Annand Module 07 Lab 01

Joseph Annand

2023-12-10

```r
library(ISLR2)
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```r
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.3.2
```

Question 8

Part A

```r
# Split data into training and test data
set.seed(1)
train <- sample(1:nrow(Carseats), nrow(Carseats) / 2)
sales.test <- Carseats[-train, "Sales"]
```

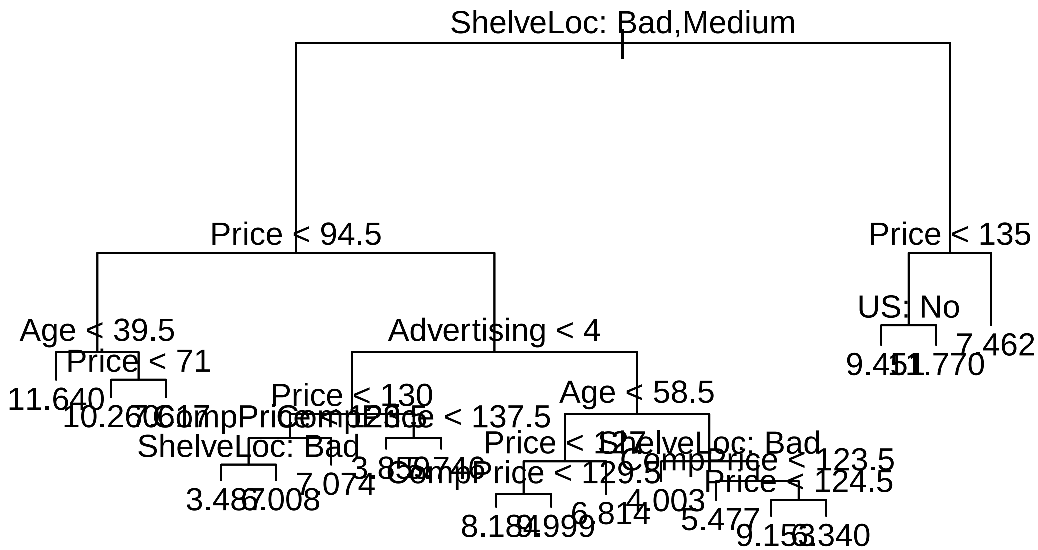
Part B

```r
tree.sales <- tree(Sales ~ ., data = Carseats, subset = train)
summary(tree.sales)
```

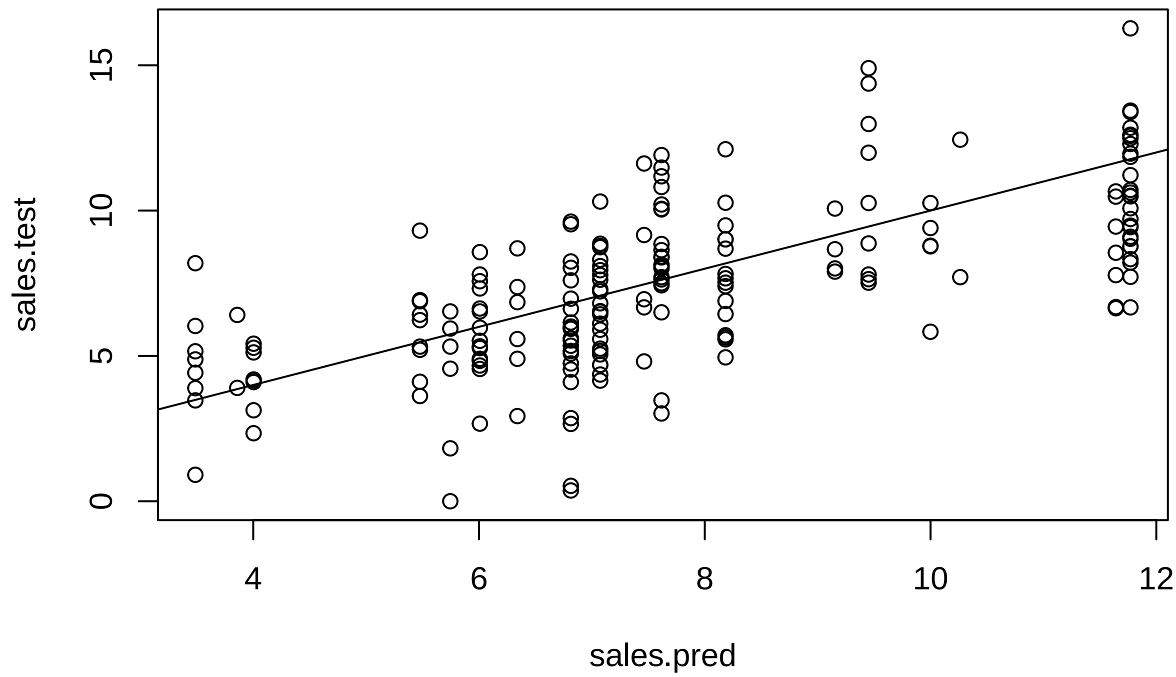
```
##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats, subset = train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Advertising" "CompPrice"
## [6] "US"
## Number of terminal nodes:  18
## Residual mean deviance:  2.167 = 394.3 / 182
```

```
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.88200 -0.88200 -0.08712  0.00000  0.89590  4.09900
```

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




```r
sales.pred <- predict(tree.sales, newdata = Carseats[-train, ])
plot(sales.pred, sales.test)
abline(0, 1)
```

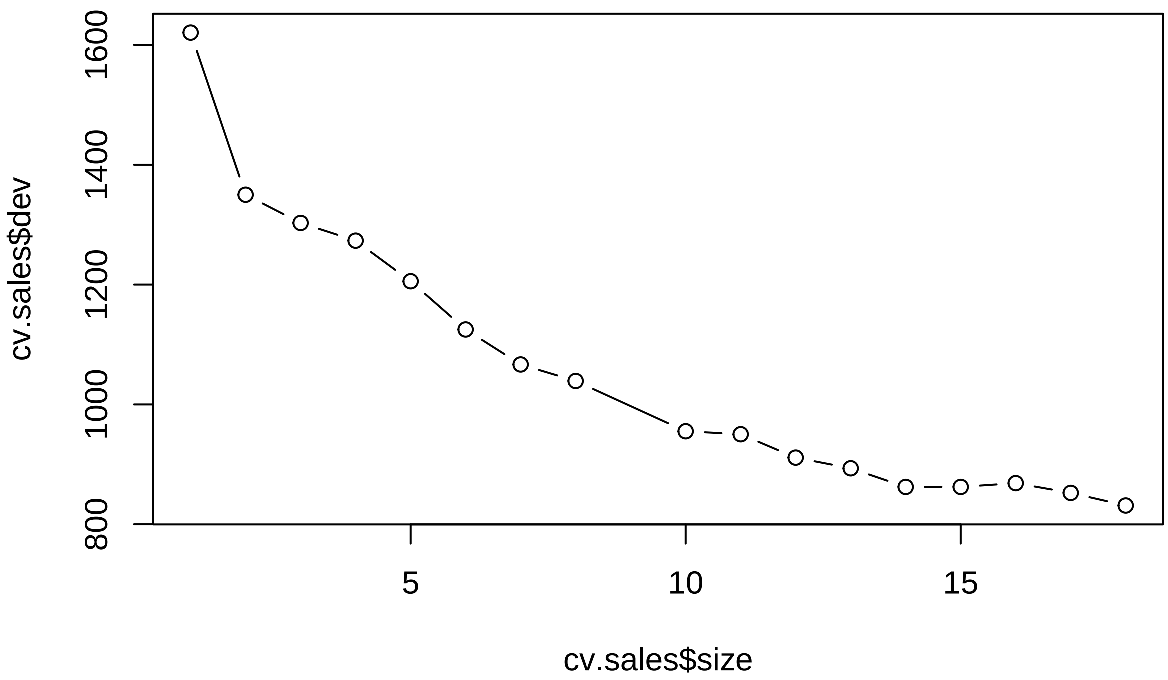



```r
mean((sales.pred - sales.test)^2)
```

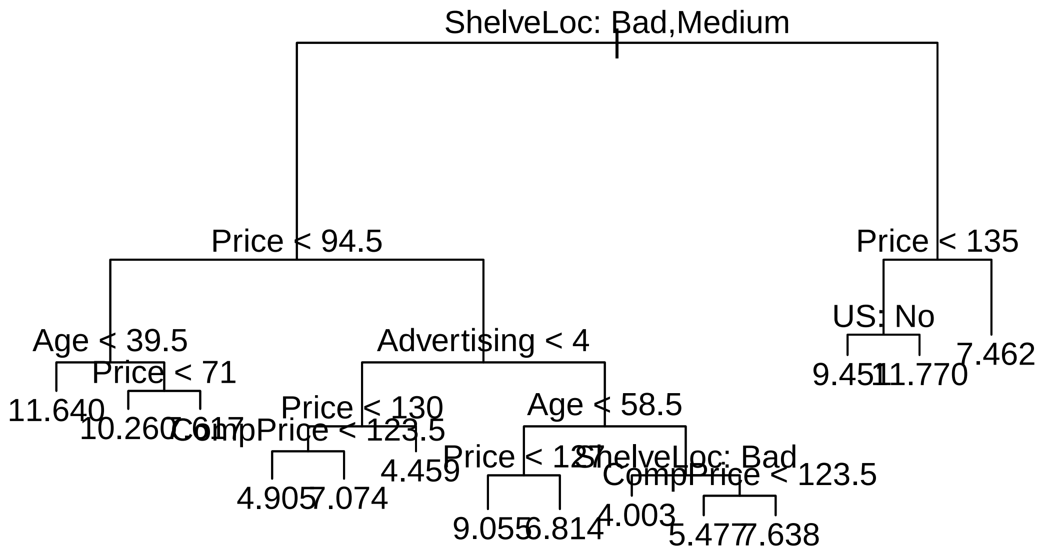
```
## [1] 4.922039
```

Part C

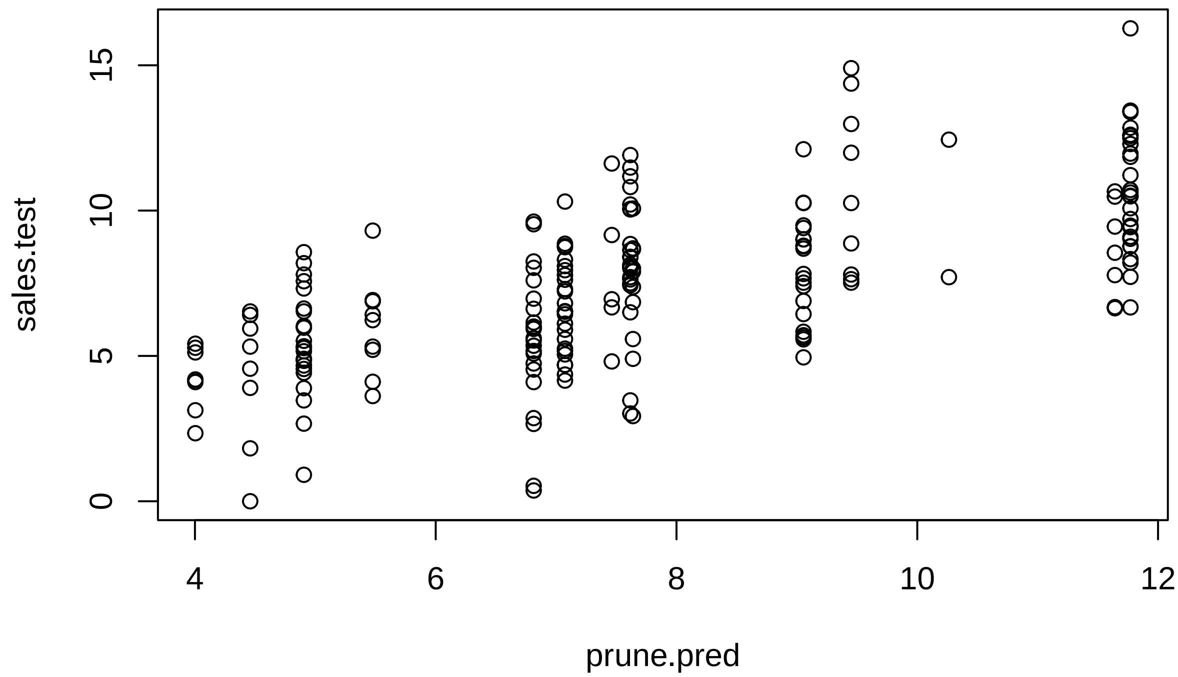
```r
cv.sales <- cv.tree(tree.sales)
plot(cv.sales$size, cv.sales$dev, type = "b")
```




```r
prune.sales <- prune.tree(tree.sales, best = 14)
plot(prune.sales)
text(prune.sales, pretty = 0)
```




```r
prune.pred <- predict(prune.sales, newdata = Carseats[-train, ])
plot(prune.pred, sales.test)
```




```r
mean((prune.pred - sales.test)^2)
```

```
## [1] 5.013738
```

Pruning the data set resulted in a higher MSE than when it was unpruned.

Part D

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.2
```

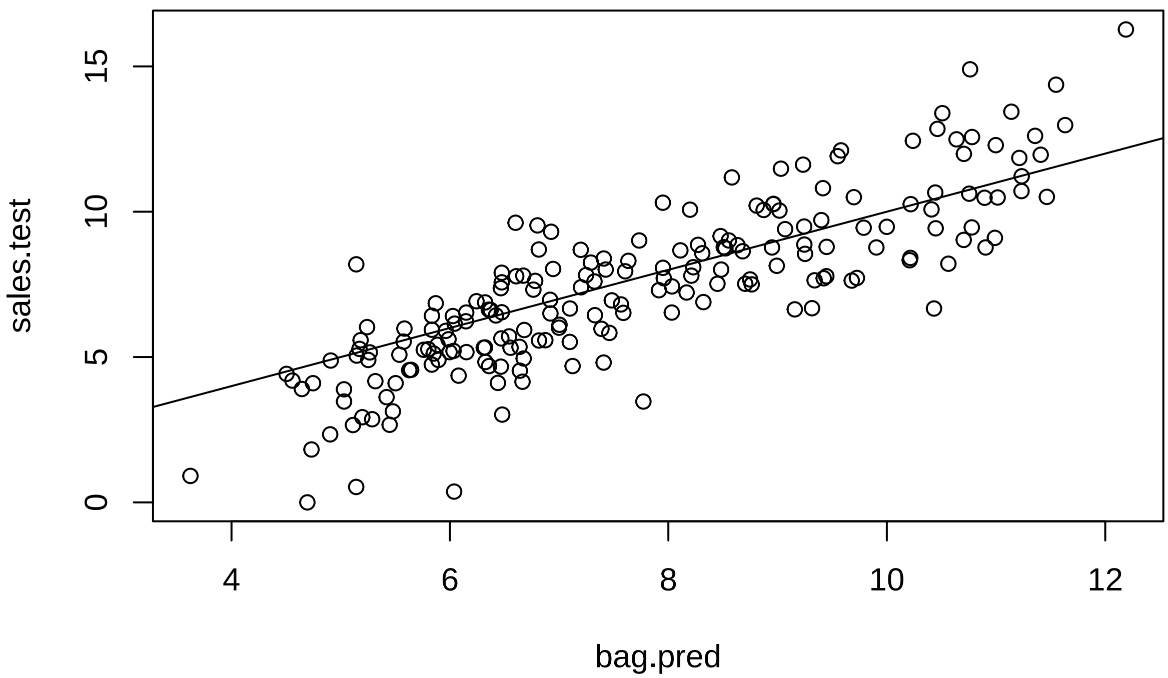
```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
set.seed(2)
bag.sales <- randomForest(Sales ~ ., data = Carseats, subset = train,
                          mtry = (ncol(Carseats) - 1), importance = T)
bag.sales
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = Carseats, mtry = (ncol(Carseats) -      1), importance = T
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 2.908014
##                     % Var explained: 63.02
```

```
bag.pred <- predict(bag.sales, newdata = Carseats[-train, ])
plot(bag.pred, sales.test)
abline(0, 1)
```




```
mean((bag.pred - sales.test)^2)
```

```
## [1] 2.586535
```

Part E

```
set.seed(2)
# Create random forest model using default m = p/3
rf.sales <- randomForest(Sales ~ ., data = Carseats, subset = train,
```

```r
                         importance = T)
rf.pred <- predict(rf.sales, newdata = Carseats[-train, ])
mean((rf.pred - sales.test)^2)
```

```
## [1] 2.972319
```

```r
set.seed(2)
# Create random forest model using default m = 6
rf.sales6 <- randomForest(Sales ~ ., data = Carseats, subset = train,
                         mtry = 6, importance = T)
rf.pred6 <- predict(rf.sales6, newdata = Carseats[-train, ])
mean((rf.pred6 - sales.test)^2)
```

```
## [1] 2.63335
```

```r
set.seed(2)
# Create random forest model using default m = 9
rf.sales9 <- randomForest(Sales ~ ., data = Carseats, subset = train,
                         mtry = 9, importance = T)
rf.pred9 <- predict(rf.sales9, newdata = Carseats[-train, ])
mean((rf.pred9 - sales.test)^2)
```

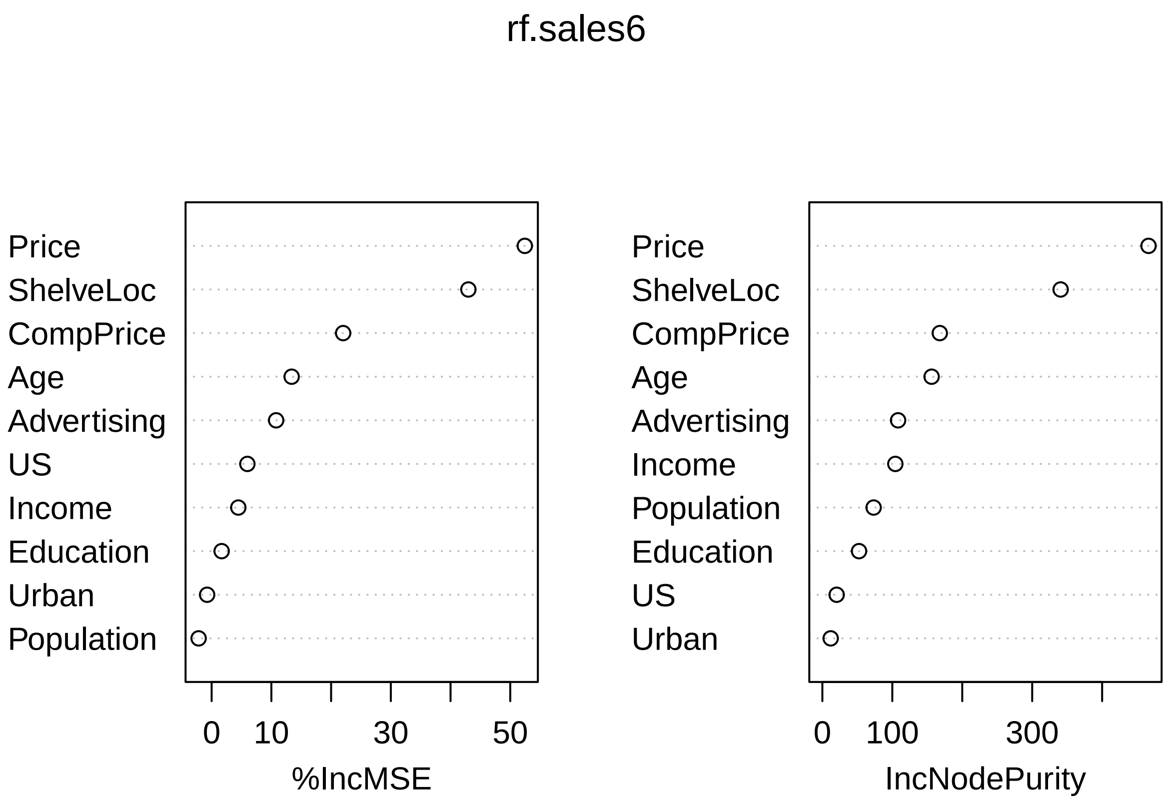
```
## [1] 2.637777
```

Setting m, the number of variables considered at each split,

```r
importance(rf.sales6)
```

```
##                %IncMSE IncNodePurity
## CompPrice   22.0178739     167.89030
## Income       4.4666039     104.36139
## Advertising 10.8014745     108.29346
## Population  -2.1801356      73.31029
## Price       52.4368458     466.35937
## ShelveLoc   42.9852979     340.65312
## Age         13.3967721     156.21255
## Education    1.6788660      52.43254
## Urban       -0.7559588      11.90340
## US           5.9833908      20.46648
```

```r
varImpPlot(rf.sales6)
```



Across all the trees considered, Price and ShelveLoc are the two most important variables.

Part F

```
# Analyze data using BART
library(BART)
```

```
## Warning: package 'BART' was built under R version 4.3.2

## Loading required package: nlme

## Loading required package: nnet

## Loading required package: survival
```

```
x <- Carseats[, 2:11]
y <- Carseats[, "Sales"]
xtrain <- x[train, ]
ytrain <- y[train]
xtest <- x[-train, ]
ytest <- y[-train]

# Run BART with default settings
set.seed(1)
bartfit <- gbart(xtrain, ytrain, x.test = xtest)
```

```
## *****Calling gbart: type=1
## *****Data:
## data:n,p,np: 200, 14, 200
## y1,yn: 2.781850, 1.091850
## x1,x[n*p]: 107.000000, 1.000000
## xp1,xp[np*p]: 111.000000, 1.000000
## *****Number of Trees: 200
## *****Number of Cut Points: 63 ... 1
## *****burn,nd,thin: 100,1000,1
## *****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.273474,3,0.23074,7.57815
## *****sigma: 1.088371
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,14,0
## *****printevery: 100
##
## MCMC
## done 0 (out of 1100)
## done 100 (out of 1100)
## done 200 (out of 1100)
## done 300 (out of 1100)
## done 400 (out of 1100)
## done 500 (out of 1100)
## done 600 (out of 1100)
## done 700 (out of 1100)
## done 800 (out of 1100)
## done 900 (out of 1100)
## done 1000 (out of 1100)
## time: 3s
## trcnt,tecnt: 1000,1000
```

```r
bart.pred <- bartfit$yhat.test.mean
mean((ytest - bart.pred)^2)
```

```
## [1] 1.450842
```

BART yields a significantly lower test error than bagging and random forests.

Question 9

Part A

```r
# Divide OJ data set into training and test data
set.seed(3)
train.oj <- sample(1:nrow(OJ), 800)
test.oj <- OJ[-train.oj, ]
```

Part B

```r
# Fit a classification tree to OJ data with Purchase as the response
tree.oj <- tree(Purchase ~ ., data = OJ, subset = train.oj)
summary(tree.oj)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ, subset = train.oj)
## Variables actually used in tree construction:
## [1] "LoyalCH"     "PriceDiff"   "PriceMM"     "SalePriceMM"
## Number of terminal nodes:  9
## Residual mean deviance:  0.7247 = 573.2 / 791
## Misclassification error rate: 0.1812 = 145 / 800
```

The variables used in the classification tree construction are LoyalCH, PriceDiff, PriceMM, and SalePriceMM. There are 9 terminal nodes, and the training error rate is 18.12%.

Part C

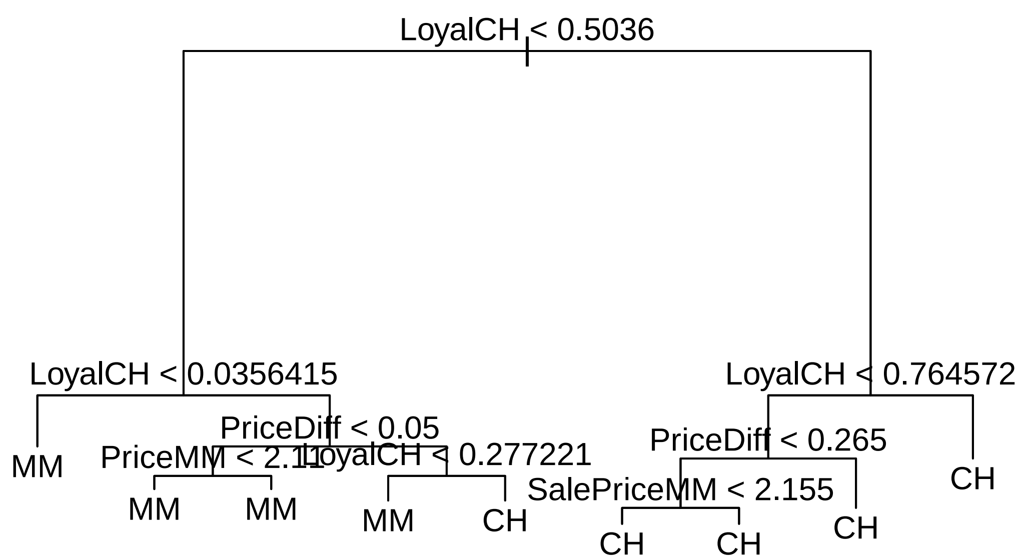
```r
tree.oj
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1068.00 CH ( 0.61250 0.38750 )
##    2) LoyalCH < 0.5036 346  414.30 MM ( 0.28613 0.71387 )
##      4) LoyalCH < 0.0356415 57    0.00 MM ( 0.00000 1.00000 ) *
##      5) LoyalCH > 0.0356415 289  371.50 MM ( 0.34256 0.65744 )
##       10) PriceDiff < 0.05 114  105.90 MM ( 0.17544 0.82456 )
##         20) PriceMM < 2.11 89   94.84 MM ( 0.22472 0.77528 ) *
##         21) PriceMM > 2.11 25    0.00 MM ( 0.00000 1.00000 ) *
##       11) PriceDiff > 0.05 175  240.90 MM ( 0.45143 0.54857 )
##         22) LoyalCH < 0.277221 62   66.24 MM ( 0.22581 0.77419 ) *
##         23) LoyalCH > 0.277221 113  154.10 CH ( 0.57522 0.42478 ) *
##    3) LoyalCH > 0.5036 454  365.70 CH ( 0.86123 0.13877 )
##      6) LoyalCH < 0.764572 187  221.10 CH ( 0.72193 0.27807 )
##       12) PriceDiff < 0.265 113  154.70 CH ( 0.56637 0.43363 )
##         24) SalePriceMM < 2.155 102  141.20 CH ( 0.51961 0.48039 ) *
##         25) SalePriceMM > 2.155 11    0.00 CH ( 1.00000 0.00000 ) *
##       13) PriceDiff > 0.265 74   25.11 CH ( 0.95946 0.04054 ) *
##      7) LoyalCH > 0.764572 267   91.71 CH ( 0.95880 0.04120 ) *
```

Taking a look at node 24, the classification process can be described as followed: if LoyalCH is greater than 0.5036 and less than 0.764572, and if PriceDiff is less than 0.265, and if SalePriceMM is less than 2.155, then the response is classified as CH.

Part D

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



The decision tree shows that generally when customer brand loyalty is the sole predictor in the first two levels of the tree. Price difference is the next major predictor to be used in the tree. Price and sale price of Minute Maid are used in the last level of the tree.

Part E

```r
# Use classification tree to predict responses of test data
oj.pred <- predict(tree.oj, test.oj, type = "class")
# Generate confusion matrix
table(oj.pred, test.oj$Purchase)
```

```
##
## oj.pred  CH  MM
##      CH 148  31
##      MM  15  76
```

```r
(15 + 31) / (148 + 31 + 15 + 76)
```

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

The test error rate is 17%.

Part F

```r
# Use cross-validation to determine if pruning the tree may result in better prediction error
set.seed(4)
cv.oj <- cv.tree(tree.oj, FUN = prune.misclass)
names(cv.oj)
```

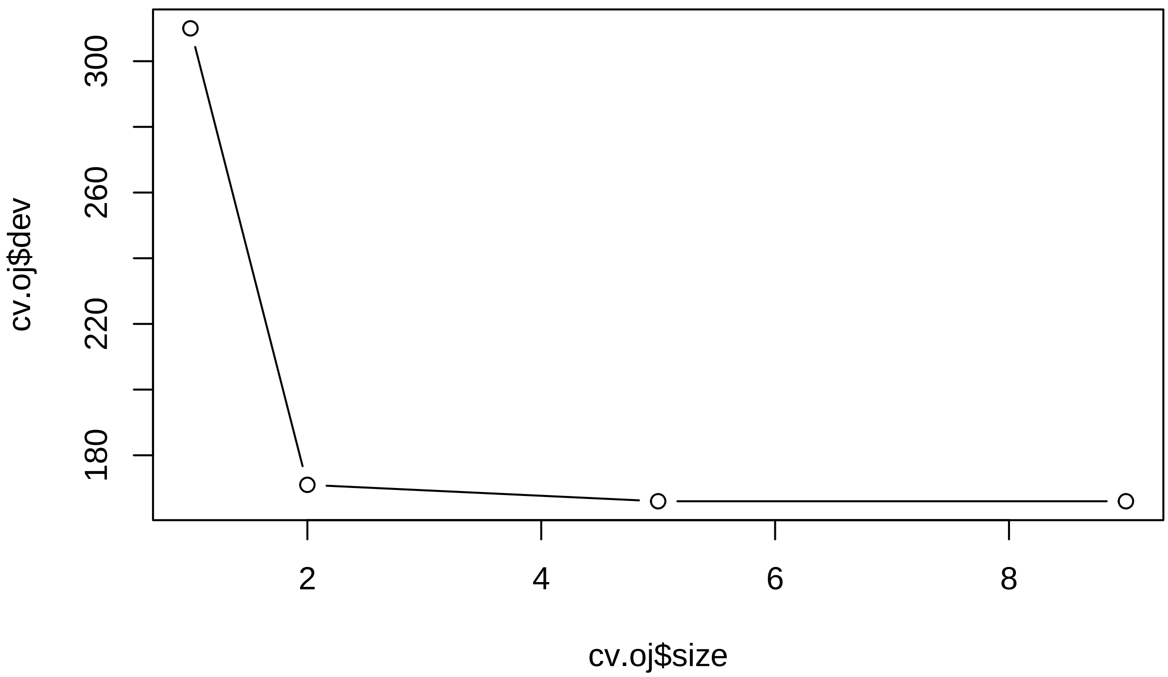
```
## [1] "size"   "dev"    "k"      "method"
```

```r
cv.oj
```

```
## $size
## [1] 9 5 2 1
##
## $dev
## [1] 166 166 171 310
##
## $k
## [1]       -Inf   0.000000   5.666667 148.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Part G

```r
# Create plot of cross-validated training error over tree size
plot(cv.oj$size, cv.oj$dev, type = "b")
```

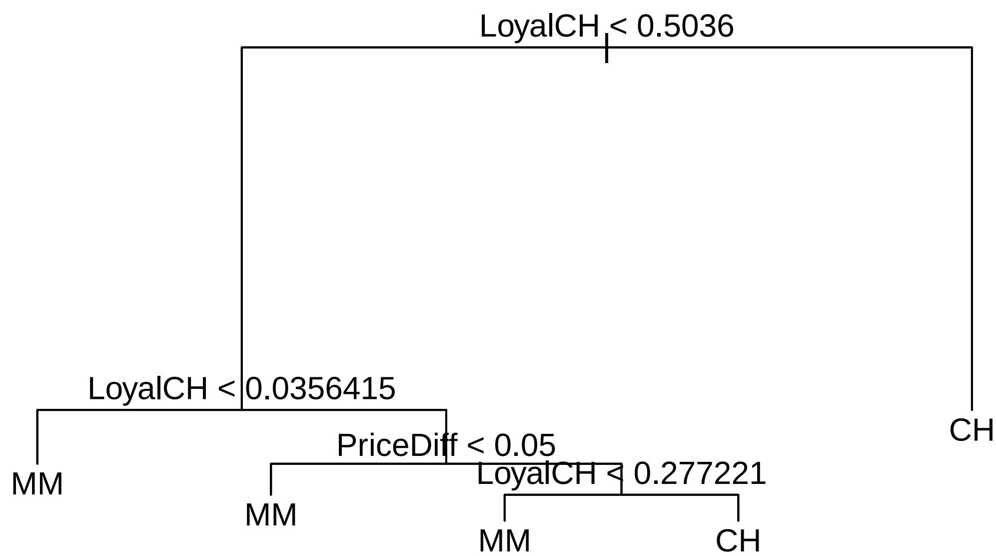


Part H

Tree size 5 corresponds to the lowest cross-validated classification error.

Part I

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




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

```
##
## Classification tree:
## snip.tree(tree = tree.oj, nodes = c(3L, 10L))
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  5
## Residual mean deviance:  0.8703 = 691.9 / 795
## Misclassification error rate: 0.1812 = 145 / 800
```

Part J

The training errors are the same for the pruned tree and the orginal classification tree we created.

Part K

```r
# Predict response with pruned tree
prune.oj.pred <- predict(prune.oj, test.oj, type = "class")
# Generate confusion matrix for pruned tree
table(prune.oj.pred, test.oj$Purchase)
```

```
##
## prune.oj.pred  CH  MM
##            CH 148  31
##            MM  15  76
```

```
(15 + 31) / (148 + 31 + 15 + 76)
```

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

The pruned tree yields the same test error as the original tree.

Question 10

Part A

```
hitters.data <- na.omit(Hitters)
hitters.data$Salary <- log(hitters.data$Salary)
```

Part B

```
hitters.train <- c(1:200)
hitters.test <- c(201:nrow(hitters.data))
```

Part C

```
library(gbm)
```

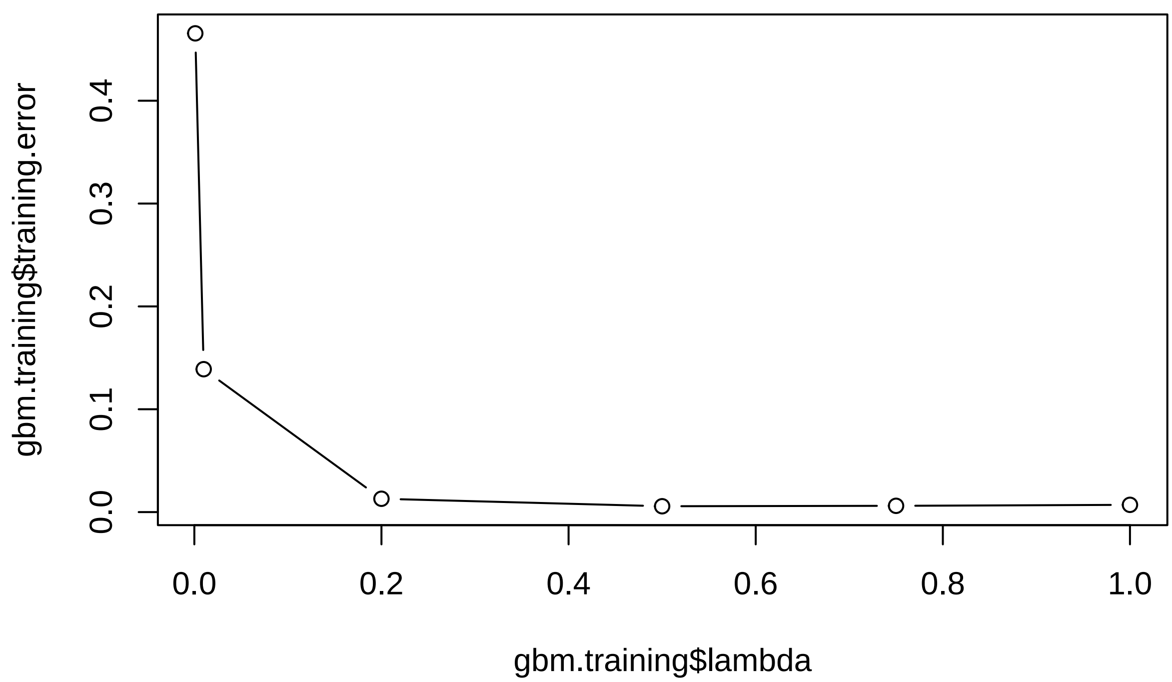
```
## Warning: package 'gbm' was built under R version 4.3.2
```

```
## Loaded gbm 2.1.8.1
```

```
tunings <- c(0.001, 0.01, 0.2, 0.5, 0.75, 1.0)
gbm.training <- data.frame(lambda = tunings,
                           training.error = rep(NA, length(tunings)))
for (x in 1:length(tunings)) {
  set.seed(5)
  boost.hitters <- gbm(Salary ~ ., data = hitters.data[hitters.train, ],
                     distribution = "gaussian", n.trees = 1000,
                     interaction.depth = 4, shrinkage = tunings[x])
  gbm.training[x, "training.error"] <-  mean(boost.hitters$train.error)
}

gbm.training
```

```
##   lambda training.error
## 1  0.001    0.465506603
## 2  0.010    0.138947034
## 3  0.200    0.012968059
## 4  0.500    0.005684627
## 5  0.750    0.006102646
## 6  1.000    0.007057622
```

```
plot(gbm.training$lambda, gbm.training$training.error, type = "b")
```



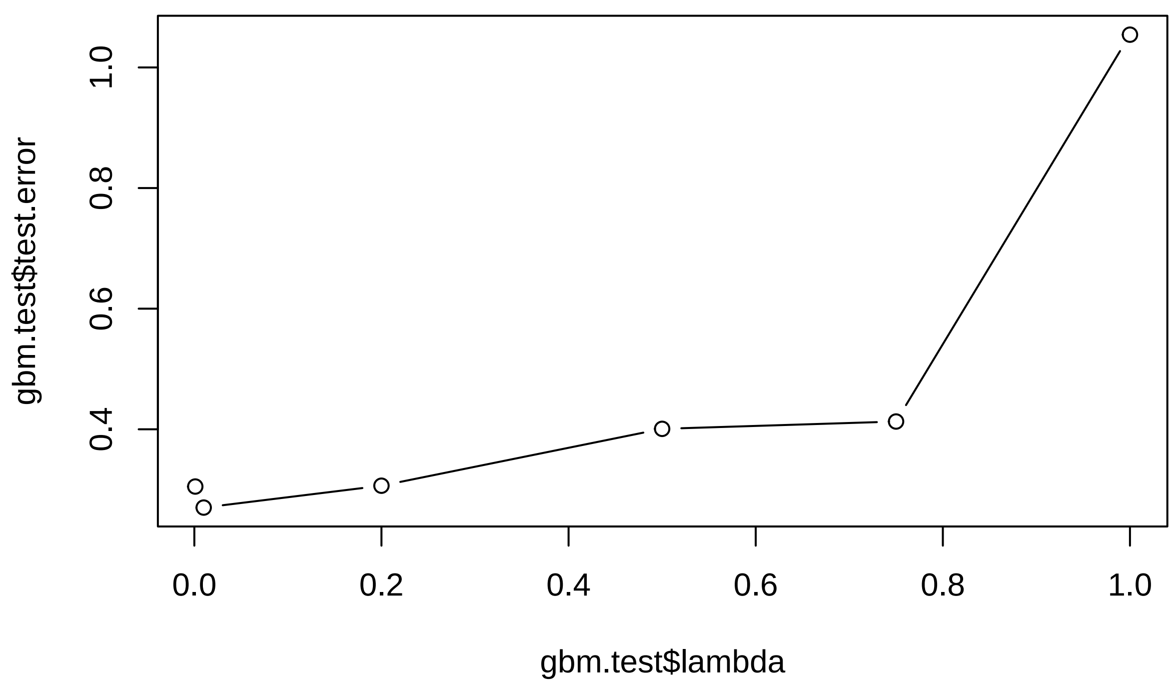
Part D

```
gbm.test <- data.frame(lambda = tunings,
                       test.error = rep(NA, length(tunings)))
for (x in 1:length(tunings)) {
  set.seed(5)
  boost.hitters <- gbm(Salary ~ ., data = hitters.data[hitters.train, ],
                     distribution = "gaussian", n.trees = 1000,
                     interaction.depth = 4, shrinkage = tunings[x])
  yhat.boost <- predict(boost.hitters, newdata = hitters.data[hitters.test, ],
                        n.trees = 1000)
  gbm.test[x, "test.error"] <-  mean((yhat.boost - hitters.data$Salary[hitters.test])^2)
}
```

```
gbm.test
```

```
##   lambda test.error
## 1  0.001  0.3050198
## 2  0.010  0.2701642
## 3  0.200  0.3064144
## 4  0.500  0.4007037
## 5  0.750  0.4128302
## 6  1.000  1.0543288
```

```
plot(gbm.test$lambda, gbm.test$test.error, type = "b")
```



Part E

```
# Create LS regression with Salary as the predictor and determine test MSE
lm.hitters <- lm(Salary ~ ., data = hitters.data, subset = hitters.train)
lm.predict <- predict(lm.hitters, newdata = hitters.data[hitters.test, ])
lm.mse <- mean((lm.predict - hitters.data$Salary[hitters.test])^2)
lm.mse
```

```
## [1] 0.4917959
```

```r
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.3.2
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

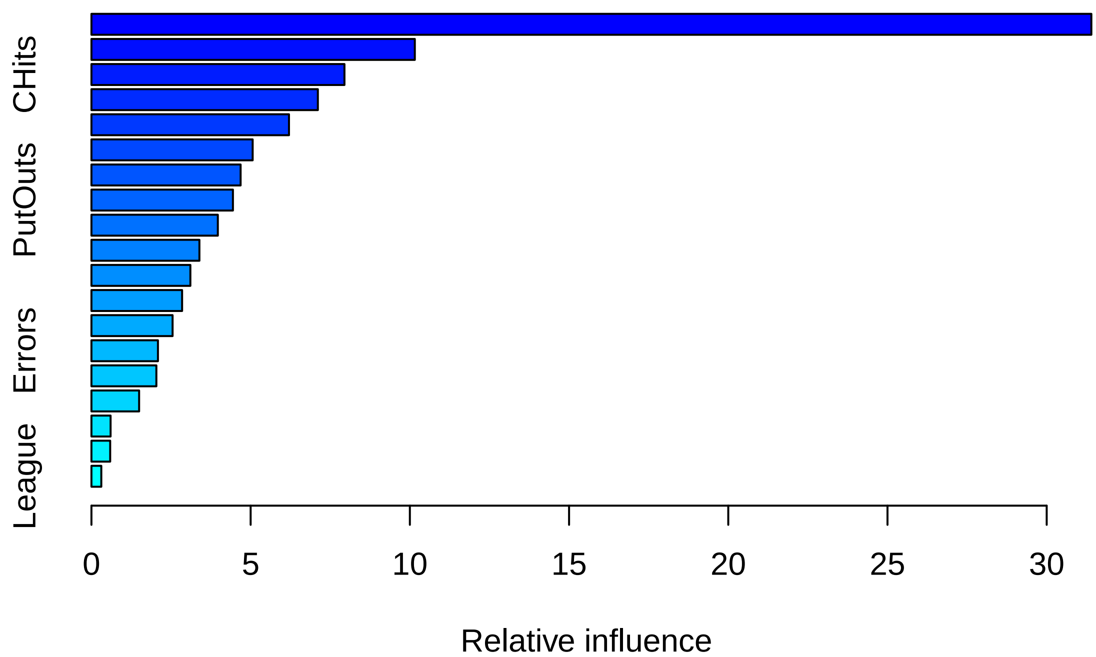
```r
x <- model.matrix(Salary ~ ., hitters.data)[, -1]
y <- hitters.data$Salary

lambda.grid <- 10^seq(10, -2, length = 100)
lasso.mod <- glmnet(x[hitters.train, ], y[hitters.train], alpha = 1,
                    lambda = lambda.grid)
set.seed(7)
cv.out <- cv.glmnet(x[hitters.train, ], y[hitters.train], alpha = 1)
bestlam <- cv.out$lambda.min

lasso.pred <- predict(lasso.mod, s = bestlam, newx = x[hitters.test, ])
lasso.mse <- mean((lasso.pred - y[hitters.test])^2)
lasso.mse
```

```
## [1] 0.470371
```

The boosting MSE when lambda = 0.01 is much smaller than the MSE of a multiple linear regression and the Lasso.

Part F

```r
# Create boosting model using shrinkage, or tuning parameter, equal to 0.01
set.seed(5)
boost.hitters <- gbm(Salary ~ ., data = hitters.data[hitters.train, ],
                   distribution = "gaussian", n.trees = 1000,
                   interaction.depth = 4, shrinkage = 0.01)
summary(boost.hitters)
```




```
##                 var    rel.inf
## CAtBat       CAtBat 31.4028722
## CWalks       CWalks 10.1559784
## CHits         CHits  7.9460562
## CRBI           CRBI  7.1113938
## CRuns         CRuns  6.2042978
## Years         Years  5.0629877
## Walks         Walks  4.6850235
## PutOuts     PutOuts  4.4432945
## CHmRun       CHmRun  3.9699283
## AtBat         AtBat  3.3906305
## Assists     Assists  3.1067186
## RBI             RBI  2.8470835
## Hits           Hits  2.5482569
## Errors       Errors  2.0887654
## HmRun         HmRun  2.0395010
## Runs           Runs  1.4972505
## NewLeague NewLeague  0.6009227
## Division   Division  0.5889558
## League       League  0.3100826
```

CAtBat is clearly the most important predictor in the boosting model. CWalks, CRuns, and CRBI are the next highest in importance.

Part G

```r
set.seed(5)
bag.hitters <- randomForest(Salary ~ ., data = hitters.data,
                            subset = hitters.train, mtry = 19,
                            importance = T)
yhat.bag <- predict(bag.hitters, newdata = hitters.data[hitters.test, ])
mean((yhat.bag - hitters.data$Salary[hitters.test])^2)
```

```
## [1] 0.233566
```

Bagging yields a test MSE of 0.234. This is slightly lower than the test MSE of boosting with tuning parameter equal to 0.01, which was 0.270.