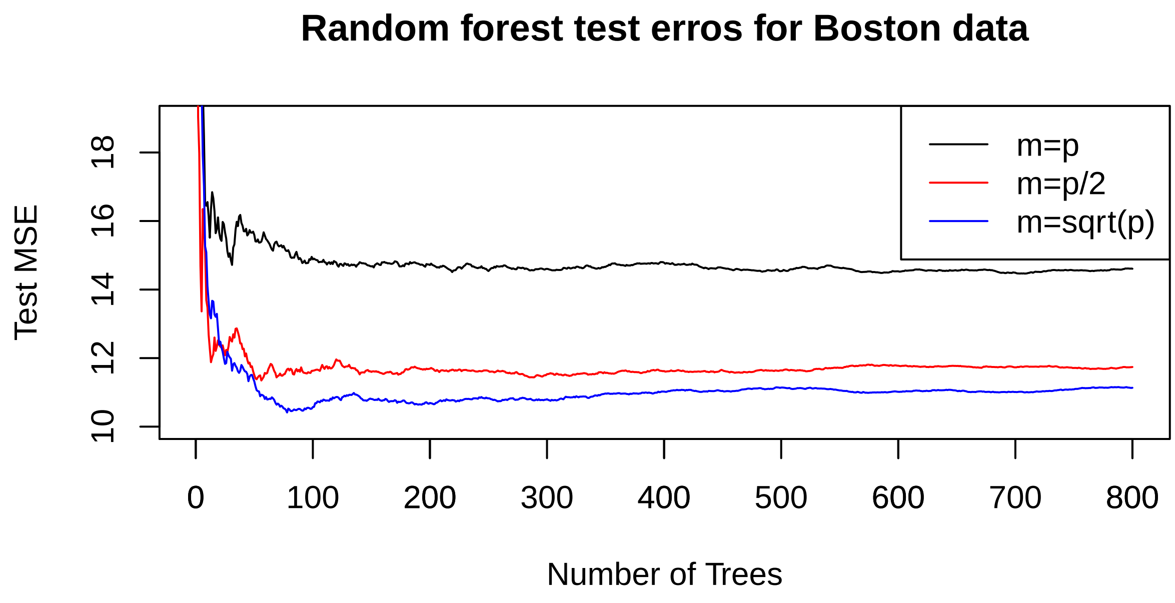
Assignment 3

Exercise 1

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.

```r
library(MASS)
library(randomForest)

# Construct the train and test matrices
set.seed(123)
train = sample(dim(Boston)[1], 2*dim(Boston)[1]/3, replace = F) # 2/3 split

# Attributes
X.train = Boston[train, -14]
X.test = Boston[-train, -14]

# Response
Y.train = Boston[train, 14]
Y.test = Boston[-train, 14]

# Number of variables used in each split
p = dim(Boston)[2]-1 # All attributes
p.2 = p/2 # Half
p.sq = sqrt(p) # Square root of all possibilities

# Models for each split type, number of trees is 800
rf.boston.p = randomForest(X.train, Y.train, xtest=X.test,
                          ytest=Y.test, mtry=p, ntree=800)
rf.boston.p.2 = randomForest(X.train, Y.train, xtest=X.test,
                            ytest=Y.test, mtry=p.2, ntree=800)
rf.boston.p.sq = randomForest(X.train, Y.train, xtest=X.test,
                             ytest=Y.test, mtry=p.sq, ntree=800)

# Plotting the results
plot(1:800, rf.boston.p$test$mse, col=1, type="l", xlab="Number of Trees",
     ylab="Test MSE", ylim=c(floor(min(c(rf.boston.p$test$mse, rf.boston.p.2$test$mse,
     rf.boston.p.sq$test$mse))),19), main="Random forest test erros for Boston data")
lines(1:800, rf.boston.p.2$test$mse, col=2, type="l")
lines(1:800, rf.boston.p.sq$test$mse, col=4, type="l")
legend("topright", c("m=p", "m=p/2", "m=sqrt(p)"), col=c(1, "red", "blue"),
       cex=1, lty=1)
axis(1, seq(0,800,100))

paste("The lowest test error:", round(min(c(rf.boston.p$test$mse,
      rf.boston.p.2$test$mse,rf.boston.p.sq$test$mse)),2))
```

```
[1] "The lowest test error: 10.41"
```



The lowest test set error is achieved with $\sqrt{p}$ as a criterion for each split, where p is the number of predictors. Note that the result varies with different seeds due to randomness of the splitting processes (data, branches). The cirerion which uses all predictors for each split has the highest test error being around 15 to 17. It seems that the test error stabilizes when the tree size is approximately 150.

Exercise 2

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.

```
# Bagging
set.seed(123)
# randomForest() method can be used to bagging if mtry is equal to number of
# all predicting attributes
bag.carseats = randomForest(Sales ~ ., data = trainCAR, mtry = 10, ntree = 500,
    importance = T)

# Making predictions
bag.pred = predict(bag.carseats, testCAR)

# Test MSE
paste("The test MSE is", mean((testCAR$Sales - bag.pred)^2))

# Importance
paste("Importance of attributes")
importance(bag.carseats)
```

```
[1] "The test MSE is 2.28200717599288"
[1] "Importance of attributes"
              %IncMSE IncNodePurity
CompPrice   26.264150     192.24925
```

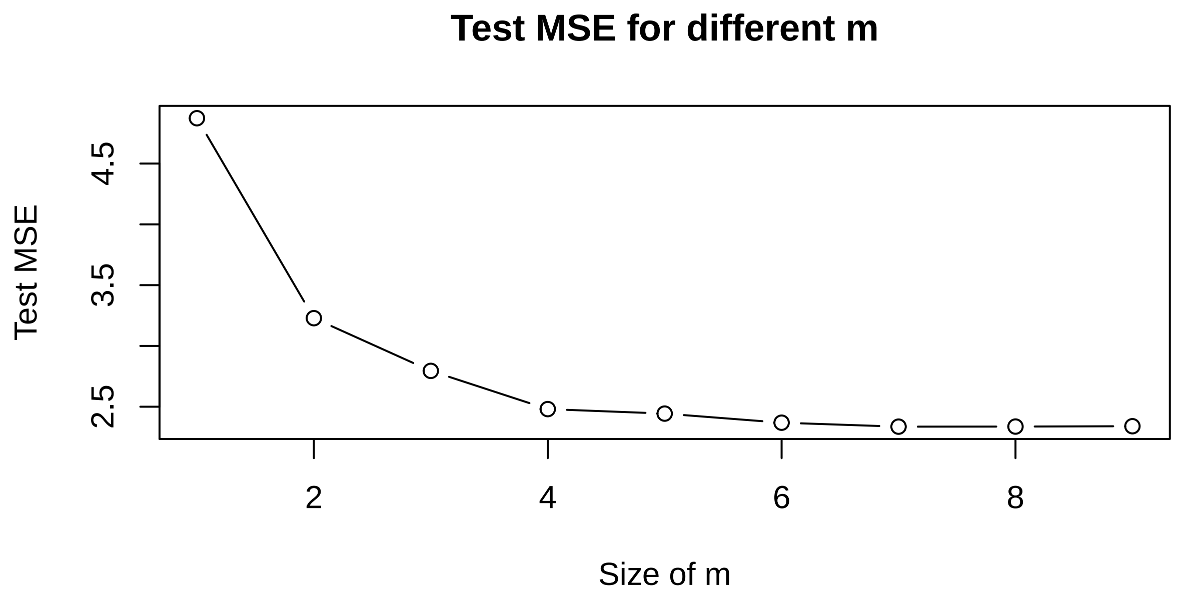
```
Income       9.517596     114.31264
Advertising 13.056754     114.11755
Population   1.195061      81.77791
Price       60.006974     598.03404
ShelveLoc   68.487544     592.00725
Age         23.012641     242.63878
Education    3.101703      66.83355
Urban        2.565957      10.91683
US           2.620459      10.43320
```

Bagging improves the test MSE to 2.28 and the three most important predictors are `ShelveLoc`, `Price` and `CompPrice` in order of importance.

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.

```r
# Random forest using squareroot of attributes
bag.carseats = randomForest(Sales~., data=trainCAR, mtry=sqrt(10),
                            ntree=500, importance=T)
bag.pred = predict(bag.carseats, testCAR)
paste("The test MSE is", mean((testCAR$Sales - bag.pred)^2))
paste("Importance of attributes")
importance(bag.carseats)

# Simulating with different m

testMSE = c()
for (m in 1:9){
  set.seed(123)
  bag.carseats = randomForest(Sales~., data=trainCAR, mtry=m, ntree=500)
  bag.pred = predict(bag.carseats, testCAR)
  testMSE[m] = mean((testCAR$Sales - bag.pred)^2)
}
plot(testMSE, type = "b", xlab = "Size of m", ylab = "Test MSE",
     main = "Test MSE for different m")
```

```
[1] "The test MSE is 2.79832577920864"
[1] "Importance of attributes"
               %IncMSE IncNodePurity
CompPrice   10.9586679     185.05891
Income       5.6921913     171.21686
Advertising 11.7827778     151.55667
Population   1.9851092     140.35235
Price       39.8149820     460.90185
ShelveLoc   47.1279904     446.25433
Age         19.2939832     265.12699
Education    2.7993297      90.39131
Urban        0.2740763      20.84900
US           5.1200974      23.81024
```



The test MSE increases a bit to 2.71 and the third most important predictor changed into `Age`.

Exercises 3 and 4

We now use boosting to predict Salary in the Hitters data set.

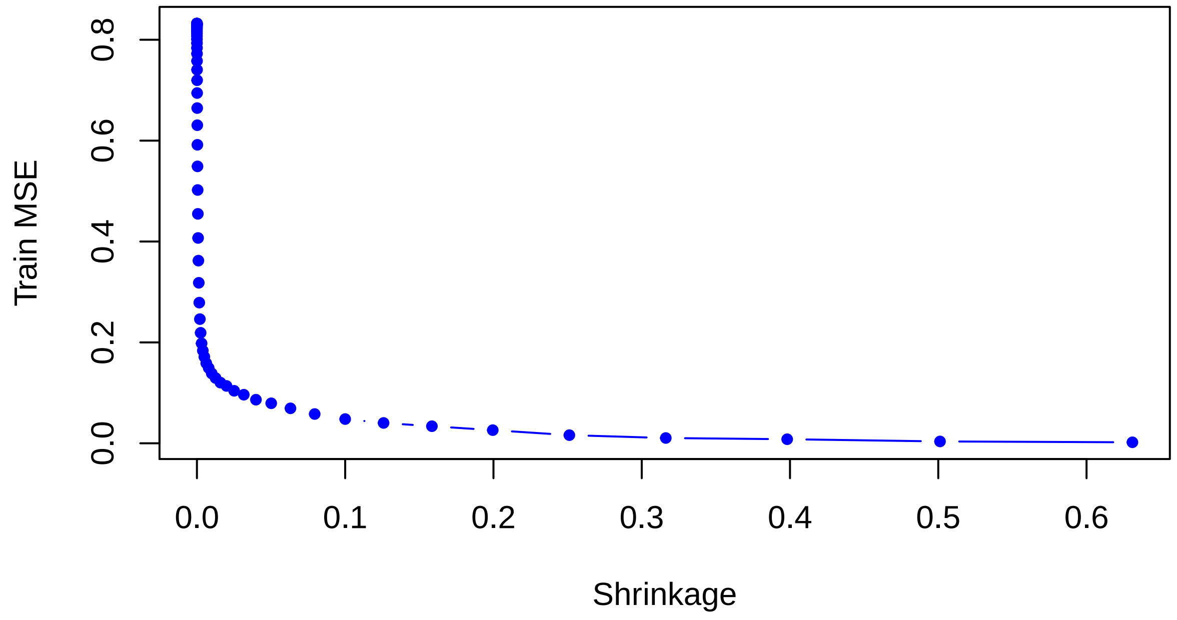
(a-b) Remove the observations for whom the salary information is unknown, and then log-transform the salaries. Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

```
# Remove observations with missing values in salary
remove = which(is.na(Hitters$Salary))
HT = Hitters[-remove, ]

# log-transform
HT$Salary = log(HT$Salary)

# Training and test set
train = 1:200
trainHT = HT[train, ]
testHT = HT[-train, ]
```

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

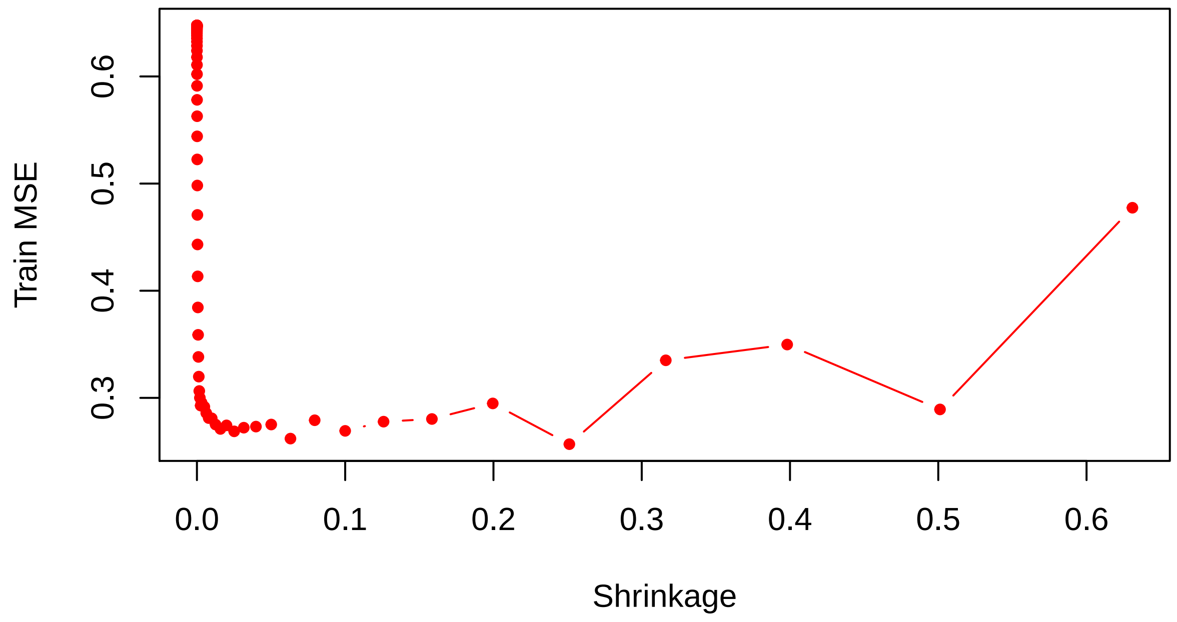
```
library(gbm)
```

```r
set.seed(123)

# Vector of different powers
pows = seq(-10, -0.2, by = 0.1)

# Vector of different lambdas
lambdas = 10^pows
length.lambdas = length(lambdas)

# Empty vectors to store errors
train.errors = rep(NA, length.lambdas)
test.errors = rep(NA, length.lambdas)

# For each value of lambda
for (i in 1:length.lambdas) {
    # Performing boosting for 100 trees with given lambda value
    boost.hitters = gbm(Salary ~ ., data = trainHT, distribution = "gaussian",
        n.trees = 1000, shrinkage = lambdas[i])
    # Storing predicted values
    train.pred = predict(boost.hitters, trainHT, n.trees = 1000)
    test.pred = predict(boost.hitters, testHT, n.trees = 1000)
    # Calculating and storing errors
    train.errors[i] = mean((trainHT$Salary - train.pred)^2)
    test.errors[i] = mean((testHT$Salary - test.pred)^2)
}

# Pltoting lambdas against training errors
par(mar = c(5.1, 4.1, 1, 2.1))
plot(lambdas, train.errors, type = "b", xlab = "Shrinkage", ylab = "Train MSE",
    col = 4, pch = 20)
```



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

```
[1] "The minimum test error for boosting is 0.26 for lambda value of 0.25"
```



As seen above there's a good reason why training error shouldn't be used as a criterion for choosing the final model or model parameters.

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

```
# Fitting linear model
lm.fit = lm(Salary ~ ., data = trainHT)

# Forming predictions
lm.pred = predict(lm.fit, testHT)

# Calculating test MSE
paste("The test MSE for linear model is", round(mean((testHT$Salary - lm.pred)^2),
    2))

library(glmnet)

set.seed(123)

# Forming model matrices and response vector y
X = model.matrix(Salary ~ ., data = trainHT)
X.test = model.matrix(Salary ~ ., data = testHT)
y = trainHT$Salary

# Fitting lasso
```

```
lasso.cv = cv.glmnet(X, y)
minlambda = lasso.cv$lambda.min

# Storing predictions
lasso.pred = predict(lasso.cv$glmnet.fit, s = minlambda, newx = X.test)

# Test MSE
paste("The test MSE for lasso is", round(mean((testHT$Salary - lasso.pred)^2),
    2))
```

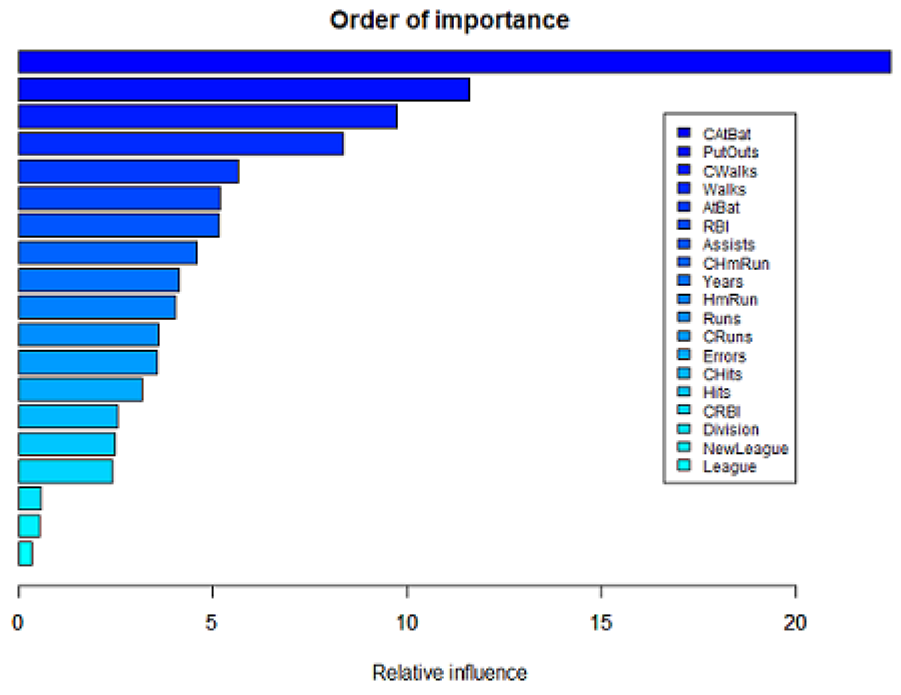
```
[1] "The test MSE for linear model is 0.49"
[1] "The test MSE for lasso is 0.47"
```

Both Lasso and OLS model have higher test error rates than boosting.

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

```
# Boosted model which uses the lambda for which the test error minimizes
boost.best = gbm(Salary ~ ., data = trainHT, distribution = "gaussian", n.trees = 1000,
    shrinkage = lambdas[which.min(test.errors)])

head(summary(boost.best, legend.text = T, args.legend = list(cex = 0.8, x = 20,
    y = 20), names.arg = "", main = "Order of importance", xlim = c(0, 20)))
```

```
            var   rel.inf
CAtBat   CAtBat 15.394543
CWalks   CWalks 11.454057
PutOuts PutOuts 10.021110
Walks     Walks  8.711869
AtBat     AtBat  6.144511
CRuns     CRuns  5.682345
```



The three most important predictors in decreasing order of importance are `CAtBat`, `PutOuts` and `CWalks`.

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

```r
set.seed(123)

# Bagging mtry = number of all predictors
rf.hitters = randomForest(Salary ~ ., data = trainHT, ntree = 500, mtry = 19)

# Predictions
rf.pred = predict(rf.hitters, testHT)

# Test MSE
paste("The test MSE for bagging is", round(mean((testHT$Salary - rf.pred)^2),
    2))
```

```
[1] "The test MSE for bagging is 0.23"
```

The test error for bagging is slightly lower than test MSE for boosting.

Exercise 5

This question uses the Caravan data set.

(a) Create a training set consisting of the first 1,000 observations, and a test set consisting of the remaining observations.

```r
# Checking for missing values
paste("Number of missing values:", sum(is.na(Caravan)))

caravan = Caravan

# Decoding response variable as dummy variable
caravan$Purchase = ifelse(caravan$Purchase == "Yes", 1, 0)

# Training and test set
train = 1:1000
trainCaravan = caravan[train, ]
testCaravan = caravan[-train, ]
```

```
[1] "Number of missing values: 0"
```

(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?

```r
# Boosting (repsonse is binary, so distribution is bernoulli)
set.seed(123)
boostCaravan = gbm(Purchase ~ ., data = trainCaravan, distribution = "bernoulli",
    n.trees = 1000, shrinkage = 0.01)
```

```r
# Importance
head(summary(boostCaravan, plotit = F))
```

```
              var   rel.inf
PPERSAUT PPERSAUT 14.681380
MKOOPKLA MKOOPKLA  9.244102
MOPLHOOG MOPLHOOG  7.156570
MBERMIDD MBERMIDD  6.311954
PBRAND     PBRAND  5.410268
MGODGE     MGODGE  4.557510
```

The three most important predictors in decreasing order of importance are PPERSAUT, MKOOPKLA and MOPLHOOG

(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?

```r
# Predictions
boost.prob = predict(boostCaravan, testCaravan, n.trees = 1000, type = "response")

# Storing if probability is > 0.2
boost.pred = ifelse(boost.prob > 0.2, 1, 0)

# Confusion matrix
confm = table(testCaravan$Purchase, boost.pred)
confm

# Actually buy
(confm[2, 2])/(sum(confm[, 2]))
```

```
   boost.pred
       0    1
  0 4420  113
  1  256   33
[1] 0.2260274
```

Approximately 23 % of customers predicted to make the purchase actually buy the product.

```r
library(class)

# Preparing data
caravan = Caravan

# Recommended to standardize data
stand.X = scale(caravan[, -86])
train = 1:1000
train.stan.X = stand.X[train, ]
test.stan.X = stand.X[-train, ]
train.y = caravan[train, 86]
test.y = caravan[-train, 86]
```

```r
# kNN
set.seed(123)
kNN = knn(train.stan.X, test.stan.X, train.y, k = 2)

# Confusion matrix
confmt = table(kNN, test.y)
confmt

# Actually buy
(confmt[2, 2])/(sum(confmt[, 2]))
```

```
     test.y
kNN     No  Yes
  No  4246  253
  Yes  287   36
[1] 0.1245675
```

Approximately 12 % of customers predicted to make the purchase actually buy the product which is less than previously got with boosting.