ch8exercises

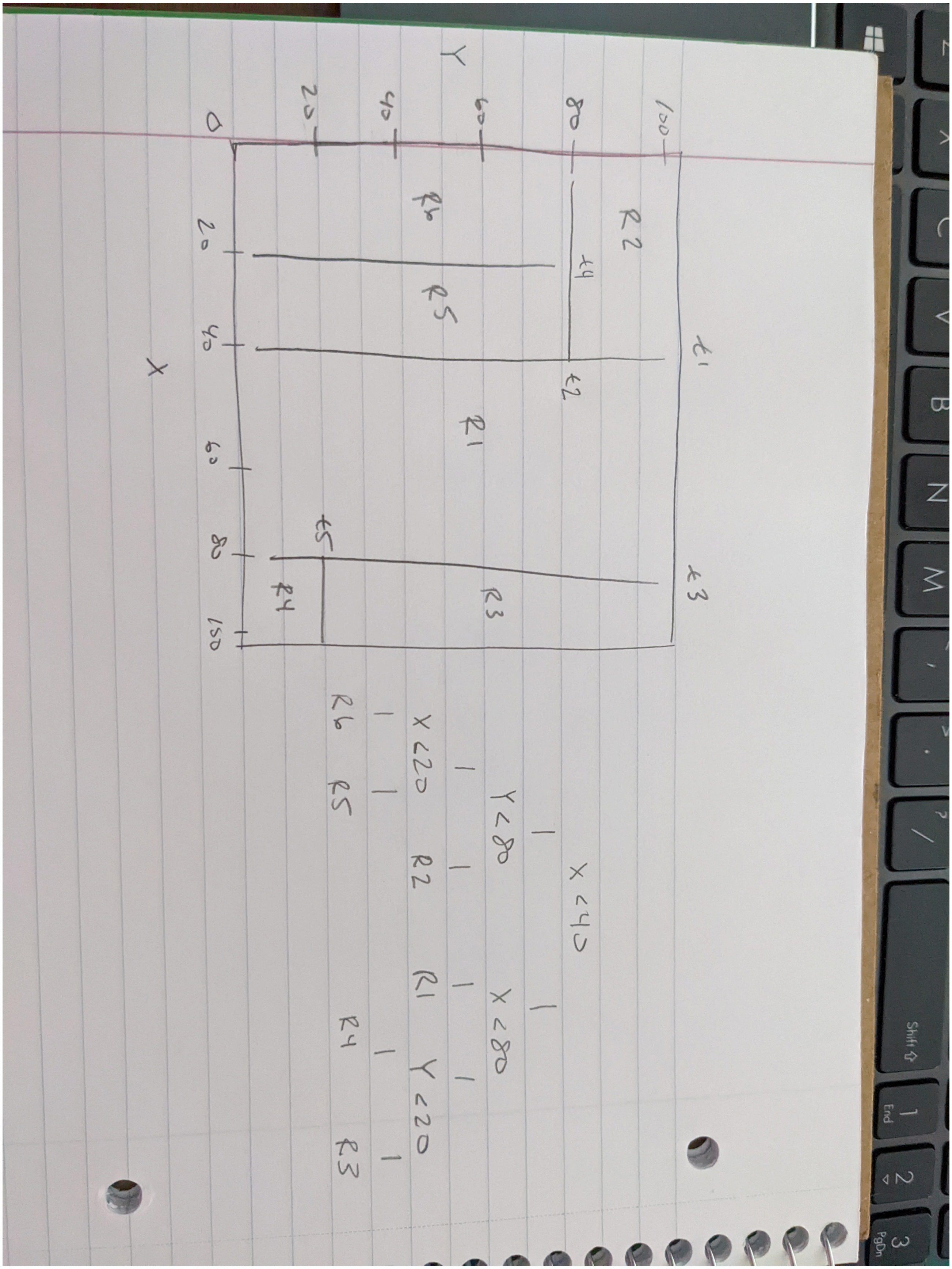
Lauren Temple

2/24/2022

```
pacman::p_load(ISLR2, tree, randomForest, gbm, BART, caTools)
```

8.1 Draw an example (of your own invention) of a partition of two-dimensional 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.

```
knitr::include_graphics("8.1.jpg")
```

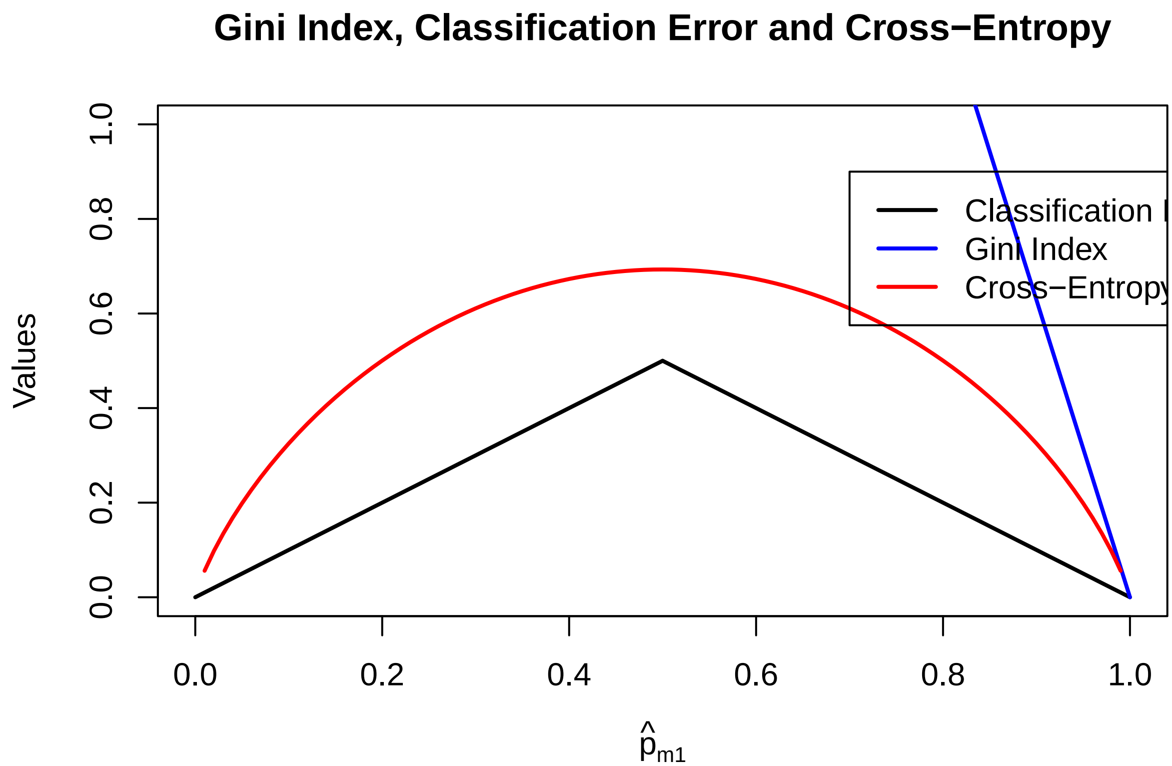

t1
t2
t3
t4
t5
R1
R2
R3
R4
R5
R6
X
Y
X < 40
Y < 80
X < 80
X < 20
Y < 20

8.2 It is mentioned in Section 8.2.3 that boosting using depth-one trees (or stumps) leads to an additive model, Explain why this is the case. You can begin with (8.12) in Algorithm 8.2.

This is because each model consists of a single split using one distinct variable. Therefore the total number of decision trees is the same as the number of predictors. A new model is fit on the residuals of the previous model and the new models output is then added to the previous models, making the final model additive.

8.3 Consider the Gini index, classification error, and entropy in a simple classification setting with two classes. Create a single plot that displays each of these quantities as a function of ˆpm1. The x-axis should display ˆpm1, ranging from 0 to 1, and the y-axis should display the value of the Gini index, classification error, and entropy. Hint: In a setting with two classes, pˆm1 = 1 − pˆm2. You could make this plot by hand, but it will be much easier to make in R.

```r
#classification error
p1 <- seq(0, 1, 0.01)
E1 <- 1-p1[51:101]
E2 <- 1-(1-p1[1:51])

plot(1, type="n", main="Gini Index, Classification Error and Cross-Entropy",
     xlab=expression(hat(p)[m1]), ylab="Values", xlim=seq(0,1), ylim=c(0, 1))
points(x=p1[1:51], y = c(E2), type = "l", lwd=2)
points(x=p1[51:101], y = c(E1), type = "l", lwd=2)

#Gini Index
G <- 2*pi*(1-p1)
lines(p1, G, col="blue", lwd= 2)

#Cross Entropy
D <- -p1*log(p1)-(1-p1)*log(1-p1)
lines(p1, D, col="red", lwd= 2)

legend(0.7, 0.9, legend= c("Classification Error", "Gini Index", "Cross-Entropy"),
       col= c("black", "blue", "red"), lty= c(1,1,1), lwd=c(2,2,2))
```



8.5 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. 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?

Majority Voting: P(Class is Red | X) $<$ 0.5 = 4 P(Class is Red | X) $>$ 0.5 = 6 X is classified as red

Average probability: The average P(Class is Red | X) is 4.5/10= 0.45 X is classified as green

8.7 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.

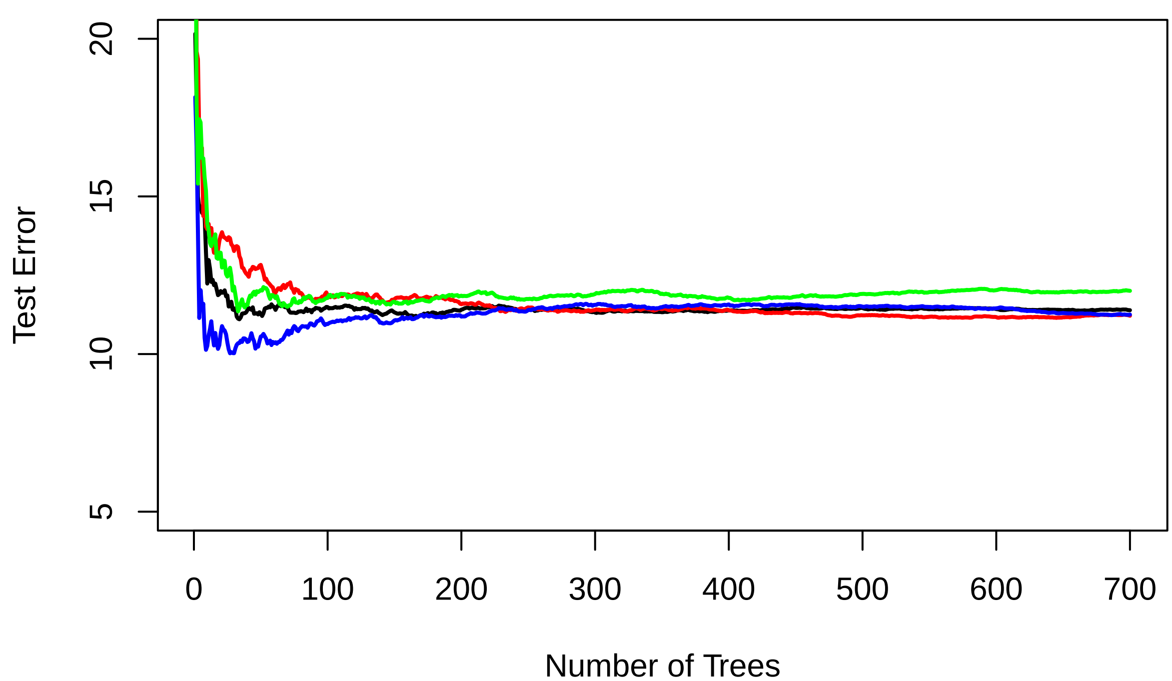
```
set.seed(42)
data(Boston)
df <- Boston
sample.data <- sample.split(df$medv, SplitRatio= 0.70)
train.set <- subset(df, select= -c(medv), sample.data==T) #drop the medv column
test.set <- subset(df, select=-c(medv), sample.data==F)
train.Y <- subset(df$medv, sample.data==T)
test.Y <- subset(df$medv, sample.data==F)
```

```
#four random forest models
p <- 13
rf1 <- randomForest(train.set, train.Y, test.set, test.Y, mtry= p, ntree= 700)
```

```
## Warning in randomForest.default(train.set, train.Y, test.set, test.Y, mtry =
## p, : invalid mtry: reset to within valid range
```

```r
rf2 <- randomForest(train.set, train.Y, test.set, test.Y, mtry= p/2, ntree= 700)

rf3 <- randomForest(train.set, train.Y, test.set, test.Y, mtry= p/3, ntree= 700)

rf4 <- randomForest(train.set, train.Y, test.set, test.Y, mtry= p/4, ntree= 700)
```

```r
x.axis <- seq(1, 700, 1)
plot(x.axis, rf1$test$mse, xlab= "Number of Trees", ylab= "Test Error", ylim= c(5,20), type="l", lwd= 2
lines(x.axis, rf2$test$mse, col="red", lwd=2)
lines(x.axis, rf3$test$mse, col="blue", lwd=2)
lines(x.axis, rf4$test$mse, col="green", lwd=2)
```



The test error decreases as the number of trees increases. Test error gets lower as m decreases from m=p up to m=p/3 and after that there is not much change.

8.8 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.

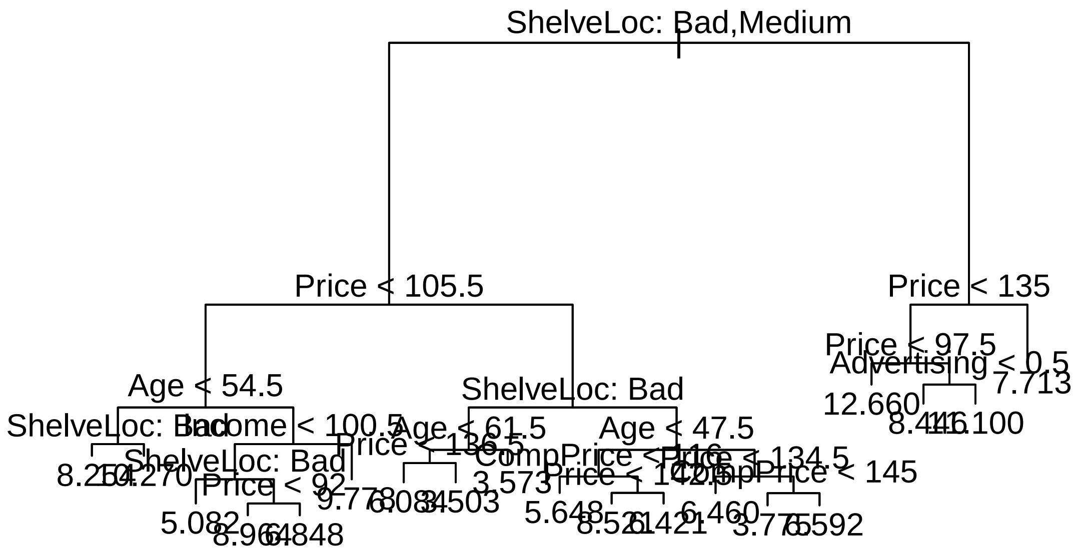
```r
set.seed(42)
data("Carseats")
df <- Carseats
sample.data <- sample.split(df$Sales, SplitRatio=0.70)

train.set <- subset(df, sample.data==T)
test.set <- subset(df, sample.data==F)
```

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

```r
tree.carseats <- tree(Sales ~., data= train.set)
summary(tree.carseats)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = train.set)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Income"      "CompPrice"
## [6] "Advertising"
## Number of terminal nodes:  19
## Residual mean deviance:  2.536 = 661.9 / 261
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.13300 -1.14700 -0.06807  0.00000  1.12900  4.05600
```

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



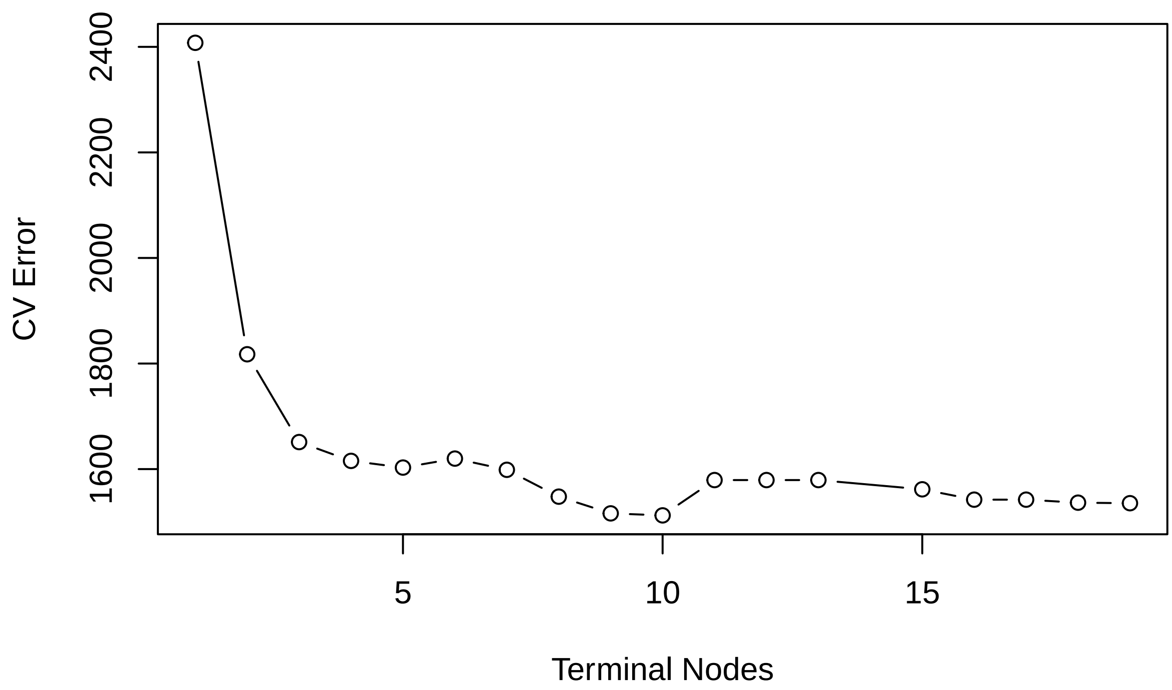

```r
#test mse
tree.pred <- predict(tree.carseats, test.set)
test.mse <- mean((tree.pred-test.set$Sales)^2)
test.mse
```

```
## [1] 3.307985
```

Shelf location and price are important predictors. The test MSE is 3.308

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

```r
set.seed(2)
cv.carseats <- cv.tree(tree.carseats)
plot(cv.carseats$size, cv.carseats$dev, xlab="Terminal Nodes", ylab="CV Error", type="b")
```



The CV Error is lowest for a tree with 9 terminal nodes

```
prune.carseats <- prune.tree(tree.carseats, best= 9)
tree.pred <- predict(prune.carseats, test.set)
test.mse <- mean((tree.pred-test.set$Sales)^2)
test.mse
```

```
## [1] 3.930777
```

I do not see the test mse improve

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.

```
set.seed(42)
bag.carseats <- randomForest(Sales ~., data= train.set, mtry= 10, importance=T)
importance(bag.carseats)
```

```
##                %IncMSE IncNodePurity
## CompPrice    27.683702     222.30466
## Income       11.400580     136.20583
## Advertising  20.286275     159.59131
## Population   -4.392642      74.73135
## Price        69.050043     659.17173
## ShelveLoc    69.486756     711.00020
```

```
## Age          21.985647     206.92845
## Education     2.342229      66.15492
## Urban        -1.043762      10.86892
## US            6.187962      14.82505
```

```
bag.yhat <- predict(bag.carseats, newdata= test.set)
mean((bag.yhat-test.set$Sales)^2)
```

```
## [1] 1.996893
```

The most important variables are price and shelf location as we saw previously. The test MSE is 1.99 which is improved from teh random forest method.

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.

```
rf1.carseats <- randomForest(Sales~., data= train.set, mtry= 10/2, importance= T)
rf2.carseats <- randomForest(Sales~., data= train.set, mtry= sqrt(10), importance= T)
rf3.carseats <- randomForest(Sales~., data= train.set, mtry= 10/4, importance= T)

importance(rf1.carseats)
```

```
##                 %IncMSE IncNodePurity
## CompPrice   17.1512392     207.63264
## Income       8.4966741     154.05599
## Advertising 17.7611077     180.17925
## Population  -2.1512650     103.48535
## Price       55.0603497     587.36061
## ShelveLoc   64.5649093     645.49925
## Age         17.7720640     235.48618
## Education    1.1196125      78.12832
## Urban       -0.9352489      10.87270
## US           5.8519594      26.16095
```

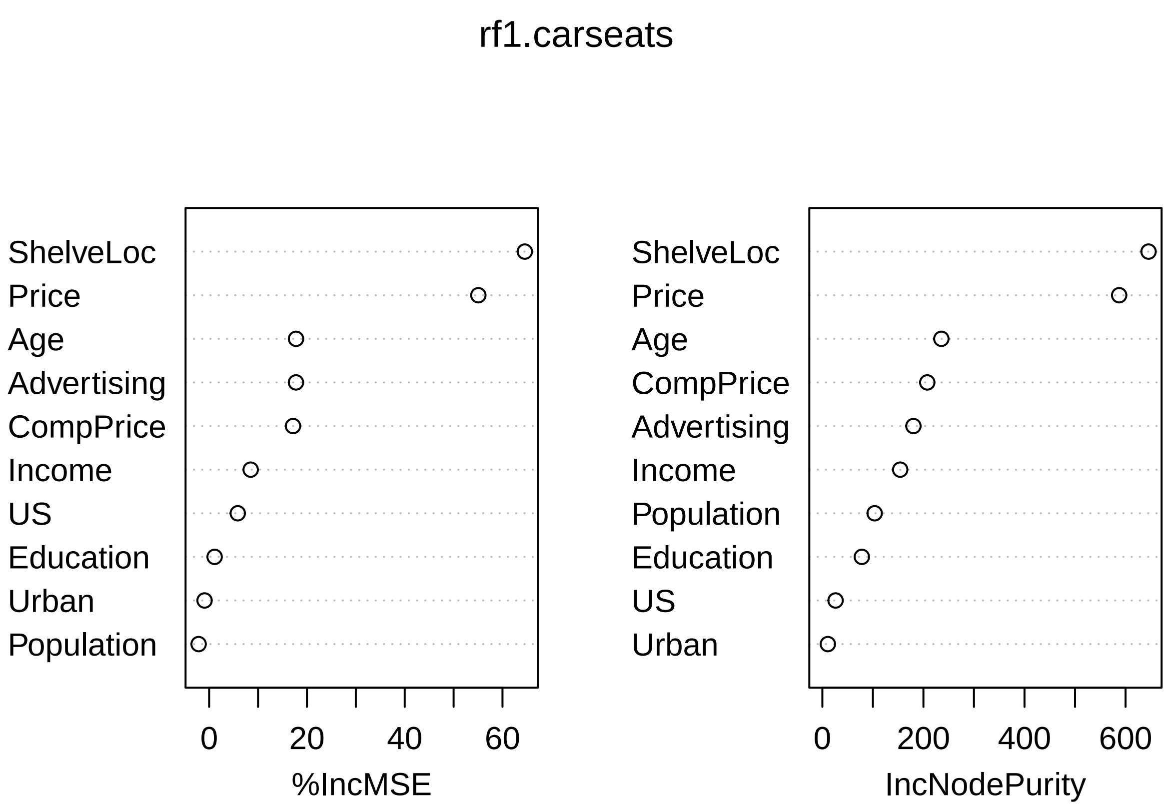
```
importance(rf2.carseats)
```

```
##                 %IncMSE IncNodePurity
## CompPrice   14.0695613     198.67337
## Income       5.7122899     174.70418
## Advertising 15.9435331     188.73969
## Population  -0.8478586     137.12146
## Price       41.1022267     538.04874
## ShelveLoc   49.3091606     548.52842
## Age         17.6845148     252.13895
## Education    1.7979946      96.42484
## Urban        1.7226146      17.57218
## US           4.1964423      35.43860
```

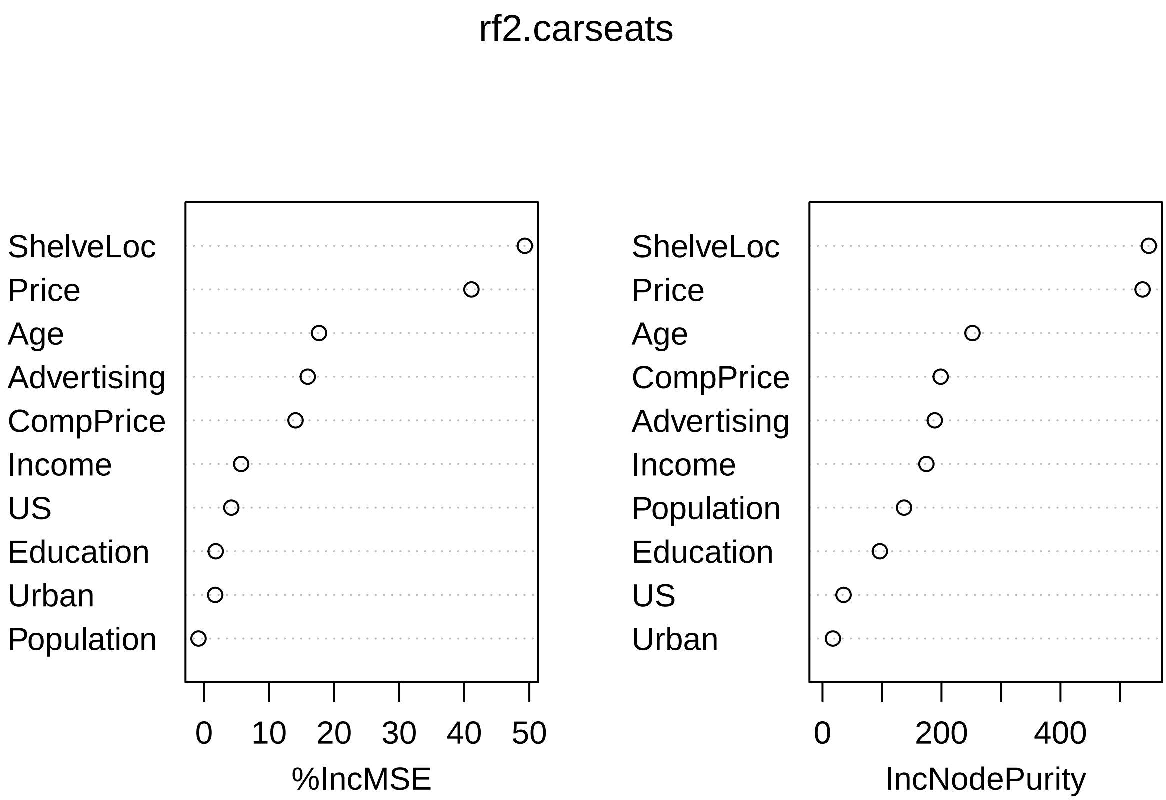
```r
importance(rf3.carseats)
```

```
##                %IncMSE IncNodePurity
## CompPrice   10.530404     200.42533
## Income       4.954577     191.78712
## Advertising 13.165061     198.07836
## Population  -2.601226     170.82258
## Price       34.466366     460.04152
## ShelveLoc   41.164442     470.96073
## Age         14.557953     252.26387
## Education    1.491281     111.32409
## Urban       -1.111478      23.36413
## US           4.822974      40.75881
```

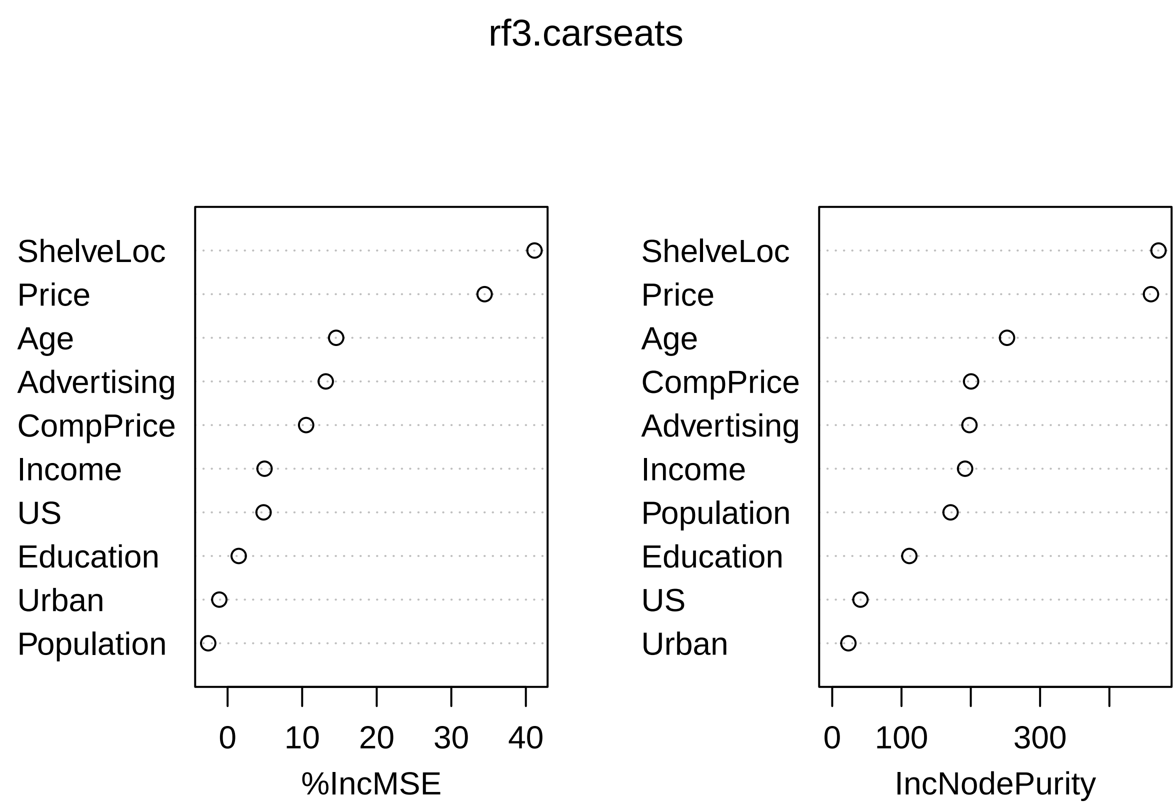
```r
varImpPlot(rf1.carseats)
```




```r
varImpPlot(rf2.carseats)
```




```
varImpPlot(rf3.carseats)
```



All models show that the most important variables are Shelve Location and Price. As m decreases the MSE becomes smaller.

f. Now analyze the data using BART, and report your results.

8.11 Caravan data set a. Create a training set consisting of the first 1,000 observations, and a test set consisting of the remaining observations.

```
Caravan$Purchase01=rep(NA, 5822)
for(i in 1:5822) if (Caravan$Purchase[i] == "Yes")
  (Caravan$Purchase01[i]=1) else (Caravan$Purchase01[i]=0)
```

```
train.set <- Caravan[1:1000, ]
test.set <- Caravan[1001:5822, ]
```

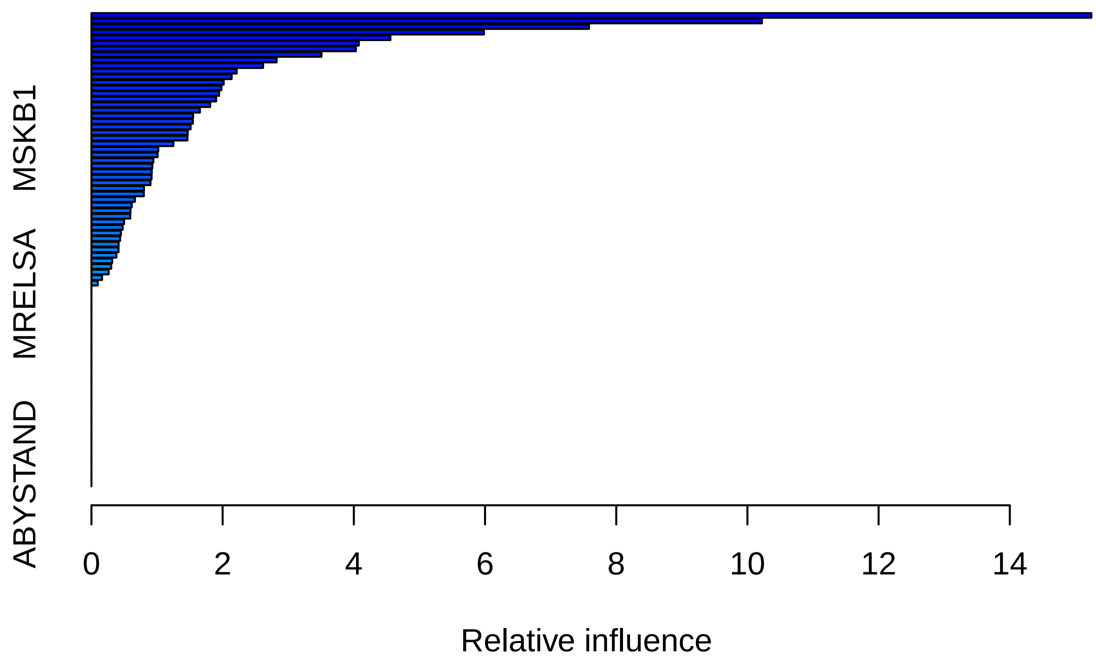
b. Fit a boosting model to the training set with Purchase as the response and the other variables as predictors. Use 1,000 trees, and a shrinkage value of 0.01. Which predictors appear to be the most important?

```
set.seed(42)
boost.Caravan <- gbm(Purchase01~.-Purchase, data= train.set, distribution= "bernoulli", n.trees= 1000, s
```

```
## Warning in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
## variable 50: PVRAAUT has no variation.
```

```
## Warning in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
## variable 71: AVRAAUT has no variation.
```

```
summary(boost.Caravan)
```




```
##               var     rel.inf
## PPERSAUT PPERSAUT 15.24304059
## MKOOPKLA MKOOPKLA 10.22049754
## MOPLHOOG MOPLHOOG  7.58473391
## MBERMIDD MBERMIDD  5.98365038
## PBRAND     PBRAND  4.55749118
## ABRAND     ABRAND  4.07601736
## MINK3045 MINK3045  4.03149141
## MGODGE     MGODGE  3.50618597
## MOSTYPE   MOSTYPE  2.82332650
## MAUT2       MAUT2  2.61711991
## MSKC         MSKC  2.21439111
## MAUT1       MAUT1  2.13764619
## MBERARBG MBERARBG  2.01645301
## MSKA         MSKA  1.98039700
## MGODPR     MGODPR  1.94608284
## PWAPART   PWAPART  1.90065796
## MGODOV     MGODOV  1.81046263
## MRELGE     MRELGE  1.65327955
## MINK7512 MINK7512  1.54952273
```

```
## MBERHOOG MBERHOOG  1.54562792
## MINKGEM   MINKGEM  1.51129086
## PBYSTAND PBYSTAND  1.46885445
## MSKB1        MSKB1  1.46349386
## MFWEKIND MFWEKIND  1.24890126
## MINKM30   MINKM30  1.01877067
## APERSAUT APERSAUT  1.00947264
## MSKD          MSKD  0.94342296
## MOSHOOFD MOSHOOFD  0.92805596
## MFGEKIND MFGEKIND  0.92012209
## MAUT0        MAUT0  0.91661495
## MGODRK      MGODRK  0.90097295
## MOPLMIDD MOPLMIDD  0.80067001
## MRELOV      MRELOV  0.79866885
## MHHUUR      MHHUUR  0.66251044
## MBERBOER MBERBOER  0.61490907
## MBERARBO MBERARBO  0.59493791
## PMOTSCO   PMOTSCO  0.59140712
## MSKB2        MSKB2  0.49738895
## PLEVEN      PLEVEN  0.47656908
## MINK4575 MINK4575  0.44932585
## MZPART      MZPART  0.43764686
## MHKOOP      MHKOOP  0.41454005
## MGEMOMV   MGEMOMV  0.41336506
## MBERZELF MBERZELF  0.38071586
## MZFONDS   MZFONDS  0.31613260
## MINK123M MINK123M  0.30173680
## MGEMLEEF MGEMLEEF  0.26253060
## MOPLLAAG MOPLLAAG  0.16165917
## MFALLEEN MFALLEEN  0.09723737
## MAANTHUI MAANTHUI  0.00000000
## MRELSA      MRELSA  0.00000000
## PWABEDR   PWABEDR  0.00000000
## PWALAND   PWALAND  0.00000000
## PBESAUT   PBESAUT  0.00000000
## PVRAAUT   PVRAAUT  0.00000000
## PAANHANG PAANHANG  0.00000000
## PTRACTOR PTRACTOR  0.00000000
## PWERKT      PWERKT  0.00000000
## PBROM        PBROM  0.00000000
## PPERSONG PPERSONG  0.00000000
## PGEZONG   PGEZONG  0.00000000
## PWAOREG   PWAOREG  0.00000000
## PZEILPL   PZEILPL  0.00000000
## PPLEZIER PPLEZIER  0.00000000
## PFIETS      PFIETS  0.00000000
## PINBOED   PINBOED  0.00000000
## AWAPART   AWAPART  0.00000000
## AWABEDR   AWABEDR  0.00000000
## AWALAND   AWALAND  0.00000000
## ABESAUT   ABESAUT  0.00000000
## AMOTSCO   AMOTSCO  0.00000000
## AVRAAUT   AVRAAUT  0.00000000
## AAANHANG AAANHANG  0.00000000
```

```
## ATRACTOR ATRACTOR  0.00000000
## AWERKT      AWERKT  0.00000000
## ABROM        ABROM  0.00000000
## ALEVEN      ALEVEN  0.00000000
## APERSONG APERSONG  0.00000000
## AGEZONG   AGEZONG  0.00000000
## AWAOREG   AWAOREG  0.00000000
## AZEILPL   AZEILPL  0.00000000
## APLEZIER APLEZIER  0.00000000
## AFIETS      AFIETS  0.00000000
## AINBOED   AINBOED  0.00000000
## ABYSTAND ABYSTAND  0.00000000
```

PPERSAUT, MKOOPKLA are the most important variables according to this booting.

c. Use the boosting model to predict the response on the test data. Predict that a person will make a purchase if the estimated probability of purchase is greater than 20 %. Form a confusion matrix. What fraction of the people predicted to make a purchase do in fact make one? How does this compare with the results obtained from applying KNN or logistic regression to this data set?

```
probs.Caravan <- predict(boost.Caravan, newdata= test.set, n.trees= 1000, type="response")

preds <- rep("No", 4822)
preds[probs.Caravan>0.20]="Yes"

#Confusion Matrix
actual <- test.set$Purchase
table(actual, preds)
```

```
##       preds
## actual   No  Yes
##    No  4415  118
##    Yes  257   32
```

The model predicted yes 150 times, out of those it was a true positive 32 times, giving it a rate of 21.33%.

```
glm.fit <- glm(Purchase~.-Purchase01, data= train.set, family= binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.probs <- predict(glm.fit, test.set, type="response")
```

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

```
glm.preds <- rep("No", 4822)
glm.preds[glm.probs>0.2]="Yes"
table(actual, glm.preds)
```

```
##        glm.preds
## actual   No  Yes
##    No  4183  350
##    Yes  231   58
```

Logistic regression predicts yes a total of 408 times, 58 of those are true positives, giving it a rate of 14.22% which is less than that of boosting.