Rakesh Veetekat

Conceptual and theoretical questions

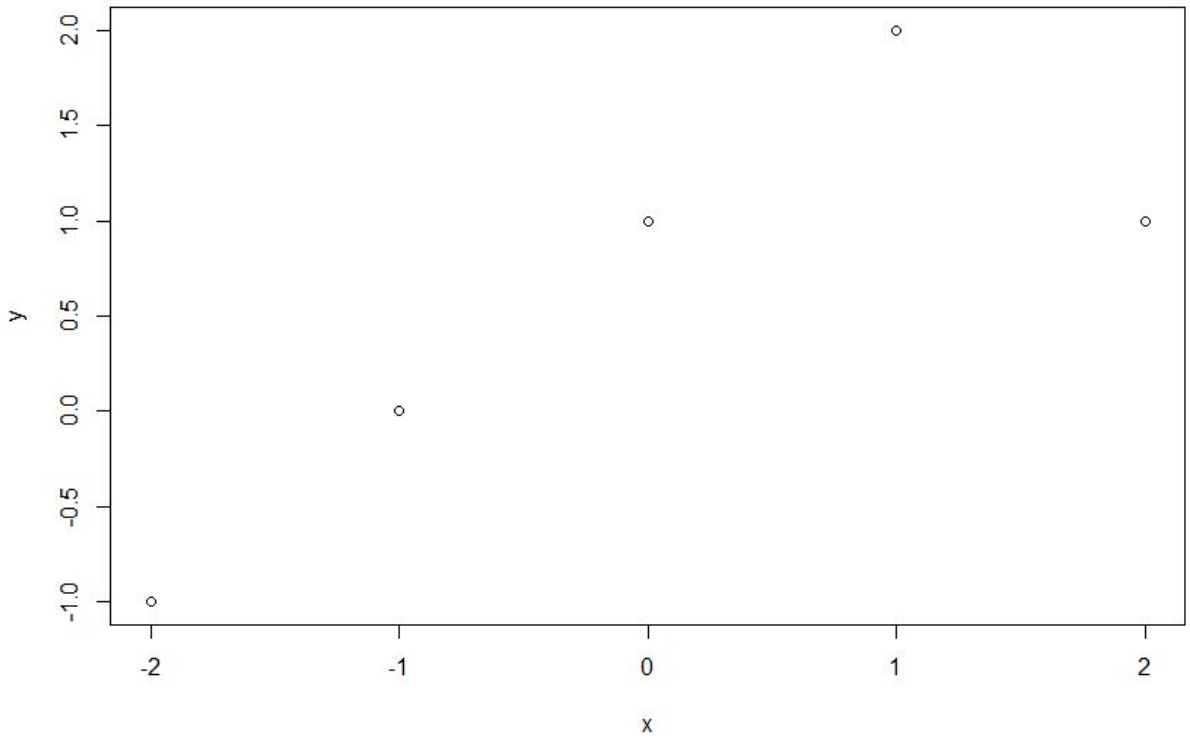
1. (Ch. 7, Question 3)

Suppose we fit a curve with basis functions $b_1(X) = X$, $b_2(X) = (X-1)^2 I(X \geq 1)$. (Note that $I(X \geq 1)$ equals 1 for $X \geq 1$ and 0 otherwise.) We fit the linear regression model

$$Y = \beta_0 + \beta_1 b_1(X) + \beta_2 b_2(X) + \epsilon,$$

and obtain coefficient estimates $\hat{\beta}_0 = 1, \hat{\beta}_1 = 1, \hat{\beta}_2 = -2$. Sketch the estimated curve between $X = -2$ and $X = 2$. Note the intercepts, slopes, and other relevant information.

```
> x = -2:2
> y = 1 + x + -2 * (x-1)^2 * I(x>1)
> plot(x, y)
>
```



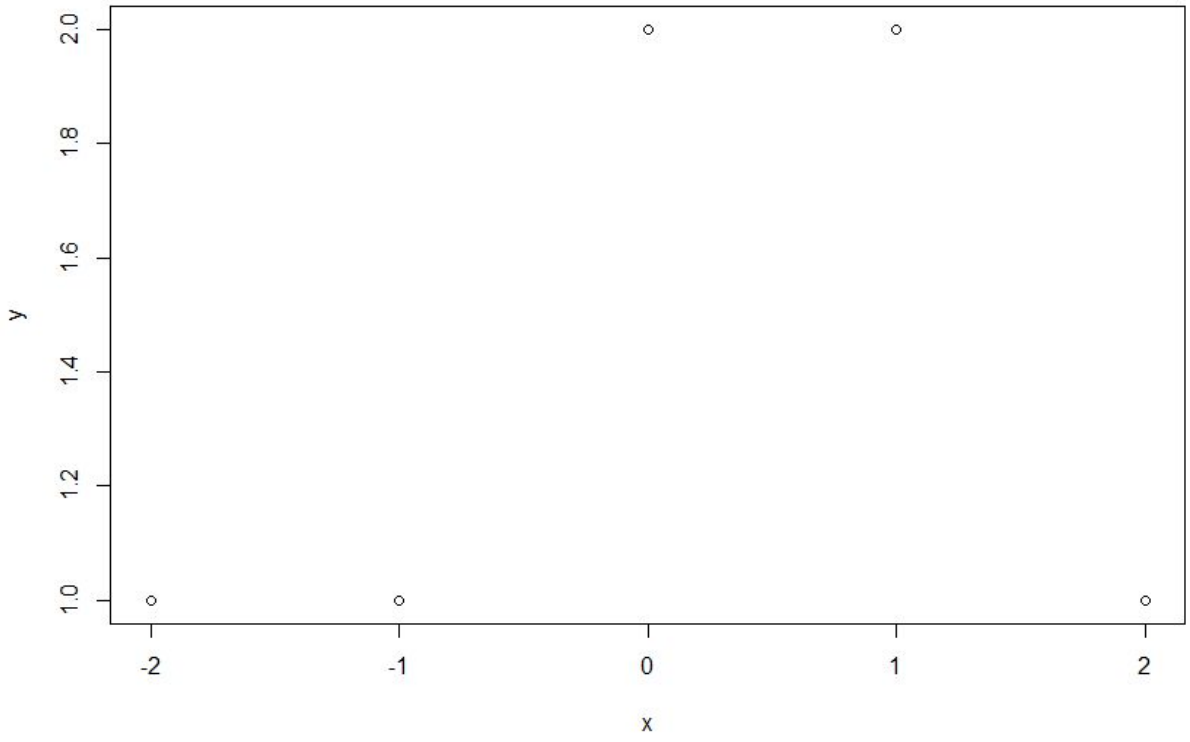
2. (Ch. 7, Question 4)

Suppose we fit a curve with basis functions $b_1(X) = I(0 \leq X \leq 2) - (X-1)I(1 \leq X \leq 2)$, $b_2(X) = (X-3)I(3 \leq X \leq 4) + I(4 < X \leq 5)$. We fit the linear regression model

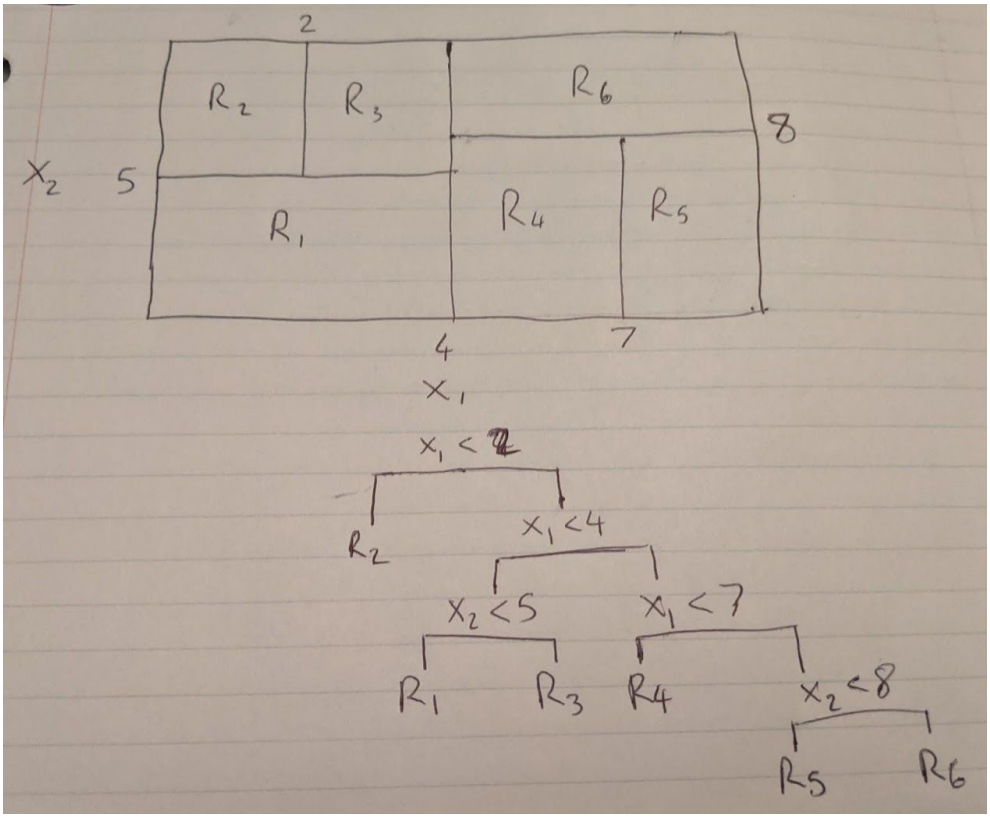
$$Y = \beta_0 + \beta_1 b_1(X) + \beta_2 b_2(X) + \epsilon,$$

and obtain coefficient estimates $\hat{\beta}_0 = 1, \hat{\beta}_1 = 1, \hat{\beta}_2 = 3$. Sketch the estimated curve between $X = -2$ and $X = 2$. Note the intercepts, slopes, and other relevant information.

```
> x = -2:2
> y = c(1 + 0 + 0, 1 + 0 + 0, 1 + 1 + 0, 1 + (1-0) + 0, 1 + (1-1) + 0)
> plot(x, y)
>
```



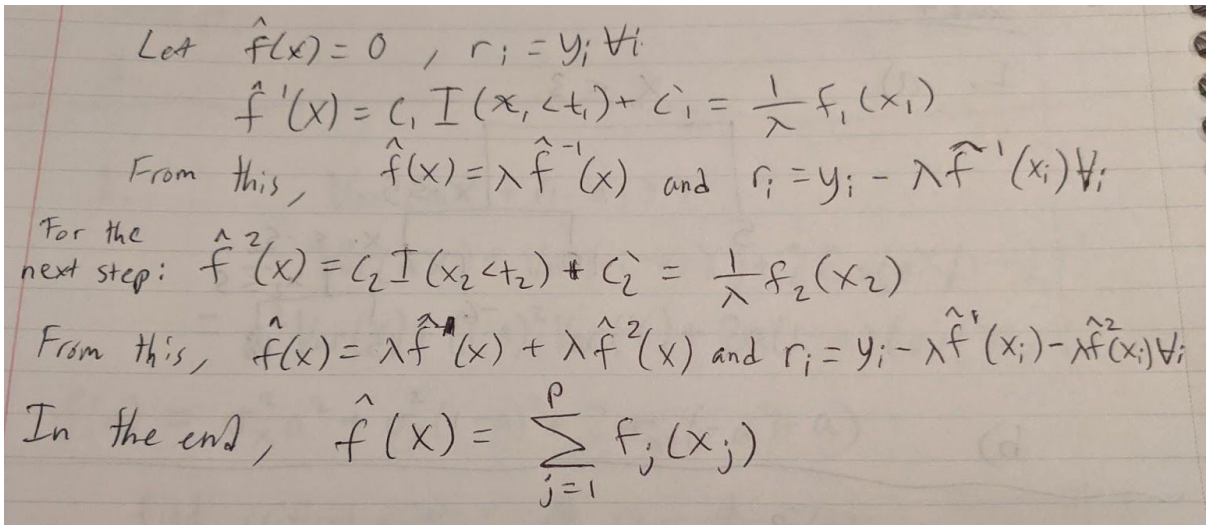
3. (Ch. 8, Question 1)
   Draw an example (of your own invention) of a partition of twodimensional feature space that could result from recursive binary splitting. Your example should contain at least six regions. Draw a decision tree corresponding to this partition. Be sure to label all aspects of your figures, including the regions R1, R2,..., the cutpoints t1, t2,..., and so forth.
   Hint: Your result should look something like Figures 8.1 and 8.2.



4. (Ch. 8, Question 2)
   It is mentioned in Section 8.2.3 that boosting using depth-one trees (or stumps) leads to an additive model: that is, a model of the form

$$f(X) = \sum_{j=1}^{p} f_j(X_j).$$

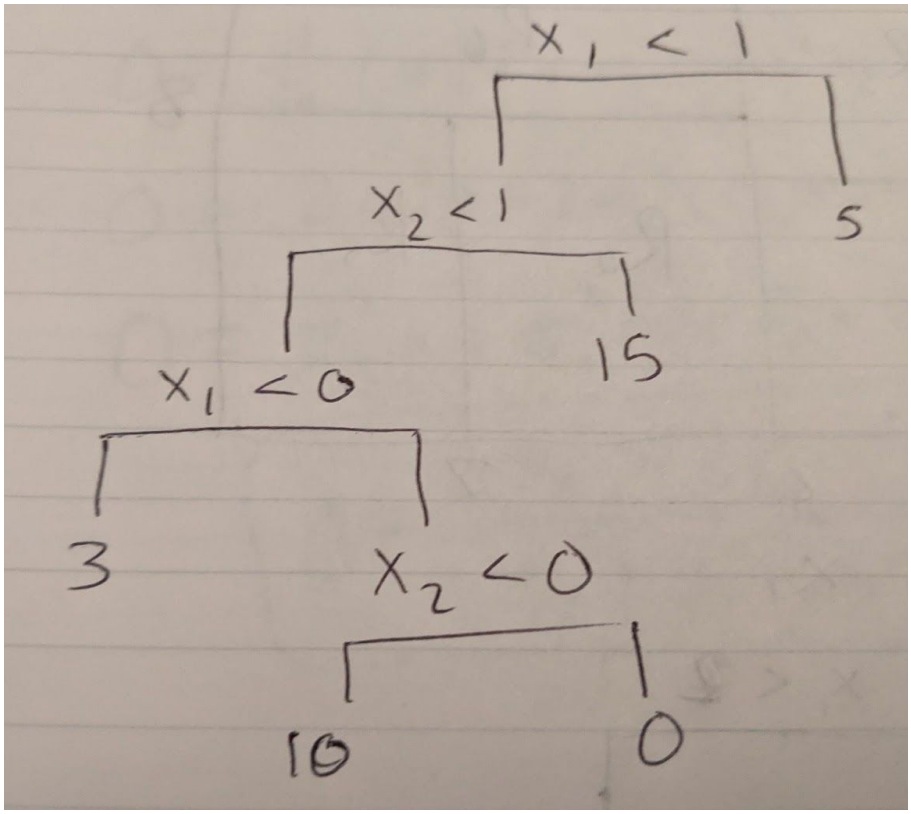
   Explain why this is the case. You can begin with (8.12) in Algorithm 8.2.



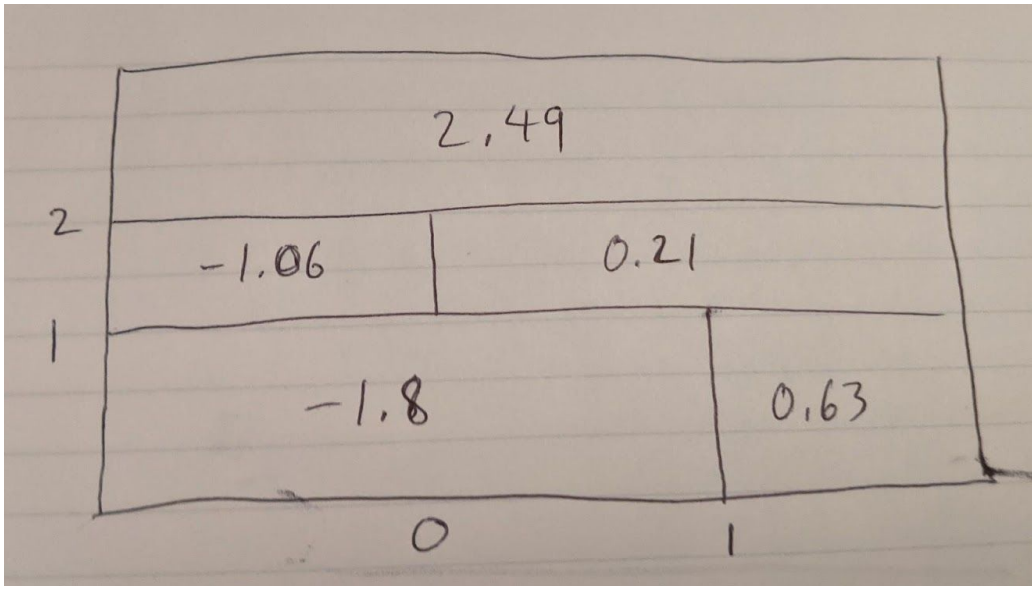
5. (Ch. 8, Question 4)

This question relates to the plots in Figure 8.12.

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.



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.



6. (Ch. 8, Question 5)
   Suppose we produce ten bootstrapped samples from a data set containing red and green classes. We then apply a classification tree to each bootstrapped sample and, for a specific value of X, produce 10 estimates of P(Class is Red|X):

   0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, and 0.75.

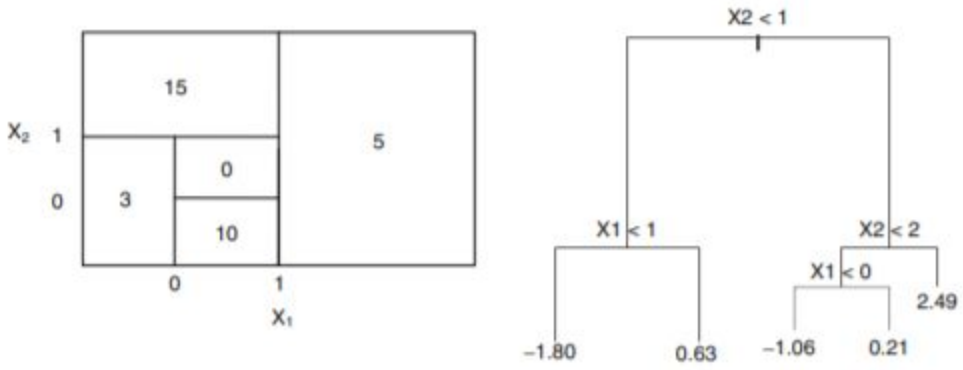


FIGURE 8.12. Left: *A partition of the predictor space corresponding to Exercise 4a.* Right: *A tree corresponding to Exercise 4b.*

There are two common ways to combine these results together into a single class prediction. One is the majority vote approach discussed in this chapter. The second approach is to classify based on the average probability. In this example, what is the final classification under each of these two approaches?

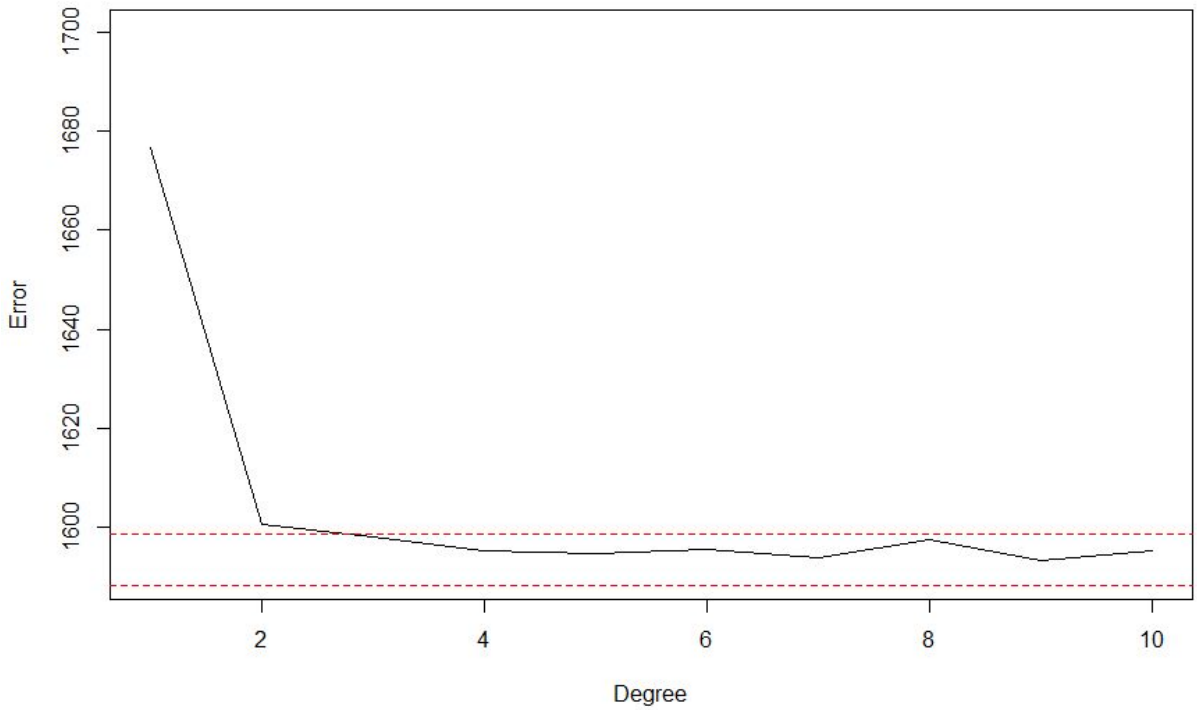
Using the majority vote approach, the final classification would be Red because there are 6 predictions for Red versus 4 for Green.

Using the average probability approach, the final classification would be Green because the average of the 10 probabilities is 0.45, which means the average leans towards Green rather than Red.

Applied

7. (Ch. 7, Question 6)
   In this exercise, you will further analyze the Wage data set considered throughout this chapter.
   a. Perform polynomial regression to predict wage using age. Use cross-validation to select the optimal degree d for the polynomial. What degree was chosen, and how does this compare to the results of hypothesis testing using ANOVA? Make a plot of the resulting polynomial fit to the data.

```
> set.seed(1)
> library(ISLR)
> library(boot)
> all.deltas = rep(NA, 10)
> for (i in 1:10) {
+   glm.fit = glm(wage~poly(age, i), data=Wage)
+   all.deltas[i] = cv.glm(Wage, glm.fit, K=10)$delta[2]
+ }
> plot(1:10, all.deltas, xlab="Degree", ylab="Error", type="l", ylim=c(1590, 1700))
> min.point = min(all.deltas)
> sd.points = sd(all.deltas)
> abline(h=min.point + 0.2 * sd.points, col="red", lty="dashed")
> abline(h=min.point - 0.2 * sd.points, col="red", lty="dashed")
>
```



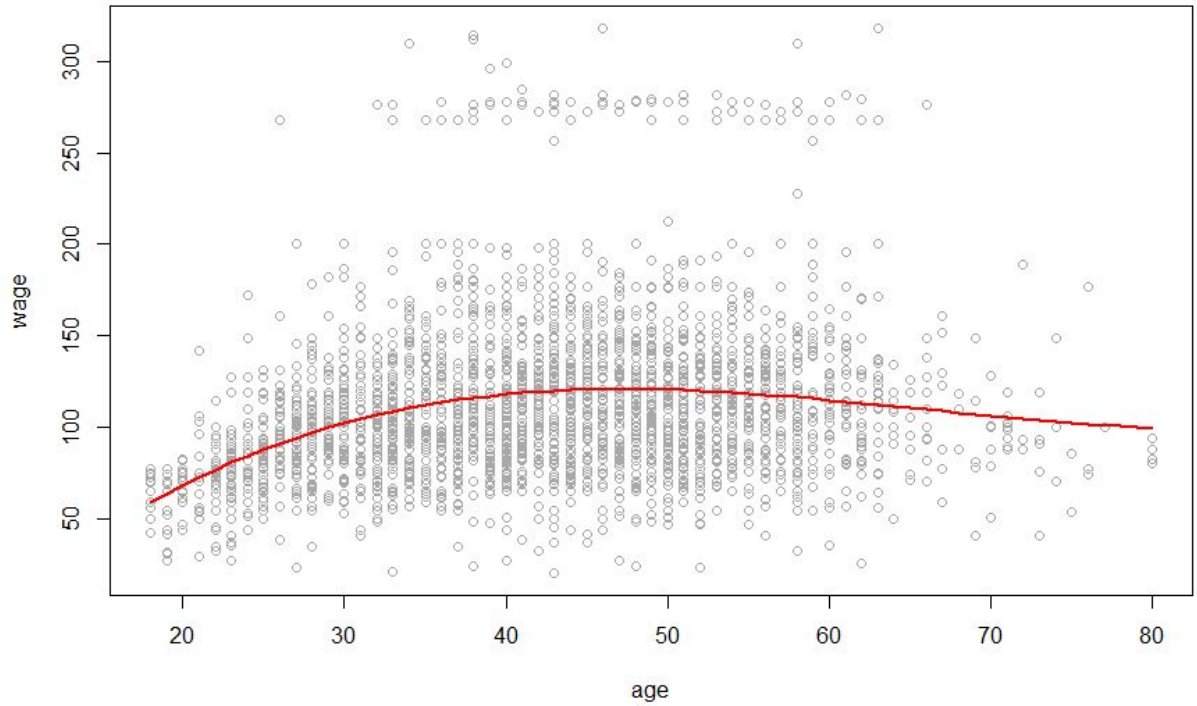
Using K-fold cross validation with K=10, the optimal degree d for the polynomial can be obtained from the plot above where the error is within the standard deviation threshold, and the point where the plot first crosses into the threshold is when the degree is 3.

```
> anova(fit.1, fit.2, fit.3, fit.4, fit.5, fit.6, fit.7, fit.8, fit.9, fit.10)
Analysis of Variance Table

Model  1: wage ~ poly(age, 1)
Model  2: wage ~ poly(age, 2)
Model  3: wage ~ poly(age, 3)
Model  4: wage ~ poly(age, 4)
Model  5: wage ~ poly(age, 5)
Model  6: wage ~ poly(age, 6)
Model  7: wage ~ poly(age, 7)
Model  8: wage ~ poly(age, 8)
Model  9: wage ~ poly(age, 9)
Model 10: wage ~ poly(age, 10)
   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
1    2998 5022216
2    2997 4793430  1    228786 143.7638 < 2.2e-16 ***
3    2996 4777674  1     15756   9.9005  0.001669 **
4    2995 4771604  1      6070   3.8143  0.050909 .
5    2994 4770322  1      1283   0.8059  0.369398
6    2993 4766389  1      3932   2.4709  0.116074
7    2992 4763834  1      2555   1.6057  0.205199
8    2991 4763707  1       127   0.0796  0.777865
9    2990 4756703  1      7004   4.4014  0.035994 *
10   2989 4756701  1         3   0.0017  0.967529
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
```

Using ANOVA on all the polynomials with varying degrees shows that all of the degrees above 4 are highly insignificant with large p-values.

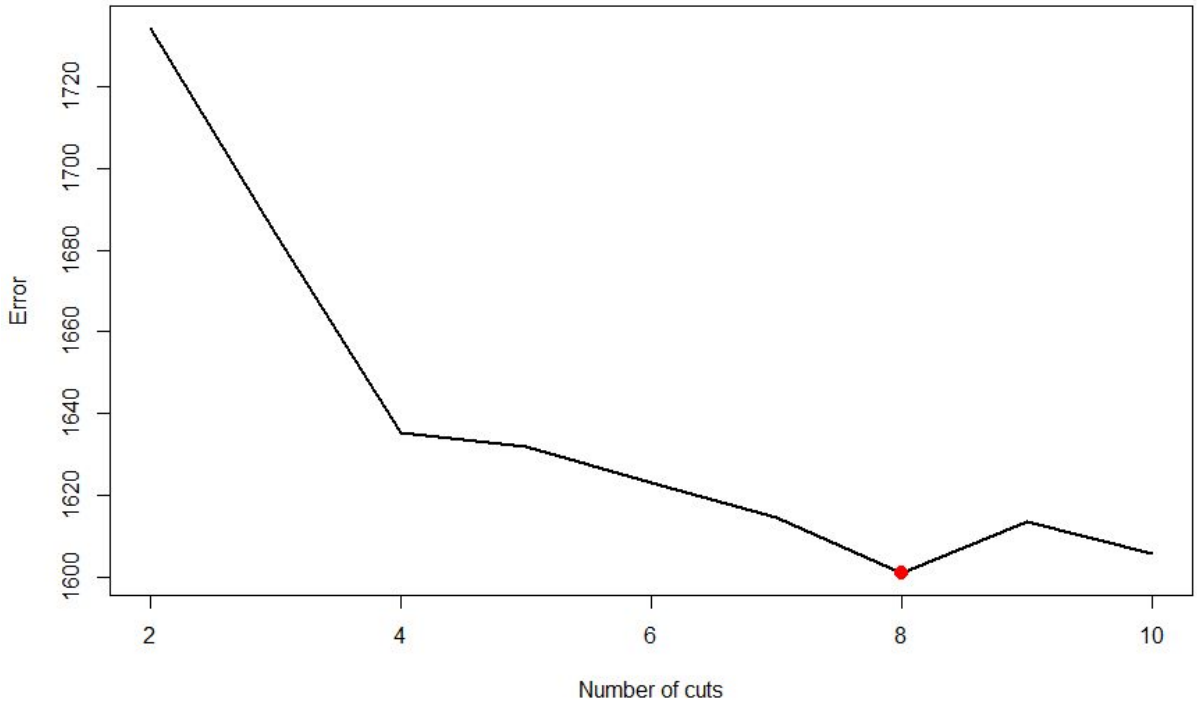
With the results from the K-fold cross validation, the polynomial will be fit to the data with a degree of 3.

```
> plot(wage ~ age, data = Wage, col = "darkgrey")
> agelims = range(Wage$age)
> age.grid = seq(from = agelims[1], to = agelims[2])
> lm.fit = lm(wage ~ poly(age, 3), data = Wage)
> lm.pred = predict(lm.fit, data.frame(age=age.grid))
> lines(age.grid, lm.pred, col = "red", lwd = 2)
>
```



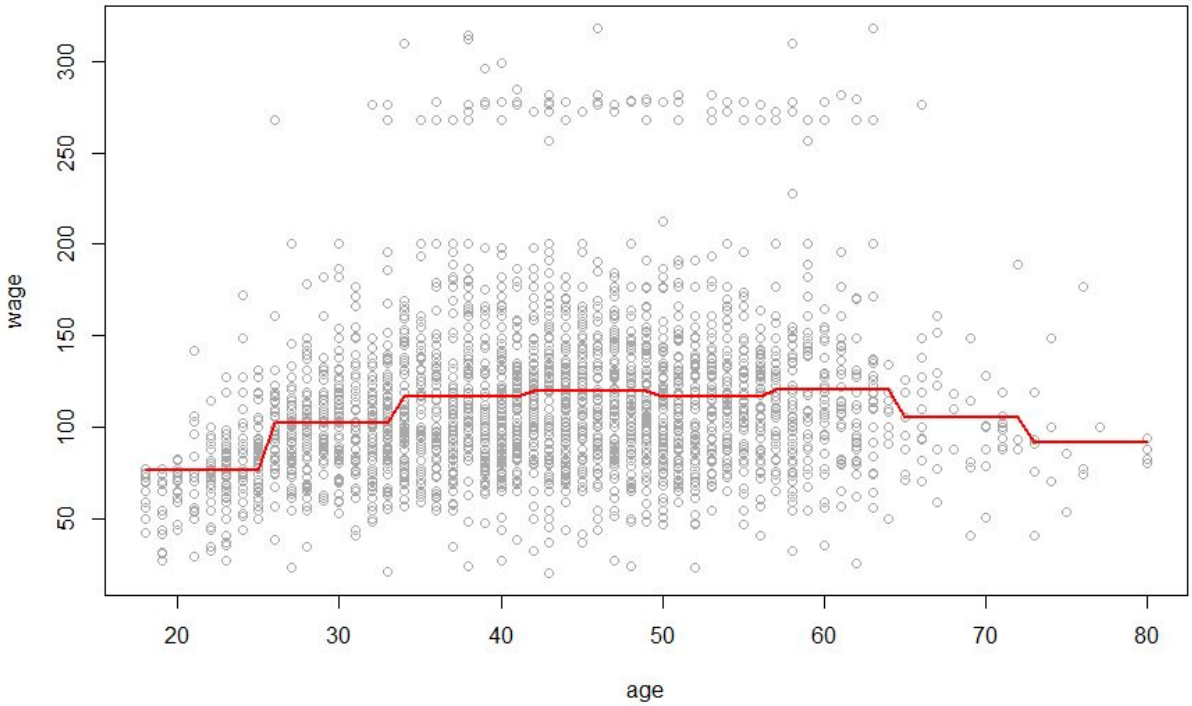
b. Fit a step function to predict wage using age, and perform crossvalidation to choose the optimal number of cuts. Make a plot of the fit obtained.

```
> set.seed(1)
> library(ISLR)
> library(boot)
>
> cvs = rep(NA, 10)
> for (i in 2:10) {
+   Wage$age.cut = cut(Wage$age, i)
+   lm.fit = glm(wage~age.cut, data=Wage)
+   cvs[i] = cv.glm(Wage, lm.fit, K=10)$delta[2]
+ }
> plot(2:10, cvs[-1], xlab="Number of cuts", ylab="CV error", type="l", pch=20, lwd=2)
> min = which.min(cvs)
> points(min, cvs[min], col = "red", cex = 2, pch = 20)
>
```



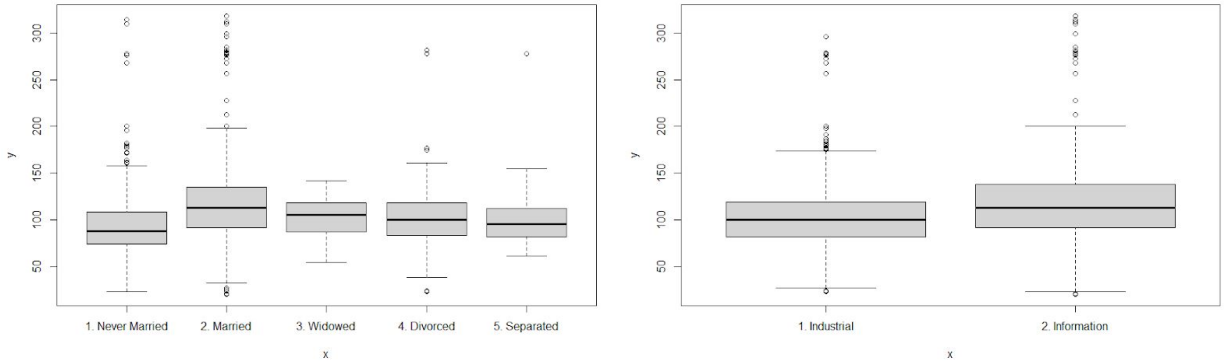
Performing cross validation found that using 8 cuts will minimize the test error. The step function will now be run using 8 cuts.

```
> lm.fit = glm(wage ~ cut(age, 8), data = Wage)
> agelims = range(Wage$age)
> age.grid = seq(from = agelims[1], to = agelims[2])
> lm.pred = predict(lm.fit, data.frame(age = age.grid))
> plot(wage ~ age, data = Wage, col = "darkgrey")
> lines(age.grid, lm.pred, col = "red", lwd=2)
>
```



8. (Ch. 7, Question 7)
   The Wage data set contains a number of other features not explored in this chapter, such as marital status (maritl), job class (jobclass), and others. Explore the relationships between some of these other predictors and wage, and use non-linear fitting techniques in order to fit flexible models to the data. Create plots of the results obtained, and write a summary of your findings.

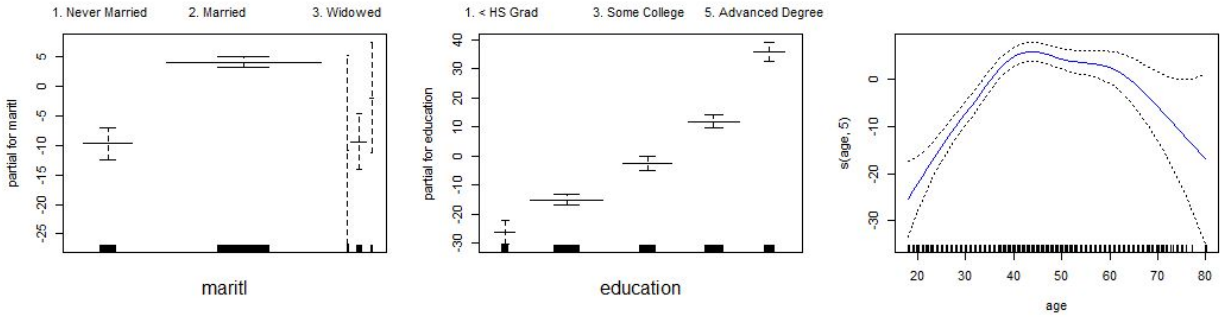
```
> library(ISLR)
> set.seed(1)
>
> par(mfrow = c(1, 2))
> plot(wage$maritl, wage$wage)
> plot(wage$jobclass, wage$wage)
>
```



From the boxplots on the left set of plots, it can be seen that married couples should make more money on average as opposed to the other categories of individuals. The two boxplots on the right show that jobs in the information sector appear to make more on average than industrial jobs.

```
> fit1 = gam(wage ~ maritl + s(age, 5), data = Wage)
> fit2 = gam(wage ~ maritl + jobclass + s(age, 5), data = Wage)
> fit3 = gam(wage ~ maritl + education + s(age, 5), data = Wage)
> fit4 = gam(wage ~ maritl + jobclass + education + s(age, 5), data = Wage)
> anova(fit1, fit2, fit3, fit4)
Analysis of Deviance Table

Model 1: wage ~ maritl + s(age, 5)
Model 2: wage ~ maritl + jobclass + s(age, 5)
Model 3: wage ~ maritl + education + s(age, 5)
Model 4: wage ~ maritl + jobclass + education + s(age, 5)
  Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
1      2990    4642806
2      2989    4472687  1   170119 < 2.2e-16 ***
3      2986    3618678  3   854009 < 2.2e-16 ***
4      2985    3604891  1    13786 0.0007283 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
```

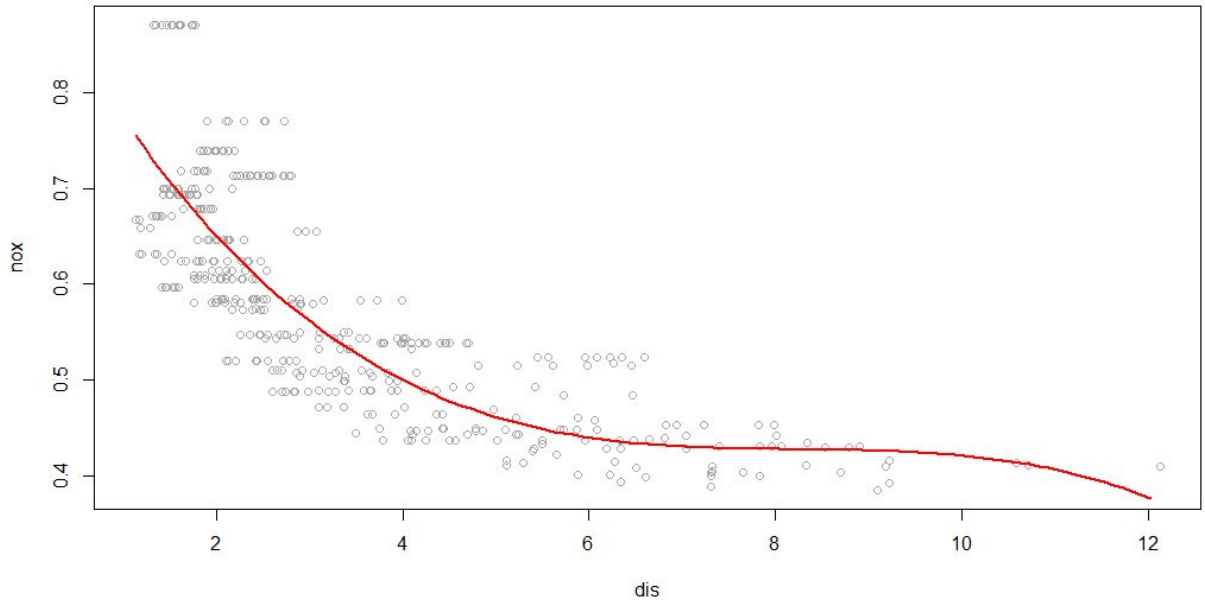
Both fit2 and fit3 have more statistically significant p-values than fit4 so those two will be plotted.



9. (Ch. 7, Question 9)
   This question uses the variables dis (the weighted mean of distances to five Boston employment centers) and nox (nitrogen oxides concentration in parts per 10 million) from the Boston data. We will treat dis as the predictor and nox as the response.
   a. Use the poly() function to fit a cubic polynomial regression to predict nox using dis. Report the regression output, and plot the resulting data and polynomial fits.

```
> library(MASS)
> set.seed(1)
> attach(Boston)
>
> lm.fit = lm(nox ~ poly(dis, 3), data = Boston)
> summary(lm.fit)

Call:
lm(formula = nox ~ poly(dis, 3), data = Boston)

Residuals:
      Min        1Q    Median        3Q       Max
-0.121130 -0.040619 -0.009738  0.023385  0.194904

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.554695   0.002759 201.021  < 2e-16 ***
poly(dis, 3)1 -2.003096   0.062071 -32.271  < 2e-16 ***
poly(dis, 3)2  0.856330   0.062071  13.796  < 2e-16 ***
poly(dis, 3)3 -0.318049   0.062071  -5.124 4.27e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.06207 on 502 degrees of freedom
Multiple R-squared:  0.7148,    Adjusted R-squared:  0.7131
F-statistic: 419.3 on 3 and 502 DF,  p-value: < 2.2e-16

> |
```

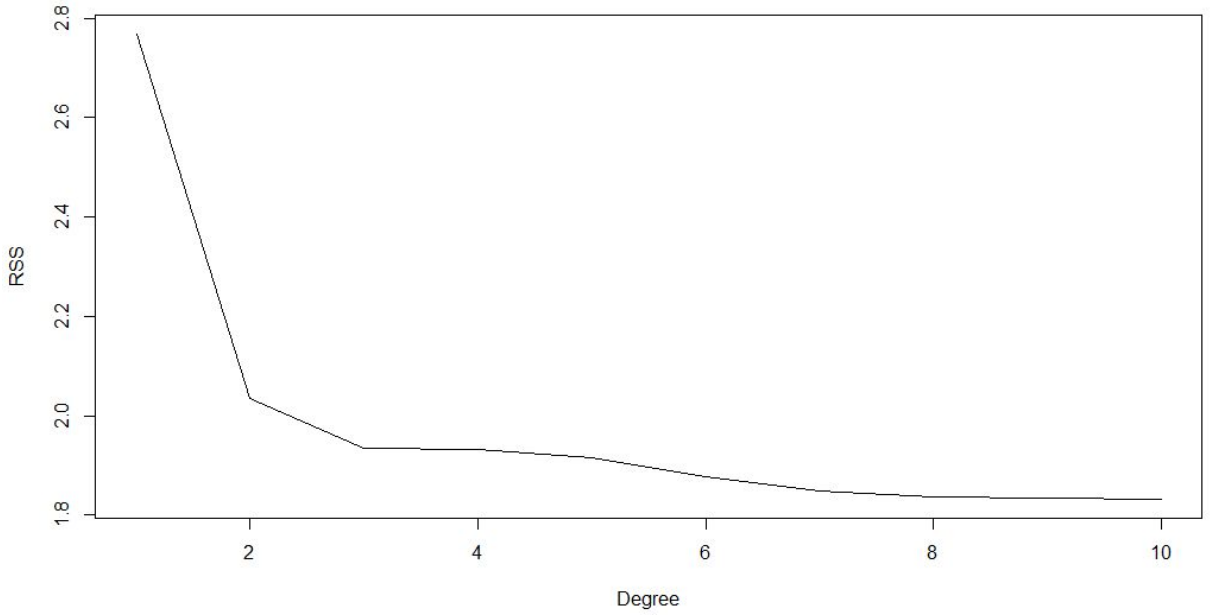
```
> library(MASS)
> set.seed(1)
> attach(Boston)
>
> lm.fit = lm(nox ~ poly(dis, 3), data = Boston)
> # summary(lm.fit)
>
> dislim = range(dis)
> dis.grid = seq(min(Boston$dis), max(Boston$dis), by = 0.1)
> lm.pred = predict(lm.fit, list(dis = dis.grid))
> plot(nox ~ dis, data = Boston, col = "darkgrey")
> lines(dis.grid, lm.pred, col = "red", lwd = 2)
> |
```



From both the summary and plot, it appears all of the polynomial terms are significant.

b. Plot the polynomial fits for a range of different polynomial degrees (say, from 1 to 10), and report the associated residual sum of squares.

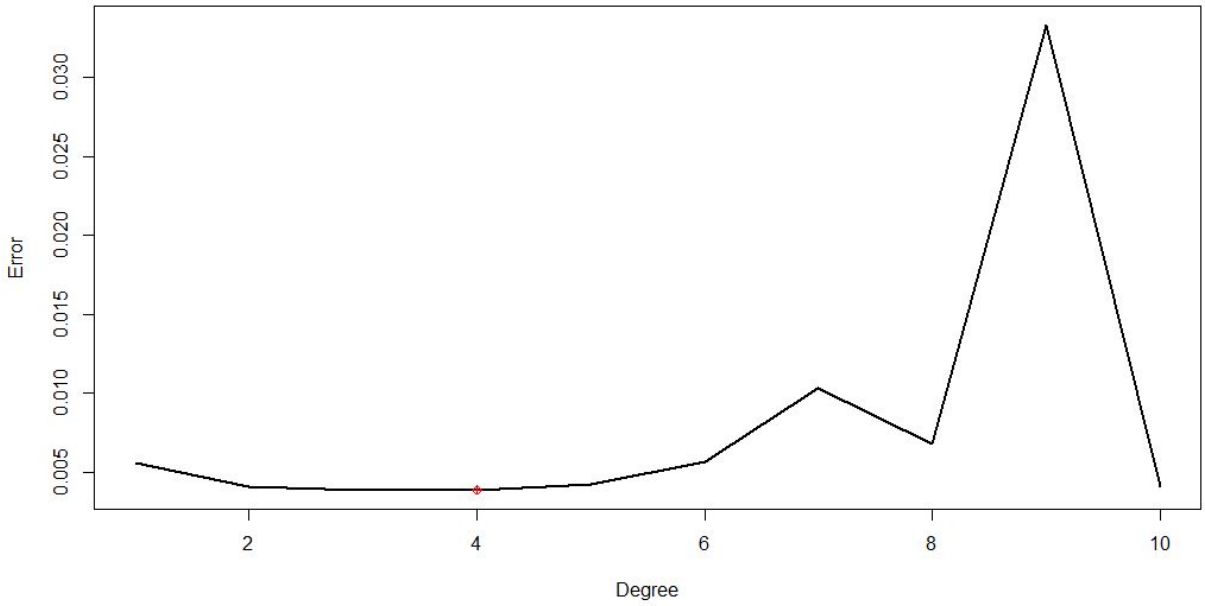
```
> rss = rep(NA, 10)
> for (i in 1:10) {
+   lm.fit = lm(nox ~ poly(dis, i), data = Boston)
+   rss[i] = sum(lm.fit$residuals^2)
+ }
> rss
 [1] 2.768563 2.035262 1.934107 1.932981 1.915290 1.878257 1.849484 1.835630 1.833331 1.832171
> plot(1:10, rss, type = 'l', xlab = "Degree", ylab = "RSS")
> |
```



The associated RSS is printed out in the snippet of code provided before the plot, and both the values seen there and the plot show that the RSS is decreasing as degree increases.

c. Perform cross-validation or another approach to select the optimal degree for the polynomial, and explain your results.

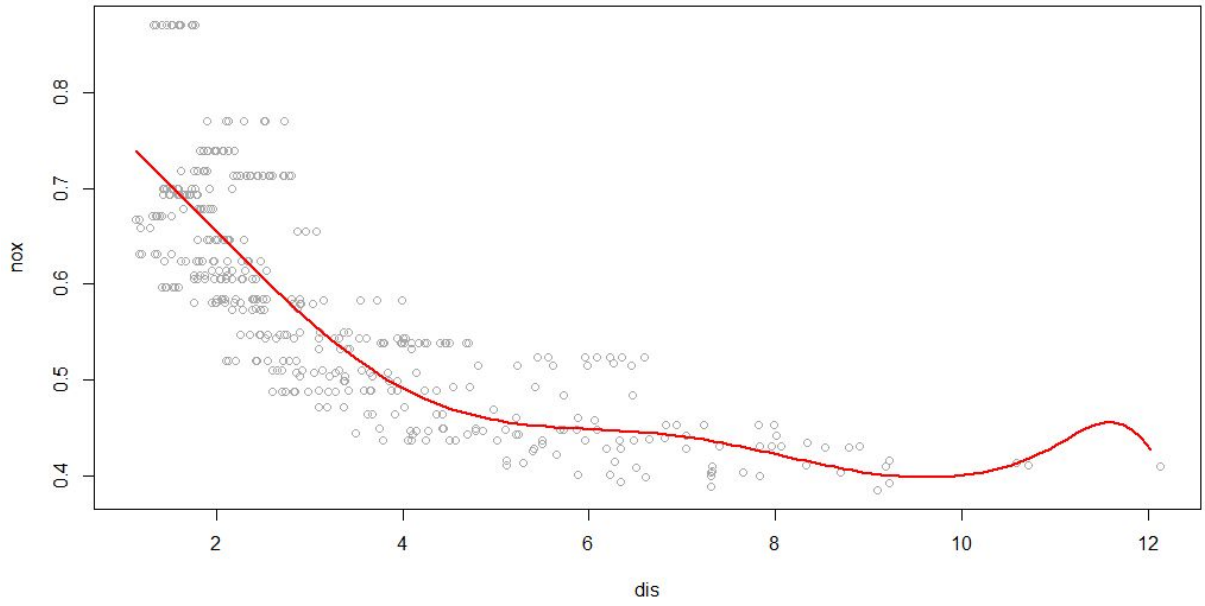
```
> library(boot)
> deltas = rep(NA, 10)
> for (i in 1:10) {
+   glm.fit = glm(nox ~ poly(dis, i), data = Boston)
+   deltas[i] = cv.glm(Boston, glm.fit, K = 10)$delta[1]
+ }
> plot(1:10, deltas, xlab = "Degree", ylab = "Error", type = "l", pch = 20, lwd = 2)
> min = which.min(deltas)
> points(min, deltas[min], col = 'red', pch = 10)
>
```



A degree of 4 should be selected because it minimizes the test error.

d. Use the bs() function to fit a regression spline to predict nox using dis. Report the output for the fit using four degrees of freedom. How did you choose the knots? Plot the resulting fit.

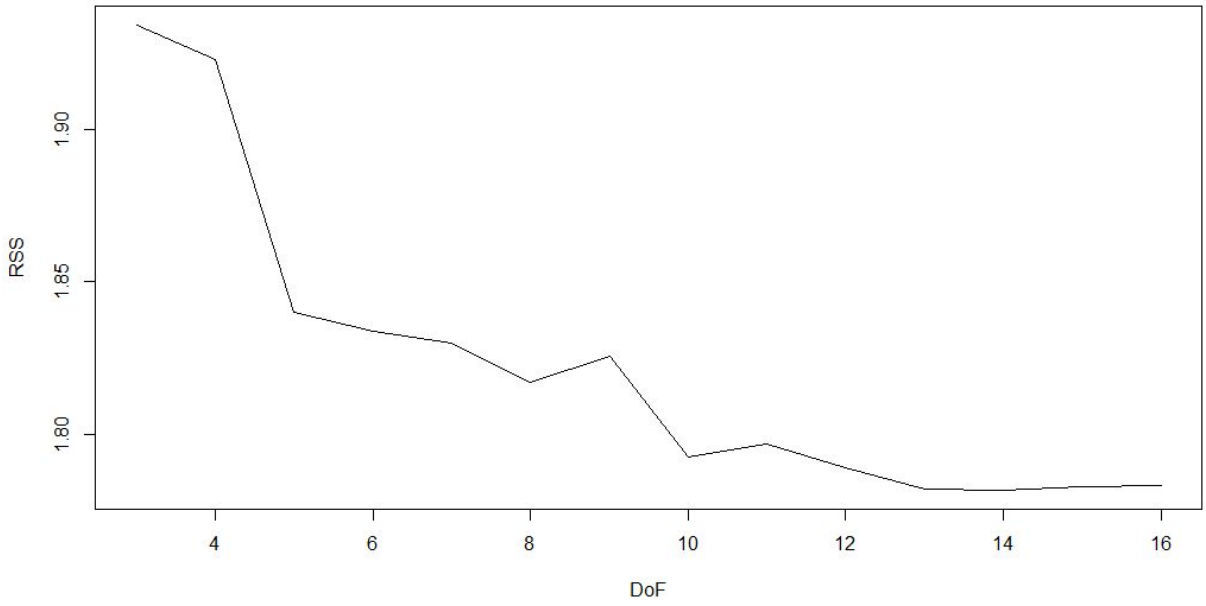
```
> library(MASS)
> set.seed(1)
> attach(Boston)
>
> library(splines)
> sp.fit = lm(nox ~ bs(dis, df = 4, knots = c(4, 7, 11)), data = Boston)
>
> sp.pred = predict(sp.fit, list(dis = dis.grid))
> plot(nox ~ dis, data = Boston, col = "darkgrey")
> lines(dis.grid, sp.pred, col = "red", lwd = 2)
>
```



The knots were chosen by splitting the dis into quartiles given the four degrees of freedom. The resulting fit is shown above.

e. Now fit a regression spline for a range of degrees of freedom, and plot the resulting fits and report the resulting RSS. Describe the results obtained.

```
> all.cv = rep(NA, 16)
> for (i in 3:16) {
+   lm.fit = lm(nox ~ bs(dis, df = i), data = Boston)
+   all.cv[i] = sum(lm.fit$residuals^2)
+ }
> plot(3:16, all.cv[-c(1, 2)], type = 'l', xlab = 'DoF', ylab = 'RSS')
>
```



From the plot above, it can be seen that the training RSS decreases until 14 degrees of freedom, and then starts increasing from there.

f. Perform cross-validation or another approach in order to select the best degrees of freedom for a regression spline on this data. Describe your results.

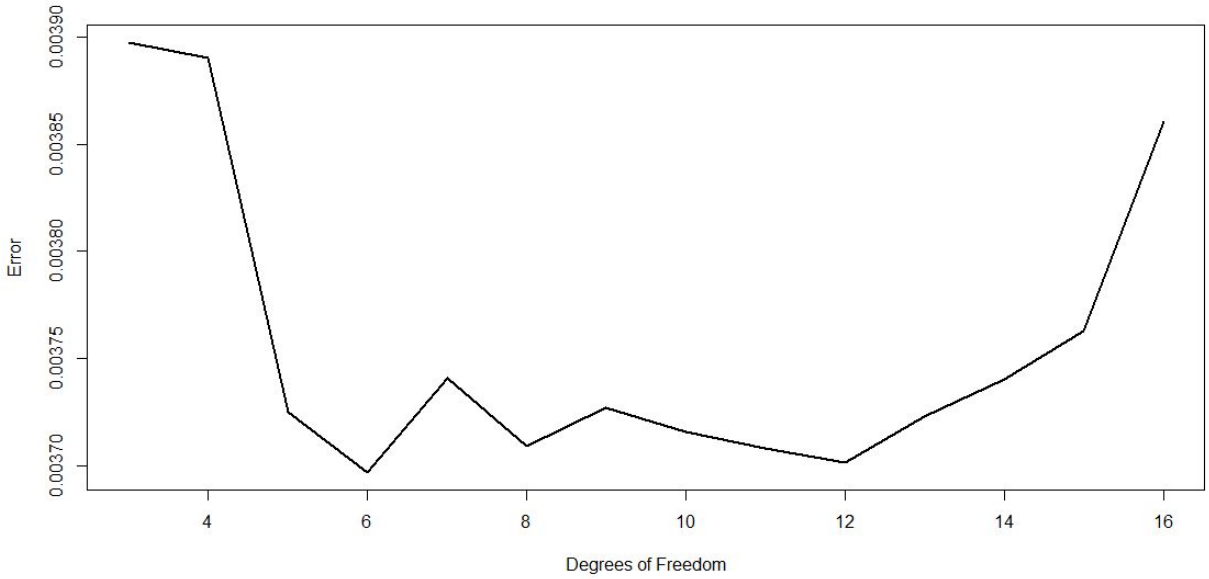
```
> all.cv = rep(NA, 16)
> for (i in 3:16) {
+   lm.fit = glm(nox ~ bs(dis, df = i), data = Boston)
+   all.cv[i] = cv.glm(Boston, lm.fit, K = 10)$delta[1]
+ }
There were 50 or more warnings (use warnings() to see the first 50)
```

```
> warnings()
Warning messages:
1: In bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.137,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
2: In bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.137,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
3: In bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.1296,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
4: In bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.1296,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
5: In bs(dis, degree = 3L, knots = c(`50%` = 3.0993), Boundary.knots = c(1.137,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
6: In bs(dis, degree = 3L, knots = c(`50%` = 3.0993), Boundary.knots = c(1.137,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
7: In bs(dis, degree = 3L, knots = c(`50%` = 3.3603), Boundary.knots = c(1.1296,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
8: In bs(dis, degree = 3L, knots = c(`50%` = 3.3603), Boundary.knots = c(1.1296,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
9: In bs(dis, degree = 3L, knots = c(`33.33333%` = 2.38876666666667,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
10: In bs(dis, degree = 3L, knots = c(`33.33333%` = 2.38876666666667,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
11: In bs(dis, degree = 3L, knots = c(`33.33333%` = 2.3088,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
12: In bs(dis, degree = 3L, knots = c(`33.33333%` = 2.3088,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
13: In bs(dis, degree = 3L, knots = c(`25%` = 2.087875, `50%` = 3.19095,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
14: In bs(dis, degree = 3L, knots = c(`25%` = 2.087875, `50%` = 3.19095,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
15: In bs(dis, degree = 3L, knots = c(`20%` = 1.92404, `40%` = 2.55946,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
16: In bs(dis, degree = 3L, knots = c(`20%` = 1.92404, `40%` = 2.55946,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
17: In bs(dis, degree = 3L, knots = c(`20%` = 1.94984, `40%` = 2.59774,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
18: In bs(dis, degree = 3L, knots = c(`20%` = 1.94984, `40%` = 2.59774,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
19: In bs(dis, degree = 3L, knots = c(`16.66667%` = 1.86636666666667,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
20: In bs(dis, degree = 3L, knots = c(`16.66667%` = 1.86636666666667,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
21: In bs(dis, degree = 3L, knots = c(`16.66667%` = 1.82085,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
22: In bs(dis, degree = 3L, knots = c(`16.66667%` = 1.82085,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
23: In bs(dis, degree = 3L, knots = c(`14.28571%` = 1.79078571428571,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
24: In bs(dis, degree = 3L, knots = c(`14.28571%` = 1.79078571428571,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
25: In bs(dis, degree = 3L, knots = c(`14.28571%` = 1.7912,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
26: In bs(dis, degree = 3L, knots = c(`14.28571%` = 1.7912,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
27: In bs(dis, degree = 3L, knots = c(`12.5%` = 1.757275,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
28: In bs(dis, degree = 3L, knots = c(`12.5%` = 1.757275,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
29: In bs(dis, degree = 3L, knots = c(`12.5%` = 1.76375, `25%` = 2.10525,  ...
  some 'x' values beyond boundary knots may cause ill-conditioned bases
30: In bs(dis, degree = 3L, knots = c(`12.5%` = 1.76375, `25%` = 2.10525,  ...
  some 'x' values beyond boundary knots may cause ill-conditioned bases
31: In bs(dis, degree = 3L, knots = c(`11.11111%` = 1.69124444444444,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
32: In bs(dis, degree = 3L, knots = c(`11.11111%` = 1.69124444444444,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
33: In bs(dis, degree = 3L, knots = c(`11.11111%` = 1.71297777777778,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
34: In bs(dis, degree = 3L, knots = c(`11.11111%` = 1.71297777777778,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
35: In bs(dis, degree = 3L, knots = c(`10%` = 1.66236, `20%` = 1.98518,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
36: In bs(dis, degree = 3L, knots = c(`10%` = 1.66236, `20%` = 1.98518,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
37: In bs(dis, degree = 3L, knots = c(`10%` = 1.6624, `20%` = 1.9769,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
38: In bs(dis, degree = 3L, knots = c(`10%` = 1.6624, `20%` = 1.9769,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
39: In bs(dis, degree = 3L, knots = c(`9.090909%` = 1.59007272727273,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
40: In bs(dis, degree = 3L, knots = c(`9.090909%` = 1.59007272727273,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
41: In bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61941818181818,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
42: In bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61941818181818,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
43: In bs(dis, degree = 3L, knots = c(`8.333333%` = 1.58948333333333,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
44: In bs(dis, degree = 3L, knots = c(`8.333333%` = 1.58948333333333,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
45: In bs(dis, degree = 3L, knots = c(`8.333333%` = 1.5874,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
46: In bs(dis, degree = 3L, knots = c(`8.333333%` = 1.5874,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
47: In bs(dis, degree = 3L, knots = c(`7.692308%` = 1.57254615384615,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
48: In bs(dis, degree = 3L, knots = c(`7.692308%` = 1.57254615384615,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
49: In bs(dis, degree = 3L, knots = c(`7.692308%` = 1.57254615384615,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
50: In bs(dis, degree = 3L, knots = c(`7.692308%` = 1.57254615384615,  ... :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
> 
```

Plotted the cross validation error against the degrees of freedom.

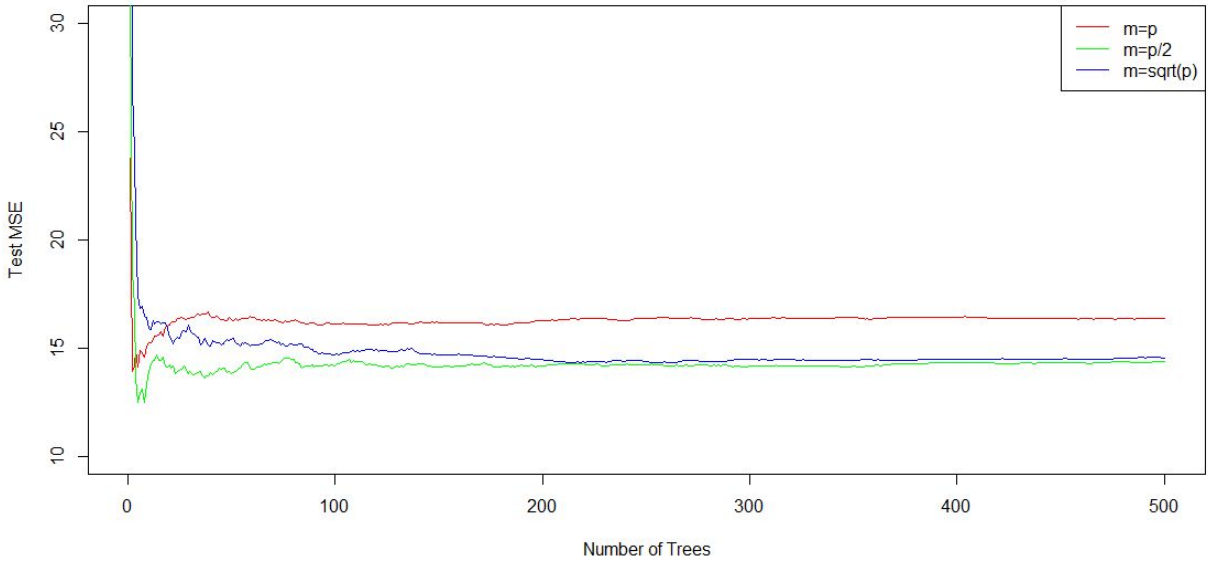
```
> plot(3:16, all.cv[-c(1, 2)], lwd = 2, type = "l", xlab = "Degrees of Freedom", ylab = "Er
ror")
> 
```



From the graph, it can be seen that the error is minimized with 6 degrees of freedom.

10. (Ch. 8, Question 7)
    In the lab, we applied random forests to the Boston data using mtry=6 and using ntree=25 and ntree=500. Create a plot displaying the test error resulting from random forests on this data set for a more comprehensive range of values for mtry and ntree. You can model your plot after Figure 8.10. Describe the results obtained.

```
> library(MASS)
> library(randomForest)
> set.seed(100)
>
> # Split the data into training and testing sets
> train = sample(dim(Boston)[1], dim(Boston)[1]/2)
> Boston.train = Boston[train, -14]
> Boston.test = Boston[-train, -14]
> Y.train = Boston[train, 14]
> Y.test = Boston[-train, 14]
>
> p = dim(Boston)[2] - 1
> p.2 = p/2
> p.sq = sqrt(p)
>
> rf.boston.p = randomForest(Boston.train, Y.train, xtest = Boston.test, ytest = Y.test, mtry = p, ntree = 500)
> rf.boston.p.2 = randomForest(Boston.train, Y.train, xtest = Boston.test, ytest = Y.test, mtry = p.2, ntree = 500)
> rf.boston.p.sq = randomForest(Boston.train, Y.train, xtest = Boston.test, ytest = Y.test, mtry = p.sq, ntree = 500)
>
> plot(1:500, rf.boston.p$test$mse, col = "red", type = "l", xlab = "Number of Trees", ylab = "Test MSE", ylim = c(10, 30))
> lines(1:500, rf.boston.p.2$test$mse, col = "green", type = "l")
> lines(1:500, rf.boston.p.sq$test$mse, col = "blue", type = "l")
> legend("topright", c("m=p", "m=p/2", "m=sqrt(p)"), col = c("red", "green", "blue"), cex = 1, lty = 1)
>
```



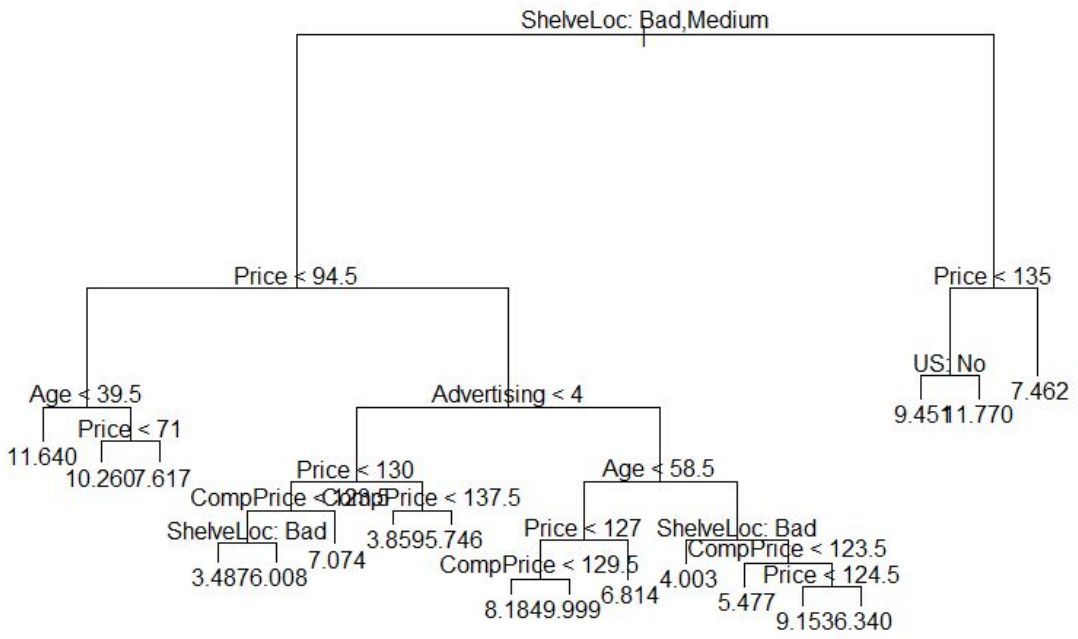
From the plot above, it can be seen that The test error for all three functions decreases as the number of trees in the random forest increases, and the error eventually stabilizes at a certain point for each one. The graph shows that m=p/2 and m=sqrt(p) are very close in displaying the minimum test error in the plot, but it appears that m=p/2 has the slight edge.

11. (Ch. 8, Question 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.
    a. Split the data set into a training set and a test set.

```
> library(ISLR)
> attach(Carseats)
> set.seed(1)
>
> train = sample(dim(Carseats)[1], dim(Carseats)[1]/2)
> Carseats.train = Carseats[train, ]
> Carseats.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?

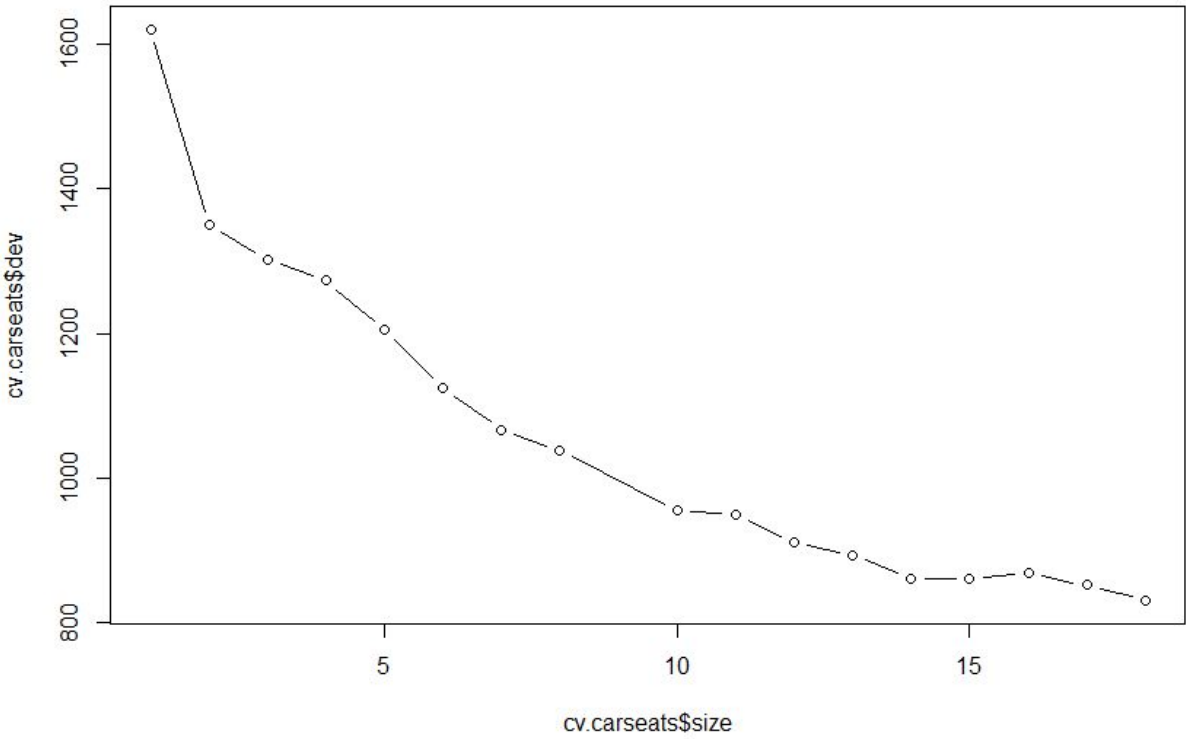
```
> library(ISLR)
> attach(Carseats)
> set.seed(1)
>
> train = sample(dim(Carseats)[1], dim(Carseats)[1]/2)
> Carseats.train = Carseats[train, ]
> Carseats.test = Carseats[-train, ]
>
> library(tree)
> tree.carseats = tree(Sales ~ ., Carseats.train)
> summary(tree.carseats)

Regression tree:
tree(formula = Sales ~ ., data = Carseats.train)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Age"         "Advertising" "CompPrice"   "US"
Number of terminal nodes:  18
Residual mean deviance:  2.167 = 394.3 / 182
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-3.88200 -0.88200 -0.08712  0.00000  0.89590  4.09900
>
> plot(tree.carseats)
> text(tree.carseats, pretty = 0)
>
> pred.carseats = predict(tree.carseats, Carseats.test)
> mean((Carseats.test$Sales - pred.carseats)^2)
[1] 4.922039
>
```



The test MSE of the tree 4.922039. The summary statistics and visual interpretation of the tree is shown in both images above.

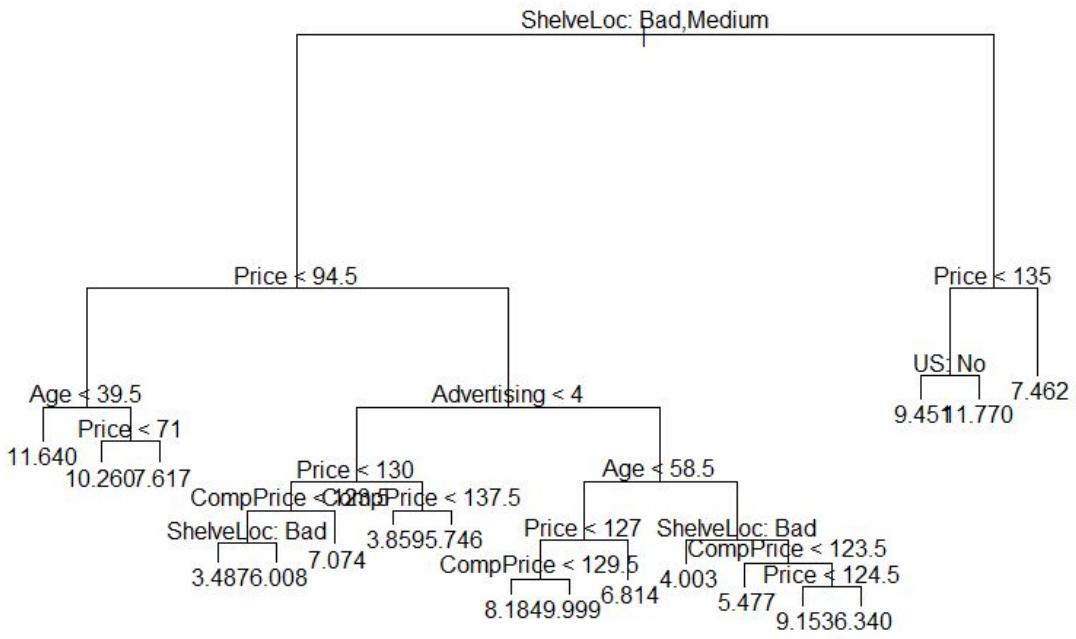
c. Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

```
> cv.carseats = cv.tree(tree.carseats)
> plot(cv.carseats$size,cv.carseats$dev,type = "b")
>
```



The tree of size 18 is the best. The tree will be pruned to obtain the 18-node tree.

```
> pruned.carseats = prune.tree(tree.carseats, best = 18)
> par(mfrow = c(1, 1))
> plot(pruned.carseats)
> text(pruned.carseats, pretty = 0)
>
> pred.pruned = predict(pruned.carseats, Carseats.test)
> mean((Carseats.test$Sales - pred.pruned)^2)
[1] 4.922039
>
```



The test MSE of the pruned tree stayed the same as what the tree had originally, most likely because the best size for the pruned tree was the size of the carseats data.

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)
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.
>
> bag.carseats = randomForest(Sales ~ ., data = Carseats.train, mtry = 10, ntree = 500, importance = T)
> importance(bag.carseats)
              %IncMSE IncNodePurity
CompPrice   24.6476262     168.57576
Income       5.3355144      90.64911
Advertising 11.7975748     100.81807
Population  -2.4347577      58.94510
Price       55.1387362     500.32765
ShelveLoc   46.3295341     376.78200
Age         17.9245890     162.68705
Education    0.8706811      43.04402
Urban        0.6850649       8.77639
US           4.3005762      17.96535
>
> bag.pred = predict(bag.carseats, Carseats.test)
> mean((Carseats.test$Sales - bag.pred)^2)
[1] 2.623527
>
```

With the bagging approach, the test MSE improves to 2.623527. Price and ShelveLoc appear to be the most significant predictors for Sales.

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.

```
> rf.carseats = randomForest(Sales ~ ., data = Carseats.train, mtry = 5, ntree = 500, importance = T)
> importance(rf.carseats)
               %IncMSE IncNodePurity
CompPrice   18.5120376     157.47383
Income       2.3560015     113.01270
Advertising 10.1445005     103.10978
Population  -2.6485481      79.72088
Price       48.3795294     447.67646
ShelveLoc   39.5323099     339.58317
Age         13.9400225     168.64176
Education    1.8601520      53.91689
Urban       -0.5242288      11.31567
US           5.9599681      28.27057
>
> rf.pred = predict(rf.carseats, Carseats.test)
> mean((Carseats.test$Sales - rf.pred)^2)
[1] 2.690746
>
```

Using random forests to analyze the data results in a worse test MSE than the bagging approach because it increases to 2.690746. The most significant predictors for Sales are still Price and ShelveLoc, although the weight on those two predictors lessens in the random forest approach as opposed to the bagging approach.

12. (Ch. 8, Question 10)
    We now use boosting to predict Salary in the Hitters data set.
    a. Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

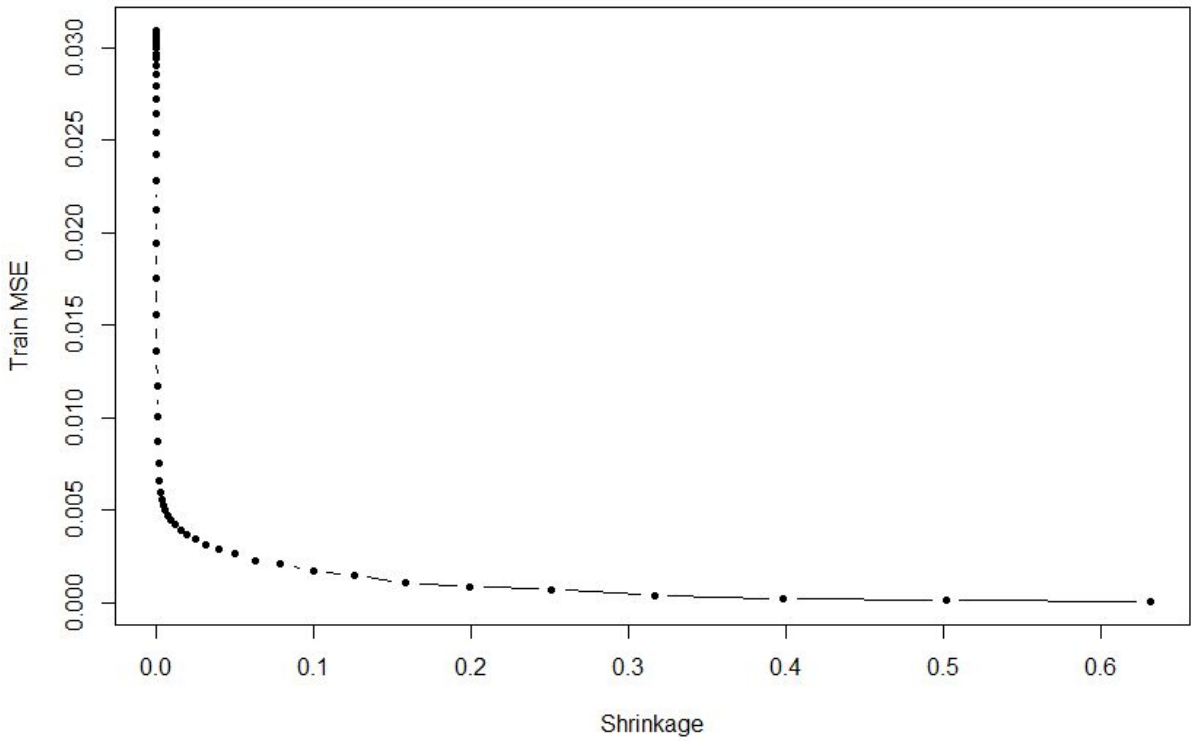
```
> library(ISLR)
> attach(Hitters)
>
> Hitters = na.omit(Hitters)
> sum(is.na(Hitters$Salary))
[1] 0
> Hitters$Salary = log(Hitters$Salary)
>
```

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

```
> train = 1:200
> Hitters.train = Hitters[train, ]
> Hitters.test = Hitters[-train, ]
>
```

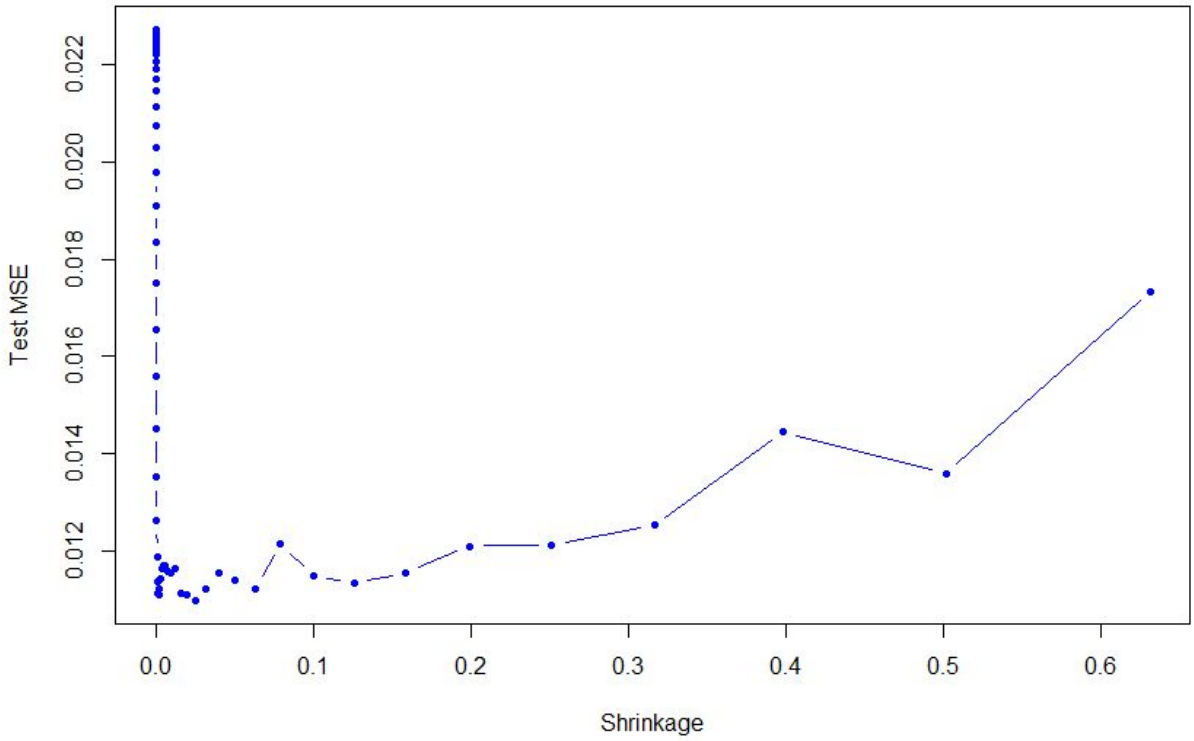
    c. Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

```
> library(gbm)
Loaded gbm 2.1.8
> set.seed(1)
>
> pows = seq(-10, -0.2, by = 0.1)
> lambdas = 10^pows
> train.errors = rep(NA, length(lambdas))
> test.errors = rep(NA, length(lambdas))
> for (i in 1:length(lambdas)) {
+   boost.hitters = gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian", n.trees = 1000, shrinkage = lambdas[i])
+   train.pred = predict(boost.hitters, Hitters.train, n.trees = 1000)
+   test.pred = predict(boost.hitters, Hitters.test, n.trees = 1000)
+   train.errors[i] = mean((Hitters.train$Salary - train.pred)^2)
+   test.errors[i] = mean((Hitters.test$Salary - test.pred)^2)
+ }
>
> plot(lambdas, train.errors, type = "b", xlab = "Shrinkage", ylab = "Train MSE", col = "black", pch = 20)
>
```



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

```
> plot(lambdas, test.errors, type = "b", xlab = "Shrinkage", ylab = "Test MSE", col = "blue", pch = 20)
>
```



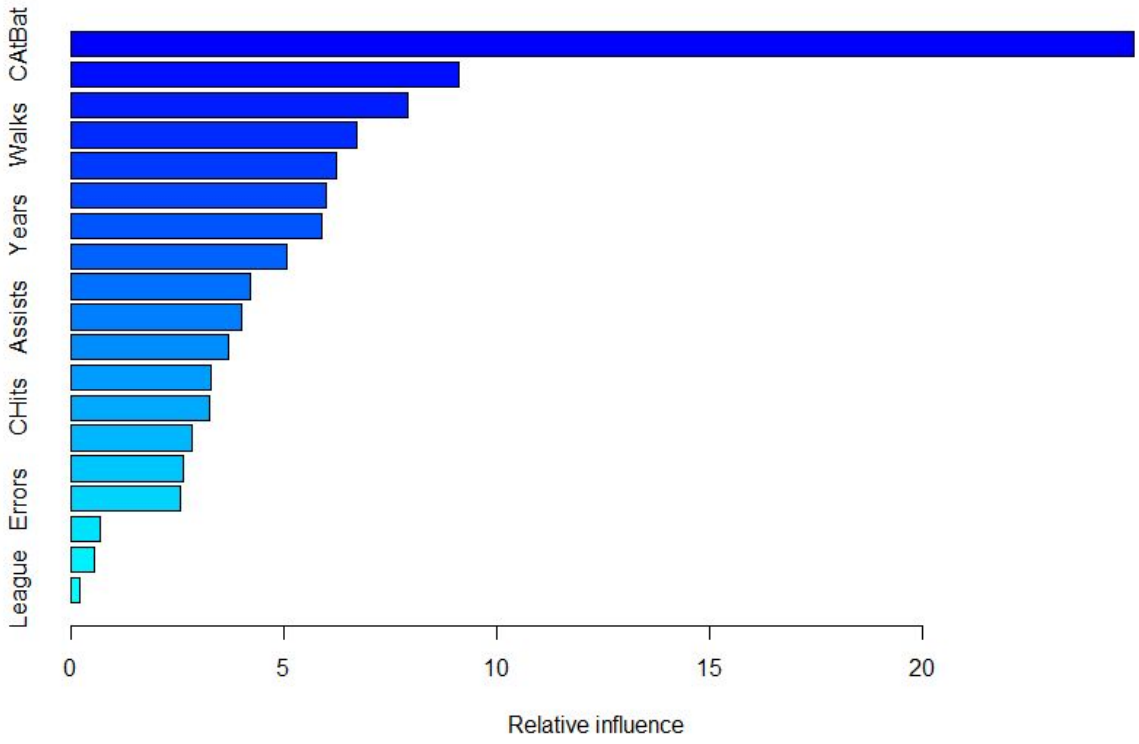
e. Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

```
> lm.fit = lm(Salary ~ ., data = Hitters.train)
> lm.pred = predict(lm.fit, Hitters.test)
> mean((Hitters.test$Salary - lm.pred)^2)
[1] 0.4917959
>
> library(glmnet)
Loading required package: Matrix
Loaded glmnet 4.0-2
>
> set.seed(100)
> x = model.matrix(Salary ~ ., data = Hitters.train)
> y = Hitters.train$Salary
> x.test = model.matrix(Salary ~ ., data = Hitters.test)
> lasso.fit = glmnet(x, y, alpha = 1)
> lasso.pred = predict(lasso.fit, s = 0.01, newx = x.test)
> mean((Hitters.test$Salary - lasso.pred)^2)
[1] 0.4700537
>
```

The linear regression and Lasso regression approaches both have higher test MSEs than boosting since the test MSEs shown for the boosting approach in part (d) are a lot smaller than the 0.4917959 and 0.4700537 test MSEs of the linear regression and Lasso regression approaches, respectively.

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

```
> boost = gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian", n.trees = 1000, shrinkage = lambdas[which.min(test.
errors)])
> summary(boost)
                var    rel.inf
CAtBat       CAtBat 24.9617941
CWalks       CWalks  9.1123297
PutOuts     PutOuts  7.9283500
Walks         Walks  6.7208325
CRBI           CRBI  6.2265743
CHmRun       CHmRun  5.9956861
Years         Years  5.9075433
RBI             RBI  5.0827594
AtBat         AtBat  4.2105182
Assists     Assists  4.0316881
Hits           Hits  3.7231678
CRuns         CRuns  3.2906939
CHits         CHits  3.2519371
HmRun         HmRun  2.8476193
Runs           Runs  2.6338671
Errors       Errors  2.5702551
Division   Division  0.7086050
NewLeague NewLeague  0.5785068
League       League  0.2172722
> |
```



From the summary and the graph, it appears that CAtBat is far and away the most important predictor, and it is followed farther behind by CWalks and PutOuts.

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

```
> library(randomForest)
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.
> set.seed(10)
>
> rf.hitters = randomForest(Salary ~ ., data = Hitters.train, ntree = 500, mtry = 19)
> rf.pred = predict(rf.hitters, Hitters.test)
> mean((Hitters.test$Salary - rf.pred)^2)
[1] 0.2325128
>
```

The test MSE of the bagging approach is 0.2325128.