Homework #10

Justin Robinette

April 2, 2019

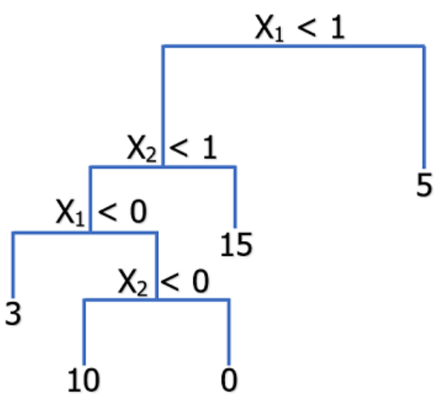
No collaborators for any problem

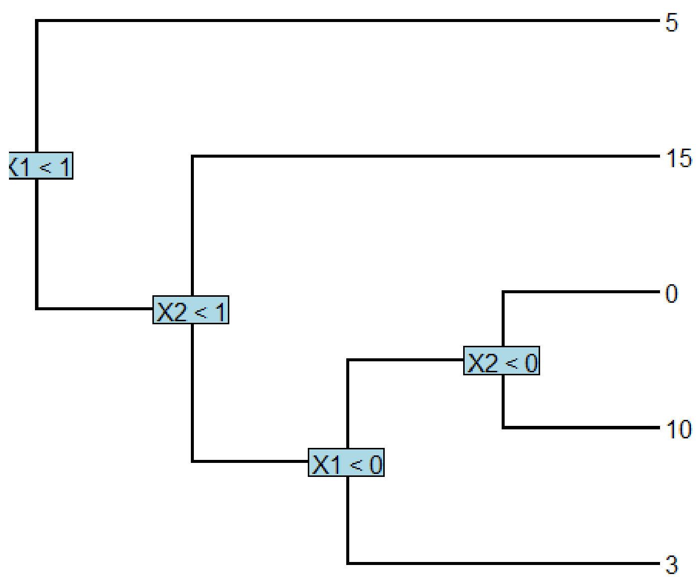
Question 8.4.4, pg 332: This question relates to the plots in Figure 8.12.

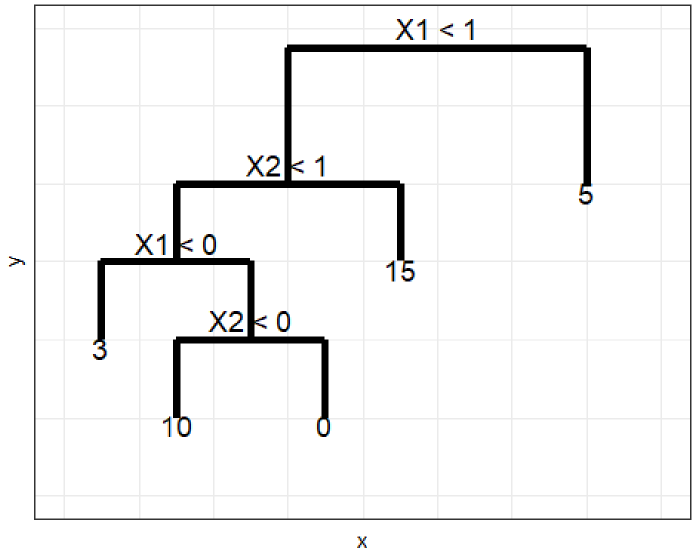
Part A: Sketch the tree corresponding to the partition of the predictor space illustrated in the left-hand panel of Figure 8.12. The numbers inside the boxes indicate the mean of Y within each region.

Results: Below I've "sketched" a depiction of the tree based on the left side of Figure 8.12 on pg 333 of the text. I've also included a statement showing the if, else if, else reasoning that was used to derive the image. As the instructions didn't specify a method for the sketch, I composed the drawing outside of R and then loaded the png file using knitr's **include_graphics()** function. I also used the **phytools** library to sketch it another way. Lastly, I used ggplot to make a comparable tree.

```
## [1] "if X1 >= 1 then y_hat = 5, else if X2 >= 1 then y_hat = 15, else if X1 < 0 then y_hat = 3, else if X2 < 0 then y_hat = 10, else y_hat = 0"
```



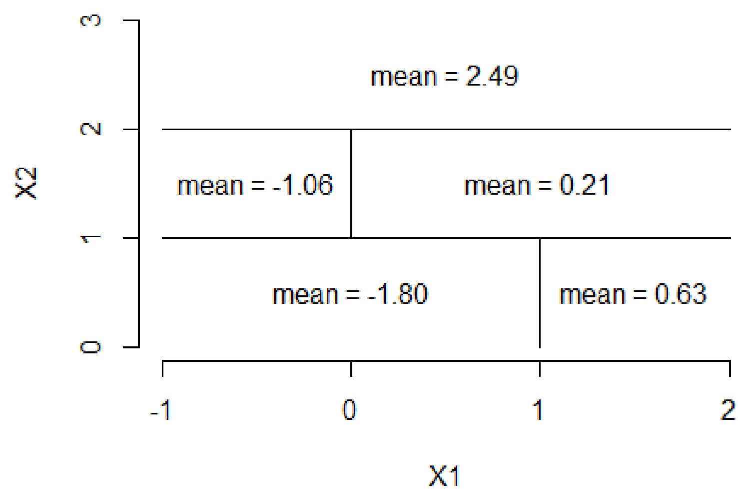


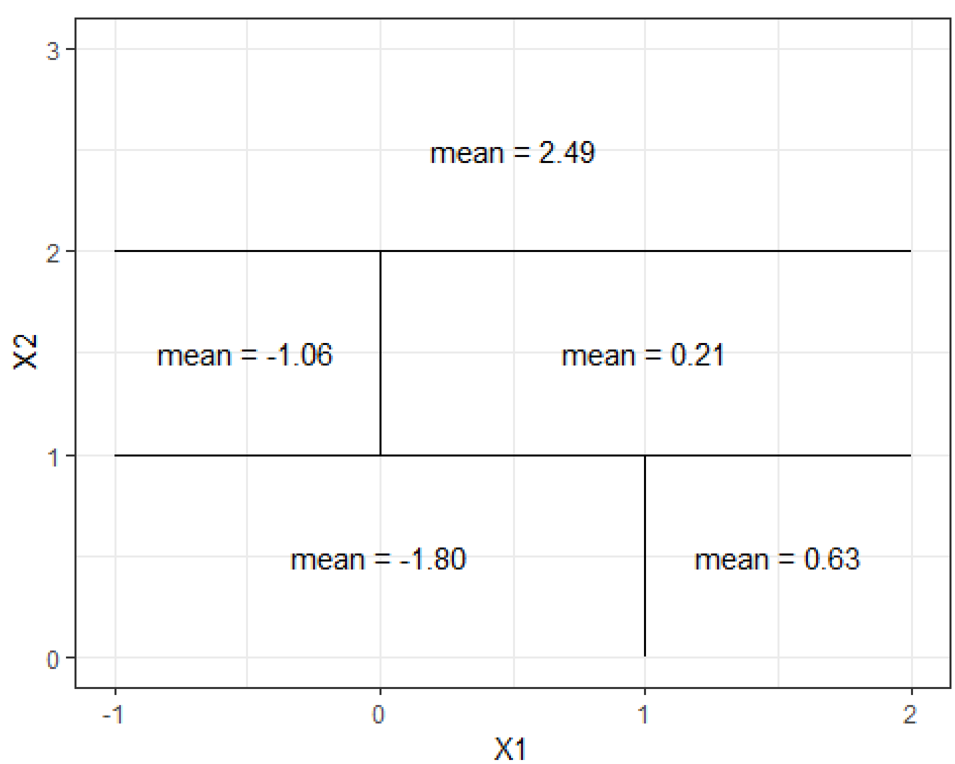


Part B: Create a diagram similar to the left-hand panel of Figure 8.12, using the tree illustrated in the right-hand panel of the same figure. You should divide up the predictor space into the correct regions, and indicate the mean for each region.

Results: First, I created a blank box w/ appropriate axes labels and axes limits. Then I added partitions throughout the box to represent the split of the tree in Figure 8.12. Lastly, I added the mean values of each region per the instructions.

Per standard homework instructions, a ggplot is included.





Question 8.4.8, pg 333: In the lab, a classification tree was applied to the **Carseats** data set after convertion **Sales** using regression trees and related approaches, treating the response as a quantitative variable.

Part A: Split the data set into a training set and a test set.

Results: Here I split the data set with 70% of the obs going to training and 30% going to the test data set. I printed a table with a breakdown of number of obs in each set to confirm.

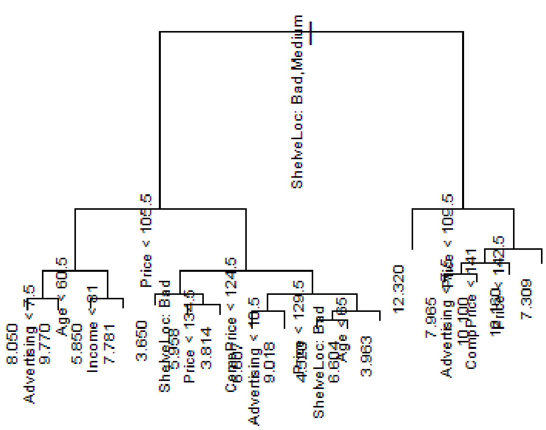
# of Obs

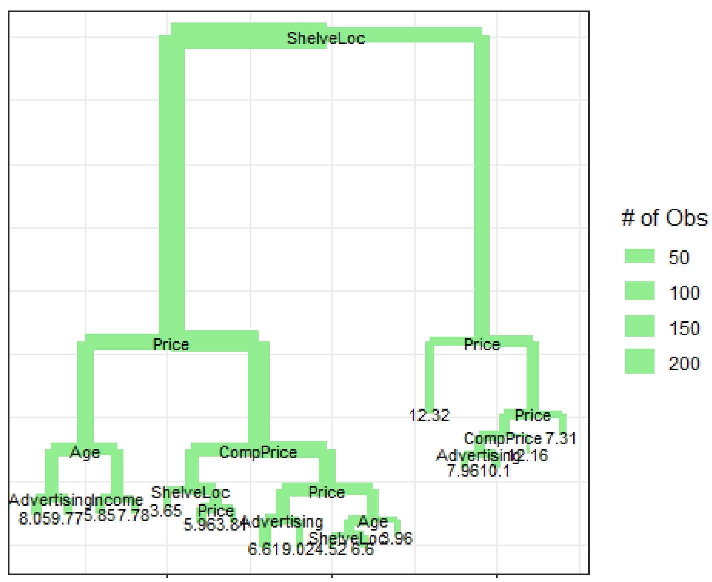
Carseats	Training	Test
400	280	120

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

Results: I fit a regression tree using the training set. After plotting, we can see that the first two splits occur based on **ShelveLoc** and **Price**. My hypothesis is that these are the two most important predictors of **Sales**. This will be examined further in a later exercise.

The test MSE of the regression tree is **5.032502**, as shown in the table below.



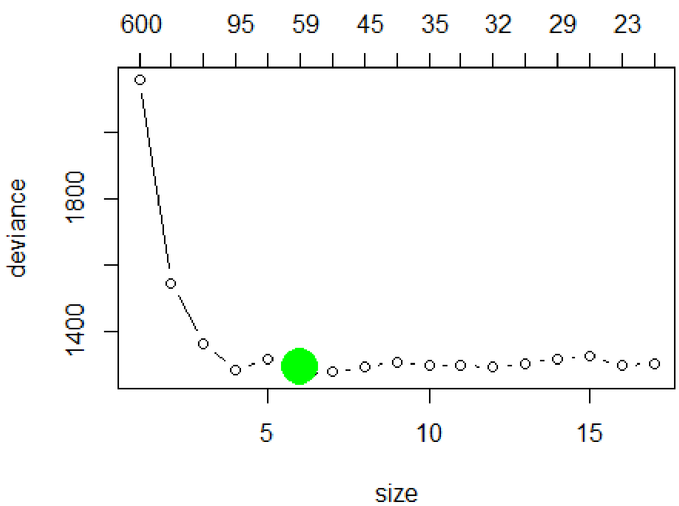


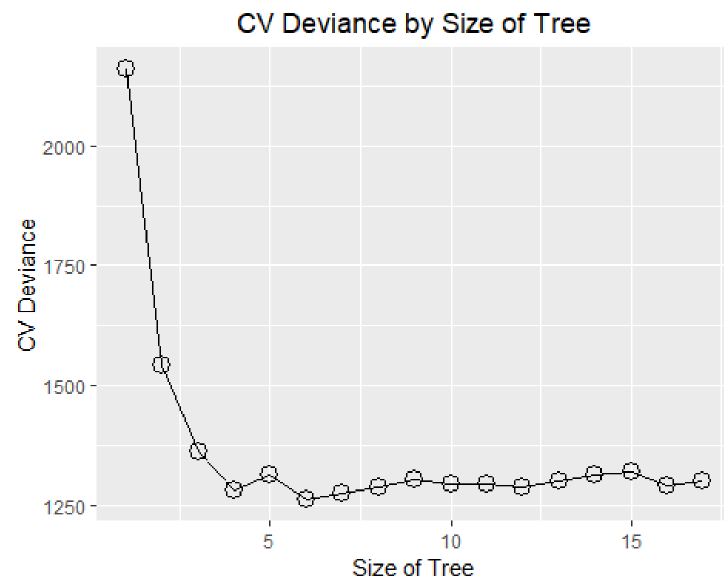
Tree Test MSE
5.032502

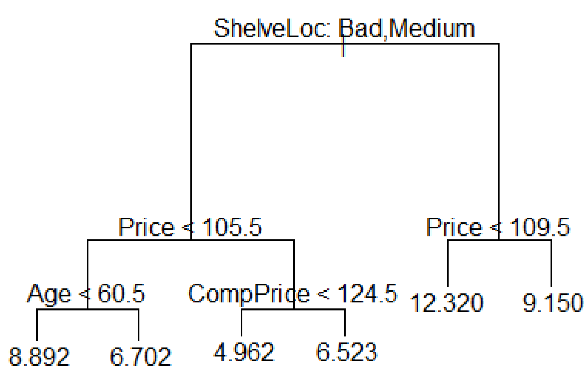
Part C: Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

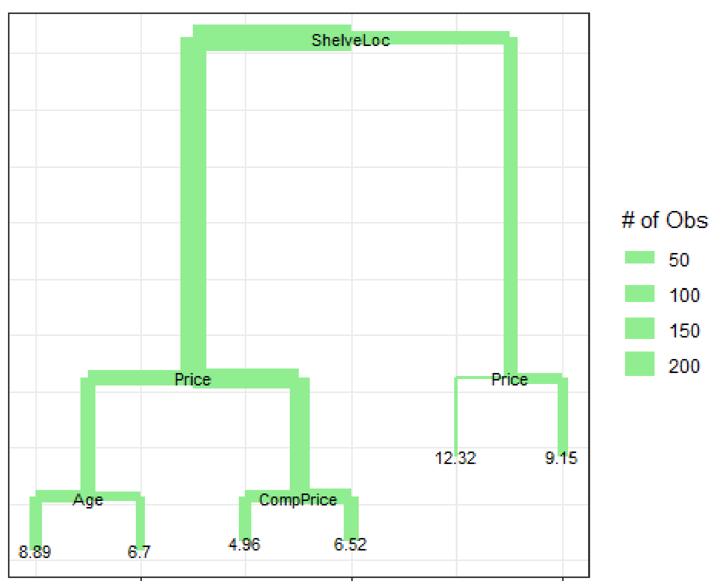
Results: I used cross-validation to determine the optimal level of tree complexity. As we can see from the plots below, the optimal level is 6.

After pruning the tree to 6 terminal nodes, as shown below, the test MSE is higher **(5.852491)** than it was with the original tree **(5.032502)**. In this case, pruning to the optimal complexity did not result in an increased level of accuracy with the test data set.









Test MSE by Tree

Tree	Pruned Tree
5.032502	5.852491

Part D: Use the bagging approach in order to analyze the data. What test MSE do you obtain? Use the **importance()** function to determine which variables are most important.

Results: Using bagging, we achieved a test MSE of **3.567435** which, as we can see from the table below, is considerably better than with the Tree and Pruned Tree.

The table and plot below show the importance by predictor. As we can see, the most important predictors in this model are **Price** and **ShelveLoc**.

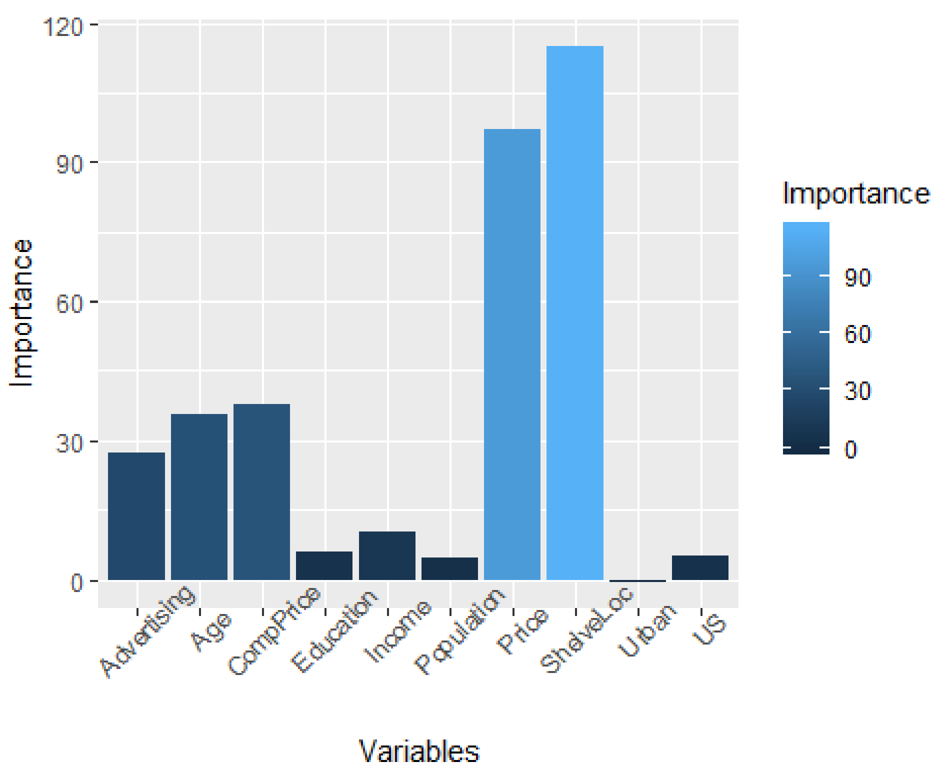
No base R plots are included as a plot was not requested.

Test MSE by Method

Tree	Pruned Tree	Bagging
5.032502	5.852491	3.567435

Carseats Predictor Importance from Bagging

	%IncMSE	IncNodePurity
CompPrice	37.9511977	203.565262
Income	10.4327518	101.834069
Advertising	27.3915648	138.046069
Population	4.7122477	75.414660
Price	97.2454385	581.675519
ShelveLoc	115.0312582	678.341223
Age	35.6170423	222.310874
Education	6.0109270	59.166851
Urban	-0.5181353	9.998805
US	5.0399548	10.185178



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.

Results: The test MSE obtain from random forest is **3.746423**.

As we can see from the plots below, **Price** and **ShelveLoc** are again the most important predictors with this model. Regarding the number of variables considered at each split **(m)** - denoted by 'mtry' in the randomForest() function, in the previous example with bagging we used 'm' ($mtry$) equal to the number of predictors (10). In this exercise, we are using the default for 'm', which is $mtry = \sqrt{p}$ where P is the number of predictors. In bagging, as mentioned above, $mtry = ncol(df) - 1$.

As we can see from the table below, the MSE is lower when using bagging with 'm' = 10 **(3.567)** than it is when using the default for 'm' **(3.746).**

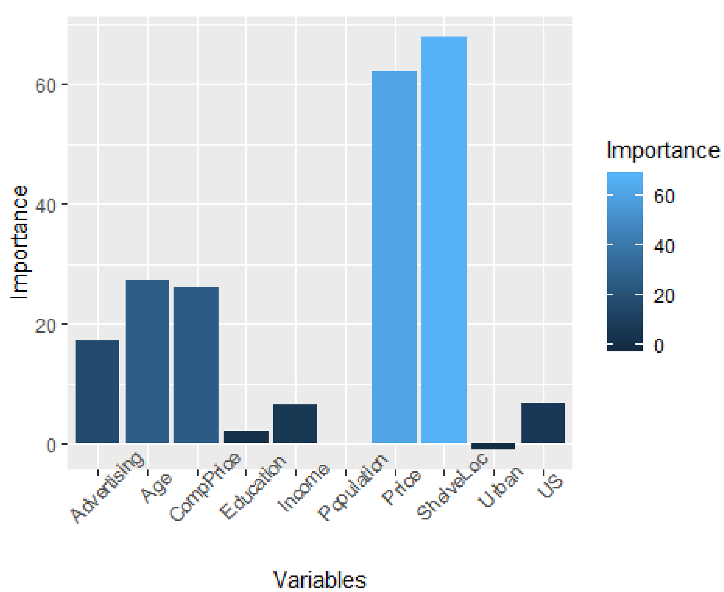
No base R plots are included as a plot was not requested.

Test MSE by Method

Tree	Pruned Tree	Bagging	Random Forest
5.032502	5.852491	3.567435	3.746423

Carseats Predictor Importance from Random Forest

	%IncMSE	IncNodePurity
CompPrice	26.1258852	192.28307
Income	6.5342518	151.10503
Advertising	17.2462107	158.98181
Population	0.2208695	131.08130
Price	62.2474163	477.18129
ShelveLoc	67.8940131	508.38997
Age	27.3053716	252.49111
Education	2.1967770	91.46808
Urban	-0.8736728	18.18939
US	6.8370183	26.12703



Question 8.4.9, pg 334: This problem involves the **OJ** data set which is part of the **ISLR** package.

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

Results: I created a training set of 800 random obs, leaving the remaining rows for the test set. I printed a breakdown of the number of obs per set to confirm.

# of Obs

OJ	Training	Test
1070	800	270

Part B: Fit a tree to the training data with **Purchase** as the response and the other variables as the 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?

Results: The tree, fit from the training data set, has an training error rate of 0.16625 and 8 terminal nodes.

Summary of Tree on OJ Training Set

Training Error Rate	# Terminal Nodes
0.16625	8

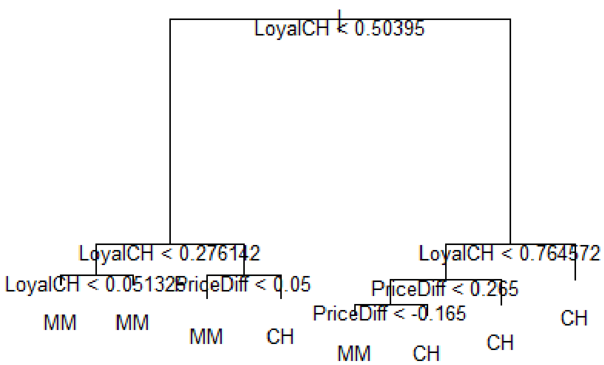
Part C: Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

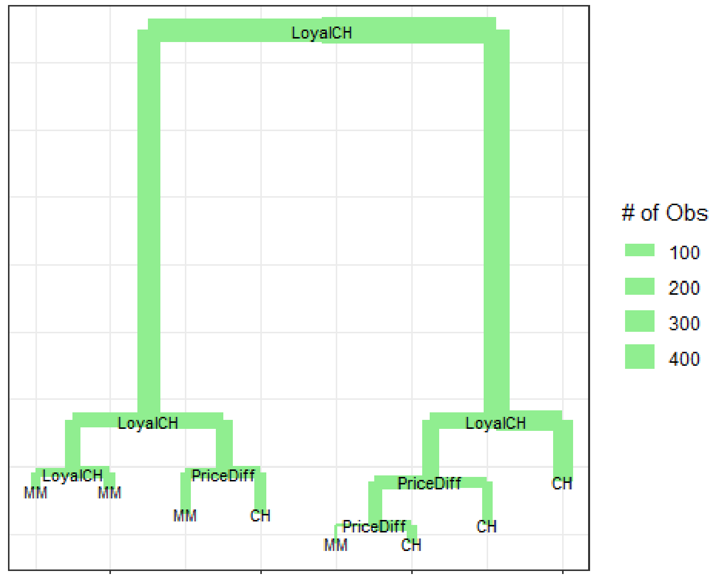
Results: Below is a detailed text output of the tree. Per the instructions, I choose the first terminal node, which is denoted by the asterisk, of LoyalCH < 0.051325. The number of observations on this branch of the tree is 60. The deviance is 10.17. The overall prediction for the branch for *Purchase* is **MM**. 1.667% of the obs in this branch have a *Purchase* of **CH**. Over 98% of the obs in this branch have a *Purchase* of **MM**.

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1073.00 CH ( 0.60500 0.39500 )
##    2) LoyalCH < 0.50395 348  411.90 MM ( 0.27874 0.72126 )
##      4) LoyalCH < 0.276142 163  121.40 MM ( 0.12270 0.87730 )
##        8) LoyalCH < 0.051325 60   10.17 MM ( 0.01667 0.98333 ) *
##        9) LoyalCH > 0.051325 103   98.49 MM ( 0.18447 0.81553 ) *
##      5) LoyalCH > 0.276142 185  251.20 MM ( 0.41622 0.58378 )
##       10) PriceDiff < 0.05 76   72.61 MM ( 0.18421 0.81579 ) *
##       11) PriceDiff > 0.05 109  148.40 CH ( 0.57798 0.42202 ) *
##    3) LoyalCH > 0.50395 452  372.30 CH ( 0.85619 0.14381 )
##      6) LoyalCH < 0.764572 191  223.70 CH ( 0.72775 0.27225 )
##       12) PriceDiff < 0.265 114  154.50 CH ( 0.58772 0.41228 )
##         24) PriceDiff < -0.165 34   42.81 MM ( 0.32353 0.67647 ) *
##         25) PriceDiff > -0.165 80   97.74 CH ( 0.70000 0.30000 ) *
##       13) PriceDiff > 0.265 77   37.01 CH ( 0.93506 0.06494 ) *
##      7) LoyalCH > 0.764572 261  103.30 CH ( 0.95019 0.04981 ) *
```

Part D: Create a plot of the tree, and interpret the results.

Results: Both base R and ggplots of the tree are below for comparison. As we can see from the base R plot, the original split is based on whether the value of *LoyalCH* is < 0.50395. Since the majority of the nodes are decided by *LoyalCH*, we could postulate that this is the most important predictor of *Purchase*. The only other split occurs based on *PriceDiff*.





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?

Results: After predicting the response on the test data, I printed a confusion matrix. As we can see, the model predicted the correct response at a higher rate when the *Purchase* value was **CH**. With that being said, the confusion matrix appears to show relatively accurate predictions of the response variable.

Lastly, I printed the test error rate, which is **0.1556**. Interestingly, this is worse than the error rate on the training set, which was **0.16625**.

```
##          Predicted
## Observed   CH  MM
##        CH 153  16
##        MM  26  75
```

Test Error Rate

Tree
0.1556

Part F: Apply the **cv.trees()** function to the training set in order to determine the optimal tree size.

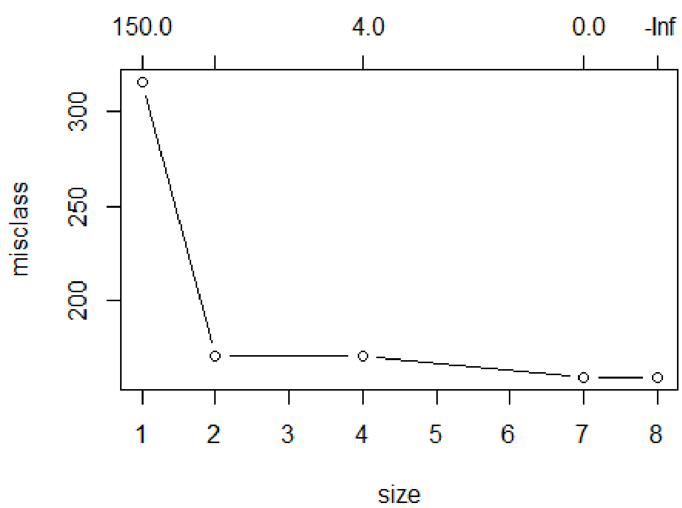
Results: After utilizing cv.trees() on the training set, we see that the optimal number of terminal nodes is 7. Both 7 and 8 misclassify 159 observations.

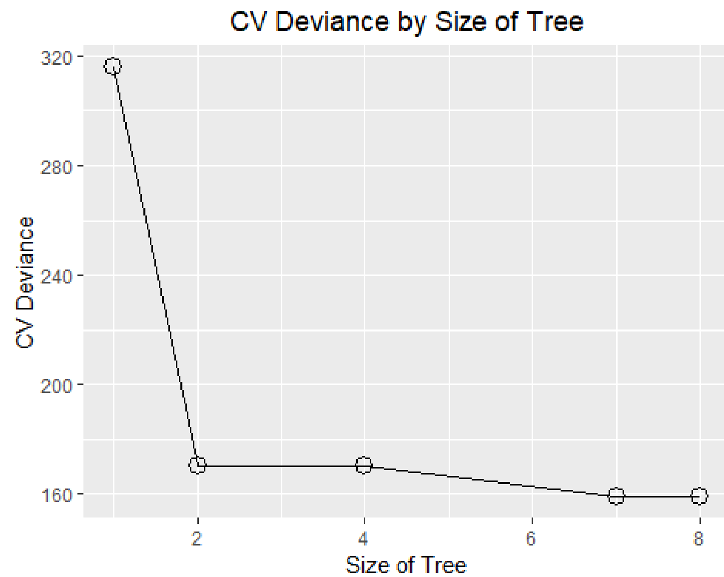
Optimal Terminal Nodes for Tree based on CV

# Terminal Nodes	# of Misclassifications
8	159
7	159

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

Results: Below are two plots representing the misclassification rate on the y-axis and the tree size on the x-axis, per the instructions. Again we see, the misclassification rate remains level when going from 7 terminal nodes to 8 terminal nodes.





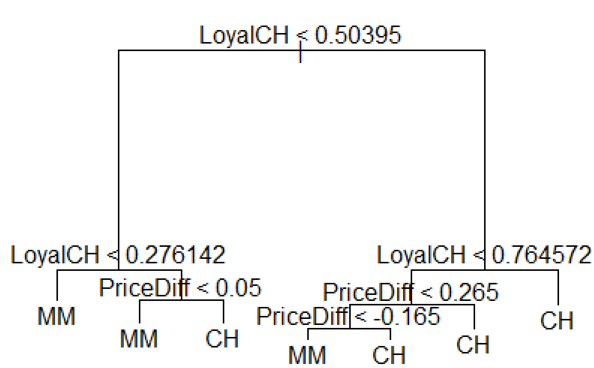
Part H: Which tree size corresponds to the lowest cross-validation error rate?

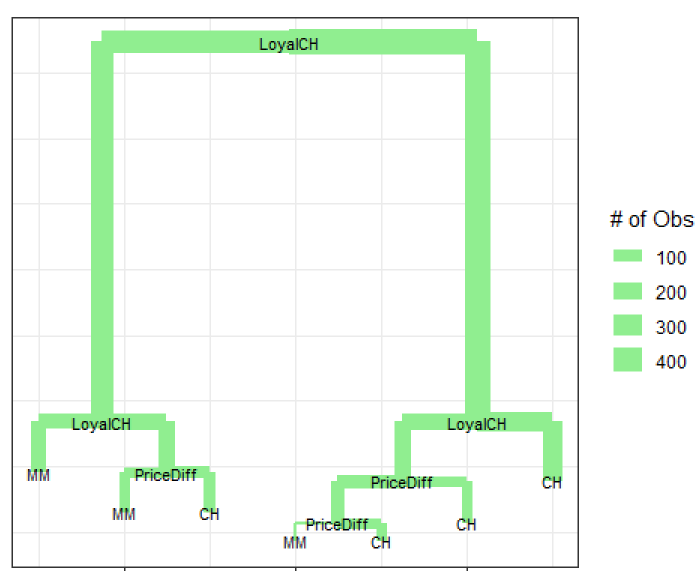
Results: A tree size with 7 terminal nodes gives us the lowest cross-validation error rate with the fewest nodes, I will include that in the subsequent exercises. I will also use 5 as an alternative number, since this number is given in *Part I* if cross-validation does not lend to a selection of a pruned tree. Technically, since 7 and 8 have the same error rate and 8 represents an unpruned tree, I think it will be beneficial to look at both options.

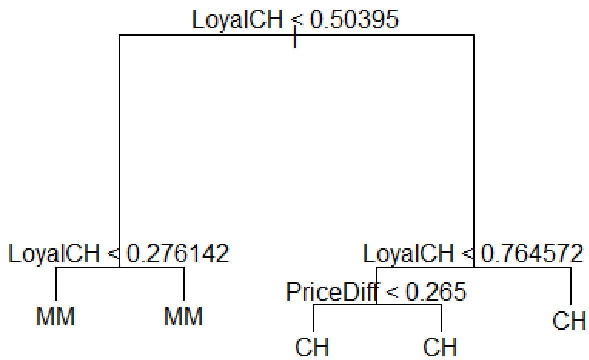
Part I: Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lend to selection of a pruned tree, then create a pruned tree with 5 terminal nodes.

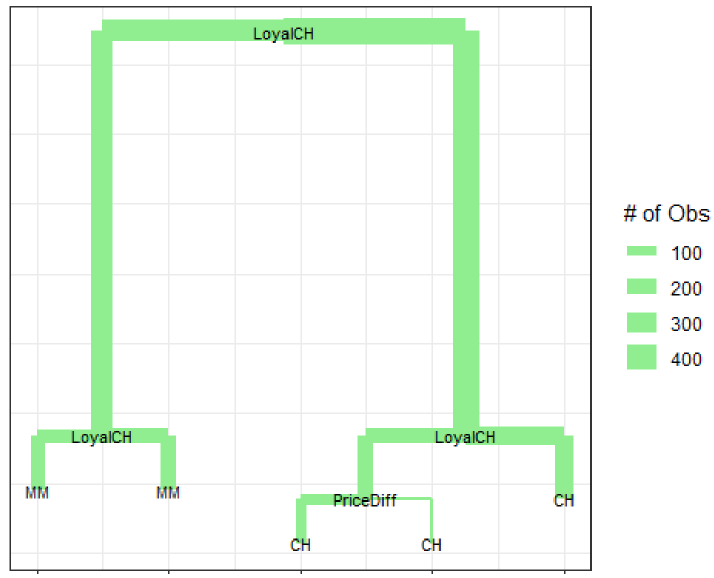
Results: Using best = 7, we see that 50% of the splits are based on **LoyalCH** - including the root, and 50% of the splits are derived from **PriceDiff.**

With best = 5, 75% of the splits are based on **LoyalCH** with only 25% being derived from **PriceDiff**. Additionally, from this set of 2 plots using best = 5, we see that if **LoyalCH** is >= 0.50395, it classifies **Purchase** as **CH**. Otherwise, it classifies as **MM.**









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

Results: As the below table indicates, there is no difference in the training error rates between the pruned tree at best = 7 and the unpruned tree. Both are **0.1662** with 133 of 800 obs being misclassified. When best = 5, the pruned tree performs slightly worse on the training data with an error rate of **0.2025** and 162 of 800 obs misclassified.

Error Rates on OJ Training Set

Original Tree	Pruned Tree - best = 7	Pruned Tree - best = 5
0.16625	0.16625	0.2025

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

Results: Here we see, once again, that the error rates of the unpruned tree and the pruned tree with best = 7 are identical at **0.1556** while the test error rate of the pruned tree with best = 5 lags behind slightly with a test error rate of **0.1926**.

Error Rates on OJ Test Set

Original Tree	Pruned Tree - best = 7	Pruned Tree - best = 5
0.1556	0.1556	0.1926

Question 8.4.10, pg 334: We now use boosting to predict **Salary** in the **Hitters** data set.

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

Results: First I removed the obs from the instructions and performed the log-transformation on the **Salary** response variable.

```
##                    AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits
## -Alan Ashby          315   81     7   24  38    39    14   3449   835
## -Alvin Davis         479  130    18   66  72    76     3   1624   457
## -Andre Dawson        496  141    20   65  78    37    11   5628  1575
## -Andres Galarraga    321   87    10   39  42    30     2    396   101
## -Alfredo Griffin     594  169     4   74  51    35    11   4408  1133
## -Al Newman           185   37     1   23   8    21     2    214    42
##                    CHmRun CRuns CRBI CWalks League Division PutOuts Assists
## -Alan Ashby            69   321  414    375      N        W     632      43
## -Alvin Davis           63   224  266    263      A        W     880      82
## -Andre Dawson         225   828  838    354      N        E     200      11
## -Andres Galarraga      12    48   46     33      N        E     805      40
## -Alfredo Griffin       19   501  336    194      A        W     282     421
## -Al Newman              1    30    9     24      N        E      76     127
##                    Errors   Salary NewLeague
## -Alan Ashby            10 6.163315         N
## -Alvin Davis           14 6.173786         A
## -Andre Dawson           3 6.214608         N
## -Andres Galarraga       4 4.516339         N
## -Alfredo Griffin       25 6.620073         A
## -Al Newman              7 4.248495         A
```

Part B: Create a training set consisting of the first 200 observations and a test set consisting of the remaining observations.

Results: I created the training and test sets per the instructions and printed the breakdown of number of obs per data set below.

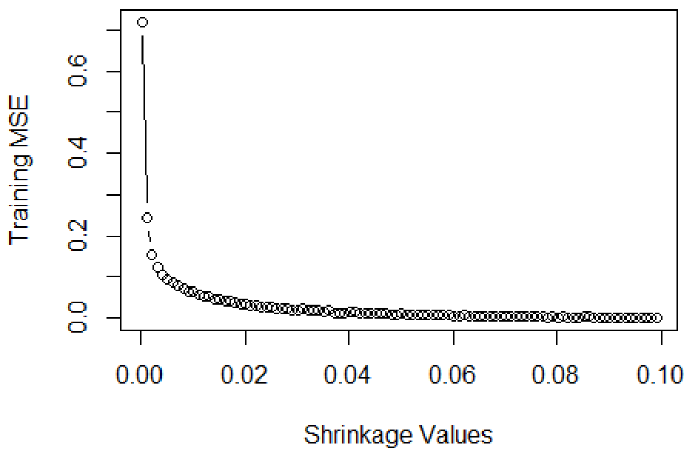
# of Obs

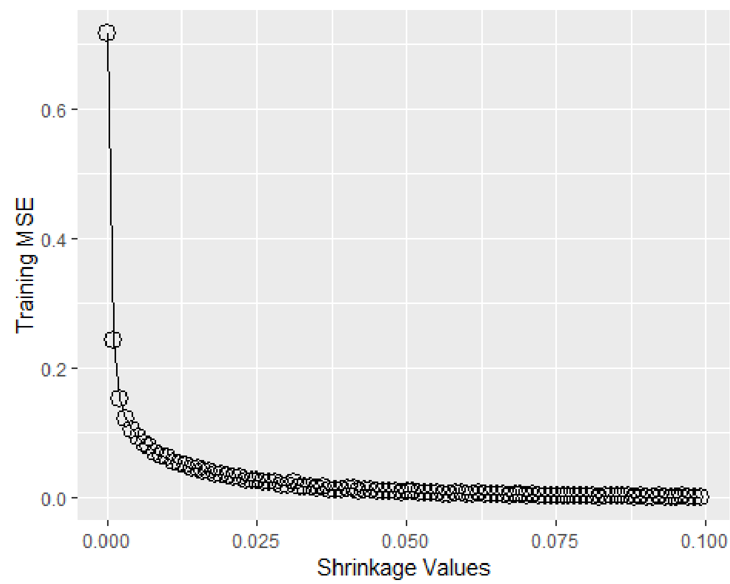
Hitters	Training	Test
263	200	63

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.

Results: First, I created a sequence of values to test as my lambda values in my loop. Next, I created a loop that iterated through the sequence of values and logged the MSE from the **gbm()** function at each iteration.

Below is a plot showing the MSE at these iterations. A complimentary ggplot is also shown. We can see that the MSE drops drastically before leveling off somewhat as we iterate through the shrinkage values.

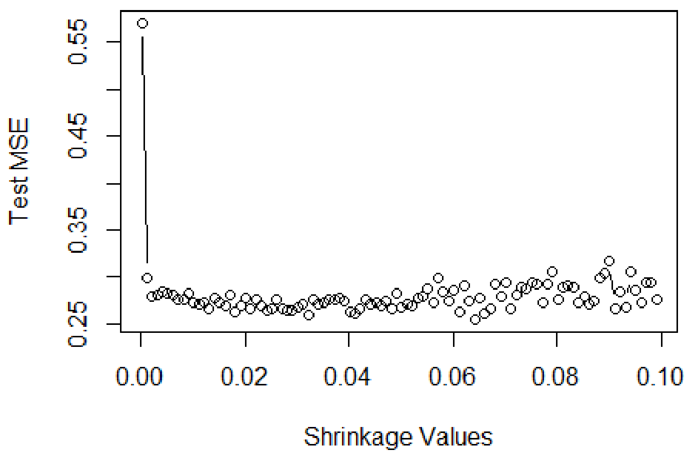


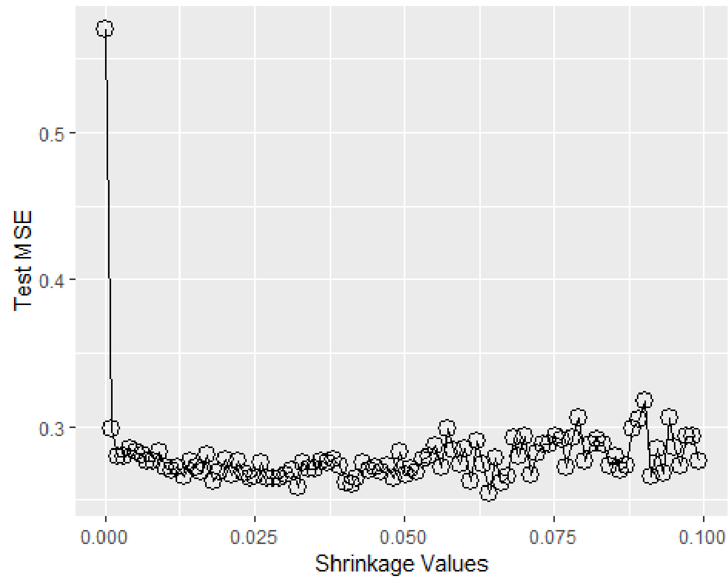


Part D: Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

Results: Below are the plots of Test MSE by shrinkage value. At first glance, we notice that the MSE values are much less uniform than they were in the training set plots above - which is to be expected. We still see a sharp decline initially, but then we can see some variation in the MSEs as the different shrinkage values are applied.

Lastly, we see that the best test error rate is **0.2547836** which occurs at a shrinkage value $\lambda = 0.0641$.





Boosting Test Error

Minimum Test Error	Corresponding Lambda
0.2547836	0.0641

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.

Results: For the two other regression approaches from previous chapters, I chose a simple linear model and ridge regression. As we can see from the table below, boosting gives us a better MSE at **0.2547836** than linear regression **(0.4917959)** or ridge regression **(0.4525826)**.

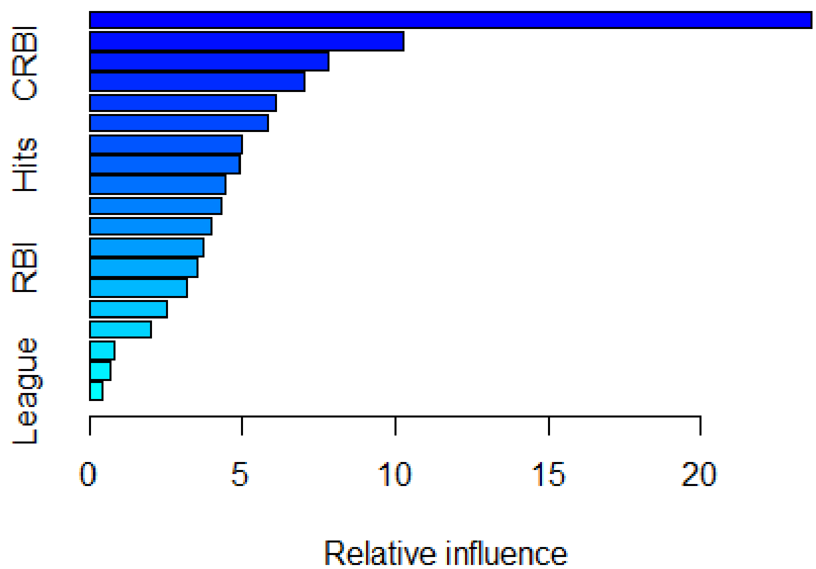
Test MSE by Model Type

Simple Linear Regression	Ridge Regression	Boosting
0.4917959	0.4525826	0.2547836

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

Results: From the table below, we can see that **CAtBat**, **CWalks**, **CRBI** and **PutOuts** are the most important predictors of **Salary** in our boosted model. Somewhat predictably, these correspond to "Number of times at bat during career", "Number of runs batter in during career", and "Number of walks during career", respectively.

The least important predictors are **League**, **Division**, and **NewLeague** which essentially all three correspond to which division of Major League Baseball you were in during the '86 and '87 seasons.




```
##                 var    rel.inf
## CAtBat       CAtBat 23.6157440
## CWalks       CWalks 10.2417824
## CRBI           CRBI  7.7784680
## PutOuts     PutOuts  7.0348328
## CRuns         CRuns  6.0949710
## Walks         Walks  5.8332024
## Years         Years  4.9652450
## Hits           Hits  4.8730551
## CHits         CHits  4.4422465
## Assists     Assists  4.2898792
## AtBat         AtBat  3.9496387
## CHmRun       CHmRun  3.7187804
## RBI             RBI  3.5017217
## HmRun         HmRun  3.2051445
## Runs           Runs  2.5381998
## Errors       Errors  2.0263545
## NewLeague NewLeague  0.8238692
## Division   Division  0.6621728
## League       League  0.4046919
```

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

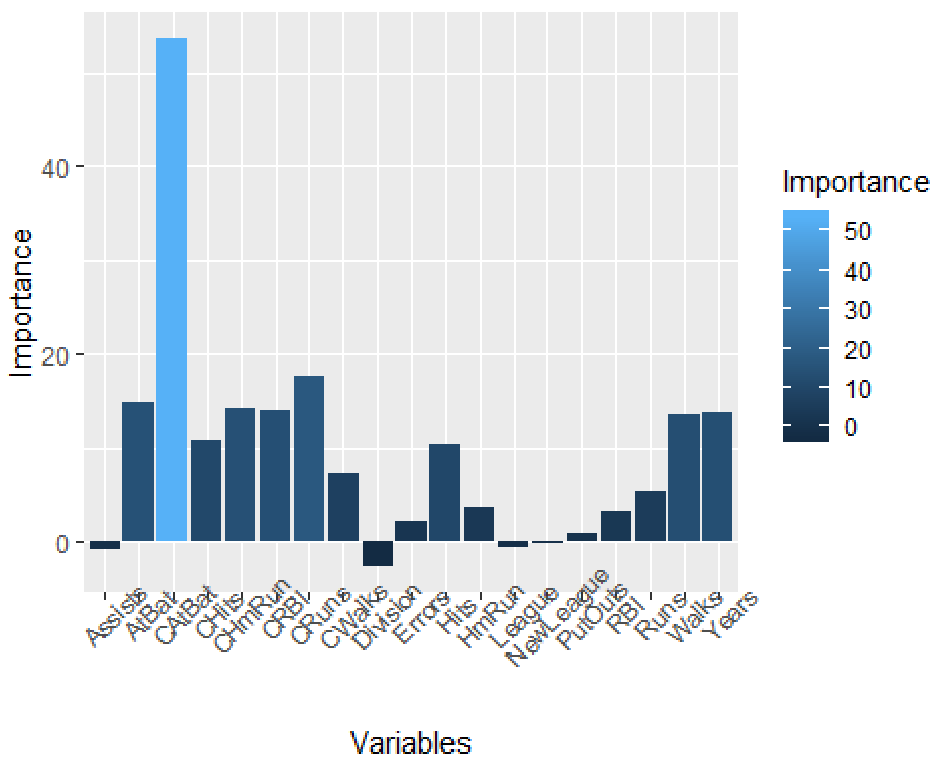
Results: The test set MSE for bagging is **0.2304351** which is slightly better than the boosting method **(0.2547836)**. Both bagging and boosting are considerably more effective than simple linear regression and ridge regression from chapters 3 and 6.

I also included an importance plot to compare with the results from the *Part F* of this exercise. Here we see that **CAtBat** is still, by far, the best predictor of **Salary**.

No base R plots are included since the question didn't request plotting.

Test MSE by Model Type

Simple Linear Regression	Ridge Regression	Boosting	Bagging
0.4917959	0.4525826	0.2547836	0.2304351



Question 5: In the past couple assignments you have usued different classification methods to analyze the dataset you chose. For this homework, use tree-based classification methods (tree, bagging, randomforest, boosting) to model your data. Find the test error using any/all methods (VSA, K-fold CV, LOOCV). Compare the results you obtained with the result from previous homework. Did the results improve? Use the table you previously made to compare.

Results: The table below represents the error rates compiled from previous assignments, joined with the error rates from the Tree, Bagging, Random Forest, and Boosting methods using the three different approaches - VSA, LOOCV, 5-Fold CV. Looking at the VSA column, we see that **Tree** method performed as well as the best methods from the prior homework assignments with test error rate of **12.0192%**. Bagging and Boosting performed poorly, relative to the other methods, with Bagging having the worst test error rate of any method under the VSA approach **(17.7885%)**.

In LOOCV, the Tree method again joined the best performing method from prior assignments (Logistic Regression and LDA) with an error rate of **14.4928%**. Random Forest also performed well, relative to the other methods. Bagging and Boosting performed poorly, relative to the other methods with Bagging having the highest test error rate of any method and any approach on the table at **20.4348%**.

In 5-Fold CV, we see that Bagging is by far the best method from the entire table with a test error rate of **10%**. Tree and Boosting also tied the previously best performing methods under the 5-Fold CV approach with an error rate of **14.4928%**. None of the new methods performed poorly, relative to the previous methods used.

Test Error by Validation Approach (%)

Method	VSA	LOOCV	5-Fold CV
Logistic Reg	12.0192	14.4928	14.4928
KNN	16.3462	18.6957	18.5507
LDA	12.0192	14.4928	14.4928
QDA	16.8269	17.3913	17.5362
MclustDA	16.8269	20	17.5362
MclustDA (EDDA)	16.8269	17.3913	17.5362
Neural Network	12.0192	14.6377	14.4928
Tree	12.0192	14.4928	14.4928
Bagging	17.7885	20.4348	10
Random Forest	13.9423	14.7826	14.7826
Boosting	16.8269	18.8406	14.4928