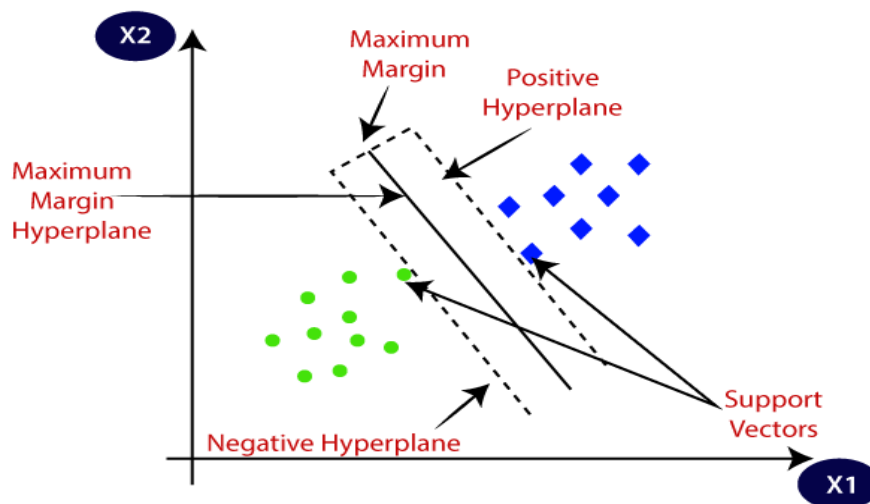


**EXPERIMENT NO:7****AIM:**

Write R program to implement Support Vector Machine(SVM) for Gene Expression Data.

**DESCRIPTION:****Support Vector Machine:**

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well it's best suited for classification. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries that the margin between the closest points of different classes should be as maximum as possible. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

**Support Vector Machine Terminology:**

1. **Hyperplane:** Hyperplane is the decision boundary that is used to separate the data points of different classes in a feature space. In the case of linear classifications, it will be a linear equation i.e.,  $wx+b = 0$ .
2. **Support Vectors:** Support vectors are the closest data points to the hyperplane, which makes a critical role in deciding the hyperplane and margin.
3. **Margin:** Margin is the distance between the support vector and hyperplane. The main objective of the support vector machine algorithm is to maximize the margin. The wider margin indicates better classification performance.
4. **Kernel:** Kernel is the mathematical function, which is used in SVM to map the original input data points into high-dimensional feature spaces, so, that the hyperplane can be easily found out even if the data points are not linearly separable in the original input space. Some of the common kernel functions are linear, polynomial, radial basis function (RBF), and sigmoid.

**Hard Margin: The maximum-margin hyperplane or the hard margin hyperplane is a hyperplane that properly separates the data points of different categories without any misclassifications.**

5. **Soft Margin:** When the data is not perfectly separable or contains outliers, SVM permits a soft margin technique. Each data point has a slack variable introduced by the soft-margin SVM formulation, which softens the strict margin requirement and permits certain misclassifications or violations. It discovers a compromise between increasing the margin and reducing violations.
6. **C:** Margin maximisation and misclassification fines are balanced by the regularisation parameter C in SVM. The penalty for going over the margin or misclassifying data items is decided by it. A stricter penalty is imposed with a greater value of C, which results in a smaller margin and perhaps fewer misclassifications.
7. **Hinge Loss:** A typical loss function in SVMs is hinge loss. It punishes incorrect classifications or margin violations. The objective function in SVM is frequently formed by combining it with the regularisation term.
8. **Dual Problem:** A dual Problem of the optimisation problem that requires locating the Lagrange multipliers related to the support vectors can be used to solve SVM. The dual formulation enables the use of kernel tricks and more effective computing.

### Mathematical intuition of Support Vector Machine:

Consider a binary classification problem with two classes, labelled as +1 and -1. We have a training dataset consisting of input feature vectors  $X$  and their corresponding class labels  $Y$ .

The equation for the linear hyperplane can be written as:

$$w^T x + b = 0$$

The vector  $W$  represents the normal vector to the hyperplane. i.e the direction perpendicular to the hyperplane. The parameter  $b$  in the equation represents the offset or distance of the hyperplane from the origin along the normal vector  $w$ .

The distance between a data point  $x_i$  and the decision boundary can be calculated as:

$$d_i = \frac{w^T x_i + b}{\|w\|}$$

where  $\|w\|$  represents the Euclidean norm of the weight vector  $w$ . Euclidean norm of the normal vector  $W$

For Linear SVM classifier:

$$\hat{y} = \begin{cases} 1 & : w^T x + b \geq 0 \\ 0 & : w^T x + b < 0 \end{cases}$$

**Optimization:**

- **For Hard margin linear SVM classifier:**

$$\begin{aligned} \underset{w,b}{\text{minimize}} \quad & \frac{1}{2} w^T w = \underset{W,b}{\text{minimize}} \quad \frac{1}{2} \|w\|^2 \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1 \text{ for } i = 1, 2, 3, \dots, m \end{aligned}$$

The target variable or label for the  $i^{\text{th}}$  training instance is denoted by the symbol  $t_i$  in this statement. And  $t_i = -1$  for negative occurrences (when  $y_i = 0$ ) and  $t_i = 1$  for positive instances (when  $y_i = 1$ ) respectively. Because we require the decision boundary that satisfy the

constraint:  $t_i(w^T x_i + b) \geq 1$

- **For Soft margin linear SVM classifier:**

$$\begin{aligned} \underset{w,b}{\text{minimize}} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^m \zeta_i \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1 - \zeta_i \text{ and } \zeta_i \geq 0 \text{ for } i = 1, 2, 3, \dots, m \end{aligned}$$

- **Dual Problem:** A dual Problem of the optimisation problem that requires locating the Lagrange multipliers related to the support vectors can be used to solve SVM. The optimal Lagrange multipliers  $\alpha(i)$  that maximize the following dual objective function

$$\underset{\alpha}{\text{maximize}} : \frac{1}{2} \sum_{i \rightarrow m} \sum_{j \rightarrow m} \alpha_i \alpha_j t_i t_j K(x_i, x_j) - \sum_{i \rightarrow m} \alpha_i$$

where,

- $\alpha_i$  is the Lagrange multiplier associated with the  $i$ th training sample.
- $K(x_i, x_j)$  is the kernel function that computes the similarity between two samples  $x_i$  and  $x_j$ . It allows SVM to handle nonlinear classification problems by implicitly mapping the samples into a higher-dimensional feature space.
- The term  $\sum \alpha_i$  represents the sum of all Lagrange multipliers.

The SVM decision boundary can be described in terms of these optimal Lagrange multipliers and the support vectors once the dual issue has been solved and the optimal Lagrange multipliers have been discovered. The training samples that have  $i > 0$  are the support vectors, while the decision boundary is supplied by:

$$w = \sum_{i \rightarrow m} \alpha_i t_i K(x_i, x) + b$$

$$t_i(w^T x_i - b) = 1 \iff b = w^T x_i - t_i$$

### Types of Support Vector Machine:

Based on the nature of the decision boundary, Support Vector Machines (SVM) can be divided into two main parts:

- **Linear SVM:** Linear SVMs use a linear decision boundary to separate the data points of different classes. When the data can be precisely linearly separated, linear SVMs are very suitable. This means that a single straight line (in 2D) or a hyperplane (in higher dimensions) can entirely divide the data points into their respective classes. A hyperplane that maximizes the margin between the classes is the decision boundary.
- **Non-Linear SVM:** Non-Linear SVM can be used to classify data when it cannot be separated into two classes by a straight line (in the case of 2D). By using kernel functions, nonlinear SVMs can handle nonlinearly separable data. The original input data is transformed by these kernel functions into a higher-dimensional feature space, where the data points can be linearly separated. A linear SVM is used to locate a nonlinear decision boundary in this modified space.

### Popular kernel functions in SVM:

The SVM kernel is a function that takes low-dimensional input space and transforms it into higher-dimensional space, i.e., it converts non separable problems to separable problems.

It is mostly useful in non-linear separation problems. Simply put the kernel, does some extremely complex data transformations and then finds out the process to separate the data based on the labels or outputs defined.

$$\text{Linear} : K(w, b) = w^T x + b$$

$$\text{Polynomial} : K(w, x) = (\gamma w^T x + b)^N$$

$$\text{Gaussian RBF} : K(w, x) = \exp(-\gamma \|x_i - x_j\|^n)$$

$$\text{Sigmoid} : K(x_i, x_j) = \tanh(\alpha x_i^T x_j + b)$$

**CODE:**

```
# Load necessary packages

library(ISLR2)

library(pROC)

library(ggplot2)

library(e1071)

library(plotmo)

library(plotly)

# 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

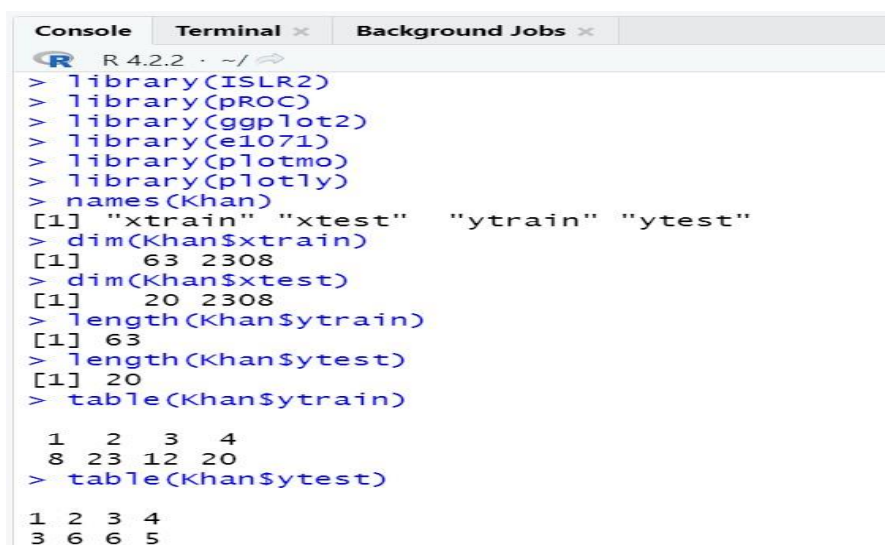
length(Khan$ytrain)

length(Khan$ytest)

# Display class distribution in training and testing labels

table(Khan$ytrain)

table(Khan$ytest)
```

**OUTPUT:**

The screenshot shows an R console window with the following output:

```
R 4.2.2 · ~/
> library(ISLR2)
> library(pROC)
> library(ggplot2)
> library(e1071)
> library(plotmo)
> library(plotly)
> names(Khan)
[1] "xtrain" "xtest"  "ytrain" "ytest"
> dim(Khan$xtrain)
[1] 63 2308
> dim(Khan$xtest)
[1] 20 2308
> length(Khan$ytrain)
[1] 63
> length(Khan$ytest)
[1] 20
> table(Khan$ytrain)
 1  2  3  4
8 23 12 20
> table(Khan$ytest)
 1  2  3  4
3  6  6  5
```

**CODE:**

```
# 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)

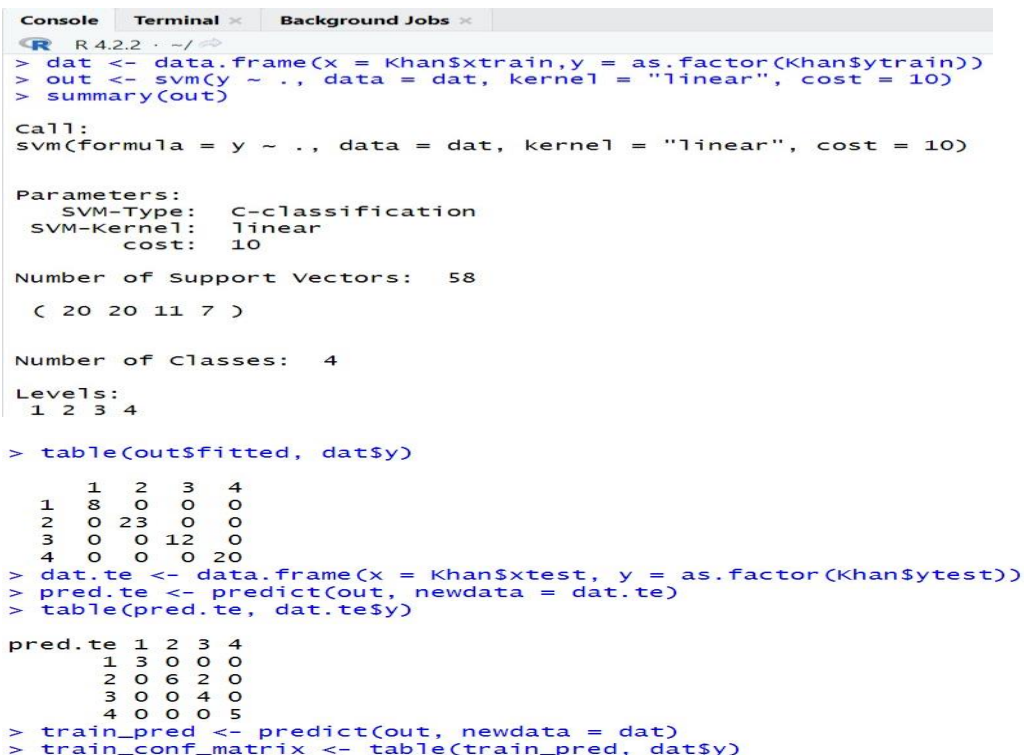
# Compare the fitted values with actual labels in the training data
table(out$fitted, dat$y)

# 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)
```

**OUTPUT :**


```
Console Terminal × Background Jobs ×
R 4.2.2 . ~/
> dat <- data.frame(x = Khan$xtrain, y = as.factor(Khan$ytrain))
> out <- svm(y ~ ., data = dat, kernel = "linear", cost = 10)
> summary(out)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10)

Parameters:
  SVM-Type:  C-classification
 SVM-Kernel: linear
      cost:  10

Number of Support Vectors:  58
( 20 20 11 7 )

Number of Classes:  4

Levels:
 1 2 3 4

> table(out$fitted, dat$y)

      1  2  3  4
1  8  0  0  0
2  0 23  0  0
3  0  0 12  0
4  0  0  0 20

> 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)

pred.te 1 2 3 4
      1 3 0 0 0
      2 0 6 2 0
      3 0 0 4 0
      4 0 0 0 5

> train_pred <- predict(out, newdata = dat)
> train_conf_matrix <- table(train_pred, dat$y)
```

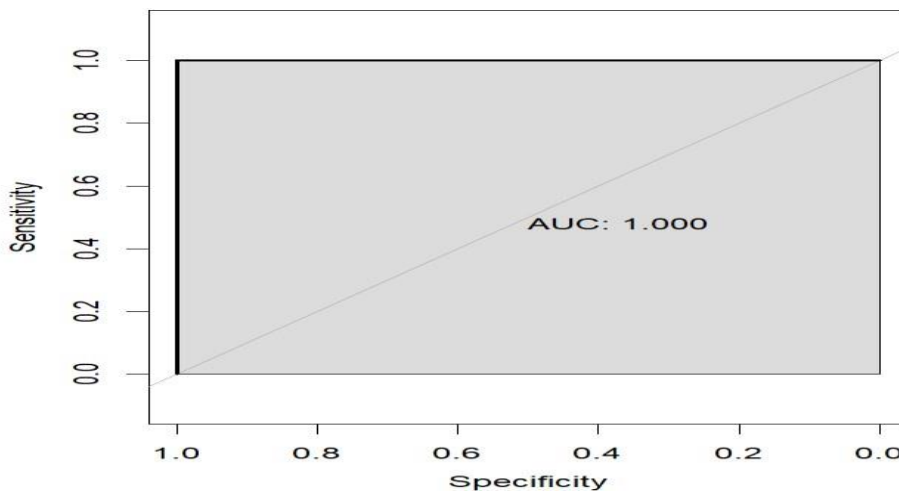
**CODE:**

```
# 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
ggplot(train_conf_df, aes(x = Predicted, y = Actual, fill = Frequency)) +
  geom_tile() +
  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)
```

**CODE:**

```
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)
```

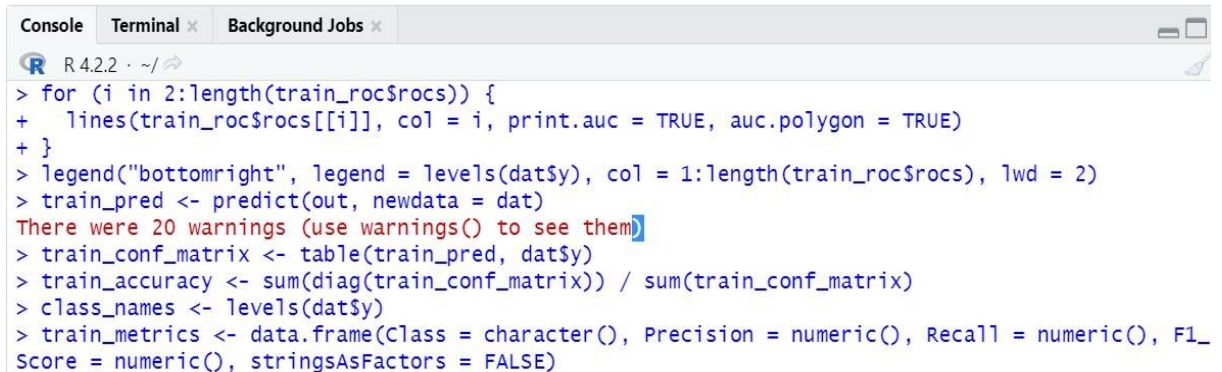
# Calculate precision, recall, and F1-score for each class in training data

```
class_names <- levels(dat$y)
```

```
train_metrics <- data.frame(Class = character(), Precision = numeric(), Recall = numeric(),
```

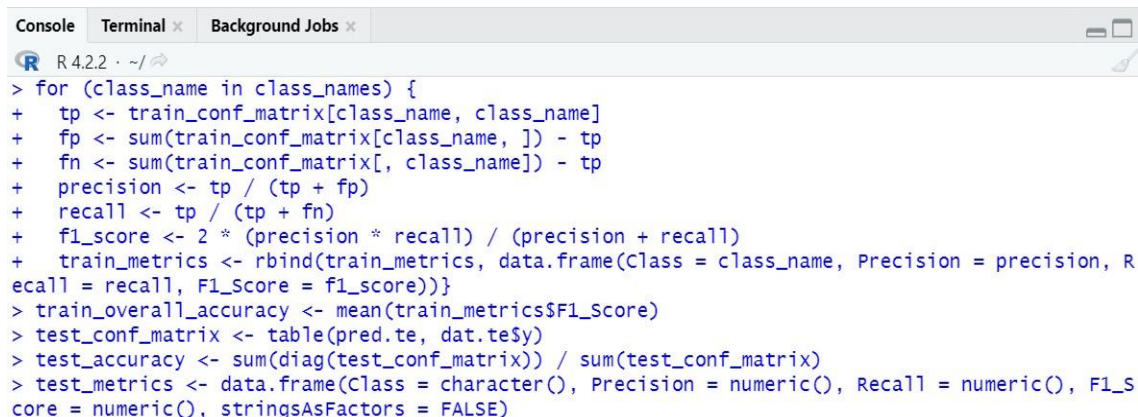
```
F1_Score = numeric(), stringsAsFactors = FALSE)
```

## OUTPUT:



```
R 4.2.2 · ~/
> 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)
> train_pred <- predict(out, newdata = dat)
There were 20 warnings (use warnings() to see them)
> train_conf_matrix <- table(train_pred, dat$y)
> train_accuracy <- sum(diag(train_conf_matrix)) / sum(train_conf_matrix)
> class_names <- levels(dat$y)
> train_metrics <- data.frame(Class = character(), Precision = numeric(), Recall = numeric(), F1_Score = numeric(), stringsAsFactors = FALSE)
```

## OUTPUT:



```
R 4.2.2 · ~/
> 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))}
> 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)
```

## CODE:

```
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")

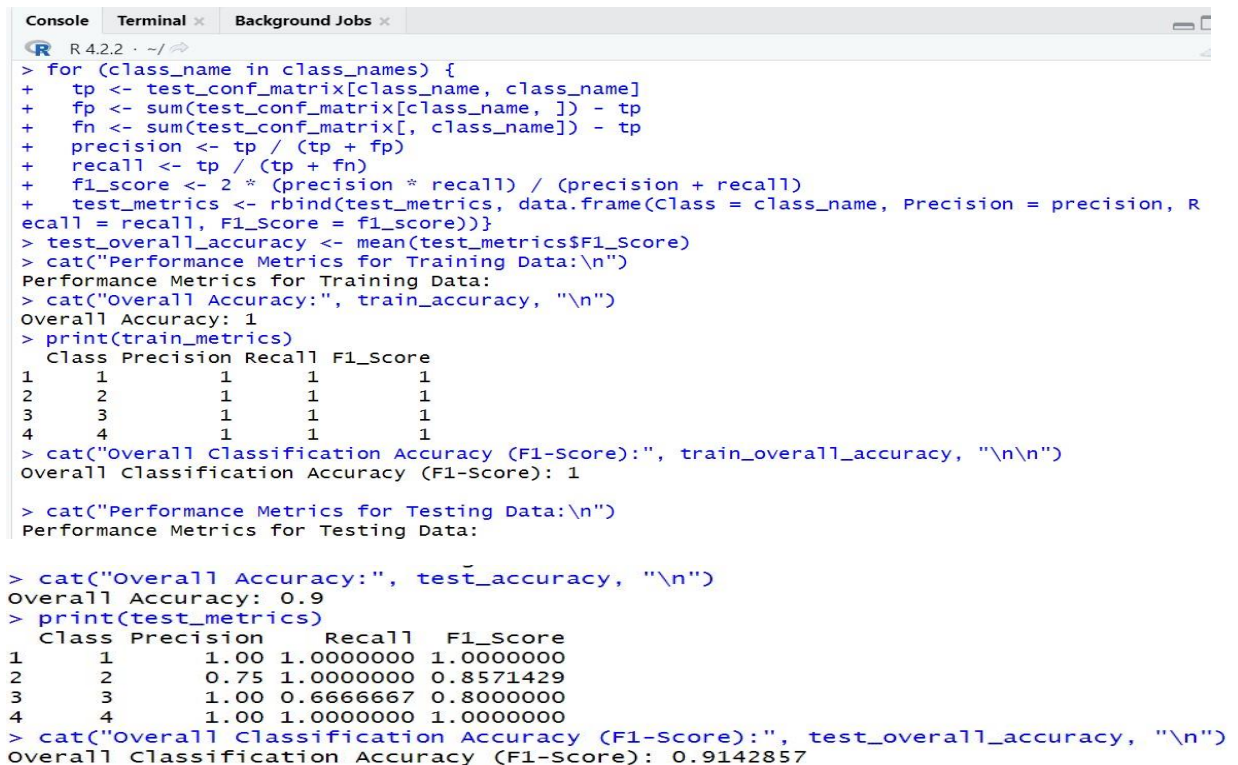
cat("Overall Accuracy:", test_accuracy, "\n")

print(test_metrics)

cat("Overall Classification Accuracy (F1-Score):", test_overall_accuracy, "\n")

```

## **OUTPUT:**



```

R 4.2.2 ~ /
> 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))
+ }
> test_overall_accuracy <- mean(test_metrics$F1_Score)
> cat("Performance Metrics for Training Data:\n")
Performance Metrics for Training Data:
> cat("Overall Accuracy:", train_accuracy, "\n")
Overall Accuracy: 1
> print(train_metrics)
  Class Precision Recall F1_Score
1     1         1      1         1
2     2         1      1         1
3     3         1      1         1
4     4         1      1         1
> cat("Overall Classification Accuracy (F1-Score):", train_overall_accuracy, "\n\n")
Overall Classification Accuracy (F1-Score): 1

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

> cat("Overall Accuracy:", test_accuracy, "\n")
Overall Accuracy: 0.9
> print(test_metrics)
  Class Precision Recall F1_Score
1     1         1.00 1.0000000 1.0000000
2     2         0.75 1.0000000 0.8571429
3     3         1.00 0.6666667 0.8000000
4     4         1.00 1.0000000 1.0000000
> cat("Overall Classification Accuracy (F1-Score):", test_overall_accuracy, "\n")
Overall Classification Accuracy (F1-Score): 0.9142857

```

## **CODE:**

```

# 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",p

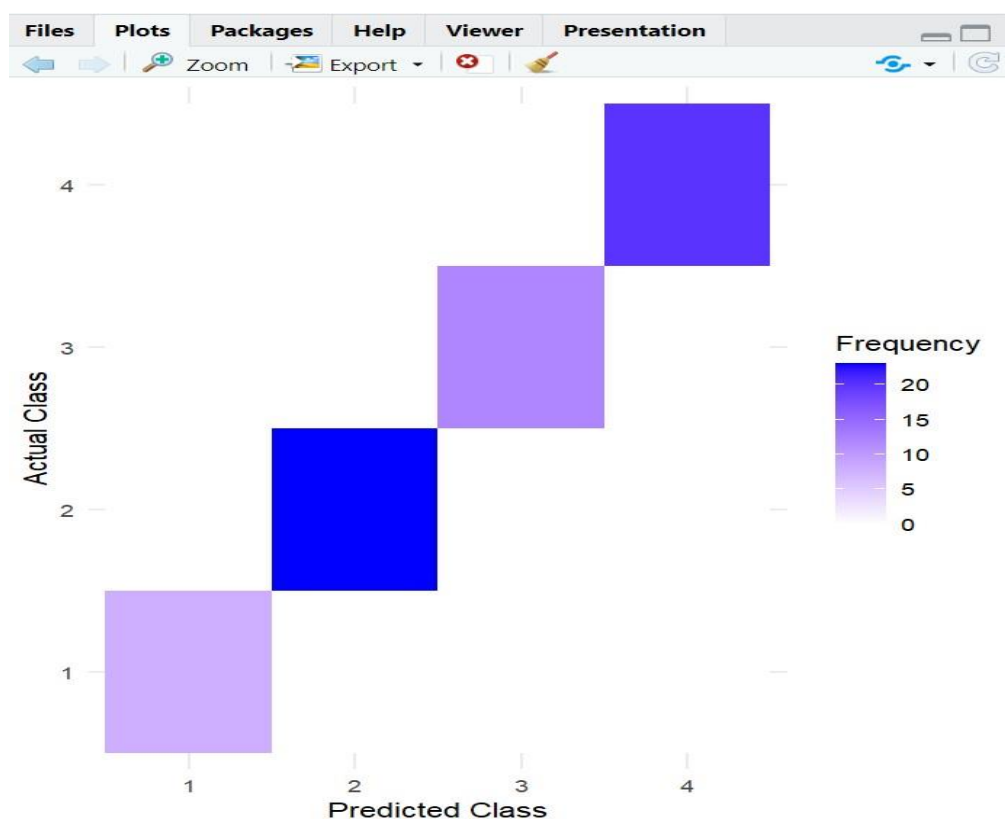
```



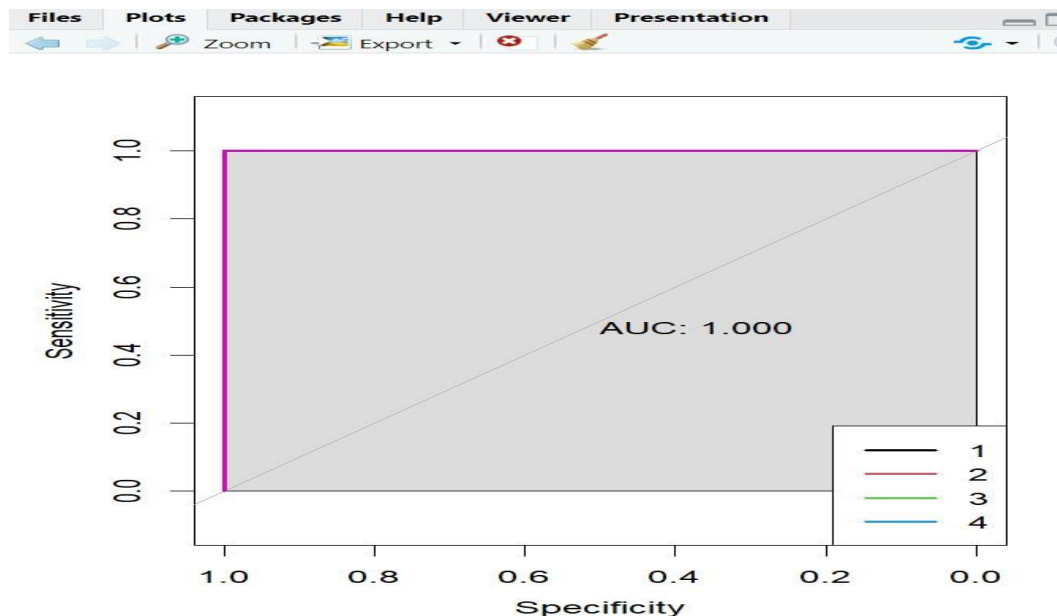
```
x = "Class" )  
  
theme_minimal() +  
theme(axis.text.x = element_text(angle = 45, hjust = 1))  
  
# Calculate multiclass ROC curve and AUC for training data  
train_roc <- multiclass.roc(dat$y, numeric_pred)  
  
# Plot the multiclass ROC curves for each class  
plot(train_roc$rocs[[1]], col = 1, print.auc = TRUE, auc.polygon =  
TRUE)
```

### **OUTPUT:**

```
> train_conf_df <- as.data.frame(as.table(train_conf_matrix))  
> colnames(train_conf_df) <- c("Predicted", "Actual", "Frequency")  
> ggplot(train_conf_df, aes(x = Predicted, y = Actual, fill = Frequency)) +  
+   geom_tile() +  
+   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$y, numeric_pred)
Setting direction: controls < cases
Setting direction: controls < cases
Setting direction: controls < cases
Setting direction: controls < cases
Setting direction: controls < cases
Setting direction: controls < cases
> plot(train_roc$rocs[[1]], col = 1, print.auc = TRUE, auc.polygon = TRUE)
```



### CODE:

```
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)
```

### OUTPUT:



### CODE:

```
# 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))
```

### OUTPUT:

