DS-6030 Homework Module 7

Matt Scheffel

DS 6030 | Spring 2022 | University of Virginia

8. In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable.

Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

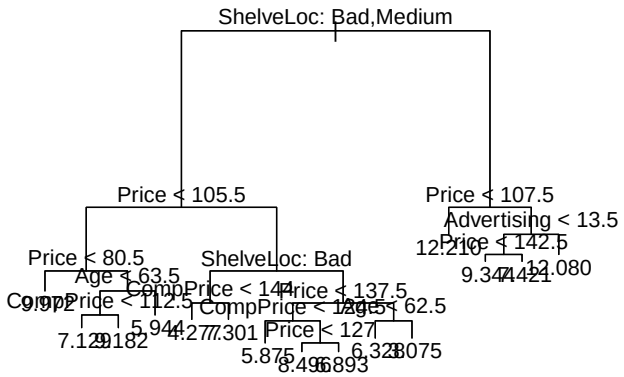
```
# packages

#install.packages("tree")
#install.packages("randomForest")

library("tree")
library("ISLR2")
attach(Carseats)
library(randomForest)
library(caret)
```

(a) Split the data set into a training set and a test set.

```
set.seed(123)

train <- createDataPartition(Carseats$Sales, p = 0.7, list = FALSE)

# training and test sets

car_train <- Carseats[train, ]
car_test <- Carseats[-train, ]
```

(b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

```
car_tree <- tree(Sales ~ ., data = car_train)

plot(car_tree)

text(car_tree, pretty=0)
```




```
summary(car_tree)
```

```
#>
#> Regression tree:
#> tree(formula = Sales ~ ., data = car_train)
#> Variables actually used in tree construction:
#> [1] "ShelveLoc"   "Price"       "Age"         "CompPrice"   "Advertising"
#> Number of terminal nodes:  15
#> Residual mean deviance:  2.379 = 632.8 / 266
#> Distribution of residuals:
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> -4.13900 -1.01800 -0.02316  0.00000  1.02600  3.51500
```

```
car_pred <- predict(car_tree, newdata = car_test)

# MSE
mean((car_pred - car_test$Sales)^2)
```

```
#> [1] 4.638469
```

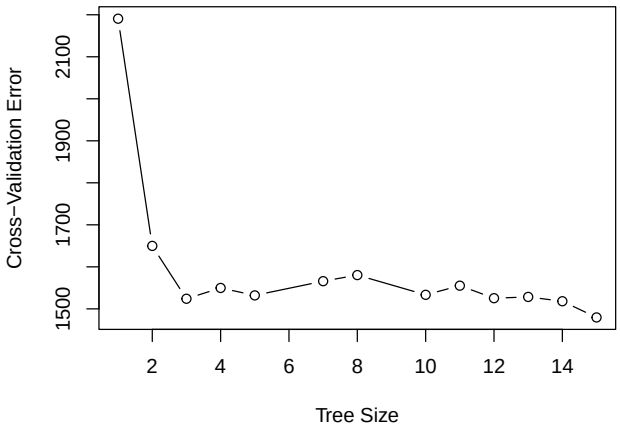
The MSE is 4.638469. This indicates that the regression tree model has a moderate level of prediction error when applied to the test set.

The residual mean deviance of the tree is 2.379, which indicates that the model has an average squared difference of 2.379 between the predicted and true sales values.

The median value of -0.02316 indicates that the model is able to predict sales accurately for half of the observations in the test set, while the mean value of 0 indicates that the model has no bias towards over- or under-predicting sales. The range of residuals (from -4.139 to 3.515) indicates that the model has some outliers or extreme values that are not well predicted by the tree.

(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

```
# fit regression tree using cross-validation to determine optimal tree complexity
library(tree)

# fit regression tree
tree_fit <- tree(Sales ~ ., data = car_train)

# perform cross-validation to determine optimal tree size
car_CV <- cv.tree(tree_fit)

# plot cross-validation error rate vs tree size
plot(car_CV$size, car_CV$dev, type = 'b', xlab = 'Tree Size', ylab = 'Cross-Validation Error')
```




```
# determine optimal tree size
optimal_size <- which.min(car_CV$dev)
print(paste("Optimal Tree Size:", optimal_size))
```

```
#> [1] "Optimal Tree Size: 1"
```

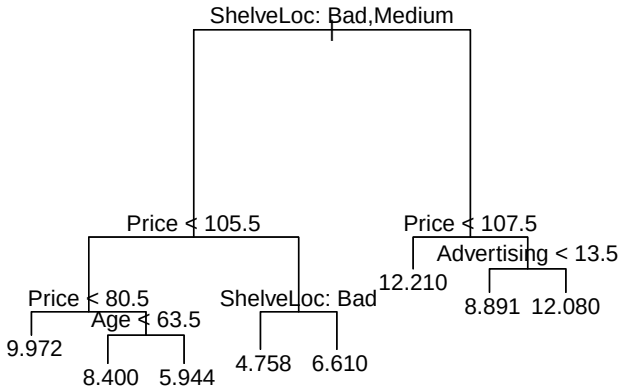
```
# prune tree using optimal size
pruned_tree_fit <- prune.tree(tree_fit, best = optimal_size)

# check if pruned tree has only one node
if (nrow(pruned_tree_fit$frame) == 1) {
  # predict mean value of response variable for test data
  pruned_pred <- mean(car_train$Sales)
} else {
  # predict sales on test set using pruned tree
  pruned_pred <- predict(pruned_tree_fit, newdata = car_test[, colnames(car_train)[-1]])
}

# calculate test MSE for pruned tree
pruned_test_mse <- mean((pruned_pred - car_test$Sales)^2)
print(paste("Pruned Tree Test MSE:", pruned_test_mse))
```

```
#> [1] "Pruned Tree Test MSE: 8.58125782536619"
```

```
# predict sales on test set using unpruned tree
tree_pred <- predict(tree_fit, newdata = car_test[, colnames(car_train)[-1]])

# calculate test MSE for unpruned tree
tree_test_mse <- mean((tree_pred - car_test$Sales)^2)
print(paste("Unpruned Tree Test MSE:", tree_test_mse))
```

```
#> [1] "Unpruned Tree Test MSE: 4.63846942159805"
```

```
par(mfrow = c(1,1))

# chose 8 as the best tree size
# 1 is too small despite being the minimum

prune_car <- prune.tree(car_tree, best = 8)
plot(prune_car)
text(prune_car, pretty = 0)
```




```
prune_car_prediction <- predict(prune_car, newdata = car_test)
mean((prune_car_prediction - car_test$Sales)^2)
```

```
#> [1] 4.526837
```

Pruning the tree made the MSE worse in thisd scenario - jumping up to over 8.

(d) Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the `importance()` function to determine which variables are most important.

```
set.seed(123)

car_bagging <- randomForest(Sales ~ ., data = Carseats, subset = train, mtry = 10, importance = TRUE)

car_bagging
```

```
#>
#> Call:
#>  randomForest(formula = Sales ~ ., data = Carseats, mtry = 10,      importance = TRUE, subset = trai
#>                Type of random forest: regression
#>                      Number of trees: 500
#> No. of variables tried at each split: 10
#>
#>           Mean of squared residuals: 2.589881
#>                     % Var explained: 66.32
```

```
car_bagging_prediction <- predict(car_bagging, newdata = car_test)
mean((car_bagging_prediction - car_test$Sales)^2)
```

```
#> [1] 2.506079
```

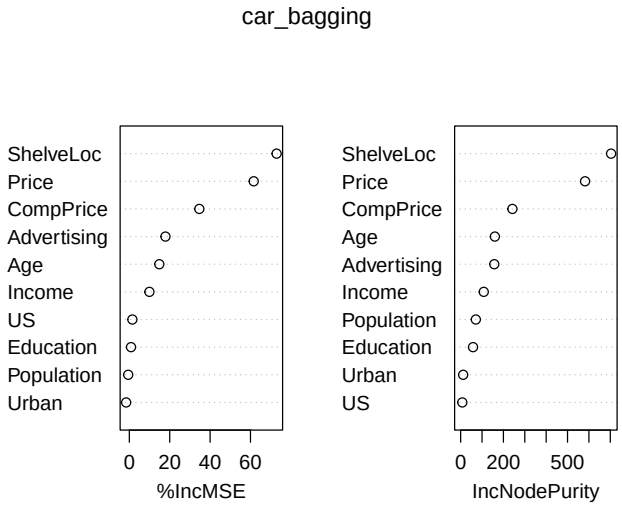
Using the bagging approach, the test MSE is 2.506079. This is lower than both the pruned tree and the original tree. This makes sense since the bagging method is designed to have lower bias and variance. It achieves this through the combination of several trees into one procedure.

```
# importance
importance(car_bagging)
```

```
#>                %IncMSE IncNodePurity
#> CompPrice   34.6595673    242.107047
#> Income       9.9448102    108.046743
#> Advertising 17.8763948    157.349629
#> Population  -0.5597694     71.053037
#> Price       61.6039899    582.003597
#> ShelveLoc   72.9312616    703.680594
```

```
#> Age          14.8535269    160.111011
#> Education     0.8330627     57.949935
#> Urban        -1.5717202     11.715601
#> US            1.5008168      7.678421
```

```
# plot
varImpPlot(car_bagging)
```



From the importance function and plot, we can see that shelf location and price are the most important predictors of how well a car seat sells. Competitor price, age, and advertising budget also have reasonable impacts, with the remaining variables holding less importance.

(e) Use random forests to analyze this data. What test MSE do you obtain? Use the `importance()` function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

```
set.seed(123)

random_forest_car_1 <- randomForest(Sales ~ ., data = Carseats, subset = train, mtry = 1, importance = T

random_forest_prediction_1 <- predict(random_forest_car_1, newdata = car_test)
mean((random_forest_prediction_1 - car_test$Sales)^2)
```

```
#> [1] 4.97386
```

We obtain an MSE of 4.97386 for m = 1.

```
set.seed(123)

random_forest_car_2 <- randomForest(Sales ~ ., data = Carseats, subset = train, mtry = 2, importance = T

random_forest_prediction_2 <- predict(random_forest_car_2, newdata = car_test)
mean((random_forest_prediction_2 - car_test$Sales)^2)
```

```
#> [1] 3.467817
```

We obtain an MSE of 3.467817 for m = 2.

```
set.seed(123)

random_forest_car_ <- randomForest(Sales ~ ., data = Carseats, subset = train, mtry = 3, importance = TR
```

```
random_forest_prediction_ <- predict(random_forest_car_, newdata = car_test)
mean((random_forest_prediction_ - car_test$Sales)^2)
```

```
#> [1] 2.84
```

We obtain an MSE of 2.84 for m = 3.

```
set.seed(123)

random_forest_car_4 <- randomForest(Sales ~ ., data = Carseats, subset = train, mtry = 4, importance =

random_forest_prediction_4 <- predict(random_forest_car_4, newdata = car_test)
mean((random_forest_prediction_4 - car_test$Sales)^2)
```

```
#> [1] 2.682774
```

We obtain an MSE of 2.682774 for m = 4.

The MSE continues declining as m increases towards 10.

```
set.seed(123)
random_forest_car_9 <- randomForest(Sales ~ ., data = Carseats, subset = train, mtry = 9, importance =

random_forest_prediction_9 <-predict(random_forest_car_9, newdata = car_test)
mean((random_forest_prediction_9 - car_test$Sales)^2)
```

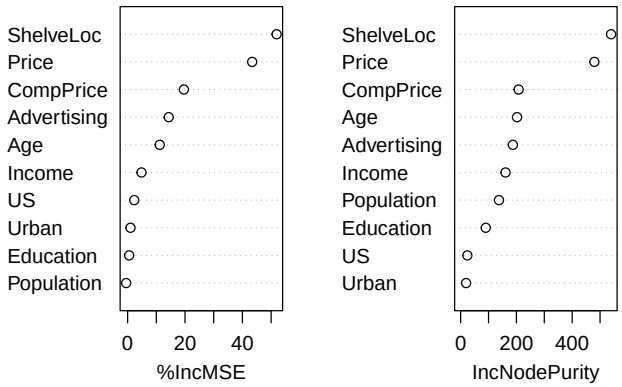
```
#> [1] 2.537588
```

We obtain an MSE of 2.537588 for m = 9.

```
importance(random_forest_car_)
```

```
#>                %IncMSE IncNodePurity
#> CompPrice   19.6139236     207.96561
#> Income       4.8313513     160.96490
#> Advertising 14.3174202     187.29802
#> Population  -0.5090991     137.49860
#> Price       43.4018716     479.67742
#> ShelveLoc   51.9073084     539.57769
#> Age         11.1786003     202.26214
#> Education    0.5260532      90.08154
#> Urban        0.9947562      19.54940
#> US           2.3111097      24.11825
```

```
varImpPlot(random_forest_car_)
```



For the random forests model, it appears that shelf location and price are again the most important predictors, but age and advertising also to have some importance again. Competitor prices appear to have less of an impact than they did in the bagging model. Income also appears to have more of an impact in the random forests model than it did in the bagging model.

(f) Now analyze the data using BART, and report your results. (skip this exercise)

skip

9. This problem involves the OJ data set which is part of the ISLR package.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
set.seed(123)

# index for the training set (800 observations)
train_index <- sample(nrow(OJ), 800)

# training set and test set
train_data <- OJ[train_index, ]
test_data <- OJ[-train_index, ]
```

(b) Fit a tree to the training data, with Purchase as the response and the other variables as predictors. Use the `summary()` function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

```
# fit the tree
tree_fit <- tree(Purchase ~ ., data = train_data)

summary(tree_fit)
```

```
#>
#> Classification tree:
#> tree(formula = Purchase ~ ., data = train_data)
#> Variables actually used in tree construction:
#> [1] "LoyalCH"   "PriceDiff"
```

```
#> Number of terminal nodes:  8
#> Residual mean deviance:  0.7625 = 603.9 / 792
#> Misclassification error rate: 0.165 = 132 / 800
```

The training error rate is 16.5%.

The tree has 8 terminal nodes.

A residual mean deviance of 0.7625 indicates that there is still a substantial amount of unexplained variation in the data.

(c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

```
tree_fit
```

```
#> node), split, n, deviance, yval, (yprob)
#>       * denotes terminal node
#>
#>  1) root 800 1071.00 CH ( 0.60875 0.39125 )
#>    2) LoyalCH < 0.5036 350  415.10 MM ( 0.28000 0.72000 )
#>      4) LoyalCH < 0.276142 170  131.00 MM ( 0.12941 0.87059 )
#>        8) LoyalCH < 0.0356415 56   10.03 MM ( 0.01786 0.98214 ) *
#>        9) LoyalCH > 0.0356415 114  108.90 MM ( 0.18421 0.81579 ) *
#>      5) LoyalCH > 0.276142 180  245.20 MM ( 0.42222 0.57778 )
#>       10) PriceDiff < 0.05 74   74.61 MM ( 0.20270 0.79730 ) *
#>       11) PriceDiff > 0.05 106  144.50 CH ( 0.57547 0.42453 ) *
#>    3) LoyalCH > 0.5036 450  357.10 CH ( 0.86444 0.13556 )
#>      6) PriceDiff < -0.39 27   32.82 MM ( 0.29630 0.70370 ) *
#>      7) PriceDiff > -0.39 423  273.70 CH ( 0.90071 0.09929 )
#>       14) LoyalCH < 0.705326 130  135.50 CH ( 0.78462 0.21538 )
#>         28) PriceDiff < 0.145 43   58.47 CH ( 0.58140 0.41860 ) *
#>         29) PriceDiff > 0.145 87   62.07 CH ( 0.88506 0.11494 ) *
#>       15) LoyalCH > 0.705326 293  112.50 CH ( 0.95222 0.04778 ) *
```

Selected terminal node: 10) PriceDiff $<$ 0.05 74 74.61 MM (0.20270 0.79730) *

Interpretation:

PriceDiff $<$ 0.05: This is the split rule that determines which observations end up in this terminal node. Specifically, it means that the node corresponds to a group of observations where the absolute difference between the regular price and the actual price of the orange juice is less than 0.05 units.

74: This is the number of observations that fall into this terminal node.

74.61: This is the mean value of the response variable (the purchase outcome) for the 74 observations in this node.

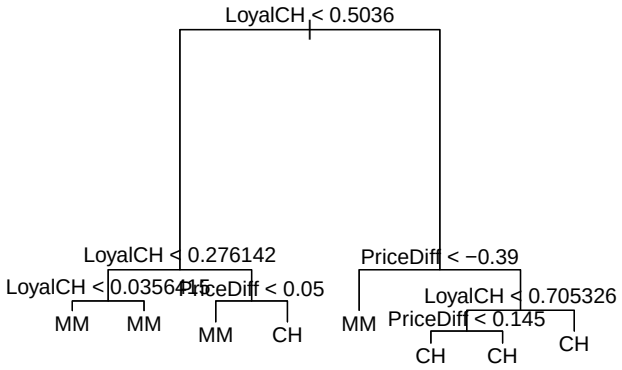
MM: This is the most common class label in this node (the purchase outcome that appears most frequently).

(0.20270 0.79730): This is the proportion of observations in this node that belong to each of the two classes. In this case, 20.27% of the 74 observations in this node purchased the brand other than "MM", while 79.73% of the 74 observations in this node purchased "MM".

Overall, this terminal node suggests that for customers who have a small absolute difference between the regular and actual price of the orange juice (less than 0.05 units), the majority of them tend to purchase the "MM" brand. The node also provides information about the proportion of observations that belong to each of the two classes (other than "MM" and "MM") in this group.

(d) Create a plot of the tree, and interpret the results.

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



In the context of the OJ data set, the tree plot shows how different variables such as price, advertising, and price difference between regular and actual price, affect the purchase behavior of customers. The plot suggests that the most important factor for predicting purchase behavior is the price of the juice, followed by the advertising budget.

(e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```
# predict
test_pred <- predict(tree_fit, newdata = test_data, type = "class")

# confusion matrix
conf_matrix <- table(test_data$Purchase, test_pred)
conf_matrix
```

```
#>     test_pred
#>       CH  MM
#>   CH 150  16
#>   MM  34  70
```

```
test_error <- (conf_matrix[1,2] + conf_matrix[2,1]) / sum(conf_matrix)
test_error
```

```
#> [1] 0.1851852
```

The test error rate is 18.5%.

(f) Apply the `cv.tree()` function to the training set in order to determine the optimal tree size.

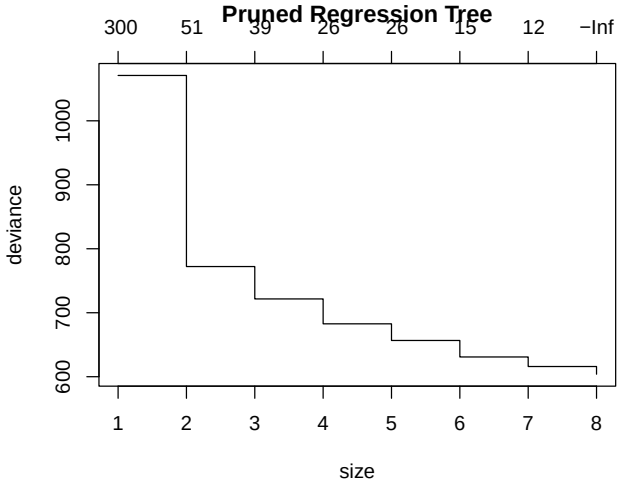
```
# cross-validation
cv_fit <- cv.tree(tree_fit)

cv_fit
```

```
#> $size
#> [1] 8 7 6 5 4 3 2 1
#>
#> $dev
#> [1]  690.2346  692.8691  683.0423  717.4446  717.4446  743.3604  793.1266
#> [8] 1073.0916
#>
#> $k
#> [1]      -Inf  12.03823  14.92474  25.76707  26.02613  38.91686  50.61655
```

```
#> [8] 298.68751
#>
#> $method
#> [1] "deviance"
#>
#> attr(,"class")
#> [1] "prune"         "tree.sequence"
```

```
# prune the tree using the optimal cp value
pruned_tree_fit <- prune.tree(tree_fit, best = cv_fit$cp[which.min(cv_fit$dev)])
```

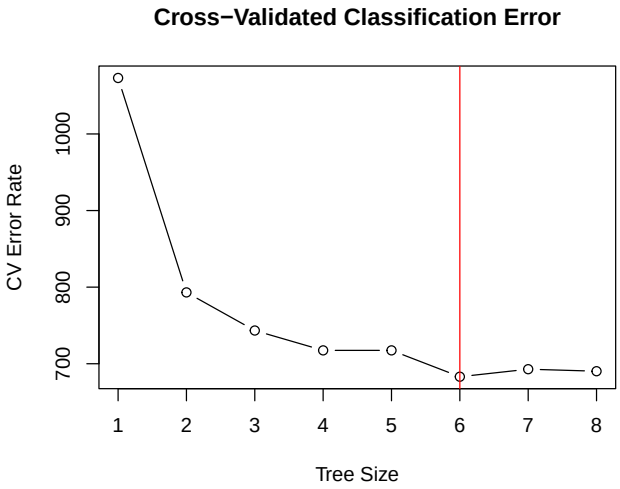
```
# plot

plot(pruned_tree_fit, main = "Pruned Regression Tree")
```



The optimal tree size appears to be around 8.

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```
# extract the cross-validation results
cv_results <- data.frame(size = cv_fit$size, error_rate = cv_fit$dev)

# plot the cross-validation error rate by tree size
plot(cv_results$size, cv_results$error_rate, type = "b",
     xlab = "Tree Size", ylab = "CV Error Rate",
     main = "Cross-Validated Classification Error")

# add vertical line at optimal tree size
abline(v = cv_fit$size[which.min(cv_fit$dev)], col = "red")
```



(h) Which tree size corresponds to the lowest cross-validated classification error rate?

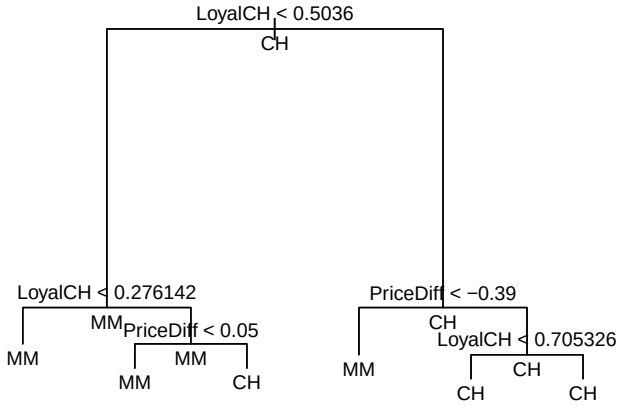
```
# extract the optimal tree size
optimal_size <- cv_results$size[which.min(cv_results$error_rate)]

# print the optimal tree size
print(optimal_size)
```

```
#> [1] 6
```

A tree size of 6 corresponds to the lowest cross-validated classification error rate.

(i) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```
# prune the tree using the optimal cp value
pruned_tree_fit <- prune.tree(tree_fit, best = optimal_size)

# plot the pruned tree
plot(pruned_tree_fit)
text(pruned_tree_fit, all = TRUE, cex = 0.8)
```



(j) Compare the training error rates between the pruned and unpruned trees. Which is higher?

```
# predict class labels for unpruned tree on training set
unpruned_train_pred <- predict(tree_fit, type = "class")
```

```r
# generate confusion matrix for unpruned tree on training set
unpruned_train_conf <- table(train_data$Purchase, unpruned_train_pred)
print(unpruned_train_conf)
```

```
#>     unpruned_train_pred
#>       CH  MM
#>   CH 442  45
#>   MM  87 226
```

```r
# calculate training error rate for unpruned tree
unpruned_train_error <- 1 - sum(diag(unpruned_train_conf)) / sum(unpruned_train_conf)
print(paste("Unpruned tree training error rate:", unpruned_train_error))
```

```
#> [1] "Unpruned tree training error rate: 0.165"
```

```r
# predict class labels for pruned tree on training set
pruned_train_pred <- predict(pruned_tree_fit, type = "class")

# generate confusion matrix for pruned tree on training set
pruned_train_conf <- table(train_data$Purchase, pruned_train_pred)
print(pruned_train_conf)
```

```
#>     pruned_train_pred
#>       CH  MM
#>   CH 442  45
#>   MM  87 226
```

```r
# calculate training error rate for pruned tree
pruned_train_error <- 1 - sum(diag(pruned_train_conf)) / sum(pruned_train_conf)
print(paste("Pruned tree training error rate:", pruned_train_error))
```

```
#> [1] "Pruned tree training error rate: 0.165"
```

The error rates appear to be the same.

(k) Compare the test error rates between the pruned and unpruned trees. Which is higher?

```r
# predict class labels for unpruned tree on test set
unpruned_test_pred <- predict(tree_fit, newdata = test_data, type = "class")

# generate confusion matrix for unpruned tree on test set
unpruned_test_conf <- table(test_data$Purchase, unpruned_test_pred)
print(unpruned_test_conf)
```

```
#>     unpruned_test_pred
#>       CH  MM
#>   CH 150  16
#>   MM  34  70
```

```r
# calculate test error rate for unpruned tree
unpruned_test_error <- 1 - sum(diag(unpruned_test_conf)) / sum(unpruned_test_conf)
print(paste("Unpruned tree test error rate:", unpruned_test_error))
```

```
#> [1] "Unpruned tree test error rate: 0.185185185185185"
```

```r
# predict class labels for pruned tree on test set
pruned_test_pred <- predict(pruned_tree_fit, newdata = test_data, type = "class")

# generate confusion matrix for pruned tree on test set
```

```
pruned_test_conf <- table(test_data$Purchase, pruned_test_pred)
print(pruned_test_conf)
```

```
#>     pruned_test_pred
#>       CH  MM
#>   CH 150  16
#>   MM  34  70
```

```
# calculate test error rate for pruned tree
pruned_test_error <- 1 - sum(diag(pruned_test_conf)) / sum(pruned_test_conf)
print(paste("Pruned tree test error rate:", pruned_test_error))
```

```
#> [1] "Pruned tree test error rate: 0.185185185185185"
```

The error rates appear to be the same.