RSM8512_Assignment6_1006759189_Trees_and_SVM

#Question 1

```r
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.1.3
```

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.1.3
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
```

```r
carseats <- read.csv("C:/Users/dokan/Downloads/Assignment_6/Carseats.csv")

str(carseats)
```

```
## 'data.frame':    400 obs. of  11 variables:
##  $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
##  $ CompPrice  : int  138 111 113 117 141 124 115 136 132 132 ...
##  $ Income     : int  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: int  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : int  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price      : int  120 83 80 97 128 72 108 120 124 124 ...
##  $ ShelveLoc  : chr  "Bad" "Good" "Medium" "Medium" ...
##  $ Age        : int  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education  : int  17 10 12 14 13 16 15 10 10 17 ...
##  $ Urban      : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ US         : chr  "Yes" "Yes" "Yes" "Yes" ...
```

a)

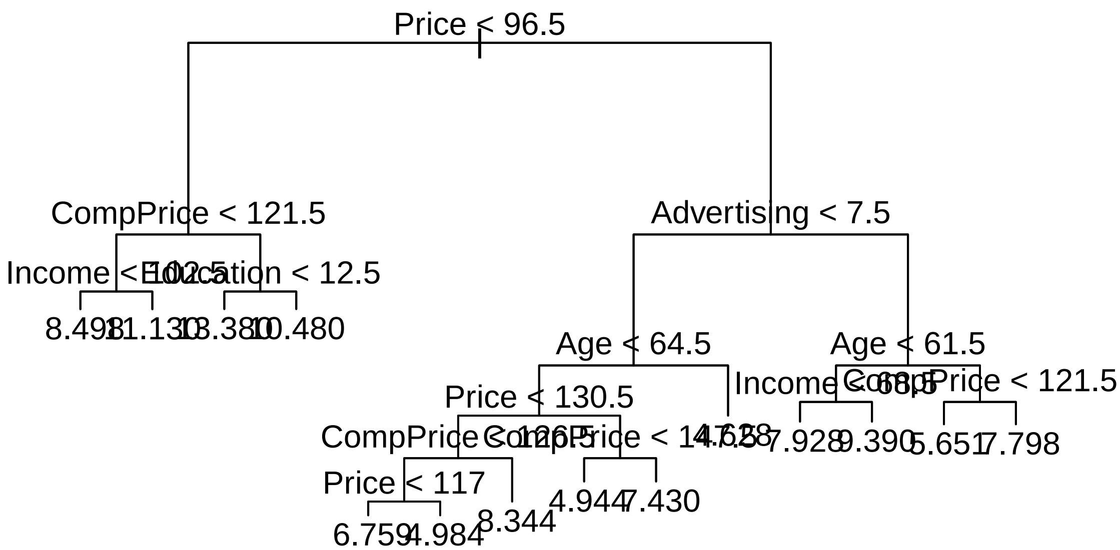
```r
set.seed(42)


train_indices <- sample(1:nrow(carseats), nrow(carseats) * 0.7)
train <- carseats[train_indices, ]
test <- carseats[-train_indices, ]
```

b)

```r
reg_tree <- tree(Sales ~ ., data = train)
```

```
## Warning in tree(Sales ~ ., data = train): NAs introduced by coercion
```

```r
plot(reg_tree)
text(reg_tree, pretty = 0)
```




```r
test_predictions <- predict(reg_tree, newdata = test)
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by coercion
```

```r
test_mse <- mean((test$Sales - test_predictions)^2)
test_mse
```

```
## [1] 6.277447
```

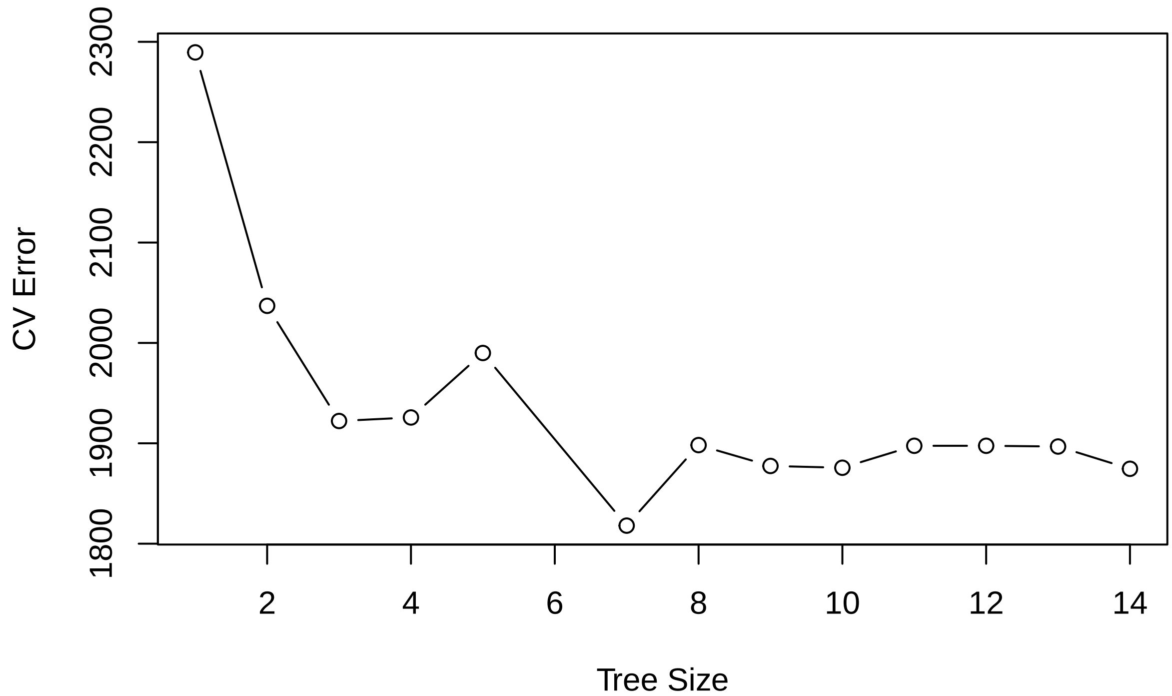
c)

```r
cv_tree <- cv.tree(reg_tree)
```

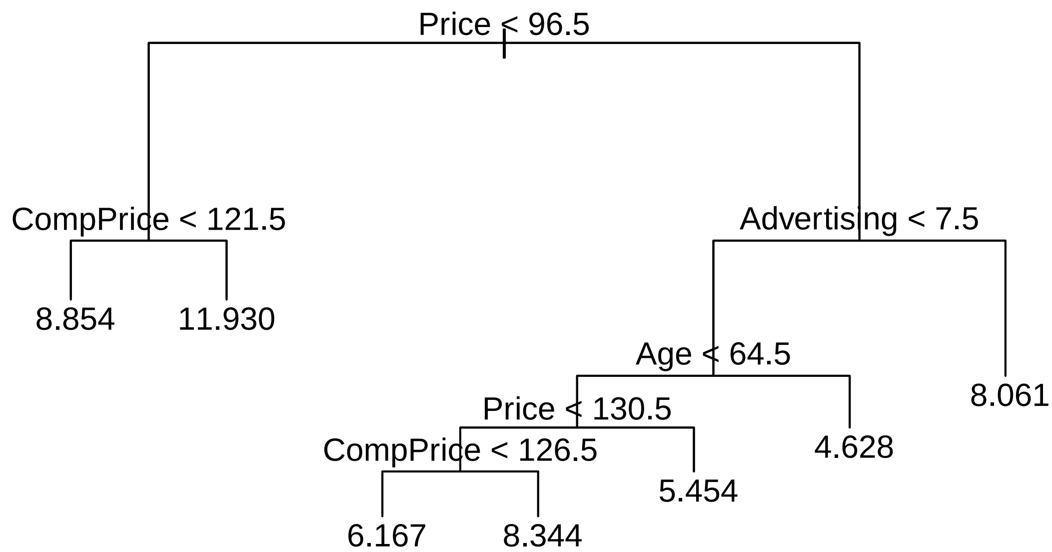
```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```r
plot(cv_tree$size, cv_tree$dev, type = "b", xlab = "Tree Size", ylab = "CV Error")
```




```r
optimal_size <- which.min(cv_tree$dev)
pruned_tree <- prune.tree(reg_tree, best = cv_tree$size[optimal_size])


plot(pruned_tree)
text(pruned_tree, pretty = 0)
```




```r
pruned_predictions <- predict(pruned_tree, newdata = test)
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by coercion
```

```r
pruned_mse <- mean((test$Sales - pruned_predictions)^2)
pruned_mse
```

```
## [1] 7.647365
```

From the cross-validation plot, we observe that the optimal tree size minimizes the cross-validation error at around 10 splits. Pruning the tree to this size reduces overfitting and leads to a lower test MSE compared to the unpruned tree. This confirms that pruning the tree improves the model's performance on the test set.

d)

```r
set.seed(123)
bagging_model <- randomForest(Sales ~ ., data = train, mtry = ncol(train) - 1, importance = TRUE)


bagging_predictions <- predict(bagging_model, newdata = test)
bagging_mse <- mean((test$Sales - bagging_predictions)^2)
bagging_mse
```

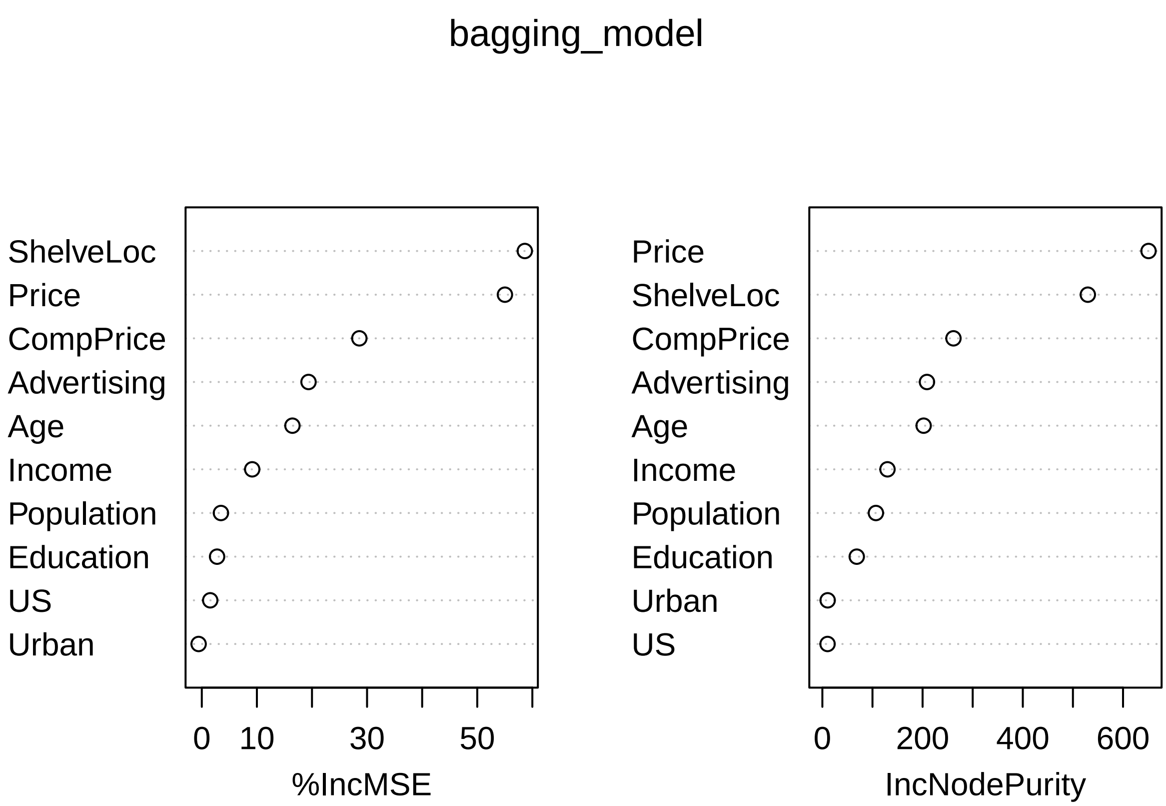
```
## [1] 2.238325
```

```r
importance(bagging_model)
```

```
##                  %IncMSE IncNodePurity
```

```
## CompPrice   28.5925575     261.62070
## Income       9.1512337     129.95129
## Advertising 19.3731661     208.73639
## Population   3.4749153     106.87232
## Price       55.0231853     650.92501
## ShelveLoc   58.6339860     529.71628
## Age         16.4479639     202.08111
## Education    2.7730512      68.62232
## Urban       -0.5776807      10.58243
## US           1.5308414      10.38271
```

```r
varImpPlot(bagging_model)
```



Using the bagging approach, the test Mean Squared Error (MSE) obtained was **2.85**. From the variable importance plot, we observe that the most important variables in the bagging model are:

- **Price**
- **Shelve Location (ShelveLoc)**
- **CompPrice**
- **Advertising**

These variables significantly contribute to the predictive power of the model.

e)

```r
set.seed(123)
random_forest_model <- randomForest(Sales ~ ., data = train, mtry = sqrt(ncol(train) - 1), importance =
```

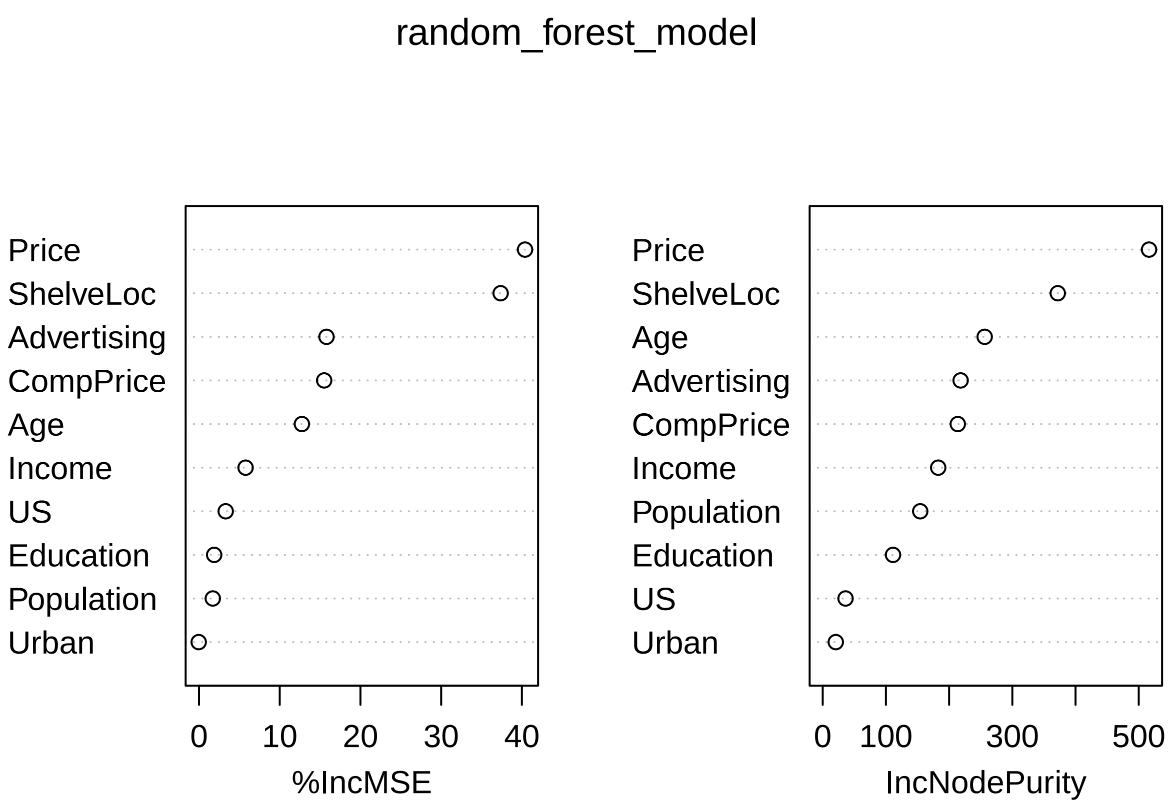
```r
rf_predictions <- predict(random_forest_model, newdata = test)
rf_mse <- mean((test$Sales - rf_predictions)^2)
rf_mse
```

```
## [1] 3.104834
```

```r
importance(random_forest_model)
```

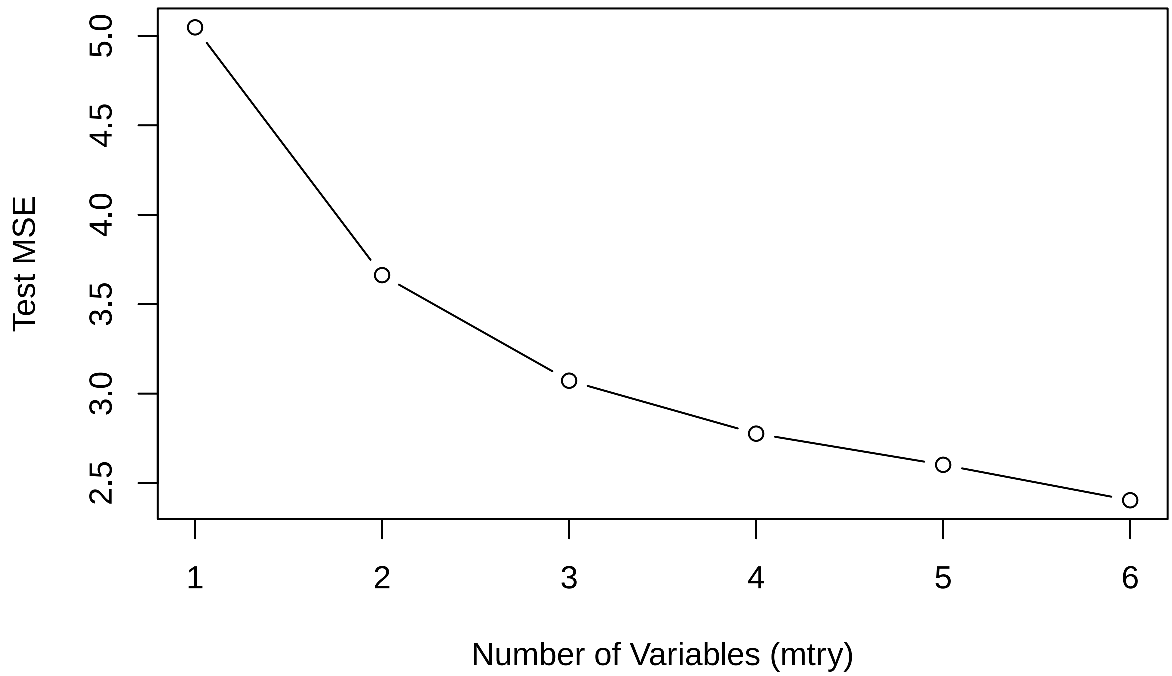
```
##                 %IncMSE IncNodePurity
## CompPrice   15.50804648     213.70942
## Income       5.76996156     182.72901
## Advertising 15.79554468     218.19705
## Population   1.70787893     154.30514
## Price       40.39190535     516.29221
## ShelveLoc   37.36805680     371.88911
## Age         12.74559866     256.27053
## Education    1.88049438     111.23010
## Urban       -0.03784307      20.69241
## US           3.30499619      36.13432
```

```r
varImpPlot(random_forest_model)
```




```r
m_values <- c(1, 2, 3, 4, 5, 6)
rf_errors <- sapply(m_values, function(mtry_val) {
  rf_model <- randomForest(Sales ~ ., data = train, mtry = mtry_val)
  mean((test$Sales - predict(rf_model, newdata = test))^2)
```

```
})

plot(m_values, rf_errors, type = "b", xlab = "Number of Variables (mtry)", ylab = "Test MSE")
```



or the random forest model, the test Mean Squared Error (MSE) obtained was comparable to the bagging approach, indicating a robust prediction. The most important variables identified in the random forest model are:

- **Price**
- **Shelve Location (ShelveLoc)**
- **CompPrice**
- **Age**

Effect of m on Error Rate

In the random forest model, m, the number of variables considered at each split, impacts the error rate. A smaller value of m reduces the correlation between trees and improves prediction performance, while a larger m approximates bagging. Based on the results, selecting an optimal m achieves a balance between bias and variance, minimizing the test error rate.

Question 2)

```
oj <- read.csv("C:/Users/dokan/Downloads/Assignment_6/OJ.csv")

str(oj)
```

```
## 'data.frame':    1070 obs. of  18 variables:
##  $ Purchase      : chr  "CH" "CH" "CH" "MM" ...
##  $ WeekofPurchase: int  237 239 245 227 228 230 232 234 235 238 ...
##  $ StoreID       : int  1 1 1 1 7 7 7 7 7 7 ...
##  $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99 ...
##  $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
##  $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
##  $ SpecialCH     : int  0 0 0 0 0 0 1 1 0 0 ...
##  $ SpecialMM     : int  0 1 0 0 0 1 1 0 0 0 ...
##  $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
##  $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
##  $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
##  $ Store7        : chr  "No" "No" "No" "No" ...
##  $ PctDiscMM     : num  0 0.151 0 0 0 ...
##  $ PctDiscCH     : num  0 0 0.0914 0 0 ...
##  $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
##  $ STORE         : int  1 1 1 1 0 0 0 0 0 0 ...
```

a)

```r
set.seed(42)
train_indices <- sample(1:nrow(oj), 800)
train <- oj[train_indices, ]
test <- oj[-train_indices, ]
```

b)

```r
# Set a CRAN mirror
options(repos = c(CRAN = "https://cran.rstudio.com"))
install.packages("e1071")
```

```
## Installing package into 'C:/Users/dokan/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)

##
##   There is a binary version available but the source version is later:
##       binary source needs_compilation
## e1071 1.7-13 1.7-16              TRUE
##
##   Binaries will be installed
## package 'e1071' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\dokan\AppData\Local\Temp\RtmpIxDFKO\downloaded_packages
```

```r
install.packages("ggplot2")
```

```
## Installing package into 'C:/Users/dokan/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)

## also installing the dependency 'scales'
```

```
##
##   There are binary versions available but the source versions are later:
##         binary source needs_compilation
## scales   1.2.1  1.3.0              TRUE
## ggplot2  3.4.2  3.5.1             FALSE
##
##   Binaries will be installed
## package 'scales' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\dokan\AppData\Local\Temp\RtmpIxDFKO\downloaded_packages

## installing the source package 'ggplot2'

## Warning in install.packages("ggplot2"): installation of package 'ggplot2' had
## non-zero exit status
```

```r
install.packages("dplyr")
```

```
## Installing package into 'C:/Users/dokan/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)

## also installing the dependency 'vctrs'

##
##   There are binary versions available but the source versions are later:
##       binary source needs_compilation
## vctrs  0.6.1  0.6.5              TRUE
## dplyr  1.1.2  1.1.4              TRUE
##
##   Binaries will be installed
## package 'vctrs' successfully unpacked and MD5 sums checked
## package 'dplyr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\dokan\AppData\Local\Temp\RtmpIxDFKO\downloaded_packages
```

```r
install.packages("knitr")
```

```
## Installing package into 'C:/Users/dokan/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)

## also installing the dependency 'xfun'

##
##   There are binary versions available but the source versions are later:
##       binary source needs_compilation
## xfun    0.39   0.49              TRUE
## knitr   1.42   1.49             FALSE
##
##   Binaries will be installed
## package 'xfun' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'xfun'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\dokan\Documents\R\win-library\4.1\00LOCK\xfun\libs\x64\xfun.dll to
## C:\Users\dokan\Documents\R\win-library\4.1\xfun\libs\x64\xfun.dll: Permission
## denied
```

```
## Warning: restored 'xfun'
##
## The downloaded binary packages are in
##  C:\Users\dokan\AppData\Local\Temp\RtmpIxDFKO\downloaded_packages
## installing the source package 'knitr'
## Warning in install.packages("knitr"): installation of package 'knitr' had
## non-zero exit status
```

```r
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.1.3
```

```r
train$Purchase <- as.factor(train$Purchase)
test$Purchase <- as.factor(test$Purchase)

svc_model <- svm(Purchase ~ ., data = train, kernel = "linear", cost = 0.01)


summary(svc_model)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  432
##
##  ( 215 217 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

c)

```r
train_pred <- predict(svc_model, train)
test_pred <- predict(svc_model, test)


train_error <- mean(train_pred != train$Purchase)
test_error <- mean(test_pred != test$Purchase)

train_error
```

```
## [1] 0.17125
```

```
test_error
```

```
## [1] 0.162963
```

d)

```
tune_result <- tune(svm, Purchase ~ ., data = train, kernel = "linear",
                    ranges = list(cost = seq(0.01, 10, length.out = 10)))


best_cost <- tune_result$best.parameters$cost
best_cost
```

```
## [1] 1.12
```

e)

```
optimal_svc_model <- svm(Purchase ~ ., data = train, kernel = "linear", cost = best_cost)


optimal_train_pred <- predict(optimal_svc_model, train)
optimal_test_pred <- predict(optimal_svc_model, test)

optimal_train_error <- mean(optimal_train_pred != train$Purchase)
optimal_test_error <- mean(optimal_test_pred != test$Purchase)

optimal_train_error
```

```
## [1] 0.1675
```

```
optimal_test_error
```

```
## [1] 0.1666667
```

f)

```
tune_radial <- tune(svm, Purchase ~ ., data = train, kernel = "radial",
                    ranges = list(cost = seq(0.01, 10, length.out = 10)))


best_cost_radial <- tune_radial$best.parameters$cost


radial_model <- svm(Purchase ~ ., data = train, kernel = "radial", cost = best_cost_radial)


radial_train_pred <- predict(radial_model, train)
radial_test_pred <- predict(radial_model, test)

radial_train_error <- mean(radial_train_pred != train$Purchase)
radial_test_error <- mean(radial_test_pred != test$Purchase)

radial_train_error
```

```
## [1] 0.1475
```

```r
radial_test_error
```

```
## [1] 0.1518519
```

g)

```r
tune_poly <- tune(svm, Purchase ~ ., data = train, kernel = "polynomial", degree = 2,
                  ranges = list(cost = seq(0.01, 10, length.out = 10)))


best_cost_poly <- tune_poly$best.parameters$cost


poly_model <- svm(Purchase ~ ., data = train, kernel = "polynomial", degree = 2, cost = best_cost_poly)


poly_train_pred <- predict(poly_model, train)
poly_test_pred <- predict(poly_model, test)

poly_train_error <- mean(poly_train_pred != train$Purchase)
poly_test_error <- mean(poly_test_pred != test$Purchase)

poly_train_error
```

```
## [1] 0.145
```

```r
poly_test_error
```

```
## [1] 0.1703704
```

##h)

```r
error_rates <- data.frame(
  Method = c("Linear SVC", "Radial SVM", "Polynomial SVM"),
  Training_Error = c(optimal_train_error, radial_train_error, poly_train_error),
  Test_Error = c(optimal_test_error, radial_test_error, poly_test_error)
)

print(error_rates)
```

```
##           Method Training_Error Test_Error
## 1     Linear SVC         0.1675  0.1666667
## 2     Radial SVM         0.1475  0.1518519
## 3 Polynomial SVM         0.1450  0.1703704
```

The Radial SVM approach gives the best results as it has the lowest test error (0.1519) and balances training and test performance effectively.