# HW 8

Jared Brotamonte

11/26/2023

## Exercise 1

### (a)

```
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.1.3
```

```
# Load Carseats dataset
data(Carseats)

# Set seed for reproducibility
set.seed(123)

# Generate indices for the training set (80%)
train_indices <- sample(1:nrow(Carseats), 0.8 * nrow(Carseats))

# Create the training set
Carseats_train <- Carseats[train_indices, ]

# Create the validation set
Carseats_valid <- Carseats[-train_indices, ]

# Print the dimensions of the training and validation sets
cat("Training set dimensions:", dim(Carseats_train), "\n")
```

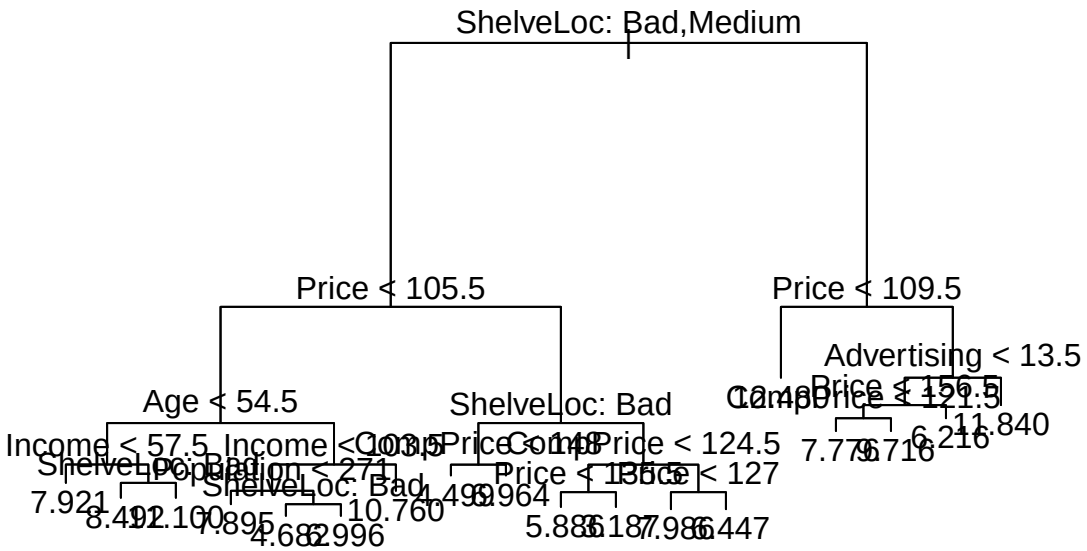
```
## Training set dimensions: 320 11
```

```
cat("Validation set dimensions:", dim(Carseats_valid), "\n")
```

```
## Validation set dimensions: 80 11
```

### (b)

```
# Fit a regression tree to the training set
tree_fit <- tree(Sales ~ ., data = Carseats_train)

# Plot the tree
plot(tree_fit)
text(tree_fit, pretty = 0)
```



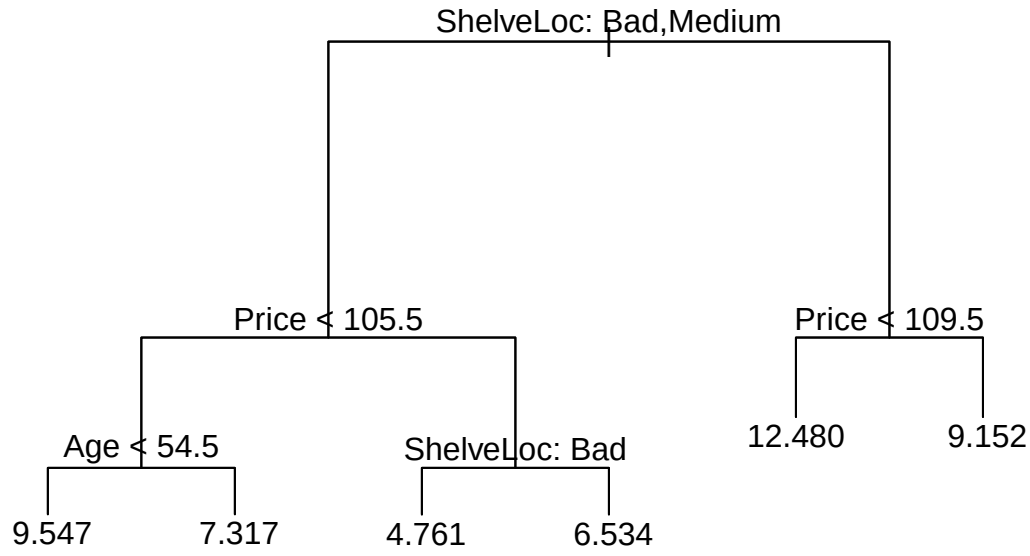

```
# Make predictions on the validation set
tree_preds <- predict(tree_fit, newdata = Carseats_valid)

# Calculate the validation MSE
validation_mse <- mean((Carseats_valid$Sales - tree_preds)^2)

# Print the validation MSE
cat("Validation MSE:", validation_mse, "\n")
```

```
## Validation MSE: 4.1593
```

## (c)

```
# Define the training control for cross-validation
ctrl <- trainControl(method = "cv", number = 10)
```

```r
# Fit a regression tree using cross-validation
cv_tree <- tree(Sales ~ ., data = Carseats_train)

# Prune the tree based on the optimal complexity parameter
pruned_tree <- prune.tree(cv_tree, best = cv_tree$frame$dev[which.min(cv_tree$frame$dev)])

# Plot the pruned tree
plot(pruned_tree)
text(pruned_tree, pretty = 0)
```




```r
# Make predictions on the validation set using the pruned tree
pruned_preds <- predict(pruned_tree, newdata = Carseats_valid)

# Calculate the validation MSE for the pruned tree
pruned_validation_mse <- mean((Carseats_valid$Sales - pruned_preds)^2)

# Print the validation MSE for the pruned tree
cat("Pruned Tree Validation MSE:", pruned_validation_mse, "\n")
```

```
## Pruned Tree Validation MSE: 4.349257
```

Pruning the tree did not improve the MSE.

## (d)

```
# Set seed for reproducibility
set.seed(123)

# Fit a random forest with 500 trees
rf_model <- randomForest(Sales ~ ., data = Carseats_train, ntree = 500)

# Make predictions on the validation set
rf_preds <- predict(rf_model, newdata = Carseats_valid)

# Calculate the validation MSE
rf_validation_mse <- mean((Carseats_valid$Sales - rf_preds)^2)

# Print the validation MSE
cat("Random Forest Validation MSE:", rf_validation_mse, "\n")
```

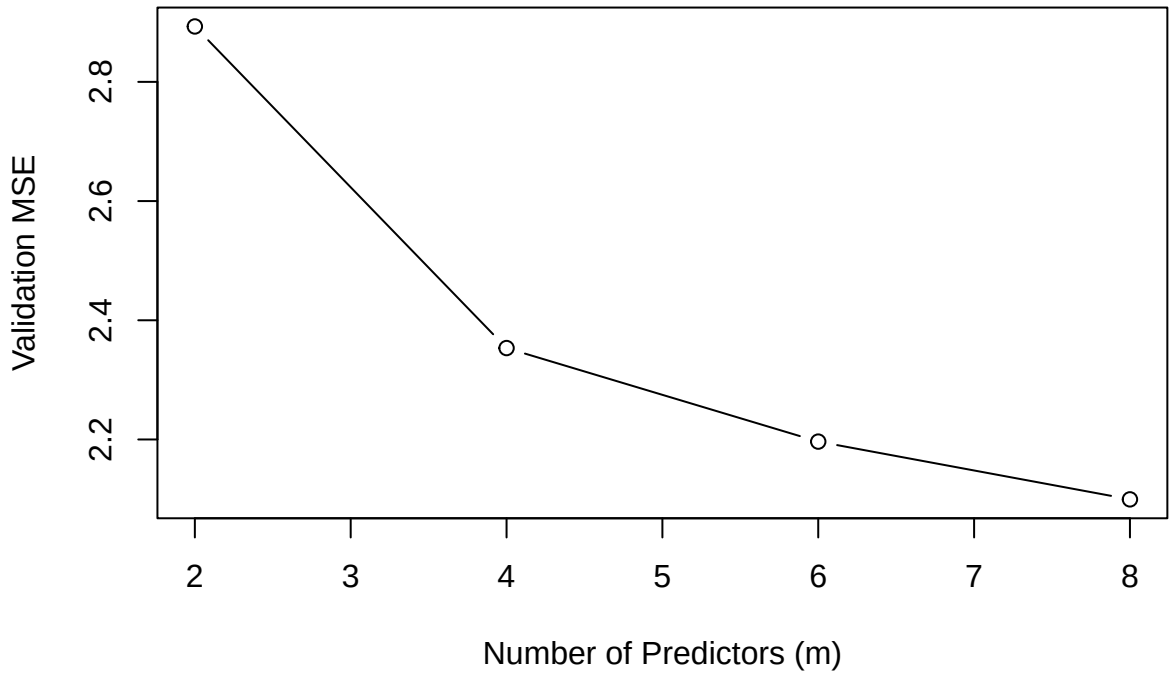
```
## Random Forest Validation MSE: 2.61238
```

## (e)

```
# Set seed for reproducibility
set.seed(123)

# Define a range of values for m
m_values <- c(2, 4, 6, 8)  # You can adjust this range

# Initialize a vector to store validation MSE for each m
mse_values <- numeric(length(m_values))

# Fit random forest models with different values of m
for (i in seq_along(m_values)) {
  m <- m_values[i]
  rf_model <- randomForest(Sales ~ ., data = Carseats_train, ntree = 500, mtry = m)

  # Make predictions on the validation set
  rf_preds <- predict(rf_model, newdata = Carseats_valid)

  # Calculate the validation MSE
  mse_values[i] <- mean((Carseats_valid$Sales - rf_preds)^2)

  # Print the results
  cat("Random Forest Validation MSE with m =", m, ":", mse_values[i], "\n")
}
```

```
## Random Forest Validation MSE with m = 2 : 2.892953
## Random Forest Validation MSE with m = 4 : 2.353349
## Random Forest Validation MSE with m = 6 : 2.196422
## Random Forest Validation MSE with m = 8 : 2.099507
```

```r
# Plot the results
plot(m_values, mse_values, type = "b", xlab = "Number of Predictors (m)", ylab = "Validation MSE")
```




As seen in the graph the Validation MSE improves as the number of predictors 'm' increase

## Exercise 2

### (a)

```r
# Load the OJ dataset
data(OJ)

# Set seed for reproducibility
set.seed(123)

# Create an index for sampling
index <- sample(1:nrow(OJ), 800)

# Create the training set
OJ_train <- OJ[index, ]

# Create the validation set
OJ_valid <- OJ[-index, ]
```

```r
# Print the dimensions of the training and validation sets
cat("Dimensions of Training Set:", dim(OJ_train), "\n")
```

```
## Dimensions of Training Set: 800 18
```

```r
cat("Dimensions of Validation Set:", dim(OJ_valid), "\n")
```

```
## Dimensions of Validation Set: 270 18
```

## (b)

```r
# Fit a tree to the training data
tree_model <- tree(Purchase ~ ., data = OJ_train)

# Display summary statistics about the tree
summary(tree_model)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ_train)
## 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
```

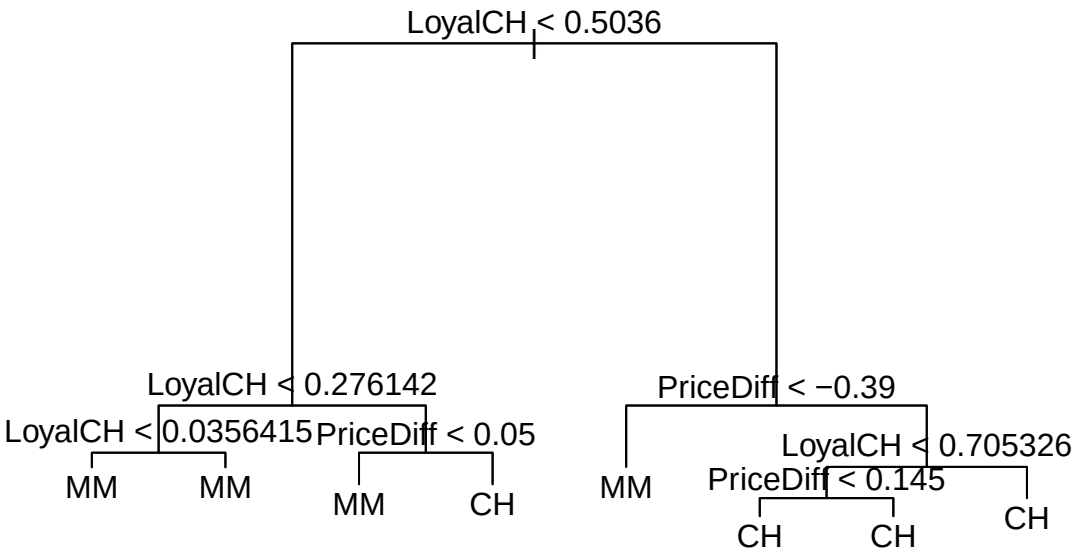
```r
# Cross-validate the tree to estimate error rate
cv_tree <- cv.tree(tree_model)
cat("Cross-Validated Error Rate:", cv_tree$dev[which.min(cv_tree$dev)] / length(OJ_train$Purchase), "\n
```

```
## Cross-Validated Error Rate: 0.8538028
```

The tree has 8 terminal nodes. The residual mean deviance describes how well the tree fits the data, in this case it's 0.7625 which means it fits decently well. A misclassification error rate of 0.1625 means that about 16.5 percent of the data is misclassified by the tree. Cross-Validated Error Rate is the expected error rate on new unseen data thus the expected error rate would be about 85.4%.

## (c)

```r
# Plot the tree
plot(tree_model)
text(tree_model, pretty = 0)
```




(d)

```
# Predict the response on the validation data
tree_preds <- predict(tree_model, newdata = OJ_valid, type = "class")

# Create a confusion matrix
conf_matrix <- table(tree_preds, OJ_valid$Purchase)

# Display the confusion matrix
cat("Confusion Matrix:\n", conf_matrix, "\n")
```

```
## Confusion Matrix:
##  150 16 34 70
```

```
# Calculate the test error rate
test_error_rate <- 1 - sum(diag(conf_matrix)) / sum(conf_matrix)
cat("Test Error Rate:", test_error_rate, "\n")
```

```
## Test Error Rate: 0.1851852
```

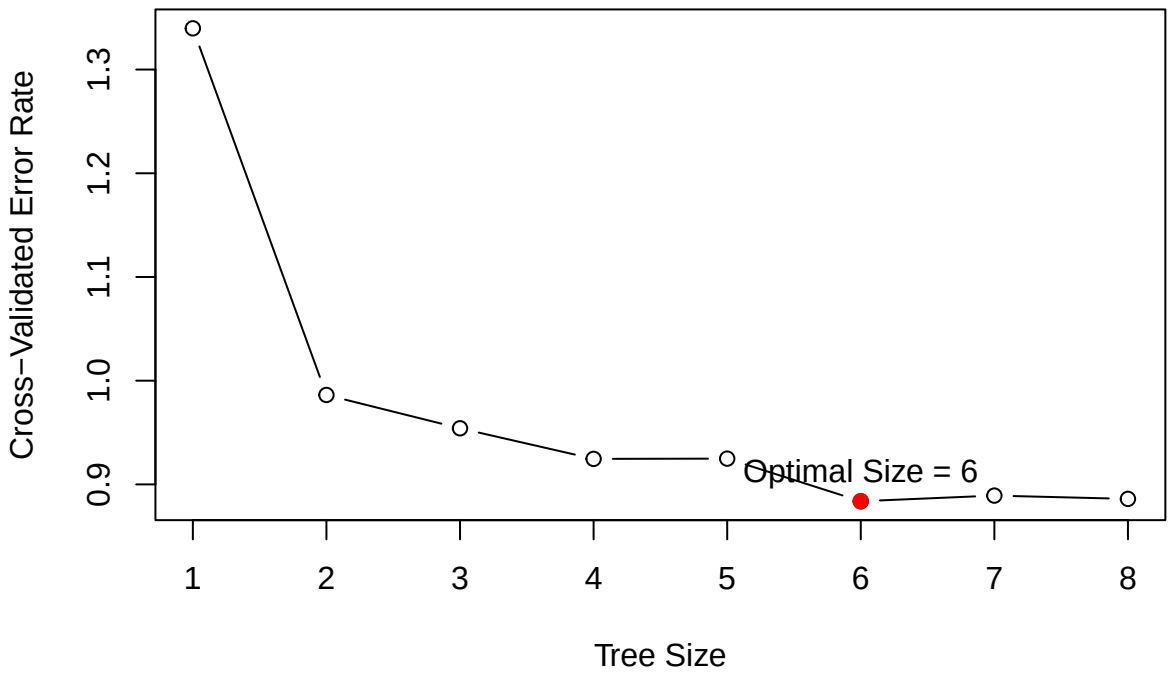
(e)

```
# Apply cv.tree() to determine the optimal tree size
cv_tree <- cv.tree(tree_model)

# Display the cross-validated results
print(cv_tree)
```

```
## $size
## [1] 8 7 6 5 4 3 2 1
##
## $dev
## [1]  708.8092  711.3580  707.0049  739.8715  739.6867  763.2801  788.9812
## [8] 1071.7839
##
## $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"
```

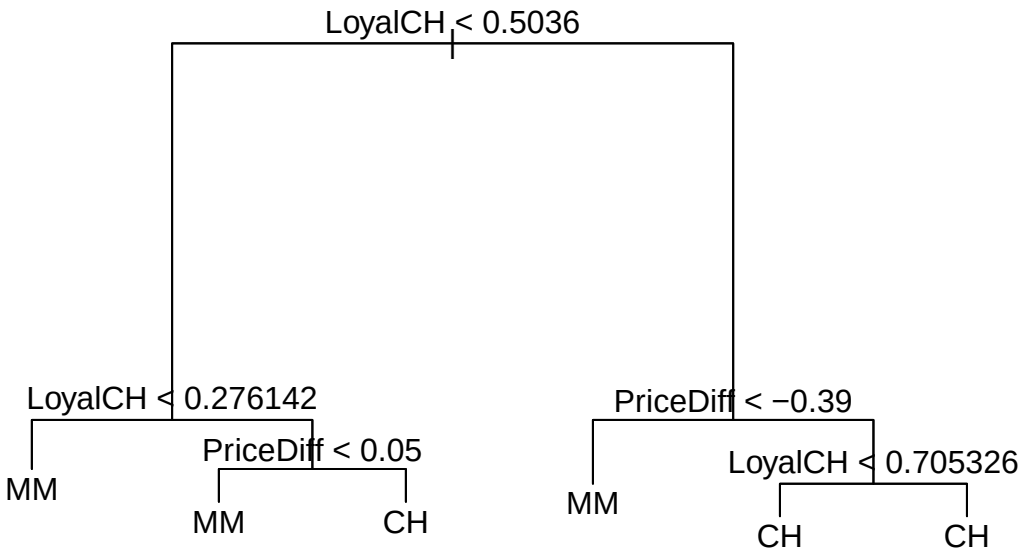
(f)

```
# Plot tree size vs. cross-validated error rate
plot(cv_tree$size, cv_tree$dev / length(OJ_train$Purchase), type = "b", xlab = "Tree Size", ylab = "Cros

# Identify the tree size with the lowest error rate
min_error_size <- cv_tree$size[which.min(cv_tree$dev)]
points(min_error_size, min(cv_tree$dev / length(OJ_train$Purchase)), col = "red", pch = 19)
text(min_error_size, min(cv_tree$dev / length(OJ_train$Purchase)), labels = paste("Optimal Size =", min_
```




(g)

```r
# Prune the tree based on the optimal complexity parameter obtained from cross-validation
pruned_tree <- prune.tree(tree_model, best = cv_tree$size[which.min(cv_tree$dev)])

# If no optimal complexity parameter is found, prune the tree to have five terminal nodes
if (is.null(pruned_tree)) {
  pruned_tree <- prune.tree(tree_model, best = 5)
}

# Plot the pruned tree
plot(pruned_tree)
text(pruned_tree, pretty = 0)
```




**(h)**

```
# Predict response on the training set using the pruned tree
pruned_preds <- predict(pruned_tree, newdata = OJ_train, type = "class")

# Create a confusion matrix for the pruned tree on the training set
conf_matrix_pruned <- table(pruned_preds, OJ_train$Purchase)

# Calculate the training error rate for the pruned tree
training_error_rate_pruned <- 1 - sum(diag(conf_matrix_pruned)) / sum(conf_matrix_pruned)
cat("Training Error Rate (Pruned Tree):", training_error_rate_pruned, "\n")
```

```
## Training Error Rate (Pruned Tree): 0.165
```

```
# Calculate the training error rate for the unpruned tree
tree_preds <- predict(tree_model, newdata = OJ_train, type = "class")
conf_matrix_unpruned <- table(tree_preds, OJ_train$Purchase)
training_error_rate_unpruned <- 1 - sum(diag(conf_matrix_unpruned)) / sum(conf_matrix_unpruned)
cat("Training Error Rate (Unpruned Tree):", training_error_rate_unpruned, "\n")
```

```
## Training Error Rate (Unpruned Tree): 0.165
```

(i)

```
# Predict response on the validation set using the pruned tree
pruned_preds_valid <- predict(pruned_tree, newdata = OJ_valid, type = "class")

# Create a confusion matrix for the pruned tree on the validation set
conf_matrix_pruned_valid <- table(pruned_preds_valid, OJ_valid$Purchase)

# Calculate the validation error rate for the pruned tree
validation_error_rate_pruned <- 1 - sum(diag(conf_matrix_pruned_valid)) / sum(conf_matrix_pruned_valid)
cat("Validation Error Rate (Pruned Tree):", validation_error_rate_pruned, "\n")
```

```
## Validation Error Rate (Pruned Tree): 0.1851852
```

```
# Calculate the validation error rate for the unpruned tree
tree_preds_valid <- predict(tree_model, newdata = OJ_valid, type = "class")
conf_matrix_unpruned_valid <- table(tree_preds_valid, OJ_valid$Purchase)
validation_error_rate_unpruned <- 1 - sum(diag(conf_matrix_unpruned_valid)) / sum(conf_matrix_unpruned_v
cat("Validation Error Rate (Unpruned Tree):", validation_error_rate_unpruned, "\n")
```

```
## Validation Error Rate (Unpruned Tree): 0.1851852
```