Bruce Campbell ST-617 Homework 4

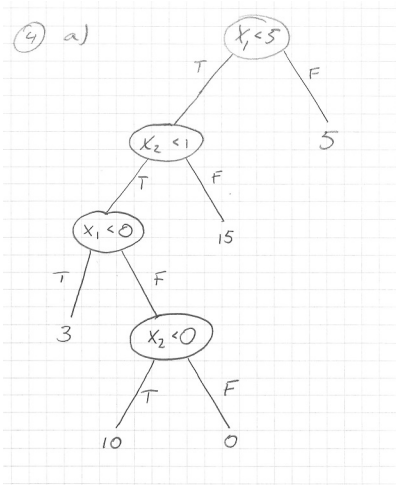
Wed Jul 20 21:19:43 2016

```
rm(list = ls())
set.seed(7)
```

Chapter 8

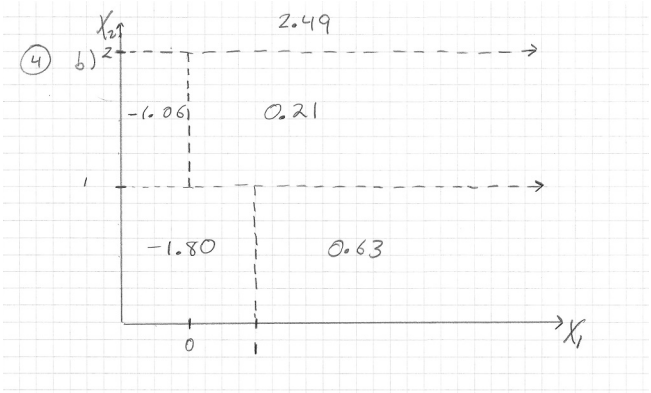
Problem 4

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.



Chapter 8

Problem 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 $P(ClassisRed|X)$

```
data <- c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)
library(pander)
pander(data)
```

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?

```
Red <- data > 0.5

sum_red <- sum(Red)

is_red_by_vote <- sum_red >= 5

mean_probability <- mean(data)

is_red_by_mean <- mean_probability > 0.5

results <- data.frame(method = c("voting", "mean"), is_red = c(is_red_by_vote,
    is_red_by_mean))

pander(results)
```

method	is_red
voting	TRUE
mean	FALSE

Chapter 8

Problem 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)
train = sample(nrow(Carseats), floor(nrow(Carseats) * 2/3))
DF <- Carseats
DFTrain <- DF[train, ]
DFTest <- DF[-train, ]
```

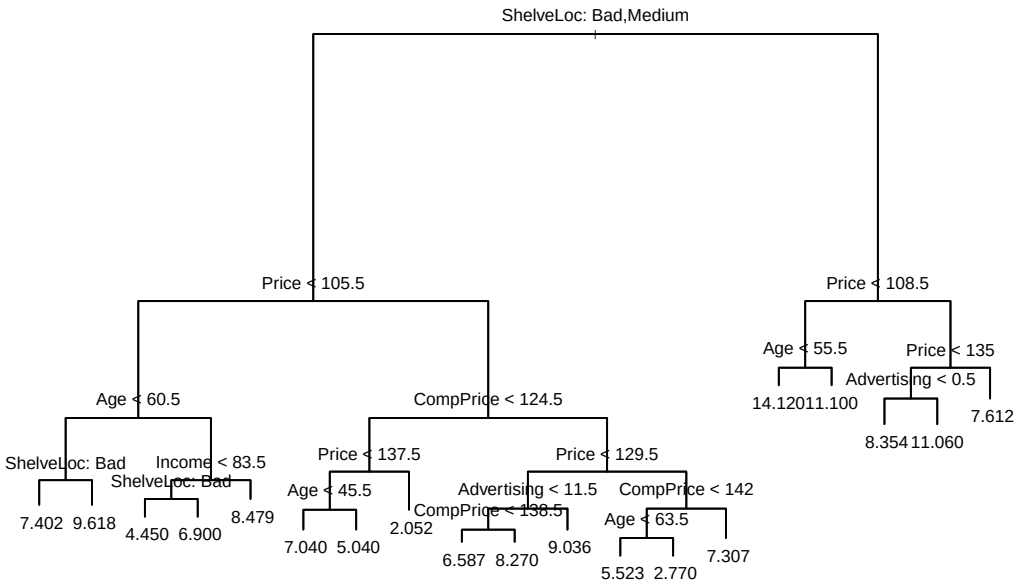
b)

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

```
library(tree)
tree.carseat = tree(Sales ~ ., DFTrain)

# Here we can set the minimum number of elements at a node.
control.settings <- tree.control(minsize = 30, nobs = nrow(DFTrain))
# tree.carseat=tree(Sales~.,DFTrain,control = control.settings)

summary(tree.carseat)
```

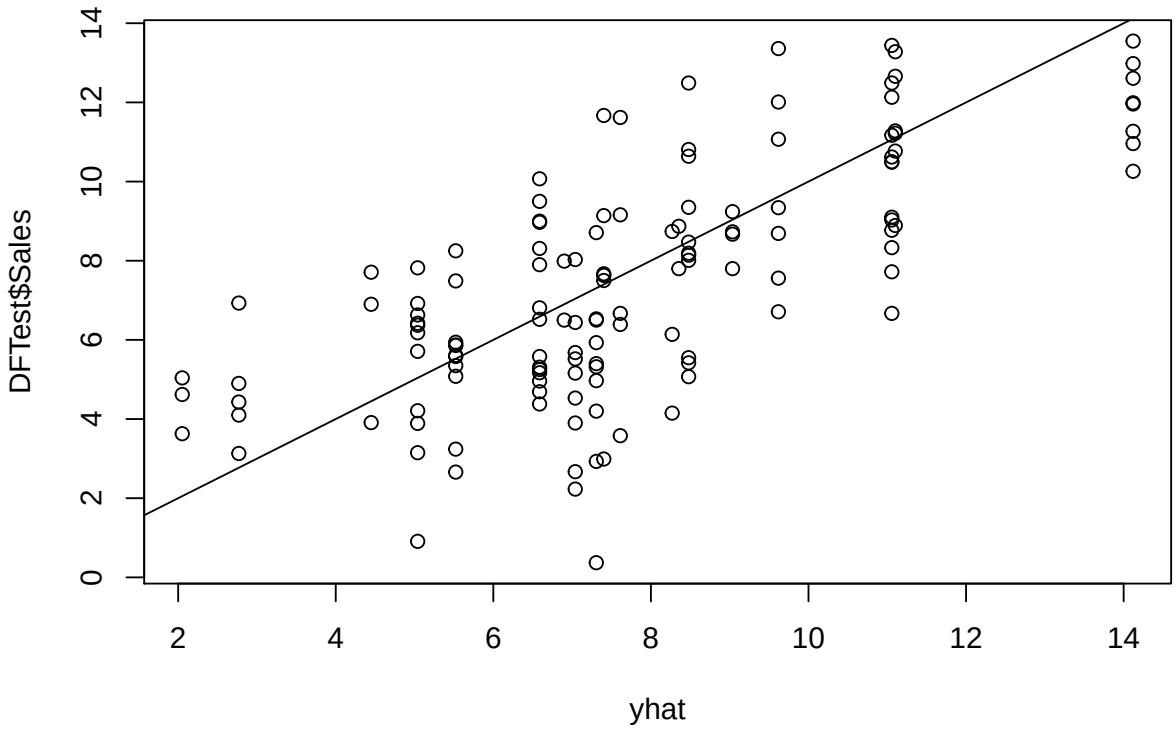
```
##
## Regression tree:
## tree(formula = Sales ~ ., data = DFTrain)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Income"      "CompPrice"
## [6] "Advertising"
## Number of terminal nodes:  19
## Residual mean deviance:  2.305 = 569.3 / 247
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.64800 -0.97940 -0.07851  0.00000  0.93590  4.57700
```

```
plot(tree.carseat, cex = 0.35)

text(tree.carseat, pretty = 0, cex = 0.6)
```



We see that ShelveLoc, Price and Age are the three most important variables. CompPrice is relevant for BadMedium shelf Loc. This is interesting as we expect those willing to consider all locations are those that would be comparing the prices.

```
yhat = predict(tree.carseat, newdata = DFTest)
plot(yhat, DFTest$Sales)
abline(0, 1)
```



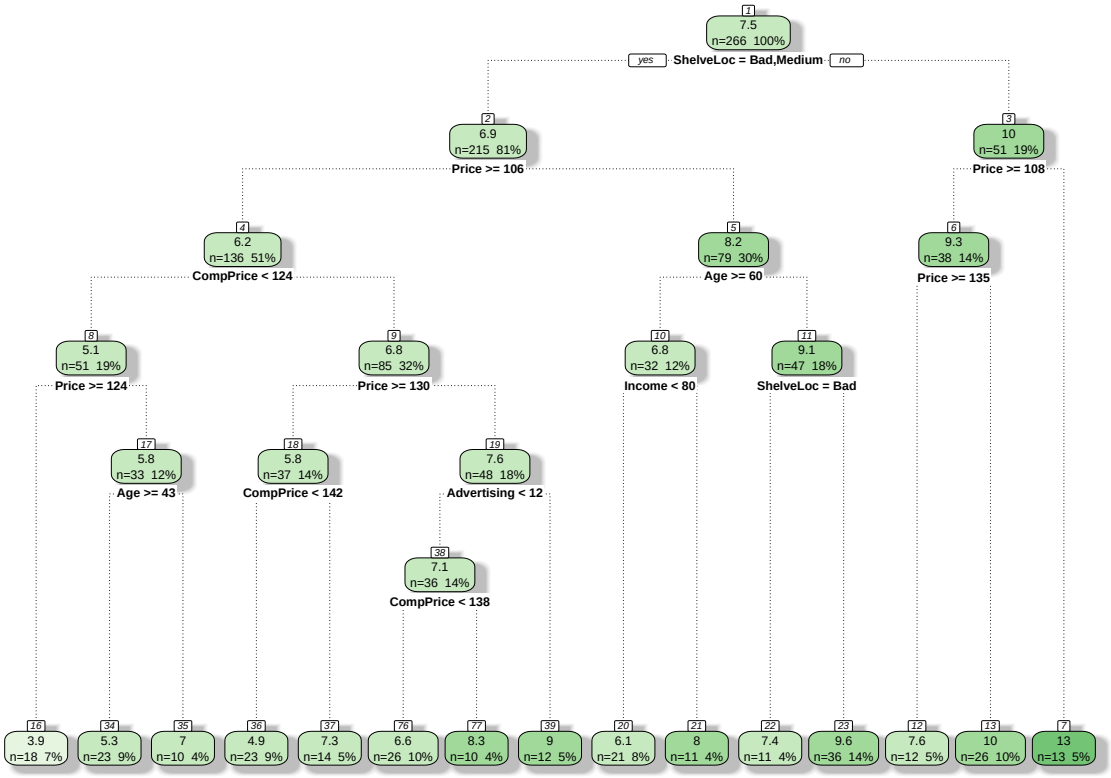

```
MSE <- mean((yhat - DFTest$Sales)^2)
library(pander)

RSS <- sum((yhat - DFTest$Sales)^2)
TSS <- sum((DFTest$Sales - mean(DFTest$Sales))^2)
RS2_Train <- 1 - (RSS/TSS)
```

The MSE of the training set is `4.9286046` and the R^2 of the training set is `0.4208888`

Here we take a quick look at the rpart package. The implementor of the the tree package reccomends rpart over tree.

https://stat.ethz.ch/pipermail/r-help/2005-May/070922.html

```
library(rpart)  # Popular decision tree algorithm
library(rattle)  # Fancy tree plot
library(rpart.plot)  # Enhanced tree plots
library(RColorBrewer)  # Color selection for fancy tree plot

control.settings <- rpart.control(minsplit = 30)
tree.rpart <- rpart(Sales ~ ., data = DFTrain, control = control.settings)

fancyRpartPlot(tree.rpart)
```

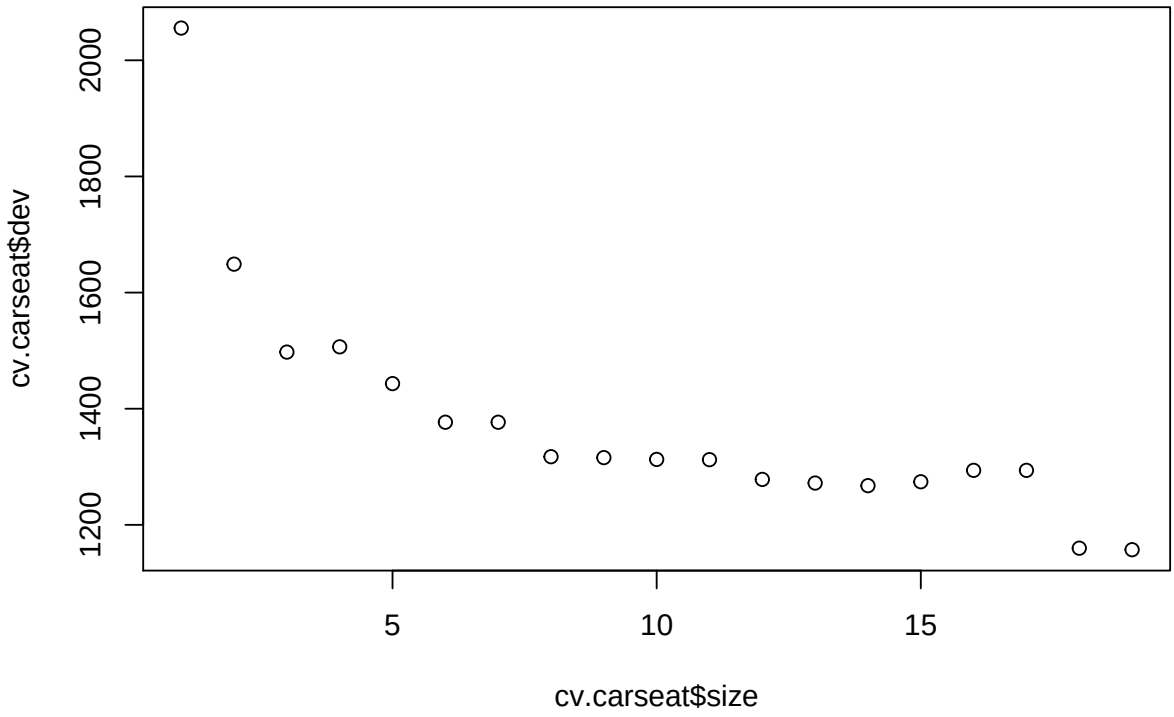


We see some differences but most of the same variables towards the top of the tree.

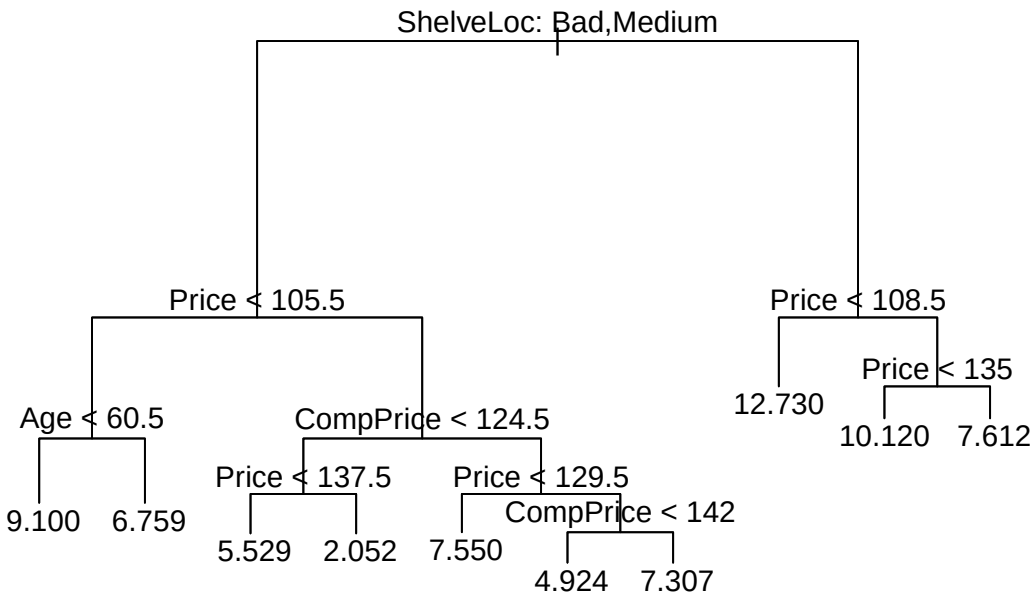
c)

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

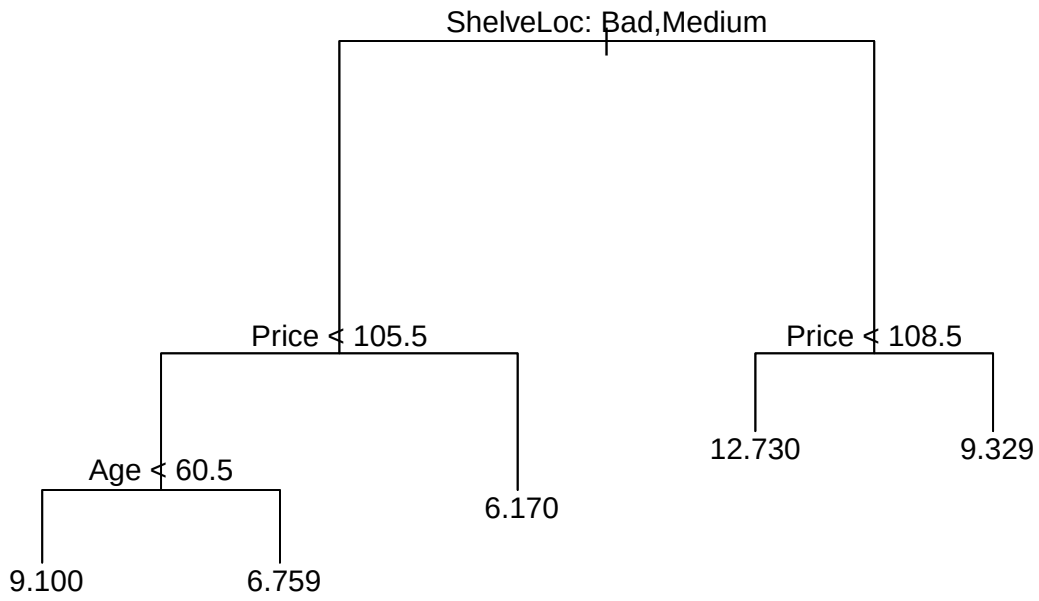
```
cv.carseat <- cv.tree(tree.carseat)
plot(cv.carseat$size, cv.carseat$dev)
```




```
prune.carseat = prune.tree(tree.carseat, best = 10)
plot(prune.carseat)
text(prune.carseat, pretty = 0)
```




```
yhat = predict(prune.carseat, newdata = DFTest)
MSE_pruned_10 <- mean((yhat - DFTest$Sales)^2)
library(pander)

RSS <- sum((yhat - DFTest$Sales)^2)
TSS <- sum((DFTrain$Sales - mean(DFTest$Sales))^2)
RS2_Train_pruned_10 <- 1 - (RSS/TSS)

prune.carseat = prune.tree(tree.carseat, best = 5)
plot(prune.carseat)
text(prune.carseat, pretty = 0)
```




```
yhat = predict(prune.carseat, newdata = DFTest)

MSE_pruned_5 <- mean((yhat - DFTest$Sales)^2)
library(pander)

RSS <- sum((yhat - DFTest$Sales)^2)
TSS <- sum((DFTrain$Sales - mean(DFTest$Sales))^2)
RS2_Train_pruned_5 <- 1 - (RSS/TSS)
```

The cross validation results suggest that we try pruning at 5 and 10 terminal nodes.

```
MSE_DF <- data.frame(model = c("MSE Full", "MSE Prune 5", "MSE Prune 10"), c(MSE,
    MSE_pruned_5, MSE_pruned_10))
pander(MSE_DF)
```

model	c.MSE..MSE_pruned_5..MSE_pruned_10.
MSE Full	4.929
MSE Prune 5	4.736
MSE Prune 10	4.611

```
RSQ_DF <- data.frame(model = c("RSQ Full", "RSQ Prune 5", "RSQ Prune 10"), c(RS2_Train,
    RS2_Train_pruned_5, RS2_Train_pruned_10))
```

```
pander(RSQ_DF)
```

model	c.RS2_Train..RS2_Train_pruned_5..RS2_Train_pruned_10.
RSQ Full	0.4209
RSQ Prune 5	0.6895
RSQ Prune 10	0.6977

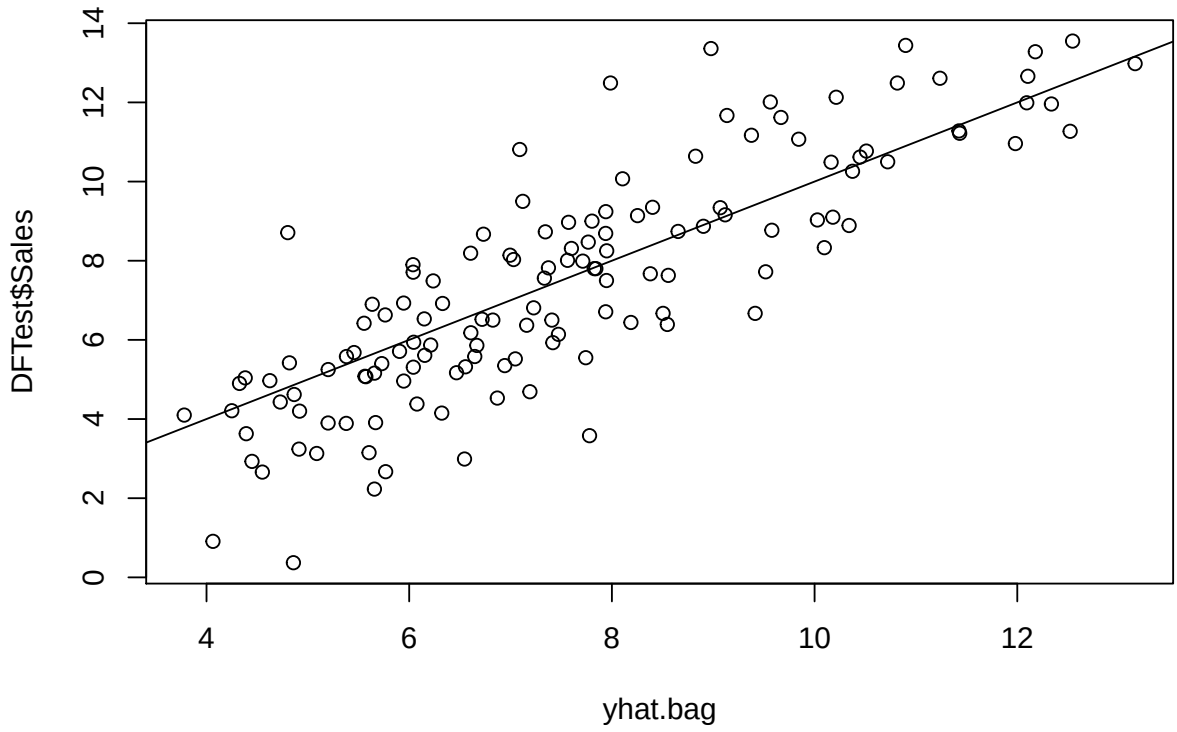
d)

Use the bagging approach in order to analyze this data. What test error rate do you obtain? Use the importance() function to determine which variables are most important

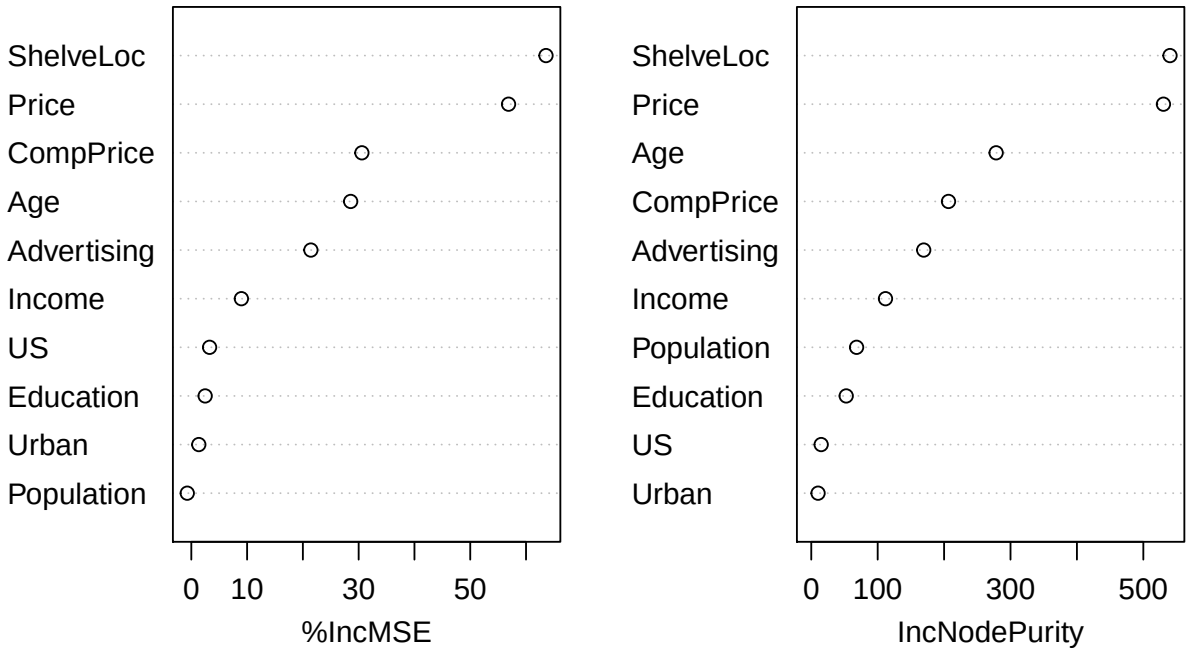
```
library(randomForest)
bag.carseat = randomForest(Sales ~ ., data = DFTrain, mtry = 11, importance = TRUE)
bag.carseat
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = DFTrain, mtry = 11,      importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 2.499298
##                     % Var explained: 67.43
```

```
yhat.bag = predict(bag.carseat, newdata = DFTest)
plot(yhat.bag, DFTest$Sales)
abline(0, 1)
```




```
MSE_Bagging <- mean((yhat.bag - DFTest$Sales)^2)


varImpPlot(bag.carseat)
```



The MSE for bagged model is `2.5398888` and is greatly reduced from that of the single regression tree MSE of `4.9286046`. We also note that the most important variables are ShelveLoc, Price, Age, CompPrice, and Advertising

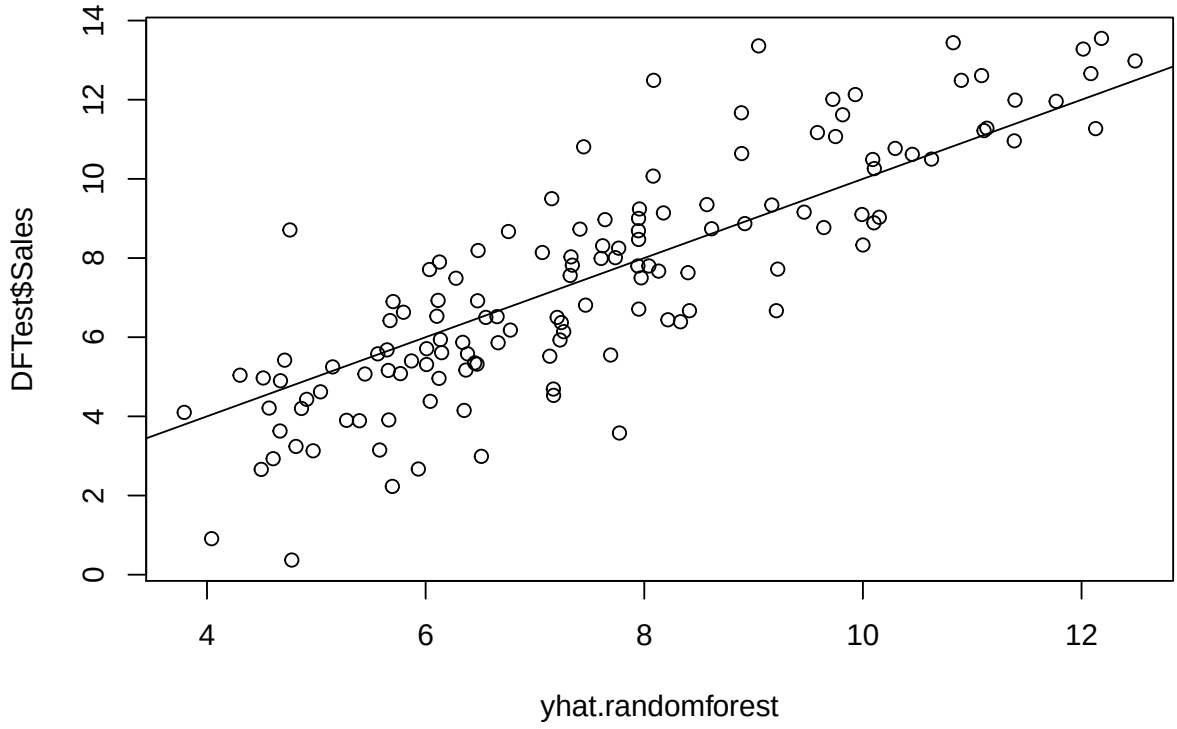
e)

Use random forests to analyze this data. What test error rate 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.

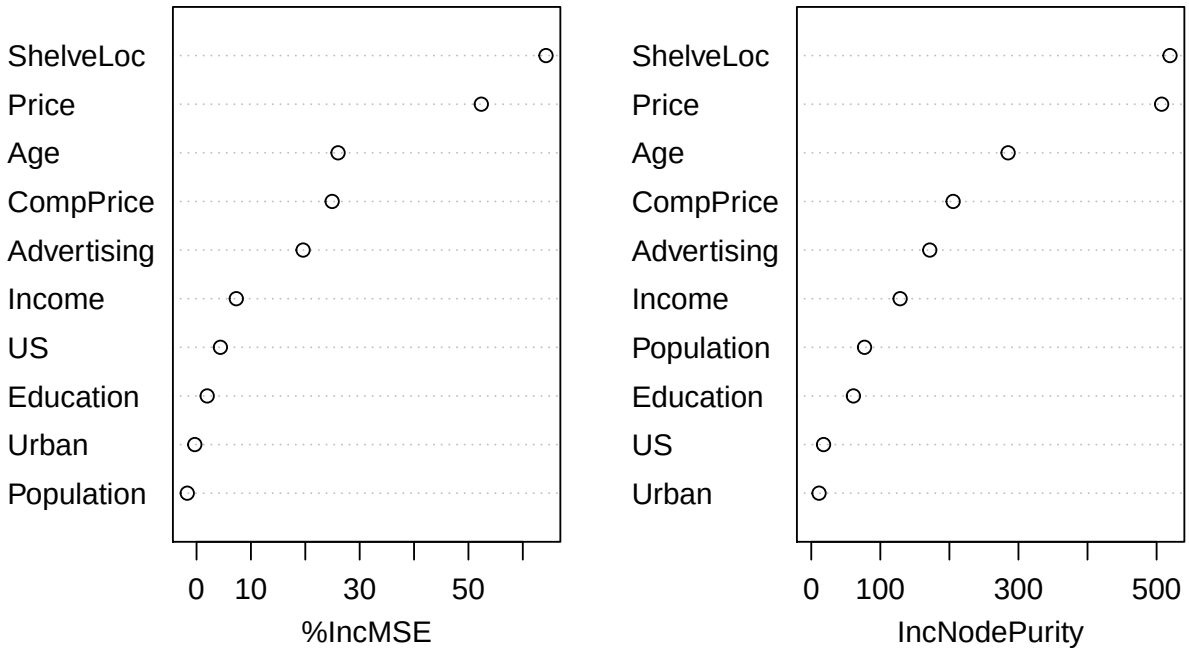
```
randomforest.carseat = randomForest(Sales ~ ., data = DFTrain, mtry = 7, importance = TRUE)
randomforest.carseat
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = DFTrain, mtry = 7, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##           Mean of squared residuals: 2.554792
##                     % Var explained: 66.7
```

```r
yhat.randomforest = predict(randomforest.carseat, newdata = DFTest)
plot(yhat.randomforest, DFTest$Sales)
abline(0, 1)
```

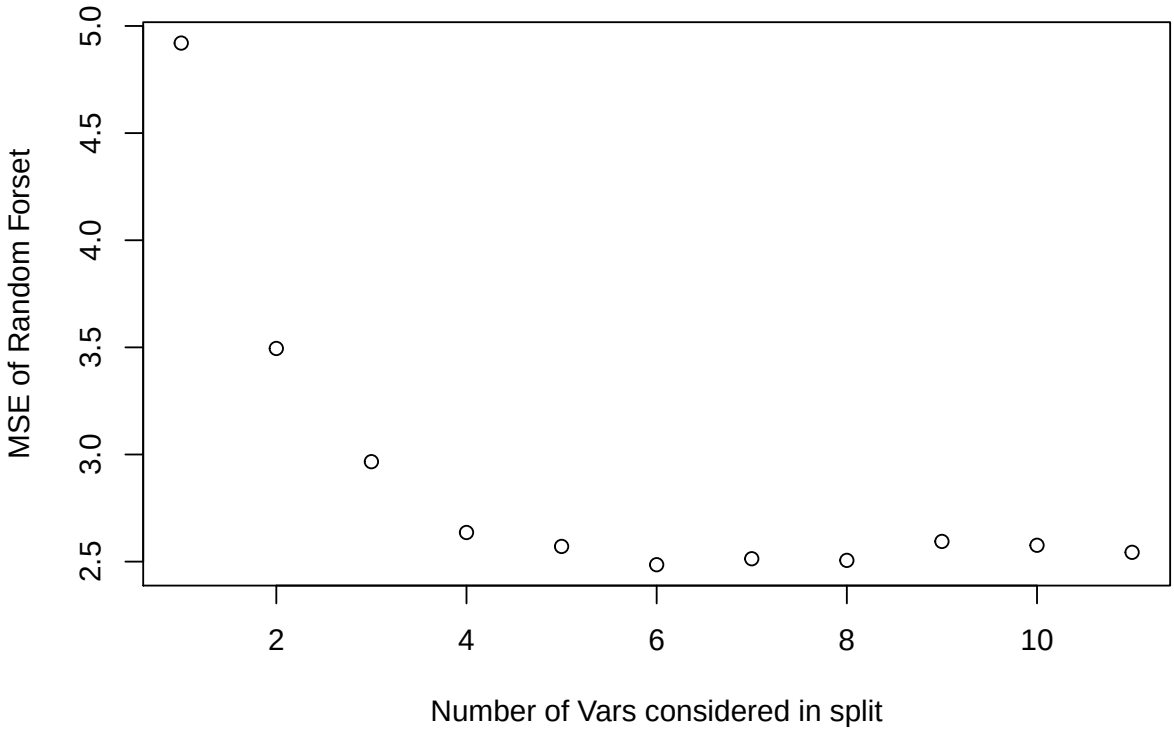



```r
MSE_RandomForest <- mean((yhat.randomforest - DFTest$Sales)^2)

varImpPlot(randomforest.carseat)
```



As expected the random forest MSE (`2.4646252`) is lower than the bagging MSE (`2.5398888`)

```
mse_vev <- matrix(data = NA, nrow = 11, ncol = 1)
for (i in 1:11) {
    randomforest.carseat = randomForest(Sales ~ ., data = DFTrain, mtry = i,
        importance = TRUE)
    yhat.randomforest = predict(randomforest.carseat, newdata = DFTest)
    mse_vev[i] <- mean((yhat.randomforest - DFTest$Sales)^2)
}
plot(1:11, mse_vev, xlab = "Number of Vars considered in split", ylab = "MSE of Random Forset")
```




```
which.min(mse_vev)
```

```
## [1] 6
```

The bagging importance plot has served us well. We chose an m of 7 based on that plot, and the plot above of the MSE by m confirms that 7 was the best choice. As expected we see the MSE goes down rapidly as we add variables, and then levels off.

Chapter 8

Problem 9

This problem involves the OJ data set which is part of the ISLR package.

a)

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

```
library(ISLR)
attach(OJ)
```

```
train = sample(nrow(OJ), 800)
DF <- OJ
DFTrain <- DF[train, ]
DFTest <- DF[-train, ]
```

b)

Fit a tree to the training data, with Purchase as the response and the other variables except for Buy as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

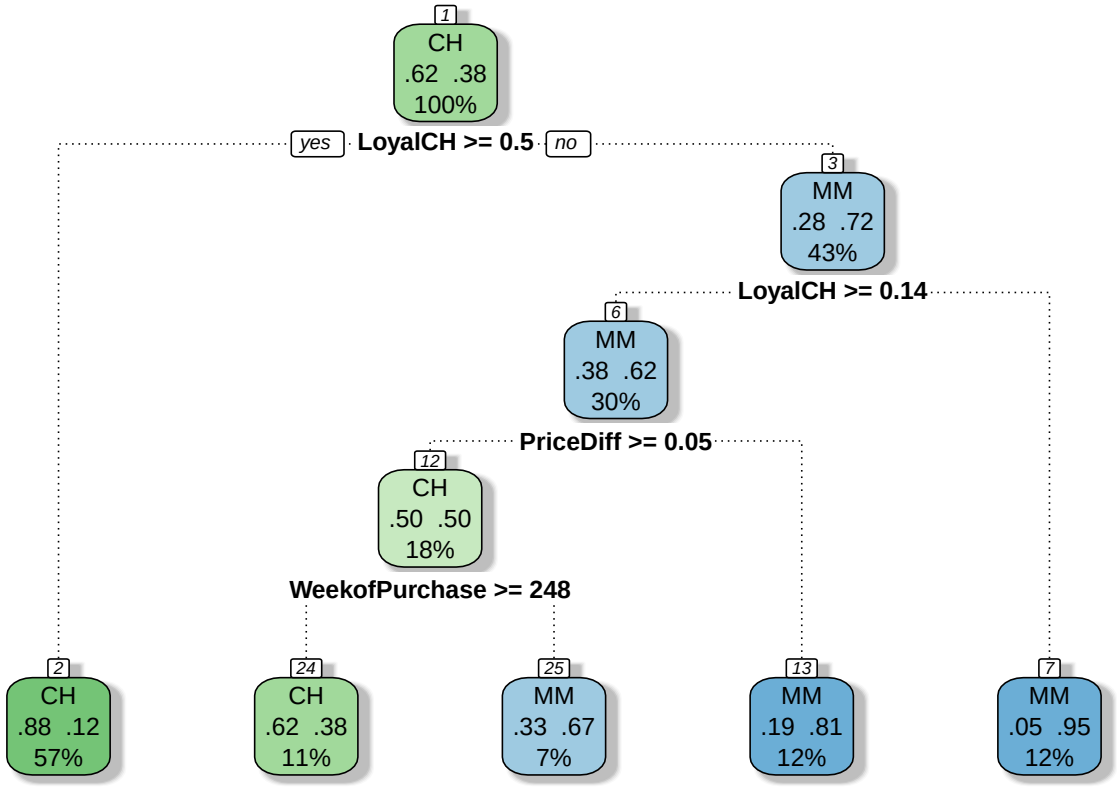
```
names(DFTrain)
```

```
##  [1] "Purchase"       "WeekofPurchase" "StoreID"        "PriceCH"
##  [5] "PriceMM"        "DiscCH"         "DiscMM"         "SpecialCH"
##  [9] "SpecialMM"      "LoyalCH"        "SalePriceMM"    "SalePriceCH"
## [13] "PriceDiff"      "Store7"         "PctDiscMM"      "PctDiscCH"
## [17] "ListPriceDiff"  "STORE"
```

```
library(tree)

control.settings <- tree.control(minsize = 30, nobs = nrow(DFTrain))
tree.oj = tree(Purchase ~ ., DFTrain, control = control.settings, split = "deviance")

summary(tree.oj)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = DFTrain, control = control.settings,
##     split = "deviance")
## Variables actually used in tree construction:
## [1] "LoyalCH"        "PriceDiff"      "WeekofPurchase" "SalePriceMM"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7334 = 580.9 / 792
## Misclassification error rate: 0.155 = 124 / 800
```

```
## We notice that few of the variables are included in the tree despite
## changing the control settings.  We check with rpart to make sure this is
## the case.  rpart does respond to changes in the control setting.
library(rpart)  # Popular decision tree algorithm
library(rattle)  # Fancy tree plot
library(rpart.plot)  # Enhanced tree plots
library(RColorBrewer)  # Color selection for fancy tree plot

control.settings <- rpart.control(minsplit = 100)
tree.rpart <- rpart(Purchase ~ ., data = DFTrain, control = control.settings)

fancyRpartPlot(tree.rpart)
```



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

```
tree.oj
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1062.00 CH ( 0.62125 0.37875 )
##    2) LoyalCH < 0.50395 343  408.60 MM ( 0.28280 0.71720 )
##      4) LoyalCH < 0.142213 100   39.70 MM ( 0.05000 0.95000 ) *
##      5) LoyalCH > 0.142213 243  322.40 MM ( 0.37860 0.62140 )
##       10) PriceDiff < 0.05 98   96.39 MM ( 0.19388 0.80612 ) *
##       11) PriceDiff > 0.05 145  201.00 CH ( 0.50345 0.49655 )
##         22) WeekofPurchase < 247.5 58   73.36 MM ( 0.32759 0.67241 ) *
##         23) WeekofPurchase > 247.5 87  115.50 CH ( 0.62069 0.37931 ) *
##    3) LoyalCH > 0.50395 457  343.90 CH ( 0.87527 0.12473 )
##      6) LoyalCH < 0.764572 195  215.40 CH ( 0.75897 0.24103 )
##       12) SalePriceMM < 2.125 122  159.50 CH ( 0.63934 0.36066 )
##         24) PriceDiff < -0.35 17   18.55 MM ( 0.23529 0.76471 ) *
##         25) PriceDiff > -0.35 105  127.40 CH ( 0.70476 0.29524 ) *
##       13) SalePriceMM > 2.125 73   25.03 CH ( 0.95890 0.04110 ) *
##      7) LoyalCH > 0.764572 262   84.93 CH ( 0.96183 0.03817 ) *
```

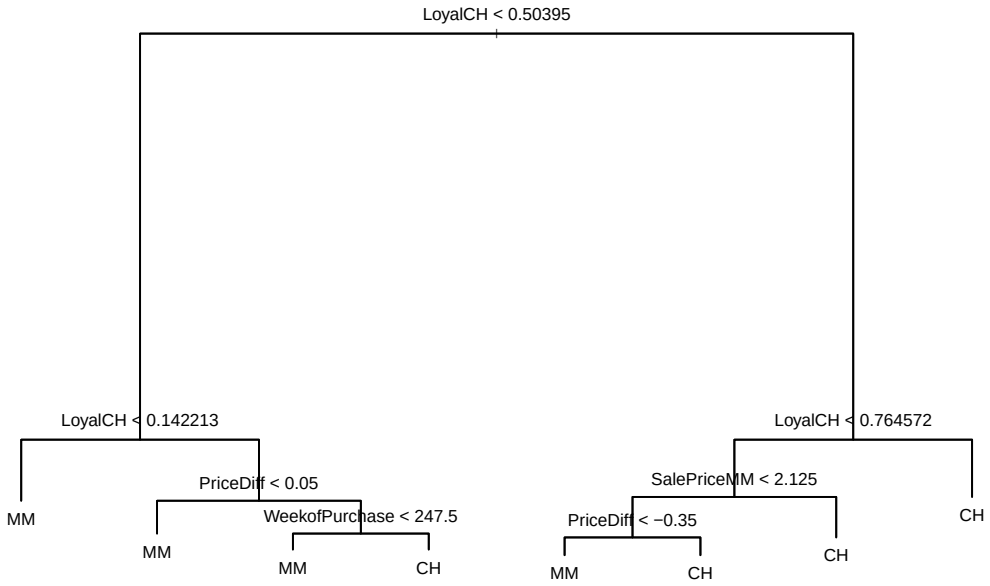
LoyalCH $>$ 0.482389 LoyalCH $<$ 0.764572 PriceDiff $<$ 0.265 PriceDiff $>$ -0.165

is an interesting node. It tells us that there may be a price difference at which a loyal customer my switch brands.

d)

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

```
plot(tree.oj, cex = 0.35)

text(tree.oj, pretty = 0, cex = 0.6)
```



We see quite clearly that brand loyalty is the dominant variable in predicting which brand is purchased.

e)

Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```
oj.probs = predict(tree.oj, newdata = DFTest)
# oj.probs is a matrix with names columns the first being probability of CH
# the second probability of MM
oj.pred = rep("CH ", nrow(DFTest))
oj.pred[oj.probs[, 1] > 0.5] = " CH"
```

```
TB <- table(oj.pred, DFTest$Purchase)
library(pander)
pander(TB)
```

	CH	MM
CH	137	34
CH	19	80

```
ACC_Tree = (TB[1] + TB[4])/length(DFTest$Purchase)
```

The accuracy of the tree classifier is `0.8037037`

f)

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

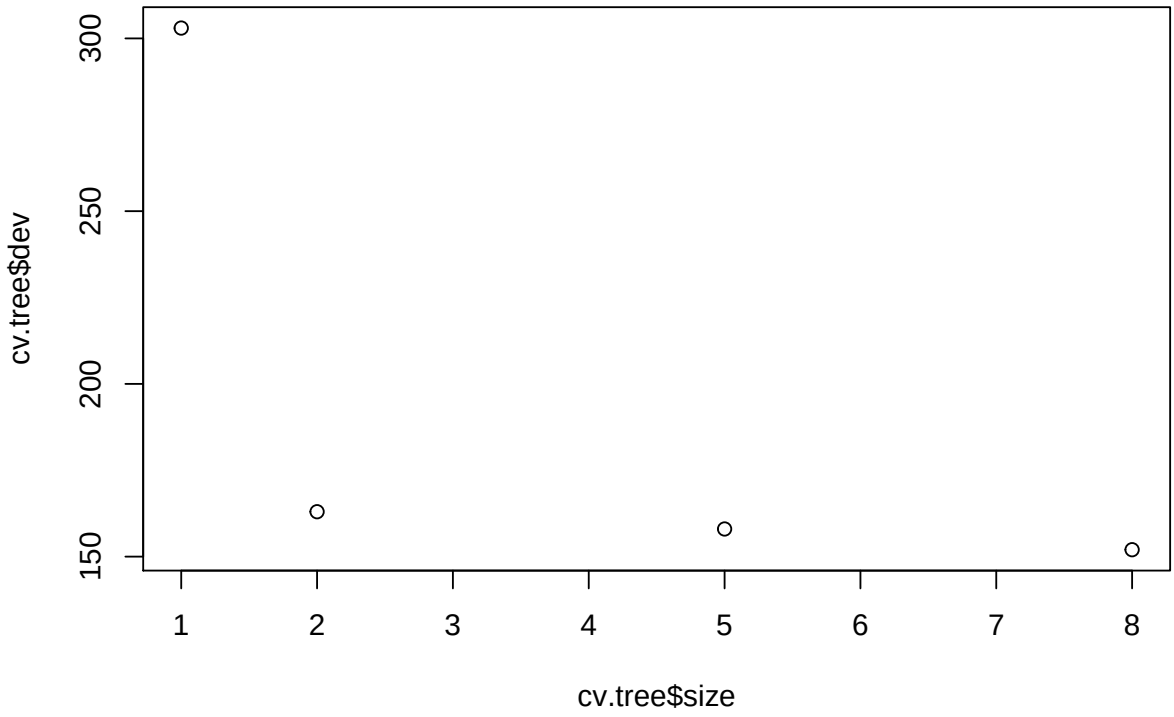
```
cv.tree <- cv.tree(tree.oj, FUN = prune.misclass)
cv.tree
```

```
## $size
## [1] 8 5 2 1
##
## $dev
## [1] 152 158 163 303
##
## $k
## [1] -Inf    3    7  149
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

g)

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

```
plot(cv.tree$size, cv.tree$dev)
```




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

h)

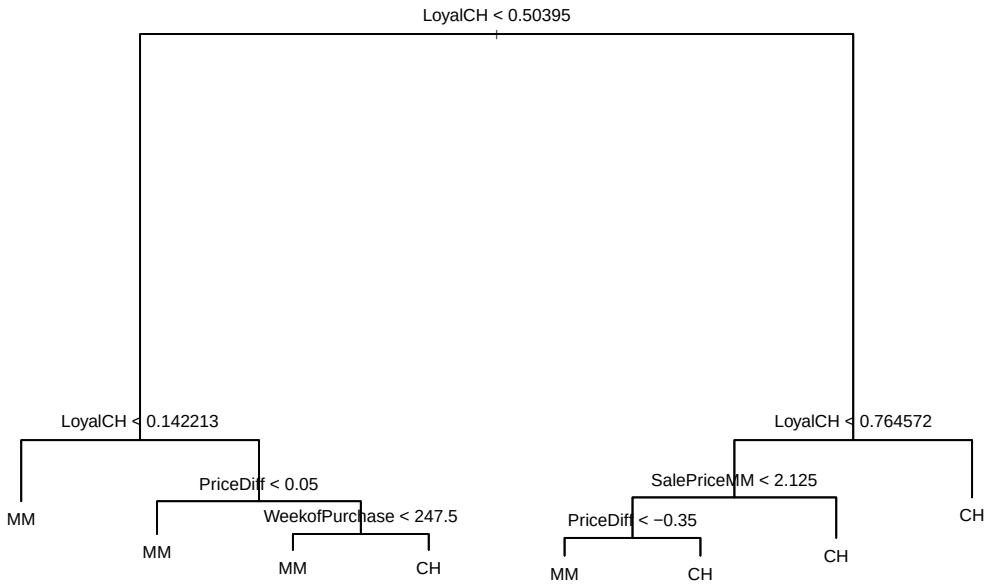
Which tree size corresponds to the lowest cross-validated classification error rate?

The optimal tree size based on the cross validation error rate is `8`

i)

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

```
prune.oj <- prune.misclass(tree.oj, best = optimal_tree_size)

plot(prune.oj, cex = 0.35)
text(prune.oj, pretty = 0, cex = 0.6)
```



j)

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

```
summ_tree.oj <- summary(tree.oj)
train_err_tree.oj <- summ_tree.oj$misclass[1]/summ_tree.oj$misclass[2]

summ_prune.oj <- summary(prune.oj)
train_err_prune.oj <- summ_prune.oj$misclass[1]/summ_prune.oj$misclass[2]

pander(data.frame(tree = c("tree.oj", "prune.tree"), training_error = c(train_err_tree.oj,
    train_err_prune.oj)))
```

tree	training_error
tree.oj	0.155
prune.tree	0.155

We have the same error.

k)

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

```
oj.probs = predict(prune.oj, newdata = DFTest)
# oj.probs is a matrix with names columns the first being probability of CH
# the second probability of MM
oj.pred = rep("CH ", nrow(DFTest))
oj.pred[oj.probs[, 1] > 0.5] = " CH"
TB <- table(oj.pred, DFTest$Purchase)
library(pander)

ACC_Prune = (TB[1] + TB[4])/length(DFTest$Purchase)

pander(data.frame(tree = c("tree.oj", "prune.tree"), test_error = c(ACC_Tree,
    ACC_Prune)))
```

tree	test_error
tree.oj	0.8037
prune.tree	0.8037

We have the same error.

This is suspicious. As noted above the tree call is returning the same tree regardless of the control setting. It is notable that this tree size is close to the same as that suggested by the cross validation routine. We'd like to debug and understand this further.