K-means algorithm

```
# Importing the dataset
dataset <- read.csv('D:\\nk\\Mall_Customers.csv')  # Reads the dataset from a CSV file
head(dataset)  # Displays the first few rows of the dataset
dataset <- dataset[4:5]  # Selects the columns 4 and 5 from the dataset (Annual Income and Spending Score)
head(dataset)  # Displays the first few rows of the updated dataset

# Compute the Within Cluster Sum of Squares (WCSS) for different number of clusters
wcss <- vector()  # Creates an empty vector to store the WCSS values
for (i in 1:10) {  # Loops through the number of clusters from 1 to 10
  wcss[i] <- sum(kmeans(dataset, i)$withinss)  # Performs K-Means clustering and calculates the WCSS for each number of clusters
}

# Plot the WCSS values
plot(1:10, wcss, type = 'b', main = paste('The Elbow Method'),
   xlab = 'Number of clusters', ylab = 'WSS')  # Plots the WCSS values against the number of clusters

# Fit K-Means to the dataset with 5 clusters
kmeans_model <- kmeans(x = dataset, centers = 5)  # Performs K-Means clustering with 5 clusters on the dataset
y_kmeans <- kmeans_model$cluster  # Retrieves the cluster labels for each data point
```

```r
# Visualize the clusters
library("cluster")
clusplot(dataset, y_kmeans, lines = 0, shade = TRUE, color = TRUE, labels = 2,
         main = paste('Clusters of customers'),
         xlab = "Annual Income",
         ylab = "Spending Score")  # Creates a cluster plot to visualize the clusters
```

Prac1B
AprioriAlgorithm .

```r
# Installing required packages
install.packages("arules")
install.packages("arulesViz")
install.packages("RColorBrewer")

# Loading libraries
library(arules)
library(arulesViz)
library(RColorBrewer)

# Importing the dataset
data(Groceries)  # Loads the "Groceries" dataset from the arules package
Groceries  # Displays the dataset
summary(Groceries)  # Provides a summary of the dataset
class(Groceries)  # Displays the class of the dataset

# Generating association rules using apriori()
```

```
rules = apriori(Groceries, parameter = list(supp = 0.02, conf = 0.2))  # Performs association rule mining using the Apriori algorithm

summary(rules)  # Provides a summary of the generated rules


# Inspecting the first 10 rules

inspect(rules[1:10])  # Displays the details of the first 10 rules


# Generating item frequency plot

arules::itemFrequencyPlot(Groceries, topN = 20,

                          col = brewer.pal(8, 'Pastel2'),

                          main = 'Relative Item Frequency Plot',

                          type = "relative",

                          ylab = "Item Frequency (Relative)")  # Creates a plot showing the relative item frequency


# Generating frequent itemsets with length 2

itemsets = apriori(Groceries, parameter = list(minlen = 2, maxlen = 2, support = 0.02, target = "frequent itemsets"))

summary(itemsets)  # Provides a summary of the generated frequent itemsets


# Inspecting the first 10 frequent itemsets

inspect(itemsets[1:10])  # Displays the details of the first 10 frequent itemsets


# Generating frequent itemsets with length 3

itemsets_3 = apriori(Groceries, parameter = list(minlen = 3, maxlen = 3, support = 0.02, target = "frequent itemsets"))

summary(itemsets_3)  # Provides a summary of the generated frequent itemsets with length 3
```

```
# Inspecting the frequent itemsets with length 3

inspect(itemsets_3)  # Displays the details of the frequent itemsets with length 3
```

Prac2A
Logistic Regression.

```
# Importing the dataset

college <- read.csv("https://raw.githubusercontent.com/ropensci/datapack/main/inst/extdata/pkg-example/binary.csv")  # Reads the dataset from the specified URL

head(college)  # Displays the first few rows of the dataset

nrow(college)  # Provides the number of rows in the dataset


# Installing and loading required packages

install.packages("caTools")  # Installs the "caTools" package

library(caTools)  # Loads the "caTools" package


# Splitting the dataset into training and test sets

split <- sample.split(college, SplitRatio = 0.75)  # Splits the dataset into a training set and a test set using a specified split ratio

split  # Displays the split result (logical vector indicating the split for each row)


training_reg <- subset(college, split == "TRUE")  # Creates the training set by subsetting the "college" dataset based on the split

test_reg <- subset(college, split == "FALSE")  # Creates the test set by subsetting the "college" dataset based on the split
```

```
# Fitting a logistic regression model

fit_logistic_model <- glm(admit ~ ., data = training_reg, family = "binomial")  # Fits a logistic regression model to the training set using the "glm" function

# The formula "admit ~ ." specifies that "admit" is the response variable and all other variables in the dataset are predictors


# Extracting coefficients from the logistic regression model

coef(fit_logistic_model)["gre"]  # Extracts the coefficient for the "gre" variable

coef(fit_logistic_model)["gpa"]  # Extracts the coefficient for the "gpa" variable

coef(fit_logistic_model)["rank"]  # Extracts the coefficient for the "rank" variable


# Predicting on the test set

predict_reg <- predict(fit_logistic_model, test_reg, type = "response")  # Generates predictions on the test set using the fitted logistic regression model

predict_reg  # Displays the predicted probabilities


# Creating conditional density plots

cdplot(as.factor(admit) ~ gpa, data = college)  # Creates a conditional density plot of "admit" against "gpa"

cdplot(as.factor(admit) ~ gre, data = college)  # Creates a conditional density plot of "admit" against "gre"

cdplot(as.factor(admit) ~ rank, data = college)  # Creates a conditional density plot of "admit" against "rank"


# Thresholding predicted probabilities

predict_reg <- ifelse(predict_reg > 0.5, 1, 0)  # Thresholds the predicted probabilities at 0.5 to obtain binary predictions
```

predict_reg *# Displays the binary predictions*

 # Creating a confusion matrix

table(test_reg$admit, predict_reg) *# Creates a confusion matrix by comparing the actual admissions status with the predicted binary outcomes*

Prac2B

MULTIPLE REGRESSION.

 # Importing the dataset

college <- read.csv("https://raw.githubusercontent.com/csquared/udacity-dlnd/master/nn/binary.csv") *# Reads the dataset from the specified URL*

head(college) *# Displays the first few rows of the dataset*

nrow(college) *# Provides the number of rows in the dataset*

 # Installing and loading required packages

install.packages("caTools") *# Installs the "caTools" package*

library(caTools) *# Loads the "caTools" package*

 # Splitting the dataset into training and test sets

split <- sample.split(college, SplitRatio = 0.75) *# Splits the dataset into a training set and a test set using a specified split ratio*

split *# Displays the split result (logical vector indicating the split for each row)*

training_reg <- subset(college, split == "TRUE") *# Creates the training set by subsetting the "college" dataset based on the split* test_reg <- subset(college, split == "FALSE") *# Creates the test set by subsetting the "college" dataset based on the split*

 # Fitting a multiple linear regression model

fit_MRegressor_model <- lm(formula = admit ~ gre + gpa + rank, data = training_reg) *# Fits a multiple linear regression model to the training set using the "lm" function*

 # The formula "admit ~ gre + gpa + rank" specifies that "admit" is the response variable and "gre", "gpa", and "rank" are the predictor variables

 # Predicting on the test set

predict_reg <- predict(fit_MRegressor_model, newdata = test_reg) *# Generates predictions on the test set using the fitted linear regression model*

predict_reg *# Displays the predicted values*

 # Creating conditional density plots

cdplot(as.factor(admit) ~ gpa, data = college) *# Creates a conditional density plot of "admit" against "gpa"*

cdplot(as.factor(admit) ~ gre, data = college) *# Creates a conditional density plot of "admit" against "gre"*

cdplot(as.factor(admit) ~ rank, data = college) *# Creates a conditional density plot of "admit" against "rank"*

 # Thresholding predicted values

predict_reg <- ifelse(predict_reg > 0.5, 1, 0) *# Thresholds the predicted values at 0.5 to obtain binary predictions*

predict_reg *# Displays the binary predictions*

 # Creating a confusion matrix

table(test_reg$admit, predict_reg) *# Creates a confusion matrix by comparing the actual admissions status with the predicted binary outcomes*

Prac3A

Decision Tree Classification.

 # Importing the dataset

dataset = read.csv('F:/ Social_Network_Ads.csv') *# Reads the dataset from the specified file path*

dataset = dataset[3:5] # Selects columns 3 to 5 from the dataset (Age, EstimatedSalary, Purchased)

print(dataset) *# Displays the dataset*

 # Encoding the target feature as factor

dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1*))* *# Converts the "Purchased" column to a factor with levels 0 and 1*

 # Splitting the dataset into the Training set and Test set

install.packages('caTools') *# Installs the "caTools" package*

library(caTools) *# Loads the "caTools" package*

set.seed(123) *# Sets a seed for reproducibility*

split = sample.split(dataset$Purchased, SplitRatio = 0.75) *# Splits the dataset into a training set and a test set using a specified split ratio*

training_set = subset(dataset, split == TRUE) *# Creates the training set by subsetting the "dataset" based on the split*

test_set = subset(dataset, split == FALSE) *# Creates the test set by subsetting the "dataset" based on the split*

 # Feature Scaling

training_set[-3] = scale(training_set[-3]) *# Performs feature scaling on the training set by standardizing the Age and EstimatedSalary columns*

test_set[-3] = scale(test_set[-3]) *# Performs feature scaling on the test set by standardizing the Age and EstimatedSalary columns*

 # Fitting Decision Tree Classification to the Training set

install.packages('rpart') *# Installs the "rpart" package*

library(rpart) *# Loads the "rpart" package*

classifier = rpart(formula = Purchased ~ ., data = training_set) *# Fits a decision tree classification model to the training set*

 # Predicting the Test set results

y_pred = predict(classifier, newdata = test_set[-3], type = 'class') *# Generates predictions on the test set using the fitted decision tree model*

Making the Confusion Matrix

cm = table(test_set[, 3], y_pred) *# Creates a confusion matrix by comparing the actual "Purchased" values with the predicted values*

Visualising the Training set results

library(ElemStatLearn) *# Loads the "ElemStatLearn" library*

set = training_set *# Assigns the training_set to the variable "set"*

X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01) *# Defines the sequence of values for the X-axis (Age) by taking the minimum and maximum values from the "set" and adding/subtracting 1, with a step of 0.01*

X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01) *# Defines the sequence of values for the Y-axis (Estimated Salary) by taking the minimum and maximum values from the "set" and adding/subtracting 1, with a step of 0.01*

grid_set = expand.grid(X1, X2) *# Creates a grid of all possible combinations of values from X1 and X2*

colnames(grid_set) = c('Age', 'EstimatedSalary') # Assigns column names to the grid_set

y_grid = predict(classifier, newdata = grid_set, type = 'class') *# Predicts the class labels (0 or 1) for the grid_set using the fitted decision tree model*

plot(set[, -3], *# Plots the scatter plot of the training set without the "Purchased" column (Age vs. Estimated Salary)*

main = 'Decision Tree Classification (Training set)', *# Sets the main title of the plot*

xlab = 'Age', ylab = 'Estimated Salary',

 # Sets the labels for the x-axis and y-axis

xlim = range(X1), ylim = range(X2))

 # Sets the limits of the x-axis and y-axis based on the range of X1 and X2

contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)

 # Adds contour lines to the plot based on the class predictions in y_grid

points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))

 # Adds individual data points from the grid_set to the plot, colored based on the class predictions in y_grid

```
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
# Adds individual data points from the training set to the plot, with different shapes and colors based on the "Purchased" column

# Visualising the Test set results
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3], main = 'Decision Tree Classification (Test set)',
    xlab = 'Age', ylab = 'Estimated Salary',
    xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

# Plotting the tree
plot(classifier)
text(classifier)
```

Prac3B

Support vector machine.

```
# Importing the dataset
dataset = read.csv('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester 2/BigDataAnalytics/Dataset/Social_Network_Ads.csv')  # Reading the CSV file into the dataset variable
dataset = dataset[3:5]  # Selecting columns 3 to 5 from the dataset

# Encoding the target feature as factor
dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))  # Encoding the "Purchased" column as a factor with levels 0 and 1

# Splitting the dataset into the Training set and Test set
install.packages('caTools')  # Installing the caTools package
library(caTools)  # Loading the caTools library
set.seed(123)  # Setting the random seed for reproducibility
split = sample.split(dataset$Purchased, SplitRatio = 0.75)  # Splitting the dataset into training and test sets
training_set = subset(dataset, split == TRUE)  # Creating the training set using the split
test_set = subset(dataset, split == FALSE)  # Creating the test set using the split

# Feature Scaling
training_set[-3] = scale(training_set[-3])  # Scaling the numerical features in the training set
test_set[-3] = scale(test_set[-3])  # Scaling the numerical features in the test set

# Fitting SVM
install.packages('e1071')  # Installing the e1071 package
library(e1071)  # Loading the e1071 library
```

```
classifier = svm(formula = Purchased ~ ., data = training_set, type = 'C-classification', kernel = 'linear')  # Fitting the SVM model to the training set

print(classifier)  # Printing the SVM model

# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-3])  # Predicting the target variable for the test set using the SVM model

# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)  # Creating the confusion matrix

# Visualising the Training set results
library(ElemStatLearn)  # Loading the ElemStatLearn library
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3],
    main = 'SVM (Training set)',
    xlab = 'Age', ylab = 'Estimated Salary',
    xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

```r
# Visualising the Test set results
library(ElemStatLearn)  # Loading the ElemStatLearn library
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3], main = 'Decision Tree Classification (Test set)',
    xlab = 'Age', ylab = 'Estimated Salary',
    xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

Prac4A

Naive Bayes.

```r
# Importing the dataset
dataset <- read.csv("F:\\ Social_Network_Ads.csv")  # Reading the CSV file into the dataset variable
dataset <- dataset[3:5]  # Selecting columns 3 to 5 from the dataset
head(dataset)  # Displaying the first few rows of the dataset
```

```r
# Encoding the target feature as factor
dataset$Purchased <- factor(dataset$Purchased, levels = c(0, 1))  # Encoding the "Purchased" column as a factor with levels 0 and 1

# Splitting the dataset into the Training set and Test set
library(caTools)  # Loading the caTools library
set.seed(123)  # Setting the random seed for reproducibility
split <- sample.split(dataset$Purchased, SplitRatio = 0.75)  # Splitting the dataset into training and test sets
training_set <- subset(dataset, split == TRUE)  # Creating the training set using the split
test_set <- subset(dataset, split == FALSE)  # Creating the test set using the split

# Feature Scaling
training_set[-3] <- scale(training_set[-3])  # Scaling the numerical features in the training set
test_set[-3] <- scale(test_set[-3])  # Scaling the numerical features in the test set

# Fitting Naive Bayes to the Training set
library(e1071)  # Loading the e1071 library
classifier <- naiveBayes(x = training_set[-3], y = training_set$Purchased)  # Fitting the Naive Bayes model to the training set

# Predicting the Test set results
y_pred <- predict(classifier, newdata = test_set[-3])  # Predicting the target variable for the test set using the Naive Bayes model
```

```
# Making the Confusion Matrix
cm <- table(test_set[, 3], y_pred)  # Creating the confusion matrix
print(cm)  # Printing the confusion matrix
```

Prac4B
TextAnalysis.

```
# Read in the data
dataset_original <-
read.delim("F:\\GitHub\\Practical_BscIT_MscIT_Ninad\\MscIT\\Semester
2\\BigDataAnalytics\\Dataset\\Restaurant_Reviews.tsv", quote = "",
stringsAsFactors = FALSE)  # Reading the TSV file into the dataset_original
variable
head(dataset_original)  # Displaying the first few rows of the dataset

# Install and load required packages
install.packages('tm')  # Installing the 'tm' package for text mining
install.packages('SnowballC')  # Installing the 'SnowballC' package for
stemming
install.packages('randomForest')  # Installing the 'randomForest' package for
random forest classifier
library(tm)  # Loading the 'tm' package
library(SnowballC)  # Loading the 'SnowballC' package for stemming
library(caTools)  # Loading the 'caTools' package for data splitting
library(randomForest)  # Loading the 'randomForest' package for random
forest classifier
```

```
# Create a corpus
corpus <- VCorpus(VectorSource(dataset_original$Review))  # Creating a corpus from the 'Review' column of the dataset_original
corpus <- tm_map(corpus, content_transformer(tolower))  # Transforming the text to lowercase
corpus <- tm_map(corpus, removeNumbers)  # Removing numbers from the text
corpus <- tm_map(corpus, removePunctuation)  # Removing punctuation from the text
corpus <- tm_map(corpus, removeWords, stopwords())  # Removing common stopwords from the text
corpus <- tm_map(corpus, stemDocument)  # Stemming the words in the text
corpus <- tm_map(corpus, stripWhitespace)  # Removing extra whitespace from the text

# Create a document term matrix
dtm <- DocumentTermMatrix(corpus)  # Creating a document-term matrix from the corpus
dtm <- removeSparseTerms(dtm, 0.999)  # Removing sparse terms from the matrix

# Convert the dtm to a data frame
dataset <- as.data.frame(as.matrix(dtm))  # Converting the document-term matrix to a data frame
dataset$Liked <- dataset_original$Liked  # Adding the 'Liked' column from the dataset_original to the dataset
dataset$Liked <- factor(dataset$Liked, levels = c(0,1))  # Converting the 'Liked' column to a factor with levels 0 and 1

# Split the data into training and test sets
```

set.seed(123) *# Setting the random seed for reproducibility*

split <- sample.split(dataset$Liked, SplitRatio = 0.8) *# Splitting the dataset into training and test sets*

training_set <- subset(dataset, split == TRUE) *# Creating the training set using the split*

test_set <- subset(dataset, split == FALSE) *# Creating the test set using the split*

 # Train a random forest classifier

classifier <- randomForest(x = training_set[-692], y = training_set$Liked, ntree = 10) *# Training the random forest classifier using the training set*

 # Make predictions on the test set and create a confusion matrix

y_pred <- predict(classifier, newdata = test_set[-692]) *# Predicting the 'Liked' column for the test set using the random forest classifier*

cm <- table(test_set[,692], y_pred) *# Creating the confusion matrix*

print(cm) *# Printing the confusion matrix*

Prac5

Comparative Study of various machine learning models (Newly added)

 # Install required packages

install.packages('rpart') *# Installing the 'rpart' package for decision trees*

install.packages('rpart.plot') *# Installing the 'rpart.plot' package for plotting decision trees*

install.packages('gmodels') *# Installing the 'gmodels' package for calculating accuracy*

```r
install.packages('e1071')  # Installing the 'e1071' package for support vector machines

# Load required libraries
library(rpart)  # Loading the 'rpart' package
library(rpart.plot)  # Loading the 'rpart.plot' package
library(gmodels)  # Loading the 'gmodels' package for calculating accuracy
library(e1071)  # Loading the 'e1071' package for support vector machines

# Load iris dataset
data(iris)  # Loading the iris dataset
summary(iris)  # Summarizing the dataset

# Normalize the continuous variables before performing any analysis on the dataset
temp <- as.data.frame(scale(iris[, 1:4]))  # Scaling the continuous variables
temp$Species <- iris$Species  # Adding the 'Species' column to the scaled dataset
summary(temp)  # Summarizing the scaled dataset

# Split the dataset into the Training set and Test set
install.packages('caTools')  # Installing the 'caTools' package for data splitting
library(caTools)  # Loading the 'caTools' package
set.seed(123)  # Setting the random seed for reproducibility
split <- sample.split(temp$Species, SplitRatio = 0.75)  # Splitting the dataset into training and test sets
train <- subset(temp, split == TRUE)  # Creating the training set using the split
```

```
test <- subset(temp, split == FALSE)  # Creating the test set using the split

nrow(train)  # Printing the number of rows in the training set

nrow(test)  # Printing the number of rows in the test set


# 1. Decision Trees

dt_classifier <- rpart(formula = Species ~ ., data = train)  # Building the decision tree classifier using the training set


# Predict the Test set results for Decision Trees

dt_y_pred <- predict(dt_classifier, newdata = test, type = 'class')  # Predicting the 'Species' column for the test set using the decision tree classifier

print(dt_y_pred)  # Printing the predicted values


# Make the Confusion Matrix for Decision Tree

cm <- table(test$Species, dt_y_pred)  # Creating the confusion matrix for the decision tree classifier

print(cm)  # Printing the confusion matrix


# Calculate the accuracy of DT model

DTaccu <- ((12+9+11)/nrow(test))*100  # Calculating the accuracy of the decision tree model

DTaccu  # Printing the accuracy percentage


# 2. k-Nearest Neighbours

install.packages('class')  # Installing the 'class' package for k-nearest neighbours

library(class)  # Loading the 'class' package
```

```
cl <- train$Species  # Extracting the 'Species' column from the training set as the class variable

set.seed(1234)  # Setting the random seed for reproducibility

knn_y_pred <- knn(train[, 1:4], test[, 1:4], cl, k = 5)  # Predicting the 'Species' column for the test set using k-nearest neighbours

# Make the Confusion Matrix for k-Nearest Neighbours

cm <- table(test$Species, knn_y_pred)  # Creating the confusion matrix for k-nearest neighbours

print(cm)  # Printing the confusion matrix


# Calculate the accuracy of KNN model

KNNaccu <- ((12+11+11)/nrow(test))*100  # Calculating the accuracy of the k-nearest neighbours model

KNNaccu  # Printing the accuracy percentage


# 3. Support Vector Machine(SVM)

svmclassifier <- svm(Species ~ ., data = train)  # Building the support vector machine classifier using the training set

svm_y_pred <- predict(svmclassifier, newdata = test)  # Predicting the 'Species' column for the test set using support vector machine


cm <- table(test$Species, svm_y_pred)  # Creating the confusion matrix for support vector machine

print(cm)  # Printing the confusion matrix


# Calculate the accuracy of SVM model

SVMaccu <- ((12+11+11)/nrow(test))*100  # Calculating the accuracy of the support vector machine model

SVMaccu  # Printing the accuracy percentage
```

```
# Comparison of the accuracy of different models on testing dataset

which(dt_y_pred != knn_y_pred)  # Comparing the predictions of decision tree and k-nearest neighbours

which(dt_y_pred != svm_y_pred)  # Comparing the predictions of decision tree and support vector machine


# Compare SVM vs kNN

which(svm_y_pred != knn_y_pred)  # Comparing the predictions of support vector machine and k-nearest neighbours


# Create a dataframe of accuracy percentages for each model

models <- data.frame(Technique = c("Decision Tree", "KNN", "SVM"),
                     Accuracy_Percentage = c(DTaccu, KNNaccu, SVMaccu))

models  # Printing the dataframe


print("Hence KNN and SVM are better than decision tree")  # Printing the conclusion
```