STT_481_Final

```r
rm(list = ls())
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.0.4

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(FNN)
```

```
## Warning: package 'FNN' was built under R version 4.0.4
```

```r
library(class)
```

```
##
## Attaching package: 'class'

## The following objects are masked from 'package:FNN':
##
##     knn, knn.cv
```

```r
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.0.3

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(leaps)
```

```
## Warning: package 'leaps' was built under R version 4.0.4
```

```r
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.0.4

## Loading required package: Matrix

## Loaded glmnet 4.1-1
```

```r
library(gam)
```

```
## Warning: package 'gam' was built under R version 4.0.4

## Loading required package: splines

## Loading required package: foreach

## Warning: package 'foreach' was built under R version 4.0.3

## Loaded gam 1.20
```

```r
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.0.5
```

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.0.5

## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.0.5

## Loaded gbm 2.1.8
```

```r
train <- read.csv("C:/Users/jacob/Downloads/train_new.csv")
test <- read.csv("C:/Users/jacob/Downloads/test_new.csv")
```

```r
train_x <- train %>% dplyr::select(-SalePrice)
train_y <- train %>% dplyr::select(SalePrice)
test_x <- test %>% dplyr::select(-SalePrice)
test_y <- test %>% dplyr::select(SalePrice)
```

This data is from Ames Housing dataset compiled by Dean De Cock. The data includes variables. The variables and their descriptions are below:

LotArea - Lot size in square feet OverallQual - Overall material and finish quality OverallCond - Overall condition rating YearBuilt - Original construction date YearRemodAdd - Remodel date BsmtFinSF1 - Type 1 finished square feet BsmtFinSF2 - Type 2 finished square feet BsmtUnfSF - Unfinished square feet of basement area X1stFlrSF - First floor square feet X2ndFlrSF - Second floor square feet LowQualFinSF - Low quality finished square feet (all floors) BsmtFullBath - Basement full bathrooms BsmtHalfBath - Basement half bathrooms FullBath - Full bathrooms above grade HalfBath - Half baths above grade BedroomAbvGr - Number of bedrooms above basement level TotRmsAbvGrd - Total rooms above grade (does not include bathrooms) Fireplaces - Number of fireplaces GarageCars - Size of garage in car capacity KitchenAbvGr - Number of kitchens TotRmsAbvGrd - Total rooms above grade (does not include bathrooms) Fireplaces - Number of fireplaces GarageCars - Size of garage in car capacity WoodDeckSF - Wood deck area in square feet MoSold - Month Sold

BsmtHalfBath, BsmtFullBath, FullBath, KitchenAbvGr, Fireplaces, and HalfBath are treated as qualitative variables instead of quantitative due to their low variety in unique values.

The problem that we are trying to uncover is how we can accurately predict the sale price of a house based on the given data. To explore this, we will use many different techniques: KNN, linear regression, subset linear regression, ridge regression, lasso regression, GAM, regression tree, bagging, random forest, and boosting models.

The data was pre-cleaned, dropping the NA values and the skewed variables.

KNN

```r
K.vt <-c(1,5,10,15,20,25,30,35,40,45,50)
error.k <-rep(0,length(K.vt))
counter <- 0
for(k in K.vt){
  counter <- counter+1
  error <- 0
  for(i in 1:nrow(train_x)){
    pred.class <- knn.reg(train_x[-i,], train_x[i,], train_y$SalePrice[-i], k=k)
    error <- error+ (train_y$SalePrice[i]-pred.class$pred)^2
  }
error.k[counter] <- error/nrow(train_x)
}
print(error.k)
```

```
##  [1] 2843179080 2344538675 2407759479 2572463698 2684498388 2838581446
##  [7] 2957657276 3052772570 3129853746 3203398426 3275446519
```

We run a cross validation algorithm to find the best K for our model. We see here that K=5 has the lowest MSE. This means that this is the best K.

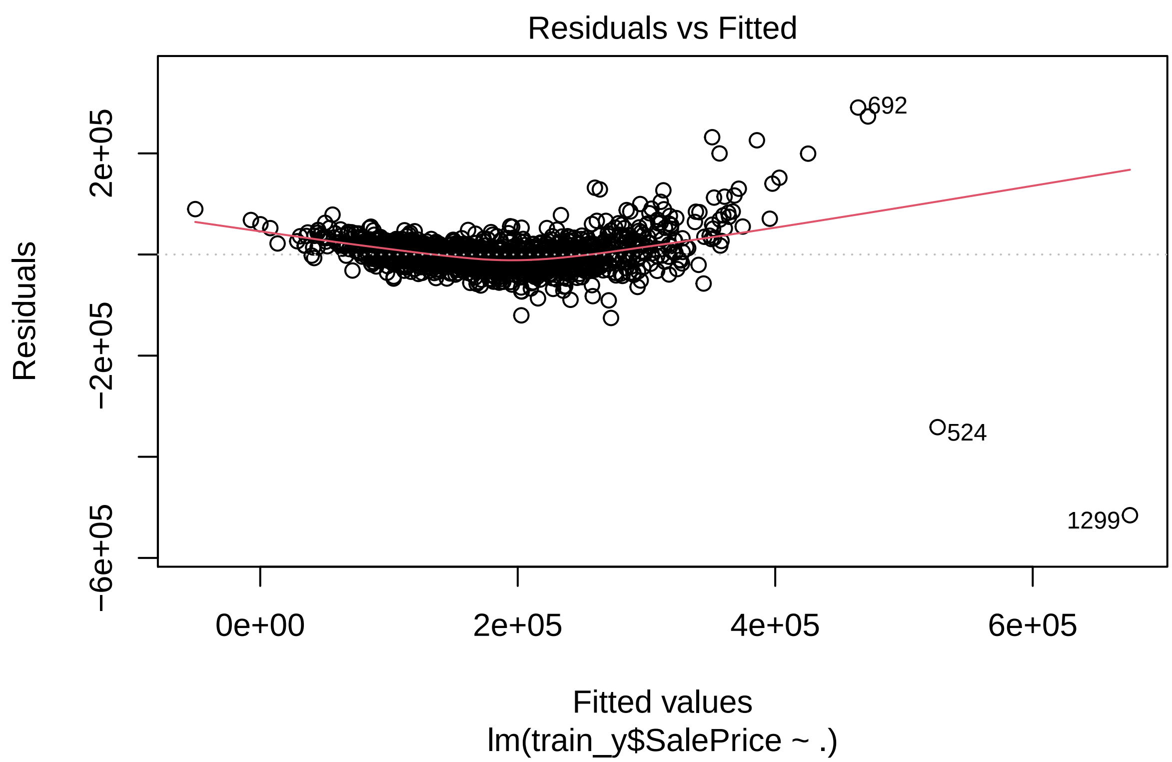
```r
head(knn.reg(train=train_x,test=test_x,y=train_y,k=5)$pred)
```

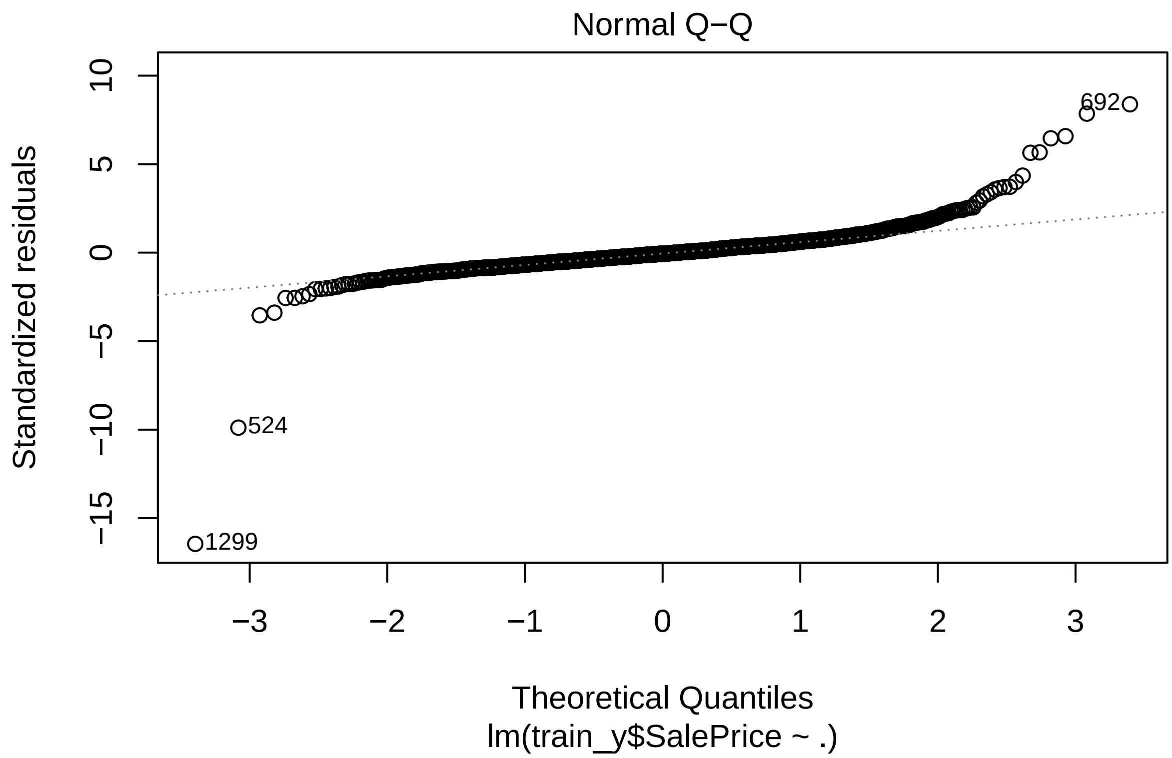
```
## [1] 142460 176560 172560 208358 129376 175335
```

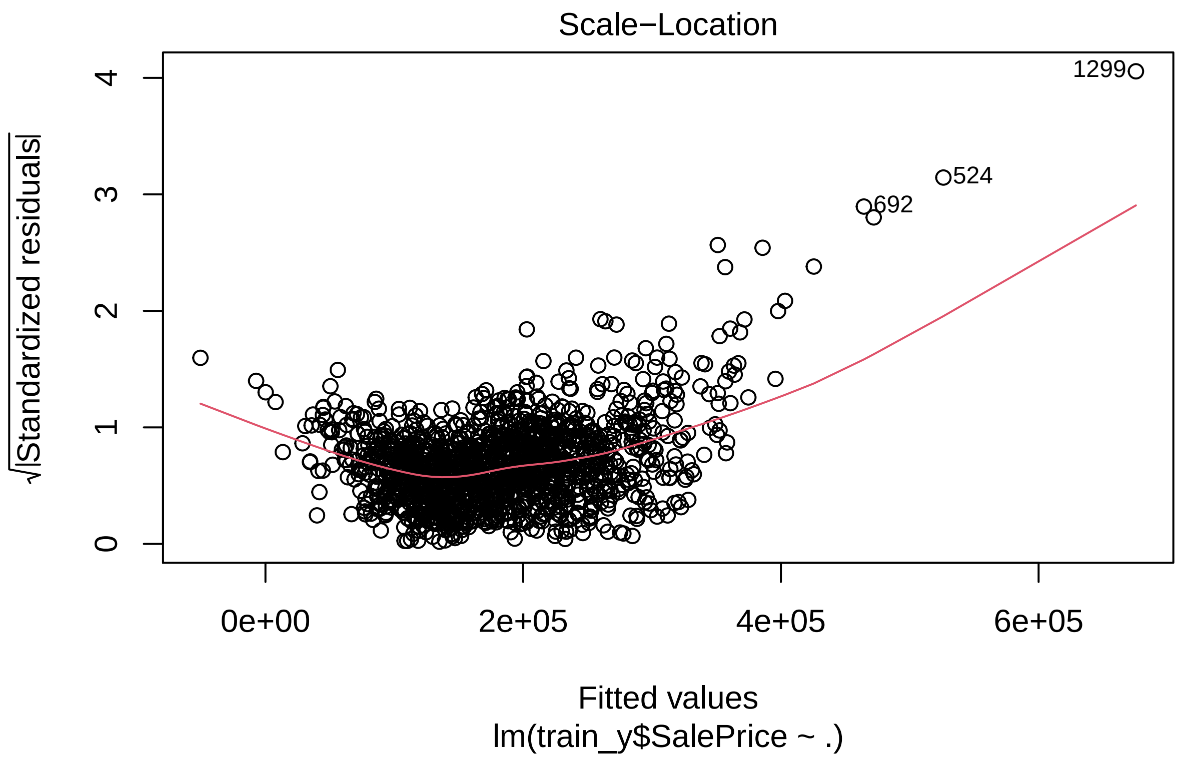
These are our final predictions from our final model with our optimized k value of 5.

Linear Regression

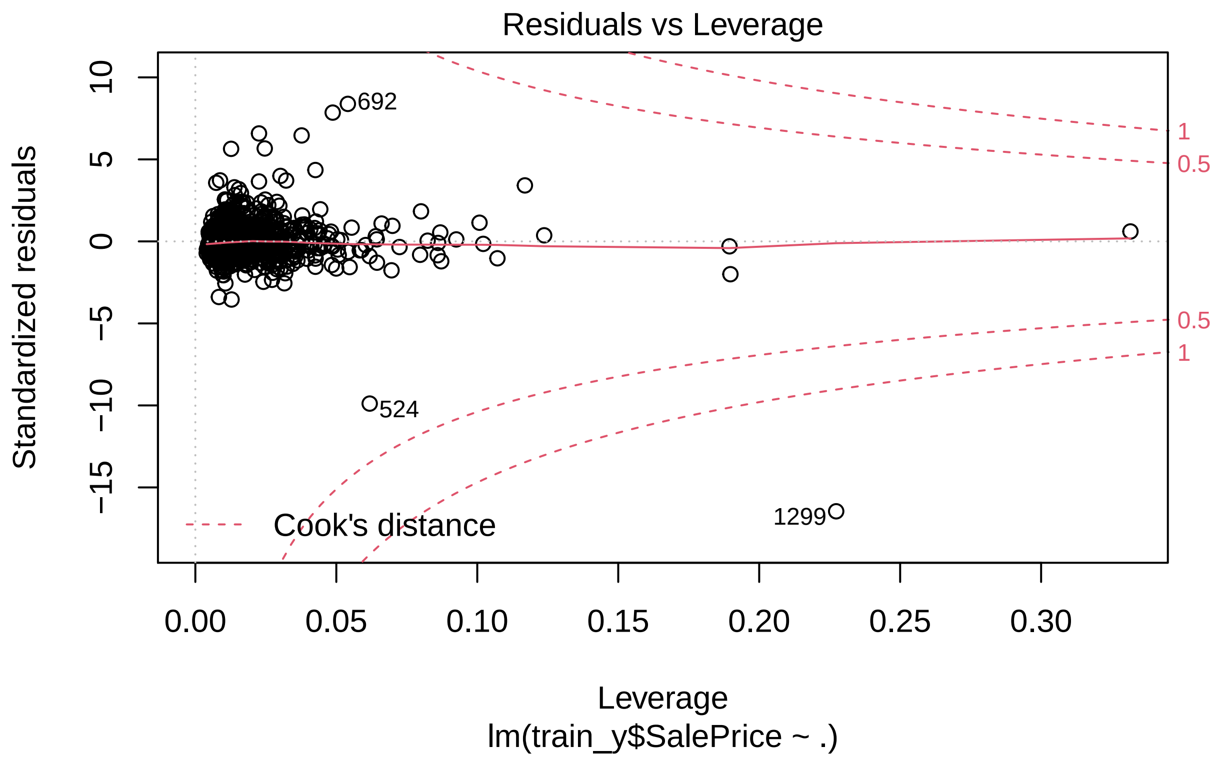
```r
linfit <- lm(train_y$SalePrice~.,data=train_x)
plot(linfit)
```




Normal Q−Q
Standardized residuals
10
5
0
−5
−10
−15
692
524
1299
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(train_y$SalePrice ~ .)



Scale−Location
√|Standardized residuals|
0
1
2
3
4
0e+00
2e+05
4e+05
6e+05
Fitted values
lm(train_y$SalePrice ~ .)
1299
524
692



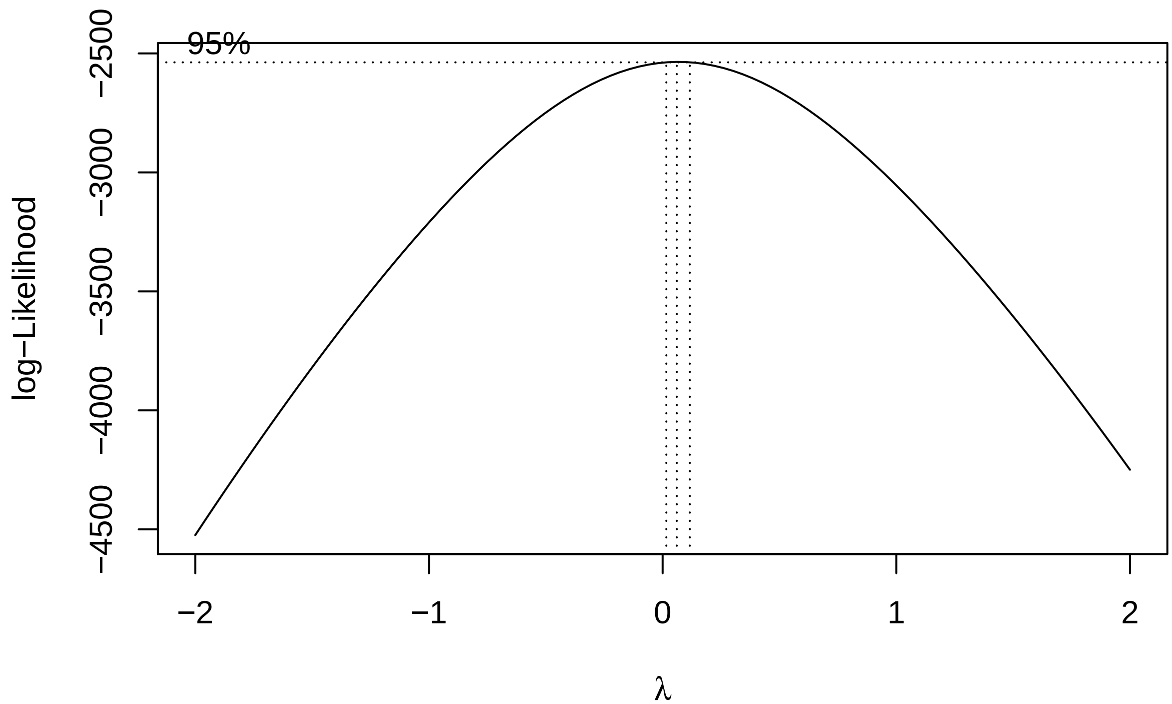
Next we run a linear regression algorithm. This algorithm utilizes the least squares method, meaning it draws a linear line through the data at which the residual sum of squares is minimized.

We see that the residuals are not normally distributed and are not linear due to some large presumed outliers. We also have one data point (1200) that falls outside of our Cook's distance of 1, meaning this point is a high leverage point.

For our outliers and our high leverage point, there is no obvious remedy as we do not have enough information to remove these from our model. To remove these, we would need evidence that it was a data collection error.

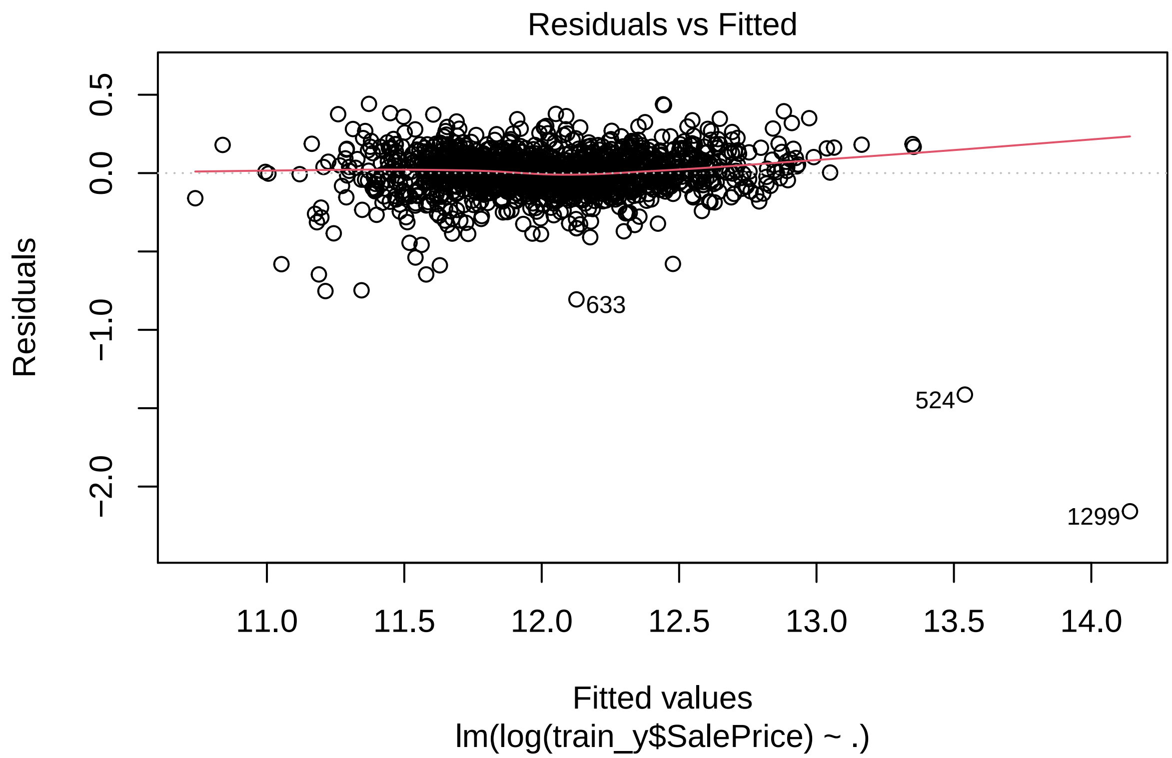
For our nonlinearity and normality issues, we can transform our Y variable to see if it improves our residuals. We can run a boxcox on our model to identify the correct strategy.

```
boxcox(SalePrice~.,data=train)
```

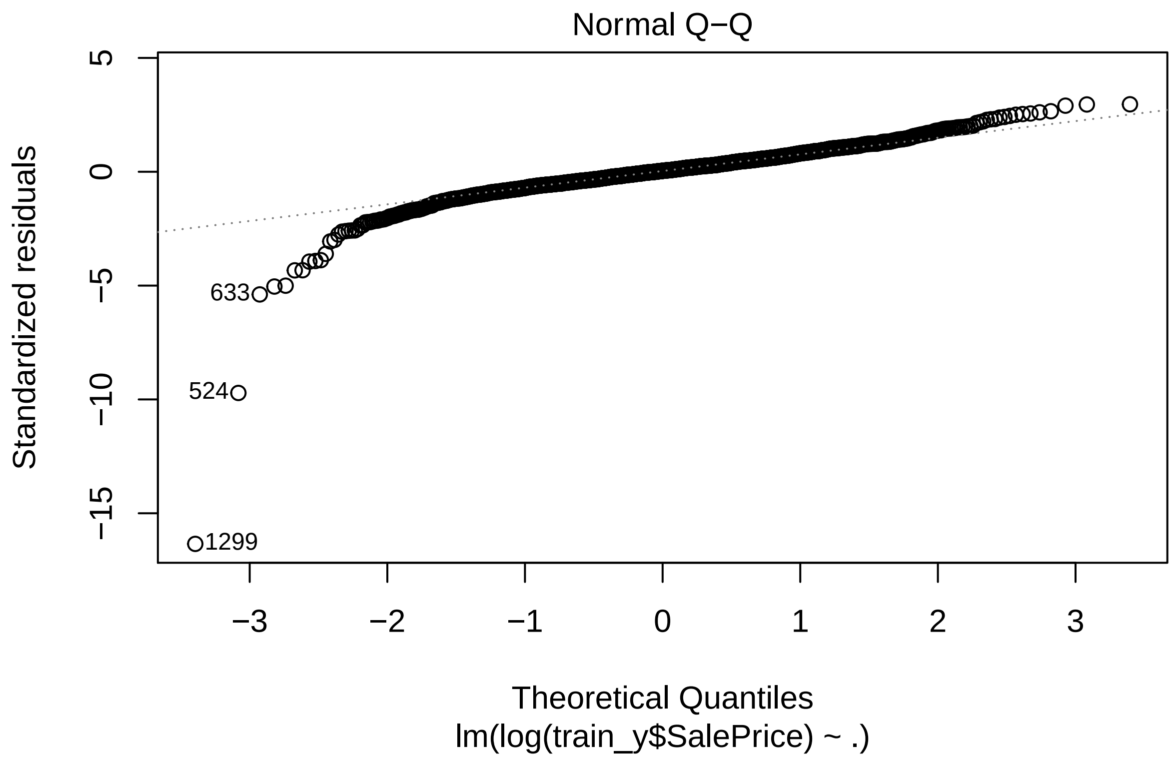


With a lambda of 0, it shows our best transformation would by log(Y).

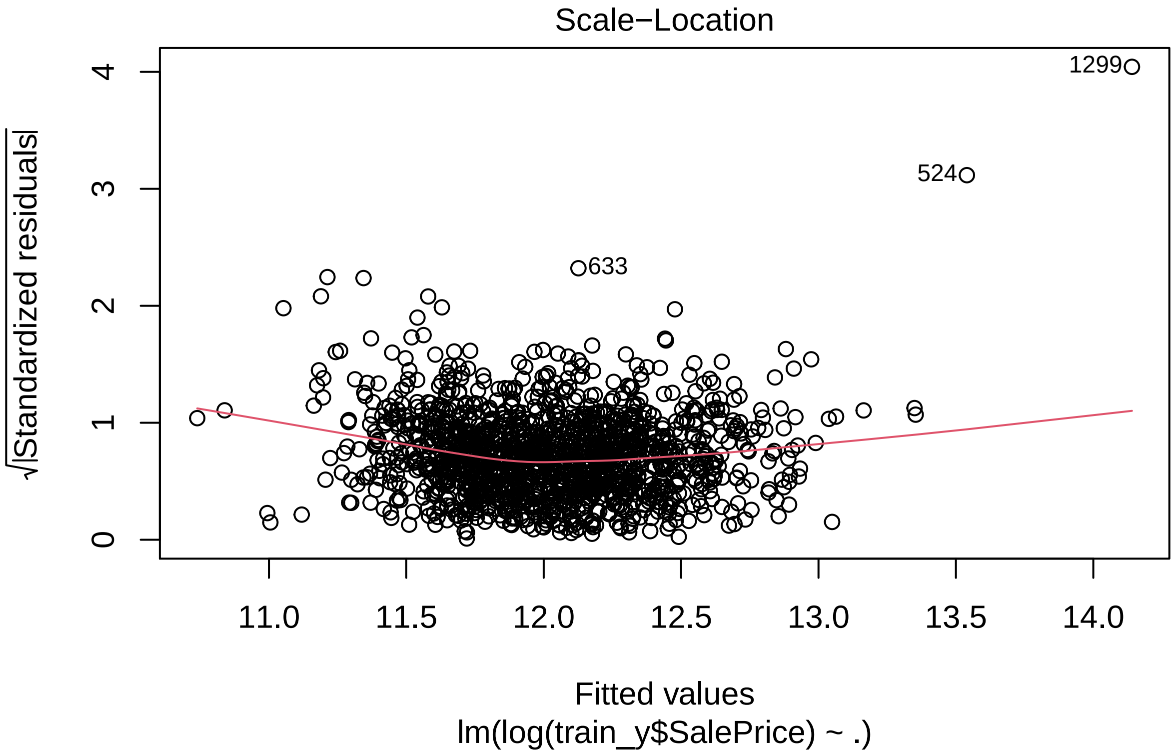
```
fit2 <- lm(log(train_y$SalePrice)~.,data=train_x)
plot(fit2)
```

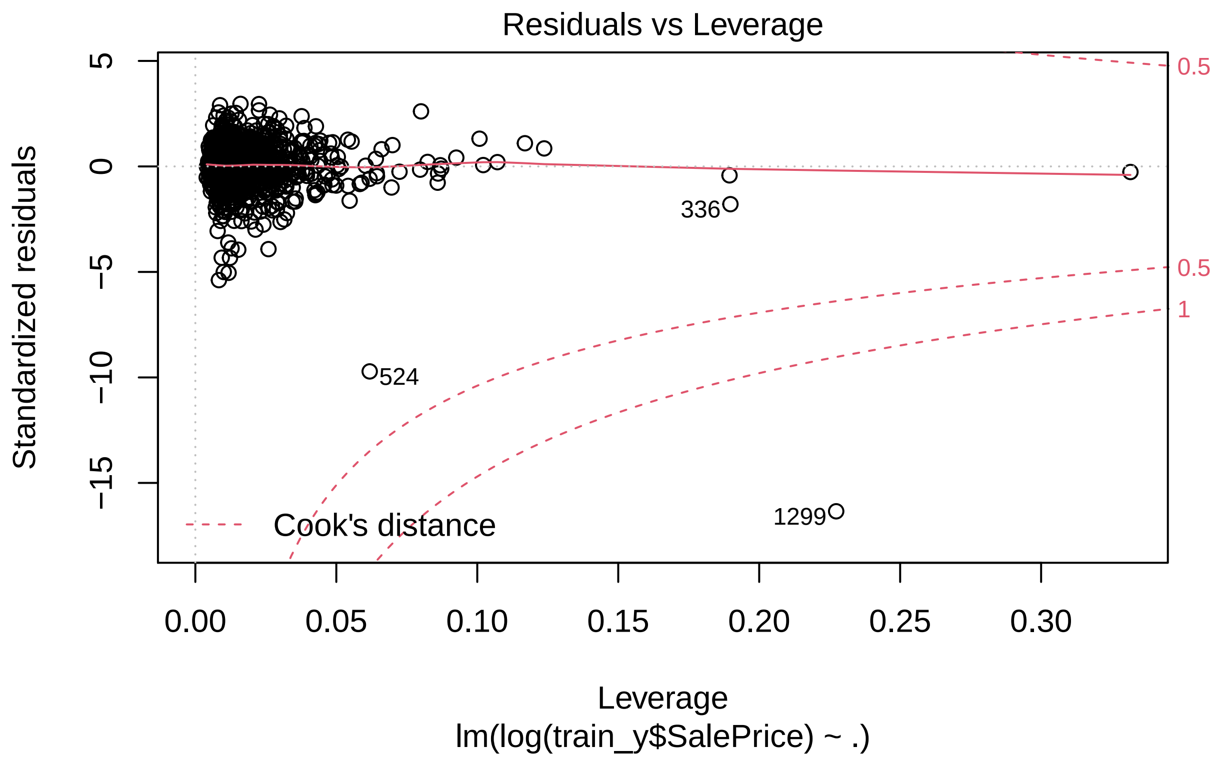


Residuals vs Fitted
Residuals
0.5
0.0
−1.0
−2.0
633
524
1299
11.0
11.5
12.0
12.5
13.0
13.5
14.0
Fitted values
lm(log(train_y$SalePrice) ~ .)



Normal Q−Q
Standardized residuals
5
0
−5
−10
−15
633
524
1299
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(log(train_y$SalePrice) ~ .)


Scale−Location
√|Standardized residuals|
0
1
2
3
4
1299
524
633
11.0
11.5
12.0
12.5
13.0
13.5
14.0
Fitted values
lm(log(train_y$SalePrice) ~ .)



With this transformation, our model has more linearity in the residuals, and our normality improved at higher values. Unfortunately, the normality issue remains for lower values.

```
summary(fit2)
```

```
##
## Call:
## lm(formula = log(train_y$SalePrice) ~ ., data = train_x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.15777 -0.06867  0.00593  0.07854  0.44156
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.422e+00  5.601e-01   6.109 1.29e-09 ***
## LotArea       1.982e-06  4.323e-07   4.585 4.93e-06 ***
## OverallQual   8.314e-02  5.054e-03  16.450  < 2e-16 ***
## OverallCond   4.970e-02  4.369e-03  11.378  < 2e-16 ***
## YearBuilt     2.571e-03  2.462e-04  10.442  < 2e-16 ***
## YearRemodAdd  1.041e-03  2.826e-04   3.684 0.000238 ***
## BsmtFinSF1    8.190e-05  1.990e-05   4.115 4.08e-05 ***
## BsmtFinSF2    7.935e-05  3.025e-05   2.623 0.008796 **
## BsmtUnfSF     6.292e-05  1.796e-05   3.504 0.000473 ***
## X1stFlrSF     1.970e-04  2.460e-05   8.009 2.38e-15 ***
## X2ndFlrSF     1.370e-04  2.051e-05   6.680 3.41e-11 ***
```

```
## LowQualFinSF  1.126e-04  8.450e-05   1.332 0.183030
## BsmtFullBath  5.864e-02  1.116e-02   5.255 1.70e-07 ***
## BsmtHalfBath  1.763e-02  1.759e-02   1.002 0.316302
## FullBath      3.697e-02  1.207e-02   3.063 0.002233 **
## HalfBath      2.658e-02  1.142e-02   2.328 0.020028 *
## BedroomAbvGr  3.376e-03  7.233e-03   0.467 0.640735
## KitchenAbvGr -1.029e-01  2.088e-02  -4.929 9.21e-07 ***
## TotRmsAbvGrd  1.891e-02  5.276e-03   3.584 0.000349 ***
## Fireplaces    4.925e-02  7.561e-03   6.514 1.01e-10 ***
## GarageCars    7.172e-02  1.226e-02   5.851 6.03e-09 ***
## GarageArea    3.569e-05  4.155e-05   0.859 0.390554
## WoodDeckSF    8.717e-05  3.386e-05   2.574 0.010142 *
## MoSold        1.411e-03  1.466e-03   0.962 0.336027
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1502 on 1436 degrees of freedom
## Multiple R-squared:  0.8609, Adjusted R-squared:  0.8586
## F-statistic: 386.3 on 23 and 1436 DF,  p-value: < 2.2e-16
```

Of our significant variables, we see that our most significant include OverallQual, OverallCond, and YearBuilt with an approximately 0 p-value. Our coefficients for these variables can tell us how each variable affects our response term. For example, if YearBuilt incremented by 1 (meaning it was newer by 1 year), we would see a change in Y of .002571, assuming all other variables remained constant. This makes sense, because people generally pay more for newer houses. This same logic can be applied to all our significant variables.

```r
head(exp(predict(fit2,newdata=test)))
```

```
##        1        2        3        4        5        6
## 113397.3 146077.0 168281.6 196719.1 182239.9 175746.7
```

These are our new predictions. We run the exp() function because our fitted values from our model are log transformed.

Subset Selection

This algorithm is similar to the linear regression method, but it narrows down the full model to choose only the most significant variables. This leads to a model that is simplified and minimizes the Cp value.

I will choose the backward stepwise subset selection. I chose this method in particular through a process of elimation. With 24 predictors, this means there can be a list of 2^24 different possible combinations of predictors if I chose a "best subset" method. This may be too computationally expensive for practical use. The other two methods, forward and backward, may not guarantee to give us the best model, but it will lower the chance of overfitting and remain computationally inexpensive.

```r
cor(train_x) >=.7
```

```
##              LotArea OverallQual OverallCond YearBuilt YearRemodAdd BsmtFinSF1
## LotArea         TRUE       FALSE       FALSE     FALSE        FALSE      FALSE
## OverallQual    FALSE        TRUE       FALSE     FALSE        FALSE      FALSE
## OverallCond    FALSE       FALSE        TRUE     FALSE        FALSE      FALSE
## YearBuilt      FALSE       FALSE       FALSE      TRUE        FALSE      FALSE
## YearRemodAdd   FALSE       FALSE       FALSE     FALSE         TRUE      FALSE
## BsmtFinSF1     FALSE       FALSE       FALSE     FALSE        FALSE       TRUE
```

```
## BsmtFinSF2     FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## BsmtUnfSF      FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## X1stFlrSF      FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## X2ndFlrSF      FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## LowQualFinSF   FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## BsmtFullBath   FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## BsmtHalfBath   FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## FullBath       FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## HalfBath       FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## BedroomAbvGr   FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## KitchenAbvGr   FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## TotRmsAbvGrd   FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## Fireplaces     FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## GarageCars     FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## GarageArea     FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## WoodDeckSF     FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
## MoSold         FALSE       FALSE       FALSE     FALSE        FALSE      FALSE
##              BsmtFinSF2 BsmtUnfSF X1stFlrSF X2ndFlrSF LowQualFinSF BsmtFullBath
## LotArea           FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## OverallQual       FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## OverallCond       FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## YearBuilt         FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## YearRemodAdd      FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## BsmtFinSF1        FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## BsmtFinSF2         TRUE     FALSE     FALSE     FALSE        FALSE        FALSE
## BsmtUnfSF         FALSE      TRUE     FALSE     FALSE        FALSE        FALSE
## X1stFlrSF         FALSE     FALSE      TRUE     FALSE        FALSE        FALSE
## X2ndFlrSF         FALSE     FALSE     FALSE      TRUE        FALSE        FALSE
## LowQualFinSF      FALSE     FALSE     FALSE     FALSE         TRUE        FALSE
## BsmtFullBath      FALSE     FALSE     FALSE     FALSE        FALSE         TRUE
## BsmtHalfBath      FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## FullBath          FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## HalfBath          FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## BedroomAbvGr      FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## KitchenAbvGr      FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## TotRmsAbvGrd      FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## Fireplaces        FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## GarageCars        FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## GarageArea        FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## WoodDeckSF        FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
## MoSold            FALSE     FALSE     FALSE     FALSE        FALSE        FALSE
##              BsmtHalfBath FullBath HalfBath BedroomAbvGr KitchenAbvGr
## LotArea             FALSE    FALSE    FALSE        FALSE        FALSE
## OverallQual         FALSE    FALSE    FALSE        FALSE        FALSE
## OverallCond         FALSE    FALSE    FALSE        FALSE        FALSE
## YearBuilt           FALSE    FALSE    FALSE        FALSE        FALSE
## YearRemodAdd        FALSE    FALSE    FALSE        FALSE        FALSE
## BsmtFinSF1          FALSE    FALSE    FALSE        FALSE        FALSE
## BsmtFinSF2          FALSE    FALSE    FALSE        FALSE        FALSE
## BsmtUnfSF           FALSE    FALSE    FALSE        FALSE        FALSE
## X1stFlrSF           FALSE    FALSE    FALSE        FALSE        FALSE
## X2ndFlrSF           FALSE    FALSE    FALSE        FALSE        FALSE
## LowQualFinSF        FALSE    FALSE    FALSE        FALSE        FALSE
## BsmtFullBath        FALSE    FALSE    FALSE        FALSE        FALSE
```

```
## BsmtHalfBath         TRUE    FALSE    FALSE        FALSE        FALSE
## FullBath            FALSE     TRUE    FALSE        FALSE        FALSE
## HalfBath            FALSE    FALSE     TRUE        FALSE        FALSE
## BedroomAbvGr        FALSE    FALSE    FALSE         TRUE        FALSE
## KitchenAbvGr        FALSE    FALSE    FALSE        FALSE         TRUE
## TotRmsAbvGrd        FALSE    FALSE    FALSE        FALSE        FALSE
## Fireplaces          FALSE    FALSE    FALSE        FALSE        FALSE
## GarageCars          FALSE    FALSE    FALSE        FALSE        FALSE
## GarageArea          FALSE    FALSE    FALSE        FALSE        FALSE
## WoodDeckSF          FALSE    FALSE    FALSE        FALSE        FALSE
## MoSold              FALSE    FALSE    FALSE        FALSE        FALSE
##              TotRmsAbvGrd Fireplaces GarageCars GarageArea WoodDeckSF MoSold
## LotArea             FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## OverallQual         FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## OverallCond         FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## YearBuilt           FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## YearRemodAdd        FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## BsmtFinSF1          FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## BsmtFinSF2          FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## BsmtUnfSF           FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## X1stFlrSF           FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## X2ndFlrSF           FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## LowQualFinSF        FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## BsmtFullBath        FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## BsmtHalfBath        FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## FullBath            FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## HalfBath            FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## BedroomAbvGr        FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## KitchenAbvGr        FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## TotRmsAbvGrd         TRUE      FALSE      FALSE      FALSE      FALSE  FALSE
## Fireplaces          FALSE       TRUE      FALSE      FALSE      FALSE  FALSE
## GarageCars          FALSE      FALSE       TRUE       TRUE      FALSE  FALSE
## GarageArea          FALSE      FALSE       TRUE       TRUE      FALSE  FALSE
## WoodDeckSF          FALSE      FALSE      FALSE      FALSE       TRUE  FALSE
## MoSold              FALSE      FALSE      FALSE      FALSE      FALSE   TRUE
```

We see that the variables are not highly correlated, so we should receive relatively the same model from both methods. So I went with backwards selection.

I will be using the Cp, or AIC, value. This is because the Cp value will help find a model that explains the observed variation in their data without a high risk of overfitting.

```r
backward.subset <- regsubsets(log(SalePrice)~.,data=train,nvmax=23,method='backward')
backward.subset.summary <- summary(backward.subset)
backward.subset.summary
```

```
## Subset selection object
## Call: regsubsets.formula(log(SalePrice) ~ ., data = train, nvmax = 23,
##     method = "backward")
## 23 Variables  (and intercept)
##              Forced in Forced out
## LotArea          FALSE      FALSE
## OverallQual      FALSE      FALSE
## OverallCond      FALSE      FALSE
```

```
## YearBuilt        FALSE      FALSE
## YearRemodAdd     FALSE      FALSE
## BsmtFinSF1       FALSE      FALSE
## BsmtFinSF2       FALSE      FALSE
## BsmtUnfSF        FALSE      FALSE
## X1stFlrSF        FALSE      FALSE
## X2ndFlrSF        FALSE      FALSE
## LowQualFinSF     FALSE      FALSE
## BsmtFullBath     FALSE      FALSE
## BsmtHalfBath     FALSE      FALSE
## FullBath         FALSE      FALSE
## HalfBath         FALSE      FALSE
## BedroomAbvGr     FALSE      FALSE
## KitchenAbvGr     FALSE      FALSE
## TotRmsAbvGrd     FALSE      FALSE
## Fireplaces       FALSE      FALSE
## GarageCars       FALSE      FALSE
## GarageArea       FALSE      FALSE
## WoodDeckSF       FALSE      FALSE
## MoSold           FALSE      FALSE
## 1 subsets of each size up to 23
## Selection Algorithm: backward
##           LotArea OverallQual OverallCond YearBuilt YearRemodAdd BsmtFinSF1
## 1  ( 1 )  " "     "*"         " "         " "       " "          " "
## 2  ( 1 )  " "     "*"         " "         " "       " "          " "
## 3  ( 1 )  " "     "*"         " "         " "       " "          " "
## 4  ( 1 )  " "     "*"         " "         "*"       " "          " "
## 5  ( 1 )  " "     "*"         "*"         "*"       " "          " "
## 6  ( 1 )  " "     "*"         "*"         "*"       " "          " "
## 7  ( 1 )  " "     "*"         "*"         "*"       " "          " "
## 8  ( 1 )  " "     "*"         "*"         "*"       " "          " "
## 9  ( 1 )  "*"     "*"         "*"         "*"       " "          " "
## 10  ( 1 ) "*"     "*"         "*"         "*"       " "          " "
## 11  ( 1 ) "*"     "*"         "*"         "*"       " "          " "
## 12  ( 1 ) "*"     "*"         "*"         "*"       "*"          " "
## 13  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
## 14  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
## 15  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
## 16  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
## 17  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
## 18  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
## 19  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
## 20  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
## 21  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
## 22  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
## 23  ( 1 ) "*"     "*"         "*"         "*"       "*"          "*"
##           BsmtFinSF2 BsmtUnfSF X1stFlrSF X2ndFlrSF LowQualFinSF BsmtFullBath
## 1  ( 1 )  " "        " "       " "       " "       " "          " "
## 2  ( 1 )  " "        " "       "*"       " "       " "          " "
## 3  ( 1 )  " "        " "       "*"       "*"       " "          " "
## 4  ( 1 )  " "        " "       "*"       "*"       " "          " "
## 5  ( 1 )  " "        " "       "*"       "*"       " "          " "
## 6  ( 1 )  " "        " "       "*"       "*"       " "          " "
## 7  ( 1 )  " "        " "       "*"       "*"       " "          "*"
```

```
## 8  ( 1 )  " "          " "        "*"        "*"        " "          "*"
## 9  ( 1 )  " "          " "        "*"        "*"        " "          "*"
## 10  ( 1 ) " "          " "        "*"        "*"        " "          "*"
## 11  ( 1 ) " "          " "        "*"        "*"        " "          "*"
## 12  ( 1 ) " "          " "        "*"        "*"        " "          "*"
## 13  ( 1 ) " "          " "        "*"        "*"        " "          "*"
## 14  ( 1 ) " "          "*"        "*"        "*"        " "          "*"
## 15  ( 1 ) "*"          "*"        "*"        "*"        " "          "*"
## 16  ( 1 ) "*"          "*"        "*"        "*"        " "          "*"
## 17  ( 1 ) "*"          "*"        "*"        "*"        " "          "*"
## 18  ( 1 ) "*"          "*"        "*"        "*"        " "          "*"
## 19  ( 1 ) "*"          "*"        "*"        "*"        "*"          "*"
## 20  ( 1 ) "*"          "*"        "*"        "*"        "*"          "*"
## 21  ( 1 ) "*"          "*"        "*"        "*"        "*"          "*"
## 22  ( 1 ) "*"          "*"        "*"        "*"        "*"          "*"
## 23  ( 1 ) "*"          "*"        "*"        "*"        "*"          "*"
##           BsmtHalfBath FullBath HalfBath BedroomAbvGr KitchenAbvGr TotRmsAbvGrd
## 1  ( 1 )  " "          " "      " "      " "          " "          " "
## 2  ( 1 )  " "          " "      " "      " "          " "          " "
## 3  ( 1 )  " "          " "      " "      " "          " "          " "
## 4  ( 1 )  " "          " "      " "      " "          " "          " "
## 5  ( 1 )  " "          " "      " "      " "          " "          " "
## 6  ( 1 )  " "          " "      " "      " "          " "          " "
## 7  ( 1 )  " "          " "      " "      " "          " "          " "
## 8  ( 1 )  " "          " "      " "      " "          " "          " "
## 9  ( 1 )  " "          " "      " "      " "          " "          " "
## 10  ( 1 ) " "          " "      " "      " "          "*"          " "
## 11  ( 1 ) " "          " "      " "      " "          "*"          "*"
## 12  ( 1 ) " "          " "      " "      " "          "*"          "*"
## 13  ( 1 ) " "          " "      " "      " "          "*"          "*"
## 14  ( 1 ) " "          " "      " "      " "          "*"          "*"
## 15  ( 1 ) " "          " "      " "      " "          "*"          "*"
## 16  ( 1 ) " "          " "      " "      " "          "*"          "*"
## 17  ( 1 ) " "          "*"      " "      " "          "*"          "*"
## 18  ( 1 ) " "          "*"      "*"      " "          "*"          "*"
## 19  ( 1 ) " "          "*"      "*"      " "          "*"          "*"
## 20  ( 1 ) "*"          "*"      "*"      " "          "*"          "*"
## 21  ( 1 ) "*"          "*"      "*"      " "          "*"          "*"
## 22  ( 1 ) "*"          "*"      "*"      " "          "*"          "*"
## 23  ( 1 ) "*"          "*"      "*"      "*"          "*"          "*"
##           Fireplaces GarageCars GarageArea WoodDeckSF MoSold
## 1  ( 1 )  " "        " "        " "        " "        " "
## 2  ( 1 )  " "        " "        " "        " "        " "
## 3  ( 1 )  " "        " "        " "        " "        " "
## 4  ( 1 )  " "        " "        " "        " "        " "
## 5  ( 1 )  " "        " "        " "        " "        " "
## 6  ( 1 )  " "        "*"        " "        " "        " "
## 7  ( 1 )  " "        "*"        " "        " "        " "
## 8  ( 1 )  "*"        "*"        " "        " "        " "
## 9  ( 1 )  "*"        "*"        " "        " "        " "
## 10  ( 1 ) "*"        "*"        " "        " "        " "
## 11  ( 1 ) "*"        "*"        " "        " "        " "
## 12  ( 1 ) "*"        "*"        " "        " "        " "
## 13  ( 1 ) "*"        "*"        " "        " "        " "
```

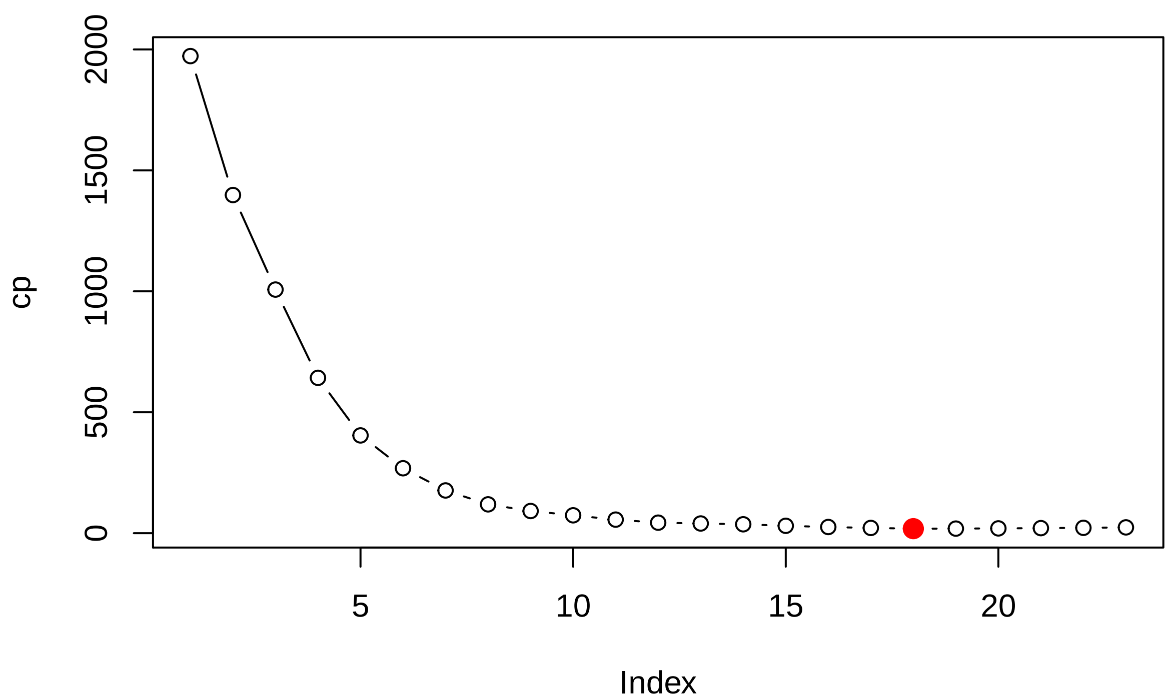
```
## 14  ( 1 ) "*"        "*"        " "        " "        " "
## 15  ( 1 ) "*"        "*"        " "        " "        " "
## 16  ( 1 ) "*"        "*"        " "        "*"        " "
## 17  ( 1 ) "*"        "*"        " "        "*"        " "
## 18  ( 1 ) "*"        "*"        " "        "*"        " "
## 19  ( 1 ) "*"        "*"        " "        "*"        " "
## 20  ( 1 ) "*"        "*"        " "        "*"        " "
## 21  ( 1 ) "*"        "*"        " "        "*"        "*"
## 22  ( 1 ) "*"        "*"        "*"        "*"        "*"
## 23  ( 1 ) "*"        "*"        "*"        "*"        "*"
```

These is our subset selection algorithm function.

```
plot(backward.subset.summary$cp,type='b',ylab='cp')
which.min(backward.subset.summary$cp)
```

```
## [1] 18
```

```
points(18,backward.subset.summary$cp[18],col='red',cex=2,pch=20)
```



We see that the Cp minimizes at 18 parameters.

```
backward.subset.summary$which
```

```
##    (Intercept) LotArea OverallQual OverallCond YearBuilt YearRemodAdd
## 1         TRUE   FALSE        TRUE       FALSE     FALSE        FALSE
## 2         TRUE   FALSE        TRUE       FALSE     FALSE        FALSE
## 3         TRUE   FALSE        TRUE       FALSE     FALSE        FALSE
## 4         TRUE   FALSE        TRUE       FALSE      TRUE        FALSE
## 5         TRUE   FALSE        TRUE        TRUE      TRUE        FALSE
## 6         TRUE   FALSE        TRUE        TRUE      TRUE        FALSE
## 7         TRUE   FALSE        TRUE        TRUE      TRUE        FALSE
## 8         TRUE   FALSE        TRUE        TRUE      TRUE        FALSE
## 9         TRUE    TRUE        TRUE        TRUE      TRUE        FALSE
## 10        TRUE    TRUE        TRUE        TRUE      TRUE        FALSE
## 11        TRUE    TRUE        TRUE        TRUE      TRUE        FALSE
## 12        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 13        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 14        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 15        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 16        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 17        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 18        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 19        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 20        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 21        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 22        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
## 23        TRUE    TRUE        TRUE        TRUE      TRUE         TRUE
##    BsmtFinSF1 BsmtFinSF2 BsmtUnfSF X1stFlrSF X2ndFlrSF LowQualFinSF
## 1       FALSE      FALSE     FALSE     FALSE     FALSE        FALSE
## 2       FALSE      FALSE     FALSE      TRUE     FALSE        FALSE
## 3       FALSE      FALSE     FALSE      TRUE      TRUE        FALSE
## 4       FALSE      FALSE     FALSE      TRUE      TRUE        FALSE
## 5       FALSE      FALSE     FALSE      TRUE      TRUE        FALSE
## 6       FALSE      FALSE     FALSE      TRUE      TRUE        FALSE
## 7       FALSE      FALSE     FALSE      TRUE      TRUE        FALSE
## 8       FALSE      FALSE     FALSE      TRUE      TRUE        FALSE
## 9       FALSE      FALSE     FALSE      TRUE      TRUE        FALSE
## 10      FALSE      FALSE     FALSE      TRUE      TRUE        FALSE
## 11      FALSE      FALSE     FALSE      TRUE      TRUE        FALSE
## 12      FALSE      FALSE     FALSE      TRUE      TRUE        FALSE
## 13       TRUE      FALSE     FALSE      TRUE      TRUE        FALSE
## 14       TRUE      FALSE      TRUE      TRUE      TRUE        FALSE
## 15       TRUE       TRUE      TRUE      TRUE      TRUE        FALSE
## 16       TRUE       TRUE      TRUE      TRUE      TRUE        FALSE
## 17       TRUE       TRUE      TRUE      TRUE      TRUE        FALSE
## 18       TRUE       TRUE      TRUE      TRUE      TRUE        FALSE
## 19       TRUE       TRUE      TRUE      TRUE      TRUE         TRUE
## 20       TRUE       TRUE      TRUE      TRUE      TRUE         TRUE
## 21       TRUE       TRUE      TRUE      TRUE      TRUE         TRUE
## 22       TRUE       TRUE      TRUE      TRUE      TRUE         TRUE
## 23       TRUE       TRUE      TRUE      TRUE      TRUE         TRUE
##    BsmtFullBath BsmtHalfBath FullBath HalfBath BedroomAbvGr KitchenAbvGr
## 1         FALSE        FALSE    FALSE    FALSE        FALSE        FALSE
## 2         FALSE        FALSE    FALSE    FALSE        FALSE        FALSE
## 3         FALSE        FALSE    FALSE    FALSE        FALSE        FALSE
## 4         FALSE        FALSE    FALSE    FALSE        FALSE        FALSE
## 5         FALSE        FALSE    FALSE    FALSE        FALSE        FALSE
```

```
## 6         FALSE        FALSE    FALSE    FALSE        FALSE        FALSE
## 7          TRUE        FALSE    FALSE    FALSE        FALSE        FALSE
## 8          TRUE        FALSE    FALSE    FALSE        FALSE        FALSE
## 9          TRUE        FALSE    FALSE    FALSE        FALSE        FALSE
## 10         TRUE        FALSE    FALSE    FALSE        FALSE         TRUE
## 11         TRUE        FALSE    FALSE    FALSE        FALSE         TRUE
## 12         TRUE        FALSE    FALSE    FALSE        FALSE         TRUE
## 13         TRUE        FALSE    FALSE    FALSE        FALSE         TRUE
## 14         TRUE        FALSE    FALSE    FALSE        FALSE         TRUE
## 15         TRUE        FALSE    FALSE    FALSE        FALSE         TRUE
## 16         TRUE        FALSE    FALSE    FALSE        FALSE         TRUE
## 17         TRUE        FALSE     TRUE    FALSE        FALSE         TRUE
## 18         TRUE        FALSE     TRUE     TRUE        FALSE         TRUE
## 19         TRUE        FALSE     TRUE     TRUE        FALSE         TRUE
## 20         TRUE         TRUE     TRUE     TRUE        FALSE         TRUE
## 21         TRUE         TRUE     TRUE     TRUE        FALSE         TRUE
## 22         TRUE         TRUE     TRUE     TRUE        FALSE         TRUE
## 23         TRUE         TRUE     TRUE     TRUE         TRUE         TRUE
##    TotRmsAbvGrd Fireplaces GarageCars GarageArea WoodDeckSF MoSold
## 1         FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## 2         FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## 3         FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## 4         FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## 5         FALSE      FALSE      FALSE      FALSE      FALSE  FALSE
## 6         FALSE      FALSE       TRUE      FALSE      FALSE  FALSE
## 7         FALSE      FALSE       TRUE      FALSE      FALSE  FALSE
## 8         FALSE       TRUE       TRUE      FALSE      FALSE  FALSE
## 9         FALSE       TRUE       TRUE      FALSE      FALSE  FALSE
## 10        FALSE       TRUE       TRUE      FALSE      FALSE  FALSE
## 11         TRUE       TRUE       TRUE      FALSE      FALSE  FALSE
## 12         TRUE       TRUE       TRUE      FALSE      FALSE  FALSE
## 13         TRUE       TRUE       TRUE      FALSE      FALSE  FALSE
## 14         TRUE       TRUE       TRUE      FALSE      FALSE  FALSE
## 15         TRUE       TRUE       TRUE      FALSE      FALSE  FALSE
## 16         TRUE       TRUE       TRUE      FALSE       TRUE  FALSE
## 17         TRUE       TRUE       TRUE      FALSE       TRUE  FALSE
## 18         TRUE       TRUE       TRUE      FALSE       TRUE  FALSE
## 19         TRUE       TRUE       TRUE      FALSE       TRUE  FALSE
## 20         TRUE       TRUE       TRUE      FALSE       TRUE  FALSE
## 21         TRUE       TRUE       TRUE      FALSE       TRUE   TRUE
## 22         TRUE       TRUE       TRUE       TRUE       TRUE   TRUE
## 23         TRUE       TRUE       TRUE       TRUE       TRUE   TRUE
```

For 18 parameters, the ones that were dropped were LowQualFinSF, BsmtHalfBath, BedroomAbvGround, GarageArea, and MoSold.

```
coef(backward.subset,18)
```

```
##   (Intercept)       LotArea   OverallQual   OverallCond     YearBuilt
##  3.516487e+00  2.008737e-06  8.320284e-02  4.996552e-02  2.536890e-03
##  YearRemodAdd    BsmtFinSF1    BsmtFinSF2     BsmtUnfSF     X1stFlrSF
##  1.032479e-03  8.605499e-05  8.414686e-05  6.352930e-05  1.976985e-04
##     X2ndFlrSF  BsmtFullBath      FullBath      HalfBath  KitchenAbvGr
```

```
##  1.381951e-04  5.540962e-02  3.668533e-02  2.522480e-02 -1.059376e-01
##  TotRmsAbvGrd    Fireplaces    GarageCars    WoodDeckSF
##  2.085906e-02  4.821921e-02  7.954215e-02  8.895361e-05
```

Here we see our final selection of variables that we will be using. It is important to know that we still have a log transformed Y response term. The interpretation of these coefficients would be different from a traditional linear model. For example, the interpretation of the FullBath variable would be for each increment of 1 in FullBath, the price of the home would increase by exp(0.03669-1)*100 = 3.7 percent.

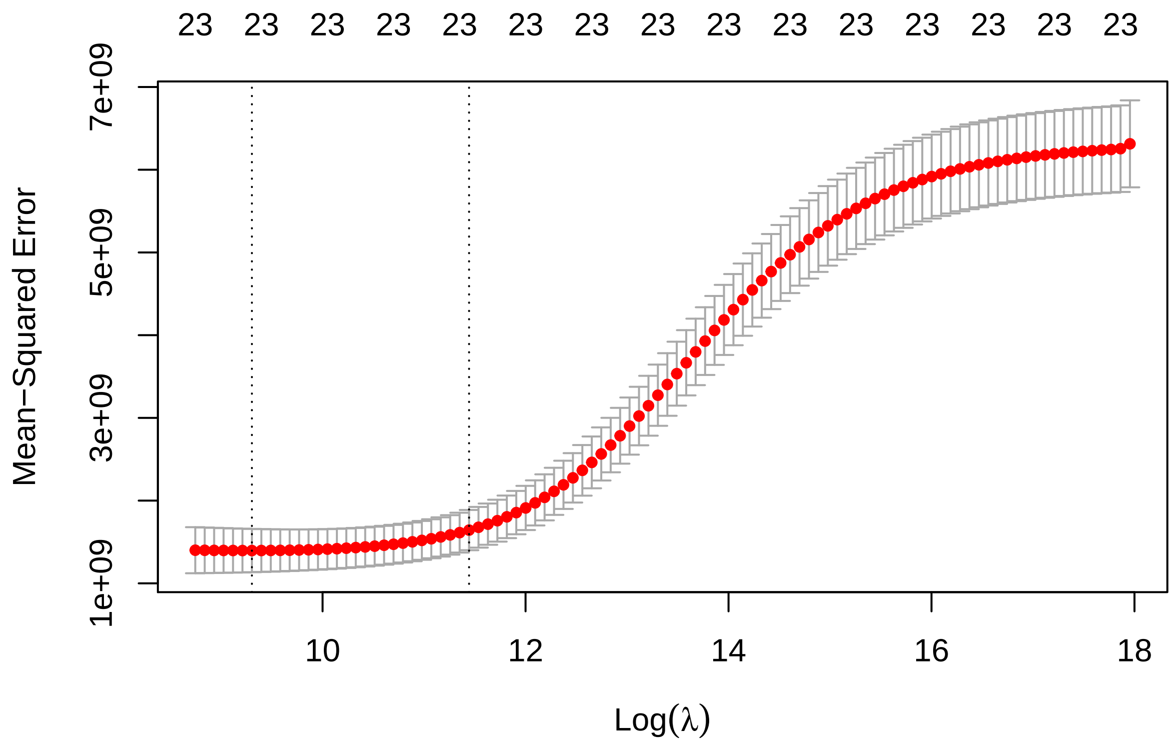
Of our variables, we see that these variables make sense, such as how the price of the home increases as the Lot Area, Overall Quality, and the number of Full Bathrooms increases.

```
predict.regsubsets <-function(object, newdata , id, ...){
  form <- as.formula(object$call[[2]])
  mat <-model.matrix(form, newdata)
  coefi <-coef(object, id = id)
  xvars <-names(coefi)
  return(mat[,xvars]%*%coefi)
}

subset.lindata <- predict.regsubsets(backward.subset,newdata=test,id=which.min(backward.subset.summary$
head(exp(subset.lindata))
```

```
##       [,1]
## 1 112296.0
## 2 146762.6
## 3 169440.7
## 4 197657.6
## 5 184108.9
## 6 176827.8
```

Shrinkage Methods

```
X <- model.matrix(log(train_y$SalePrice)~.,data=train_x)[,-1]
y <- train$SalePrice
grid <- 10^seq(10,-2,length=1000)


ridge.mod <- glmnet(X,y,alpha=0,lambda=grid)
cv.out <- cv.glmnet(X,y,alpha=0,nfolds=10)
plot(cv.out)
```




```
bestlam <- cv.out$lambda.min
coef(ridge.mod,s=bestlam)
```

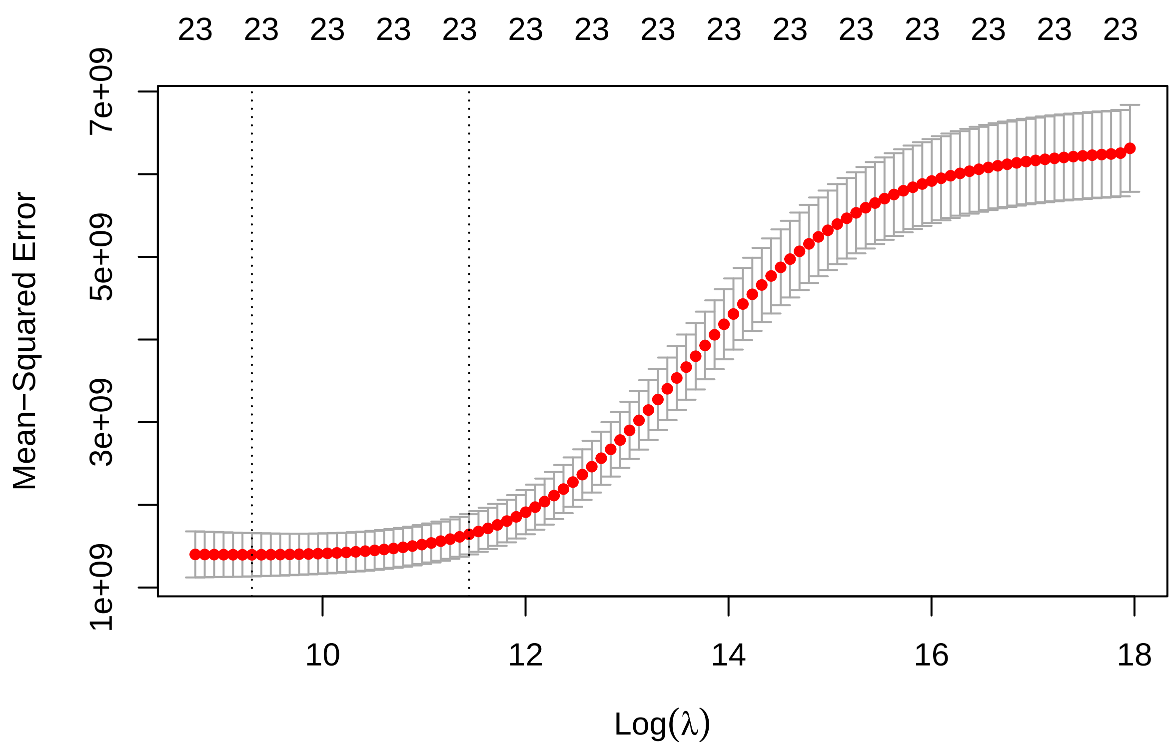
```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  -9.519563e+05
## LotArea       4.314769e-01
## OverallQual   1.544608e+04
## OverallCond   3.407742e+03
## YearBuilt     2.199290e+02
## YearRemodAdd  2.318874e+02
## BsmtFinSF1    2.156152e+01
## BsmtFinSF2    7.608335e+00
## BsmtUnfSF     1.140119e+01
## X1stFlrSF     3.813270e+01
## X2ndFlrSF     2.605920e+01
## LowQualFinSF  1.016257e+01
## BsmtFullBath  7.399182e+03
## BsmtHalfBath  8.656893e+02
## FullBath      8.345980e+03
## HalfBath      4.505173e+03
## BedroomAbvGr -5.451508e+03
## KitchenAbvGr -2.410988e+04
## TotRmsAbvGrd  6.197832e+03
## Fireplaces    7.281929e+03
```

```
## GarageCars    8.973880e+03
## GarageArea    2.260301e+01
## WoodDeckSF    2.530776e+01
## MoSold        9.238410e+01
```

```r
testx <- model.matrix(log(SalePrice)~.,test)[,-1]
ridge.pred <- predict(ridge.mod,s=bestlam,newx=testx)

head(ridge.pred)
```

```
##          1
## 1 119072.4
## 2 163287.6
## 3 182747.6
## 4 207489.9
## 5 199049.9
## 6 187450.2
```

```r
lasso.mod <- glmnet(X,y,alpha=1,lambda=grid)
cv.out2 <- cv.glmnet(X,y,alpha=1)
plot(cv.out)
```




```r
bestlam2 <- cv.out2$lambda.min
coef(lasso.mod, s=bestlam2)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  -9.180985e+05
## LotArea       4.210919e-01
## OverallQual   1.888715e+04
## OverallCond   3.305530e+03
## YearBuilt     2.619258e+02
## YearRemodAdd  1.680358e+02
## BsmtFinSF1    1.643315e+01
## BsmtFinSF2    .
## BsmtUnfSF     3.817340e+00
## X1stFlrSF     5.496407e+01
## X2ndFlrSF     3.833328e+01
## LowQualFinSF  .
## BsmtFullBath  5.793133e+03
## BsmtHalfBath  .
## FullBath      1.603800e+03
## HalfBath      .
## BedroomAbvGr -5.754066e+03
## KitchenAbvGr -2.268600e+04
## TotRmsAbvGrd  5.035539e+03
## Fireplaces    4.158788e+03
## GarageCars    1.039467e+04
## GarageArea    8.461287e+00
## WoodDeckSF    2.090570e+01
## MoSold        .
```

```r
lasso.pred <- predict(lasso.mod, s=bestlam2, newx = testx)
head(lasso.pred)
```

```
##          1
## 1 114273.8
## 2 165814.3
## 3 175820.9
## 4 201297.9
## 5 205937.5
## 6 182549.4
```

Next, we run a ridge and lasso regression. These two methods are called shrinkage regression methods because they utilize shrinking, which is when the coefficient estimates are shrunk down towards 0. The difference between ridge and lasso are that lasso utilizes a regularization term in absolute value. Lasso also sets irrelevant variables to 0.

We used cross validation to determine the best tuning parameters for both models. We utilized the cv.glmnet function that optimized lambda. The optimized lambda are saved as bestlam and bestlam2 for the ridge and lasso regression respectively.

In general, the Lasso method is preferred in terms of model interpretation. This is because in Ridge Regression, there can be many coefficients that are not 0. This does not mean that Lasso always leads to a higher prediction accuracy though. We will still need a process of cross validation to determine which model will be more accurate to our data.

Generalized Additive Models

```r
gamfit <- gam(log(train_y$SalePrice)~s(LotArea) + s(OverallQual) + s(OverallCond) + s(YearBuilt) + s(Yea

summary(gamfit)
```

```
##
## Call: gam(formula = log(train_y$SalePrice) ~ s(LotArea) + s(OverallQual) +
##     s(OverallCond) + s(YearBuilt) + s(YearRemodAdd) + s(BsmtFinSF1) +
##     s(BsmtFinSF2) + s(BsmtUnfSF) + s(X1stFlrSF) + s(X2ndFlrSF) +
##     s(LowQualFinSF) + BsmtFullBath + BsmtHalfBath + FullBath +
##     HalfBath + s(BedroomAbvGr) + KitchenAbvGr + s(TotRmsAbvGrd) +
##     Fireplaces + s(GarageCars) + s(GarageArea) + s(WoodDeckSF) +
##     s(MoSold), data = train_x)
## Deviance Residuals:
##       Min        1Q    Median        3Q       Max
## -1.158350 -0.048885  0.004274  0.063715  0.470254
##
## (Dispersion Parameter for gaussian family taken to be 0.0152)
##
##     Null Deviance: 232.8007 on 1459 degrees of freedom
## Residual Deviance: 21.0589 on 1385 degrees of freedom
## AIC: -1893.446
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                   Df  Sum Sq Mean Sq  F value    Pr(>F)
## s(LotArea)         1  17.421  17.421 1145.7527 < 2.2e-16 ***
## s(OverallQual)     1 134.867 134.867 8869.8819 < 2.2e-16 ***
## s(OverallCond)     1   0.434   0.434   28.5528 1.065e-07 ***
## s(YearBuilt)       1   8.785   8.785  577.7814 < 2.2e-16 ***
## s(YearRemodAdd)    1   0.539   0.539   35.4469 3.317e-09 ***
## s(BsmtFinSF1)      1   5.071   5.071  333.5046 < 2.2e-16 ***
## s(BsmtFinSF2)      1   0.404   0.404   26.5451 2.947e-07 ***
## s(BsmtUnfSF)       1   3.015   3.015  198.2597 < 2.2e-16 ***
## s(X1stFlrSF)       1   2.170   2.170  142.7446 < 2.2e-16 ***
## s(X2ndFlrSF)       1  15.441  15.441 1015.5015 < 2.2e-16 ***
## s(LowQualFinSF)    1   0.070   0.070    4.5917  0.032301 *
## BsmtFullBath       1   0.118   0.118    7.7855  0.005339 **
## BsmtHalfBath       1   0.016   0.016    1.0831  0.298180
## FullBath           1   0.015   0.015    0.9539  0.328891
## HalfBath           1   0.262   0.262   17.2320 3.510e-05 ***
## s(BedroomAbvGr)    1   0.086   0.086    5.6369  0.017722 *
## KitchenAbvGr       1   0.472   0.472   31.0451 3.025e-08 ***
## s(TotRmsAbvGrd)    1   0.071   0.071    4.6381  0.031442 *
## Fireplaces         1   0.794   0.794   52.2245 8.151e-13 ***
## s(GarageCars)      1   1.107   1.107   72.8293 < 2.2e-16 ***
## s(GarageArea)      1   0.003   0.003    0.2099  0.646940
## s(WoodDeckSF)      1   0.073   0.073    4.7798  0.028962 *
## s(MoSold)          1   0.000   0.000    0.0146  0.903960
## Residuals       1385  21.059   0.015
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Anova for Nonparametric Effects
##                   Npar Df Npar F     Pr(F)
## (Intercept)
## s(LotArea)              3 19.642 1.797e-12 ***
## s(OverallQual)          3  6.545  0.000215 ***
## s(OverallCond)          3  4.913  0.002125 **
## s(YearBuilt)            3 16.077 2.820e-10 ***
## s(YearRemodAdd)         3  3.758  0.010527 *
## s(BsmtFinSF1)           3 46.648 < 2.2e-16 ***
## s(BsmtFinSF2)           3  1.353  0.255522
## s(BsmtUnfSF)            3  1.659  0.173910
## s(X1stFlrSF)            3 56.406 < 2.2e-16 ***
## s(X2ndFlrSF)            3  1.321  0.265984
## s(LowQualFinSF)         3  2.486  0.059096 .
## BsmtFullBath
## BsmtHalfBath
## FullBath
## HalfBath
## s(BedroomAbvGr)         3  1.910  0.126070
## KitchenAbvGr
## s(TotRmsAbvGrd)         3  0.960  0.410588
## Fireplaces
## s(GarageCars)           3  7.530 5.344e-05 ***
## s(GarageArea)           3  4.747  0.002679 **
## s(WoodDeckSF)           3  0.687  0.560266
## s(MoSold)               3  3.157  0.023940 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note: BsmtHalfBath, BsmtFullBath, FullBath, KitchenAbvGr, Fireplaces, and HalfBath were not splined due to their low number of unique variables. This may imply we treat them as qualitative variables instead of quantitative.

Our GAM model utilizes smoothing splines and local regression on our quantitative predictors.

This is our model for the default df, 4. We see in the summary of the model that some variables are linearly significant, some are nonlinearly significant, and some are both or neither. For example, we see that GarageArea is not significant in a linear setting but is significant in a nonlinear setting.

We will try different values for the df to see if we can get better results.

```r
error = 0
for(i in 1:nrow(train_x)){
  gamfit <- gam(log(train_y$SalePrice[-i])~s(LotArea) + s(OverallQual) + s(OverallCond) + s(YearBuilt) +

  pred <- predict(gamfit,train_x[i,])
  error <- error + (log(train_y$SalePrice[i])-pred)^2
}
sqrt(error/nrow(train_x))
```

```
##        1
## 0.130365
```

```r
error = 0
for(i in 1:nrow(train_x)){
  gamfit <- gam(log(train_y$SalePrice[-i])~s(LotArea,df=5) + s(OverallQual,df=5) + s(OverallCond,df=5) +

  pred <- predict(gamfit,train_x[i,])
  error <- error + (log(train_y$SalePrice[i])-pred)^2
}
sqrt(error/nrow(train_x))
```

```
##        1
## 0.134176
```

```r
error = 0
for(i in 1:nrow(train_x)){
  gamfit <- gam(log(train_y$SalePrice[-i])~s(LotArea,df=10) + s(OverallQual,df=10) + s(OverallCond,df=1

  pred <- predict(gamfit,train_x[i,])
  error <- error + (log(train_y$SalePrice[i])-pred)^2
}
sqrt(error/nrow(train_x))
```

```
##         1
## 0.1987622
```

We see the estimated test MSEs from df = 4 (0.184), df= 5 (0.190), and df=10(0.281). The lowest MSE came from df= 4. This will be the model we will use.

```r
gampreds <- exp(predict(gamfit, newdata=test_x))
head(gampreds)
```

```
##        1        2        3        4        5        6
## 124083.2 162391.1 191346.3 208782.0 173370.8 167742.3
```

These are the first 6 values that our model predicts using the GAM fit.

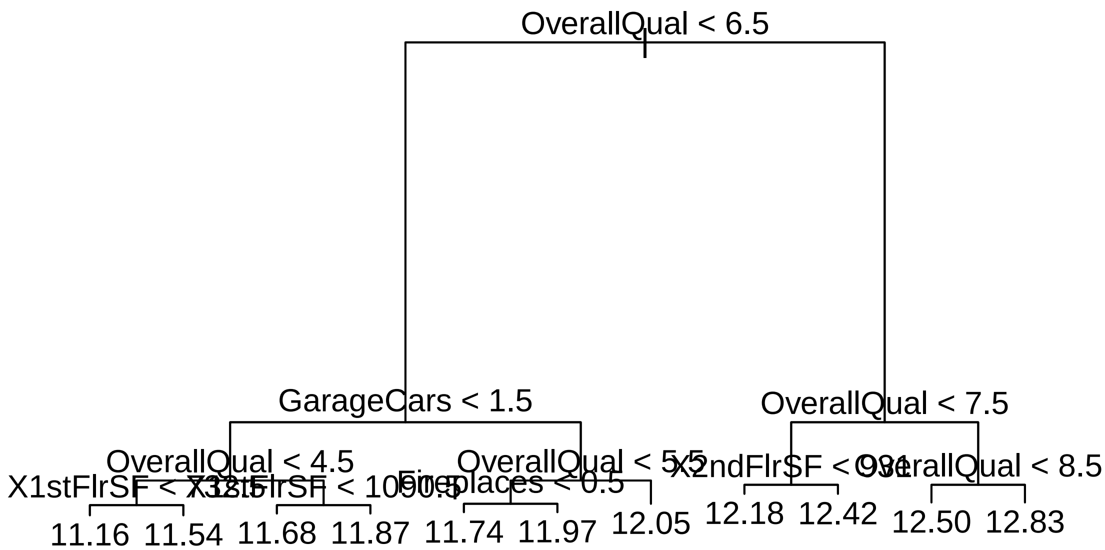
Regression Trees

```r
set.seed(10)
treefit <- tree(log(train_y$SalePrice)~.,data=train_x)
summary(treefit)
```

```
##
## Regression tree:
## tree(formula = log(train_y$SalePrice) ~ ., data = train_x)
## Variables actually used in tree construction:
## [1] "OverallQual" "GarageCars"  "X1stFlrSF"   "Fireplaces"  "X2ndFlrSF"
## Number of terminal nodes:  11
## Residual mean deviance:  0.0431 = 62.46 / 1449
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.8696 -0.1198  0.0106  0.0000  0.1257  0.7034
```

In this case, we are using a regression tree as our model. A regression tree utilizes a decision tree to estimate the response variable SalePrice. We see that the training MSE is 0.04 and the number of terminal nodes is 11. This is our unpruned model because we have not yet pruned, or narrowed down, our model.

```
plot(treefit)
text(treefit,pretty=0)
```



We have a visualization of the tree we are using. We have OverallQual as our top variable, which makes sense as the quality of a home intuitively should have a high impact on the sale price.

To optimize the number of trees, we will utilize cross validation.

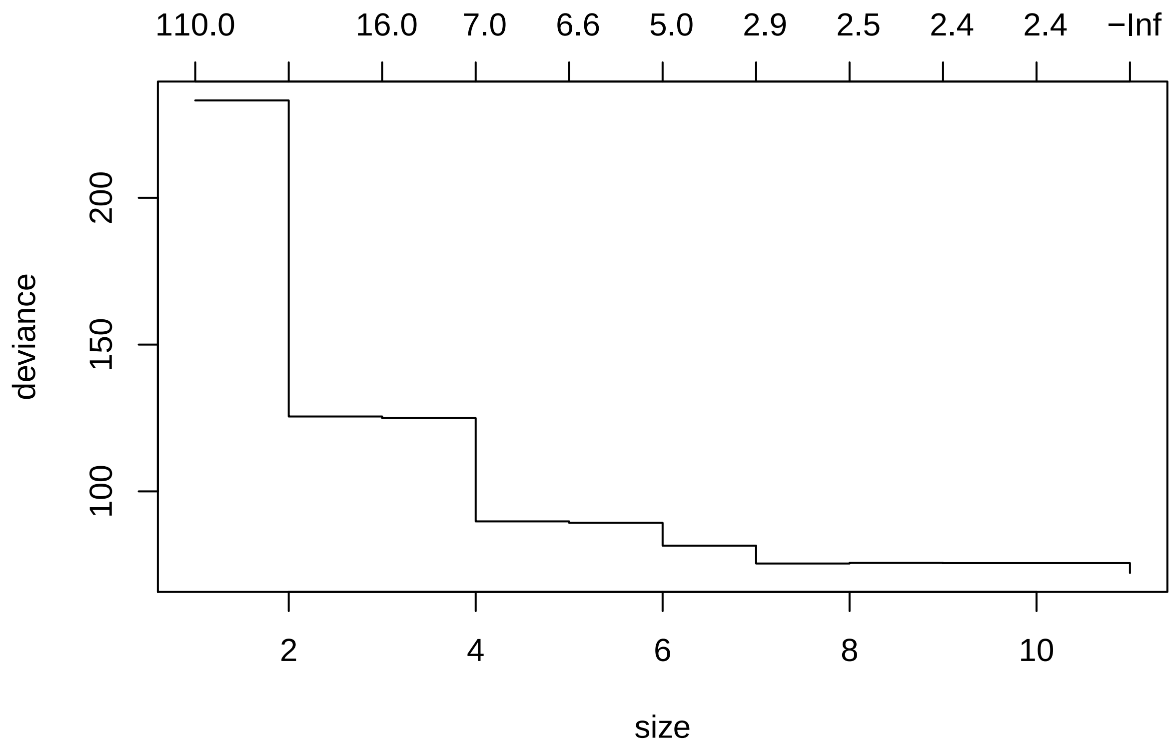
```
treefitcv <- cv.tree(treefit)
treefitcv
```

```
## $size
##  [1] 11 10  9  8  7  6  5  4  3  2  1
##
## $dev
##  [1]  72.18475  75.56561  75.56561  75.62371  75.42584  81.50909  89.28855
##  [8]  89.77814 124.96357 125.50608 233.16618
##
## $k
##  [1]       -Inf   2.377175   2.378557   2.548762   2.872628   4.988011
##  [7]   6.578577   6.964175  16.486284  17.696352 107.453137
##
```

```
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

This cross validation function allows us to optimize our regression tree based on the estimated test MSE.

```r
plot(treefitcv)
```



The lowest cross-validated error corresponds to 11, which is the same as our original model. This means that cross-validation did not lead to the selection of a pruned tree.

```r
treepreds <- exp(predict(treefit,newdata=test_x))
head(treepreds)
```

```
##        1        2        3        4        5        6
## 118735.1 143158.2 157864.8 170271.6 267584.7 170271.6
```

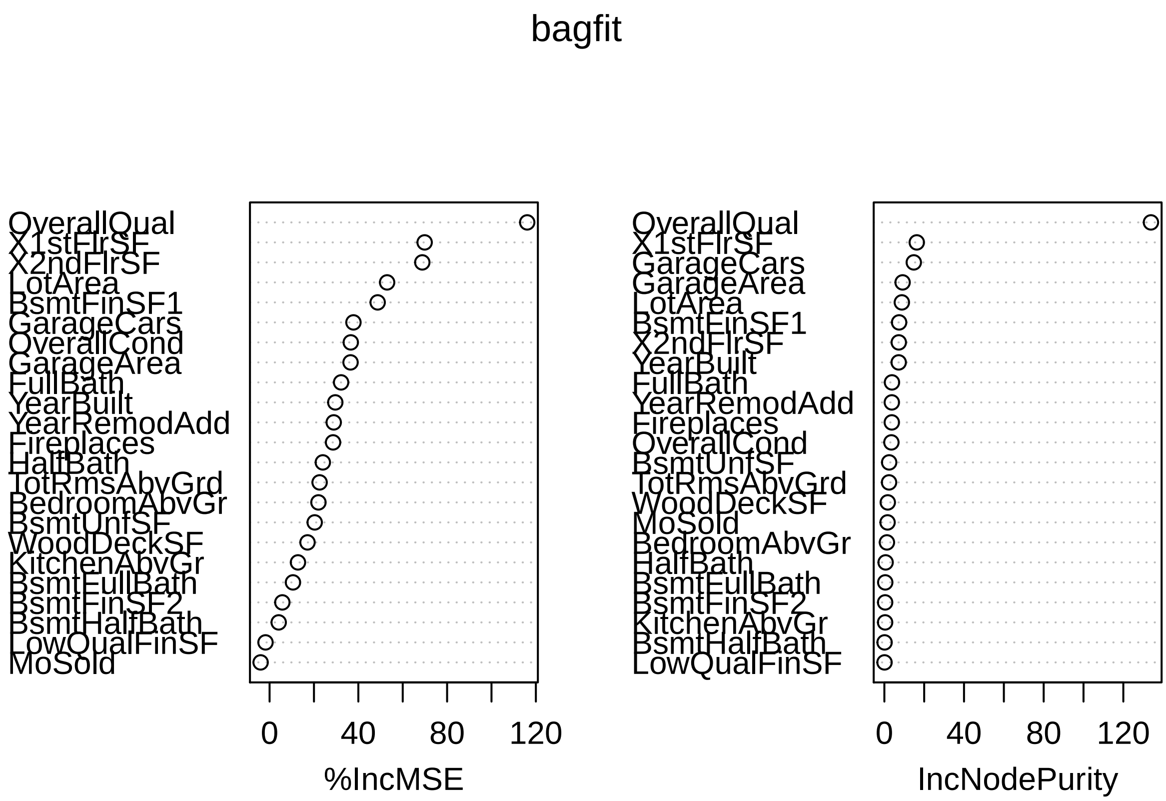
These are the first 6 values of our predicted data.

Bagging

```r
bagfit <- randomForest(log(train_y$SalePrice)~.,data=train_x,mtry=ncol(train_x)-1,importance=TRUE,ntree=
importance(bagfit)
```

```
##                 %IncMSE IncNodePurity
## LotArea       52.965825    8.78637797
## OverallQual  116.100045  133.86284315
## OverallCond   36.592322    3.53481149
## YearBuilt     29.612277    7.20910147
## YearRemodAdd  28.911480    3.74815508
## BsmtFinSF1    48.716660    7.41038379
## BsmtFinSF2     5.751527    0.39368867
## BsmtUnfSF     20.332966    2.40614956
## X1stFlrSF     69.879023   16.27392075
## X2ndFlrSF     68.821226    7.24387807
## LowQualFinSF  -1.830095    0.08021749
## BsmtFullBath  10.543830    0.44671128
## BsmtHalfBath   4.109108    0.10950879
## FullBath      32.235365    3.81585333
## HalfBath      23.990772    0.62457744
## BedroomAbvGr  22.020290    1.28546345
## KitchenAbvGr  12.797801    0.37270524
## TotRmsAbvGrd  22.548826    2.35409212
## Fireplaces    28.602873    3.73037627
## GarageCars    37.795252   14.77616756
## GarageArea    36.497118    9.15489362
## WoodDeckSF    17.095666    1.71277946
## MoSold        -4.032062    1.60589395
```

```r
varImpPlot(bagfit)
```



We use a bagging model now to estimate our response variable SalePrice. A bagging model utilizes bootstrap aggregation. This means that it averages a set of bootstrapped decision trees. The advantage of the bagging model is that it leads to lower bias and variance.

Here we use ncol(train_x)-1 as our parameter for the mtry value to indicate a bagging model. This is because m=p in a bagging model. We are also using ntree as 1000. We see from the plot that OverallQual has the most impact. This is consistent with our regression tree model results which also indicated that OverallQual had the highest impact on our model.

```
bagpreds <- exp(predict(bagfit,newdata=test_x))
head(bagpreds)
```

```
##        1        2        3        4        5        6
## 129564.0 156959.9 170363.3 180191.0 193017.9 186423.5
```

Here are the first 6 predictors for SalePrice using this model.

Random Forest

```
randomforestfit <- randomForest(log(train_y$SalePrice)~.,data=train_x,mtry=round(sqrt(ncol(train_x)-1))
randomforestfit
```

```
##
## Call:
##  randomForest(formula = log(train_y$SalePrice) ~ ., data = train_x,      mtry = round(sqrt(ncol(trai
##                Type of random forest: regression
```

```
##                      Number of trees: 1000
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 0.02046131
##                     % Var explained: 87.17
```

```r
importance(randomforestfit)
```

```
##                 %IncMSE IncNodePurity
## LotArea      42.1060144     9.8556156
## OverallQual  44.8490145    48.3017665
## OverallCond  30.1898257     4.0955320
## YearBuilt    33.0077264    27.1339375
## YearRemodAdd 26.4714437    12.2480556
## BsmtFinSF1   39.3373516     9.1625327
## BsmtFinSF2    5.0495237     0.5496874
## BsmtUnfSF    22.0093933     3.5792635
## X1stFlrSF    42.8593378    20.5800772
## X2ndFlrSF    44.6441867     8.5541598
## LowQualFinSF -0.6679745     0.1531464
## BsmtFullBath 19.0485290     1.2562851
## BsmtHalfBath  3.9639152     0.2144397
## FullBath     24.0269287    14.0704301
## HalfBath     24.5934326     2.0322275
## BedroomAbvGr 24.2131303     2.8001258
## KitchenAbvGr 13.8536934     1.0412902
## TotRmsAbvGrd 29.1442981     7.2862428
## Fireplaces   35.1744497    10.0320273
## GarageCars   25.0226640    21.4497581
## GarageArea   31.0366421    19.6286497
## WoodDeckSF   13.8338137     2.5482668
## MoSold        1.5890547     2.0921834
```

```r
varImpPlot(randomforestfit)
```

Now we will use a random forest model to predict our response variable SalePrice. A random forest model is similar to the bagging model where trees are averaged, but random forest takes the average of the decorrelated bootstrapped trees. This tends to lead to a lower variance model.

We use round(sqrt(ncol(train_x)-1)) as our parameter for the mtry model to indicate that it is a random forest model. This is because m is approximately sqrt(p) for a random forest model. We see that the training MSE is 0.0204. We see that the most important variable is OverallQual. This is consistent with previous models.

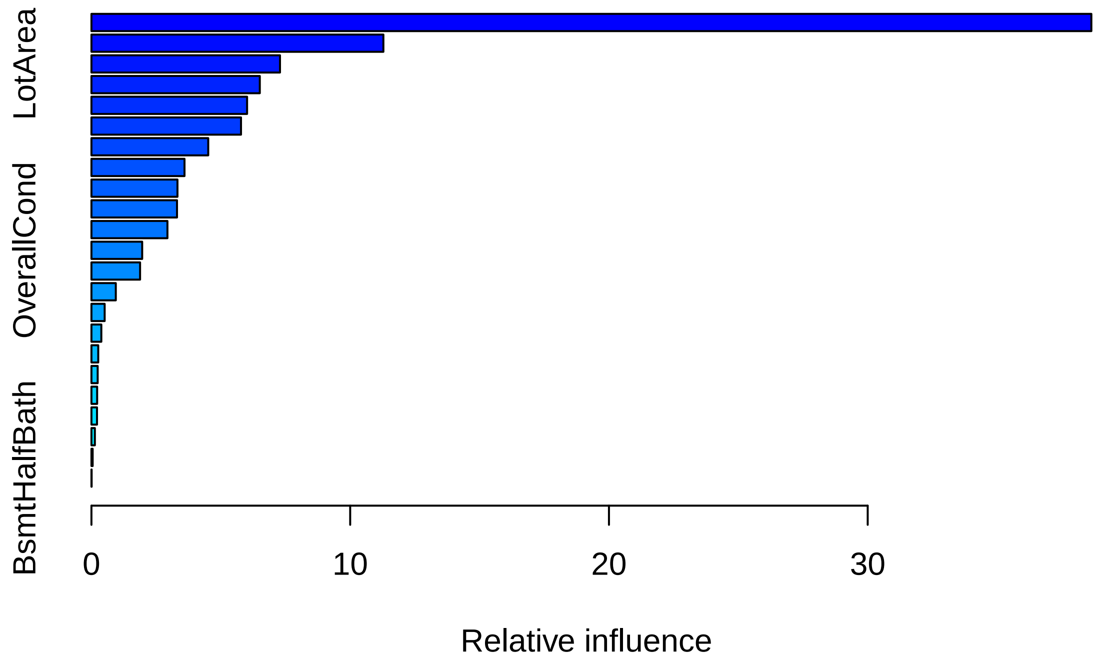
```
randomforestpreds <- exp(predict(randomforestfit,newdata=test_x))
head(randomforestpreds)
```

```
##        1        2        3        4        5        6
## 127401.6 151255.9 179730.6 185810.3 187941.8 188874.2
```

These are the first 6 predictions of our response variable SalePrice using this model.

Boosting

```
boostfit <- gbm(log(train_y$SalePrice) ~ .,data=train_x,distribution="gaussian",n.trees=1000,shrinkage=
summary(boostfit)
```




```
##                       var      rel.inf
## OverallQual   OverallQual 38.643405593
## X1stFlrSF       X1stFlrSF 11.282410895
## LotArea           LotArea  7.287485927
## GarageArea     GarageArea  6.507522872
## BsmtFinSF1     BsmtFinSF1  6.015260929
## YearBuilt       YearBuilt  5.780625989
## Fireplaces     Fireplaces  4.513748042
## FullBath         FullBath  3.597207404
## X2ndFlrSF       X2ndFlrSF  3.324287570
## GarageCars     GarageCars  3.307410792
## YearRemodAdd YearRemodAdd  2.936657057
## OverallCond   OverallCond  1.960690618
## TotRmsAbvGrd TotRmsAbvGrd  1.879884162
## BsmtUnfSF       BsmtUnfSF  0.942233227
## HalfBath         HalfBath  0.510656714
## MoSold             MoSold  0.381871872
## WoodDeckSF     WoodDeckSF  0.262944338
## BedroomAbvGr BedroomAbvGr  0.241348412
## BsmtFullBath BsmtFullBath  0.220444912
## BsmtFinSF2     BsmtFinSF2  0.215657635
## KitchenAbvGr KitchenAbvGr  0.133191571
## LowQualFinSF LowQualFinSF  0.047278127
## BsmtHalfBath BsmtHalfBath  0.007775341
```

```
boostfit
```

```
## gbm(formula = log(train_y$SalePrice) ~ ., distribution = "gaussian",
##     data = train_x, n.trees = 1000, shrinkage = 0.1, cv.folds = 10)
## A gradient boosted model with gaussian loss function.
## 1000 iterations were performed.
## The best cross-validation iteration was 1000.
## There were 23 predictors of which 23 had non-zero influence.
```

```
which.min(boostfit$cv.error)
```

```
## [1] 1000
```

Now we use a boosting model to predict our response variable SalePrice. A boosting model utilizes trees once again, but it averages a bunch of nonbootstrapped trees. This method grows sequentially as opposed to the random forest and bagging models.

Here we see the distribution of influence of the variables. OverallQual again is the top variable.

We use a cross validation method to find the best number of trees for our model. The result of the cross validation indicated that 920 was the best number of trees.

```
boostpreds <- exp(predict(boostfit,newdata=test_x,n.trees=920))
head(boostpreds)
```

```
## [1] 128028.8 160166.5 181043.3 197594.1 180576.2 176057.4
```

These are our first 6 predictions of SalePrice using this model.

Estimated Test Errors and True Test Errors

Saving my data

```
knndata <- knn.reg(train=train_x,test=test_x,y=train_y,k=5)
knndata <- data.frame('Id' = c(1461:2919),'SalePrice'=knndata$pred)
write.csv(knndata,'C:/Users/jacob/Documents/STT 481/knndata.csv')

lindata <- exp(predict(fit2,newdata=test))
lindata <- data.frame('ID' = c(1461:2919),'SalePrice'=lindata)
write.csv(lindata,'C:/Users/jacob/Documents/STT 481/lindata.csv')

subsetlindata <- exp(subset.lindata)
subsetlindata <- data.frame('Id' = c(1461:2919),'SalePrice'=subsetlindata)
write.csv(subsetlindata,'C:/Users/jacob/Documents/STT 481/subsetlindata.csv')

ridgedata <- ridge.pred
ridgedata <- data.frame('Id' = c(1461:2919),'SalePrice'=ridgedata)
write.csv(ridgedata,'C:/Users/jacob/Documents/STT 481/ridgedata.csv')

lassodata <- lasso.pred
lassodata <- data.frame('Id' = c(1461:2919),'SalePrice'=lassodata)
write.csv(lassodata,'C:/Users/jacob/Documents/STT 481/lassodata.csv')
```

```
gamdata <- gampreds
gamdata <- data.frame('ID' = c(1461:2919),'SalePrice'=gamdata)
write.csv(gamdata,'C:/Users/jacob/Documents/STT 481/gamdata.csv')

treedata <- treepreds
treedata <- data.frame('ID' = c(1461:2919),'SalePrice'=treedata)
write.csv(treedata,'C:/Users/jacob/Documents/STT 481/treedata.csv')

bagdata <- bagpreds
bagdata <- data.frame('ID' = c(1461:2919),'SalePrice'=bagdata)
write.csv(bagdata,'C:/Users/jacob/Documents/STT 481/bagdata.csv')

randomforestdata <- randomforestpreds
randomforestdata <- data.frame('ID' = c(1461:2919),'SalePrice'=randomforestdata)
write.csv(randomforestdata,'C:/Users/jacob/Documents/STT 481/randomforestdata.csv')

boostdata <- boostpreds
boostdata <- data.frame('ID' = c(1461:2919),'SalePrice'=boostdata)
write.csv(boostdata,'c:/Users/jacob/Documents/STT 481/boostdata.csv')
```

```
#KNN MSE
error = 0
for(i in 1:nrow(train_x)){
  set.seed(10)
  pred.class <- knn.reg(train_x[-i,], train_x[i,], train_y$SalePrice[-i], k=5)
  error <- error+ (log(train_y$SalePrice[i])-log(pred.class$pred))^2
}
sqrt(error/nrow(train_x))
```

```
## [1] 0.2306037
```

```
#Lin Reg MSE
error = 0
for(i in 1:nrow(train_x)){
  fitmse <- lm(log(train_y$SalePrice[-i])~.,data=train_x[-i,])
  pred <- predict(fitmse,train_x[i,],type='response')
  error <- error + (log(train_y$SalePrice[i])-pred)^2
}
sqrt(error/nrow(train_x))
```

```
##         1
## 0.1586374
```

```
#Subset
error = 0
for(i in 1:nrow(train_x)){
  subsetfit <- lm(log(train_y$SalePrice[-i])~LotArea+OverallQual+OverallCond+YearBuilt+YearRemodAdd+Bsmt
  pred <- predict(subsetfit,train_x[i,])
  error <- error + (log(train_y$SalePrice[i])-pred)^2
}
sqrt(error/nrow(train_x))
```

```
##          1
## 0.1575491
```

```
#Ridge
error <- 0
for(i in 1:nrow(train_x)){

  Xs <- model.matrix(log(train_y$SalePrice[-i])~.,data=train_x[-i,])[,-1]
  ys <- train_y$SalePrice[-i]
  ridgemod <- glmnet(Xs,ys,alpha=0,lambda=grid)

  testx = as.matrix(train_x[i,])
  pred <- predict(ridgemod,s=bestlam,newx=testx)

  if(pred > 0){ #filter out negative predictions
      error <- error + (log(train_y$SalePrice[i])-log(pred))^2
  }
}
sqrt(error/(nrow(train_x)-2))
```

```
##           1
## 1 0.1850845
```

```
#Lasso
error <- 0
for(i in 1:nrow(train_x)){
  set.seed(10)
  Xs <- model.matrix(log(train_y$SalePrice[-i])~.,data=train_x[-i,])[,-1]
  ys <- train_y$SalePrice[-i]
  lassomod <- glmnet(Xs,ys,alpha=1,lambda=grid)

  testx = as.matrix(train_x[i,])
  pred <- predict(lassomod,s=bestlam2,newx=testx)

  if(pred > 0){ #filter out negative predictions
      error <- error + (log(train_y$SalePrice[i])-log(pred))^2
  }
}
sqrt(error/(nrow(train_x)-2))
```

```
##           1
## 1 0.1872023
```

```
#GAM
error = 0
for(i in 1:nrow(train_x)){
  gamfit <- gam(log(train_y$SalePrice[-i])~s(LotArea) + s(OverallQual) + s(OverallCond) + s(YearBuilt) +

  pred <- predict(gamfit,train_x[i,])
  error <- error + (log(train_y$SalePrice[i])-pred)^2
}
sqrt(error/nrow(train_x))
```

```
##        1
## 0.130365
```

```r
#Regression Tree
error = 0
for (i in 1:nrow(train_x)){
  set.seed(10)
  treefit <- tree(log(train_y$SalePrice[-i]) ~ ., data=train_x[-i,])

  pred <- predict(treefit,train_x[i,])
  error <- error + (log(train_y$SalePrice[i])-pred)^2
}
sqrt(error/nrow(train_x))
```

```
##         1
## 0.2219954
```

```r
#Bagging
error = 0
for (i in 1:nrow(train_x)){
  set.seed(10)
  bagfit <- randomForest(log(train_y$SalePrice[-i])~.,data=train_x[-i,],mtry=ncol(train_x)-1,importance=
  pred <- predict(bagfit,train_x[i,])
  error <- error + (log(train_y$SalePrice[i])-pred)^2
}
sqrt(error/nrow(train_x))
```

```
##         1
## 0.1470316
```

```r
#Random Forest
error = 0
for (i in 1:nrow(train_x)){
  set.seed(10)
  randomforestfit <- randomForest(log(train_y$SalePrice[-i])~.,data=train_x[-i,],mtry=round(sqrt(ncol(t
  pred <- predict(randomforestfit,train_x[i,])
  error <- error + (log(train_y$SalePrice[i])-pred)^2
}
sqrt(error/nrow(train_x))
```

```
##         1
## 0.1427294
```

```r
#Boosting
error = 0
for (i in 1:nrow(train_x)){
  set.seed(10)
  boostfit <- gbm(log(train_y$SalePrice[-i])~.,data=train_x[-i,],distribution="gaussian",n.trees=100,sh
  pred <- predict(boostfit,train_x[i,])
  error <- error + (log(train_y$SalePrice[i])-pred)^2
}
```

```
## Using 100 trees...
##
## Using 100 trees...
```

```
sqrt(error/nrow(train_x))
```

```
## [1] 0.1562022
```

Based on the CV estimates of each model, I believe that the model that will return the lowest test MSe will be the GAM model. This model returned a 0.13 estimated test MSE, which is the lowest of all the methods.

```
MSEs <- data.frame("Method" = c("KNN","Linear Reg","Subset Linear Reg", "Ridge","Lasso","GAM","Regressi
MSEs
```

```
##                Method   MSE True.MSE
## 1                 KNN 0.231    0.249
## 2          Linear Reg 0.159    0.151
## 3  Subset Linear Reg 0.158    0.151
## 4               Ridge 0.186    0.222
## 5               Lasso 0.264    0.454
## 6                 GAM 0.130    0.134
## 7     Regression Tree 0.222    0.228
## 8             Bagging 0.146    0.153
## 9       Random Forest 0.144    0.148
## 10           Boosting 0.156    0.143
```

Here, we calculated our Mean squared errors. We used the same method kaggle did, sqrt(log of observed-log of predicted)^2/n. Looking at these MSE, they are very similar to the results we received through kaggle, except for the lasso regression. This did not perform well in kaggle, but performed very well in our MSE Cross Validation algorithm.

Of the true test MSEs, we find that GAM had the lowest. This is consistent with our initial prediction based on the estimate test MSEs.

Based on our true test MSEs from Kaggle, we can try to answer why each model performed the way that it did. We see that, with the exception of our inconsistent lasso result, KNN resulted in the worst test MSE. This may be due to how KNN is very sensitive to the quality and the scale of the data. This is the only method in which we did not include a logarithm scale on our response variable.

We can also speculate as to why the GAM was the best performing model. We know that GAM works better with many predictors, which is great for this model because we have over 20 predictors. We also know that it is more flexible than other predictive regression models such as linear regression. This is consistent with our results because GAM performed better than the linear regression model.

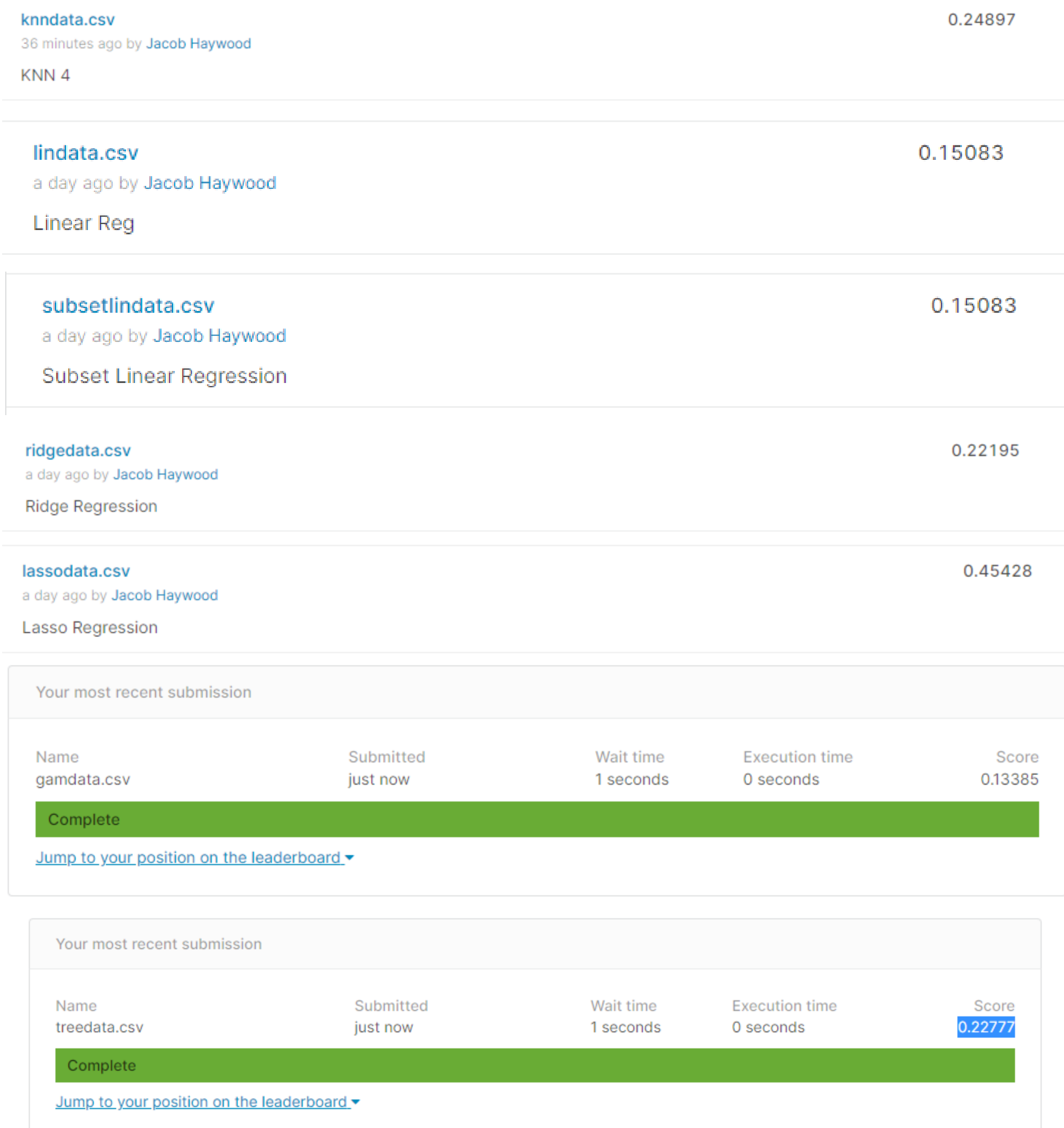
As for the other methods, one algorithm is not inherently better than all others. Each method has their own pros and cons, and even the best performing algorithms may have consequences in computation time and cost. Thus, with different data, we may see that a different method performed better. Also, because our training data was picked randomly, we may see different results with a different training data from the same overall dataset.

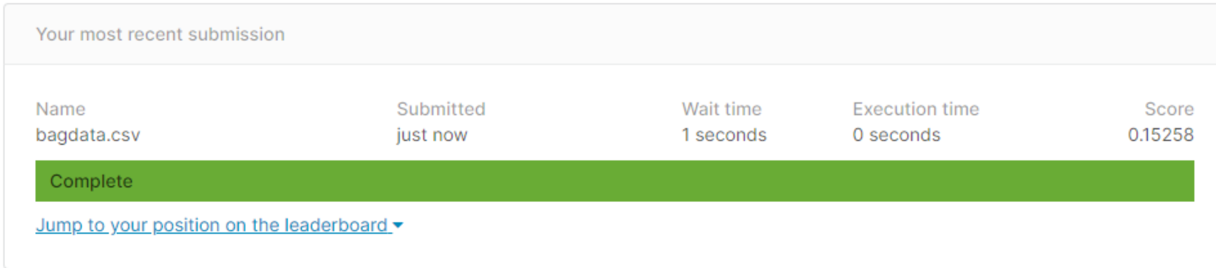
In conclusion, we ran 9 different algorithms: KNN, Linear Regression, Subsetted Linear Regression, Shrinkage (Ridge/Lasso), GAM, Regression Tree, Bagging, Random Forest, and Boosting. We used cross validation to optimize the parameters of each model and used those models to predict the sale price of houses. The test MSE was estimated using LOOCV for all the methods and then the test MSE was actually calculated through Kaggle. The best performing method for both the LOOCV and Kaggle was GAM. This may be due to GAM's flexibility over other models.

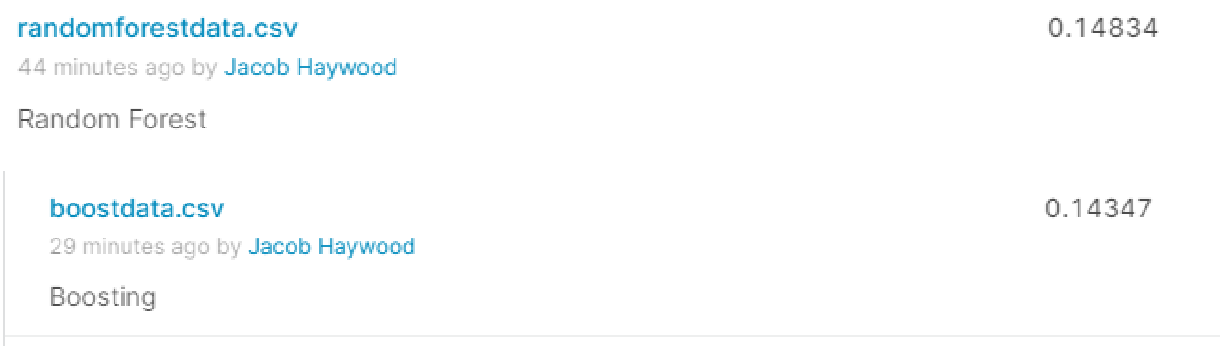
This process raises further questions on the data that can be later explored as well. For example, it is assumed that with better cleaning of the data, we can reach better test MSE values. Instead of extracting NA values, we could estimate the NA values using a median/mean. This may produce better results.

Another question that could be explored is how the results of these algorithms compare to a similar data set but for a different state. It is known that this data set comes from Iowa, but it is unknown how this data may compare to a different state. With the same variables, we can run these same algorithms to see which performs the best.

Below are the screenshots of our Kaggle results.




Your most recent submission
Name
bagdata.csv
Submitted
just now
Wait time
1 seconds
Execution time
0 seconds
Score
0.15258
Complete
Jump to your position on the leaderboard


randomforestdata.csv
0.14834
44 minutes ago by Jacob Haywood
Random Forest
boostdata.csv
0.14347
29 minutes ago by Jacob Haywood
Boosting