1.#Basic R-program

print("Hello,R!")

x=c(1,2,3,4,5)

y=x\*2

print(y)

#Graphics

plot(x,y,main="Basic plot",xlab="X Axis",ylab="Y Axis")

#Indexing Data

print("Second element of X: ")

print(x[2])

print("Element of x greater than 2: ")

print(x[x>2])

#Loading the data from dataset

data(iris)

print("First 5 rows of data: ")

print(head(iris,5))

#Additional Graphic and Numerical summaries

#histgram

hist(iris$Sepal.Length,main="Histgram",xlab="Sepal length",ylab="sepal sepecies",col="green")

#Boxplot

boxplot(iris$Sepal.Length~iris$Species)

print(summary(iris$Sepal.Length))

plot(iris$Sepal.Length,iris$Sepal.Width,main="Scatter plot of sepal dimension",xlab="Sepal length",ylab="Scatter width",col=iris$Species)

aggregate(iris$Sepal.Length,by=list(species=iris$Species),FUN=mean)

2. library(ISLR)

library(ggplot2)

library(GGally)

library(car)

data(Auto)

IM\_Model=lm(mpg~horsepower,data=Auto)

summary(IM\_Model)

plot(Auto$horsepower,Auto$mpg,main="Mpg vs Horsepower",Xlab="Horsepower",ylab="Mpg",pch=19,col="blue")

abline(IM\_Model,col="red",lwd=2)

par(mfrw=c(2,2))

plot(IM\_Model)

pairs(Auto)

3.library(ISLR)

library(ggplot2)

library(GGally)

library(car)

data(Auto)

head(Auto)

#ggpairs(Auto [,-9])

#cor\_martix=cor(Auto,[,9])

#print(cor\_martix)

Im\_fit = lm(mpg ~ - name ,data=Auto)

summary(Im\_fit)

par(nfrow = c(2,2))

plot(Im\_fit)

Im\_fit\_interaction =lm(mpg ~ - name + cylinders: displacement + horsepower: weight + acceleration :year,data=Auto)

summary(Im\_fit\_interaction)

Auto$log\_horsepower=lof(Auto$horsepower)

4. # Load necessary libraries

library(mlbench)

library(class)

library(ggplot2)

# Load the dataset

data(BreastCancer)

breast\_cancer\_data <- BreastCancer

# Prepare the data

x <- breast\_cancer\_data[, -1] # Exclude the first column (ID)

y <- breast\_cancer\_data[, 1] # Target variable

# Normalize the numeric columns

numeric\_cols <- sapply(x, is.numeric)

normalized\_x <- scale(x[, numeric\_cols])

# Split the data into training and testing sets

set.seed(123)

train\_indices <- sample(1:nrow(normalized\_x), 0.8 \* nrow(normalized\_x))

train\_data <- normalized\_x[train\_indices, ]

train\_labels <- y[train\_indices]

test\_data <- normalized\_x[-train\_indices, ]

test\_labels <- y[-train\_indices]

# Define the range of k values

k\_values <- 1:450

# Initialize vectors to store accuracy and confusion matrices

accuracy\_values <- numeric(length(k\_values))

confusion\_matrices <- list()

# Perform KNN for each k value

for (i in seq\_along(k\_values)) {

k <- k\_values[i]

# Make predictions

knn\_predictions <- knn(train\_data, test\_data, train\_labels, k = k)

# Calculate accuracy

accuracy <- sum(knn\_predictions == test\_labels) / length(test\_labels)

accuracy\_values[i] <- accuracy

# Create confusion matrix

confusion\_matrix <- table(Actual = test\_labels, Predicted = knn\_predictions)

confusion\_matrices[[i]] <- confusion\_matrix

}

# Print the confusion matrices for each k

for (i in seq\_along(k\_values)) {

cat("Confusion Matrix for k =", k\_values[i], ":\n")

print(confusion\_matrices[[i]])

cat("\n")

}

# Plot accuracy vs. k values

accuracy\_df <- data.frame(K = k\_values, Accuracy = accuracy\_values)

ggplot(accuracy\_df, aes(x = K, y = Accuracy)) +

geom\_line() +

geom\_point() +

labs(title = "Accuracy vs. K Value",

x = "Number of Neighbors (k)",

y = "Accuracy")

# Determine the optimal k value

max\_accuracy\_index <- which.max(accuracy\_values)

optimal\_k <- k\_values[max\_accuracy\_index]

# Print the accuracy for each k and the optimal k value

for (i in seq\_along(k\_values)) {

cat("Accuracy for k =", k\_values[i], ":", accuracy\_values[i], "\n")

}

cat("\nBased on the results:\n")

cat("The highest accuracy is achieved with k =", optimal\_k, "\n")

5.

library(quantmod)

library(MASS)# For LDA and QDA

library(e1071)# for Naive

library(caret)

library(ggplot2)

library(pROC)

symbols = c("AAPL","MSFT")

start\_date = "2020-01-01"

end\_date = "2020-10-30"

getSymbols(symbols,from = start\_date ,to = end\_date)

stock\_date = merge(Ad(AAPL),Ad(MSFT))

colnames(stock\_date) = symbols

#cal

stock\_returns = diff(log(stock\_date))

return\_threshold =0.01

stock\_labels = ifelse(apply(abs(stock\_returns),1,max) >= return\_threshold , "High" ,"Low")

stock\_df = data.frame(stock\_returns,stock\_labels)

set.seed(123)#split data into training and testing sets

train\_indices = sample(1:nrow(stock\_df),0.8\*nrow(stock\_df))

train\_data = stock\_df[train\_indices,]

test\_data =stock\_df[-train\_indices,]

any\_na = apply(test\_data , 2,function(column) any(is.na(column)))

print(any\_na)

test\_data = test\_data[complete.cases(test\_data),]

#Linear

lda\_model = lda(stock\_labels ~ ., data = train\_data)

lda\_predictions = predict(lda\_model , newdata = test\_data)$class

#Quadratic

qda\_model = qda(stock\_labels ~ ., data = train\_data)

qda\_predictions = predict(qda\_model ,newdata = test\_data)$class

#Navie

nb\_model = naiveBayes(stock\_labels ~ ., data=train\_data)

nb\_predictions = predict(nb\_model , newdata = test\_data)

#cal accurate

lda\_accuracy = mean(lda\_predictions == test\_data$stock\_labels)

qda\_accuracy = mean(qda\_predictions == test\_data$stock\_labels)

nb\_accuracy = mean(nb\_predictions == test\_data$stock\_labels)

#print

cat("LDA Accuracy:", lda\_accuracy,"\n")

cat("QDA Accuracy:", qda\_accuracy,"\n")

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

#cal

test\_data$stock\_labels = as.factor(test\_data$stock\_labels)

lda\_cm = confusionMatrix(lda\_predictions,test\_data$stock\_labels)

qda\_cm = confusionMatrix(qda\_predictions,test\_data$stock\_labels)

nb\_cm = confusionMatrix(nb\_predictions,test\_data$stock\_labels)

print(lda\_cm)

print(qda\_cm)

print(nb\_cm)

accuracy\_df = data.frame(Classifier = c("LDA" , "QDA" , "Navie Bayes"), Accuracy = c(lda\_accuracy,qda\_accuracy,nb\_accuracy))

accuracy\_plot = ggplot(accuracy\_df , aes(x=Classifier , y=Accuracy ,fill = Classifier)) + geom\_bar(stat = "identity",position = "dodge")+labs(y ="Acurracy",title = "Classifier")+theme\_minimal()+theme(legend.position = "bottom")

print(accuracy\_plot)

roc\_lda = roc(test\_data$stock\_labels,as.numeric(lda\_predictions == "High"))

roc\_qda = roc(test\_data$stock\_labels,as.numeric(qda\_predictions == "High"))

roc\_nb = roc(test\_data$stock\_labels,as.numeric(nb\_predictions == "High"))

min\_length = min(length(roc\_lda$sensitivities),length(roc\_qda$sensitivities),length(roc\_nb$sensitivities))

roc\_data = data.frame(

FPR =c(roc\_lda$specificities[1:min\_length],roc\_qda$specificities[1:min\_length],roc\_nb$specificities[1:min\_length]),TPR =c(roc\_lda$sensitivities[1:min\_length],roc\_qda$sensitivities[1:min\_length],roc\_nb$sensitivities[1:min\_length]), Classifier = rep(c("LDA","QDA","Naive Bayes"),each = min\_length)

)

roc\_combined\_plot = ggplot(roc\_data ,aes(x = FPR,y = TPR,color = Classifier , linetype = Classifier)) + geom\_line(linewidth =1) + labs(x = "False Positive Rate(1- Specificity",y = "True Positive Rate(Sensitivity)",title= "ROC Curves - Classifier Comparsion") + scale\_color\_manual(values = c("blue","green","red"))+ scale\_linetype\_manual(values = c("solid","dashed","dotted"))+theme\_minimal() + theme(legend.position = "bottom")

print(roc\_combined\_plot)

6A  
# Load necessary libraries

library(rpart)

library(rpart.plot)

library(partykit)

library(caret)

library(pROC)

library(ROCR)

# Load the dataset

dataset <- read.csv("C:\\Users\\manoj\\OneDrive\\Documents\\mushrooms.csv")

dataset$class <- as.factor(dataset$class)

# Split the dataset into training and testing sets

set.seed(123) # Reproducibility

sample\_indices <- sample(nrow(dataset), 0.8 \* nrow(dataset))

train\_data <- dataset[sample\_indices, ]

test\_data <- dataset[-sample\_indices, ]

# Train the decision tree model

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

print(tree\_model)

# Make predictions on the test set

predictions <- predict(tree\_model, test\_data, type = "class")

# Calculate accuracy

accuracy <- sum(predictions == test\_data$class) / nrow(test\_data)

cat("Accuracy: ", accuracy, "\n")

# Convert the rpart object to a party object for visualization

party\_tree <- as.party(tree\_model)

# Visualize the decision tree

plot(party\_tree)

# Display the first few predictions

head(predictions)

# Create a confusion matrix

actual\_labels <- test\_data$class

confusion\_matrix <- table(actual\_labels, predictions)

print(confusion\_matrix)

# Calculate precision and recall

precision <- diag(confusion\_matrix) / rowSums(confusion\_matrix)

recall <- diag(confusion\_matrix) / colSums(confusion\_matrix)

# Calculate F1 Score

f1\_score <- 2 \* (precision \* recall) / (precision + recall)

# Create a result data frame

result\_df <- data.frame(Precision = precision, Recall = recall, F1\_score = f1\_score)

# Print the result data frame

print(result\_df)

# Create binary predictions for ROC analysis

binary\_predictions <- ifelse(predictions == "p", 1, 0)

# Create ROC object and calculate AUC

roc\_obj <- roc(as.numeric(test\_data$class == "p"), binary\_predictions)

plot(roc\_obj, print.auc = TRUE)

# Calculate AUC score

auc\_score <- auc(roc\_obj)

cat("AUC-ROC:", auc\_score, "\n")

# Summarize the original tree model

summary(tree\_model)

# Prune the tree

pruned\_tree <- prune(tree\_model, cp = 0.01)

summary(pruned\_tree)

rpart.plot(pruned\_tree)

pruned\_predictions<-predict(pruned\_tree,newdata=test\_data,type="class")

pruned\_confusion\_matrix<-confusionMatrix(pruned\_predictions,test\_data$class)

print(pruned\_confusion\_matrix)

pruned\_accuracy<-sum(pruned\_predictions==test\_data$class)/length(test\_data$class)

cat("Pruned Model Accuracy:", pruned\_accuracy,"\n")

pruned\_confusion\_matrix<-confusionMatrix(pruned\_predictions,test\_data$class)

cat("Prunred Model Accuracy:", pruned\_accuracy,"\n")

pruned\_confusion\_matrix<-confusionMatrix(pruned\_predictions,test\_data$class)

pruned\_prediction<-pruned\_confusion\_matrix$byClass["Positive predictive value"]

pruned\_recall<-pruned\_confusion\_matrix$byClass["Sensitivity"]

pruned\_f1\_score<-pruned\_confusion\_matrix["F1"]

cat("Pruned Model Precision:",pruned\_prediction,"\n")

cat("")

6B  
# Load necessary packages

library(ISLR2)

library(pROC)

library(ggplot2)

library(e1071)

library(plotmo)

library(plotly)

library(PRROC) # For Precision-Recall curve

library(caret) # For confusion matrix and metrics

# Display column names of the Khan dataset

names(Khan)

# Check dimensions of training and testing sets

dim(Khan$xtrain)

dim(Khan$xtest)

# Check the length of training and testing labels

length(Khan$ytrain)

length(Khan$ytest)

# Display class distribution in training and testing labels

table(Khan$ytrain)

table(Khan$ytest)

# Create a dataframe 'dat' with training features and labels

dat <- data.frame(

x = Khan$xtrain,

y = as.factor(Khan$ytrain)

)

# Train an SVM model with linear kernel

out <- svm(y ~ ., data = dat, kernel = "linear", cost = 10)

# Display summary of the trained SVM model

summary(out)

# Compare the fitted values with actual labels in the training data

table(out$fitted, dat$y)

# Create a dataframe 'dat.te' with testing features and labels

dat.te <- data.frame(x = Khan$xtest, y = as.factor(Khan$ytest))

# Predict using the trained SVM model on the testing data

pred.te <- predict(out, newdata = dat.te)

# Compare the predicted values with actual labels in the testing data

table(pred.te, dat.te$y)

# Confusion matrix for training data

train\_pred <- predict(out, newdata = dat)

train\_conf\_matrix <- table(train\_pred, dat$y)

# Convert the confusion matrix to a dataframe

train\_conf\_df <- as.data.frame(as.table(train\_conf\_matrix))

# Rename the columns for better readability

colnames(train\_conf\_df) <- c("Predicted", "Actual", "Frequency")

# Create the heatmap with annotations

ggplot(train\_conf\_df, aes(x = Predicted, y = Actual, fill = Frequency)) +

geom\_tile() +

geom\_text(aes(label = Frequency), color = "black", size = 4) + # Add text annotations

scale\_fill\_gradient(low = "white", high = "blue") +

labs(x = "Predicted Class", y = "Actual Class", fill = "Frequency") +

theme\_minimal()

# Convert factor levels to numeric (1 or 2)

numeric\_pred <- as.numeric(train\_pred)

# Calculate multiclass ROC curve and AUC for training data

train\_roc <- multiclass.roc(dat$y, numeric\_pred)

# Plot the multiclass ROC curves for each class

plot(train\_roc$rocs[[1]], col = 1, print.auc = TRUE, auc.polygon = TRUE)

for (i in 2:length(train\_roc$rocs)) {

lines(train\_roc$rocs[[i]], col = i, print.auc = TRUE, auc.polygon = TRUE)

}

# Add a legend

legend("bottomright", legend = levels(dat$y), col = 1:length(train\_roc$rocs), lwd = 2)

# Calculate performance metrics for training data

train\_pred <- predict(out, newdata = dat)

train\_conf\_matrix <- table(train\_pred, dat$y)

# Calculate accuracy for training data

train\_accuracy <- sum(diag(train\_conf\_matrix)) / sum(train\_conf\_matrix)

# Calculate precision, recall, F1-score for each class in training data

class\_names <- levels(dat$y)

train\_metrics <- data.frame(Class = character(), Precision = numeric(), Recall = numeric(), F1\_Score = numeric(), stringsAsFactors = FALSE)

for (class\_name in class\_names) {

tp <- train\_conf\_matrix[class\_name, class\_name]

fp <- sum(train\_conf\_matrix[class\_name, ]) - tp

fn <- sum(train\_conf\_matrix[, class\_name]) - tp

precision <- tp / (tp + fp)

recall <- tp / (tp + fn)

f1\_score <- 2 \* (precision \* recall) / (precision + recall)

train\_metrics <- rbind(train\_metrics, data.frame(Class = class\_name, Precision = precision, Recall = recall, F1\_Score = f1\_score))

}

# Calculate overall classification accuracy for training data

train\_overall\_accuracy <- mean(train\_metrics$F1\_Score)

# Calculate performance metrics for testing data

test\_conf\_matrix <- table(pred.te, dat.te$y)

# Calculate accuracy for testing data

test\_accuracy <- sum(diag(test\_conf\_matrix)) / sum(test\_conf\_matrix)

# Calculate precision, recall, and F1-score for each class in testing data

test\_metrics <- data.frame(Class = character(), Precision = numeric(), Recall = numeric(), F1\_Score = numeric(), stringsAsFactors = FALSE)

for (class\_name in class\_names) {

tp <- test\_conf\_matrix[class\_name, class\_name]

fp <- sum(test\_conf\_matrix[class\_name, ]) - tp

fn <- sum(test\_conf\_matrix[, class\_name]) - tp

precision <- tp / (tp + fp)

recall <- tp / (tp + fn)

f1\_score <- 2 \* (precision \* recall) / (precision + recall)

test\_metrics <- rbind(test\_metrics, data.frame(Class = class\_name, Precision = precision, Recall = recall, F1\_Score = f1\_score))

}

# Calculate overall classification accuracy for testing data

test\_overall\_accuracy <- mean(test\_metrics$F1\_Score)

# Print performance metrics for both training and testing data

cat("Performance Metrics for Training Data:\n")

cat("Overall Accuracy:", train\_accuracy, "\n")

print(train\_metrics)

cat("Overall Classification Accuracy (F1-Score):", train\_overall\_accuracy, "\n\n")

cat("Performance Metrics for Testing Data:\n")

cat("Overall Accuracy:", test\_accuracy, "\n")

print(test\_metrics)

cat("Overall Classification Accuracy (F1-Score):", test\_overall\_accuracy, "\n")

# Create a dataframe for training metrics

train\_metrics\_df <- data.frame(

Class = train\_metrics$Class,

Precision = train\_metrics$Precision,

Recall = train\_metrics$Recall,

F1\_Score = train\_metrics$F1\_Score

)

# Create a bar plot for training metrics

ggplot(train\_metrics\_df, aes(x = Class, y = F1\_Score, fill = Class)) +

geom\_bar(stat = "identity", position = "dodge") +

labs(

title = "Performance Metrics for Training Data",

y = "F1-Score",

x = "Class"

) +

theme\_minimal() +

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

# Create a dataframe for testing metrics

test\_metrics\_df <- data.frame(

Class = test\_metrics$Class,

Precision = test\_metrics$Precision,

Recall = test\_metrics$Recall,

F1\_Score = test\_metrics$F1\_Score

)

# Create a bar plot for testing metrics

ggplot(test\_metrics\_df, aes(x = Class, y = F1\_Score, fill = Class)) +

geom\_bar(stat = "identity", position = "dodge") +

labs(

title = "Performance Metrics for Testing Data",

y = "F1-Score",

x = "Class"

) +

theme\_minimal() +

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

# Precision-Recall Curve for Testing Data

test\_pred\_proba <- attr(predict(out, newdata = dat.te, decision.values = TRUE), "decision.values")

pr <- pr.curve(scores.class0 = test\_pred\_proba[dat.te$y == 1],

scores.class1 = test\_pred\_proba[dat.te$y == 2], curve = TRUE)

# Plot the Precision-Recall Curve

plot(pr, main = "Precision-Recall Curve for Test Data", auc.main = TRUE)

# Log-Loss for Training and Testing Data

train\_pred\_proba <- attr(predict(out, newdata = dat, decision.values = TRUE), "decision.values")

train\_log\_loss <- -mean(log(ifelse(train\_pred\_proba > 0, 0.999, 0.001)))

test\_log\_loss <- -mean(log(ifelse(test\_pred\_proba > 0, 0.999, 0.001)))

cat("Log-Loss for Training Data:", train\_log\_loss, "\n")

cat("Log-Loss for Testing Data:", test\_log\_loss, "\n")

# Custom function to calculate Matthews Correlation Coefficient (MCC)

calculate\_mcc <- function(conf\_matrix) {

TP <- conf\_matrix[1, 1]

TN <- conf\_matrix[2, 2]

FP <- conf\_matrix[1, 2]

FN <- conf\_matrix[2, 1]

numerator <- (TP \* TN) - (FP \* FN)

denominator <- sqrt((TP + FP) \* (TP + FN) \* (TN + FP) \* (TN + FN))

if (denominator == 0) {

return(0) # MCC is 0 if there's a division by zero

} else {

return(numerator / denominator)

}

}

# Confusion matrix for training data

train\_conf\_matrix <- table(train\_pred, dat$y)

# Confusion matrix for testing data

test\_conf\_matrix <- table(pred.te, dat.te$y)

# Calculate MCC for training and testing data

mcc\_train <- calculate\_mcc(train\_conf\_matrix)

mcc\_test <- calculate\_mcc(test\_conf\_matrix)

cat("Matthews Correlation Coefficient for Training Data:", mcc\_train, "\n")

cat("Matthews Correlation Coefficient for Testing Data:", mcc\_test, "\n")

# Calculate Specificity for Training and Testing Data

calculate\_specificity <- function(conf\_matrix) {

specificity <- numeric()

for (class\_name in class\_names) {

tn <- sum(conf\_matrix) - (sum(conf\_matrix[class\_name, ]) + sum(conf\_matrix[, class\_name]) - conf\_matrix[class\_name, class\_name])

fp <- sum(conf\_matrix[class\_name, ]) - conf\_matrix[class\_name, class\_name]

specificity <- c(specificity, tn / (tn + fp))

}

return(specificity)

}

train\_specificity <- calculate\_specificity(train\_conf\_matrix)

test\_specificity <- calculate\_specificity(test\_conf\_matrix)

train\_metrics$Specificity <- train\_specificity

test\_metrics$Specificity <- test\_specificity

cat("Updated Performance Metrics for Training Data with Specificity:\n")

print(train\_metrics)

cat("Updated Performance Metrics for Testing Data with Specificity:\n")

print(test\_metrics)

# Calculate Cohen's Kappa for Training and Testing Data

kappa\_train <- confusionMatrix(train\_pred, dat$y)$overall["Kappa"]

kappa\_test <- confusionMatrix(pred.te, dat.te$y)$overall["Kappa"]

cat("Cohen's Kappa for Training Data:", kappa\_train, "\n")

cat("Cohen's Kappa for Testing Data:", kappa\_test, "\n")

# Calculate Balanced Accuracy for Training and Testing Data

calculate\_balanced\_accuracy <- function(conf\_matrix) {

sensitivity <- diag(conf\_matrix) / rowSums(conf\_matrix)

specificity <- (colSums(conf\_matrix) - diag(conf\_matrix)) / (sum(conf\_matrix) - rowSums(conf\_matrix))

balanced\_accuracy <- mean((sensitivity + specificity) / 2)

return(balanced\_accuracy)

}

train\_balanced\_accuracy <- calculate\_balanced\_accuracy(train\_conf\_matrix)

test\_balanced\_accuracy <- calculate\_balanced\_accuracy(test\_conf\_matrix)

cat("Balanced Accuracy for Training Data:", train\_balanced\_accuracy, "\n")

cat("Balanced Accuracy for Testing Data:", test\_balanced\_accuracy, "\n")

# Create a dataframe for metrics

metrics\_df <- data.frame(

Metric = c("Accuracy", "Precision", "Recall", "F1 Score", "Specificity", "MCC", "Cohen's Kappa", "Balanced Accuracy"),

Training = c(train\_accuracy, colMeans(train\_metrics[,-1]), mcc\_train, kappa\_train, train\_balanced\_accuracy),

Testing = c(test\_accuracy, colMeans(test\_metrics[,-1]), mcc\_test, kappa\_test, test\_balanced\_accuracy)

)

# Create a bar plot for the metrics

metrics\_long <- reshape2::melt(metrics\_df, id.vars = "Metric")

ggplot(metrics\_long, aes(x = Metric, y = value, fill = variable)) +

geom\_bar(stat = "identity", position = position\_dodge()) +

labs(title = "Performance Metrics for Training and Testing Data",

x = "Metrics", y = "Value") +

scale\_fill\_manual(values = c("darkgrey", "yellow"), labels = c("Training", "Testing")) +

theme\_minimal() +

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

7  
library(ISLR2)

library(pROC)

library(ggplot2)

library(e1071)

library(plotmo)

library(plotly)

library(PRROC)

library(caret)

names(Khan)

dim(Khan$xtrain)

dim(Khan$xtest)

length(Khan$ytrain)

length(Khan$ytest)

table(Khan$ytrain)

table(Khan$ytest)

dat <- data.frame(

x = Khan$xtrain,

y = as.factor(Khan$ytrain)

)

out <- svm(y ~ ., data = dat, kernel = "linear", cost = 10)

summary(out)

table(out$fitted, dat$y)

dat.te <- data.frame(x = Khan$xtest, y = as.factor(Khan$ytest))

pred.te <- predict(out, newdata = dat.te)

table(pred.te, dat.te$y)

train\_pred <- predict(out, newdata = dat)

train\_conf\_matrix <- table(train\_pred, dat$y)

train\_conf\_df <- as.data.frame(as.table(train\_conf\_matrix))

colnames(train\_conf\_df) <- c("Predicted", "Actual", "Frequency")

# Corrected ggplot line

ggplot(train\_conf\_df, aes(x = Predicted, y = Actual, fill = Frequency)) +

geom\_tile() +

geom\_text(aes(label = Frequency), color = "black", size = 4) +

scale\_fill\_gradient(low = "white", high = "blue") +

labs(x = "Predicted Class", y = "Actual Class", fill = "Frequency") +

theme\_minimal()

numeric\_pred <- as.numeric(train\_pred)

train\_roc <- multiclass.roc(dat$y, numeric\_pred)

plot(train\_roc$rocs[[1]], col = 1, print.auc = TRUE, auc.polygon = TRUE)

for (i in 2:length(train\_roc$rocs)) {

lines(train\_roc$rocs[[i]], col = i, print.auc = TRUE, auc.polygon = TRUE)

}

legend("bottomright", legend = levels(dat$y), col = 1:length(train\_roc$rocs), lwd = 2)

# Corrected line for accuracy

train\_accuracy <- sum(diag(train\_conf\_matrix)) / sum(train\_conf\_matrix)

class\_names <- levels(dat$y)

train\_metrics <- data.frame(Class = character(), Precision = numeric(), Recall = numeric(), F1\_Score = numeric(), stringsAsFactors = FALSE)

for (class\_name in class\_names) {

tp <- train\_conf\_matrix[class\_name, class\_name]

fp <- sum(train\_conf\_matrix[class\_name, ]) - tp

fn <- sum(train\_conf\_matrix[, class\_name]) - tp

precision <- tp / (tp + fp)

recall <- tp / (tp + fn) # Fixed the assignment operator

f1\_score <- 2 \* (precision \* recall) / (precision + recall)

# Corrected rbind line

train\_metrics <- rbind(train\_metrics, data.frame(Class = class\_name, Precision = precision, Recall = recall, F1\_Score = f1\_score))

}

train\_overall\_accuracy<-mean(train\_metrics$F1\_Score)

test\_conf\_matrix<-table(pred.te,dat.te$y)

test\_accuracy<-sum(diag(test\_conf\_matrix)) / sum(test\_conf\_matrix)

test\_metrics <- data.frame(Class = character(), Precision = numeric(), Recall = numeric(), F1\_Score = numeric(), stringsAsFactors = FALSE)

for (class\_name in class\_names) {

tp <- test\_conf\_matrix[class\_name, class\_name]

fp <- sum(test\_conf\_matrix[class\_name, ]) - tp

fn <- sum(test\_conf\_matrix[, class\_name]) - tp

precision <- tp / (tp + fp)

recall <- tp / (tp + fn) # Fixed the assignment operator

f1\_score <- 2 \* (precision \* recall) / (precision + recall)

# Corrected rbind line

test\_metrics <- rbind(test\_metrics, data.frame(Class = class\_name, Precision = precision, Recall = recall, F1\_Score = f1\_score))

}

test\_overall\_accuracy<-mean(test\_metrics$F1\_Score)

cat("Performance Metrics for Training Data:\n")

cat("Overall Accuracy:",train\_accuracy,"\n")

print(train\_metrics)

cat("Overall Classification Accuracy (F1-Score):",train\_overall\_accuracy,"\n\n")

cat("Performance Metrics for Testing Data:\n")

cat("Overall Accuracy:",test\_accuracy,"\n")

print(test\_metrics)

cat("Overall Classification Accuracy (F1-Score):",test\_overall\_accuracy,"\n\n")

train\_metrics\_df<-data.frame(

Class=train\_metrics$Class,

Precision=train\_metrics$Precision,

Recall=train\_metrics$Recall,

F1\_Score=train\_metrics$F1\_Score

)

ggplot(train\_metrics\_df, aes(x = Class, y = F1\_Score, fill = Class)) +

geom\_bar(stat = "identity", position = "dodge") +

labs(

title = "Performance Metrics for Training Data",

y = "F1-Score",

x = "Class"

) +

theme\_minimal() +

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

test\_metrics\_df<-data.frame(

Class=test\_metrics$Class,

Precision=test\_metrics$Precision,

Recall=test\_metrics$Recall,

F1\_Score=test\_metrics$F1\_Score

)

ggplot(test\_metrics\_df, aes(x = Class, y = F1\_Score, fill = Class)) +

geom\_bar(stat = "identity", position = "dodge") +

labs(

title = "Performance Metrics for Testing Data",

y = "F1-Score",

x = "Class"

) +

theme\_minimal() +

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

test\_pred\_proba<-attr(predict(out,newdata=dat.te,decision.values=TRUE),

"decision.values")

pr<-pr.curve(scores.class0 =test\_pred\_proba[dat.te$y==1],

scores.class1=test\_pred\_proba[dat.te$y==2],curve=TRUE)

train\_pred\_proba<-attr(predict(out,newdata=dat,decision.values = TRUE),

"decision.values")

train\_log\_loss<- -mean(log(ifelse(train\_pred\_proba>0,0.999,0.001)))

test\_log\_loss<- -mean(log(ifelse(test\_pred\_proba>0,0.999,0.001)))

cat("Log-Loss for Training Data:",train\_log\_loss,"\n")

cat("Log-Loss for Testing Data:", test\_log\_loss,"\n")

calculate\_mcc<-function(conf\_matrix){

TP<-conf\_matrix[1,1]

TN<-conf\_matrix[2,2]

FP<-conf\_matrix[1,2]

FN<-conf\_matrix[2,1]

numerator<-(TP\*TN)-(FP\*FN)

denominator<-sqrt((TP+FP)\*(TP+FN)\*(TN+FP)\*(TN+FN))

if (denominator==0){

return(0)

} else{

return(numerator/denominator)

}

}

#confusion matrix for training data

train\_conf\_matrix<-table(train\_pred,dat$y)

test\_conf\_matrix<-table(pred.te,dat.te$y)

mcc\_train<-calculate\_mcc(train\_conf\_matrix)

mcc\_test<-calculate\_mcc(test\_conf\_matrix)

cat("Matthews Correlation Coefficient for Training Data:", mcc\_train,"\n")

cat("Matthews Correlation Coefficient for Testing Data:", mcc\_test,"\n")

calculate\_specificity <- function(conf\_matrix) {

specificity <- numeric()

for (class\_name in class\_names) {

tn <- sum(conf\_matrix) - (sum(conf\_matrix[class\_name, ]) + sum(conf\_matrix[, class\_name]) - conf\_matrix[class\_name, class\_name])

fp <- sum(conf\_matrix[, class\_name])

specificity <- c(specificity, tn / (tn + fp))

}

return(specificity)

}

train\_specificity<-calculate\_specificity(train\_conf\_matrix)

test\_specificity<-calculate\_specificity(test\_conf\_matrix)

cat("Updated Performance Metrics fro Training Data with Spectificity:\n")

print(train\_metrics)

cat("Updated Performance Metrics fro Testing Data with Spectificity:\n")

print(test\_metrics)

8A  
library(caret)

library(ggplot2)

library(rpart)

library(rpart.plot)

data(mtcars)

my\_pca = prcomp(mtcars[, -which(names(mtcars) == "disp")], scale. =TRUE, center = TRUE)

train.data = data.frame(disp = mtcars$disp, my\_pca$x[, 1:4])

set.seed(123)

train\_index = sample(1:nrow(train.data), 0.8\*nrow(train.data))

train\_set=train.data[train\_index, ]

test\_set= train.data[-train\_index,]

rpart.model<-rpart(disp ~ ., data=train\_set,method="anova")

predictions<-predict(rpart.model,newdata=test\_set)

mse<-mean((test\_set$disp-predictions)^2)

ss\_residual<-sum((test\_set$disp-mean(test\_set$disp))^2)

r\_squared<-1-(ss\_residual/ss\_total)

print(paste("Mean Squared Error:",mse))

print(paste("R-squared:",r\_squared))

residuals<-test\_set$disp-predictions

residuals\_df<-data.frame(Fitted=predictions,Residuals=residuals)

ggplot(residuals\_df,aes(x=Fitted,y=Residuals))+

geom\_point()+

geom\_hline(yintercept = 0, linetype="dashed", color="red")+

labs(title="Residuals vs Fitted", x="Fitted Values", y="Residuals")+

theme\_minimal()

predicted\_vs\_actual<-data.frame(Actual=test\_set$disp,Predicted=predictions)

ggplot(predicted\_vs\_actual,aes(x=Actual,y=Predicted))+

geom\_point()+

geom\_abline(slope=1,intercept = 0, linetype="dashed", color="red")+

labs(title="Predicted vs Actual", x="Actual Disp", y="Predicted Disp")+

theme\_minimal()

cat("Enter values for the following variables (example values in parentheses):\n")

cat("1. mpg (e.g., 21.0): ")

mpg\_input = as.numeric(readLines(n = 1))

cat("2. cyl (e.g., 6): ")

cyl\_input = as.numeric(readLines(n = 1))

cat("3. disp (e.g., 160): ")

disp\_input = as.numeric(readLines(n = 1))

cat("4. hp (e.g., 100): ")

hp\_input = as.numeric(readLines(n = 1))

cat("5. drat (e.g., 4.0): ")

drat\_input = as.numeric(readLines(n = 1))

cat("6. wt (e.g., 2.5): ")

wt\_input = as.numeric(readLines(n = 1))

cat("7. qsec (e.g., 17.0): ")

qsec\_input = as.numeric(readLines(n = 1))

cat("8. vs (e.g., 0 for V/S, 1 for V): ")

vs\_input = as.numeric(readLines(n = 1))

cat("9. am (e.g., 0 for automatic, 1 for manual): ")

am\_input = as.numeric(readLines(n = 1))

cat("10. gear (e.g., 4): ")

gear\_input = as.numeric(readLines(n = 1))

cat("11. carb (e.g., 2): ")

carb\_input = as.numeric(readLines(n = 1))

new\_data <- data.frame(

mpg = mpg\_input,

cyl = cyl\_input,

disp = disp\_input,

hp = hp\_input,

drat = drat\_input,

wt = wt\_input,

qsec = qsec\_input,

vs = vs\_input,

am = am\_input,

gear = gear\_input,

carb = carb\_input

)

# Scale the new data

new\_data\_scaled <- scale(new\_data)

# Perform PCA (assuming my\_pca is already created)

new\_pca <- as.data.frame(predict(my\_pca, newdata = new\_data\_scaled))

# Predict displacement using the regression model (assuming rpart.model is defined)

predicted\_disp <- predict(rpart.model, newdata = new\_pca[, 1:4])

# Print the predicted displacement

print(paste("Predicted Displacement (disp):", predicted\_disp))

# Combine results into a data frame

combined\_results <- data.frame(

Actual = c(test\_set$disp, predicted\_disp),

Predicted = c(test\_set$disp, predicted\_disp), # Assuming you want to predict the same values

Source = c(rep("Test Set", length(test\_set$disp)), "User Input")

)

ggplot(combined\_results,aes(x=Actual, y=Predicted, color=Source))+

geom\_point()+

geom\_abline(slope=1,intercept = 0, color="black",linetype="dashed")+

labs(title = "Cobined Predicted vs Actual", x="Actual Disp", y="Predicted Disp")+

scale\_color\_manual(values=c("blue","red"))+

theme\_minimal()

8B

# R program to implement bagging, boosting, stacking on IRIS dataset

library(kknn)

# Load necessary libraries

library(caret)

library(randomForest)

library(xgboost)

library(mlr)

library(ggplot2)

library(ROCR)

library(reshape2)

# 1. Load dataset and split into train and test

data(iris) # Example dataset

set.seed(123)

# Splitting the data into train (80%) and test (20%)

trainIndex <- createDataPartition(iris$Species, p = 0.8, list = FALSE)

trainData <- iris[trainIndex, ]

testData <- iris[-trainIndex, ]

# 2. Bagging Model (using randomForest)

set.seed(123)

bagging\_model <- randomForest(Species ~ ., data = trainData, ntree = 500)

bagging\_pred <- predict(bagging\_model, testData)

# 3. Boosting Model (using xgboost)

set.seed(123)

train\_matrix <- model.matrix(Species ~ ., data = trainData)[,-1]

test\_matrix <- model.matrix(Species ~ ., data = testData)[,-1]

train\_label <- as.numeric(trainData$Species) - 1 # XGBoost requires numeric labels

test\_label <- as.numeric(testData$Species) - 1

xgb\_train <- xgb.DMatrix(data = train\_matrix, label = train\_label)

xgb\_test <- xgb.DMatrix(data = test\_matrix, label = test\_label)

boosting\_model <- xgboost(data = xgb\_train, max.depth = 3, eta = 0.3, nrounds = 100, objective = "multi:softprob", num\_class = 3)

boosting\_pred <- predict(boosting\_model, xgb\_test)

boosting\_pred <- max.col(matrix(boosting\_pred, ncol=3)) - 1

# Convert numeric prediction back to factor levels

boosting\_pred <- factor(boosting\_pred, levels = c(0, 1, 2), labels = levels(testData$Species))

# 4. Stacking Model

# Define learners (using classif.kknn for KNN with probabilities)

lrn1 <- makeLearner("classif.randomForest", predict.type = "prob")

lrn2 <- makeLearner("classif.xgboost", predict.type = "prob")

lrn3 <- makeLearner("classif.kknn", predict.type = "prob") # kknn for probabilities

# Define the classification task

task <- makeClassifTask(data = trainData, target = "Species")

# Create the stacking model (with randomForest as meta-learner)

base\_learners <- list(lrn1, lrn2, lrn3)

stacking\_model <- makeStackedLearner(base.learners = base\_learners, super.learner = "classif.randomForest", predict.type = "prob")

# Train the stacking model

stacking\_trained <- train(stacking\_model, task)

# Predict on the test set

stacking\_pred <- predict(stacking\_trained, newdata = testData)$data$response

# 5. Performance metrics function for multiclass

performance\_metrics <- function(actual, predicted) {

# Confusion matrix

cm <- confusionMatrix(as.factor(predicted), as.factor(actual))

# Overall accuracy

accuracy <- cm$overall['Accuracy']

# Per-class metrics (Precision, Recall, F1)

precision <- cm$byClass[, 'Pos Pred Value'] # Precision for each class

recall <- cm$byClass[, 'Sensitivity'] # Recall for each class

f1 <- 2 \* (precision \* recall) / (precision + recall) # F1 for each class

# Averaging metrics across classes

avg\_precision <- mean(precision, na.rm = TRUE)

avg\_recall <- mean(recall, na.rm = TRUE)

avg\_f1 <- mean(f1, na.rm = TRUE)

list(

Accuracy = accuracy,

Precision = avg\_precision,

Recall = avg\_recall,

F1\_Score = avg\_f1

)

}

# 6. Calculate performance metrics for each model

bagging\_metrics <- performance\_metrics(testData$Species, bagging\_pred)

boosting\_metrics <- performance\_metrics(testData$Species, boosting\_pred)

stacking\_metrics <- performance\_metrics(testData$Species, stacking\_pred)

# Print Accuracy values

cat("Accuracy (Bagging): ", bagging\_metrics$Accuracy, "\n")

cat("Accuracy (Boosting): ", boosting\_metrics$Accuracy, "\n")

cat("Accuracy (Stacking): ", stacking\_metrics$Accuracy, "\n")

# Combine the metrics into a dataframe for comparison

metrics\_df <- data.frame(

Model = c("Bagging", "Boosting", "Stacking"),

Accuracy = c(bagging\_metrics$Accuracy, boosting\_metrics$Accuracy, stacking\_metrics$Accuracy),

Precision = c(bagging\_metrics$Precision, boosting\_metrics$Precision, stacking\_metrics$Precision),

Recall = c(bagging\_metrics$Recall, boosting\_metrics$Recall, stacking\_metrics$Recall),

F1\_Score = c(bagging\_metrics$F1\_Score, boosting\_metrics$F1\_Score, stacking\_metrics$F1\_Score)

)

# 7. Plotting the comparison of models

metrics\_melt <- melt(metrics\_df, id.vars = 'Model')

ggplot(metrics\_melt, aes(x = Model, y = value, fill = variable)) +

geom\_bar(stat = 'identity', position = 'dodge') +

labs(title = "Model Performance Comparison", y = "Metric Value", x = "Model") +

theme\_minimal()

9  
# R program to implement K-Means clustering on USArrests dataset

library(factoextra)

library(cluster)

#load data

df <- USArrests

#remove rows with missing values

df <- na.omit(df)

#scale each variable to have a mean of 0 and sd of 1

df <- scale(df)

#view first six rows of dataset

head(df)

#Find the Optimal Number of Clusters

#Number of Clusters vs. the Total Within Sum of Squares

fviz\_nbclust(df, kmeans, method = "wss")

#Number of Clusters vs. Gap Statistic

#calculate gap statistic based on number of clusters

gap\_stat <- clusGap(df,

FUN = kmeans,

nstart = 25,

K.max = 10,

B = 50)

#plot number of clusters vs. gap statistic

fviz\_gap\_stat(gap\_stat)

#Perform k-means clustering on the dataset using the optimal value for k of 4:

#make this example reproducible

set.seed(1)

#perform k-means clustering with k = 4 clusters

km <- kmeans(df, centers = 4, nstart = 25)

#view results

km

#plot results of final k-means model

fviz\_cluster(km, data = df)

#find means of each cluster

aggregate(USArrests, by=list(cluster=km$cluster), mean)

#add cluster assigment to original data

final\_data <- cbind(USArrests, cluster = km$cluster)

#view final data

head(final\_data)

10

# R program to implement Hierachial clustering on USArrests dataset

library(factoextra)

library(cluster)

library(ggplot2)

library(dplyr)

#load data

df <- USArrests

#remove rows with missing values

df <- na.omit(df)

#scale each variable to have a mean of 0 and sd of 1

df <- scale(df)

#define linkage methods

m <- c( "average", "single", "complete", "ward")

names(m) <- c( "average", "single", "complete", "ward")

#function to compute agglomerative coefficient

ac <- function(x) {

agnes(df, method = x)$ac

}

#calculate agglomerative coefficient for each clustering linkage method

sapply(m, ac)

#perform hierarchical clustering using Ward's minimum variance

clust <- agnes(df, method = "ward")

#produce dendrogram

pltree(clust, cex = 0.6, hang = -1, main = "Dendrogram")

#calculate gap statistic for each number of clusters (up to 10 clusters)

gap\_stat <- clusGap(df, FUN = hcut, nstart = 25, K.max = 10, B = 50)

#produce plot of clusters vs. gap statistic

fviz\_gap\_stat(gap\_stat)

#compute distance matrix

d <- dist(df, method = "euclidean")

#perform hierarchical clustering using Ward's method

final\_clust <- hclust(d, method = "ward.D2" )

#cut the dendrogram into 4 clusters

groups <- cutree(final\_clust, k=4)

# Number of members in each cluster

table(groups)

#append cluster labels to original data

final\_data <- cbind(USArrests, cluster = groups)

#display first six rows of final data

head(final\_data)

#find mean values for each cluster

aggregate(final\_data, by=list(cluster=final\_data$cluster), mean)

# Scatterplot of cluster assignments (assuming the first two columns are relevant)

plot(final\_data[, c(1, 2)], col = groups, pch = 19, main = "Cluster Assignments")

legend("topright", legend = unique(groups), col = 1:max(groups), pch = 19)

# Calculate cluster means

cluster\_means <- final\_data %>%

group\_by(cluster) %>%

summarise(mean\_Murder = mean(Murder))

# Create the bar plot

ggplot(final\_data, aes(x = as.factor(cluster), y = Murder, fill = as.factor(cluster))) +

geom\_bar(stat = "summary", fun = "mean") +

labs(x = "Cluster", y = "Mean Murder Rate", title = "Cluster Means for Murder Rate") +

scale\_fill\_discrete(name = "Cluster") +

geom\_text(data = cluster\_means, aes(label = round(mean\_Murder, 2), y = mean\_Murder), vjust = -0.5, size = 4)

11

# Load necessary libraries

library(MASS)

library(caret)

library(e1071)

library(randomForest)

library(rpart)

library(rpart.plot)

# Load BostonHousing dataset

data(Boston)

# Set seed for reproducibility

set.seed(123)

# Split data into training and testing sets

trainIndex <- createDataPartition(Boston$medv, p = 0.8, list = FALSE)

trainData <- Boston[trainIndex, ]

testData <- Boston[-trainIndex, ]

# Linear Regression

lm\_model <- lm(medv ~ ., data = trainData)

lm\_predictions <- predict(lm\_model, newdata = testData)

plot(lm\_predictions, testData$medv, main = "Linear Regression", xlab = "Predicted", ylab = "Actual")

# Polynomial Regression using only the variables present in the dataset

poly\_model <- lm(medv ~ poly(crim, 2) + poly(zn, 2) + poly(indus, 2) +

poly(nox, 2) + poly(rm, 2) + poly(age, 2) +

poly(dis, 2) + poly(rad, 2) + poly(tax, 2) +

poly(ptratio, 2) + poly(lstat, 2),

data = trainData)

poly\_predictions <- predict(poly\_model, newdata = testData)

plot(poly\_predictions, testData$medv, main = "Polynomial Regression", xlab = "Predicted", ylab = "Actual")

# Decision Tree Regression

tree\_model <- rpart(medv ~ ., data = trainData, method = "anova")

tree\_predictions <- predict(tree\_model, newdata = testData)

rpart.plot(tree\_model, main = "Decision Tree Regression")

# Support Vector Regression

svm\_model <- svm(medv ~ ., data = trainData)

svm\_predictions <- predict(svm\_model, newdata = testData)

plot(svm\_predictions, testData$medv, main = "Support Vector Regression", xlab = "Predicted", ylab = "Actual")

# Random Forest Regression

rf\_model <- randomForest(medv ~ ., data = trainData)

rf\_predictions <- predict(rf\_model, newdata = testData)

plot(rf\_predictions, testData$medv, main = "Random Forest Regression", xlab = "Predicted", ylab = "Actual")

# Calculate RMSE for each model

rmse <- function(error) sqrt(mean(error^2))

lm\_rmse <- rmse(testData$medv - lm\_predictions)

poly\_rmse <- rmse(testData$medv - poly\_predictions)

tree\_rmse <- rmse(testData$medv - tree\_predictions)

svm\_rmse <- rmse(testData$medv - svm\_predictions)

rf\_rmse <- rmse(testData$medv - rf\_predictions)

# Print RMSEs for each model

cat("Linear Regression RMSE:", lm\_rmse, "\n")

cat("Polynomial Regression RMSE:", poly\_rmse, "\n")

cat("Decision Tree Regression RMSE:", tree\_rmse, "\n")

cat("Support Vector Regression RMSE:", svm\_rmse, "\n")

cat("Random Forest Regression RMSE:", rf\_rmse, "\n")

# Create bar graphs for each model's predictions

models <- c("Linear Regression", "Polynomial Regression", "Decision Tree Regression",

"Support Vector Regression", "Random Forest Regression")

predictions <- list(lm\_predictions, poly\_predictions, tree\_predictions,

svm\_predictions, rf\_predictions)

# Bar plot for predictions

par(mfrow = c(2, 3)) # Adjust layout for better visibility

for (i in 1:length(models)) {

barplot(predictions[[i]], main = paste(models[i], "Predictions"), ylab = "Predicted Values")

}

# Create a data frame for RMSE values

rmse\_values <- data.frame(

Model = c("Linear Regression", "Polynomial Regression", "Decision Tree Regression",

"Support Vector Regression", "Random Forest Regression"),

RMSE = c(lm\_rmse, poly\_rmse, tree\_rmse, svm\_rmse, rf\_rmse)

)

# Plot RMSE comparison using ggplot2

ggplot(rmse\_values, aes(x = Model, y = RMSE, fill = Model)) +

geom\_bar(stat = "identity") +

labs(title = "RMSE Comparison of Regression Models", x = "Regression Model", y = "RMSE") +

theme\_minimal() +

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

scale\_fill\_brewer(palette = "Set3")

# Find the minimum RMSE

rmse\_values <- c(lm\_rmse, poly\_rmse, tree\_rmse, svm\_rmse, rf\_rmse)

min\_rmse <- min(rmse\_values)

cat("\n The model with the minimum RMSE is: ")

if (min\_rmse == lm\_rmse) {

cat("Linear Regression")

} else if (min\_rmse == poly\_rmse) {

cat("Polynomial Regression")

} else if (min\_rmse == tree\_rmse) {

cat("Decision Tree Regression")

} else if (min\_rmse == svm\_rmse) {

cat("Support Vector Regression")

} else {

cat("Random Forest Regression")

}

12

# Load required libraries

library(caret)

library(mlbench)

library(glmnet)

library(randomForest)

library(e1071)

library(class)

library(naivebayes)

library(nnet)

library(xgboost)

library(kernlab)

library(ada)

library(MASS)

library(ggplot2)

library(pROC)

library(class)

# Load Iris dataset

data("iris")

set.seed(123)

# Split the dataset into training and testing sets

trainIndex <- createDataPartition(iris$Species, p = 0.8, list = FALSE)

trainData <- iris[trainIndex, ]

testData <- iris[-trainIndex, ]

# Logistic Regression (Multinomial for multi-class)

log\_model <- multinom(Species ~ ., data = trainData)

log\_pred <- predict(log\_model, newdata = testData)

log\_accuracy <- mean(log\_pred == testData$Species)

# Decision Trees

tree\_model <- train(Species ~ ., data = trainData, method = "rpart")

tree\_pred <- predict(tree\_model, newdata = testData)

tree\_accuracy <- mean(tree\_pred == testData$Species)

# Random Forest

rf\_model <- randomForest(Species ~ ., data = trainData)

rf\_pred <- predict(rf\_model, newdata = testData)

rf\_accuracy <- mean(rf\_pred == testData$Species)

# Support Vector Machines

svm\_model <- svm(Species ~ ., data = trainData)

svm\_pred <- predict(svm\_model, newdata = testData)

svm\_accuracy <- mean(svm\_pred == testData$Species)

# K-Nearest Neighbors

knn\_pred <- class::knn(train = trainData[, -5], test = testData[, -5], cl = trainData$Species, k = 3)

knn\_accuracy <- mean(knn\_pred == testData$Species)

# Naive Bayes

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

nb\_pred <- predict(nb\_model, newdata = testData)

nb\_accuracy <- mean(nb\_pred == testData$Species)

# Neural Networks

nn\_model <- nnet(Species ~ ., data = trainData, size = 5)

nn\_pred <- predict(nn\_model, newdata = testData, type = "class")

nn\_accuracy <- mean(nn\_pred == testData$Species)

# Gradient Boosting

gb\_model <- train(Species ~ ., data = trainData, method = "gbm")

gb\_pred <- predict(gb\_model, newdata = testData)

gb\_accuracy <- mean(gb\_pred == testData$Species)

# Linear Discriminant Analysis

lda\_model <- lda(Species ~ ., data = trainData)

lda\_pred <- predict(lda\_model, newdata = testData)$class

lda\_accuracy <- mean(lda\_pred == testData$Species)

# Compare the predictions

comparison <- data.frame(

Actual = testData$Species,

Logistic\_Regression = log\_pred,

Decision\_Trees = tree\_pred,

Random\_Forest = rf\_pred,

SVM = svm\_pred,

KNN = knn\_pred,

Naive\_Bayes = nb\_pred,

Neural\_Networks = nn\_pred,

Gradient\_Boosting = gb\_pred,

Linear\_Discriminant\_Analysis = lda\_pred

)

print(head(comparison))

# Accuracy comparison data frame

accuracy\_df <- data.frame(

Algorithm = c("Logistic Regression", "Decision Trees", "Random Forest",

"Support Vector Machines", "K-Nearest Neighbors",

"Naive Bayes", "Neural Networks", "Gradient Boosting",

"Linear Discriminant Analysis"),

Accuracy = c(log\_accuracy, tree\_accuracy, rf\_accuracy, svm\_accuracy,

knn\_accuracy, nb\_accuracy, nn\_accuracy, gb\_accuracy,

lda\_accuracy)

)

print(accuracy\_df)

# Plot Accuracy

accuracy\_plot <- ggplot(accuracy\_df, aes(x = Algorithm, y = Accuracy, fill = Algorithm)) +

geom\_bar(stat = "identity") +

labs(title = "Accuracy of Classification Algorithms", x = "Algorithm", y = "Accuracy") +

theme\_minimal() +

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

# Print the accuracy plot

print(accuracy\_plot)

# Identify the maximum accuracy value

max\_accuracy <- max(accuracy\_df$Accuracy)

# Find all algorithms that have the maximum accuracy

best\_algorithms <- accuracy\_df[accuracy\_df$Accuracy == max\_accuracy, ]

# Print the best algorithms and their accuracy

cat("Algorithm(s) with the Best Performance:\n")

for (i in 1:nrow(best\_algorithms)) {

cat("Algorithm:", best\_algorithms$Algorithm[i], "\n")

cat("Accuracy:", best\_algorithms$Accuracy[i], "\n")

}

13

# Load required libraries

install.packages("mclust")

library(ISLR2)

library(cluster)

library(dbscan)

library(factoextra)

library(mclust)

library(ggplot2)

library(kohonen)

library(e1071) # For FCM

library(reshape2) # For data manipulation

# Load the College dataset

data("College")

# Select relevant features for clustering (excluding the college name)

college\_data <- College[, -1] # Exclude the first column (college name)

# Standardize the data

scaled\_data <- scale(college\_data)

# Create a function to calculate silhouette scores

calculate\_silhouette <- function(method, data, clusters) {

if (!is.null(clusters)) {

silhouette\_score <- mean(silhouette(clusters, dist(data)))

return(silhouette\_score)

} else {

return(NA)

}

}

# Initialize a dataframe to store results

results <- data.frame(Method = character(0), Silhouette = numeric(0))

# Perform K-Means clustering

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

kmeans\_silhouette <- calculate\_silhouette("K-Means", scaled\_data, kmeans\_result$cluster)

results <- rbind(results, data.frame(Method = "K-Means", Silhouette = kmeans\_silhouette))

# Perform Hierarchical Clustering

hierarchical\_result <- hclust(dist(scaled\_data))

hierarchical\_silhouette <- calculate\_silhouette("Hierarchical", scaled\_data, cutree(hierarchical\_result, k = 3))

results <- rbind(results, data.frame(Method = "Hierarchical", Silhouette = hierarchical\_silhouette))

# Perform DBSCAN clustering

dbscan\_result <- dbscan(scaled\_data, eps = 3, MinPts = 5)

dbscan\_silhouette <- calculate\_silhouette("DBSCAN", scaled\_data, dbscan\_result$cluster)

results <- rbind(results, data.frame(Method = "DBSCAN", Silhouette = dbscan\_silhouette))

# Perform Gaussian Mixture Model (GMM) clustering

gmm\_result <- Mclust(scaled\_data, G = 3)

gmm\_silhouette <- calculate\_silhouette("GMM", scaled\_data, gmm\_result$classification)

results <- rbind(results, data.frame(Method = "GMM", Silhouette = gmm\_silhouette))

# Perform Self-Organizing Maps (SOM) clustering

som\_result <- som(scale(scaled\_data), grid = somgrid(3, 3, "hexagonal"))

som\_silhouette <- calculate\_silhouette("SOM", scale(scaled\_data), som\_result$unit.classif)

results <- rbind(results, data.frame(Method = "SOM", Silhouette = som\_silhouette))

# Perform Fuzzy C-Means (FCM) clustering

fcm\_result <- cmeans(scaled\_data, centers = 3, m = 1.2)

fcm\_clusters <- fcm\_result$cluster

fcm\_silhouette <- calculate\_silhouette("FCM", scaled\_data, fcm\_clusters)

results <- rbind(results, data.frame(Method = "FCM", Silhouette = fcm\_silhouette))

# Print the results

print(results)

# Find the index of the method with the highest silhouette score

best\_method\_index <- which.max(results$Silhouette)

# Retrieve the best method based on the index

best\_method <- results$Method[best\_method\_index]

# Visualize K-Means clustering

fviz\_cluster(kmeans\_result, data = scaled\_data, geom = "point", main = "K-Means Clustering")

# Visualize Hierarchical Clustering

plot(hierarchical\_result, main = "Hierarchical Clustering Dendrogram")

rect.hclust(hierarchical\_result, k = 3, border = 2:4)

# Visualize DBSCAN clustering

fviz\_cluster(dbscan\_result, data = scaled\_data, geom = "point", main = "DBSCAN Clustering")

# Visualize GMM clustering

plot(gmm\_result, what = "classification", main = "GMM Clustering")

# Perform Self-Organizing Maps (SOM) clustering

som\_result <- som(scale(scaled\_data), grid = somgrid(3, 3, "hexagonal"))

cat("SOM Cluster Summary:\n")

print(som\_result)

# Get cluster assignments from the SOM result

cluster\_assignments <- som\_result$unit.classif

# Create a vector of unique cluster IDs

unique\_clusters <- unique(cluster\_assignments)

# Create a color palette for the clusters

cluster\_colors <- rainbow(length(unique\_clusters))

# Map cluster assignments to colors based on cluster IDs

assigned\_colors <- cluster\_colors[cluster\_assignments]

# Visualize SOM Clustering with assigned colors

plot(som\_result, type = "mapping", pchs = 20, col = assigned\_colors, main = "SOM Clustering")

# Plot FCM clustering assignments

plot(scaled\_data, col = fcm\_clusters, pch = 19, main = "FCM Clustering")

# Create a bar plot of silhouette scores with different colors for each bar

ggplot(results, aes(x = Method, y = Silhouette, fill = Method)) +

geom\_bar(stat = "identity") +

geom\_text(aes(label = round(Silhouette, 2), vjust = -0.5), position = position\_dodge(0.9)) + # Add labels on top of bars

labs(title = "Silhouette Scores for Clustering Algorithms", y = "Silhouette Score") +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 45, hjust = 1)) # Rotate x-axis labels for better readability

# Print the best method and its silhouette score

cat("Best Clustering Method:", best\_method, "\n")

cat("Silhouette Score:", results$Silhouette[best\_method\_index], "\n")

14

# Load required libraries

library(arules)

library(rCBA)

library(ggplot2)

library(reshape2)

library(rJava)

# Load the Groceries dataset

data("Groceries")

# Apriori Algorithm

apriori\_rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))

cat("Apriori Summary:\n")

summary(apriori\_rules)

# Eclat Algorithm

eclat\_rules <- eclat(Groceries, parameter = list(support = 0.1))

cat("Eclat Summary:\n")

summary(eclat\_rules)

# FP-Growth Algorithm (using rCBA)

fpgrowth\_rules <- apriori(Groceries, parameter = list(supp = 0.03, conf = 0.03, minlen = 1, maxlen = 2))

cat("FP-Growth Summary:\n")

summary(fpgrowth\_rules)

# Check rule counts

num\_rules\_apriori <- length(apriori\_rules)

num\_rules\_eclat <- length(eclat\_rules)

num\_rules\_fpgrowth <- length(fpgrowth\_rules)

cat("Number of Apriori Rules:", num\_rules\_apriori, "\n")

cat("Number of Eclat Rules:", num\_rules\_eclat, "\n")

cat("Number of FP-Growth Rules:", num\_rules\_fpgrowth, "\n")

# Summarize rule metrics for Apriori

avg\_support\_apriori <- mean(quality(apriori\_rules)$support)

avg\_confidence\_apriori <- mean(quality(apriori\_rules)$confidence)

avg\_lift\_apriori <- mean(quality(apriori\_rules)$lift)

# Summarize rule metrics for Eclat (only support is typically available)

avg\_support\_eclat <- mean(quality(eclat\_rules)$support)

# Summarize rule metrics for FP-Growth

avg\_support\_fpgrowth <- mean(quality(fpgrowth\_rules)$support)

avg\_confidence\_fpgrowth <- mean(quality(fpgrowth\_rules)$confidence)

avg\_lift\_fpgrowth <- mean(quality(fpgrowth\_rules)$lift)

# Combine metrics for visualization

summary\_data <- data.frame(

Algorithm = c("Apriori", "Eclat", "FP-Growth"),

Num\_Rules = c(num\_rules\_apriori, num\_rules\_eclat, num\_rules\_fpgrowth),

Avg\_Support = c(avg\_support\_apriori, avg\_support\_eclat, avg\_support\_fpgrowth),

Avg\_Confidence = c(avg\_confidence\_apriori, NA, avg\_confidence\_fpgrowth),

Avg\_Lift = c(avg\_lift\_apriori, NA, avg\_lift\_fpgrowth)

)

print(summary\_data)

# Melt the data for ggplot

melted\_summary\_data <- melt(summary\_data, id.vars = "Algorithm")

melted\_summary\_data <- na.omit(melted\_summary\_data) # Remove NA values

# Visualize support, confidence, and lift

ggplot(melted\_summary\_data, aes(x = Algorithm, y = value, fill = variable)) +

geom\_bar(stat = "identity", position = "dodge") +

labs(title = "Algorithm Comparison", y = "Value") +

facet\_wrap(~ variable, scales = "free\_y", nrow = 1) +

theme\_minimal()

# Remove NA values from Avg\_Confidence for the search

cleaned\_confidence <- summary\_data$Avg\_Confidence[!is.na(summary\_data$Avg\_Confidence)]

best\_algorithm\_confidence <- summary\_data[which.max(cleaned\_confidence), ]

# Find the algorithm with the highest average support

best\_algorithm\_support <- summary\_data[which.max(summary\_data$Avg\_Support), ]

# Remove NA values from Avg\_Lift for the search

cleaned\_lift <- summary\_data$Avg\_Lift[!is.na(summary\_data$Avg\_Lift)]

best\_algorithm\_lift <- summary\_data[which.max(cleaned\_lift), ]

# Print the best-performing algorithms

cat("Best-Performing Algorithm based on Average Confidence:\n")

print(best\_algorithm\_confidence)

cat("Best-Performing Algorithm based on Average Support:\n")

print(best\_algorithm\_support)

cat("Best-Performing Algorithm based on Average Lift:\n")

print(best\_algorithm\_lift)