Problem_Set_3

Sief Salameh

5/4/2023

Chapter 6

Question 1 Part A:

Generally, models with more predictor variables (k predictors) tend to have a smaller training RSS because the increased flexibility of the model allows it to fit the training set more accurately. However, in many cases, models with too many predictors (p) can lead to overfitting, where the model fits the noise in the training data instead of the randomized patterns. Training data that has many (p) predictors does not generalize well to new data. Therefore, limiting the number of predictors can lead to a smaller training RSS because it increases the bias and decreases the variance.

Across all three models with k predictors, the best subset selection method yields the smallest training RSS because it analyzes all possible subsets of k-predictor models and selects the one with the smallest training RSS. However, it's important to acknowledge that this method is computationally expensive and time-consuming.

Question 1 Part B:

This question does not have a clear answer, and it is difficult to determine which three models with k predictors have the smallest test RSS without evaluating the specific test data. This is because the test RSS is computed by applying the predictors to new and unseen data, which will vary from one model to another.

Question 1 Part C i:

This Statement is True.

This is because, in the forward stepwise selection model, predictors are included one at a time until the model reaches a bias and variance tradeoff where the Mean Squared Error (MSE) is minimized and the model achieves the best fit by selecting the appropriate number of (p) predictors. Therefore, the predictors in the k-variable model identified by forward stepwise are indeed a subset of the predictors in the (k+1)-variable model. This occurs because if the k-variable model is identified first, then the (k+1)-variable model is identified by considering all possible predictors that were not included in the k-variable

model and selecting the best one based on the selection criteria. As the (k+1)-variable model analyzes all the predictors considered by the k-variable model, plus at least one more predictor, it is true that the predictors in the k-variable model are a subset of the predictors in the (k+1)-variable model.

Question 1 Part C ii:

This Statement is True.

This is because in the backward stepwise selection model, all the predictors are initially included in the model, and then one predictor is removed at a time until the model reaches a balance between bias and variance, minimizing the Mean Squared Error (MSE) and achieving the best fit by selecting the appropriate number of (p) predictors. Therefore, if a k-variable model is identified first, the (k+1)-variable model is chosen by analyzing all the possible predictors that were removed from the k-variable model, and selecting the best one based on the selection criteria set. Since the (k+1)-variable model considers all predictors within the k-variable model, plus at least one more predictor, it is true that the predictors in the k-variable model are a subset of the predictors in the (k+1)-variable model because the (k+1)-variable model may include some predictors that were not in the k-variable model but includes all the predictors in the k-variable model.

Question 1 Part C iii:

This Statement is False.

The predictors in the k-variable model identified by backward stepwise selection are not necessarily a subset of the predictors in the (k + 1)-variable model identified by forward stepwise selection. This is because, in the backward stepwise selection model, all predictors are initially included in the model, and then they are removed one at a time. In contrast, in forward stepwise selection, predictors are added one at a time to the model until the desired number of predictors is reached.

Therefore, it is highly unlikely that the predictors in the k-variable model identified by backward stepwise selection are a subset of the predictors in the (k + 1)-variable model identified by forward stepwise selection because these two methods start from different points. Backward stepwise selection starts from the full set of predictors, while forward stepwise selection starts from a single predictor and builds up. Thus, since backward stepwise selection starts with all predictors included, it is not possible for forward stepwise selection to have evaluated all predictors, as it is only measuring one predictor at a time.

Question 1 Part C iv:

This Statement is False.

The predictors in the k-variable model identified by forward stepwise are not necessarily a subset of the predictors in the (k+1)-variable model identified by backward stepwise selection. This is because, in the backward stepwise selection model, all predictors are initially included in the model, and then they are removed one at a time. In contrast, in forward stepwise selection, predictors are added one at a time to the model until the desired number of predictors is reached.

Therefore, it is highly unlikely that the predictors in the k-variable model identified by forward stepwise selection are a subset of the predictors in the (k + 1)-variable model identified by backward stepwise selection because these two methods start from different points. Backward stepwise selection starts from the full set of predictors, while forward stepwise selection starts from a single predictor and builds up. Thus, since backward stepwise selection starts with all predictors included, it is not possible for forward stepwise selection to have evaluated all predictors, as it is only measuring one predictor at a time and vice versa.

Question 1 Part C v:

This Statement is False.

The predictors identified in the k-variable model by best subset selection may not necessarily be a subset of the predictors in the (k + 1)-variable model identified by best subset selection.

This is because the best subset selection method determines the smallest training RSS by analyzing all possible subsets of k-predictor models and selecting the one with the smallest training RSS.

Therefore, while the method may evaluate the performance of many predictors, the final subset that is chosen will be a combination of all the different possible subsets that were analyzed by the method. Since this method is not nested, it is incorrect to assume that the predictors in the k-variable model identified by best subset selection are always a subset of the predictors in the (k + 1)-variable model identified by best subset selection.

Question 2 Part A:

The Lasso, relative to least squares, is iii: “Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.”

This is because the Lasso adds a regularization term to the standard linear regression model that penalizes the absolute value of the coefficients of the predictors. This encourages the model to select only the most significant predictors when measuring the relationship between the independent and dependent variables. The amount of regularization is controlled by the lambda term (λ), which determines the strength of the penalty.

The Lasso penalty is effective when our model has many predictors, and when we want to identify the ones that have the highest statistical significance in terms of impacting the dependent variable. By shrinking the coefficients of irrelevant predictors to zero, Lasso helps to simplify the model, address overfitting, and improve its generalization performance on unseen and test data. In general, the model's flexibility decreases as we increase the bias and decrease the variance. Hence, the Lasso penalty can help strike a balance between bias and variance and improve the model's predictive performance.

Question 2 Part B:

The Ridge Regression relative to least squares, is iii: "Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance."

Ridge Regression is similar to Lasso, but unlike Lasso, it does not necessarily force the coefficients to become zero. Instead, it forces them to get really close to zero. Ridge Regression is also a penalty term that is added to the loss function that the model minimizes. This penalty term is proportional to the square of the magnitude of the coefficients of the regression model. The goal of Ridge Regression is to select the values of the coefficients that minimize the sum of the squared errors of the model, subject to the constraint that the magnitude of the coefficients is minimized as well. This constraint helps prevent the coefficients from becoming too large, which in turn helps address overfitting.

The amount of regularization is controlled by a hyperparameter called the regularization strength or alpha. A higher value of alpha will lead to more regularization and smaller coefficients, while a lower value of alpha will lead to less regularization and larger coefficients. In general, the model's flexibility decreases as we increase the bias and decrease the variance. Hence, the Ridge Regression penalty can also help strike a balance between bias and variance and improve the model's predictive performance - identical to Lasso.

CITATION - https://dataaspirant.com/ridge-regression/

Question 2 Part C:

Non-linear methods relative to least squares, are ii: "More flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias."

Non-linear methods in machine learning are techniques that allow models to learn complex, non-linear relationships between independent and dependent variables. These methods are typically used when the relationship between inputs and outputs is too complex, possibly quadratic, and cannot be represented by a linear model. Thus, these models are highly flexible and demonstrate low bias and high variance.

Question 3 Part A:

As we increase s from 0, the training RSS will: "Steadily Decrease."

This is because, as we increase the value of s, the constraint on the absolute value of the coefficients becomes less restrictive. This is because we are moving from a constraint of being less than or equal to 0, to a constraint of being less than 1, then to less than 2, 3, and so on. Consequently, the range within which the coefficients can vary expands, allowing for larger-valued coefficients and more flexible models. The increased flexibility enables the model to fit the training data more precisely, which leads to a decrease in the residual sum of squares (RSS). As we increase the model's flexibility, we also reduce the bias and increase the variance.

Question 3 Part B:

As we increase s from 0, the test RSS will: "Decrease initially, and then eventually start increasing in a U shape."

As we increase the value of s, the constraint on the absolute value of the coefficients becomes less restrictive. Moving from a constraint of being less than or equal to 0, to being less than 1, then to less than 2, 3, and so on, expanding the range within which coefficients can vary. This allows for larger-valued coefficients and more flexible models, which can fit the test data more precisely and lead to a decrease in the residual sum of squares (RSS) initially.

However, as s continues to increase, the model becomes too flexible, leading to overfitting and an increase in the test RSS. Therefore, the test RSS is expected to decrease initially and then increase in a U-shape as s increases further. This does not usually occur in the training data because the model does not generalize well and is built on that specific data. Thus, the model learns to adapt and fit too well. However, since the test data is new and untrained to the model, the RSS will increase as we include too many predictors, capture noise, and overfit the model.

Question 3 Part C:

As we increase s from 0, the variance will: "Steadily Increase."

This is because as we increase the model's flexibility, the variance generally increases. The variance of a statistical model refers to the amount by which its predictions differentiate for contrasting training datasets. When a model is more flexible, it includes and measures more predictor variables or it fits more complex functions. This can potentially fit the training data more accurately and capture much more of the noise in the data. As a consequence, however, this can also cause the model to become overly sensitive to the specific training data it was fitted on, leading to high variance.

Question 3 Part D:

As we increase s from 0, the (squared) bias will: "Steadily Decrease."

As we increase a model's flexibility, its bias generally decreases. Bias is defined as the error that occurs from the simplifying assumptions made by a model. Therefore, by increasing the model's flexibility and adding more predictor variables, it can more accurately capture the underlying patterns in the relationship and result in a lower bias.

Question 8 Part A:

```
library(tidyverse)
```

```
## Warning: package 'ggplot2' was built under R version 4.1.2
```

```
## Warning: package 'tibble' was built under R version 4.1.2
```

```
## Warning: package 'tidyr' was built under R version 4.1.2
```

```
## Warning: package 'readr' was built under R version 4.1.2
```

```
## Warning: package 'purrr' was built under R version 4.1.2
```

```
## Warning: package 'dplyr' was built under R version 4.1.2
```

```
## Warning: package 'stringr' was built under R version 4.1.2
```

Use the rnorm() function to generate a predictor X of length n = 100, as well as a noise vector ϵ of length n = 100.

```
set.seed(1)

X <- rnorm(100)

e_term <- rnorm(100)
```

Question 8 Part B:

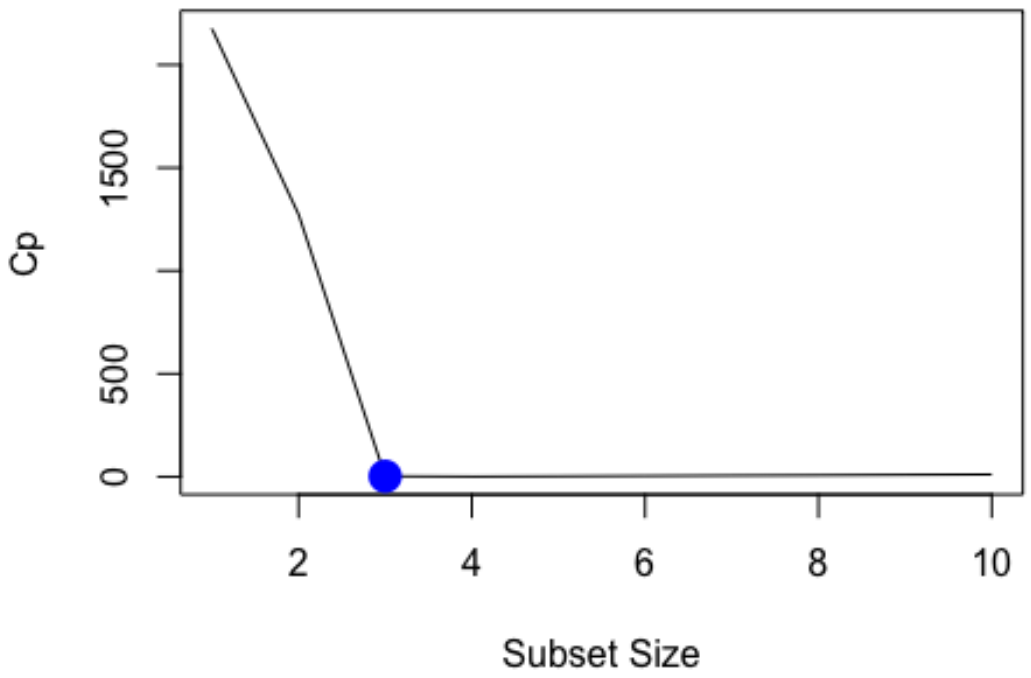
Generate a response vector Y of length n = 100 according to the model Y = β0 + β1X + β2X^2 + β3X^3 + ϵ, where β0, β1, β2, and β3 are constants of your choice

```
b0 <- 6
b1 <- 8
b2 <- 9
b3 <- (-3)

Y <- b0 + b1*X + b2*X^2 + b3*X^3 + e_term
```

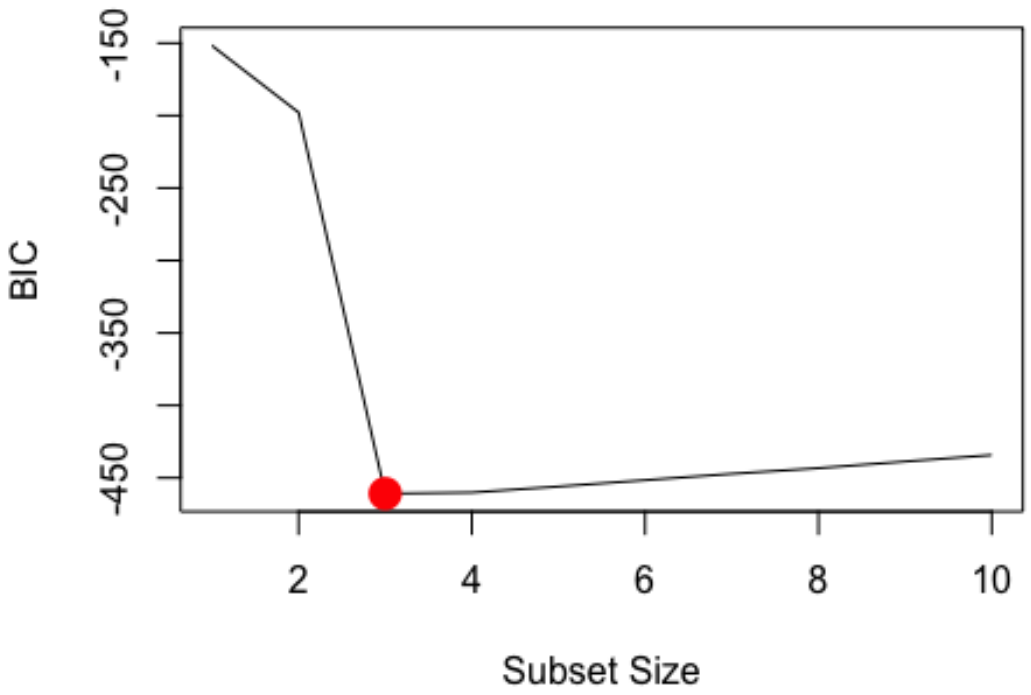
Question 8 Part C:

Use the regsubsets() function to perform best subset selection in order to choose the best model containing the predictors X, X2,…,X10. What is the best model obtained according to Cp, BIC, and adjusted R2? Show some plots to provide evidence for your answer, and report the coefficients of the best model obtained

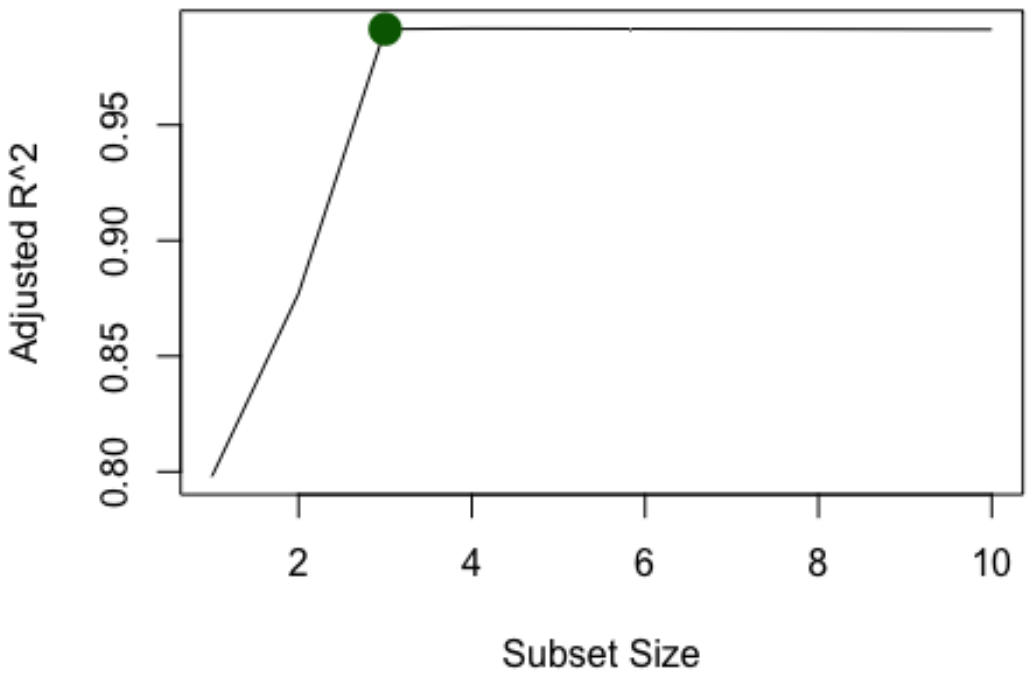
```
library(leaps)

full_df <- data.frame(y = Y, x = X)

full_model <- regsubsets(y ~ poly(x, 10, raw = T), data = full_df, nvmax =
10)

model_summary <- summary(full_model)

# Identifying the best model size according to Cp, BIC, and adjusted R^2

which.min(model_summary$cp)

## [1] 4

which.min(model_summary$bic)

## [1] 3

which.max(model_summary$adjr2)

## [1] 4

plot(model_summary$cp,
  xlab = "Subset Size",
  ylab = "Cp", pch = 16,
  type = "l"
)
points(3, model_summary$cp[3], pch = 16, col = "blue", lwd = 7, cex = 2)
```




```r
plot(model_summary$bic,
  xlab = "Subset Size",
  ylab = "BIC",
  pch = 16,
  type = "l"
)
points(3, model_summary$bic[3], pch = 16, col = "red", lwd = 7, cex = 2)
```




```r
plot(model_summary$adjr2,
  xlab = "Subset Size",
  ylab = "Adjusted R^2",
  pch = 16,
  type = "l"
)
points(3, model_summary$adjr2[3],
  pch = 16, col = "dark green",
  lwd = 7, cex = 2
)
```



According to the Cp, the best model size is 3. According to the BIC, the best model size is 3. And according to the adjusted R^2, the best model size is 3. Because 3 was selected three times, that will be the subset size that I evaluate moving forward.

```
coefficients(full_model, id = 3)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##              6.061507              7.975280              8.876209
## poly(x, 10, raw = T)3
##             -2.982361
```

```
coefficients(full_model, id = 4)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##            6.07200775            8.38745596            8.84575641
## poly(x, 10, raw = T)3 poly(x, 10, raw = T)5
##           -3.44202574            0.08072292
```

In this case, the best subset model with three variables maintains X3. While the best subset model with four variables selects X3 but selects X5 over X4.

Question 8 Part D:

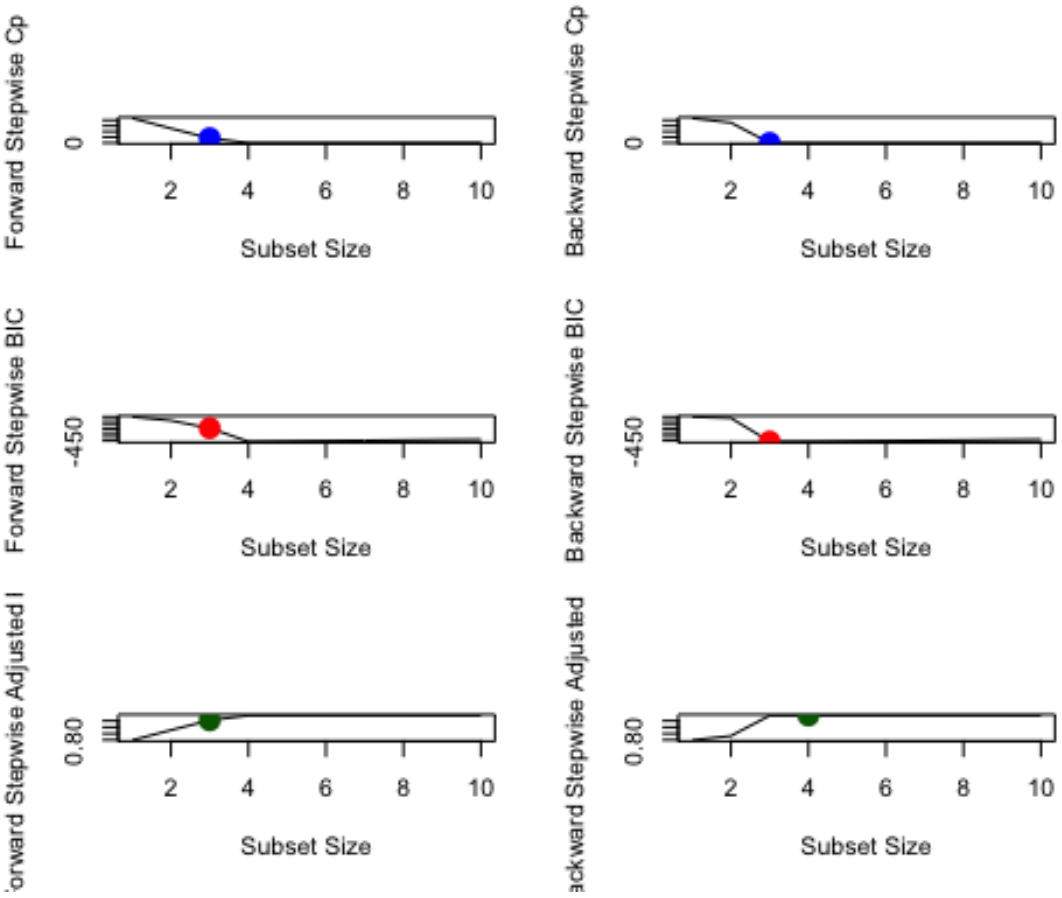
Repeat (c), using forward stepwise selection and also using backwards stepwise selection. How does your answer compare to the results in (c)?

```
forward_model <- regsubsets(y ~ poly(x, 10, raw = T),
  data = full_df, nvmax = 10,
  method = "forward"
)

backward_model <- regsubsets(y ~ poly(x, 10, raw = T),
  data = full_df, nvmax = 10,
  method = "backward"
)

forward_summary <- summary(forward_model)

backward_summary <- summary(backward_model)

which.min(forward_summary$cp)

## [1] 4

which.min(backward_summary$cp)

## [1] 4

which.min(forward_summary$bic)

## [1] 4

which.min(backward_summary$bic)

## [1] 3

which.max(forward_summary$adjr2)

## [1] 4

which.max(backward_summary$adjr2)

## [1] 4
```

Plotting

```
par(mfrow = c(3, 2))

plot(forward_summary$cp,
  xlab = "Subset Size", ylab = "Forward Stepwise Cp", pch = 16,
  type = "l"
)
points(3, forward_summary$cp[3], pch = 16, col = "blue", lwd = 7, cex = 2)
```

```r
plot(backward_summary$cp,
  xlab = "Subset Size", ylab = "Backward Stepwise Cp",
  pch = 16, type = "l"
)
points(3, backward_summary$cp[3], pch = 16, col = "blue", lwd = 7, cex = 2)


plot(forward_summary$bic,
  xlab = "Subset Size", ylab = "Forward Stepwise BIC",
  pch = 16,
  type = "l"
)
points(3, forward_summary$bic[3], pch = 16, col = "red", lwd = 7, cex = 2)


plot(backward_summary$bic,
  xlab = "Subset Size", ylab = "Backward Stepwise BIC",
  pch = 16,
  type = "l"
)
points(3, backward_summary$bic[3], pch = 16, col = "red", lwd = 7, cex = 2)


plot(forward_summary$adjr2,
  xlab = "Subset Size", ylab = "Forward Stepwise Adjusted R^2",
  pch = 16, type = "l"
)
points(3, forward_summary$adjr2[3],
  pch = 16, col = "dark green", lwd = 7,
  cex = 2
)


plot(backward_summary$adjr2,
  xlab = "Subset Size", ylab = "Backward Stepwise Adjusted R^2",
  pch = 16, type = "l"
)
points(4, backward_summary$adjr2[4],
  pch = 16, col = "dark green", lwd = 7,
  cex = 2
)
```



According to most of the statistical measures, 3 is the best model size. However, the backward stepwise method using the Adjusted R^2 statistical measure, computes 4 as the best model size.

```
coefficients(forward_model, id = 4)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           6.076252450          8.291401608          8.838232869
## poly(x, 10, raw = T)3 poly(x, 10, raw = T)7
##          -3.252652678          0.009133754
```

```
coefficients(backward_model, id = 4)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           6.079236362          8.231905828          8.833494180
## poly(x, 10, raw = T)3 poly(x, 10, raw = T)9
##          -3.180444193          0.001290827
```

```
coefficients(backward_model, id = 3)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##              6.061507             7.975280             8.876209
```

```
## poly(x, 10, raw = T)3
##             -2.982361
```

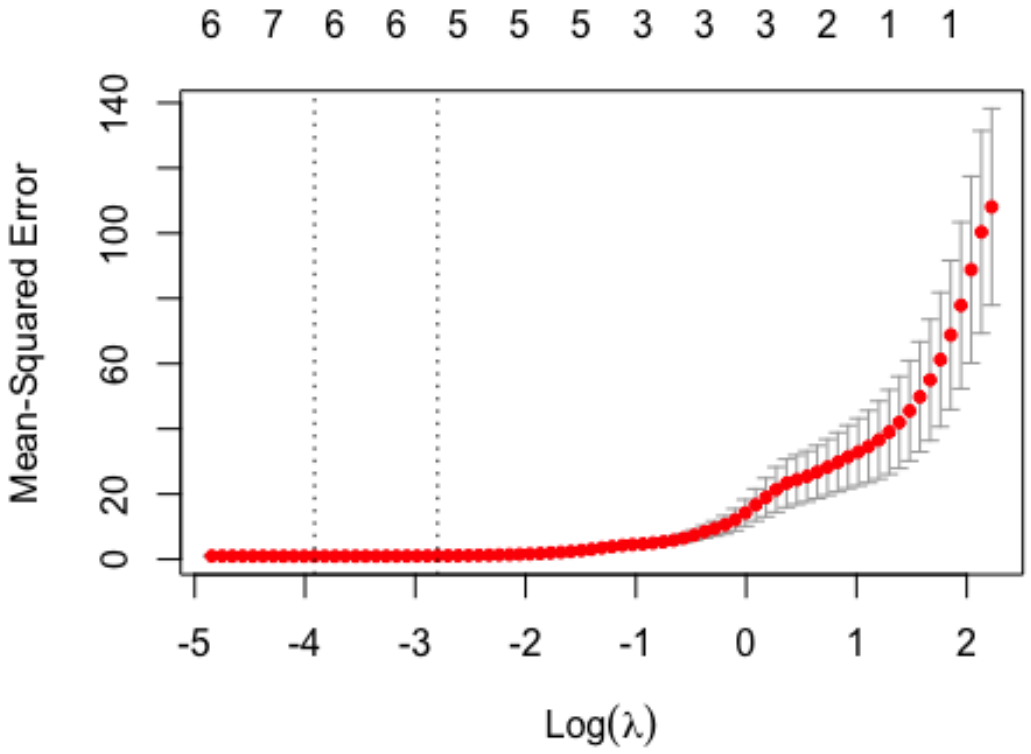
In this example, the forward stepwise with four variables selects X7 over X4. While the backward stepwise with four variables selects X9 over X4. Lastly, the backward stepwise with three variables maintains X3.

Question 8 Part E:

```
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.1.2

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-7
```

Now fit a lasso model to the simulated data, again using X, X2, …,X10 as predictors. Use cross-validation to select the optimal value of λ. Create plots of the cross-validation error as a function of λ. Report the resulting coefficient estimates, and discuss the results obtained.

```
# We train the Lasso on the full data we generated earlier

matrix_df <- model.matrix(y ~ poly(x, 10, raw = T), data = full_df)[, -1]

lasso_model <- cv.glmnet(matrix_df, Y, alpha = 1)

lambda_selection <- lasso_model$lambda.min

lambda_selection

## [1] 0.01995879

plot(lasso_model)
```




```
# Now we apply the lambda term we computed to the model

optimal_model <- glmnet(matrix_df, Y, alpha = 1)

predict(optimal_model, s = lambda_selection, type = "coefficients")

## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                s1
## (Intercept)             6.153975e+00
## poly(x, 10, raw = T)1   7.892415e+00
## poly(x, 10, raw = T)2   8.691806e+00
## poly(x, 10, raw = T)3  -2.962043e+00
## poly(x, 10, raw = T)4   4.459022e-03
## poly(x, 10, raw = T)5   .
## poly(x, 10, raw = T)6   6.781552e-03
## poly(x, 10, raw = T)7   .
## poly(x, 10, raw = T)8   1.627830e-04
## poly(x, 10, raw = T)9   .
## poly(x, 10, raw = T)10  1.943663e-06
```

This model tells us that the Lasso selects X6 over X5, X8 over X7, and X10 over X9. Some variables were dropped because they lacked statistical significance or collinearity with the other regression terms.

Question 8 Part F:

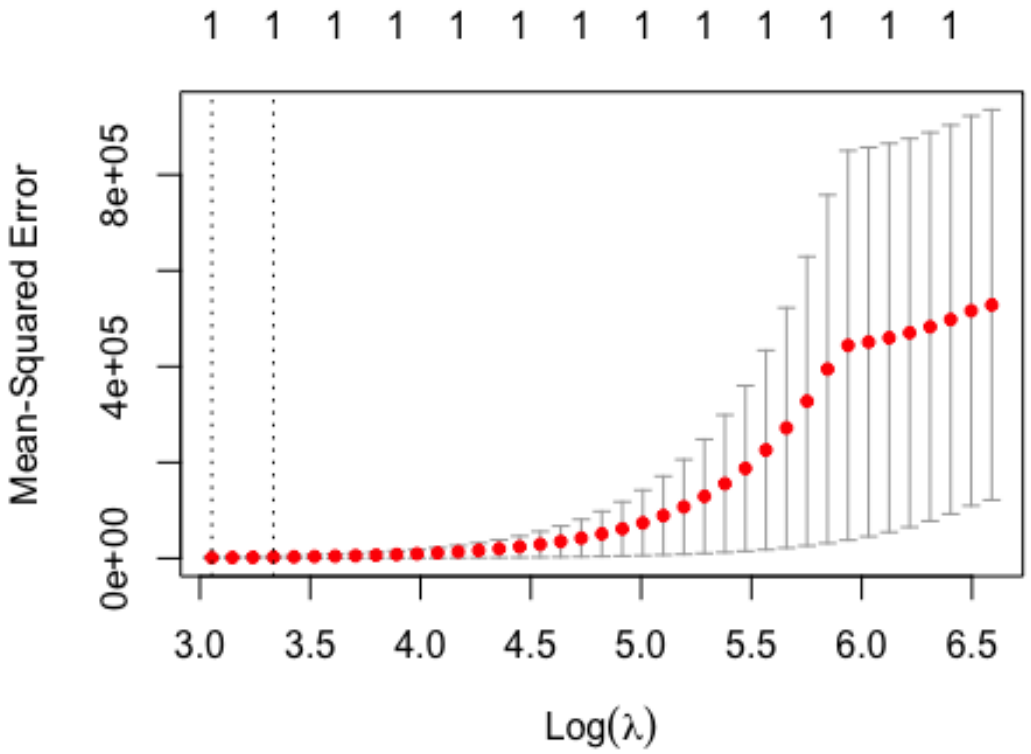
Now generate a response vector Y according to the model Y = β0 + β7X7 + ϵ, and perform best subset selection and the lasso. Discuss the results obtained.

```
b7 = 12

Y = b0 + b7*X^7 + e_term

full_df <- data.frame(y = Y, x = X)

full_model <- regsubsets(y ~ poly(x, 10, raw = T), data = full_df, nvmax =
10)

model_summary <- summary(full_model)

which.min(model_summary$cp)

## [1] 2

which.min(model_summary$bic)

## [1] 1

which.max(model_summary$adjr2)

## [1] 4

coefficients(full_model, id = 2)

##           (Intercept) poly(x, 10, raw = T)2 poly(x, 10, raw = T)7
##             6.0704904            -0.1417084            12.0015552

coefficients(full_model, id = 1)

##           (Intercept) poly(x, 10, raw = T)7
##               5.95894              12.00077

coefficients(full_model, id = 4)

##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##             6.0762524             0.2914016            -0.1617671
## poly(x, 10, raw = T)3 poly(x, 10, raw = T)7
##            -0.2526527            12.0091338
```

The coefficient generated by the BIC statistical measure is the most accurate, since it generated a one-variable model that has the highest statistical effect on the dependent variable. This is in contrast to the other statistical measures that included more than one regression variable (independent variables).

```
matrix_df <- model.matrix(y ~ poly(x, 10, raw = T), data = full_df)[, -1]
```

```
lasso_model <- cv.glmnet(matrix_df, Y, alpha = 1)

lambda_selection <- lasso_model$lambda.min

lambda_selection

## [1] 21.20275

plot(lasso_model)
```




```
optimal_model <- glmnet(matrix_df, Y, alpha = 1)

predict(optimal_model, s = lambda_selection, type = "coefficients")

## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                s1
## (Intercept)             7.435343
## poly(x, 10, raw = T)1   .
## poly(x, 10, raw = T)2   .
## poly(x, 10, raw = T)3   .
## poly(x, 10, raw = T)4   .
## poly(x, 10, raw = T)5   .
## poly(x, 10, raw = T)6   .
## poly(x, 10, raw = T)7  11.650942
```

```
## poly(x, 10, raw = T)8   .
## poly(x, 10, raw = T)9   .
## poly(x, 10, raw = T)10  .
```

This model tells us that the Lasso also selected a one-variable model where X7 is preferred over X1-X6, and where X8-10 are dropped because they lack statistical significance or collinearity with the other regression terms. It is important to highlight that the Lasso intercept value differs slighlt from the intercept value estimated in the BIC method.

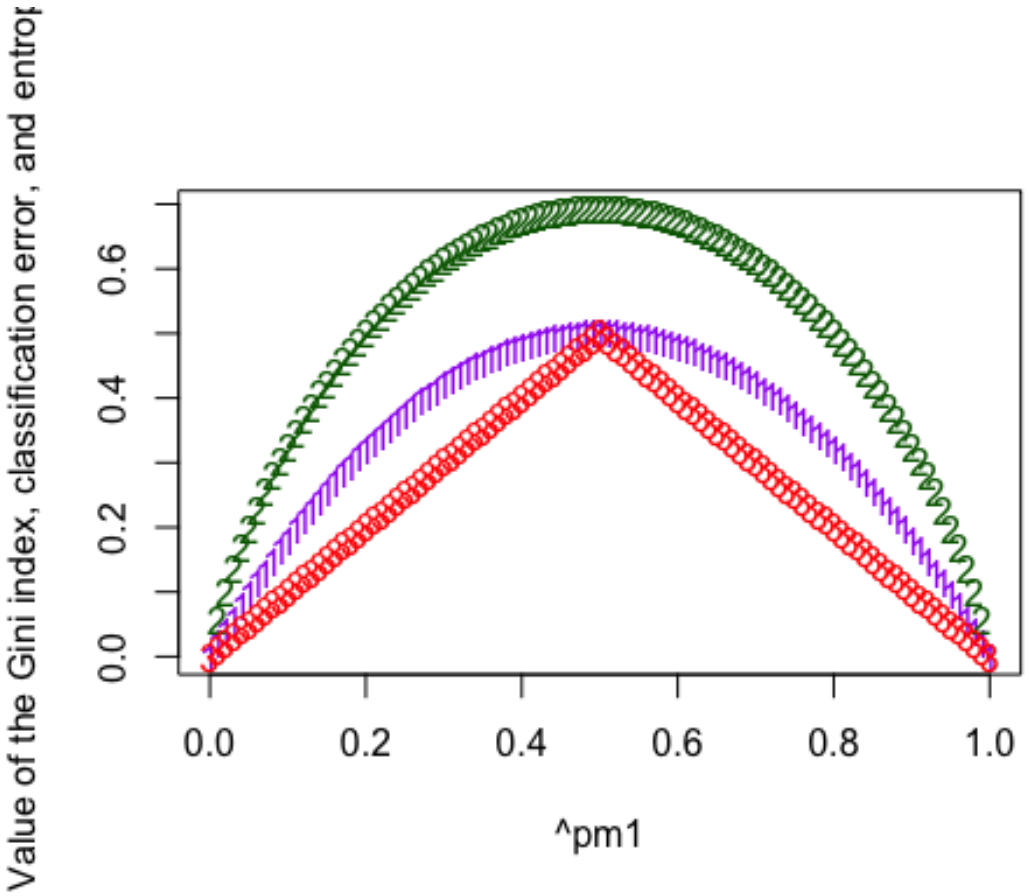
CITATION: https://www.statology.org/lasso-regression-in-r/

Chapter 8

Question 3:

Consider the Gini index, classification error, and entropy in a simple classification setting with two classes. Create a single plot that displays each of these quantities as a function of ˆpm1. The x-axis should display ˆpm1, ranging from 0 to 1, and the y-axis should display the value of the Gini index, classification error, and entropy.

```
plot_df <- seq(0, 1, 0.01)

gini_index <- plot_df * (1 - plot_df) * 2

entropy <- -(plot_df * log(plot_df) + (1 - plot_df) * log(1 - plot_df))

class_error <- 1 - pmax(plot_df, 1 - plot_df)

x_label <- "^pm1"

y_label <- "Value of the Gini index, classification error, and entropy"

matplot(plot_df, cbind(gini_index, entropy, class_error),
        col = c("purple", "dark green", "red"),
        xlab = x_label,
        ylab = y_label)
```



Question 8 Part A:

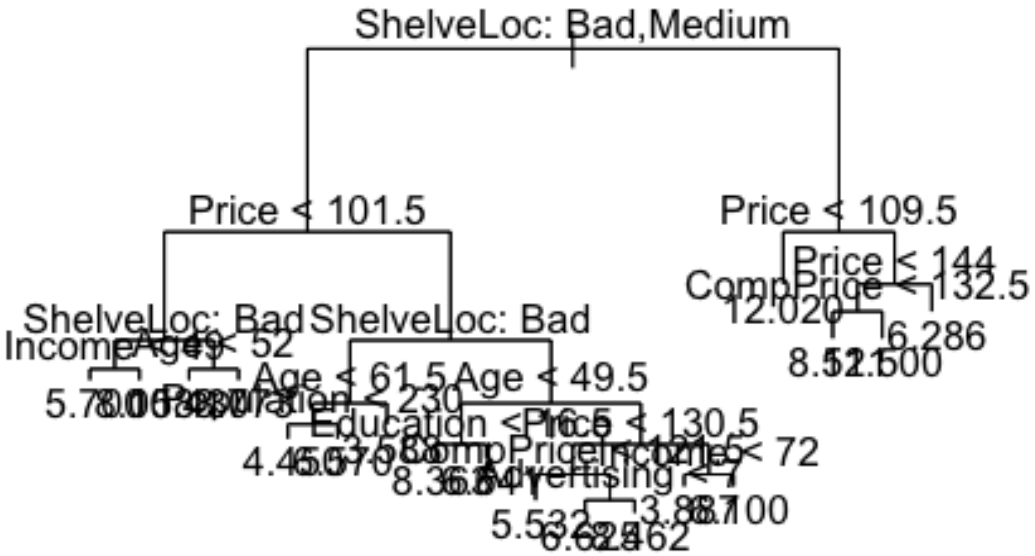
Split the data set into a training set and a test set

```
library(ISLR)

carseats_df <- Carseats

set.seed(123)

split_df <- sample(dim(carseats_df)[1], dim(carseats_df)[1] / 2)

training_df <- carseats_df[split_df, ]

test_df <- carseats_df[-split_df, ]
```

Question 8 Part B:

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

```r
library(tree)

## Warning: package 'tree' was built under R version 4.1.2

regression_tree <- tree(Sales ~ ., data = training_df)

summary(regression_tree)

##
## Regression tree:
## tree(formula = Sales ~ ., data = training_df)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "Age"         "Population"
## [6] "Education"   "CompPrice"   "Advertising"
## Number of terminal nodes:  18
## Residual mean deviance:  2.132 = 388.1 / 182
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.08000 -0.92870  0.06244  0.00000  0.87020  3.71700

plot(regression_tree)

text(regression_tree, pretty = 0)
```



The number of terminal nodes in the tree is 18, which means that the data has been divided into 18 groups based on the predictor variables. The residual mean deviance of 2.132 indicates that the model has a good fit to the data, and the distribution of residuals shows that the mean residual is zero, which indicates that the model is unbiased. The range of the residuals is from -4.08 to 3.71, which tells us that there might be some outliers in the data.

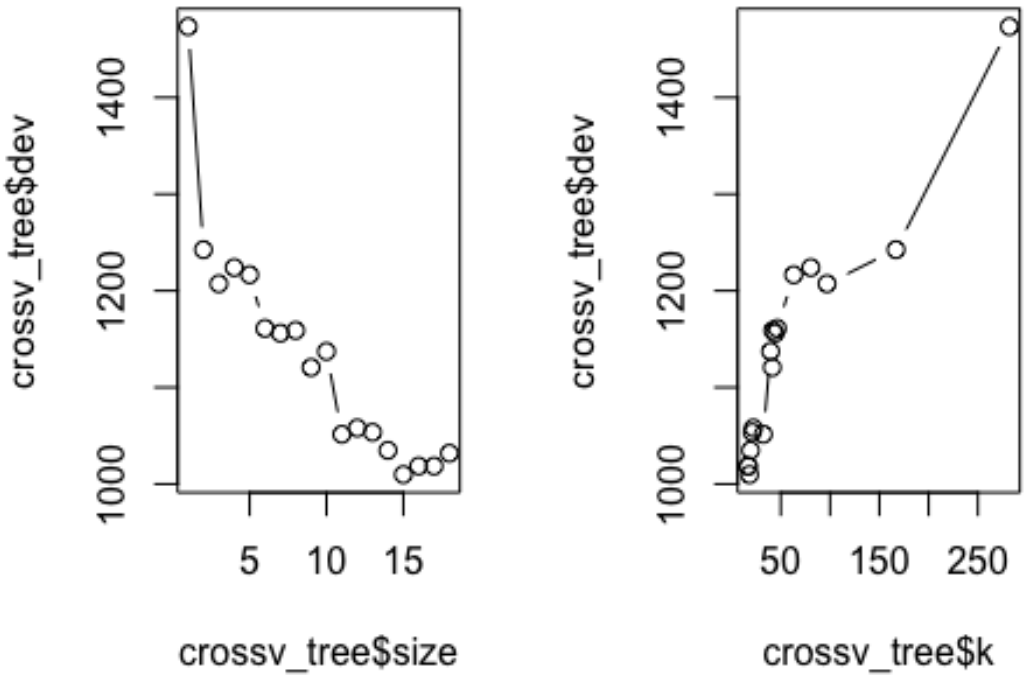
Overall, the regression tree model suggests that the predictor variables included in the analysis have a significant influence on the sales outcome, and that the model has a good fit to the data.

```
predict_carsts <- predict(regression_tree, test_df)

MSE_test <- mean((test_df$Sales - predict_carsts)^2)

output_string <- paste("The MSE for the test data is equal to", MSE_test)

print(output_string)

## [1] "The MSE for the test data is equal to 4.39535740766638"
```

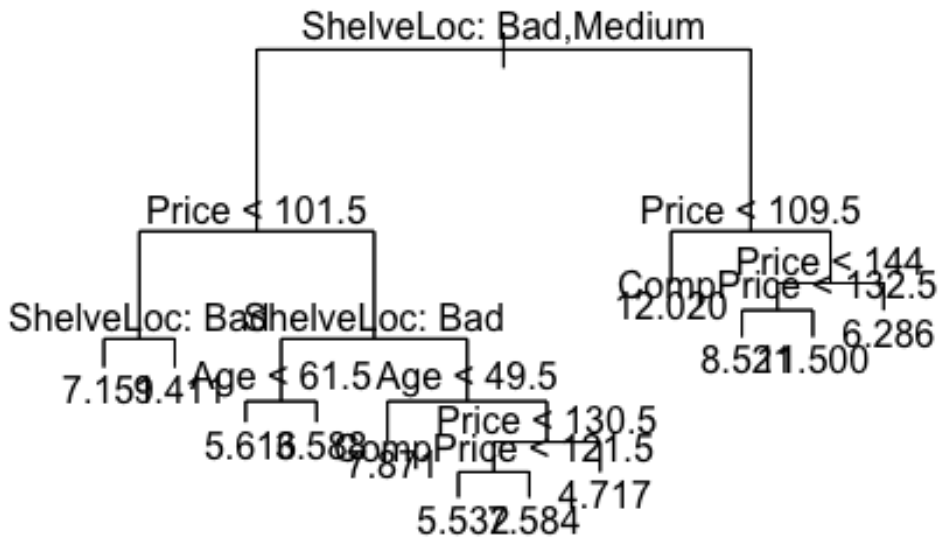
Question 8 Part C:

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

```
crossv_tree <- cv.tree(regression_tree, FUN = prune.tree)

par(mfrow = c(1, 2))

plot(crossv_tree$size, crossv_tree$dev, type = "b")

plot(crossv_tree$k, crossv_tree$dev, type = "b")
```




```r
# The best size according to the plot = 12

pruned_carseats <- prune.tree(regression_tree, best = 12)

par(mfrow = c(1, 1))

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




```
predict_pruned <- predict(pruned_carseats, test_df)

MSE_test2 <- mean((test_df$Sales - predict_pruned)^2)

output_string2 <- paste("The MSE for the test data is equal to", MSE_test2)

print(output_string2)

## [1] "The MSE for the test data is equal to 4.66297464277414"
```

Pruning the tree increased the MSE from 4.39 to 4.66.

Question 8 Part 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.

```
library(randomForest)

## Warning: package 'randomForest' was built under R version 4.1.2

## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
bagging_carseats <- randomForest(Sales ~ .,
  data = training_df, mtry = 10, ntree = 500,
  importance = T
)

predict_bagging <- predict(bagging_carseats, test_df)

MSE_test3 <- mean((test_df$Sales - predict_bagging)^2)

output_string3 <- paste("The MSE for the test data is equal to", MSE_test3)

print(output_string3)
```

```
## [1] "The MSE for the test data is equal to 2.70694468681881"
```

```
importance(bagging_carseats)
```

```
##                %IncMSE IncNodePurity
## CompPrice   20.45893952    163.315084
## Income       5.99352172     88.626184
## Advertising  6.70900949     73.007073
## Population  -1.84004720     53.079505
## Price       46.01586429    395.251820
## ShelveLoc   49.31816789    391.948958
## Age         17.74691675    171.659574
## Education    2.98753578     57.308595
## Urban        0.04864498      7.721022
## US           0.05207339      6.011265
```

The bagging approach improved the MSE by decreasing the value to 2.70. Further, the model shows that ShelveLoc, Price, CompPrice, and Age are the four variables that have the highest statistical impact on predicting Sale.

Question 8 Part 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.

```
random_carseats <- randomForest(Sales ~ .,
  data = training_df, mtry = 5, ntree = 500,
  importance = T
)

predict_random <- predict(random_carseats, test_df)

MSE_test4 <- mean((test_df$Sales - predict_random)^2)

output_string4 <- paste("The MSE for the test data is equal to", MSE_test4)

print(output_string4)

## [1] "The MSE for the test data is equal to 3.0920756110569"

importance(random_carseats)

##                %IncMSE IncNodePurity
## CompPrice   12.8153277    146.261391
## Income       7.0766835    111.252330
## Advertising  6.2121481     89.901889
## Population  -1.2975016     79.429067
## Price       35.5344924    331.725876
## ShelveLoc   41.3002362    335.894363
## Age         17.9183484    204.422771
## Education   -0.6312320     64.334408
## Urban       -1.0644618     10.562869
## US          -0.0673826      9.792536
```

The random forest method worsens the MSE by increasing the value to 3.09. We also see that ShelveLoc, Price, CompPrice, and Age are still the four variables that have the highest statistical impact on predicting Sale. Except Age moved up ahead of CompPrice.

```
random_carseats2 <- randomForest(Sales ~ .,
  data = training_df, mtry = 10, ntree = 500,
  importance = T
)

predict_random2 <- predict(random_carseats2, test_df)

MSE_test5 <- mean((test_df$Sales - predict_random2)^2)

output_string5 <- paste("The MSE for the test data is equal to", MSE_test5)
```

```
print(output_string5)

## [1] "The MSE for the test data is equal to 2.81576265274837"
```

In this case, when we increase the m - the number of variables considered at each split, the MSE decreased. In this example, I increased the m from 5 to 10 and the MSE decreased to 2.815, which improved the error rate.

Question 8 Part F:

Now analyze the data using BART, and report your results

```
library(dbarts)

bart_fit <- dbarts(
  formula = Sales ~ .,
  data = training_df,
  verbose = TRUE,
  n.samples = 500
)
```

Question 9 Part A:

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

```
library(ISLR)

OJ_df <- OJ

set.seed(321)

split_data2 <- sample(dim(OJ)[1], 800)

OJ_train <- OJ[split_data2, ]

OJ_test <- OJ[-split_data2, ]
```

Question 9 Part 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?

```r
library(tree)

tree_OJ <- tree(Purchase ~ ., data = OJ_train)

summary(tree_OJ)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ_train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "SalePriceMM"   "SpecialCH"     "ListPriceDiff"
## [5] "PriceDiff"
## Number of terminal nodes:  9
## Residual mean deviance:  0.7443 = 588.7 / 791
## Misclassification error rate: 0.1525 = 122 / 800
```

The tree only measures five variables: LoyalCH, SalePriceMM, SpecialCH, ListPriceDiff, and PriceDiff. The tree has 9 terminal nodes and the training error rate (which is also the Misclassification error rate) is equal to .1525.

Question 9 Part 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.

```r
tree_OJ
```

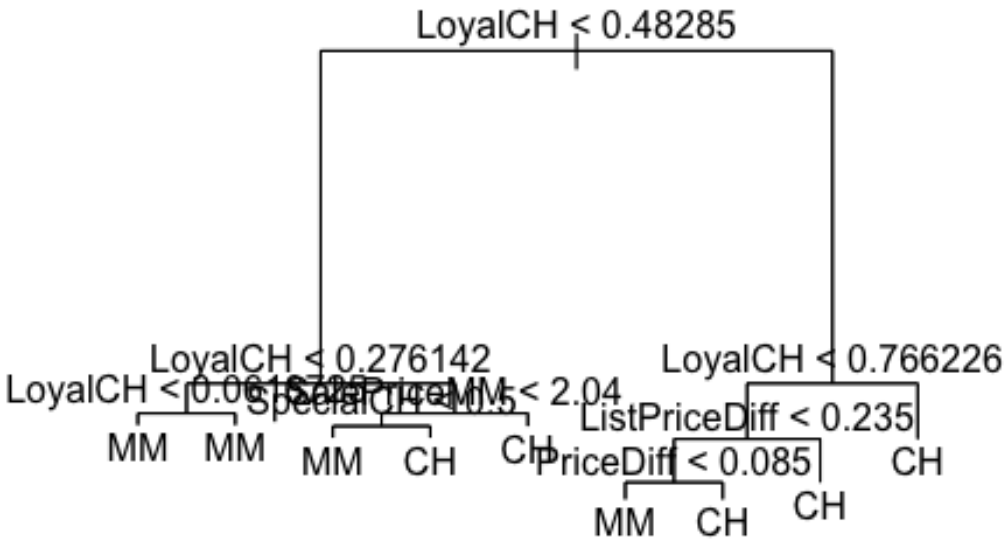
```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1072.00 CH ( 0.60750 0.39250 )
##    2) LoyalCH < 0.48285 308  332.60 MM ( 0.23052 0.76948 )
##      4) LoyalCH < 0.276142 175  128.40 MM ( 0.12000 0.88000 )
##        8) LoyalCH < 0.0616725 70   18.16 MM ( 0.02857 0.97143 ) *
##        9) LoyalCH > 0.0616725 105   99.30 MM ( 0.18095 0.81905 ) *
##      5) LoyalCH > 0.276142 133  176.10 MM ( 0.37594 0.62406 )
##       10) SalePriceMM < 2.04 76   85.47 MM ( 0.25000 0.75000 )
##         20) SpecialCH < 0.5 61   54.43 MM ( 0.16393 0.83607 ) *
##         21) SpecialCH > 0.5 15   20.19 CH ( 0.60000 0.40000 ) *
##       11) SalePriceMM > 2.04 57   78.58 CH ( 0.54386 0.45614 ) *
##    3) LoyalCH > 0.48285 492  426.90 CH ( 0.84350 0.15650 )
##      6) LoyalCH < 0.766226 236  277.80 CH ( 0.72458 0.27542 )
##       12) ListPriceDiff < 0.235 100  138.60 CH ( 0.51000 0.49000 )
##         24) PriceDiff < 0.085 60   77.69 MM ( 0.35000 0.65000 ) *
##         25) PriceDiff > 0.085 40   44.99 CH ( 0.75000 0.25000 ) *
##       13) ListPriceDiff > 0.235 136   98.52 CH ( 0.88235 0.11765 ) *
##      7) LoyalCH > 0.766226 256   96.87 CH ( 0.95312 0.04688 ) *
```

I will analyze terminal node 4. The splitting variable for this node is LoyalCH. The splitting value for this node is equal to .276142. There are 175 points in the subtree below this node. The deviance for all the points contained in the region below this node is 128.40. The * symbol for this node indicates that this is a terminal node. The prediction for this node is Sales is equal to MM. Therefore, approximately 12% of the points in this node have CH as the value of Sales, and the remaining 88% have MM as the value of Sales.

Question 9 Part D:

Create a plot of the tree, and interpret the results.

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



The plot shows that LoyalCH is the most important variable in the tree. It shows that the top 3 nodes contain LoyalCH. If LoyalCH is less than 0.27, then the tree will most likely predict MM. However, if LoyalCH is greater than 0.76, then the tree will most likely predict CH. For LoyalCH values that fall in between that range, the decision will also incorporate the variables PriceDiff and ListPriceDiff when deciding if it will be MM or CH

Question 9 Part 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_OJ <- predict(tree_OJ, OJ_test, type = "class")

table(OJ_test$Purchase, predict_OJ)

##     predict_OJ
##       CH  MM
##   CH 153  14
##   MM  30  73

test_error <- 1 - mean(predict_OJ == OJ_test$Purchase)

output_1 <- paste("The test error rate is equal to", test_error)

print(output_1)

## [1] "The test error rate is equal to 0.162962962962963"
```

Question 9 Part F:

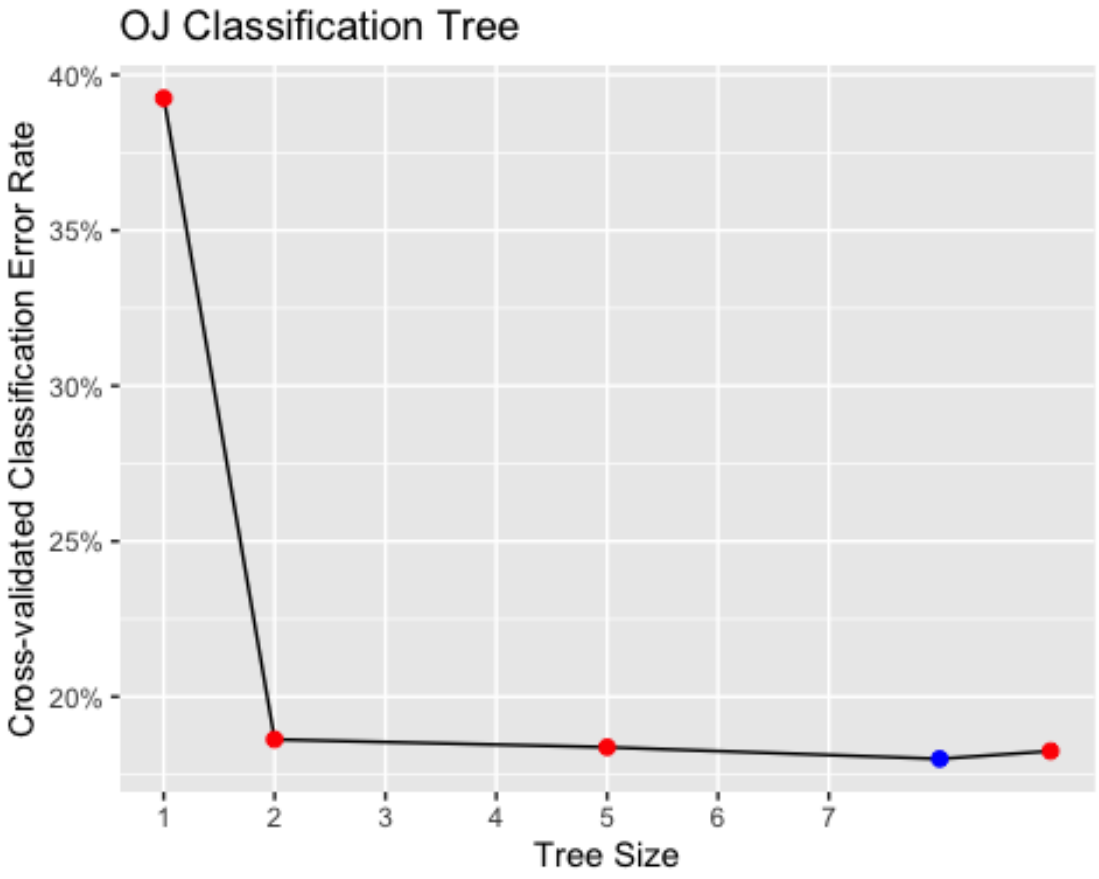
Apply the cv.tree() function to the training set in order to determine the optimal tree size.

```
set.seed(12)

cv_tree_df <- cv.tree(tree_OJ, K = 10, FUN = prune.misclass)

cv_tree_df

## $size
## [1] 9 8 5 2 1
##
## $dev
## [1] 146 144 147 149 314
##
## $k
## [1]       -Inf   0.000000   2.666667   6.000000 166.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Question 9 Part G:

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

```
data.frame(
  size = cv_tree_df$size,
  CV_Error = cv_tree_df$dev / nrow(OJ_train)
) %>%
  mutate(min_CV_Error = as.numeric(min(CV_Error) == CV_Error)) %>%
  ggplot(aes(x = size, y = CV_Error)) +
  geom_line(col = "black") +
  geom_point(size = 2, aes(col = factor(min_CV_Error))) +
  scale_x_continuous(breaks = seq(1, 7), minor_breaks = NULL) +
  scale_y_continuous(labels = scales::percent_format()) +
  scale_color_manual(values = c("red", "blue")) +
  theme(legend.position = "none") +
  labs(
    title = "OJ Classification Tree",
    x = "Tree Size",
    y = "Cross-validated Classification Error Rate"
  )
```



Question 9 Part H:

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

The tree sizes that correspond to the lowest cross-validated classification error rates are 6 and 7 trees (which have identical error rates). I will continue with the 6 since it has a lower tree size value than 7.

Question 9 Part 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.

```
pruned_tree_model <- prune.tree(tree_OJ, best = 6)

pruned_tree_model
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1072.00 CH ( 0.60750 0.39250 )
##    2) LoyalCH < 0.48285 308  332.60 MM ( 0.23052 0.76948 )
##      4) LoyalCH < 0.276142 175  128.40 MM ( 0.12000 0.88000 ) *
##      5) LoyalCH > 0.276142 133  176.10 MM ( 0.37594 0.62406 ) *
##    3) LoyalCH > 0.48285 492  426.90 CH ( 0.84350 0.15650 )
##      6) LoyalCH < 0.766226 236  277.80 CH ( 0.72458 0.27542 )
##       12) ListPriceDiff < 0.235 100  138.60 CH ( 0.51000 0.49000 )
##         24) PriceDiff < 0.085 60   77.69 MM ( 0.35000 0.65000 ) *
##         25) PriceDiff > 0.085 40   44.99 CH ( 0.75000 0.25000 ) *
##       13) ListPriceDiff > 0.235 136   98.52 CH ( 0.88235 0.11765 ) *
##      7) LoyalCH > 0.766226 256   96.87 CH ( 0.95312 0.04688 ) *
```

Question 9 Part J:

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

```
mean(predict(tree_OJ, type = "class") != OJ_train$Purchase)
```

```
## [1] 0.1525
```

```
mean(predict(pruned_tree_model, type = "class") != OJ_train$Purchase)
```

```
## [1] 0.1625
```

```
The training error for the pruned trees is higher.
```

Question 9 Part K:

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

```
mean(predict(tree_OJ, type = "class", newdata = OJ_test) != OJ_test$Purchase)

## [1] 0.162963

mean(predict(pruned_tree_model, type = "class", newdata = OJ_test) !=
OJ_test$Purchase)

## [1] 0.1777778

The test error for the pruned tree is higher.
```

Question 10 Part A:

Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

```
Hitters_new <- Hitters[!is.na(Hitters$Salary), ]

Hitters_new$log_salary <- log(Hitters_new$Salary)
```

Question 10 Part B:

Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

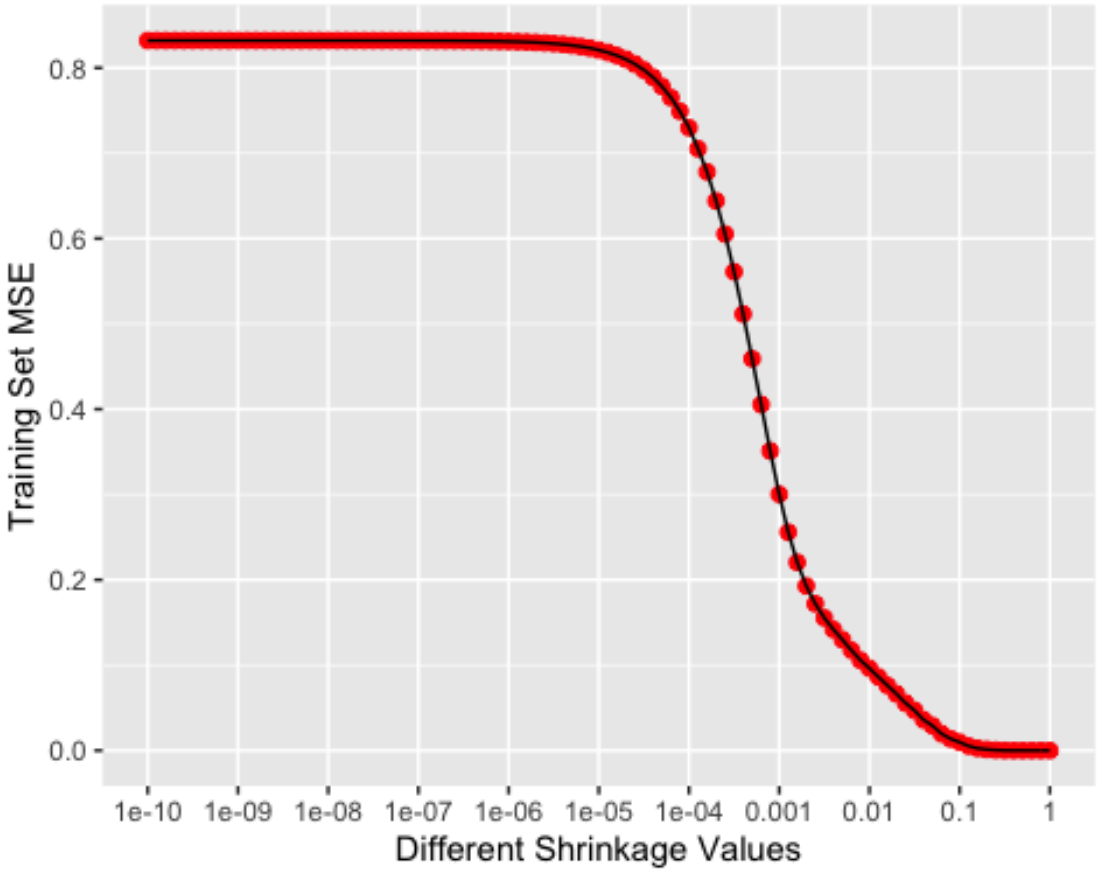
```
set.seed(910)

train_hitters <- Hitters_new[1:200, ]

test_hitters <- Hitters_new[201:nrow(Hitters_new), ]
```

Question 10 Part C:

Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

```
library(gbm)

## Loaded gbm 2.1.8.1

lambda_values <- 10^seq(-10, 0, 0.1)

set.seed(0)
```

```r
H.train_MSE <- c()
H.test_MSE <- c()

for (i in 1:length(lambda_values)) {
  boosting_H <- gbm(log_salary ~ . - Salary,
    data = train_hitters,
    distribution = "gaussian",
    n.trees = 1000,
    interaction.depth = 2,
    shrinkage = lambda_values[i]
  )

  H.train_MSE[i] <- mean((predict(boosting_H, train_hitters, n.trees = 1000)
- train_hitters$log_salary)^2)

  H.test_MSE[i] <- mean((predict(boosting_H, test_hitters, n.trees = 1000) -
test_hitters$log_salary)^2)
}

data.frame(lambda = lambda_values, H.train_MSE) %>%
  ggplot(aes(x = lambda, y = H.train_MSE)) +
  geom_point(size = 2, col = "red") +
  geom_line(col = "black") +
  scale_x_continuous(
    trans = "log10", breaks = 10^seq(-10, 0),
    labels = 10^seq(-10, 0), minor_breaks = NULL
  ) +
  labs(
    x = "Different Shrinkage Values",
    y = "Training Set MSE"
  )
```




```
min(H.test_MSE)
```

```
## [1] 0.2646783
```

```
lambda_values[which.min(H.test_MSE)]
```

```
## [1] 0.06309573
```

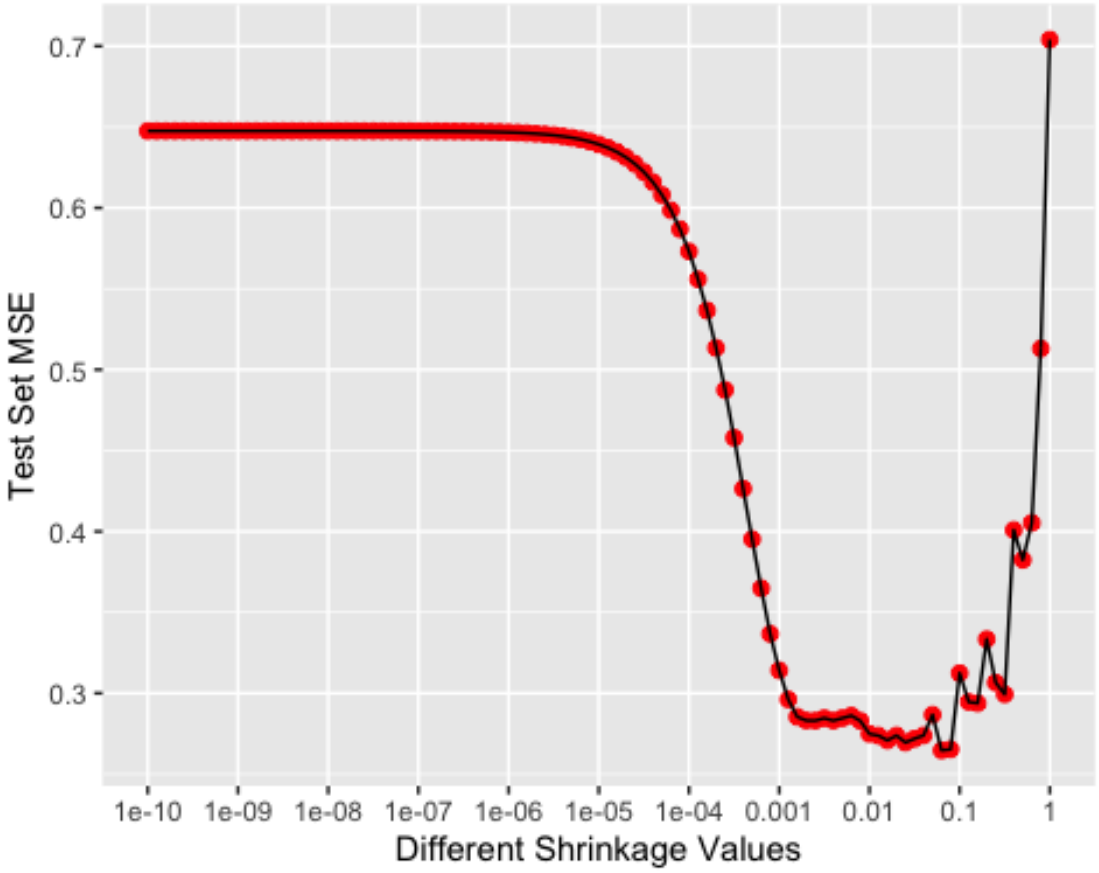
```
# The minimum test error is obtained at lambda = 0.06
```

Question 10 Part D:

Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

```
data.frame(lambda = lambda_values, H.test_MSE) %>%
  ggplot(aes(x = lambda, y = H.test_MSE)) +
  geom_point(size = 2, col = "red") +
  geom_line(col = "black") +
  scale_x_continuous(
    trans = "log10", breaks = 10^seq(-10, 0),
    labels = 10^seq(-10, 0), minor_breaks = NULL
  ) +
```

```
  labs(
    x = "Different Shrinkage Values",
    y = "Test Set MSE"
  )
```



Question 10 Part E:

Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

```
# Linear Regression Model

linear_regression <- lm(log_salary ~ . - Salary, data = train_hitters)

linear_prediction <- predict(linear_regression, test_hitters)

MSE_linear <- mean((test_hitters$log_salary - linear_prediction)^2)

output_2 <- paste("The MSE for the linear regression", MSE_linear)

print(output_2)
```

```
## [1] "The MSE for the linear regression 0.491795937545494"
```

```
# Lasso Model
```

```
library(glmnet)
```

```
set.seed(135)
```

```
lasso_train <- model.matrix(log_salary ~ . - Salary, data = train_hitters)
```

```
y <- train_hitters$log_salary
```

```
lasso_test = model.matrix(log_salary ~ . - Salary, data = test_hitters)
```

```
lasso_regression <- glmnet(lasso_train, y, alpha = 1)
```

```
lasso_prediction = predict(lasso_regression, s = 0.01, newx = lasso_test)
```

```
MSE_lasso <- mean((test_hitters$log_salary - lasso_prediction)^2)
```

```
output_3 <- paste("The MSE for the lasso method", MSE_lasso)
```

```
print(output_3)
```

```
## [1] "The MSE for the lasso method 0.470053732715625"
```

```
# Comparing the test MSE for all three regression models
```

```
min(H.test_MSE)
```

```
## [1] 0.2646783
```

```
print(output_2)
```

```
## [1] "The MSE for the linear regression 0.491795937545494"
```

```
print(output_3)
```

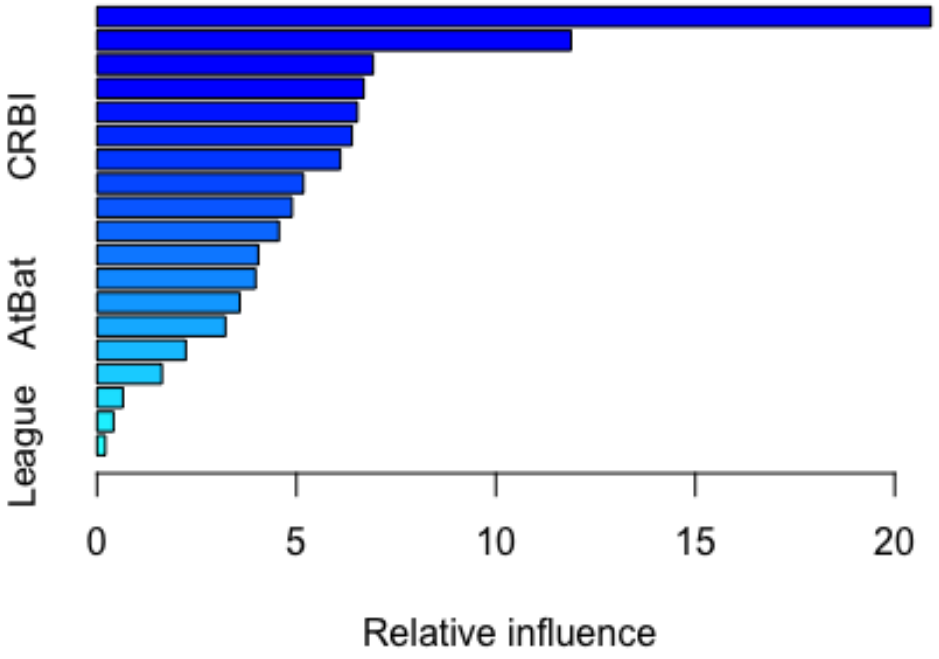
```
## [1] "The MSE for the lasso method 0.470053732715625"
```

Comparing the test MSE of the boosting method to the test MSE of a standard linear regression and the lasso regression model, it appears that the MSE is lowest in the boosting regression model. Both the the standard linear regression and the lasso regression model have similar values in the .47 and .49 range, while the MSE for the boosting method is in the .26 range. The boosting method has the smallest MSE across the 3 regression approaches.

Question 10 Part F:

Which variables appear to be the most important predictors in the boosted model?

```
important_predictors <- gbm(log_salary ~ . - Salary,
  data = train_hitters, distribution = "gaussian",
  n.trees = 1000, shrinkage = lambda_values[which.min(H.test_MSE)]
)

summary(important_predictors)
```




```
##                 var    rel.inf
## CAtBat       CAtBat 20.9070760
## CRuns         CRuns 11.8872633
## PutOuts     PutOuts  6.9221832
## Walks         Walks  6.6873035
## Years         Years  6.5273284
## CRBI           CRBI  6.3912514
## CHmRun       CHmRun  6.1019028
## CWalks       CWalks  5.1753076
## Assists     Assists  4.8904841
## Hits           Hits  4.5597406
## RBI             RBI  4.0437128
```

```
## CHits         CHits  3.9753497
## AtBat         AtBat  3.5799580
## HmRun         HmRun  3.2194977
## Errors       Errors  2.2273434
## Runs           Runs  1.6273560
## Division   Division  0.6524872
## NewLeague NewLeague  0.4155593
## League       League  0.2088952
```

It appears that CAtBat and CRuns were the two most important predictors in the boosted model in that order.

Question 10 Part G:

Now apply bagging to the training set. What is the test set MSE for this approach?

```
library(randomForest)

set.seed(963)

random_hitters <- randomForest(log_salary ~ . - Salary,
  data = train_hitters,
  ntree = 500,
  mtry = 10
)

random_predict <- predict(random_hitters, test_hitters)

MSE_random <- mean((test_hitters$log_salary - random_predict)^2)

output_5 <- paste("The MSE for the bagging method", MSE_random)

print(output_5)
```

```
## [1] "The MSE for the bagging method 0.224379967352073"
```