Assignment 4

Statistics and Data Science 365/565

Due: October 23 (before 11:59 pm)

This assignment focuses on the topics of tree-based methods, the Adaboost algorithm, and gradient boosting.

Problem 1 (20 points)

This problem is based on the `Carseats` data set from the `ISLR` package. We will seek to predict `Sales` using regression trees and related approaches, treating the response as a quantitative variable.

Part a.

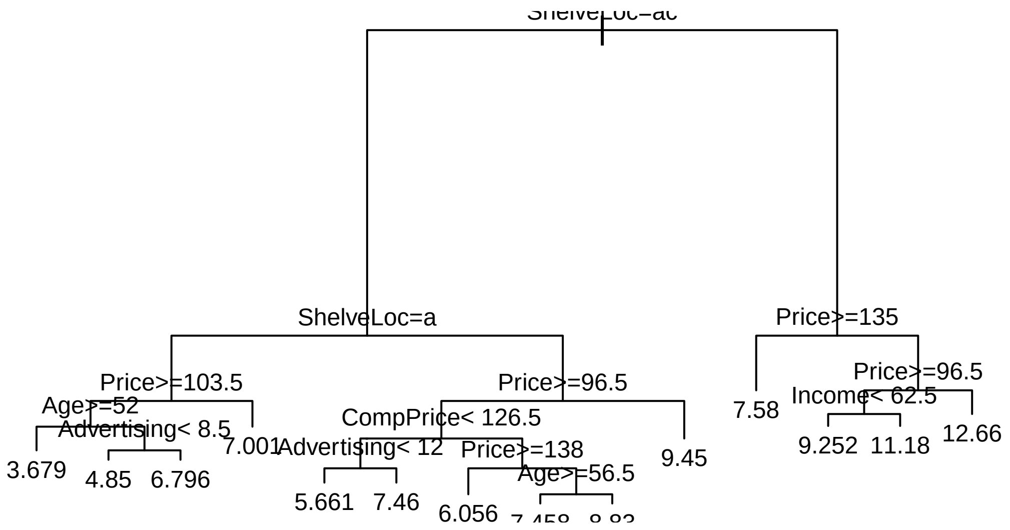
Split the data set into a training set and a test set (50/50). Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

```
#Randomly split into training and test set.
set.seed(1)
testrows <- sample(nrow(Carseats),nrow(Carseats)/2,replace=F)
test <- Carseats[testrows,]
train <- Carseats[-testrows,]

#Fit tree using rpart() function.
mod1 <- rpart(Sales~., data=train, method="anova")
mod1
```

```
## n= 200
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 200 1645.38700  7.657600
##    2) ShelveLoc=Bad,Medium 151  774.83190  6.706821
##      4) ShelveLoc=Bad 47  240.75950  5.385745
##        8) Price>=103.5 34  128.11490  4.768235
##         16) Age>=52 14   28.98617  3.678571 *
##         17) Age< 52 20   70.86938  5.531000
##           34) Advertising< 8.5 13   35.07780  4.850000 *
##           35) Advertising>=8.5 7   18.56617  6.795714 *
##        9) Price< 103.5 13   65.77189  7.000769 *
##      5) ShelveLoc=Medium 104  414.97630  7.303846
##       10) Price>=96.5 91  309.44880  6.997253
##         20) CompPrice< 126.5 41   99.52852  6.143415
##           40) Advertising< 12 30   50.67039  5.660667 *
##           41) Advertising>=12 11   22.79940  7.460000 *
##         21) CompPrice>=126.5 50  155.51940  7.697400
##           42) Price>=138 13   24.96331  6.056154 *
##           43) Price< 138 37   83.23449  8.274054
##             86) Age>=56.5 15   29.38444  7.458000 *
##              87) Age< 56.5 22   37.05010  8.830455 *
##        11) Price< 96.5 13   37.09580  9.450000 *
##    3) ShelveLoc=Good 49  313.40850 10.587550
##      6) Price>=135 9   38.29540  7.580000 *
##      7) Price< 135 40  175.38800 11.264250
##        14) Price>=96.5 26   82.40410 10.510380
##          28) Income< 62.5 9   23.25476  9.252222 *
##          29) Income>=62.5 17   37.36019 11.176470 *
##        15) Price< 96.5 14   50.76634 12.664290 *
```

```
plot(mod1)
text(mod1,cex=0.75)
```



```
#Find MSE
testpred <- predict(mod1,newdata = test)
testMSE <- sum((testpred-test$Sales)^2)/nrow(test)
print(paste("Test MSE:", testMSE))
```

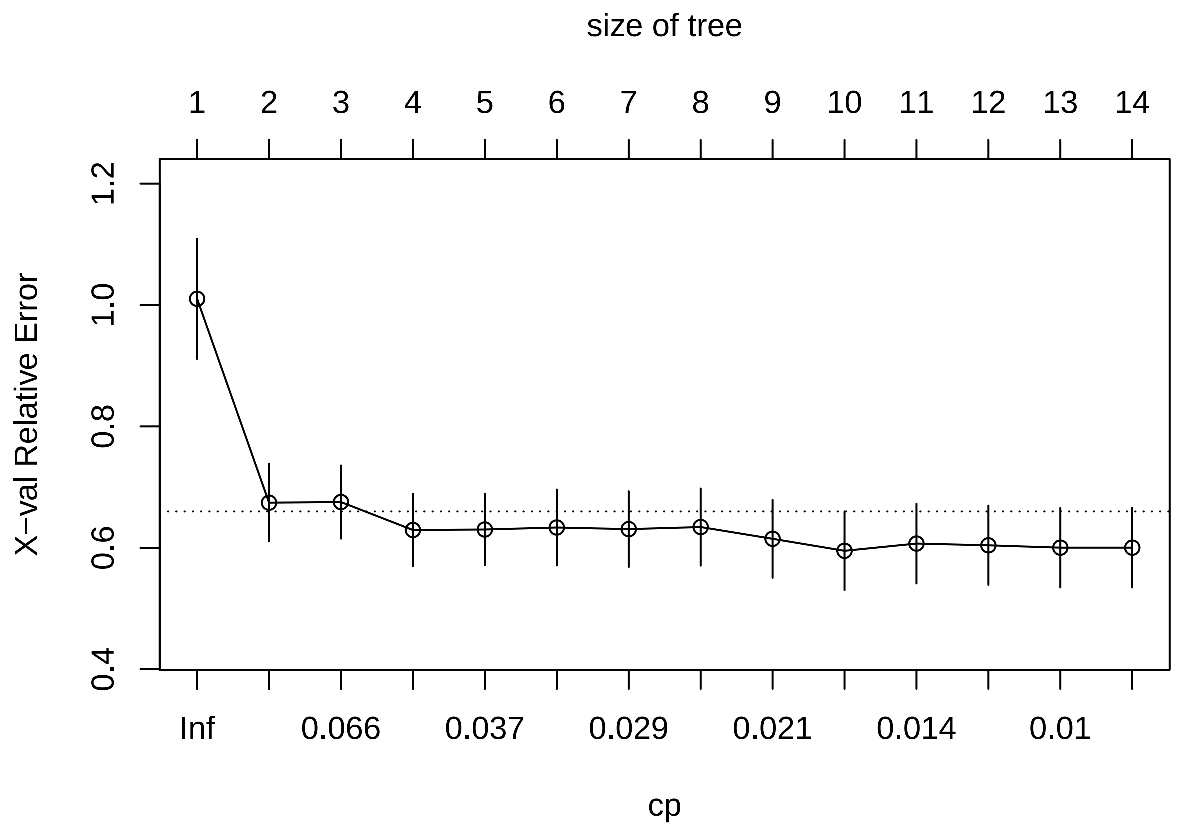
```
## [1] "Test MSE: 4.31414212544268"
```

Comments: It is important to note that if the data observation achieves the condition at each node, then it moves to the left. In the plotted tree, "ShelveLoc=ac" means that the shelving location for the carseat is either bad or medium and "ShelveLoc=a"means the shelving location was bad. Age indicates the age of the local population, thus we can see carseats sold in older population areas tend to be less expensive (they are not having kids and thus do not need to purchase carseats). Price and comparable pricing makes sense, as cheaper carseats will sell better particularly when compeating carseats are more expensive.

Part b.

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

```
set.seed(1)
plotcp(mod1)
```



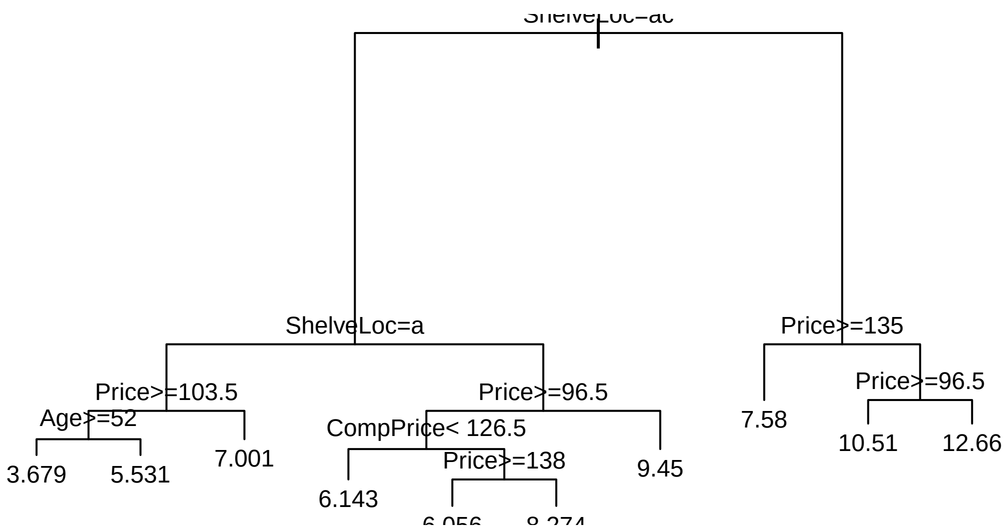
```
cpmin <- mod1$cptable[which.min(mod1$cptable[,4]),1]
splitsmin <- mod1$cptable[which.min(mod1$cptable[,4]),1]
print(paste("CP for Min Error:",cpmin))
```

```
## [1] "CP for Min Error: 0.0158374462599844"
```

```
print(paste("Number of Splits for Min Error:",splitsmin))
```

```
## [1] "Number of Splits for Min Error: 0.0158374462599844"
```

```
#Now we prune with 10 splits (our minimum cross-validation error rate), and the equivalent cp.
mod2 <- prune(mod1,cp=cpmin)
plot(mod2)
text(mod2,cex=0.75)
```



```
#New test MSE with pruned tree determined by cross-validation.
testpred <- predict(mod2,newdata = test)
testMSE <- sum((testpred-test$Sales)^2)/nrow(test)
print(paste("Test MSE:", testMSE))
```

```
## [1] "Test MSE: 4.55936247208962"
```

Comments: We select the number of 10 splits (our minimum cross-validation error rate), and the equivalent cp. Note that the top x-axis is wrong for the cross-validation plot, it should start with 0 not 1. Thus we achieve a minimum error at the number of splits equal to 9 not 10. The pruning does not improve the test MSE. We have an MSE of 4.559, which is larger than the MSE of 4.314. This can be imagined as a larger increase in bias than the decrease in variance that comes with reducing the depth of a tree. The increase in bias outweights in the change in variance.

Part c.

Now use random forests to analyze the 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.

```
set.seed(1)
#Random Forest fit using the default m-variables (p/3 for regression).
mod3 <- randomForest(Sales~., data=train, importance=T,ntree=500)
mod3$importance
```

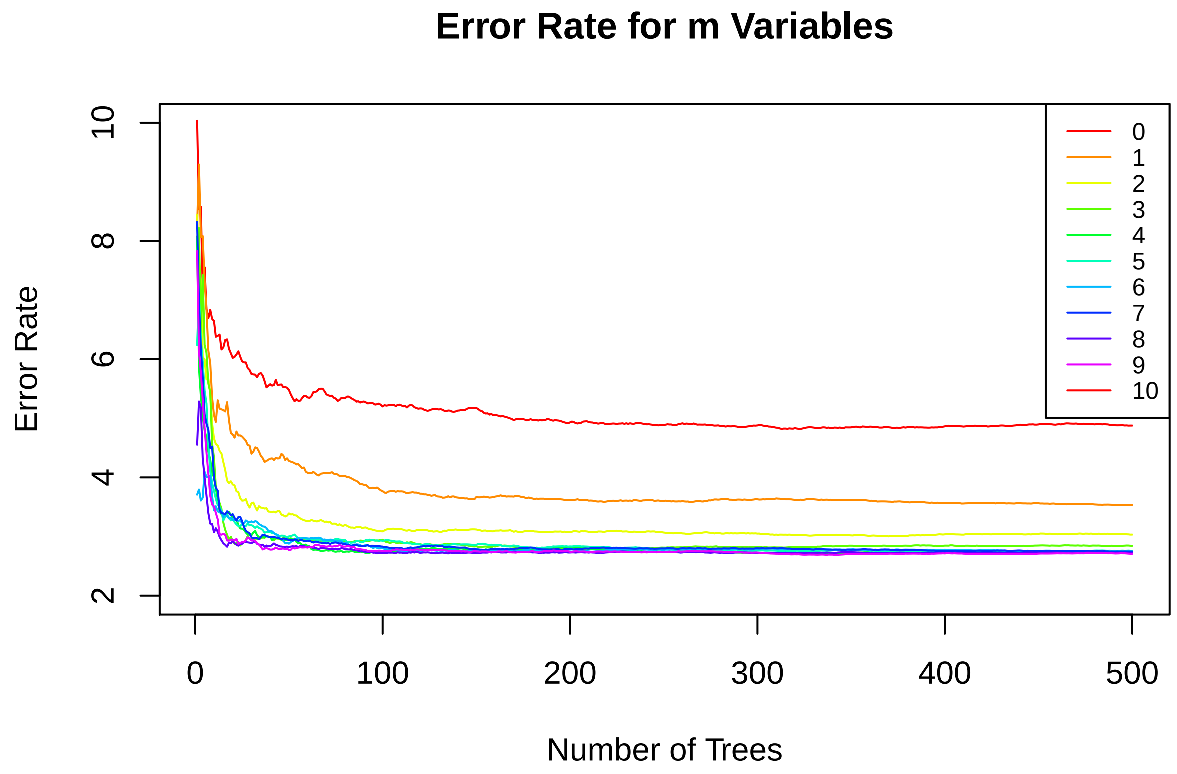
```
##                  %IncMSE IncNodePurity
## CompPrice    0.847421862     159.92376
## Income       0.120496450     117.51558
## Advertising  0.431348487     131.98549
## Population  -0.084490768      96.09845
## Price        2.664125912     334.63730
## ShelveLoc    4.418385286     457.03627
## Age          0.593059095     150.08894
## Education    0.126624931      72.46522
## Urban        0.002923479      14.38398
## US           0.025755596      14.83391
```

```
testpred <- predict(mod3,newdata = test)
testMSE <- sum((testpred-test$Sales)^2)/nrow(test)
```

```
print(paste("Test MSE:", testMSE))
```

```
## [1] "Test MSE: 3.02067056938934"
```

```
#Color palette.
colors <- rainbow(n=length(Carseats))
#Plot the error rate for the number of trees and number of variable m considered at each split.
mod4 <- randomForest(Sales~., data=train, importance=T,mtry=1,ntree=500)
plot(mod4$mse, col=colors[1], type="l", ylim=c(2,10),xlab="Number of Trees",ylab="Error Rate",main="Erro
for (i in 2:(length(Carseats)-1)) {
  mod4 <- randomForest(Sales~., data=train, importance=T,mtry=i,ntree=500)
  lines(mod4$mse, col=colors[i], type="l")
}
legend("topright",legend=c(1:length(Carseats)-1),col=colors[1:length(Carseats)-1], lty=1, cex=0.75,y.int
```



Comments: Here the mean of squared residuals (MSE) on the test data is 3.021 for the default $m = 3$. We can see that as the number of variables considered for each split increases, he error rate on the training data decreases. However, for $m \geq 4$ we see that error rate does not change much for a large number of trees. These error rates are far lower than those for pruning and the single tree formation. From the importance chart, we see that, using the node purity measurements, shelve location, price, and competitor price are the most imporant variables in predicitng sales. This makes sense as price is particularly imporant if a consumer is buying a product and so is shelving location, as consumers as often easily swayed but what products they can easily find and see. These are also the three most important predictors using the % inlcuded in the MSE measurements.

Problem 2 (20 points)

Suppose we have a dataset $\{(x_i, y_i)\}_{i=1}^n$ where $y_i \in \{\pm 1\}$ for each $i \in [n]$. Recall that at each iteration $t \in \{0, ..., T\}$, the Adaboost algorithm computes a classifier $f_t(x_i) \in \{\pm 1\}$ by minimizing the weighted error.

The algorithm then updates the weight for each point i in the following way:

$$w_{t+1}(i) = \frac{w_t(i)e^{-\alpha_t y_i f_t(x_i)}}{Z_t}$$

where $Z_t = \sum_{i \le n} w_t(i)e^{-\alpha_t y_i f_t(x_i)}$.

Part a.

Let $\epsilon_t = \sum_{i:f_t(x_i)\ne y_i} w_t(i)$; that is, ϵ_t is the sum of the weights at the points incorrectly classified at step t of the algorithm. Show that

$$Z_t = e^{\alpha_t}\epsilon_t + e^{-\alpha_t}(1-\epsilon_t).$$

Solution:

$$Z_t = \sum_{i\le n} w_t(i)e^{-\alpha_t y_i f_t(x_i)} = \sum_{i\le n} w_t(i)\begin{cases} e^{\alpha_t}, & \text{if } y_i \ne f_t(x_i) \\ e^{-\alpha_t}, & \text{if } y_i = f_t(x_i)\end{cases}$$

$$= \sum_{i\le n; y_i \ne f_t(x_i)} w_t(i) * e^{\alpha_t} + \sum_{i \le n; y_i = f_t(x_i)} w_t(i) * e^{-\alpha_t} = \epsilon_t e^{\alpha_t} + e^{-\alpha_t}(1-\epsilon_t)$$

This makes since because if $\epsilon_t = \sum_{i:f_t(x_i)\ne y_i} w_t(i)$, the sum of weights for the incorrectly clasified points. Then the sum of weights for the correctly classified points will just be $1-\epsilon_t$, as our weights are always being normalized to sum to one.

Part b.

Now show that the formula for the optimal α_t from class is given by the $\widehat{\alpha}_t$ that minimizes Z_t:

$$\widehat{\alpha}_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right).$$

Solution: We can minimize Z_t by taking the derivative of $Z_t = \epsilon_t e^{\alpha_t} + e^{-\alpha_t}(1-\epsilon_t)$ from part a) and setting it to zero.

$$\frac{\partial Z_t}{\partial \alpha_t} = \epsilon_t e^{\alpha_t} - e^{-\alpha_t}(1-\epsilon_t) = 0$$

$$\Rightarrow \epsilon_t e^{\alpha_t} = e^{-\alpha_t}(1-\epsilon_t) e^{2\alpha_t} = \frac{(1-\epsilon_t)}{\epsilon_t}\widehat{\alpha}_t = \frac{1}{2}ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

Going from the first to second line, we move the second term to the right side. Then we divide both sides by $e^{-\alpha_t}$. Last, we take the natural logarithm of both sides and divide both sides by 2 to get out solution for $\widehat{\alpha}_t$.

Problem 3 (40 points)

In this problem, you will train random forests to forecast the sale price of real estate listings. Random forests are nonparametric methods for classification and regression. As discussed in class, the method is based on the following thinking. A good predictor will have low bias and low variance. A deep decision tree has low bias, but high variance. To reduce the variance, multiple trees are fit and averaged together. By introducing randomness in the construction of the trees, the correlation between them is reduced, to facilitate the variance reduction.

Use the following variables: `Lat`, `Long`, `ListPrice`, `SaleYear`, `Bathroom`, `Bedroom`, `BuildDecade`, `MajorRenov`, `FinishSqFt`, `LotSqFt`, `MSA`, `City`, `HighSchool`, `SalePrice`. You will build regression models to predict `SalePrice`.

Read in the training and test sets as follows:

Part a.

Explore the data. As usual, you might ask yourself what n and p are here. Make plots of the distributions of the variables. Include a plot of the response, `SalePrice`. Does it appear that the data are "raw" or that they have been pre-processed in different ways? If so, how?

```
dim(train)
```

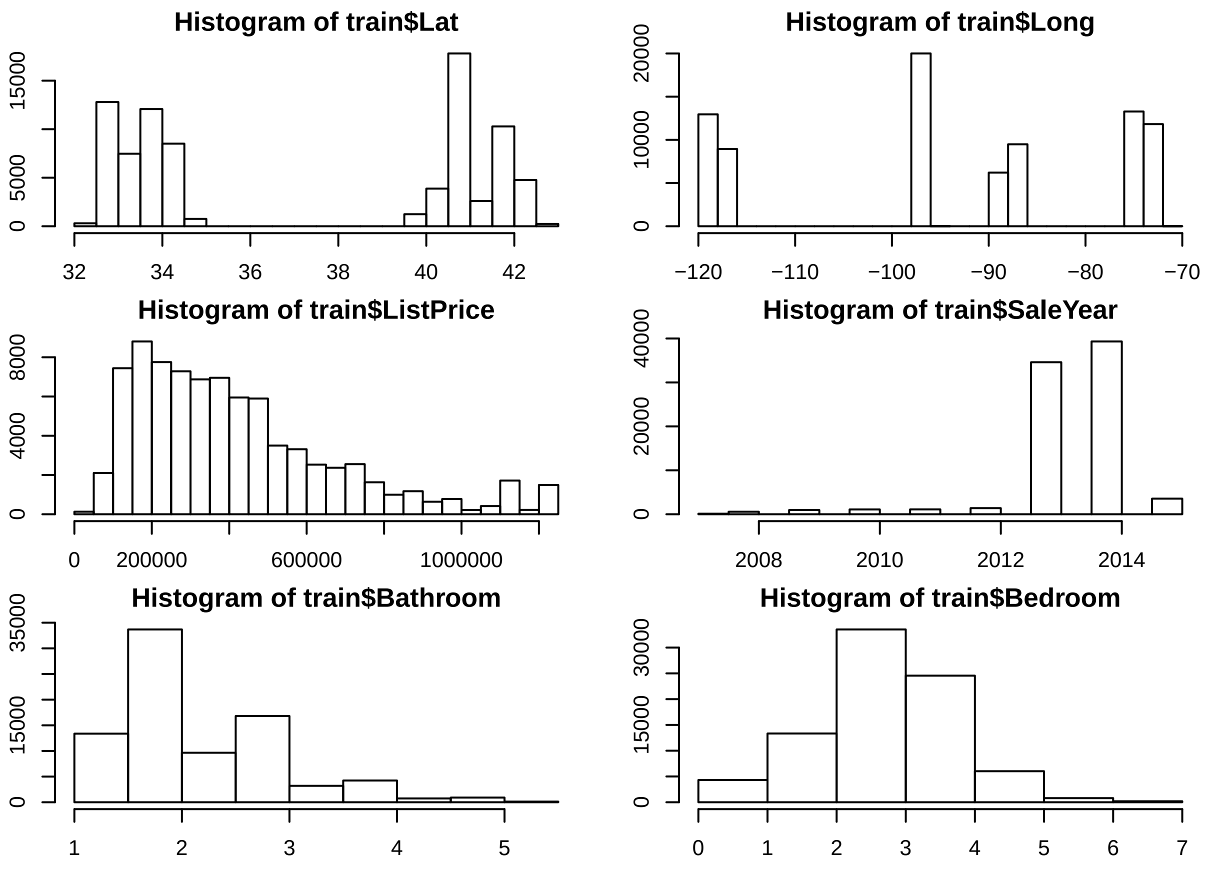
```
## [1] 82728    14
```

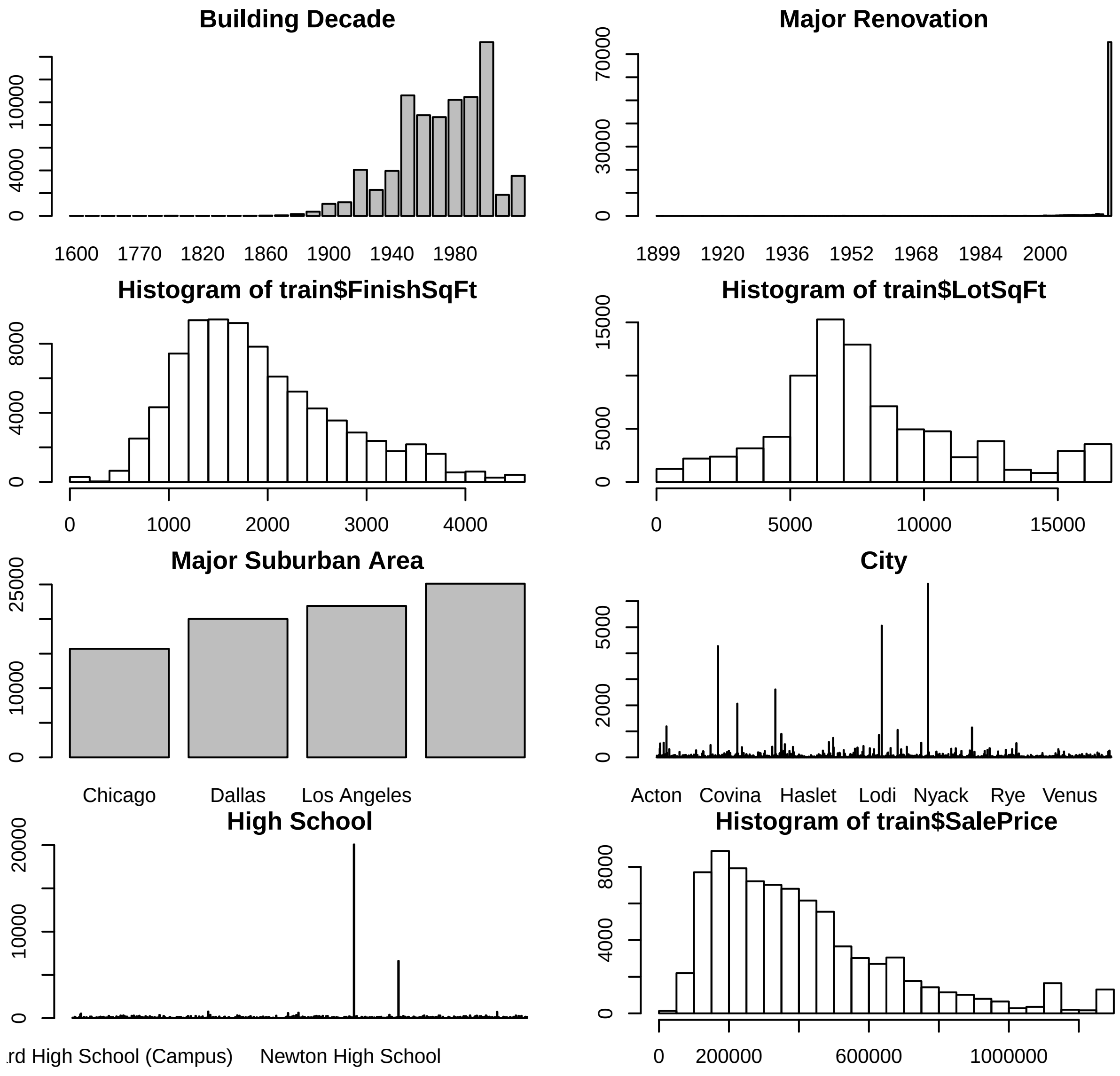
```
sum(is.na(train))
```

```
## [1] 0
```

```
summary(train)
```

```
##       Lat             Long           ListPrice          SaleYear
##  Min.   :32.18   Min.   :-118.85   Min.   :  15000   Min.   :2007
##  1st Qu.:33.52   1st Qu.:-117.66   1st Qu.: 219900   1st Qu.:2013
##  Median :34.58   Median : -96.47   Median : 360000   Median :2014
##  Mean   :37.25   Mean   : -93.87   Mean   : 423732   Mean   :2013
##  3rd Qu.:40.87   3rd Qu.: -74.32   3rd Qu.: 549000   3rd Qu.:2014
##  Max.   :42.67   Max.   : -71.92   Max.   :1247608   Max.   :2015
##
##     Bathroom        Bedroom       BuildDecade      MajorRenov
##  Min.   :1.000   Min.   :0.000   2000   :15287   NONE   :75180
##  1st Qu.:2.000   1st Qu.:3.000   1950   :10601   2013   :  842
##  Median :2.000   Median :3.000   1990   :10467   2014   :  679
##  Mean   :2.338   Mean   :3.206   1980   :10212   2012   :  491
##  3rd Qu.:3.000   3rd Qu.:4.000   1960   : 8857   2007   :  411
##  Max.   :5.500   Max.   :7.000   1970   : 8690   2010   :  397
##                                  (Other):18614   (Other): 4728
##    FinishSqFt      LotSqFt                  MSA                  City
##  Min.   :   0   Min.   :    0   Chicago      :15698   New York   : 6672
##  1st Qu.:1323   1st Qu.: 5750   Dallas       :20014   Los Angeles: 5067
##  Median :1768   Median : 7302   Los Angeles  :21900   Chicago    : 4277
##  Mean   :1923   Mean   : 7923   New York City:25116   Fort Worth : 2613
##  3rd Qu.:2400   3rd Qu.: 9583                         Dallas     : 2069
##  Max.   :4531   Max.   :16979                         Arlington  : 1194
##                                                       (Other)    :60836
##                                            HighSchool
##  NONE                                           :20082
##  Queens High School Of Teaching Liberal Arts And Sc: 6621
##  Garland High School                            :  772
##  Wells Community Academy High School            :  733
##  Lincoln Park High School                       :  640
##  Lake View High School                          :  594
##  (Other)                                        :53286
##    SalePrice
##  Min.   :  15000
##  1st Qu.: 214000
##  Median : 355000
##  Mean   : 414660
##  3rd Qu.: 535000
##  Max.   :1257245
##
```





Comments: The train data set has dimensions $n = 82,728$ and $p = 14$. There are no missing values. The `MajorRenov` variable has 75,000 "NONE" variables, meaning no renovation was done on many of these homes. The data is has not been processed to handle these. Thus this variable is treated as an ordinal categorical variable with an additional "NONE" level. Additionally, we have 3530 houses with an "UNKNOWN" value for `BuildDecade`.

Part b.

Some of the variables in the data are categorical; how many values do they take? Why might factor variables with many categories present a problem when fitting decision trees? Describe a couple different ways of handling factor variables when fitting decision trees.

Now we will use a few methods to predict `SalePrice`. Throughout, evaluate the predictions in terms of the absolute relative error:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i - \widehat{Y}_i|}{Y_i}$$

Explain why this is an appropriate choice of accuracy, compared with squared error.

Comments:

First, we see that our categorical variables and the number of levels they have are the follow: `BuildDecade`-29, `MajorRenov`-113, `MSA`-4, `City`-1291, `HighSchool`-655. If our categorical variables contain too many levels, then there will be a problem with too many splits in our decision (the tree can have a node for each level) and thus not enough data observations in each leaf to make accurate predictions. We are essentially overfitting over data. Thus, it may be best to ignore these variables (like `City` or `HighSchool`). For ordinal categorical variables, we can just treat them as numerical values (e.g. `BuildDecade`), but bucket them into fewer categories (e.g. pre1900 and post1900). For categorical variables with only a few levels (`MSA`), we can split them into dummy variables for each level.

Absolute relative error may the best predictor because then we are better accounting for the true sale price. Here, we penalize based on the percentage difference from the true sale price. If we were to use squared error than we are penalizing on the actual difference from the sale price. Imagine predicting one house's sale price for $\$1,050,000$ and the true sale price is $\$1,000,000$ and a second house for $\$100,000$ with a true sale price of $\$50,000$. Both differences are 50,000 and thus would contributed equally to squared error ($MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{Y_i} - Y_i)^2 = 50,000^2$). However, usuing absolute relative error, the error in the prediction of the second house is weighted more (since the error is a higher percentage of the true sale price). We can see,$ARE = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i - \widehat{Y_i}|}{Y_i} = \frac{1}{2} * (1 + 0.05) = 0.525$. Thus we can see that ARE weights moreheavily predictions that are higher percentage of the true sale price. This makes sense when because a person we will want to predict sale price with the closest percentage to the true price, not depending of the magnitude price difference.

Part c.

Build random forest models to predict `SalePrice` from the other covariates, using the R package `ranger`. The parameters to vary are `num.trees` and `min.node.size`; these regulate the variance and bias. Another parameter is `mtry`, which is the total number of variables allowed in splits; this regulates the correlation between the trees by introducing randomness. In addition, the `ranger` package has multiple options for how to handle factor variables; choose the one you think is best.

Train several random forest models, using different configurations of parameters. Evaluate each using 5-fold cross validation. Which setting of the parameters performs best? Comment on your findings and explain why they do or do not make sense to you.

```r
#First let's clean up our data.
#Convert MSA to binary indicator variables.
train$Chicago <- ifelse(train$MSA == "Chicago",1,0)
test$Chicago <- ifelse(test$MSA == "Chicago",1,0)
train$Dallas <- ifelse(train$MSA == "Dallas",1,0)
test$Dallas <- ifelse(test$MSA == "Dallas",1,0)
train$NYC <- ifelse(train$MSA == "New York City",1,0)
test$NYC <- ifelse(test$MSA == "New York City",1,0)
train$LA <- ifelse(train$MSA == "Los Angeles",1,0)
test$LA <- ifelse(test$MSA == "Los Angeles",1,0)
#There surely has to be a better way to do this dummy encoding, but I did not have time to figure it ou

#Let's encode our ordinal variables as well. We can just put building decade and major renovation year i
levels(train$MajorRenov)[1:46] <- "renov_pre1950"
levels(train$MajorRenov)[2:51] <- "renov_1950to1999"
levels(train$MajorRenov)[3:18] <- "renov_post2000"

levels(test$MajorRenov)[1:32] <- "renov_pre1950"
levels(test$MajorRenov)[2:51] <- "renov_1950to1999"
levels(test$MajorRenov)[3:18] <- "renov_post2000"
```

```
train$MajorRenov <- as.numeric(factor(train$MajorRenov,
                      levels=c("renov_pre1950","renov_1950to1999","renov_post2000","NONE")))
test$MajorRenov <- as.numeric(factor(test$MajorRenov,
                     levels=c("renov_pre1950","renov_1950to1999","renov_post2000","NONE")))

levels(train$BuildDecade)[1:16] <- "build_pre1900"
levels(train$BuildDecade)[2:7] <- "build_1900to1949"
levels(train$BuildDecade)[3:8] <- "build_post1950"

levels(test$BuildDecade)[1:12] <- "build_pre1900"
levels(test$BuildDecade)[2:6] <- "build_1900to1949"
levels(test$BuildDecade)[3:9] <- "build_post1950"

train$BuildDecade <- as.numeric(factor(train$BuildDecade,
                       levels=c("build_pre1900","build_1900to1949","build_post1950","UNKNOWN")))
test$BuildDecade <- as.numeric(factor(test$BuildDecade,
                       levels=c("build_pre1900","build_1900to1949","build_post1950","UNKNOWN")))

#Now we can remove some categorical variables (city, high school). Also eliminate variables that we have
col <- which(names(train) %in% c("City","HighSchool","MSA"))
train <- train[-col]
test <- test[-col]
```

```
#Let's build a function to perform the cross validation for us.
crossval <- function(traindata,K,numtrees,nodesize,m) {
  kfolds <- createFolds(c(1:nrow(traindata)), k = 5, list = TRUE, returnTrain = FALSE)
  errors <- rep(0,K)
  for (i in 1:K) {
    mod <- ranger(SalePrice~., data=traindata[-kfolds[[i]],], num.trees = numtrees,
                  min.node.size = nodesize, mtry=m,respect.unordered.factors = F)
    pred <- predict(mod,data=traindata[kfolds[[i]],-11])
    cverror <- mean(abs(traindata[kfolds[[i]],11] - pred$predictions)/traindata[kfolds[[i]],11])
    errors[i] <- cverror
  }
  print(mean(errors))
}
```

```
set.seed(1)
#Let's first adjust the number of trees. We can keep the other parameters constant.
for (j in c(10,100,200)) {
  crossval(train,5,j,1000,5)
}
```

```
## [1] 0.06931002
## [1] 0.06220653
## [1] 0.06093412
```

```
set.seed(1)
#Now let's look at minimum node size.
for (j in c(100,1000,5000)) {
  crossval(train,5,100,j,5)
}
```

```
## [1] 0.0315337
## [1] 0.06220653
## [1] 0.1255329
```

```
set.seed(1)
#Finally, let's look at the number of variables we use when splitting our random forest (m).
for (j in c(2,5,14)) {
  crossval(train,5,100,500,j)
}
```

```
## [1] 0.1577624
## [1] 0.04871944
## [1] 0.02349211
```

```
#Now let's fit a model on the entire train set and predict on the test.
rfmod <- ranger(SalePrice~., data=train, num.trees = 200, min.node.size = 100, mtry=14,
                respect.unordered.factors = F)
```

```
## Growing trees.. Progress: 35%. Estimated remaining time: 1 minute, 0 seconds.
## Growing trees.. Progress: 69%. Estimated remaining time: 28 seconds.
```

```
pred <- predict(rfmod,data=test[-11])
testerror <- mean(abs(test[,11] - pred$predictions)/test[,11])
testerror
```

```
## [1] 0.02250613
```

Comments: First, I cleaned up the data to remove the categorical variables with too many levels and that were not ordinal (`City` and `HighSchool`). It is hard to predict housing prices depending on high school and city, because there will not be very many observations for each level. Then I converted the `MSA` variable into a set of dummy indicators significying which major suburban area the house is in (NYC, Los Angeles, Chicago, Dallas). This was we compare whether they are in each area individually (not the levels grouped together in a traditional categorical variable).

Then, I built a function that does cross validation and gives out the average error for the random forest model. The `crossval` function takes the train data set, number of folds (K), number of trees, minimum node size, and m value for the number of variables considered when splitting the tree all as input parameters. I then tested 3 different parameter values for each number of trees, min node size, and m to observe how the cross validation error rate changes. For number of trees, the more trees we grow, the lower the error rate is. This makes sense because we are then averaging more and more trees when we make our prediction, reducing the prediction power of each individual tree. For smaller minimum node size, we see a decrease in error rate. This makes sense because we are reducing bias when we grow a deeper tree (think k-NN with small k). Even though variance is increasing for each tree, we control variance by averaging these deep trees through the random forest algorithm. Finally, as m increases, we see our error decrease. This makes sense because we have the ability to split our tree on more varaibles, thus minimizing the error at each split.

When I do my final random forest model, I want to use the best paramters, but also try to save on computation effort. Fitting deeper trees or more numbers of trees, will both decrease the error, but will end up increasing computation time drastically (e.g. 1000 trees, 1 minimum node size will grow lots of trees very deep). I used num trees = 200, minimum node size = 100, and m = 14.

Part d.

Now build models using gradient tree boosting, using the R package `xgboost`. The parameters to vary are `max.depth`, `eta`, and `nrounds`, which regulate the maximum depth of each tree, the step size, and the number of trees in the final model. Try several different runs of gradient boosting, each time using a different configuration of the parameters. As above, evaluate each using 5-fold cross validation. Which setting of the parameters performs best? Comment on your findings and explain why they do or do not make sense to you.

```
#Let's define another cross validation function for gradient boosting.
crossval_gb <- function(traindata,K,stepsize,depth,rounds) {
  kfolds <- createFolds(c(1:nrow(traindata)), k = K, list = TRUE, returnTrain = FALSE)
  errors <- rep(0,K)
  for (i in 1:K) {
    mod <- xgboost(data=data.matrix(traindata[kfolds[[i]],-11]),label=traindata[kfolds[[i]],11],
                   eta=stepsize, max.depth=depth, nrounds=rounds, verbose=F)
    pred <- predict(mod,newdata=data.matrix(traindata[kfolds[[i]],-11]))
    cverror <- mean(abs(traindata[kfolds[[i]],11] - pred)/traindata[kfolds[[i]],11])
    errors[i] <- cverror
  }
  print(mean(errors))
}
```

```
set.seed(1)
#Look through some of the stepsize (eta) parameters.
for (j in c(.01,.1,1,2)) {
  crossval_gb(train,5,j,20,10)
}
```

```
## [1] 0.9043893
## [1] 0.348462
## [1] 0.002721851
## [1] 0.9634917
```

```
set.seed(1)
#Now look through max depth of tree (similar to minimum node size above, except the opposite).
for (j in c(1,10,50)) {
  crossval_gb(train,5,1,j,10)
}
```

```
## [1] 0.128151
## [1] 0.01642383
## [1] 1.943131e-05
```

```
set.seed(1)
#Finally, look at nrounds. Not exactly sure what this is? Slightly similar to number of trees?
for (j in c(10,50,100)) {
  crossval_gb(train,5,1,10,j)
}
```

```
## [1] 0.01712745
## [1] 0.006120602
## [1] 0.001858727
```

```
gbmod <- xgboost(data=data.matrix(train[,-11]), label=train[,11],
                 eta=1, max.depth=50, nrounds=100, verbose=F)

pred <- predict(gbmod,newdata=data.matrix(test[,-11]))
testerror <- mean(abs(test[,11] - pred)/test[,11])
testerror
```

```
## [1] 0.02979151
```

Comments: It is interesting to see that first as η increases towards 1 the error decreases quickly. However, once η goes above one, the error rate begin to increase again. This makes sense because we being to weight the model on the residuals more than the model on the actual data itself. I am not 100% sure why I get such a low error $\eta = 1$, this seems too long to be correct, but I do not see an issue in my code or how I am calculating the error.

For `max.depth`, as we increase tree depth the error decreases. This is the opposite of minimum node size for the random forest models in part 3c. The decrease in bias from growing a deep tree outweighs the increase in varaince. Again, I see an extremely low error rate when the tree depth increases, which seems slightly incorrect, but again not sure where in my code I would be miscalculating the error rate.

For `nrounds`, as the number of rounds increases the error decreases. This is similar to increase the number of trees in a random forest model, we are using more decision trees in our final model.

I chose the parameters that would work best with the trends identified below while still trying to not do too much computation (e.g. `max.depth`=1000 or `max.depth`=1000).

Part e.

Using your best model, predict the sale prices for the houses given in the test data. Report your error rate (again, the absolute relative error). In addition, show plots that compare the performance of boosting and random forests, as the number of component trees is varied.

```
#The random forest model is the best model given the array of parameters I tested.
rfmod <- ranger(SalePrice~., data=train, num.trees = 200, min.node.size = 100, mtry=14,
                respect.unordered.factors = F)
```

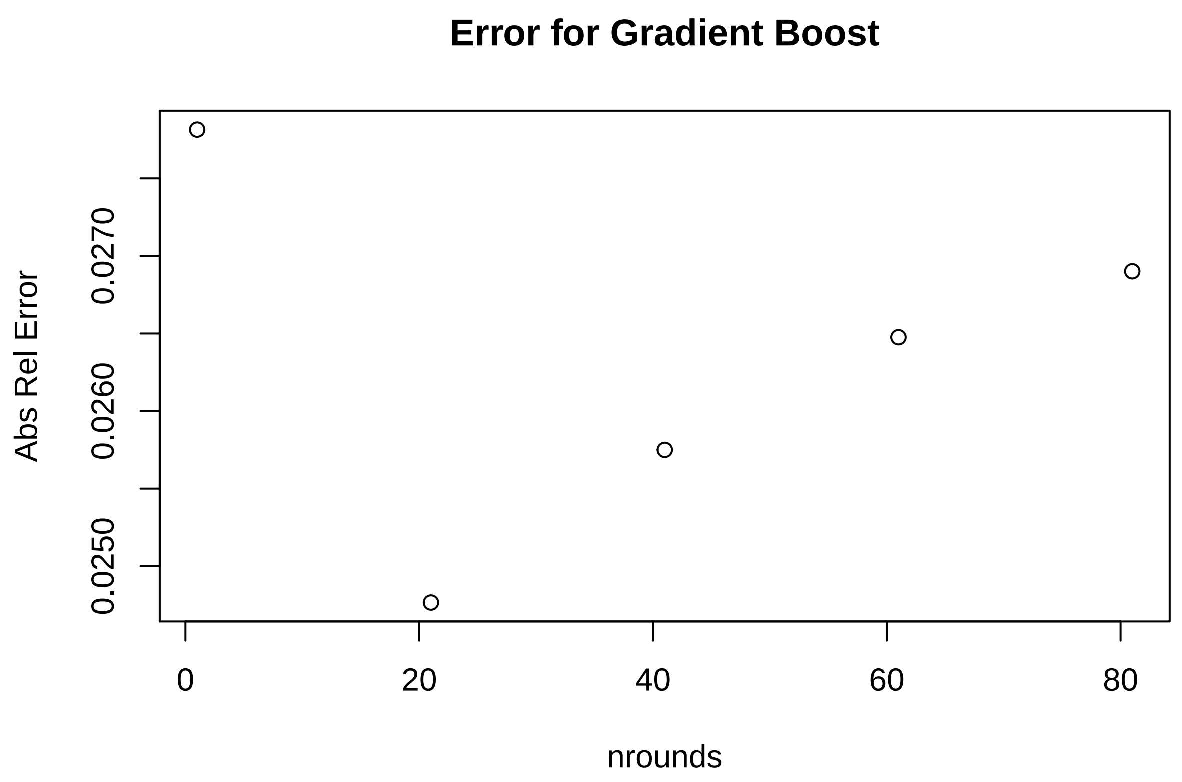
```
## Growing trees.. Progress: 35%. Estimated remaining time: 58 seconds.
## Growing trees.. Progress: 71%. Estimated remaining time: 26 seconds.
```

```
pred <- predict(rfmod,data=test[-11])
testerror <- mean(abs(test[,11] - pred$predictions)/test[,11])
testerror
```

```
## [1] 0.02250788
```

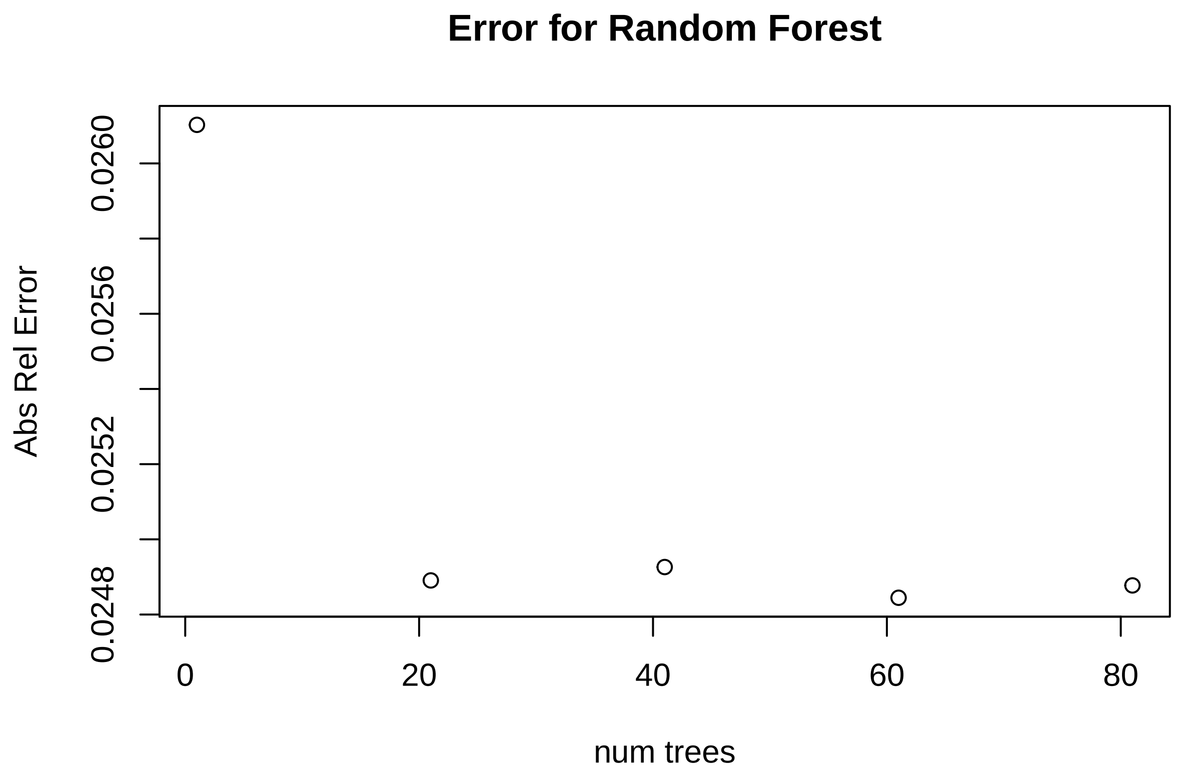
```
set.seed(1)
sequence <- seq(1,100,20)
gb_errors <- rep(0,length(sequence))
for (n in 1:length(sequence)) {
  gbmod <- xgboost(data=data.matrix(train[,-11]), label=train[,11],
                 eta=1, max.depth=10, nrounds=sequence[n], verbose=F)

  pred <- predict(gbmod,newdata=data.matrix(test[,-11]))
  testerror <- mean(abs(test[,11] - pred)/test[,11])
  gb_errors[n] <- testerror
}
plot(sequence,gb_errors,main="Error for Gradient Boost",ylab="Abs Rel Error",xlab="nrounds")
```



```
rf_errors <- rep(0,length(sequence))
for (n in 1:length(sequence)) {
  rfmod <- ranger(SalePrice~., data=train, num.trees = sequence[n], min.node.size = 1000, mtry=14,
                  respect.unordered.factors = F)

  pred <- predict(rfmod,data=test[-11])
  testerror <- mean(abs(test[,11] - pred$predictions)/test[,11])
  rf_errors[n] <- testerror
}
plot(sequence,rf_errors,main="Error for Random Forest",ylab="Abs Rel Error",xlab="num trees")
```



Comments: I am confused by the results of the error as a factor of `nrounds` for gradient boosting. In problem 3d, I see that error rate decrease with an increase in nrounds. However, here in the graph we get an initial decrease in error rate, but it then increases. I believe conceptually that what I see with the error rate in 3d is correct. That as nrounds increases, the error rate decreases. With the random forest model, we see that the error rate does decrease as expected with the increase in the number of trees grown.