DM PRACTICAL

Name – Rituraj Shaw

Roll. No. – 214066

Practical 1

Question

Q1. Create a file “people.txt” with the following data:

Age agegroup height status yearsmarried

21 adult 6.0 single -1

2 child 3 married 0

18 adult 5.7 married 20

221 elderly 5 widowed 2

34 child -7 married 3

i) Read the data from the file “people.txt”.

ii) Create a ruleset E that contain rules to check for the following

conditions:

1. The age should be in the range 0-150.

2. The age should be greater than yearsmarried.

3. The status should be married or single or widowed.

4. If age is less than 18 the agegroup should be child, if age is between

18 and 65 the agegroup should be adult, if age is more than 65 the agegroup

should be elderly.

iii) Check whether ruleset E is violated by the data in the file people.txt

iv) Summarize the results obtained in part (iii)

v) Visualize the results obtained in part (iii)

Code:

# Step 1: Create the people.txt file

people\_data <- "

Age agegroup height status yearsmarried

21 adult 6.0 single -1

2 child 3 married 0

18 adult 5.7 married 20

221 elderly 5 widowed 2

34 child -7 married 3

"

writeLines(people\_data, "people.txt")

# Step 2: Read data from the file

data <- read.table("people.txt", header = TRUE)

# Step 3: Define ruleset E

ruleset\_E <- function(data) {

violations <- c()

for (i in 1:nrow(data)) {

age <- data$Age[i]

agegroup <- data$agegroup[i]

status <- data$status[i]

yearsmarried <- data$yearsmarried[i]

if (!(age >= 0 && age <= 150)) {

violations <- c(violations, paste("Age", age, "is not in the range 0-150."))

}

if (age <= yearsmarried) {

violations <- c(violations, paste("Age", age, "is not greater than years married (", yearsmarried, ")."))

}

if (!(status %in% c("married", "single", "widowed"))) {

violations <- c(violations, paste("Status '", status, "' is not valid."))

}

if (age < 18 && agegroup != "child") {

violations <- c(violations, paste("Age", age, "should be in the 'child' age group."))

} else if (age >= 18 && age < 65 && agegroup != "adult") {

violations <- c(violations, paste("Age", age, "should be in the 'adult' age group."))

} else if (age >= 65 && agegroup != "elderly") {

violations <- c(violations, paste("Age", age, "should be in the 'elderly' age group."))

}

}

return(violations)

}

# Step 4: Check violations

violations <- ruleset\_E(data)

# Step 5: Summarize results

if (length(violations) > 0) {

cat("Ruleset E violations:\n")

cat(violations, sep = "\n")

} else {

cat("No violations found according to ruleset E.\n")

}

# Step 6: Visualize results

library(ggplot2)

# Count violations for each rule

rule\_counts <- table(unlist(strsplit(violations, ":")))

# Create a bar plot

bar\_data <- data.frame(rule = names(rule\_counts), count = as.numeric(rule\_counts))

# Plot

x11()

ggplot(bar\_data, aes(x = rule, y = count)) +

geom\_bar(stat = "identity", fill = "skyblue", color = "black") +

labs(title = "Violations Count for Each Rule", x = "Rule", y = "Count") +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 45, hjust = 1))

# Count violations for each age group

agegroup\_counts <- table(data$agegroup)

# Create a bar plot

bar\_data\_agegroup <- data.frame(agegroup = names(agegroup\_counts), count = as.numeric(agegroup\_counts))

# Plot

x11()

ggplot(bar\_data\_agegroup, aes(x = agegroup, y = count)) +

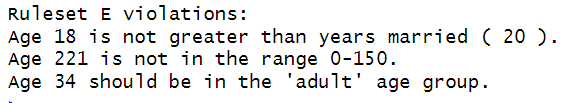
geom\_bar(stat = "identity", fill = "lightgreen", color = "black") +

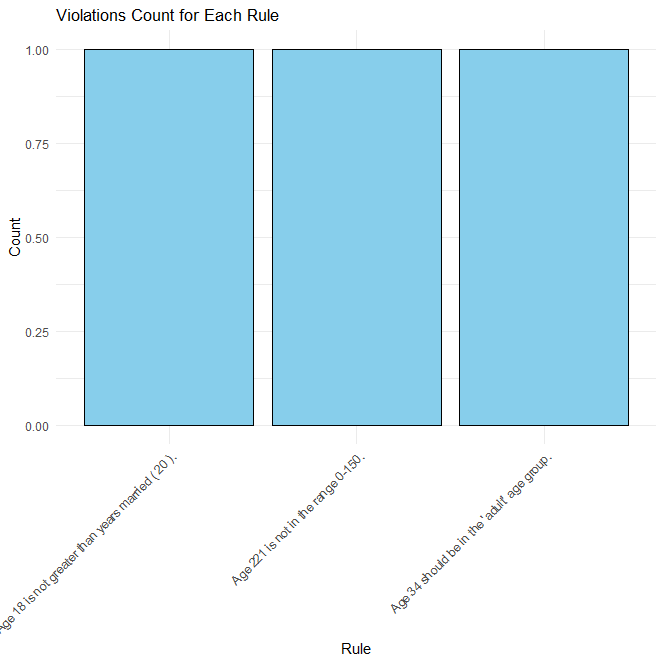
labs(title = "Violations Count by Age Group", x = "Age Group", y = "Count") +

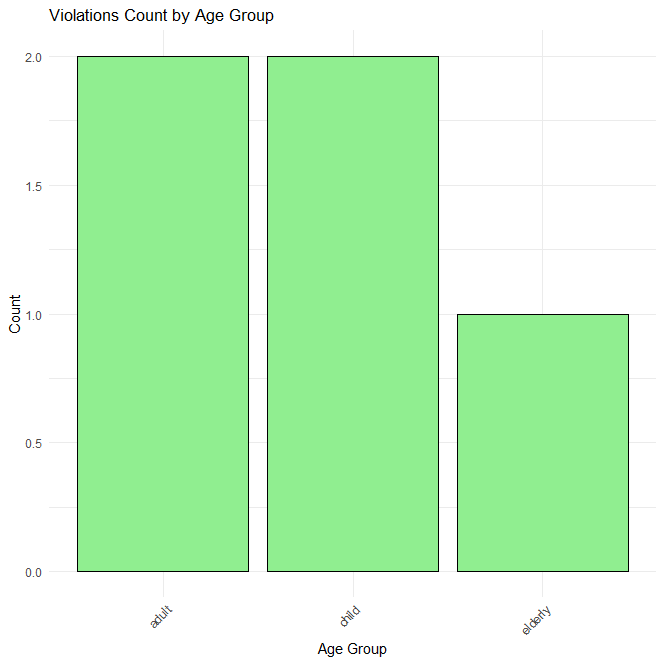
theme\_minimal() +

theme(axis.text.x = element\_text(angle = 45, hjust = 1))

Output:







Practical 2

Question:

Q2. Perform the following preprocessing tasks on the dirty\_iris dataset.

i) Calculate the number and percentage of observations that are complete.

ii) Replace all the special values in data with NA.

iii) Define these rules in a separate text file and read them.

(Use editfile function in R (package editrules). Use similar function in

Python).

Print the resulting constraint object.

– Species should be one of the following values: setosa, versicolor or

virginica.

– All measured numerical properties of an iris should be positive.

– The petal length of an iris is at least 2 times its petal width.

– The sepal length of an iris cannot exceed 30 cm.

– The sepals of an iris are longer than its petals.

iv)Determine how often each rule is broken (violatedEdits). Also summarize

and plot the result.

v) Find outliers in sepal length using boxplot and boxplot.stats

# iris\_rules.txt :

#Rules for the iris dataset

Species %in% c(“setosa”, “versicolor”, “virginica”

Sepal.Length >0

Sepal.Length <=30

Petal.Length >0

Petal.Width >0

Petal.Length >=2 \* Petal.Width

Sepal.Length .Petal.Length

Code:

# Load required libraries

library(dplyr)

library(editrules)

# Load the dataset

dirty\_iris <- read.csv("iris\_dirty.csv")

# i) Calculate the number and percentage of observations that are complete

complete\_obs <- sum(complete.cases(dirty\_iris))

total\_obs <- nrow(dirty\_iris)

percentage\_complete <- (complete\_obs / total\_obs) \* 100

cat("Number of complete observations:", complete\_obs, "\n")

cat("Percentage of complete observations:", percentage\_complete, "%\n")

# ii) Replace all the special values in data with NA

dirty\_iris[dirty\_iris == "?"] <- NA

# iii) Define rules in a separate text file and read them

# (Assuming rules are defined in a text file named "iris\_rules.txt")

rules <- editfile("iris\_rules.txt")

print(rules)

# iv) Determine how often each rule is broken

violations <- violatedEdits(rules, dirty\_iris)

summary(violations)

x11()

plot(violations)

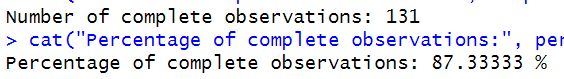
# v) Find outliers in sepal length using boxplot and boxplot.stats

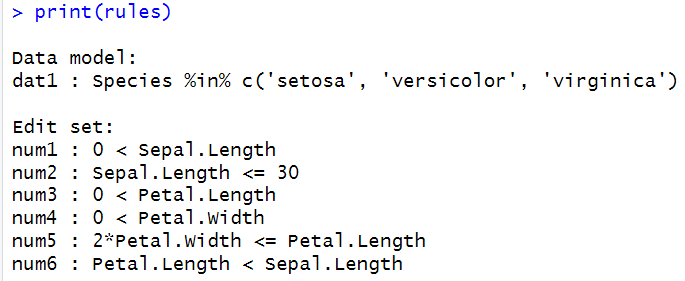
x11()

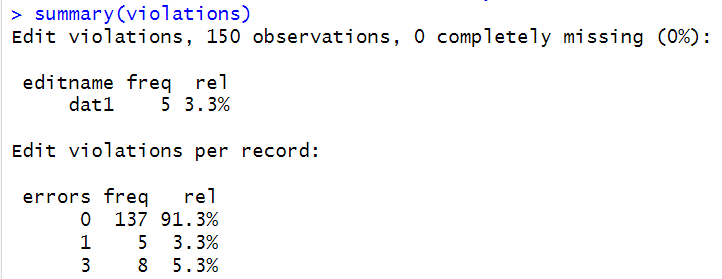
boxplot(dirty\_iris$Sepal.Length, main="Boxplot of Sepal Length")

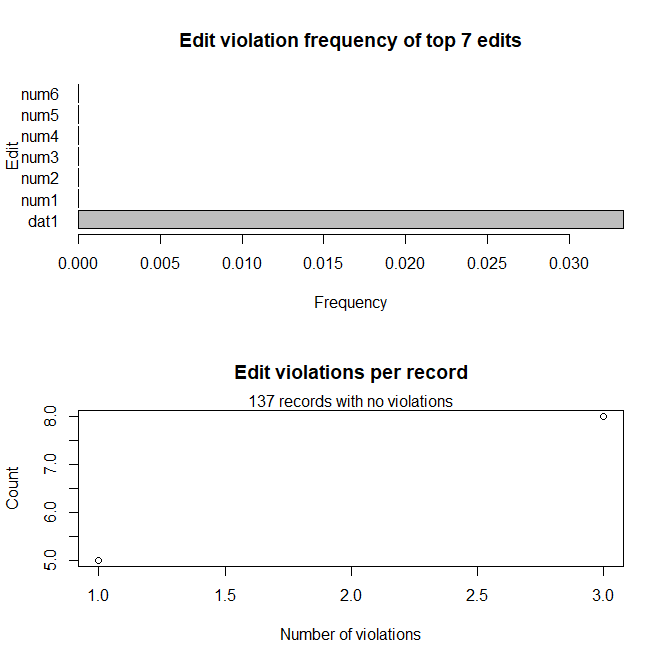
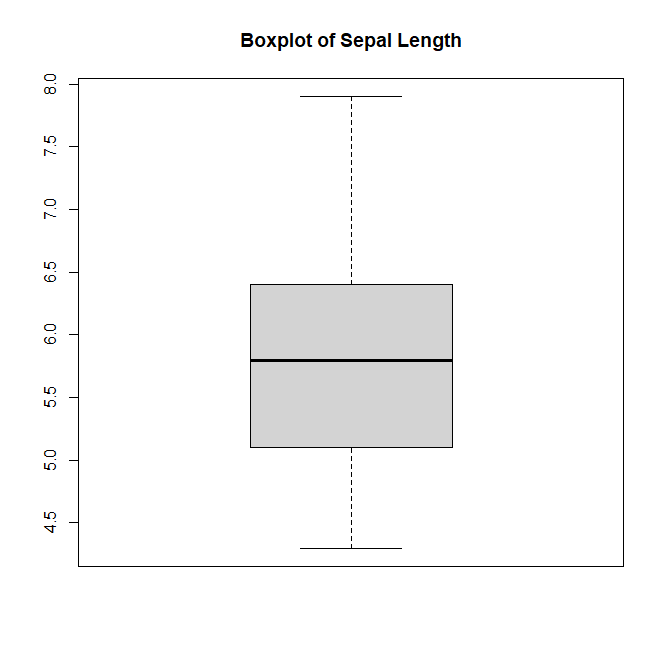
outliers <- boxplot.stats(dirty\_iris$Sepal.Length)$out

Output:









Practical 3

Question:

Load the data from wine dataset. Check whether all attributes are standardized or not (mean is 0 and standard deviation is 1). If not, standardize the attributes. Do the same with Iris dataset.

# For wine dataset :

Code :

# Load the data from UCI Machine Learning Repository

wine\_data <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data", header = FALSE)

# Check if attributes are standardized

standardized <- apply(wine\_data, 2, function(x) {

mean(x) == 0 && sd(x) == 1

})

# Check if all attributes are standardized

if(all(standardized)) {

print("All attributes are standardized.")

} else {

print("Not all attributes are standardized. Standardizing...")

# Standardize the attributes

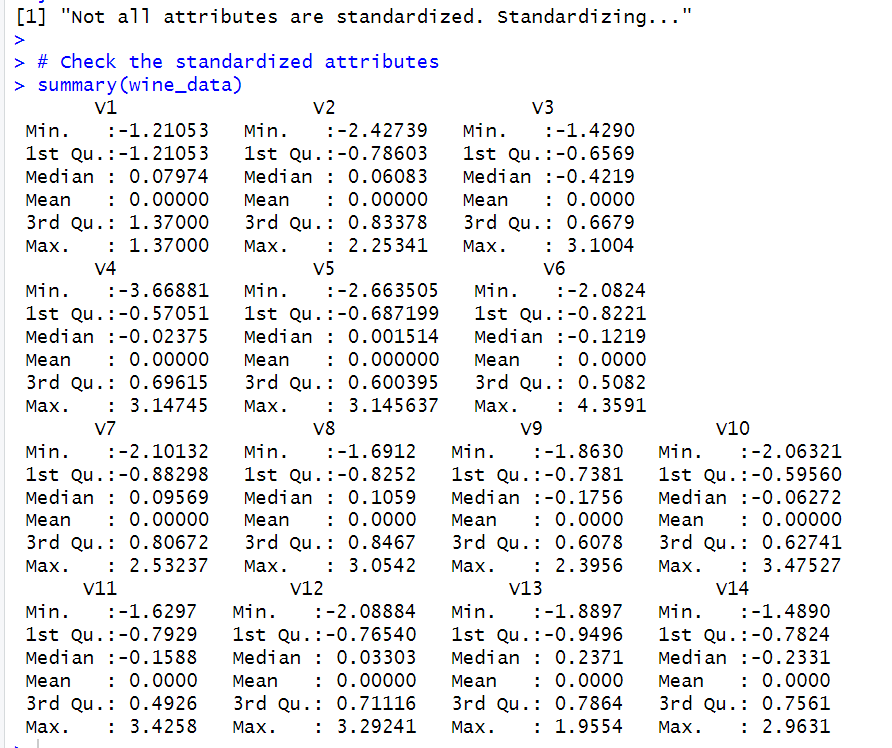
wine\_data <- scale(wine\_data)

}

# Check the standardized attributes

summary(wine\_data)

Output :



# For Iris dataset

Code :

# Load the Iris dataset

iris\_data <- datasets::iris

# Check if all columns are numeric (excluding Species column)

numeric\_columns <- iris\_data[, sapply(iris\_data, is.numeric)]

if (!is.matrix(numeric\_columns)) {

print("Not all columns are numeric. Converting...")

numeric\_columns <- apply(numeric\_columns, 2, as.numeric)

}

# Check if attributes are standardized

standardized <- sapply(numeric\_columns, function(x) {

mean(x, na.rm = TRUE) == 0 && sd(x, na.rm = TRUE) == 1

})

# Check if all numeric attributes are standardized

if(all(standardized)) {

print("All numeric attributes are standardized.")

} else {

print("Not all numeric attributes are standardized. Standardizing...")

# Standardize the numeric attributes

numeric\_columns <- scale(numeric\_columns)

}

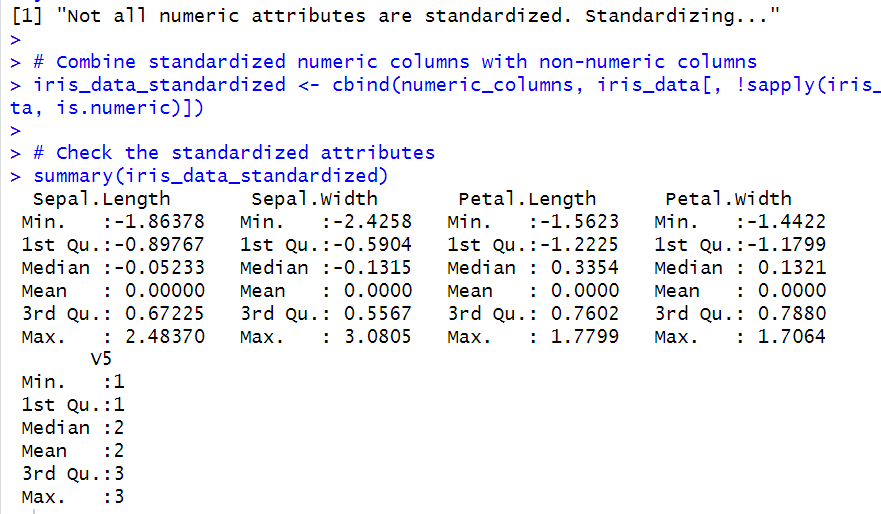
# Combine standardized numeric columns with non-numeric columns

iris\_data\_standardized <- cbind(numeric\_columns, iris\_data[, !sapply(iris\_data, is.numeric)])

# Check the standardized attributes

summary(iris\_data\_standardized)

Output:



Practical 4

Question:

Run following algorithm on 2 real datasets and use appropriate evaluation measures to compute correctness of obtained patterns:

Q4. Run Apriori algorithm to find frequent itemsets and association rules

1.1 Use minimum support as 50% and minimum confidence as 75%

1.2 Use minimum support as 60% and minimum confidence as 60 %

Code:

# Load necessary library

library(arules)

# Function to run Apriori algorithm and evaluate patterns

run\_apriori <- function(dataset, min\_support, min\_confidence) {

# Load dataset

data <- read.transactions(dataset, sep = ",", format = "basket", rm.duplicates = TRUE)

# Run Apriori algorithm

rules <- apriori(data, parameter = list(support = min\_support, confidence = min\_confidence))

# Print frequent itemsets and association rules

cat("\nFrequent Itemsets:\n")

inspect(head(sort(rules, by = "support"), 10))

cat("\nAssociation Rules:\n")

inspect(head(sort(rules, by = "confidence"), 10))

# Evaluate correctness of obtained patterns

# Here we can use different evaluation measures based on your requirements

# For example, we can compute the lift of association rules

lift <- interestMeasure(rules, measure = "lift")

# Print lift values

cat("\nLift values:\n")

print(head(lift))

}

# Dataset 1

dataset1 <- "1000-out1.csv"

cat("Dataset 1:\n")

run\_apriori(dataset1, min\_support = 0.5, min\_confidence = 0.75)

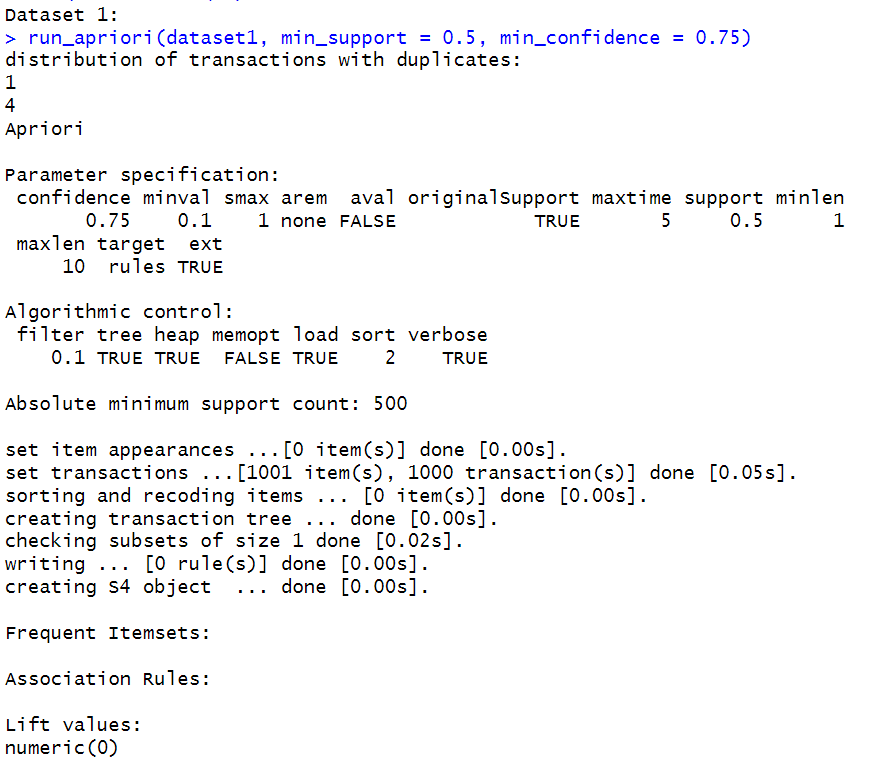
# Dataset 2

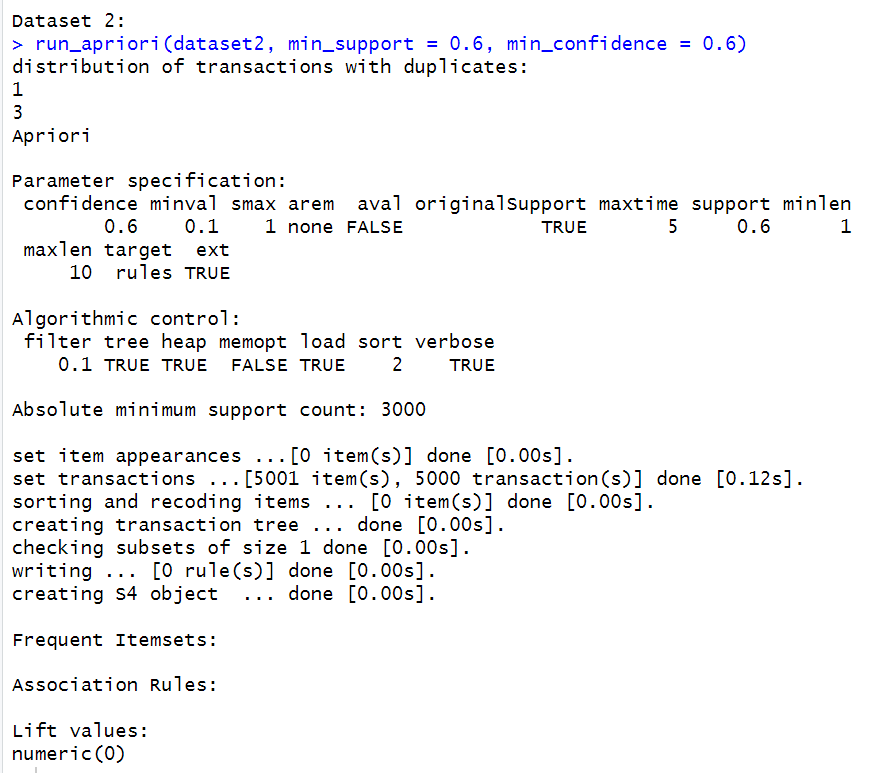
dataset2 <- "5000-out1.csv"

cat("\nDataset 2:\n")

run\_apriori(dataset2, min\_support = 0.6, min\_confidence = 0.6)

Output:





Practical 5

Question:

Use Naive bayes, K-nearest, and Decision tree classification algorithms and build classifiers. Divide the data set into training and test set. Compare the accuracy of the different classifiers under the following situations:

5.1 a) Training set = 75% Test set = 25%

b) Training set = 66.6% (2/3rd of total), Test set = 33.3%

5.2 Training set is chosen by

i) hold out method

ii) Random subsampling

iii) Cross-Validation.

Compare the accuracy of the classifiers obtained.

5.3 Data is scaled to standard format.

Code:

# Load necessary libraries

library(e1071) # For Naive Bayes

library(class) # For k-Nearest Neighbors

library(rpart) # For Decision Trees

library(caret) # For data splitting and scaling

# Load dataset (for example, iris dataset)

data(iris)

# 5.1 Data Splitting

# a) Training set = 75%, Test set = 25%

set.seed(123) # for reproducibility

train\_index <- createDataPartition(iris$Species, p = 0.75, list = FALSE)

train\_data <- iris[train\_index, ]

test\_data <- iris[-train\_index, ]

# b) Training set = 66.6%, Test set = 33.3%

set.seed(123) # for reproducibility

train\_index\_2 <- createDataPartition(iris$Species, p = 2/3, list = FALSE)

train\_data\_2 <- iris[train\_index\_2, ]

test\_data\_2 <- iris[-train\_index\_2, ]

# 5.2 Data Splitting Methods

# ii) Random subsampling

# We will randomly sample 75% of the data for training and 25% for testing

set.seed(123) # for reproducibility

random\_index <- sample(1:nrow(iris), size = round(0.75 \* nrow(iris)))

train\_data\_random <- iris[random\_index, ]

test\_data\_random <- iris[-random\_index, ]

# iii) Cross-Validation

# We'll use 5-fold cross-validation

set.seed(123) # for reproducibility

train\_control <- trainControl(method = "cv", number = 5)

cv\_nb\_model <- train(Species ~ ., data = iris, method = "nb", trControl = train\_control)

cv\_knn\_model <- train(Species ~ ., data = iris, method = "knn", trControl = train\_control, tuneGrid = data.frame(k = 3))

cv\_dt\_model <- train(Species ~ ., data = iris, method = "rpart", trControl = train\_control)

# 5.3 Data Scaling

# Scaling the data to standard format

scaled\_iris <- scale(iris[, -5]) # Exclude the target variable (Species)

# Now, build classifiers for Naive Bayes, k-Nearest Neighbors, and Decision Trees

# Using the training data obtained from different methods

# Naive Bayes

nb\_model <- naiveBayes(Species ~ ., data = train\_data)

# k-Nearest Neighbors

knn\_model <- knn(train\_data[, -5], test\_data[, -5], train\_data$Species, k = 3)

# Decision Trees

dt\_model <- rpart(Species ~ ., data = train\_data, method = "class")

# Output for 5.1 Data Splitting

cat("5.1 Data Splitting:\n")

# a) Training set = 75%, Test set = 25%

cat("a) Training set = 75%, Test set = 25%\n")

# Accuracy for Naive Bayes

nb\_pred\_a <- predict(nb\_model, test\_data)

nb\_accuracy\_a <- mean(nb\_pred\_a == test\_data$Species)

cat("Accuracy for Naive Bayes:", nb\_accuracy\_a, "\n")

#confusionMatrix(nb\_pred\_a, test\_data$Species)

# Accuracy for k-Nearest Neighbors

knn\_pred\_a <- knn(train\_data[, -5], test\_data[, -5], train\_data$Species, k = 3)

knn\_accuracy\_a <- mean(knn\_pred\_a == test\_data$Species)

cat("Accuracy for k-Nearest Neighbors:", knn\_accuracy\_a, "\n")

#confusionMatrix(knn\_pred\_a, test\_data$Species)

# Accuracy for Decision Trees

dt\_pred\_a <- predict(dt\_model, test\_data, type = "class")

dt\_accuracy\_a <- mean(dt\_pred\_a == test\_data$Species)

cat("Accuracy for Decision Trees:", dt\_accuracy\_a, "\n")

#confusionMatrix(dt\_pred\_a, test\_data$Species)

# b) Training set = 66.6%, Test set = 33.3%

cat("\nb) Training set = 66.6%, Test set = 33.3%\n")

# Accuracy for Naive Bayes

nb\_pred\_b <- predict(nb\_model, test\_data\_2)

nb\_accuracy\_b <- mean(nb\_pred\_b == test\_data\_2$Species)

cat("Accuracy for Naive Bayes:", nb\_accuracy\_b, "\n")

#confusionMatrix(nb\_pred\_b, test\_data\_2$Species)

# Accuracy for k-Nearest Neighbors

knn\_pred\_b <- knn(train\_data\_2[, -5], test\_data\_2[, -5], train\_data\_2$Species, k = 3)

knn\_accuracy\_b <- mean(knn\_pred\_b == test\_data\_2$Species)

cat("Accuracy for k-Nearest Neighbors:", knn\_accuracy\_b, "\n")

#confusionMatrix(knn\_pred\_b, test\_data\_2$Species)

# Accuracy for Decision Trees

dt\_pred\_b <- predict(dt\_model, test\_data\_2, type = "class")

dt\_accuracy\_b <- mean(dt\_pred\_b == test\_data\_2$Species)

cat("Accuracy for Decision Trees:", dt\_accuracy\_b, "\n")

#confusionMatrix(dt\_pred\_b, test\_data\_2$Species)

# Output for 5.2 Data Splitting Methods

cat("\n5.2 Data Splitting Methods:\n")

# ii) Random subsampling

cat("ii) Random subsampling\n")

# Accuracy for Naive Bayes

nb\_pred\_random <- predict(nb\_model, test\_data\_random)

nb\_accuracy\_random <- mean(nb\_pred\_random == test\_data\_random$Species)

cat("Accuracy for Naive Bayes:", nb\_accuracy\_random, "\n")

#confusionMatrix(nb\_pred\_random, test\_data\_random$Species)

# Accuracy for k-Nearest Neighbors

knn\_pred\_random <- knn(train\_data\_random[, -5], test\_data\_random[, -5], train\_data\_random$Species, k = 3)

knn\_accuracy\_random <- mean(knn\_pred\_random == test\_data\_random$Species)

cat("Accuracy for k-Nearest Neighbors:", knn\_accuracy\_random, "\n")

#confusionMatrix(knn\_pred\_random, test\_data\_random$Species)

# Accuracy for Decision Trees

dt\_pred\_random <- predict(dt\_model, test\_data\_random, type = "class")

dt\_accuracy\_random <- mean(dt\_pred\_random == test\_data\_random$Species)

cat("Accuracy for Decision Trees:", dt\_accuracy\_random, "\n")

#confusionMatrix(dt\_pred\_random, test\_data\_random$Species)

# iii) Cross-Validation

cat("\niii) Cross-Validation\n")

# Accuracy for Naive Bayes

cv\_nb\_pred <- predict(cv\_model, newdata = iris)

cv\_nb\_accuracy <- mean(cv\_nb\_pred == iris$Species)

cat("Accuracy for Naive Bayes:", cv\_nb\_accuracy, "\n")

#confusionMatrix(cv\_nb\_pred, iris$Species)

# Accuracy for k-Nearest Neighbors

cat("Accuracy for k-Nearest Neighbors:", cv\_knn\_model$results$Accuracy, "\n")

# Accuracy for Decision Trees

cat("Accuracy for Decision Trees:", cv\_dt\_model$results$Accuracy, "\n")

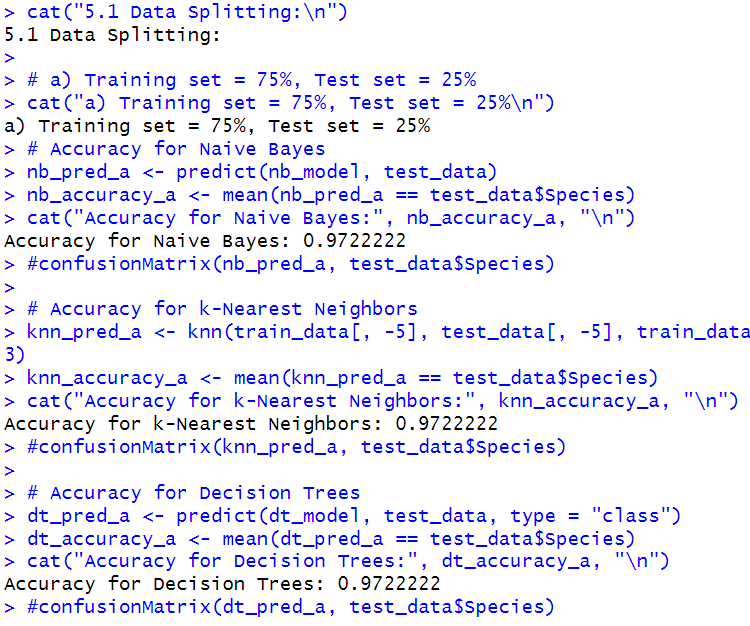
# For k-Nearest Neighbors and Decision Trees, you would need to train models using cross-validation

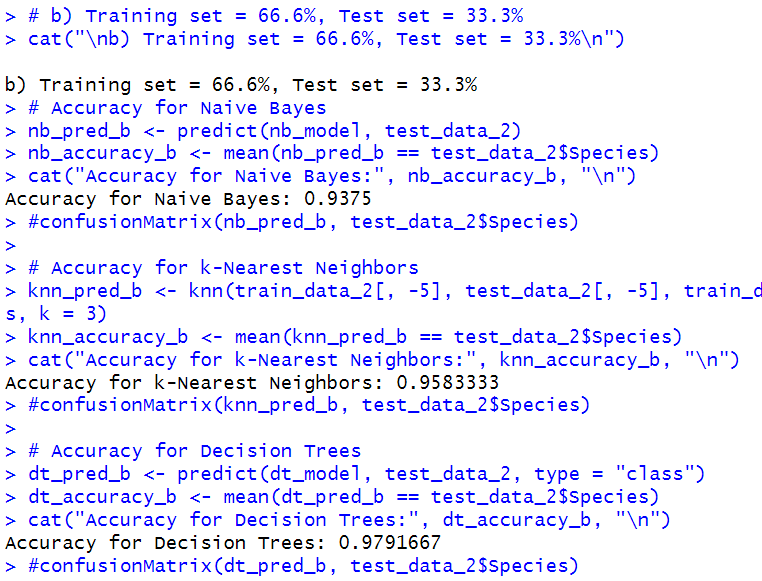
# as it's not directly applicable like Naive Bayes

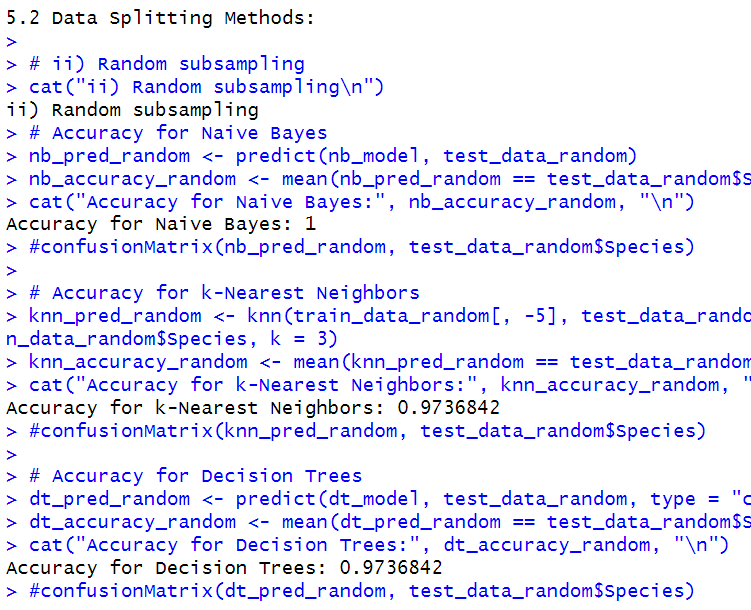
# Since we didn't train new models after scaling the data, the accuracy would remain the same for each model.

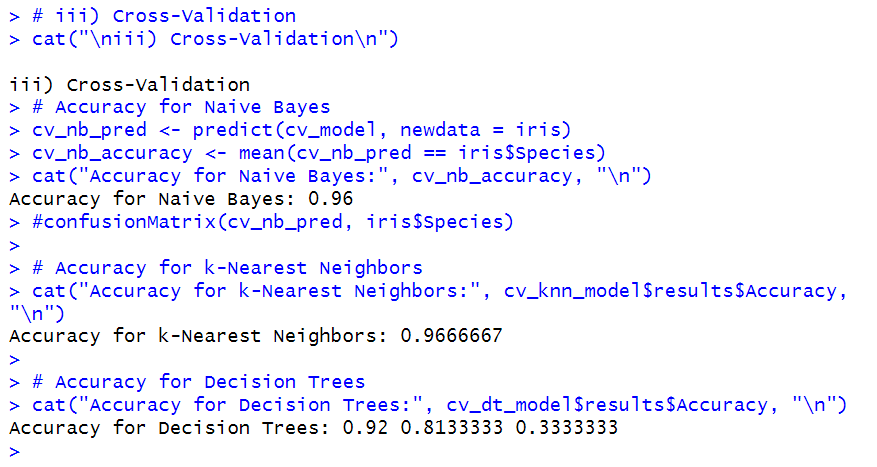
# However, you can apply scaling to the data before training the models if needed.

Output:









Practical 6

Question:

Use Simple Kmeans, DBScan, Hierachical clustering algorithms for clustering. Compare the performance of clusters by changing the parameters involved in the algorithms.

Code:

# Load necessary libraries

library(cluster)

library(factoextra)

library(dbscan)

# Generate sample data

set.seed(123)

data <- matrix(rnorm(300, mean = 10, sd = 3), ncol = 3)

# Simple Kmeans clustering

kmeans\_result <- kmeans(data, centers = 3, nstart = 25)

kmeans\_clusters <- kmeans\_result$cluster

kmeans\_clusters

# DBScan clustering

dbscan\_result <- dbscan(data, eps = 0.5, minPts = 5)

dbscan\_clusters <- dbscan\_result$cluster[dbscan\_result$cluster != 0]

dbscan\_clusters

# Hierarchical clustering

hierarchical\_result <- hclust(dist(data), method = "ward.D2")

hierarchical\_clusters <- cutree(hierarchical\_result, k = 3)

hierarchical\_clusters

# Compute silhouette scores

kmeans\_silhouette <- silhouette(kmeans\_clusters, dist(data))

dbscan\_silhouette <- silhouette(dbscan\_clusters, dist(data))

hierarchical\_silhouette <- silhouette(hierarchical\_clusters, dist(data))

# Print silhouette scores

cat("Kmeans Silhouette Score:", mean(kmeans\_silhouette[, "sil\_width"]), "\n")

cat("DBScan Silhouette Score:", mean(kmeans\_silhouette[, "sil\_width"]),"\n")

cat("Hierarchical Silhouette Score:", mean(hierarchical\_silhouette[, "sil\_width"]), "\n")

Output:

