STT 481 HW 5

Derien Weatherspoon

2023-04-10

```r
# Load libraries
library(ISLR)
library(MASS)
library(gam)
library(glmnet)
library(boot)
library(leaps)
library(rpart)
library(randomForest)
library(splines)
library(tree)
library(gbm)
```

Question 1

Fit a GAM to predict Salary in the Hitters dataset.

```r
data("Hitters")
Hitters <- Hitters[!is.na(Hitters$Salary),]
set.seed(1111)
train.h <- sample(nrow(Hitters), 200)
Hitters.train <- Hitters[train.h,]
Hitters.test <- Hitters[-train.h,]
```

First, remove the observations for whom the salary information is unknown, then split the data set by using the following command lines:

```r
fwd.hitters <- regsubsets(log(Salary) ~ ., data = Hitters.train, method = 'forward', nvmax = ncol(Hitte

fwd.summary.hitters <- summary(fwd.hitters)

which.min(fwd.summary.hitters$bic)
```

a) Using log(Salary) as response and the other variables as the predictors, perform forward stepwise selection on the training set in order to identify a satisfactory model that uses just a subset of the predictors.

```
## [1] 3
```

```r
which.min(fwd.summary.hitters$adjr2)
```

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

```r
which.min(fwd.summary.hitters$cp) # will use the cp value
```

```
## [1] 5
```

```r
coef(fwd.hitters, id = 3)
```

```
## (Intercept)        Hits       CRuns     LeagueN
## 4.757429171 0.004477675 0.001631643 0.229046812
```

Using forward stepwise selection and the lowest BIC score of 3, the best subset of predictors for the model are "Hits", "CRuns", "League".

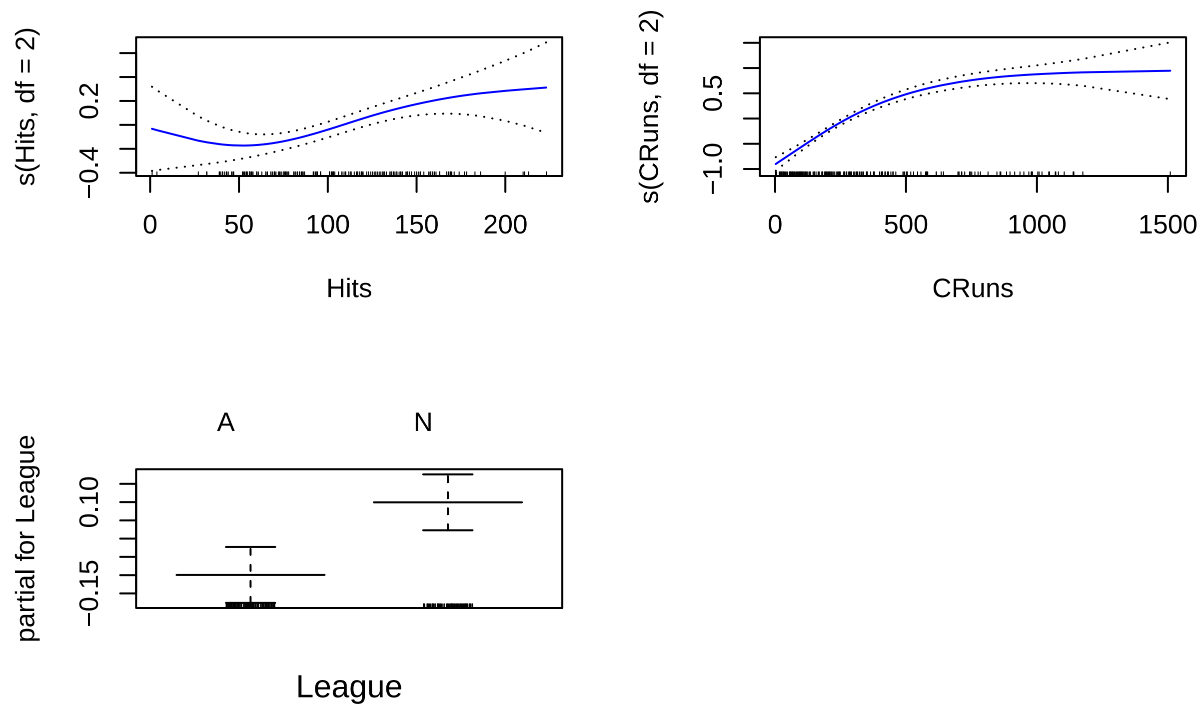
```r
hitters.gam <- gam(log(Salary) ~ s(Hits, df = 2) + s(CRuns, df = 2) + League, data = Hitters.train)

par(mfrow = c(2,2))
plot(hitters.gam, se = T, col = "blue")
summary(hitters.gam)
```

b) Fit a GAM on the training data, using log(Salary) as the response and the features selected in the previous step as the predictors. Plot the results, explain your findings.

```
##
## Call: gam(formula = log(Salary) ~ s(Hits, df = 2) + s(CRuns, df = 2) +
##     League, data = Hitters.train)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.16990 -0.35359  0.01577  0.34166  2.54946
##
## (Dispersion Parameter for gaussian family taken to be 0.2876)
##
##     Null Deviance: 149.2826 on 199 degrees of freedom
## Residual Deviance: 55.7994 on 194.0001 degrees of freedom
## AIC: 326.2644
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                   Df Sum Sq Mean Sq  F value    Pr(>F)
## s(Hits, df = 2)    1 18.575  18.575  64.5819 8.852e-14 ***
## s(CRuns, df = 2)   1 50.172  50.172 174.4364 < 2.2e-16 ***
## League             1  1.941   1.941   6.7467   0.01011 *
## Residuals        194 55.799   0.288
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                     Npar Df Npar F     Pr(F)
## (Intercept)
## s(Hits, df = 2)          1 10.380  0.001495 **
## s(CRuns, df = 2)         1 56.885 1.738e-12 ***
## League
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



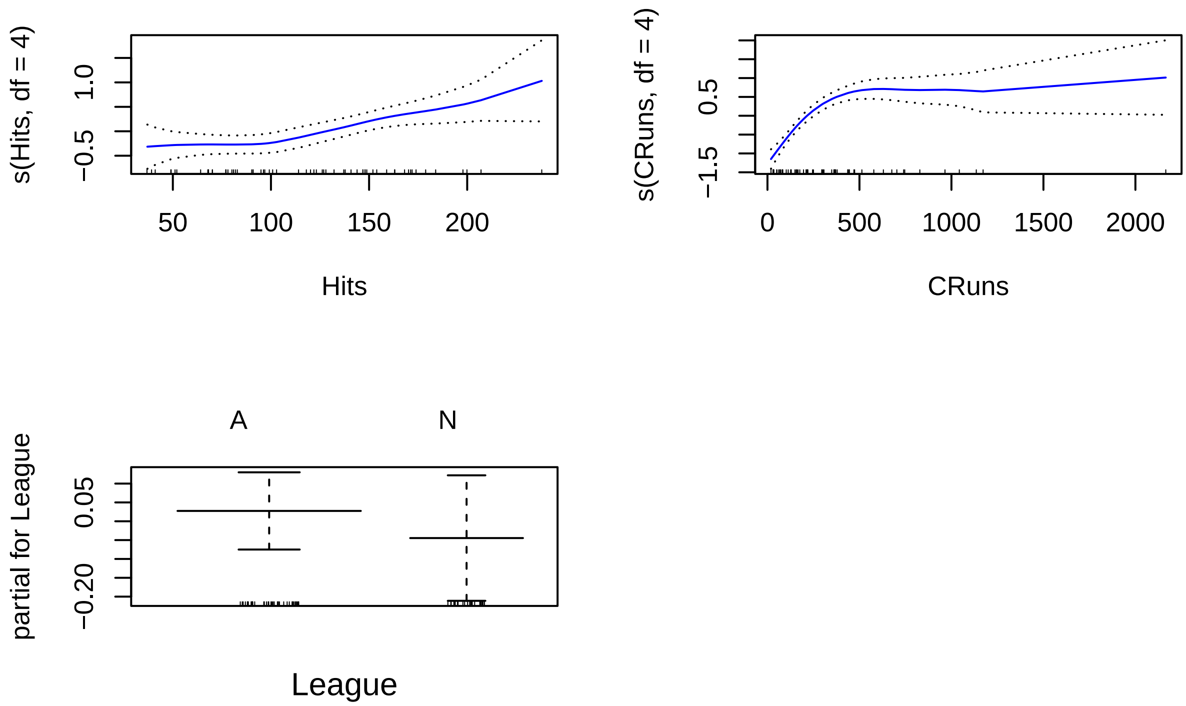
This model produced an AIC of 326.2644, which seems to be alright for a model of this size. All variables are significant for with Parametrix Effects.

```r
hitters.gam.test <- gam(log(Salary) ~ s(Hits, df = 4) + s(CRuns, df = 4) + League, data = Hitters.test)

par(mfrow = c(2,2))
plot(hitters.gam.test, se = T, col = "blue")
summary(hitters.gam.test)
```

c) Evaluate the model obtained on the test set. try different tuning parameters (if using s() then try different df's; if using local regression lo(), try different span's) and explain the results obtained.

```
##
## Call: gam(formula = log(Salary) ~ s(Hits, df = 4) + s(CRuns, df = 4) +
##     League, data = Hitters.test)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.50460 -0.25034  0.01673  0.26679  1.14772
##
## (Dispersion Parameter for gaussian family taken to be 0.2528)
##
##     Null Deviance: 57.8336 on 62 degrees of freedom
## Residual Deviance: 13.3982 on 53.0001 degrees of freedom
## AIC: 103.2612
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                  Df  Sum Sq Mean Sq F value    Pr(>F)
## s(Hits, df = 4)   1  6.1621  6.1621  24.376 8.281e-06 ***
## s(CRuns, df = 4)  1 14.1996 14.1996  56.170 7.219e-10 ***
## League            1  0.0725  0.0725   0.287    0.5944
## Residuals        53 13.3982  0.2528
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                  Npar Df  Npar F     Pr(F)
## (Intercept)
## s(Hits, df = 4)        3  1.0452    0.3802
## s(CRuns, df = 4)       3 13.1168 1.601e-06 ***
## League
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



Comparing the test and training set GAM, the first obvious difference is that the plots are much more "bumpier" in a sense. The AIC score dropped significantly to 103.2612, which is great. As I increased the degree of freedom for Hits and CRuns, it seems their significance also increased.

d) For which variables, if any, is there evidence of a non-linear relationship with the response?
It seems that there is a non-linear relationship with the CRuns and the response, shown from ANOVA for Nonparametric Effects.

Question 2

This question relates to the Credit data set. (Regression problem)

```
data("Credit")
set.seed(1234)
Credit <- Credit[,-1] # remove ID column
train.c <- sample(nrow(Credit), 300)
Credit.train <- Credit[train.c,]
Credit.test <- Credit[-train.c,]
```

First, split the data set by running the following command lines:

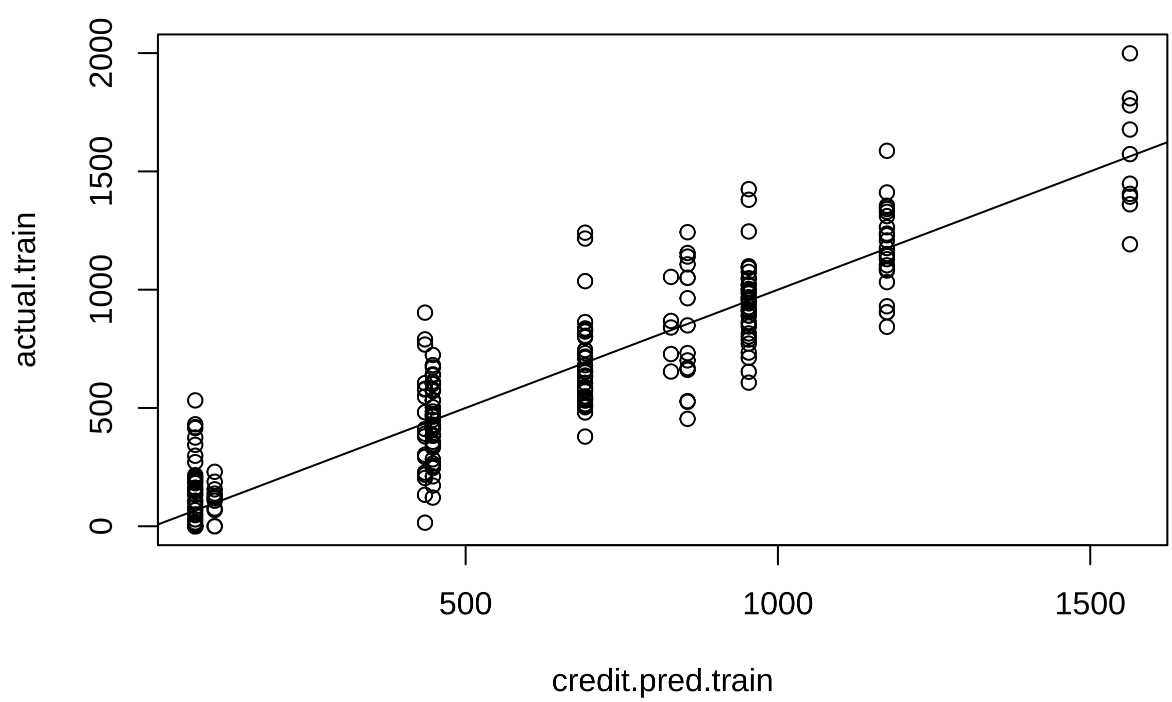
```r
credit.tree <- tree(Balance ~ ., data = Credit.train)
summary(credit.tree)
```

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

```
##
## Regression tree:
## tree(formula = Balance ~ ., data = Credit.train)
## Variables actually used in tree construction:
## [1] "Rating"  "Income"  "Student" "Limit"
## Number of terminal nodes:  10
## Residual mean deviance:  27320 = 7923000 / 290
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -419.90  -67.17  -38.31    0.00   92.56  549.60
```

```r
# MSE

credit.pred.train <- predict(credit.tree, newdata = Credit.train)
actual.train <-Credit.train$Balance
plot(credit.pred.train, actual.train)
abline(0,1)
```



```r
round(mean((credit.pred.train-actual.train)^2))
```

```
## [1] 26410
```

For this tree model, the only variables used in construction are: "Rating", "Income", "Student", "Limit". There are 10 terminal nodes. I found the MSE to be 26410 for the training set.

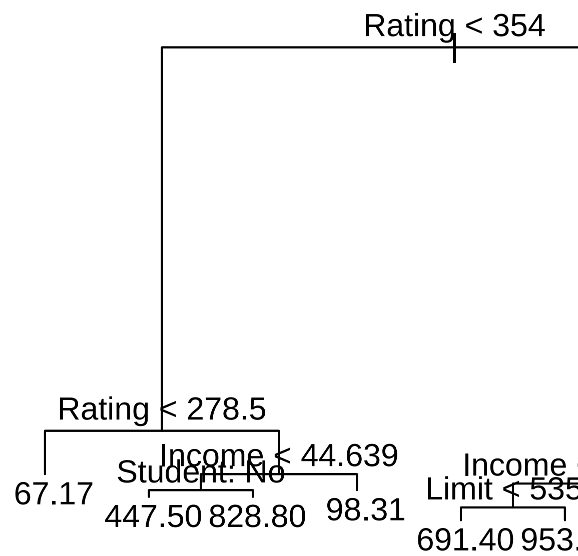
```r
credit.tree
```

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

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 300 63110000  519.90
##    2) Rating < 354 160  8789000  191.00
##      4) Rating < 278.5 101  1398000   67.17 *
##      5) Rating > 278.5 59  3193000  402.90
##       10) Income < 44.639 46  1584000  489.00
##         20) Student: No 41   842900  447.50 *
##         21) Student: Yes 5    93090  828.80 *
##       11) Income > 44.639 13    62740   98.31 *
##    3) Rating > 354 140 17210000  895.90
##      6) Limit < 6769 91  6815000  755.80
##       12) Income < 60.2745 73  3509000  834.90
##         24) Limit < 5353 33  1203000  691.40 *
##         25) Limit > 5353 40  1066000  953.20 *
##       13) Income > 60.2745 18   996300  434.90 *
##      7) Limit > 6769 49  5286000 1156.00
##       14) Rating < 714 39  2645000 1052.00
##         28) Income < 100.151 24   745100 1175.00 *
##         29) Income > 100.151 15   959600  855.30 *
##       15) Rating > 714 10   555900 1564.00 *
```

Choosing node 12. This node asked whether or not the income was less than 60.2745. If income was in this threshold, then the tree split the average, and if you're income was less than 60.2745, then the balance for this path would be 834.90.

```r
plot(credit.tree)
text(credit.tree, pretty = 0)
```



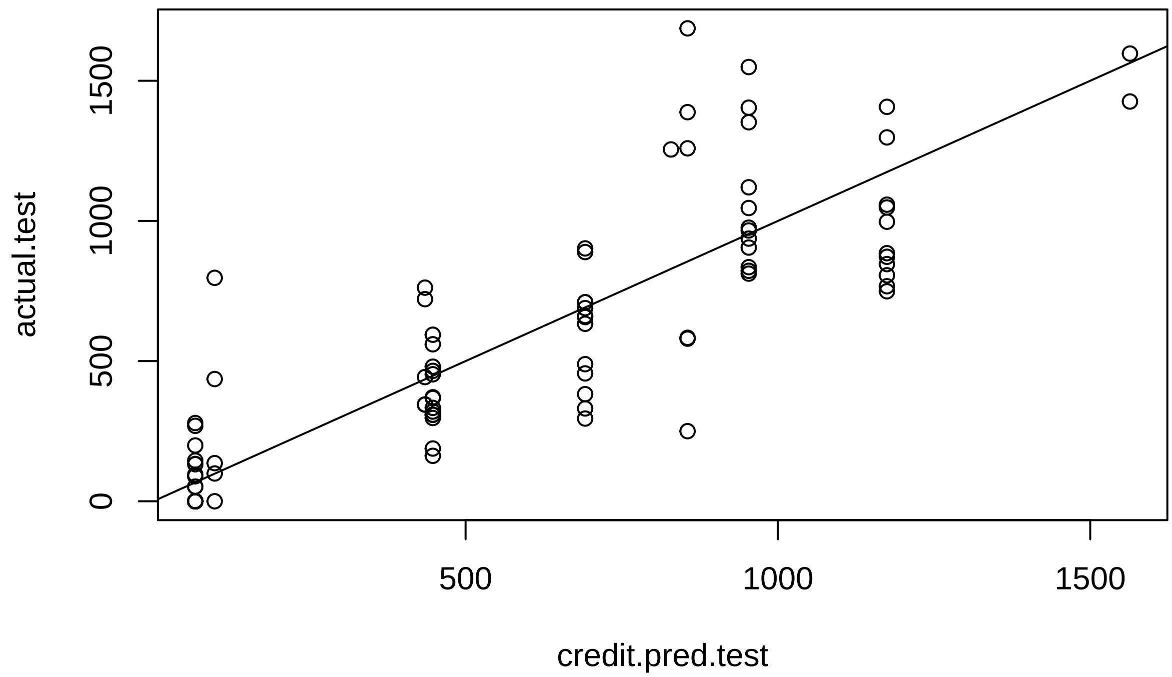
c) Create a plot of the tree, and interpret the results.
We can see here from the plot that there are indeed 10 terminal nodes, with the most important ones being Rating, Limit, and Income.

```
credit.pred.test <- predict(credit.tree, newdata = Credit.test)
head(credit.pred.test)
```

d) Predict the response on the test data. What is the test MSE?

```
##          1          3          5          8          9         11
##  447.51220  855.33333  691.36364 1174.54167   67.16832 1174.54167
```

```
actual.test <-Credit.test$Balance
plot(credit.pred.test, actual.test)
abline(0,1)
```




```
round(mean((credit.pred.test-actual.test)^2))
```

```
## [1] 54493
```

Making predictions on the test set, the predictions look pretty scattered all over the place. I would think that this is due to the distribution of wealth not being equal across the globe. I found the MSE for the test set to be much much higher than training MSE, which is worrisome.

```
cv.credit <- cv.tree(credit.tree)
best.size <- cv.credit$size[which.min(cv.credit$dev)]
best.size
```

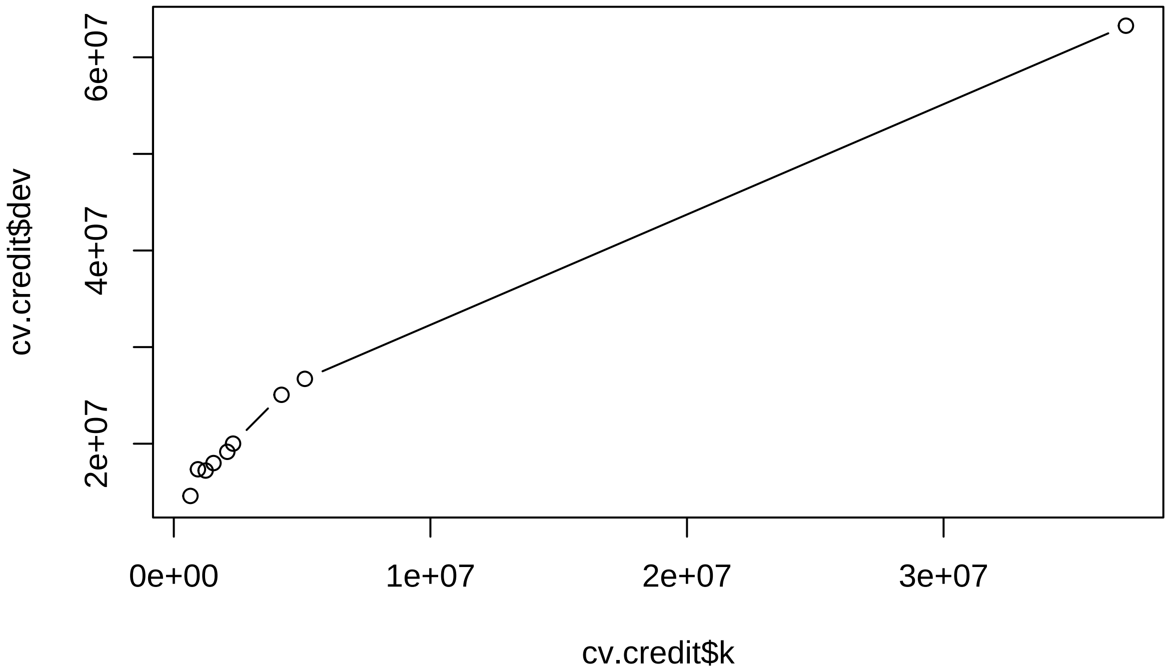
e) Apply the cv.tree() function to the training set in order to determine the optimal tree size.

```
## [1] 10
```

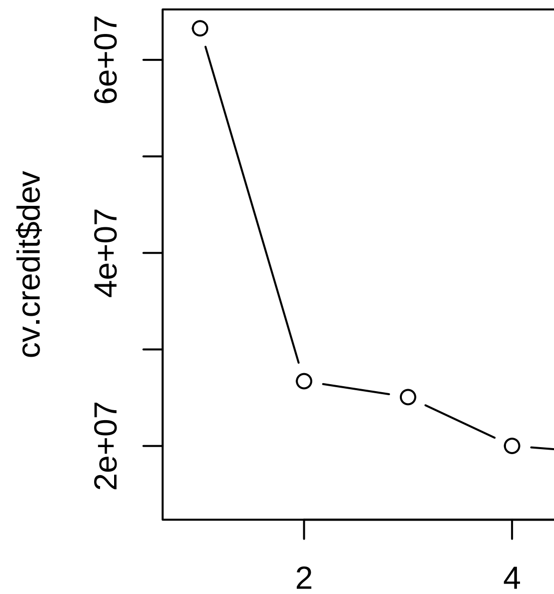
The best size for the tree seems to be K = 10.

```
plot(cv.credit$k, cv.credit$dev, type = "b") # main plot
```

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




```
plot(cv.credit$size, cv.credit$dev, type = "b")
```

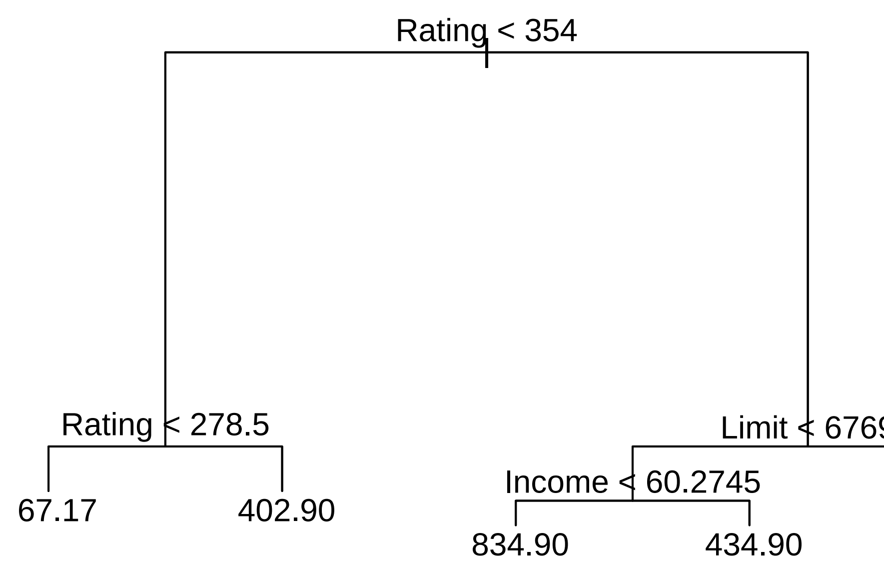


g) Which tree size corresponds to the lowest cross-validated error?
Judging from the plot, it looks like the size of 10 is corresponds to the lowest error.

```
#prune.credit <- prune.tree(credit.tree, best = best.size) #use prune.misclass for next part
#plot(prune.credit) # the plot is the same
#text(prune.credit, pretty = 0)
#prune.credit

prune.credit <- prune.tree(credit.tree, best = 5)
plot(prune.credit)
text(prune.credit, pretty = 0)
```

h) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned



tree with five terminal nodes.

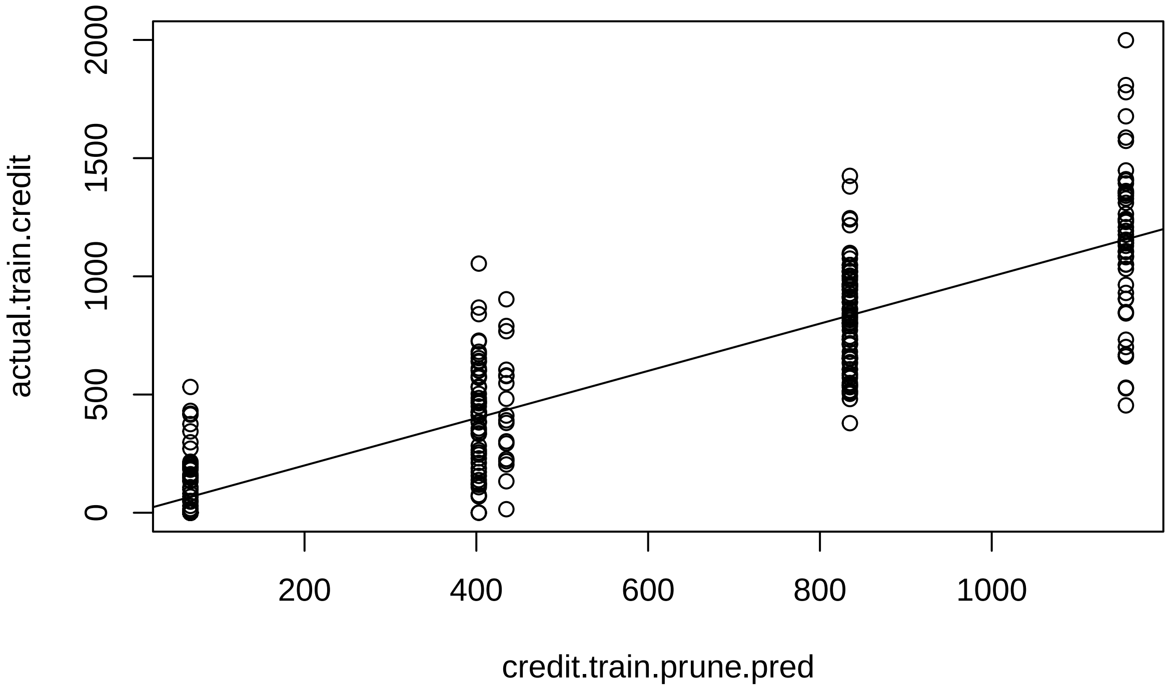
```
prune.credit
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 300 63110000  519.90
##    2) Rating < 354 160  8789000  191.00
##      4) Rating < 278.5 101  1398000   67.17 *
##      5) Rating > 278.5 59  3193000  402.90 *
##    3) Rating > 354 140 17210000  895.90
##      6) Limit < 6769 91  6815000  755.80
##       12) Income < 60.2745 73  3509000  834.90 *
##       13) Income > 60.2745 18   996300  434.90 *
##      7) Limit > 6769 49  5286000 1156.00 *
```

The plot is the same, so use a pruned tree with 5 terminal nodes.

```
# PRUNED TRAINING MSE
credit.train.prune.pred <- predict(prune.credit, newdata = Credit.train)
actual.train.credit <-Credit.train$Balance
plot(credit.train.prune.pred, actual.train.credit)
abline(0,1)
```

i) Compare the training MSEs between the pruned and unpruned trees. Which is higher?

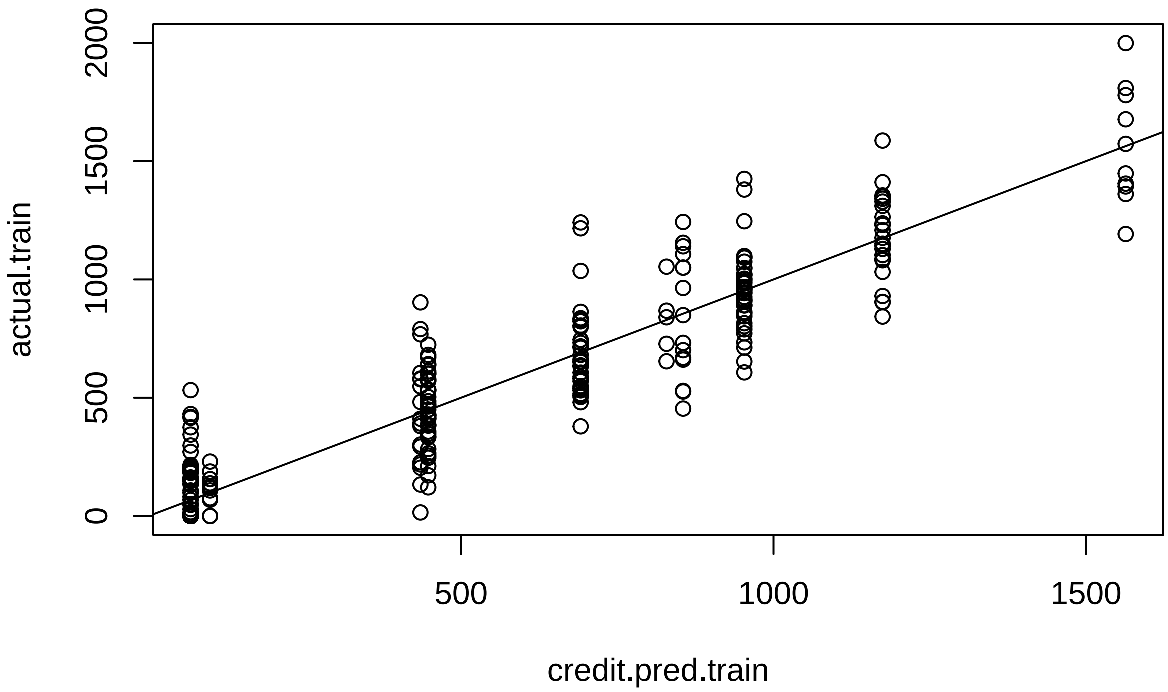



```
(mean((credit.train.prune.pred-actual.train.credit)^2))
```

```
## [1] 47944.59
```

```
# UNPRUNED TRAINING MSE

credit.pred.train <- predict(credit.tree, newdata = Credit.train)
plot(credit.pred.train, actual.train)
abline(0,1)
```



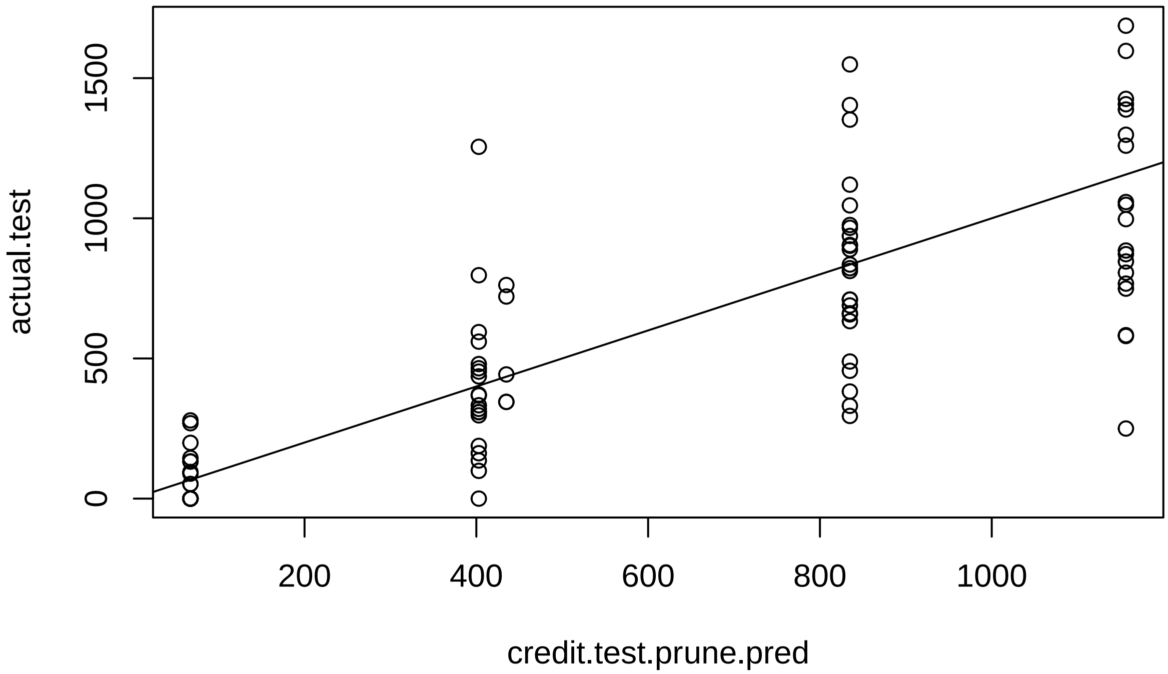

```
(mean((credit.pred.train-actual.train)^2))
```

```
## [1] 26409.8
```

Comparatively, the training MSE for unpruned trees is a lot lower than the training MSE for pruned trees. Why is this? Is the pruned model poor?

```
# PRUNED TEST MSE
credit.test.prune.pred <- predict(prune.credit, newdata = Credit.test)
actual.test <-Credit.test$Balance
plot(credit.test.prune.pred, actual.test)
abline(0,1)
```

j) Compare the test MSEs between the pruned and unpruned trees. Which is higher?

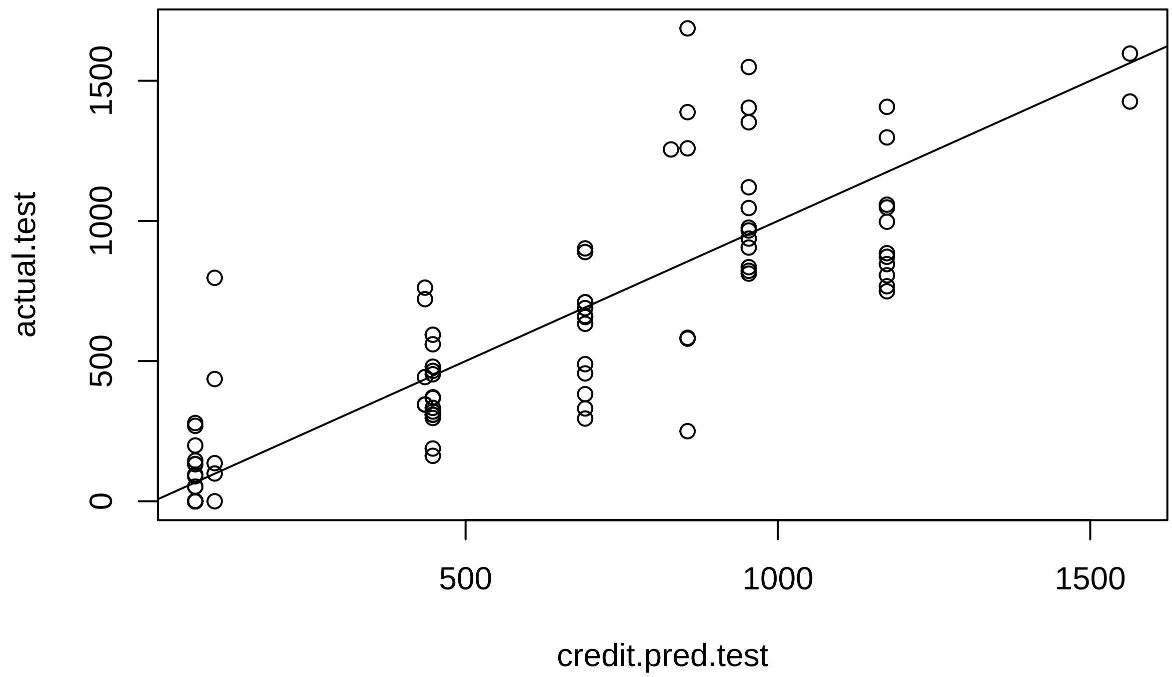



```
(mean((credit.test.prune.pred-actual.test)^2))
```

```
## [1] 72198.94
```

```
# UNPRUNED TEST MSE

credit.pred.test <- predict(credit.tree, newdata = Credit.test)
plot(credit.pred.test, actual.test)
abline(0,1)
```




```r
(mean((credit.pred.test-actual.test)^2))
```

```
## [1] 54493.28
```

Again, the pruned test MSE is much higher than the unpruned test MSE.

```r
ncol(Credit.train) - 1 # number of predictors
```

k) Fit a bagging model to the training set with Balance as the response and the other variables. Use 1,000 trees (ntree = 1000). Use the importance() function to determine which variables are most important.

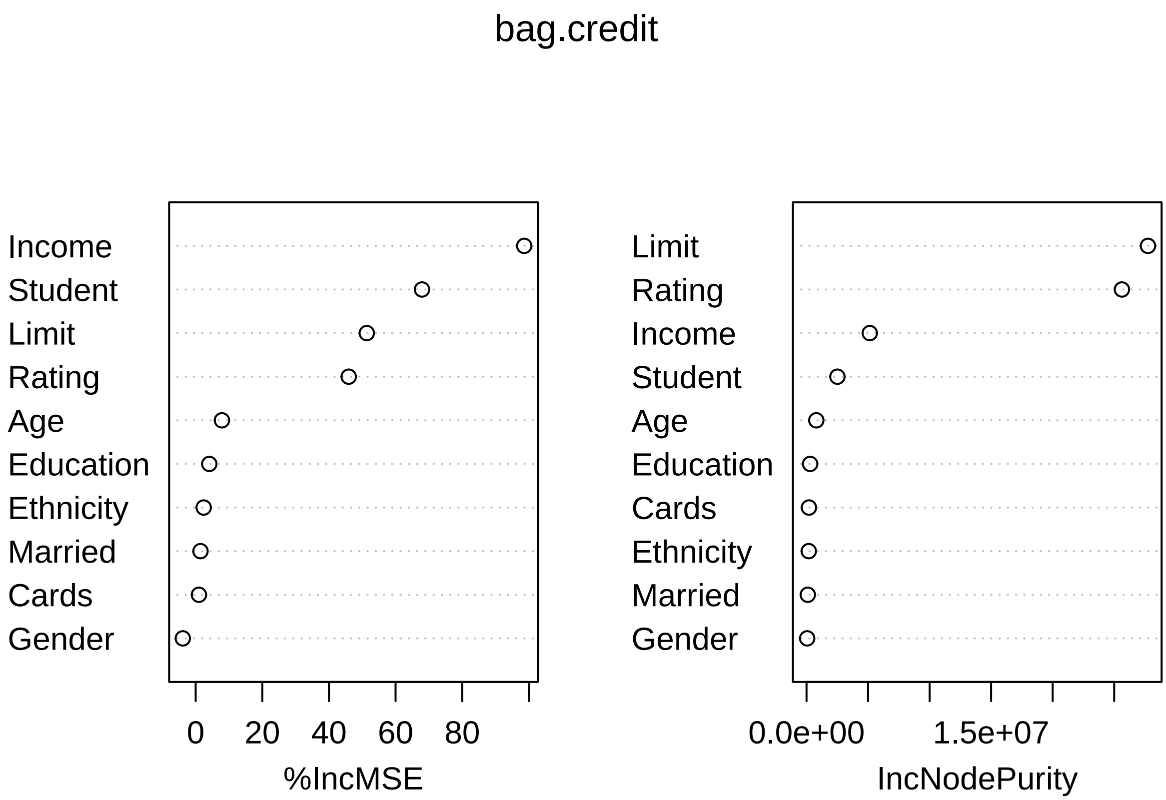
```
## [1] 10
```

```r
bag.credit <- randomForest(Balance ~ ., data = Credit.train, mtry = 10, ntree = 1000, importance = T)
importance(bag.credit)
```

```
##              %IncMSE IncNodePurity
## Income    98.5998454    5138157.81
## Limit     51.3538311   27742422.16
## Rating    45.9294371   25627921.30
```

```
## Cards       0.9755303      196120.46
## Age         7.8811882      796145.29
## Education   4.0931546      295200.23
## Gender     -3.8625972       51875.26
## Student    67.9278815     2511423.46
## Married     1.4525244      103298.40
## Ethnicity   2.4087330      180770.62
```

```
varImpPlot(bag.credit)
```



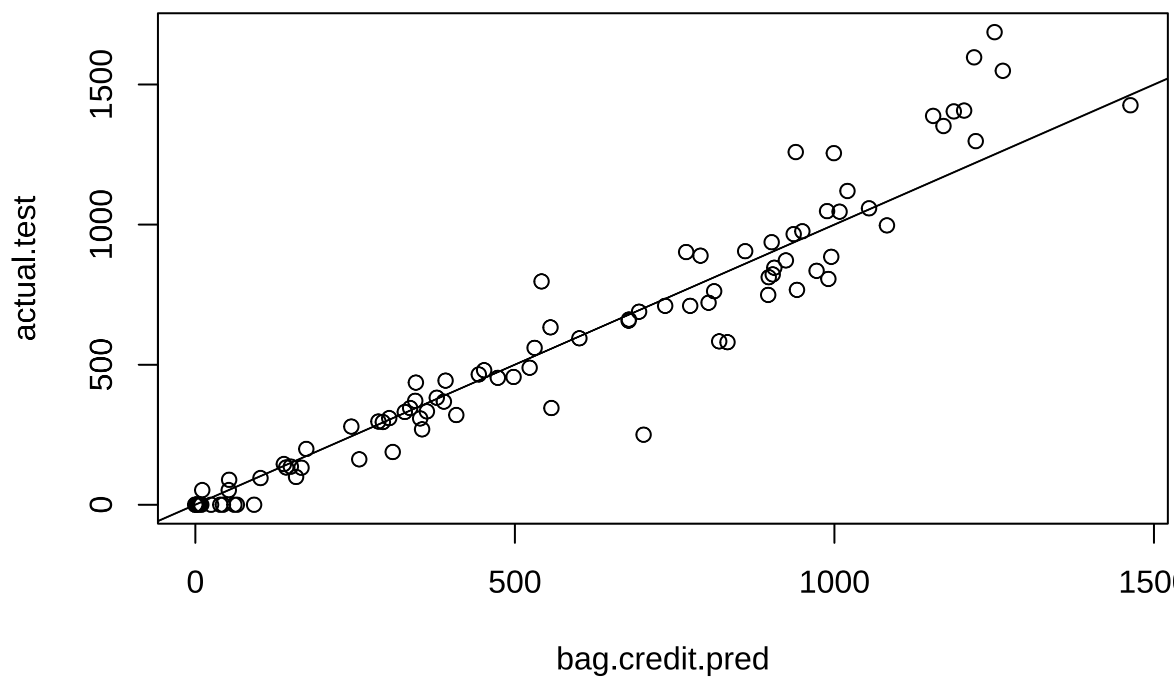
mtry is set to 10 because I determined earlier in the chunk that the number of predictors should be 10. Using the importance function too, I have found the 10 predictors and their measure of importance.

```
bag.credit.pred <- predict(bag.credit, newdata = Credit.test)
head(bag.credit.pred) # predicting response
```

l) Use the bagging model to predict the response on the test data. Compute the test MSE.

```
##         1         3         5         8         9        11
##  362.3289  832.6484  327.7744  924.0371  244.0596 1202.9326
```

```r
actual.test <-Credit.test$Balance
plot(bag.credit.pred, actual.test)
abline(0,1)
```




```r
(mean((bag.credit.pred-actual.test)^2))
```

```
## [1] 15024.21
```

The test MSE was found to be 15024.21, and I have also predicted the first 11 values.

```r
sqrt(ncol(Credit.train) - 1)
```

m) Fit a random forest model to the training set with Balance as the response and the other variables. Use 1,000 trees (ntree = 1000). Use the importance() function to determine which variables are most important.

```
## [1] 3.162278
```

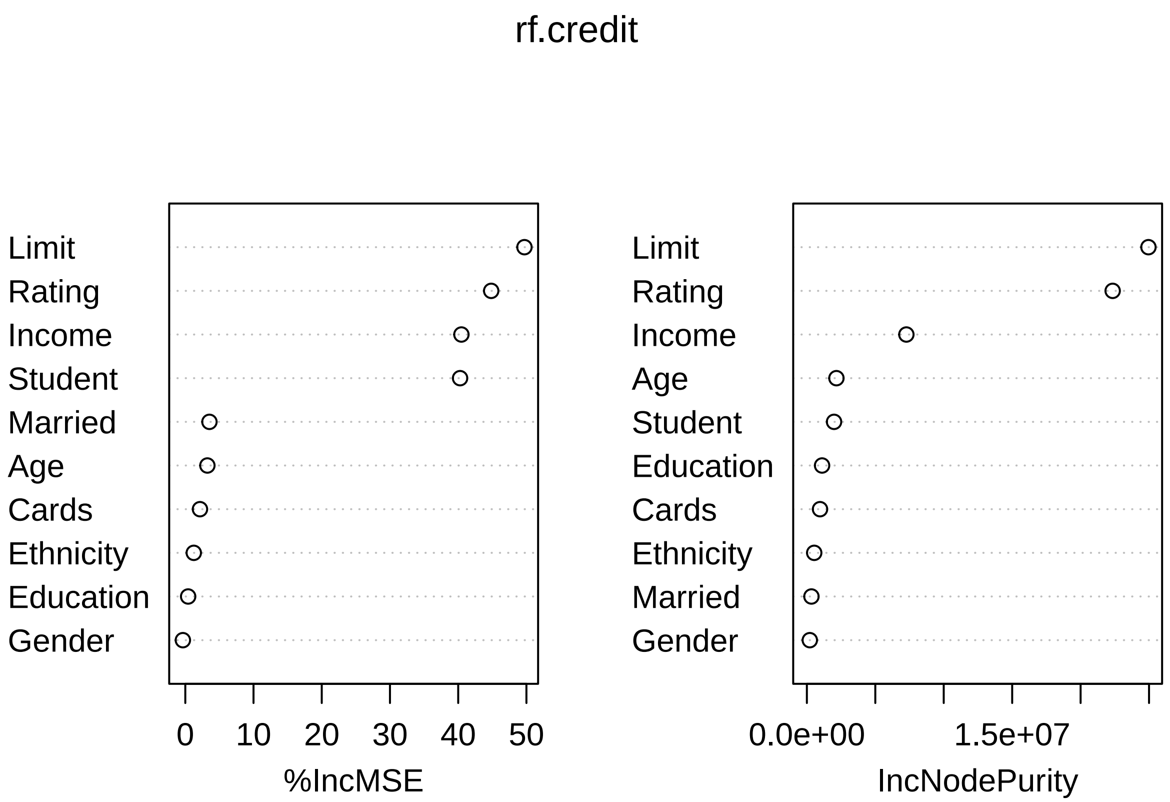
```r
rf.credit <- randomForest(Balance ~ ., data = Credit.train, mtry = 3, importance = T, ntree = 1000)
rf.credit
```

```
##
## Call:
##  randomForest(formula = Balance ~ ., data = Credit.train, mtry = 3,      importance = T, ntree = 100
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 22239.29
##                     % Var explained: 89.43
```

```
importance(rf.credit)
```

```
##              %IncMSE IncNodePurity
## Income    40.4695663     7270185.6
## Limit     49.7083027    24955913.6
## Rating    44.8366090    22342692.2
## Cards      2.1487242      961089.2
## Age        3.2364938     2158534.1
## Education  0.4173546     1110805.1
## Gender    -0.3556838      215434.2
## Student   40.2746512     1986382.4
## Married    3.5371939      332790.2
## Ethnicity  1.2453850      537017.9
```

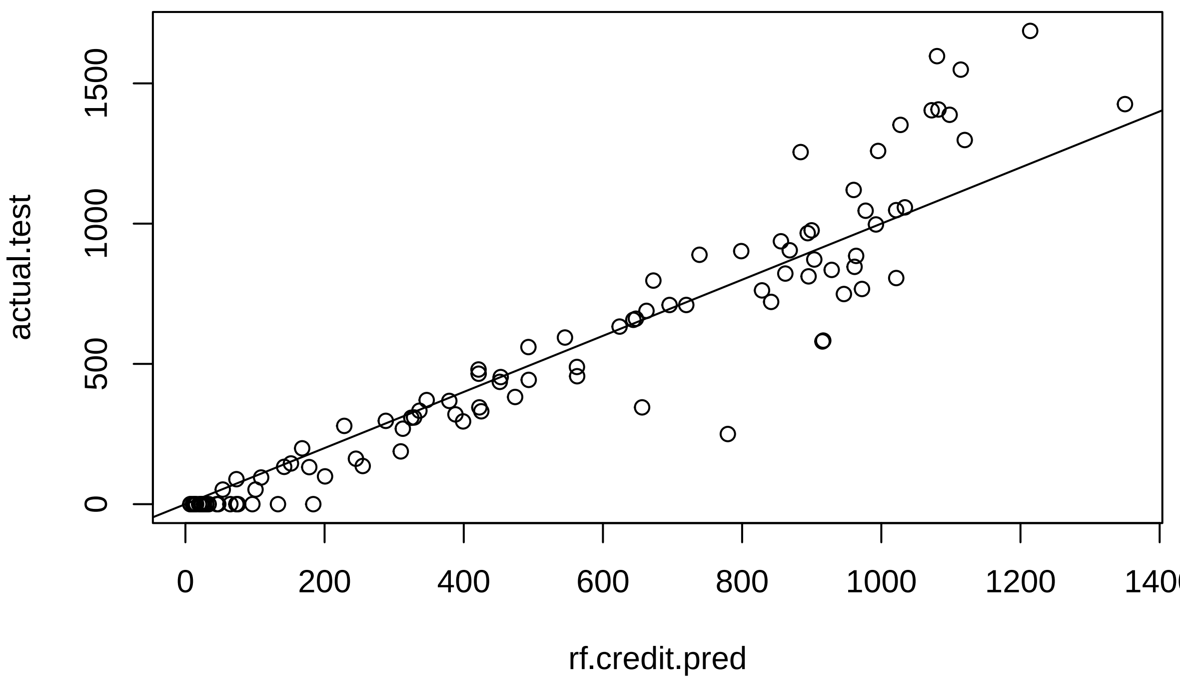
```
varImpPlot(rf.credit)
```



The importance of each variable is shown. It looks like Cards, Age, Education, Gender, Married, and Ethnicity are almost useless in the importance factor.

```
rf.credit.pred <- predict(rf.credit, newdata = Credit.test)
actual.test <-Credit.test$Balance
plot(rf.credit.pred, actual.test)
abline(0,1)
```

n) Use the random forest to predict the response on the test data. Compute the MSE.




```
(mean((rf.credit.pred-actual.test)^2))
```

```
## [1] 24306.02
```

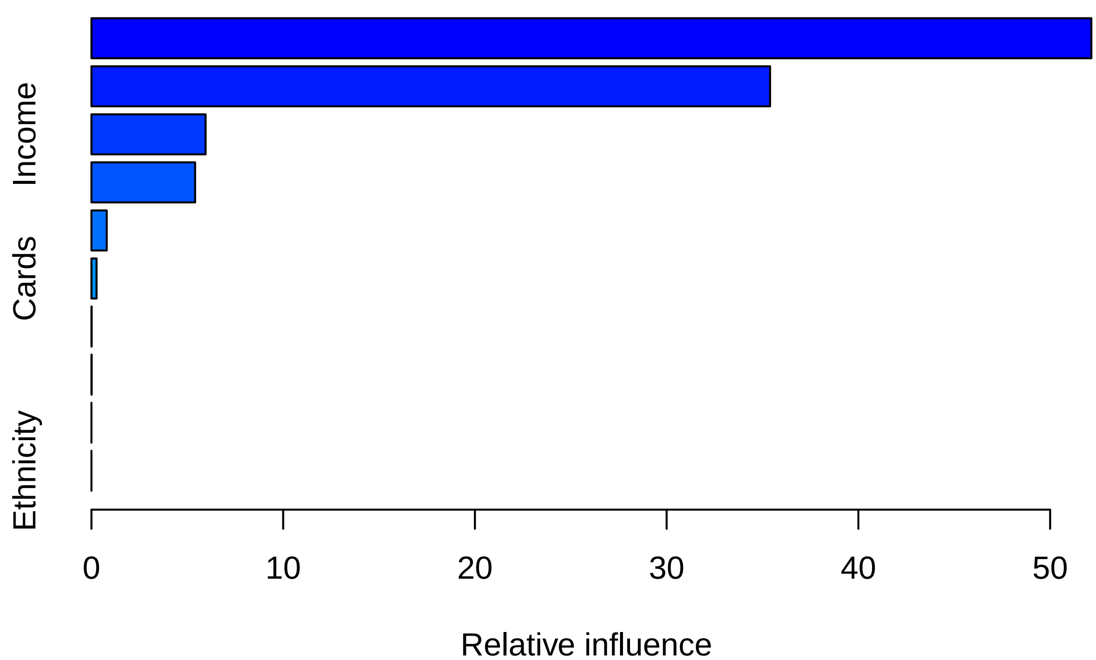
I calculated the test MSE to be 24306.02, which isn't the best or the worst that we've seen so far.

```
credit.boost <- gbm(Balance ~., data = Credit.train, distribution = "gaussian", n.trees = 1000, shrinkag
credit.boost
```

o) Fit a boosting model to the training set with Balance as the response and the other variables. Use 1,000 trees, and a shrinkage value of 0.01 (lambda = 0.01). Which predictors appear to be the most important?

```
## gbm(formula = Balance ~ ., distribution = "gaussian", data = Credit.train,
##     n.trees = 1000, shrinkage = 0.01)
## A gradient boosted model with gaussian loss function.
## 1000 iterations were performed.
## There were 10 predictors of which 8 had non-zero influence.
```

```
summary(credit.boost)
```




```
##                  var     rel.inf
## Limit          Limit 52.15080126
## Rating        Rating 35.39457042
## Income        Income  5.95442974
## Student      Student  5.40685682
## Age              Age  0.79802739
## Cards          Cards  0.26733156
## Education Education  0.01442402
## Married      Married  0.01355879
## Gender        Gender  0.00000000
## Ethnicity Ethnicity  0.00000000
```

The boosted model states that there 10 predictors included, 8 of those had non-zero influence. The most influence was provided by Limit and Rating.

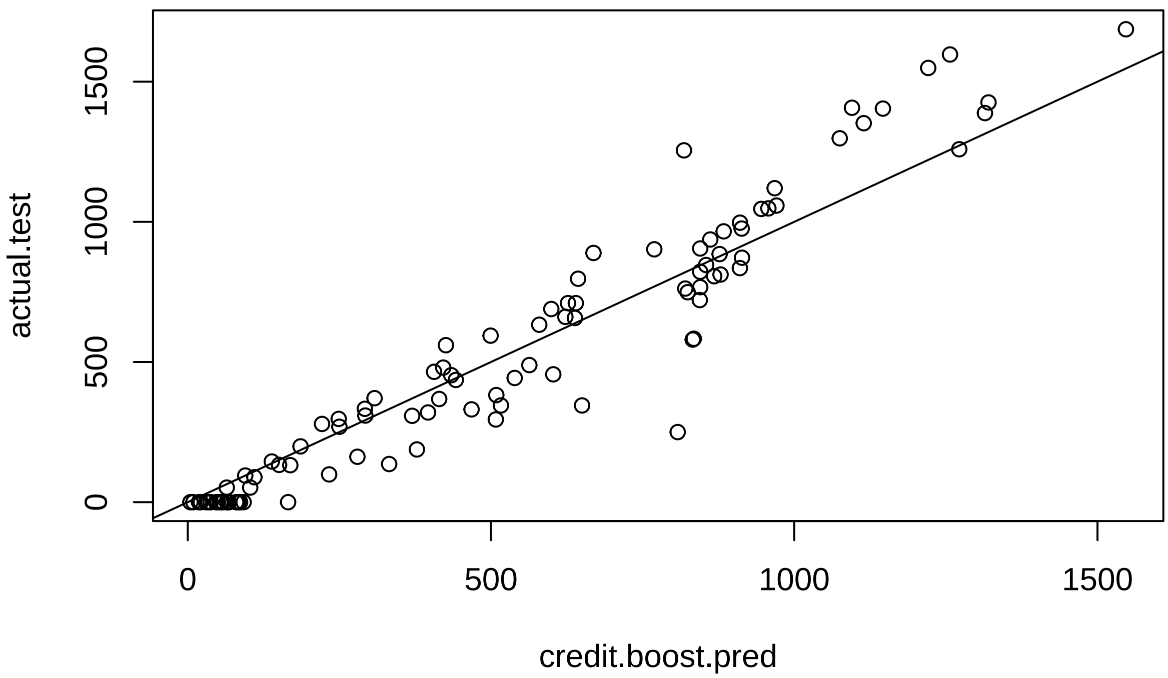
```r
#First, we need to find the best number of trees.
credit.boost.cv <- gbm(Balance ~ ., data = Credit.train,
                       distribution = "gaussian", shrinkage = 0.01,
                       n.tree = 1000, cv.folds = 10)
which.min(credit.boost.cv$cv.error)
```

p) Use the boosting model to predict the response on the test data. Compute the test MSE.

```
## [1] 1000
```

```r
# 1000 trees is the best number of trees to make predictions with.

credit.boost.pred <- predict(credit.boost, newdata = Credit.test, n.trees = which.min(credit.boost.cv$cv
actual.test <-Credit.test$Balance
plot(credit.boost.pred, actual.test)
abline(0,1)
```




```r
(mean((credit.boost.pred-actual.test)^2))
```

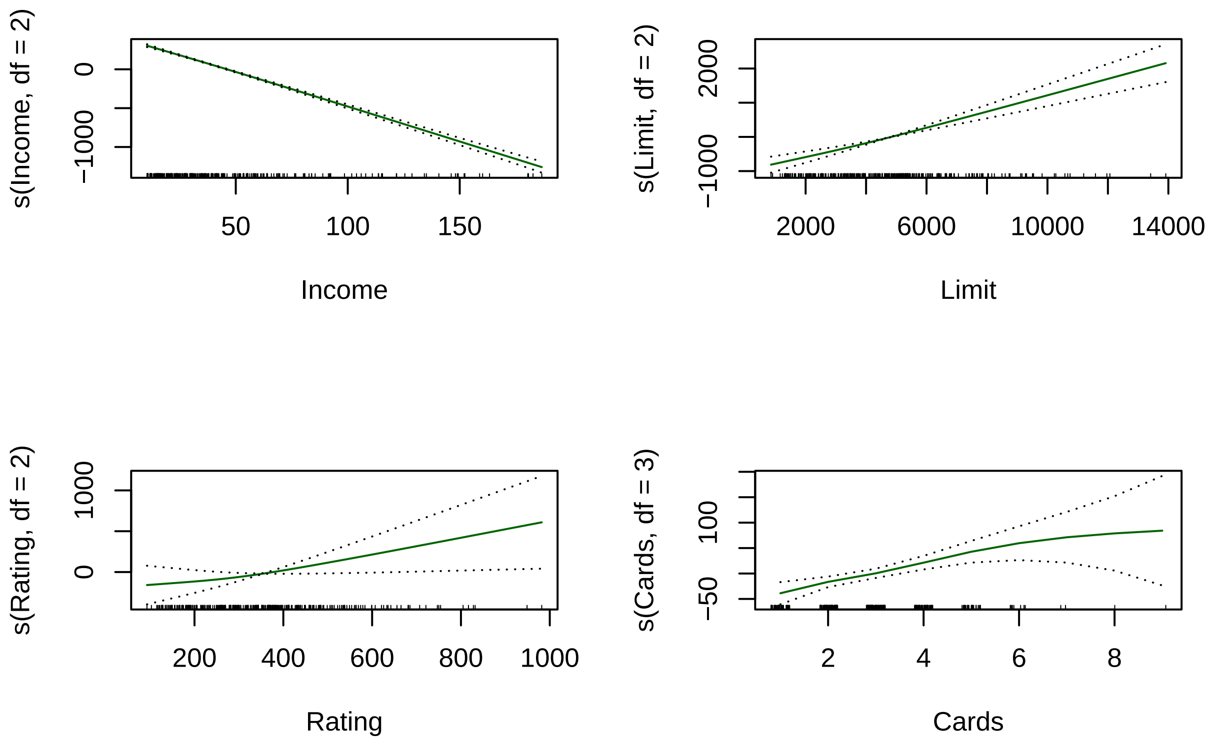
```
## [1] 19036.85
```

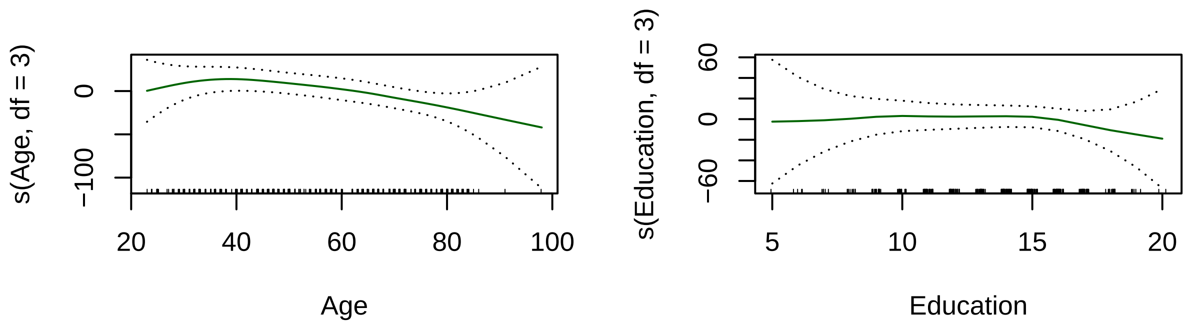
After finding out the best number of trees was 1000 for the prediction, the test MSE for the boosting model was found to be 19036.85.

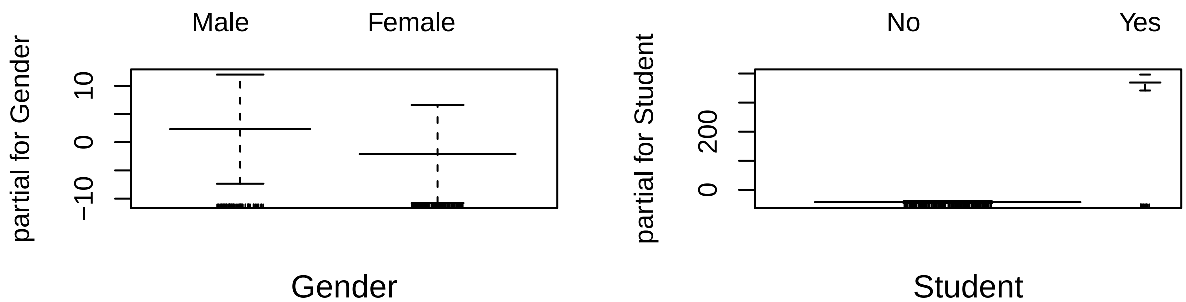
```
credit.gam <- gam(Balance ~ s(Income, df = 2) + s(Limit, df = 2) + s(Rating, df = 2) + s(Cards, df = 3)

par(mfrow = c(2,2))
plot(credit.gam, se = T, col = "darkgreen")
```

q) Fit a GAM to the training set with Balance as the response and the other variables, and use the GAM to predict the response on the test data. Compute the test MSE.







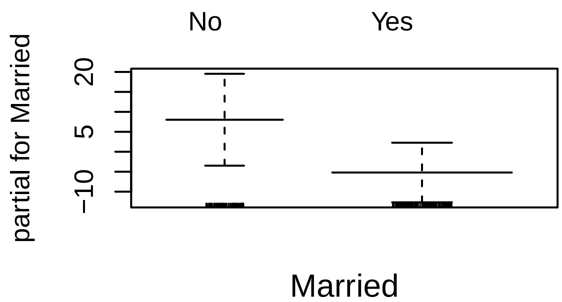

```
summary(credit.gam)
```

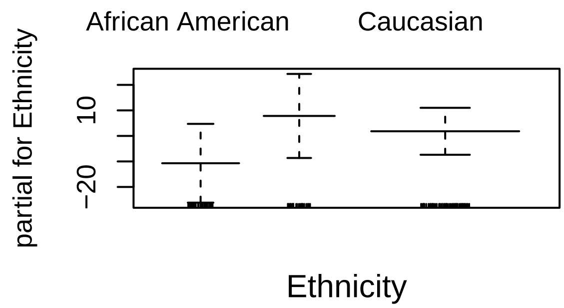
```
##
## Call: gam(formula = Balance ~ s(Income, df = 2) + s(Limit, df = 2) +
##     s(Rating, df = 2) + s(Cards, df = 3) + s(Age, df = 3) + s(Education,
##     df = 3) + Gender + Student + Married + Ethnicity, data = Credit.train)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -212.070  -54.988   -4.021   38.925  233.941
##
## (Dispersion Parameter for gaussian family taken to be 6216.927)
##
##     Null Deviance: 63105022 on 299 degrees of freedom
## Residual Deviance: 1734521 on 278.9998 degrees of freedom
## AIC: 3494.101
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                       Df   Sum Sq  Mean Sq   F value    Pr(>F)
## s(Income, df = 2)      1 12270404 12270404 1973.7089 < 2.2e-16 ***
## s(Limit, df = 2)       1 45598637 45598637 7334.5942 < 2.2e-16 ***
## s(Rating, df = 2)      1   208632   208632   33.5587 1.863e-08 ***
## s(Cards, df = 3)       1    43794    43794    7.0444 0.0084078 **
## s(Age, df = 3)         1    78595    78595   12.6421 0.0004429 ***
```

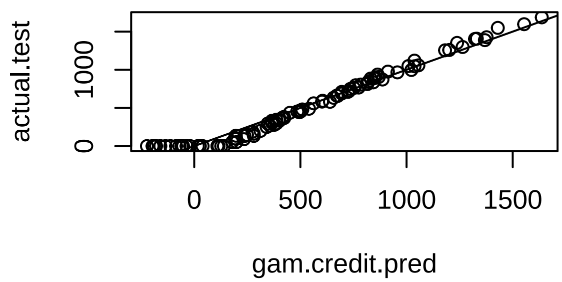
```
## s(Education, df = 3)   1    29318    29318    4.7158 0.0307300 *
## Gender                 1    34402    34402    5.5337 0.0193489 *
## Student                1  4574434  4574434  735.8031 < 2.2e-16 ***
## Married                1    10128    10128    1.6291 0.2028852
## Ethnicity              2    13640     6820    1.0970 0.3352967
## Residuals            279  1734521     6217
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                       Npar Df Npar F     Pr(F)
## (Intercept)
## s(Income, df = 2)           1  2.311    0.1296
## s(Limit, df = 2)            1 68.157 5.995e-15 ***
## s(Rating, df = 2)           1 58.658 3.091e-13 ***
## s(Cards, df = 3)            2  0.625    0.5358
## s(Age, df = 3)              2  1.288    0.2776
## s(Education, df = 3)        2  0.696    0.4993
## Gender
## Student
## Married
## Ethnicity
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Compute test MSE
gam.credit.pred <- predict(credit.gam, newdata = Credit.test)
actual.test <-Credit.test$Balance
plot(gam.credit.pred, actual.test)
abline(0,1)
(mean((gam.credit.pred-actual.test)^2))
```

```
## [1] 6708.778
```







The Test MSE for the GAM model was calculated to be 6708.778, which is very low.

```r
print(c("unpruned tree mse: ",(mean((credit.pred.test-actual.test)^2)))) # unpruned tree
```

r) Compare the test MSEs between the unpruned trees, pruned trees, bagging, random forest, boosting, and GAM. Which performs the best?

```
## [1] "unpruned tree mse: " "54493.2755026102"
```

```r
print(c("pruned tree mse: ",(mean((credit.test.prune.pred-actual.test)^2)))) # pruned tree
```

```
## [1] "pruned tree mse: " "72198.9361444461"
```

```r
print(c("bagging mse: ",(mean((bag.credit.pred-actual.test)^2)))) # bagging
```

```
## [1] "bagging mse: "    "15024.2134195537"
```

```r
print(c("random forest mse: ",(mean((rf.credit.pred-actual.test)^2)))) # random forest
```

```
## [1] "random forest mse: " "24306.0216776133"
```

```r
print(c("boosting mse: ",(mean((credit.boost.pred-actual.test)^2)))) # boosting
```

```
## [1] "boosting mse: "   "19036.8532693583"
```

```r
print(c("gam mse: ",(mean((gam.credit.pred-actual.test)^2)))) # gam
```

```
## [1] "gam mse: "        "6708.77813477811"
```

It looks like the GAM performs the best. This could be because the modifications it makes to the variables included in the model.

Question 3

This question relates to the OJ data set. (Classification problem.)

```r
data("OJ")
set.seed(200)
train.o <- sample(nrow(OJ), 800)
OJ.train <- OJ[train.o,]
OJ.test <- OJ[-train.o,]
```

First, we split the data set by using the following command lines:

```r
OJ.tree <- tree(Purchase ~ ., data = OJ.train)
summary(OJ.tree)
```

a) Fit a tree to the training data, with Purchase as the reponse and the other variables. 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?

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ.train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "ListPriceDiff" "PctDiscMM"
## Number of terminal nodes:  6
## Residual mean deviance:  0.7964 = 632.4 / 794
## Misclassification error rate: 0.1713 = 137 / 800
```

The training error rate observed here is 17%.

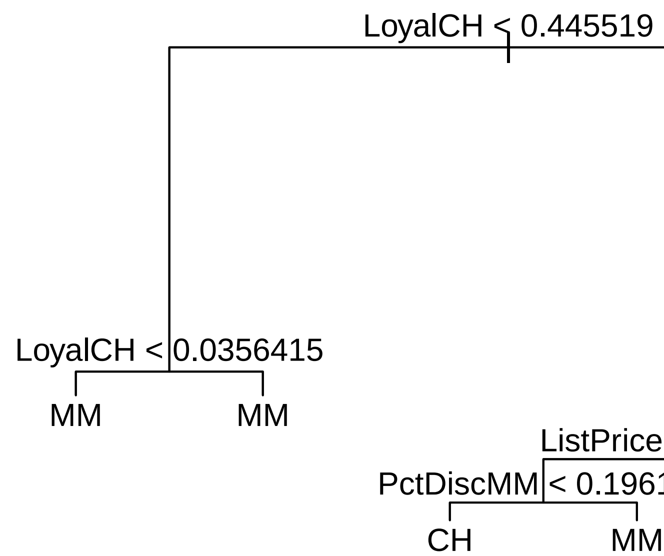
```r
OJ.tree
```

b) 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.

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1075.000 CH ( 0.60250 0.39750 )
##    2) LoyalCH < 0.445519 279  285.200 MM ( 0.20789 0.79211 )
##      4) LoyalCH < 0.0356415 55    9.996 MM ( 0.01818 0.98182 ) *
##      5) LoyalCH > 0.0356415 224  254.100 MM ( 0.25446 0.74554 ) *
##    3) LoyalCH > 0.445519 521  500.800 CH ( 0.81382 0.18618 )
##      6) LoyalCH < 0.764572 269  338.600 CH ( 0.67658 0.32342 )
##       12) ListPriceDiff < 0.235 110  151.900 MM ( 0.46364 0.53636 )
##         24) PctDiscMM < 0.196196 86  118.100 CH ( 0.55814 0.44186 ) *
##         25) PctDiscMM > 0.196196 24   18.080 MM ( 0.12500 0.87500 ) *
##       13) ListPriceDiff > 0.235 159  148.000 CH ( 0.82390 0.17610 ) *
##      7) LoyalCH > 0.764572 252   84.130 CH ( 0.96032 0.03968 ) *
```

I am choosing node 4. The split criterion here is LoyalCH $<$ 0.035, the number of observations in the branch is 55 with a deviance of 9.996 and an overall prediction for the branch of MM. Less than 2% of the observations in that branch take the value of CH, and the remaining 98% take the value of MM.

```r
plot(OJ.tree)
text(OJ.tree, pretty = 0)
```



c) Create a plot of the tree, interpret the results.
The main branch is LoyalCH, and if you met the threshold or didn't, the tree split up into the two sub-LoyalCH paths, one where it has many terminal nodes, meaning more outcomes, and one that has few terminal nodes, meaning if you follow this path then you are easily classified.

```
OJ.test.pred <- predict(OJ.tree, OJ.test, type = "class")
table(OJ.test.pred, OJ.test$Purchase)
```

d) 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?

```
##
## OJ.test.pred  CH  MM
##           CH 147  24
##           MM  24  75
```

```
mean(OJ.test.pred != OJ.test$Purchase)
```

```
## [1] 0.1777778
```

The test error rate here is found to 17%.

```r
cv.oj <- cv.tree(OJ.tree, FUN = prune.misclass)
names(cv.oj)
```

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

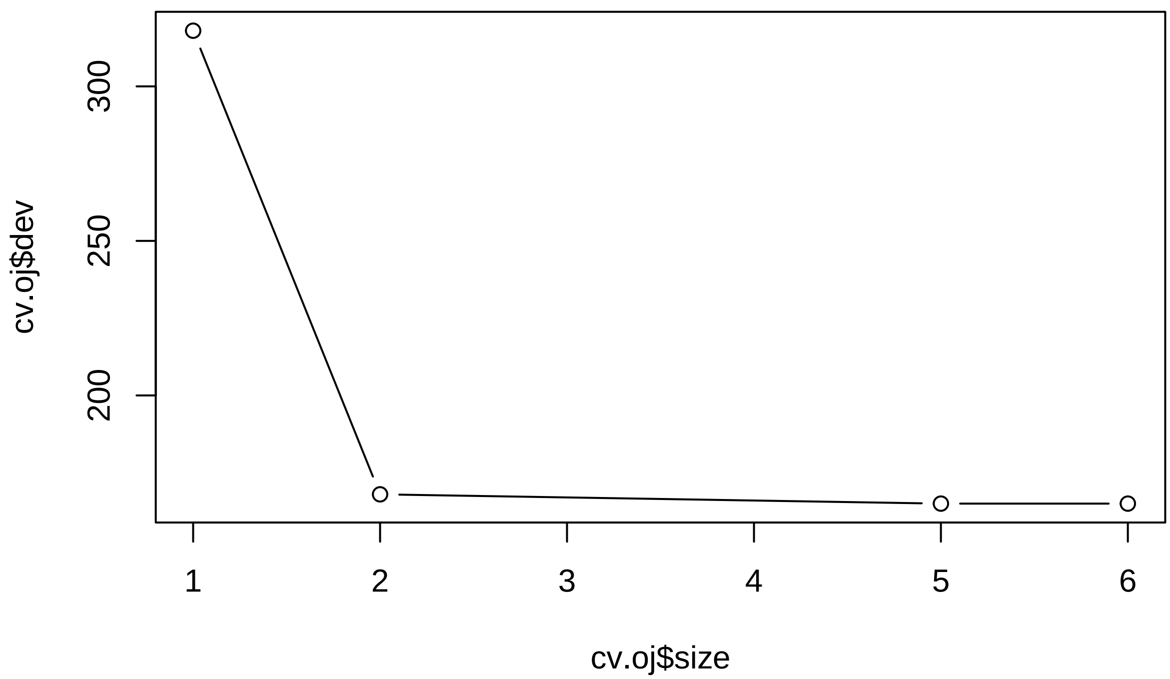
```
## [1] "size"   "dev"    "k"      "method"
```

```r
cv.oj
```

```
## $size
## [1] 6 5 2 1
##
## $dev
## [1] 165 165 168 318
##
## $k
## [1] -Inf    0    6  163
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
plot(cv.oj$size, cv.oj$dev, type = "b")
```

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



y-axis.

```
which.min(cv.oj$size) # 4 is the lowest
```

```
## [1] 4
```

```
which.min(cv.oj$size)
```

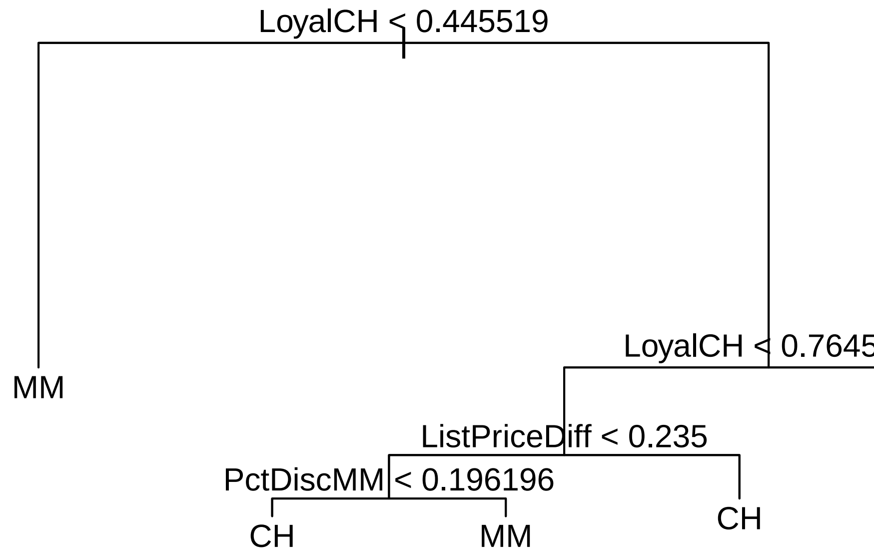
g) Which tree size corresponds to the lowest cross-validated classification error rate?

```
## [1] 4
```

4 is the lowest size that is best fit for the model.

```
prune.oj <- prune.misclass(OJ.tree, best = 4)
plot(prune.oj)
text(prune.oj, pretty = 0)
```

h) 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.
The pruned tree provided a different tree than before, so there is no need to use 5 as best size.

```
summary(OJ.tree)
```

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

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ.train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "ListPriceDiff" "PctDiscMM"
## Number of terminal nodes:  6
## Residual mean deviance:  0.7964 = 632.4 / 794
## Misclassification error rate: 0.1713 = 137 / 800
```

```
summary(prune.oj)
```

```
##
## Classification tree:
## snip.tree(tree = OJ.tree, nodes = 2L)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "ListPriceDiff" "PctDiscMM"
## Number of terminal nodes:  5
```

```
## Residual mean deviance:  0.822 = 653.5 / 795
## Misclassification error rate: 0.1713 = 137 / 800
```

They are same. I experimented with the seed, and the similarities between the two are directly because of the seed that was set at 200. If I change the seed, then the two will vary slightly. But for the case of simplicity, I will keep the seed at 200, which was set by the professor.

```
OJ.test.pred <- predict(OJ.tree, OJ.test, type = "class")
table(OJ.test.pred, OJ.test$Purchase)
```

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

```
##
## OJ.test.pred  CH  MM
##           CH 147  24
##           MM  24  75
```

```
print(c("Unpruned test error rate",mean(OJ.test.pred != OJ.test$Purchase)))
```

```
## [1] "Unpruned test error rate" "0.177777777777778"
```

```
OJ.prune.pred <- predict(prune.oj, OJ.test, type = "class")
table(OJ.prune.pred, OJ.test$Purchase)
```

```
##
## OJ.prune.pred  CH  MM
##            CH 147  24
##            MM  24  75
```

```
print(c("Pruned test error rate",mean(OJ.prune.pred != OJ.test$Purchase)))
```

```
## [1] "Pruned test error rate" "0.177777777777778"
```

As one would expect, the error rates are identical again just like the training error rates were. Again, if we change the seed these will also change too.

```
ncol(OJ.train) - 1
```

k) Fit a bagging model to the training set with Purchase as the response and the other variables as predictors. Use 1,000 trees (ntree = 1000). Use the importance() function to determine which variables are most important.

```
## [1] 17
```

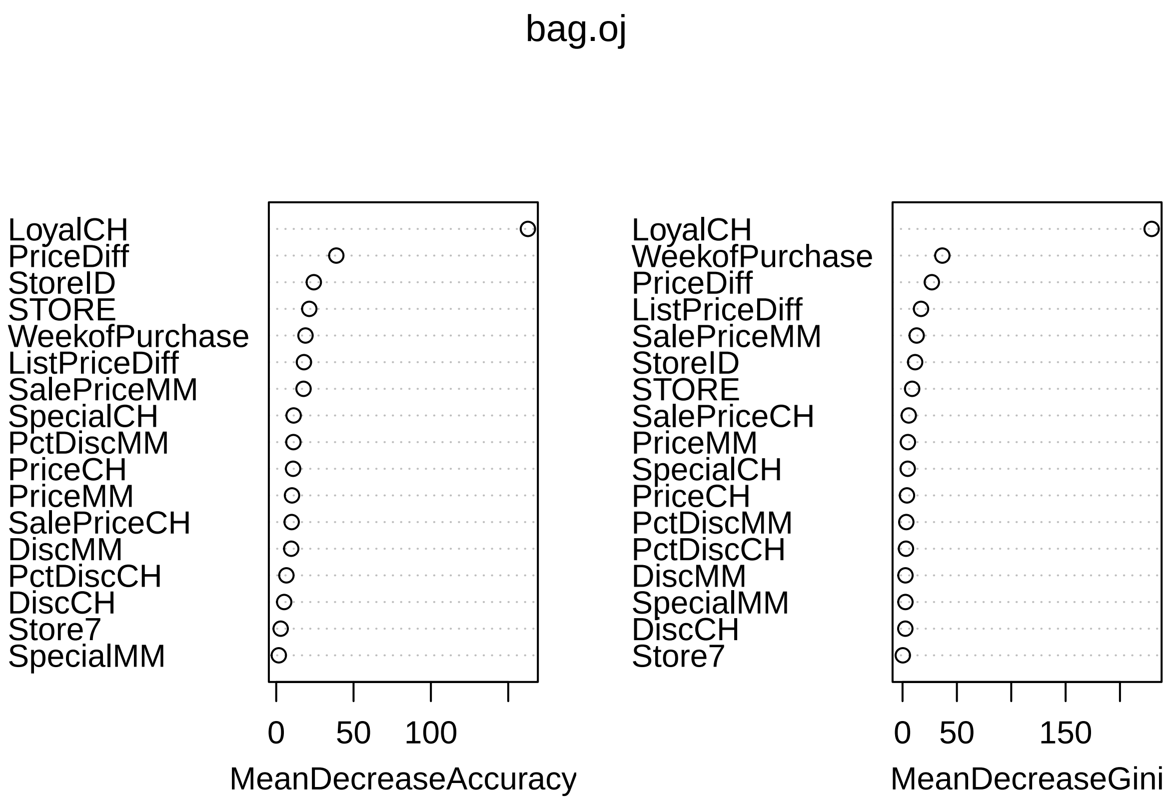
```r
bag.oj <- randomForest(Purchase ~ ., data = OJ.train, mtry = 17, ntree = 1000, importance = T)
bag.oj
```

```
##
## Call:
##  randomForest(formula = Purchase ~ ., data = OJ.train, mtry = 17,      ntree = 1000, importance = T)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 17
##
##         OOB estimate of  error rate: 21.25%
## Confusion matrix:
##     CH  MM class.error
## CH 395  87   0.1804979
## MM  83 235   0.2610063
```

```r
importance(bag.oj)
```

```
##                          CH         MM MeanDecreaseAccuracy MeanDecreaseGini
## WeekofPurchase  11.52366903  14.338286            18.912222       36.6145004
## StoreID          7.92375214  23.042325            24.302699       11.5734760
## PriceCH          5.67055317   8.388555            10.910932        4.0779326
## PriceMM          8.73787293   4.763314            10.198585        4.9460420
## DiscCH           0.22469914   5.985853             5.124062        2.5660685
## DiscMM           6.95607100   6.192964             9.690413        2.6930698
## SpecialCH       10.10262663   4.623100            11.256978        4.7606557
## SpecialMM        0.03223215   1.952477             1.660351        2.6475442
## LoyalCH        100.53521666 155.081177           162.724141      229.1272114
## SalePriceMM      5.82645302  17.177485            17.655577       12.9825355
## SalePriceCH      8.27224617   4.471144             9.967056        5.7057716
## PriceDiff       23.63852089  29.254158            38.835960       26.9606811
## Store7           1.14107653   2.677296             2.870949        0.2638024
## PctDiscMM        6.14264310   9.370572            11.098326        3.4556880
## PctDiscCH       -0.71074959   9.106061             6.578887        3.1414427
## ListPriceDiff   15.98535230   6.008752            17.925661       17.0072410
## STORE           10.21409923  18.283972            21.360629        8.8388805
```

```r
varImpPlot(bag.oj)
```




```
bag.oj.pred <- predict(bag.oj, newdata = OJ.test)
table(bag.oj.pred, OJ.test$Purchase)
```

l) Use the bagging model to predict the response on the test data. Compute the test error rates.

```
##
## bag.oj.pred  CH  MM
##          CH 146  21
##          MM  25  78
```

```
mean(bag.oj.pred != OJ.test$Purchase)
```

```
## [1] 0.1703704
```

```
sqrt(ncol(OJ.train) -1)
```

m) Fit a random forest model to the training set with with Purchase as the response and the other variables as predictors. Use 1,000 trees (ntree = 1000). Use the importance() function to determine which variables are most important.

```
## [1] 4.123106
```

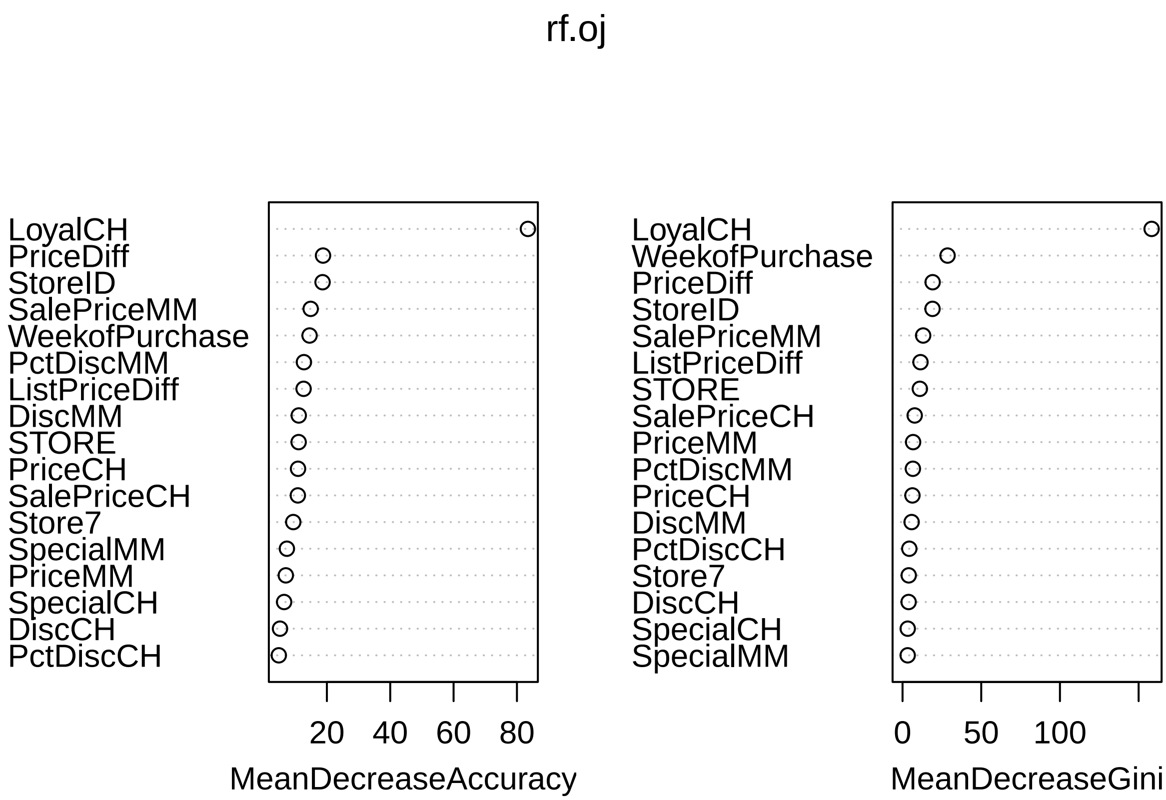
```r
rf.oj <- randomForest(Purchase ~ ., data = OJ.train, mtry = 4, importance = T)
rf.oj
```

```
##
## Call:
##  randomForest(formula = Purchase ~ ., data = OJ.train, mtry = 4,      importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 20.62%
## Confusion matrix:
##     CH  MM class.error
## CH 409  73   0.1514523
## MM  92 226   0.2893082
```

```r
importance(rf.oj)
```

```
##                           CH        MM MeanDecreaseAccuracy MeanDecreaseGini
## WeekofPurchase  6.42049435 11.883905            14.536368        28.605301
## StoreID         6.15662370 18.577110            18.638258        19.046636
## PriceCH         8.02421281  5.300843            10.893290         6.292529
## PriceMM         5.05825147  3.713223             7.038659         6.716811
## DiscCH         -0.09509072  7.458482             5.201732         3.871706
## DiscMM          7.53486609  7.284313            11.122560         5.787801
## SpecialCH       4.82462999  3.016908             6.495153         3.302863
## SpecialMM       2.13968811  6.849358             7.378164         3.266877
## LoyalCH        55.70114017 79.679634            83.457143       158.258848
## SalePriceMM     9.73279538  9.273382            14.887677        13.102843
## SalePriceCH     7.05270342  7.091315            10.832771         7.724022
## PriceDiff      10.58097036 14.983896            18.781723        19.131443
## Store7          5.76605384  8.584945             9.389279         3.935823
## PctDiscMM       8.24029825  7.295805            12.754095         6.584049
## PctDiscCH       1.92374081  4.999250             4.821712         4.305319
## ListPriceDiff   9.03387988  9.187913            12.648142        11.408676
## STORE           6.59676483  9.903088            11.085625        10.989150
```

```r
varImpPlot(rf.oj)
```




```
rf.oj.pred <- predict(rf.oj, newdata = OJ.test)
table(rf.oj.pred, OJ.test$Purchase)
```

n) Use the random forest to predict the response on the test data. Compute the test error rates.

```
##
## rf.oj.pred  CH  MM
##         CH 148  27
##         MM  23  72
```

```
mean(rf.oj.pred != OJ.test$Purchase)
```

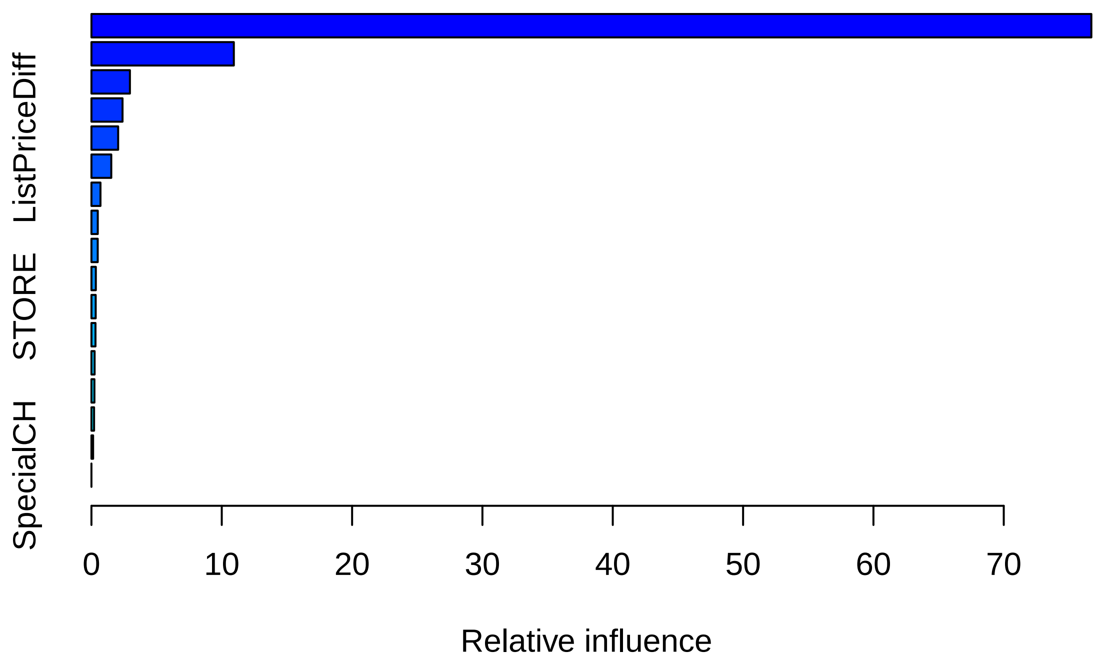
```
## [1] 0.1851852
```

```
binary.purchase <- ifelse(OJ.train$Purchase == "CH", 1,0)
boost.oj <- gbm(binary.purchase ~ ., data = OJ.train[,colnames(OJ.train) != "Purchase"], distribution =
boost.oj
```

o) Fit a boosting model to the training set with Purchase as the response and the other variables as predictors. Use 1,000 trees, and a shrinkage value of 0.01 (lambda = 0.01). Which predictors appear to be the most important?

```
## gbm(formula = binary.purchase ~ ., distribution = "bernoulli",
##     data = OJ.train[, colnames(OJ.train) != "Purchase"], n.trees = 1000,
##     shrinkage = 0.01)
## A gradient boosted model with bernoulli loss function.
## 1000 iterations were performed.
## There were 17 predictors of which 16 had non-zero influence.
```

```
summary(boost.oj)
```




```
##                           var    rel.inf
## LoyalCH               LoyalCH 76.7207959
## PriceDiff           PriceDiff 10.9248998
## StoreID               StoreID  2.9523840
## SalePriceMM       SalePriceMM  2.3856547
## ListPriceDiff   ListPriceDiff  2.0505331
## WeekofPurchase WeekofPurchase  1.5222454
## DiscCH                 DiscCH  0.6939194
## PriceCH               PriceCH  0.4862603
## SalePriceCH       SalePriceCH  0.4825903
## PriceMM               PriceMM  0.3377038
## STORE                   STORE  0.3274832
```

```
## PctDiscCH          PctDiscCH  0.3111644
## Store7                Store7  0.2432096
## PctDiscMM          PctDiscMM  0.2295017
## DiscMM                DiscMM  0.2034040
## SpecialMM          SpecialMM  0.1282502
## SpecialCH          SpecialCH  0.0000000
```

LoyalCH and PriceDiff are the most important ones, but LoyalCH is the most important in this model.

```
# First, do cross-validation to find the best number of trees
boost.cv.oj <- gbm(binary.purchase ~ ., data = OJ.train[,colnames(OJ.train) != "Purchase"], distributio
which.min(boost.cv.oj$cv.error) # 900 trees is the best
```

p) Use the boosting model to predict the response on the test data. Compute the test error rates.

```
## [1] 953
```

```
# now perform predictions with 900 trees
boost.oj.pred <- predict(boost.cv.oj, newdata = OJ.test, n.trees = 900, type = "response")
boost.oj.pred <- ifelse(boost.oj.pred > 0.5, "CH", "MM")
table(boost.oj.pred, OJ.test$Purchase)
```

```
##
## boost.oj.pred  CH  MM
##            CH 154  22
##            MM  17  77
```

```
mean(boost.oj.pred != OJ.test$Purchase)
```

```
## [1] 0.1444444
```

```
log.oj <- glm(binary.purchase ~ ., data = OJ.train[,colnames(OJ.train) != "Purchase"], family = binomia
log.oj.pred <- predict(log.oj, newdata = OJ.test, type = "response")
```

q) Fit a logistic regression to the training set with Purchase as the response and the other variables as predictors, and predict on the test data. Compute the test error rates.

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
log.oj.pred <- ifelse(log.oj.pred > 0.5, "CH", "MM")
mean(log.oj.pred != OJ.test$Purchase)
```

```
## [1] 0.1555556
```

```r
summary(log.oj)
```

r) Rank the significance of the coefficients of the logistic regression. Is the result consistent with (k)?

```
##
## Call:
## glm(formula = binary.purchase ~ ., family = binomial(link = "logit"),
##     data = OJ.train[, colnames(OJ.train) != "Purchase"])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8167  -0.5298   0.2289   0.5492   2.7638
##
## Coefficients: (5 not defined because of singularities)
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -4.154778   2.302143  -1.805  0.07111 .
## WeekofPurchase    0.007607   0.012741   0.597  0.55047
## StoreID           0.251174   0.159990   1.570  0.11643
## PriceCH          -4.345284   2.108327  -2.061  0.03930 *
## PriceMM           3.314032   1.024782   3.234  0.00122 **
## DiscCH          -18.653271  20.838025  -0.895  0.37070
## DiscMM          -28.001029  10.396661  -2.693  0.00708 **
## SpecialCH        -0.649287   0.390816  -1.661  0.09664 .
## SpecialMM        -0.331091   0.311964  -1.061  0.28855
## LoyalCH           6.310054   0.465670  13.550  < 2e-16 ***
## SalePriceMM             NA         NA      NA       NA
## SalePriceCH             NA         NA      NA       NA
## PriceDiff               NA         NA      NA       NA
## Store7Yes        -0.640207   0.829980  -0.771  0.44050
## PctDiscMM        54.833568  21.806070   2.515  0.01192 *
## PctDiscCH        42.727682  39.344567   1.086  0.27748
## ListPriceDiff           NA         NA      NA       NA
## STORE                   NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1075.18  on 799  degrees of freedom
## Residual deviance:  620.47  on 787  degrees of freedom
## AIC: 646.47
##
## Number of Fisher Scoring iterations: 5
```

This is significant when it comes to showing the LoyalCH is the most significant. However, PriceDiff is listed as NA in the logistic model, thus not having any significance.

```r
print(c("Unpruned test error rate",mean(OJ.test.pred != OJ.test$Purchase)))
```

s) Compare the test error rates between the unpruned trees, pruned trees, bagging, random forest, boosting, and logistic regression. Which performs the best?

```
## [1] "Unpruned test error rate" "0.177777777777778"
```

```
print(c("Pruned test error rate",mean(OJ.prune.pred != OJ.test$Purchase)))
```

```
## [1] "Pruned test error rate" "0.177777777777778"
```

```
print(c("Bagging test error rate",mean(bag.oj.pred != OJ.test$Purchase)))
```

```
## [1] "Bagging test error rate" "0.17037037037037"
```

```
print(c("Random Forest test error rate",mean(rf.oj.pred != OJ.test$Purchase)))
```

```
## [1] "Random Forest test error rate" "0.185185185185185"
```

```
print(c("Boosting test error rate",mean(boost.oj.pred != OJ.test$Purchase)))
```

```
## [1] "Boosting test error rate" "0.144444444444444"
```

```
print(c("Logistic Regression test error rate",mean(log.oj.pred != OJ.test$Purchase)))
```

```
## [1] "Logistic Regression test error rate" "0.155555555555556"
```

Looking at each of the error rates, it looks like Logistic Regression and Boosting methods perform the best. This could be because of the thresholds set in the code that make it perform better, and more accurately.