity.

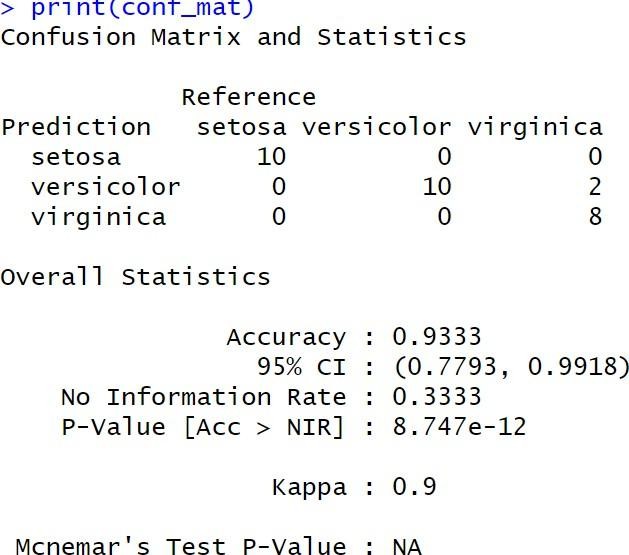
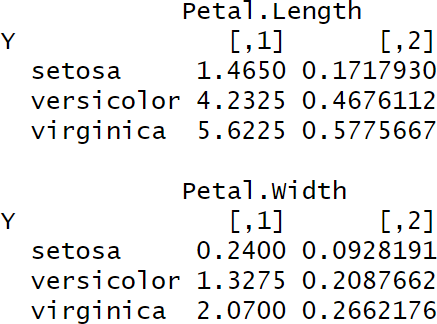
# Code:

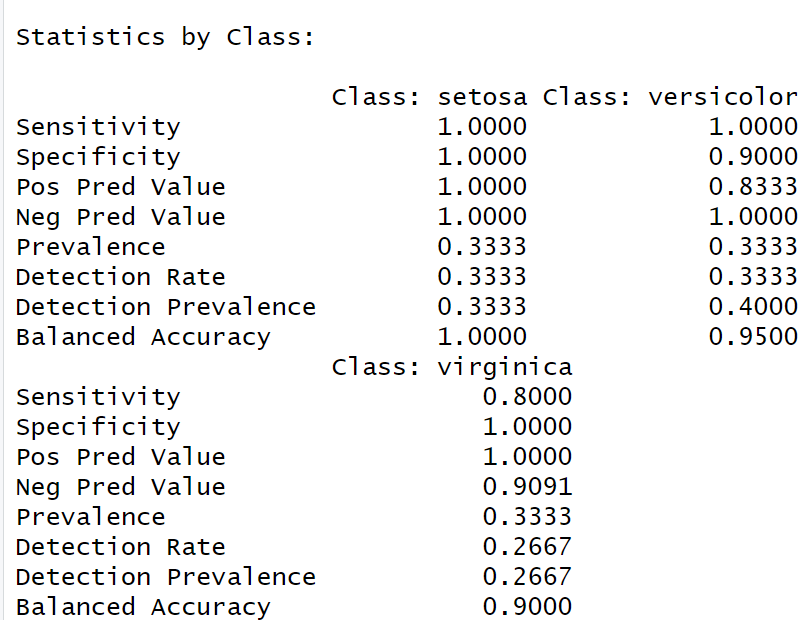
conf\_mat <- confusionMatrix(pred, test\_labels) print(conf\_mat)

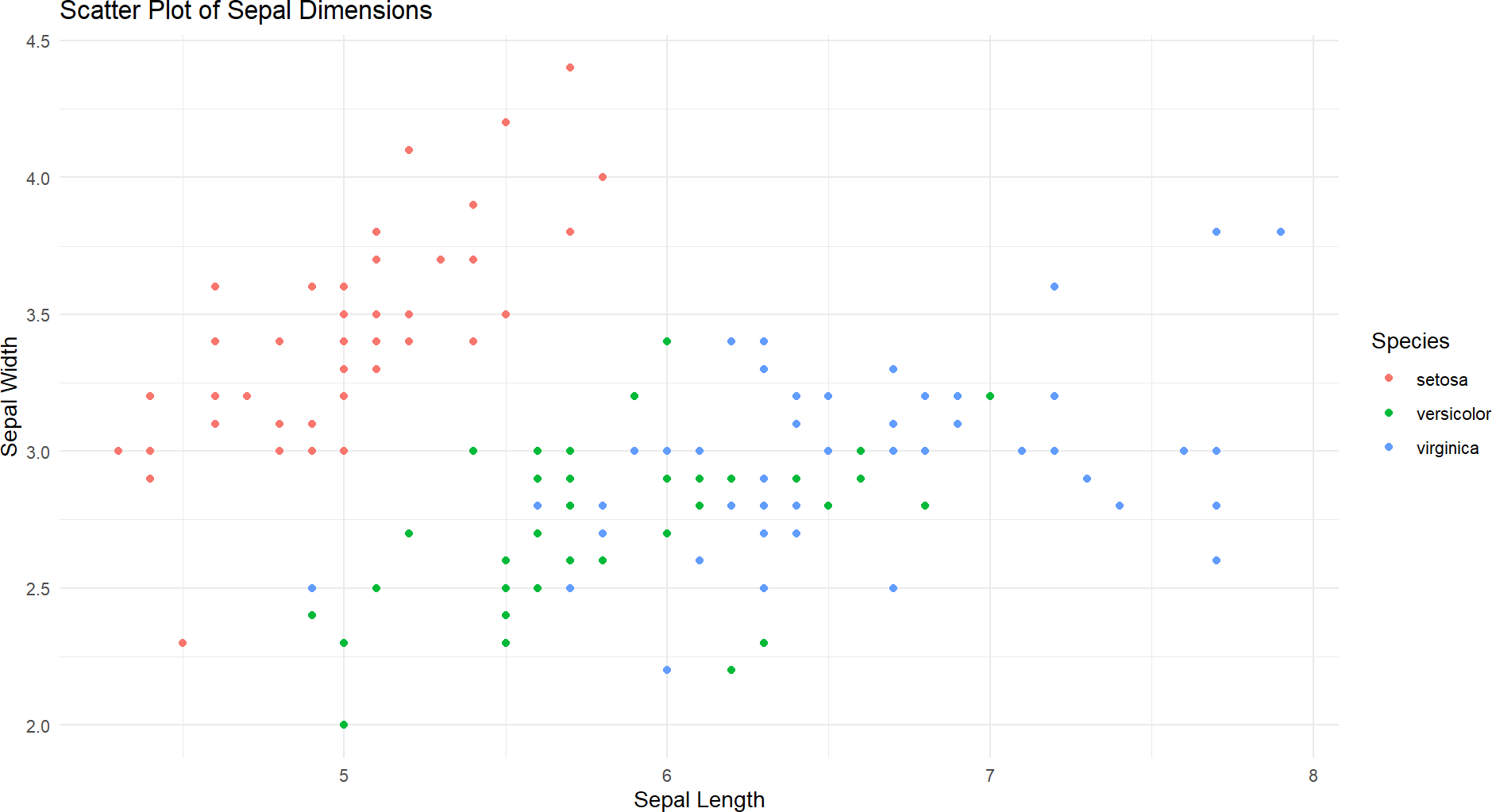
# Output:

****



****



****

**Result:**

The Naïve Bayes Classification is Successfully Implemented.

**IMPLEMENT LINEAR REGRESSION IN R**

**EXP NO: 7**

# Aim:

**EXP NO: 1 Title**

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)

# 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)

# 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, ]

# 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)

# 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)

# 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")

# 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,

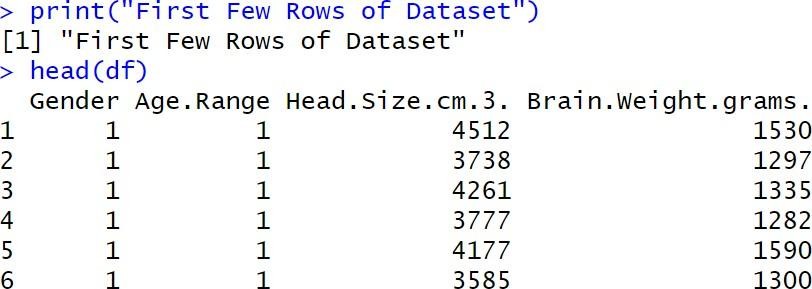
xlab = "Head Size (cm³)",

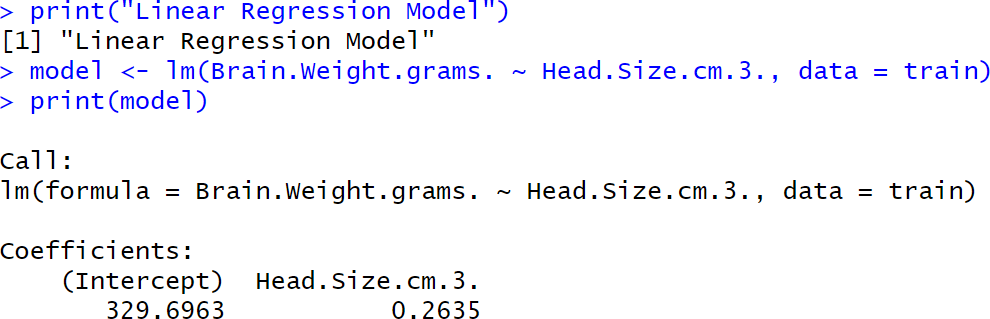
ylab = "Brain Weight (grams)",

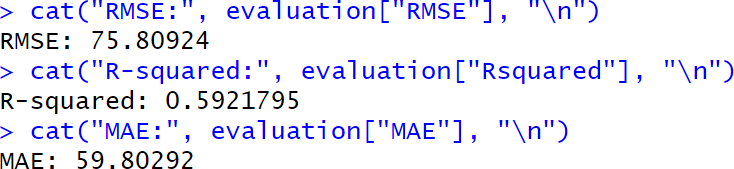
main = "Head Size vs Brain Weight with Regression Line")

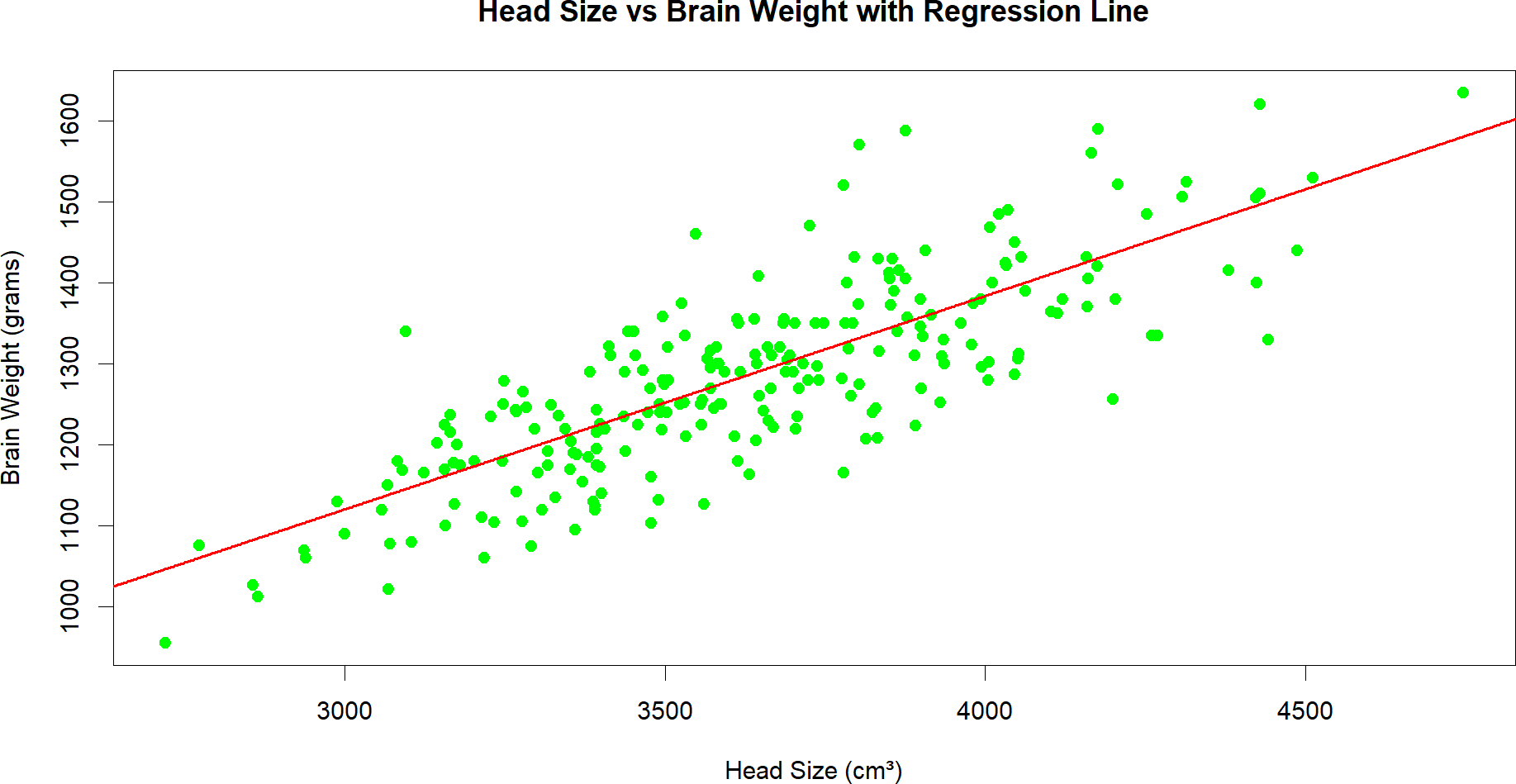
lines(pred\_line$Head.Size.cm.3., pred\_line$Brain.Weight.grams., col = "red", lwd = 2)

# Output:

****



****

****

**Result:**

The Linear Regression is Successfully Implemented.

**IMPLEMENT K-MEANS CLUSTERING ALGORITHM IN R**

**EXP NO: 8**

# 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)

# 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)

# 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")

# 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)

# 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)

# 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")

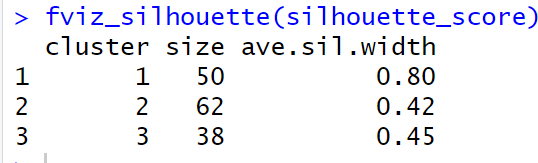
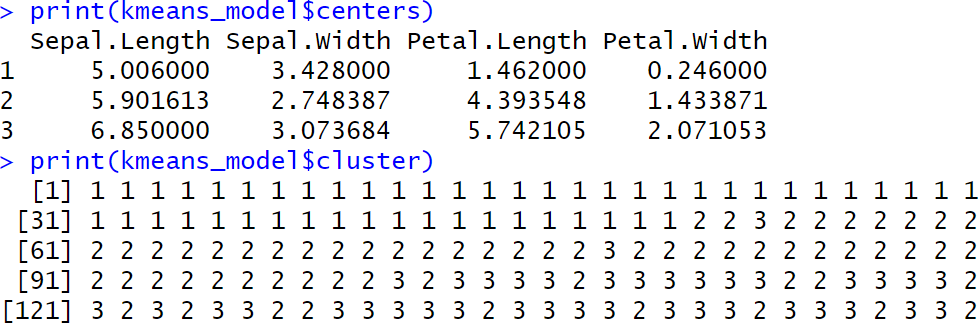
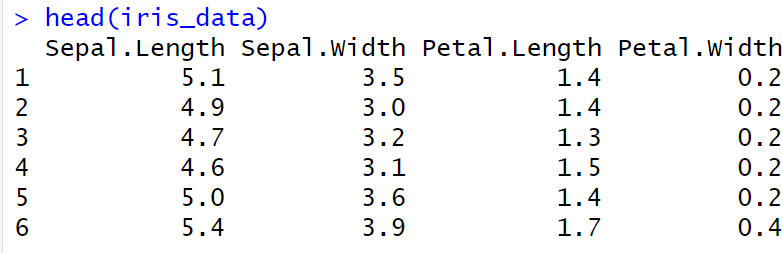
# Evaluate the Clustering (Silhouette Analysis)

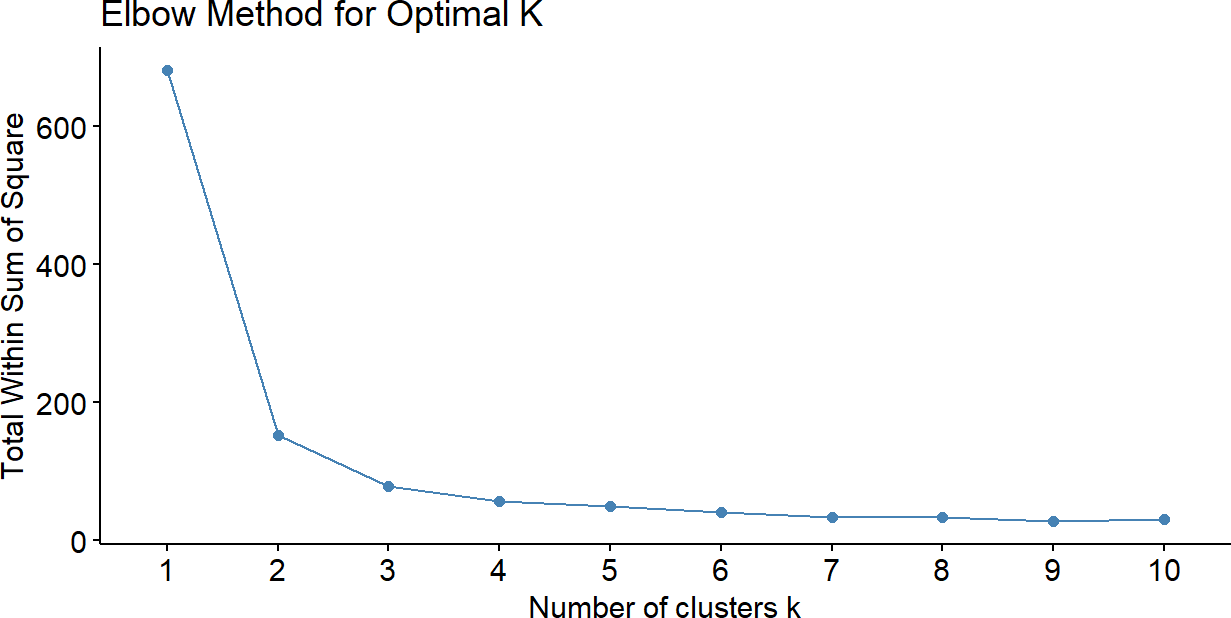
* + Perform silhouette analysis to assess the quality of the clustering.

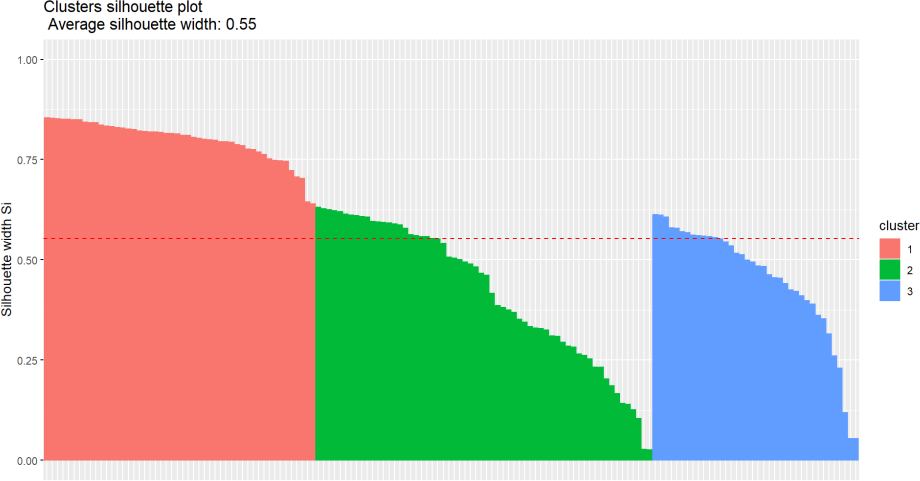
# Code:

silhouette\_score <- silhouette(kmeans\_model$cluster, dist(iris\_data)) fviz\_silhouette(silhouette\_score)

# Output:

****



****

**Result:**

The Kmeans is Successfully Implemented.