STT481Final

2022-12-04

R Markdown

My whole goal of this project is to predict the final price of each home using explanatory variables that pretty much describe every aspect of the residential homes in Ames, Iowa. I use 9 different statistical learning methods that we've learned in class to get my predicted results.

The raw data has 79 explanatory variables, categorical and numerical. I decided to use the given processed data though, which has no missing values and removed categorical variables. Size of the house, quality of different parts of the house, and number of different furniture are some examples of the explanatory variables in the data set.

The data has been split into 2 different data sets, test and training. The training set is the set of data that is used to train and make the model learn the hidden features/patterns in the data. The test set is a separate set of data used to test the model after completing the training. It provides an unbiased final model performance metric in terms of accuracy, precision, etc. To put it simply, it answers the question of "How well does the model perform?"

Here is a preview of the training and test data sets, and the explanatory variables. The response variable is SalePrice.

```
train <- read.csv('train_new.csv')
test <- read.csv('test_new.csv')
head(train)
```

```
##   LotArea OverallQual OverallCond YearBuilt YearRemodAdd BsmtFinSF1 BsmtFinSF2
## 1    8450           7           5      2003         2003        706          0
## 2    9600           6           8      1976         1976        978          0
## 3   11250           7           5      2001         2002        486          0
## 4    9550           7           5      1915         1970        216          0
## 5   14260           8           5      2000         2000        655          0
## 6   14115           5           5      1993         1995        732          0
##   BsmtUnfSF X1stFlrSF X2ndFlrSF LowQualFinSF BsmtFullBath BsmtHalfBath FullBath
## 1       150       856       854            0            1            0        2
## 2       284      1262         0            0            0            1        2
## 3       434       920       866            0            1            0        2
## 4       540       961       756            0            1            0        1
## 5       490      1145      1053            0            1            0        2
## 6        64       796       566            0            1            0        1
##   HalfBath BedroomAbvGr KitchenAbvGr TotRmsAbvGrd Fireplaces GarageCars
## 1        1            3            1            8          0          2
## 2        0            3            1            6          1          2
## 3        1            3            1            6          1          2
## 4        0            3            1            7          1          3
## 5        1            4            1            9          1          3
## 6        1            1            1            5          0          2
##   GarageArea WoodDeckSF MoSold SalePrice
## 1        548          0      2    208500
## 2        460        298      5    181500
## 3        608          0      9    223500
## 4        642          0      2    140000
## 5        836        192     12    250000
## 6        480         40     10    143000
```

```
head(test)
```

```
##   LotArea OverallQual OverallCond YearBuilt YearRemodAdd BsmtFinSF1 BsmtFinSF2
## 1   11622           5           6      1961         1961        468        144
## 2   14267           6           6      1958         1958        923          0
## 3   13830           5           5      1997         1998        791          0
## 4    9978           6           6      1998         1998        602          0
## 5    5005           8           5      1992         1992        263          0
## 6   10000           6           5      1993         1994          0          0
##   BsmtUnfSF X1stFlrSF X2ndFlrSF LowQualFinSF BsmtFullBath BsmtHalfBath FullBath
## 1       270       896         0            0            0            0        1
## 2       406      1329         0            0            0            0        1
## 3       137       928       701            0            0            0        2
## 4       324       926       678            0            0            0        2
## 5      1017      1280         0            0            0            0        2
## 6       763       763       892            0            0            0        2
##   HalfBath BedroomAbvGr KitchenAbvGr TotRmsAbvGrd Fireplaces GarageCars
## 1        0            2            1            5          0          1
## 2        1            3            1            6          0          1
## 3        1            3            1            6          1          2
## 4        1            3            1            7          1          2
## 5        0            2            1            5          0          2
## 6        1            3            1            7          1          2
##   GarageArea WoodDeckSF MoSold SalePrice
## 1        730        140      6         0
## 2        312        393      6         0
## 3        482        212      3         0
## 4        470        360      6         0
## 5        506          0      1         0
## 6        440        157      4         0
```

Dimensions of each of the data sets

```
dim(test)
```

```
## [1] 1459   24
```

```
dim(train)
```

```
## [1] 1460   24
```

ID variable (will be used when predicting SalePrice)

```
ID <- seq(1461, 2919, 1)
```

KNN Method

standardization is needed for KNN Method, so the first step for KNN regression is to scale all of the independent variables to improve the accuracy of the model.

```
train.x <- train[,-24]
train.x.scale <- scale(train.x)
head(train.x.scale)
```

```
##          LotArea OverallQual OverallCond  YearBuilt YearRemodAdd  BsmtFinSF1
## [1,] -0.20707076  0.65125610  -0.5170227  1.0506338    0.8783671  0.57522774
## [2,] -0.09185490 -0.07181151   2.1788812  0.1566800   -0.4294298  1.17159068
## [3,]  0.07345481  0.65125610  -0.5170227  0.9844150    0.8299302  0.09287536
## [4,] -0.09686428  0.65125610  -0.5170227 -1.8629933   -0.7200514 -0.49910256
## [5,]  0.37501979  1.37432370  -0.5170227  0.9513056    0.7330564  0.46340969
## [6,]  0.36049258 -0.79487911  -0.5170227  0.7195398    0.4908717  0.63223302
##      BsmtFinSF2   BsmtUnfSF   X1stFlrSF  X2ndFlrSF LowQualFinSF BsmtFullBath
## [1,]  -0.288554 -0.94426706 -0.79316202  1.1614536   -0.1202005    1.1074307
## [2,]  -0.288554 -0.64100836  0.25705235 -0.7948909   -0.1202005   -0.8196835
## [3,]  -0.288554 -0.30153966 -0.62761099  1.1889432   -0.1202005    1.1074307
## [4,]  -0.288554 -0.06164845 -0.52155486  0.9369551   -0.1202005    1.1074307
## [5,]  -0.288554 -0.17480468 -0.04559563  1.6173231   -0.1202005    1.1074307
## [6,]  -0.288554 -1.13889578 -0.94836612  0.5017028   -0.1202005    1.1074307
##      BsmtHalfBath  FullBath   HalfBath BedroomAbvGr KitchenAbvGr TotRmsAbvGrd
## [1,]   -0.2409785  0.789470  1.2271649     0.163723   -0.2113812    0.9118973
## [2,]    3.9474568  0.789470 -0.7613598     0.163723   -0.2113812   -0.3185741
## [3,]   -0.2409785  0.789470  1.2271649     0.163723   -0.2113812   -0.3185741
## [4,]   -0.2409785 -1.025689 -0.7613598     0.163723   -0.2113812    0.2966616
## [5,]   -0.2409785  0.789470  1.2271649     1.389547   -0.2113812    1.5271330
## [6,]   -0.2409785 -1.025689  1.2271649    -2.287924   -0.2113812   -0.9338098
##      Fireplaces GarageCars  GarageArea WoodDeckSF     MoSold
## [1,] -0.9509007  0.3116179  0.35088009 -0.7519182 -1.5985634
## [2,]  0.6002892  0.3116179 -0.06071021  1.6256378 -0.4889425
## [3,]  0.6002892  0.3116179  0.63150985 -0.7519182  0.9905519
## [4,]  0.6002892  1.6497417  0.79053338 -0.7519182 -1.5985634
## [5,]  0.6002892  1.6497417  1.69790291  0.7799299  2.1001728
## [6,] -0.9509007  0.3116179  0.03283304 -0.4327832  1.3604256
```

The next step is to choose the turning parameter, K, by using Cross validation. I chose to use k-fold Cross Validation to choose my K value. The K value with the lowest MSE is the value I will use.

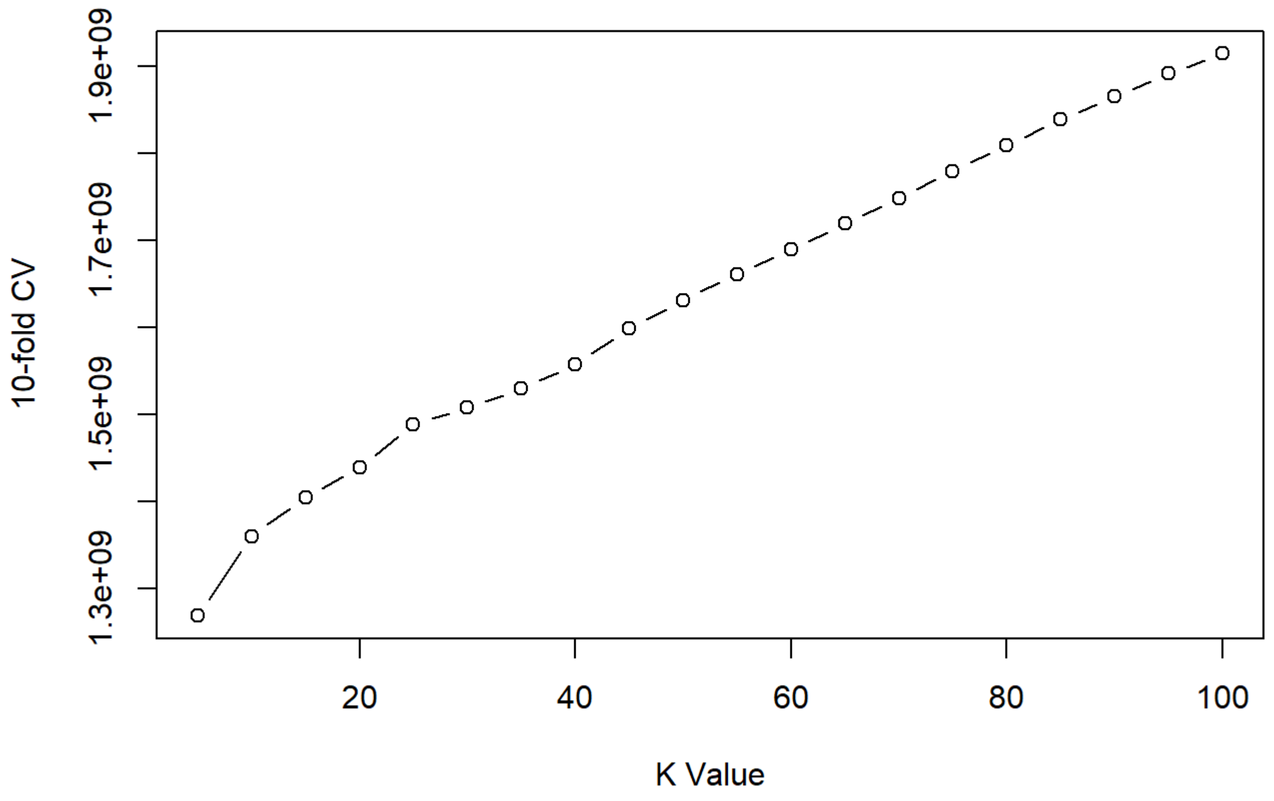
```
library(FNN)

set.seed(100)
fold.index <- cut(sample(1:nrow(train.x.scale)), breaks=10, labels=FALSE)

k.vt <- seq(5,100,5)
error.k <- rep(0, length(k.vt))
counter <- 0

for(k in k.vt){
  counter <- counter+1
  mse <- rep(0,10)
  for(i in 1:10){
    pred.out <- knn.reg(train.x.scale[fold.index!=i,], train.x.scale[fold.index==i,], train$Sale
Price[fold.index!=i], k=k)
    mse[i] <- mean((pred.out$pred - train$SalePrice[fold.index==i])^2)
  }
  error.k[counter] <- sum(mse)/10
}

plot(k.vt, error.k, xlab='K Value', ylab='10-fold CV', type='b')
```



The K value with the smallest CV MSE is the value I will use for my KNN prediction.

```
k.vt[which.min(error.k)]
```

```
## [1] 5
```

Scaling the independent variables on the test data

```
test.x <- test[,-24]
test.x.scale <- scale(test.x)
head(test.x.scale)
```

```
##          LotArea OverallQual OverallCond  YearBuilt YearRemodAdd  BsmtFinSF1
## [1,]  0.36380438 -0.75084380   0.4006287 -0.3408277   -1.0725169  0.06326966
## [2,]  0.89755290 -0.05485835   0.4006287 -0.4395442   -1.2144920  1.06302381
## [3,]  0.80936836 -0.75084380  -0.4972473  0.8437697    0.6785093  0.77298524
## [4,]  0.03205295 -0.05485835   0.4006287  0.8766752    0.6785093  0.35770275
## [5,] -0.97147497  1.33711256  -0.4972473  0.6792423    0.3945591 -0.38716902
## [6,]  0.03649244 -0.05485835  -0.4972473  0.7121478    0.4892091 -0.96504889
##      BsmtFinSF2  BsmtUnfSF  X1stFlrSF  X2ndFlrSF LowQualFinSF BsmtFullBath
## [1,]  0.5171828 -0.6504061 -0.6543370 -0.7749878  -0.08045553   -0.8192752
## [2,] -0.2977886 -0.3392720  0.4331497 -0.7749878  -0.08045553   -0.8192752
## [3,] -0.2977886 -0.9546770 -0.5739684  0.8916384  -0.08045553   -0.8192752
## [4,] -0.2977886 -0.5268676 -0.5789915  0.8369559  -0.08045553   -0.8192752
## [5,] -0.2977886  1.0585438  0.3100853 -0.7749878  -0.08045553   -0.8192752
## [6,] -0.2977886  0.4774551 -0.9883686  1.3457405  -0.08045553   -0.8192752
##      BsmtHalfBath   FullBath   HalfBath BedroomAbvGr KitchenAbvGr TotRmsAbvGrd
## [1,]   -0.2584165 -1.0283671 -0.7507821   -1.0291897     -0.20384   -0.9180200
## [2,]   -0.2584165 -1.0283671  1.2372235    0.1759369     -0.20384   -0.2552831
## [3,]   -0.2584165  0.7728185  1.2372235    0.1759369     -0.20384   -0.2552831
## [4,]   -0.2584165  0.7728185  1.2372235    0.1759369     -0.20384    0.4074537
## [5,]   -0.2584165  0.7728185 -0.7507821   -1.0291897     -0.20384   -0.9180200
## [6,]   -0.2584165  0.7728185  1.2372235    0.1759369     -0.20384    0.4074537
##      Fireplaces GarageCars  GarageArea WoodDeckSF     MoSold
## [1,] -0.8977474 -0.9876745  1.18553776  0.3665526 -0.0382676
## [2,] -0.8977474 -0.9876745 -0.74095888  2.3470625 -0.0382676
## [3,]  0.6468439  0.3015187  0.04254454  0.9301760 -1.1402235
## [4,]  0.6468439  0.3015187 -0.01276158  2.0887351 -0.0382676
## [5,] -0.8977474  0.3015187  0.15315679 -0.7293817 -1.8748608
## [6,]  0.6468439  0.3015187 -0.15102689  0.4996304 -0.7729049
```

Using KNN method to predict SalePrice based off the data

```
knn.reg <- knn.reg(train=train.x.scale, test=test.x.scale, y=train$SalePrice, k=5)
knn.predict <- knn.reg$pred
knn.final <- cbind(ID, knn.predict)
head(knn.final)
```

```
##        ID knn.predict
## [1,] 1461      118780
## [2,] 1462      152180
## [3,] 1463      179200
## [4,] 1464      177700
## [5,] 1465      173967
## [6,] 1466      187400
```

Linear Regression

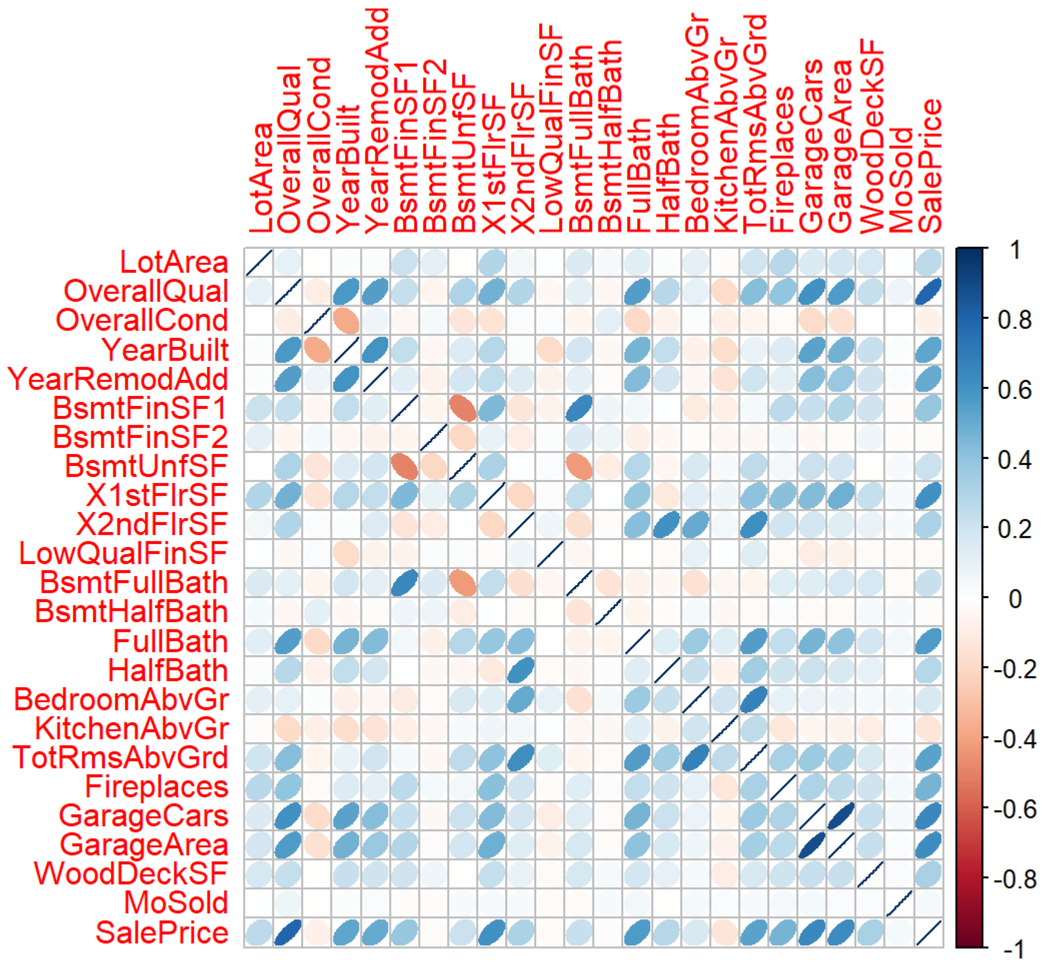
First step when making a linear regression model is to make sure that the explanatory variables don’t have a correlation with each other. They should all be independent.

```
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.2.2
```

```
## corrplot 0.92 loaded
```

```
cor <- cor(train)
corrplot(cor, method='ellipse')
```



Now I start by using the lm() function to fit a multiple linear regression model, with SalePrice as the response and all other variables as the predictors.

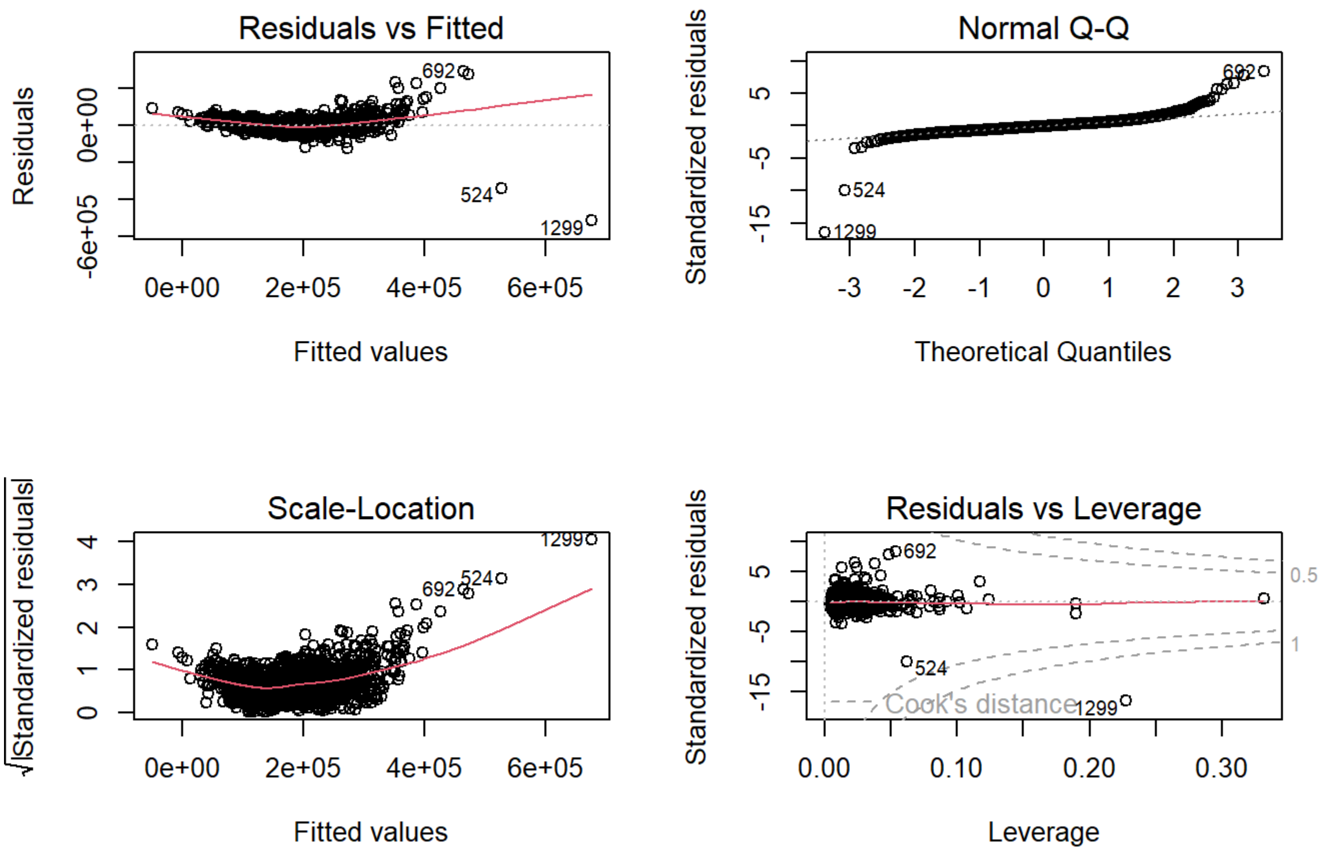
```
lm.sales <- lm(SalePrice~., data=train)
summary(lm.sales)
```

```
##
## Call:
## lm(formula = SalePrice ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -515526  -17128   -2129   13537  290610
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -9.410e+05  1.329e+05  -7.079 2.26e-12 ***
## LotArea       4.416e-01  1.026e-01   4.305 1.79e-05 ***
## OverallQual   1.730e+04  1.199e+03  14.424  < 2e-16 ***
## OverallCond   4.737e+03  1.037e+03   4.569 5.32e-06 ***
## YearBuilt     3.121e+02  5.842e+01   5.342 1.07e-07 ***
## YearRemodAdd  1.292e+02  6.707e+01   1.926 0.054286 .
## BsmtFinSF1    2.319e+01  4.723e+00   4.912 1.01e-06 ***
## BsmtFinSF2    9.958e+00  7.178e+00   1.387 0.165565
## BsmtUnfSF     1.276e+01  4.262e+00   2.994 0.002805 **
## X1stFlrSF     5.171e+01  5.837e+00   8.860  < 2e-16 ***
## X2ndFlrSF     4.313e+01  4.867e+00   8.861  < 2e-16 ***
## LowQualFinSF  1.480e+01  2.005e+01   0.738 0.460532
## BsmtFullBath  7.064e+03  2.648e+03   2.667 0.007732 **
## BsmtHalfBath  1.253e+03  4.174e+03   0.300 0.764104
## FullBath      2.500e+03  2.865e+03   0.873 0.383034
## HalfBath     -5.415e+02  2.709e+03  -0.200 0.841604
## BedroomAbvGr -9.028e+03  1.716e+03  -5.260 1.66e-07 ***
## KitchenAbvGr -2.524e+04  4.955e+03  -5.095 3.96e-07 ***
## TotRmsAbvGrd  6.007e+03  1.252e+03   4.798 1.77e-06 ***
## Fireplaces    4.130e+03  1.794e+03   2.302 0.021482 *
## GarageCars    1.102e+04  2.909e+03   3.787 0.000159 ***
## GarageArea    5.430e+00  9.861e+00   0.551 0.581942
## WoodDeckSF    2.125e+01  8.036e+00   2.644 0.008278 **
## MoSold        5.284e+01  3.480e+02   0.152 0.879328
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 35640 on 1436 degrees of freedom
## Multiple R-squared:  0.8019, Adjusted R-squared:  0.7987
## F-statistic: 252.7 on 23 and 1436 DF,  p-value: < 2.2e-16
```

Looking at the p-values, you can see that there are many significant coefficients to the model. The more stars a variable has (*), along with the smaller p-value a variable has, the more significant it is. Looking at both of those, you can see that OverallQual, X1stFlrSF, and X2ndFlrSF are the most significant variables. Multiple R-Squared value (indicates how well a regression model predicts responses for new observations) = 0.8019 = strong model

All variables have a positive effect with SalePrice, besides HalfBath, BedroomAbvGr, and KitchenAbvGr. Estimates say that as the variable increases by 1 unit, SalePrice increases (or for a few cases, decreases) by that estimate. ex: As the number of full bathrooms above ground increases by 1, SalePrice increases by $2,500.

```
par(mfrow=c(2,2))
plot(lm.sales)
```



From the Residuals vs Fitted plot, the constant variance assumption looks fine. From the Normal Q-Q plot, most of points are located on the straight line. Although there seems to be a little bit of a long-tailed distribution, the distribution seems not to be far away from normal distribution. From the Residual vs Leverage plot, observation 1299 has large Cook's distance, so I will identify this observation as large leverage point

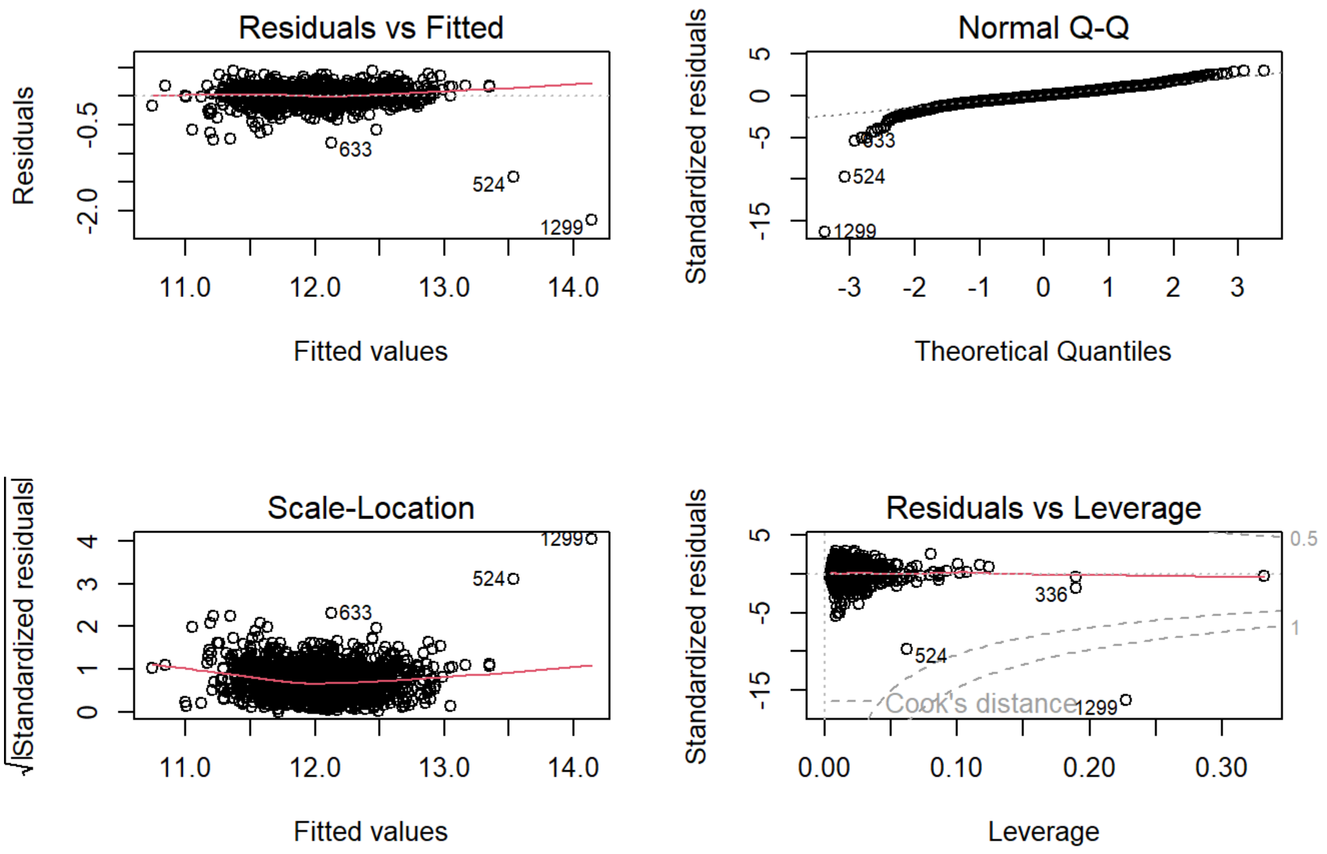
I wanted to add an effect to my model to see if the results would be better. I chose to add log to the response variable.

```
lm.sales.log <- lm(log(SalePrice)~., data=train)
summary(lm.sales.log)
```

```
##
## Call:
## lm(formula = log(SalePrice) ~ ., data = train)
##
## 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
```

You can see that the multiple R-Squared value is actually higher in this model than the previous model. KitchenAbvGr is now the only variable that has a negative effect on SalePrice.

```
par(mfrow=c(2,2))
plot(lm.sales.log)
```



We can see that the plots look much more linear and a little bit more normal and also decreased the error.

```
lm.predict <- predict(lm.sales.log, newdata=test)
lm.final <- cbind(ID, exp(lm.predict))
head(lm.final)
```

```
##     ID
## 1 1461 113397.3
## 2 1462 146077.0
## 3 1463 168281.6
## 4 1464 196719.1
## 5 1465 182239.9
## 6 1466 175746.7
```

Subset Selection

I will now perform Subset Selection Method. I am choosing to do Forward Stepwise Selection. Forward and Backward Stepwise are pretty similar, but I chose forward just based off personal preference. I also did not want to deal with Best Subset Selection.

```
library(leaps)
```

```
## Warning: package 'leaps' was built under R version 4.2.2
```

```
regfit.fwd <- regsubsets(SalePrice ~., data=train, nvmax = ncol(train)-1, method="forward")
reg.fwd.summary <- summary(regfit.fwd)
```

Now I will plot RSS, adjusted R2, Cp, and BIC for all of the models selected by Forward Stepwise Selection.

```
which.max(reg.fwd.summary$adjr2)
```

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

```
which.min(reg.fwd.summary$cp)
```

```
## [1] 16
```

```
which.min(reg.fwd.summary$bic)
```

```
## [1] 13
```

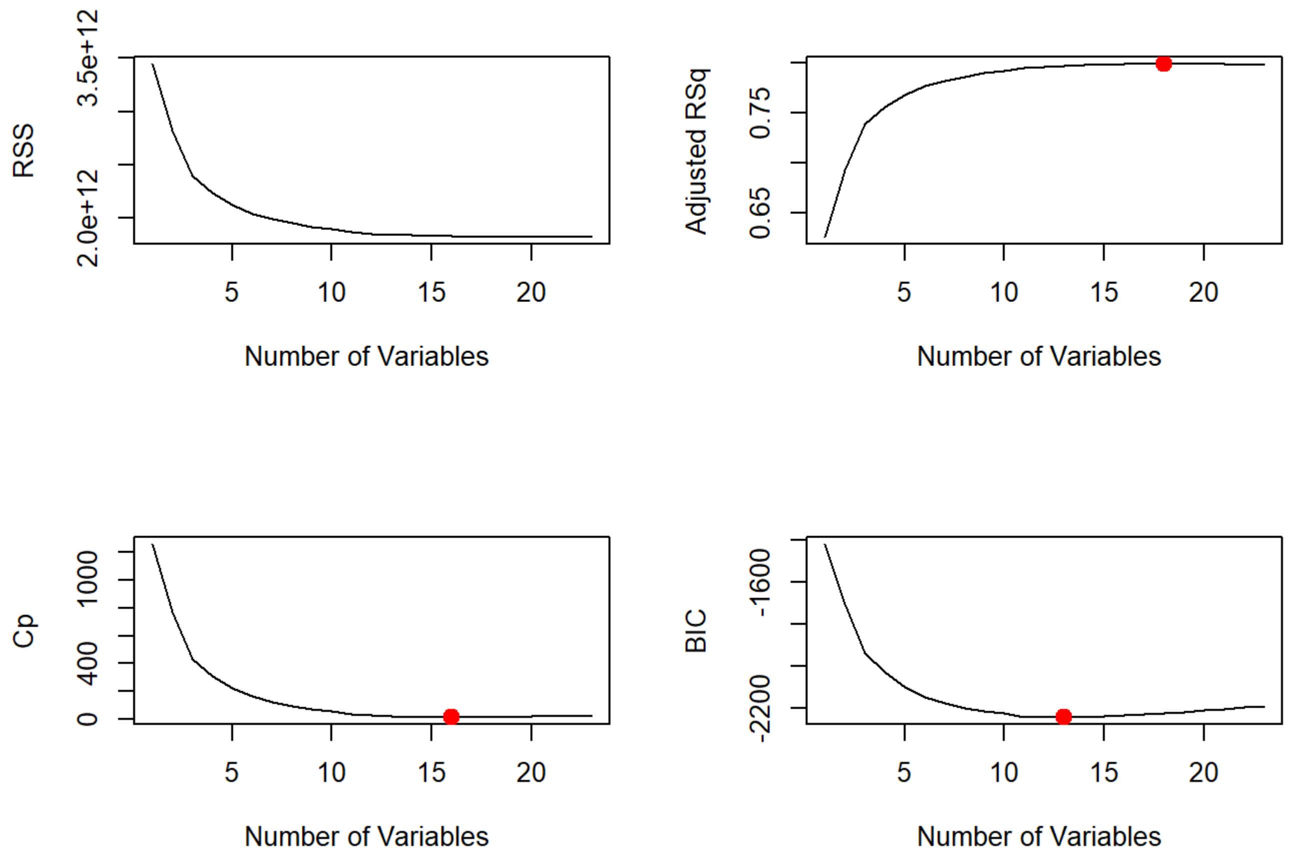
```
par(mfrow=c(2,2))

plot(reg.fwd.summary$rss ,xlab="Number of Variables ", ylab="RSS", type="l")

plot(reg.fwd.summary$adjr2 ,xlab="Number of Variables ", ylab="Adjusted RSq", type="l")
points(18, reg.fwd.summary$adjr2[18], col = "red", cex = 2, pch = 20)

plot(reg.fwd.summary$cp, xlab = "Number of Variables ", ylab = "Cp", type = "l")
points(16,reg.fwd.summary$cp[16], col = "red", cex = 2, pch = 20)

plot(reg.fwd.summary$bic, xlab = "Number of Variables ", ylab = "BIC", type = "l")
points(13,reg.fwd.summary$bic[13], col = "red", cex = 2, pch = 20)
```



I will use BIC to evaluate the model because I like a smaller model and it will be easier to interpret. BIC criterion indicates it selects 13 predictors

```
coef(regfit.fwd, 13)
```

```
##   (Intercept)       LotArea   OverallQual   OverallCond     YearBuilt
## -8.100743e+05  4.779045e-01  1.888382e+04  5.248841e+03  3.719968e+02
##    BsmtFinSF1     X1stFlrSF     X2ndFlrSF  BsmtFullBath  BedroomAbvGr
##  1.260681e+01  6.478192e+01  4.488349e+01  6.754390e+03 -9.294672e+03
##  KitchenAbvGr  TotRmsAbvGrd    GarageCars    WoodDeckSF
## -2.746996e+04  6.199982e+03  1.231648e+04  2.208571e+01
```

Few examples of some explanations of the coefficients: An increase of 1 square foot in lot size indicates an increase in SalePrice of $0.48 An increase of 1 in overall quality rating indicates an increase in SalePrice of $18,883.82 An increase of 1 in overall condition rating indicates an increase in SalePrice of $5,248.84 An increase of 1 year in year built indicates an increase in SalePrice of $371.99

Because there is no predict() method for regsubsets(), I have to write my own predict method

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

```
fwd.predict <- predict.regsubsets(regfit.fwd, newdata=test, id=13)
fwd.final <- cbind(ID, fwd.predict)
head(fwd.final)
```

```
##     ID
## 1 1461 115171.4
## 2 1462 170482.9
## 3 1463 172790.0
## 4 1464 201377.8
## 5 1465 206460.0
## 6 1466 181252.4
```

Ridge Regression

I will use the glmnet package in order to perform ridge regression and the lasso. I must pass in an x matrix as well as a y vector, and I do not use the y ~ x syntax. In order to do so, I use model.matrix() to create x, which not only produces a matrix corresponding to the 23 predictors but it also automatically transforms any qualitative variables into dummy variables.

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.2.2
```

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

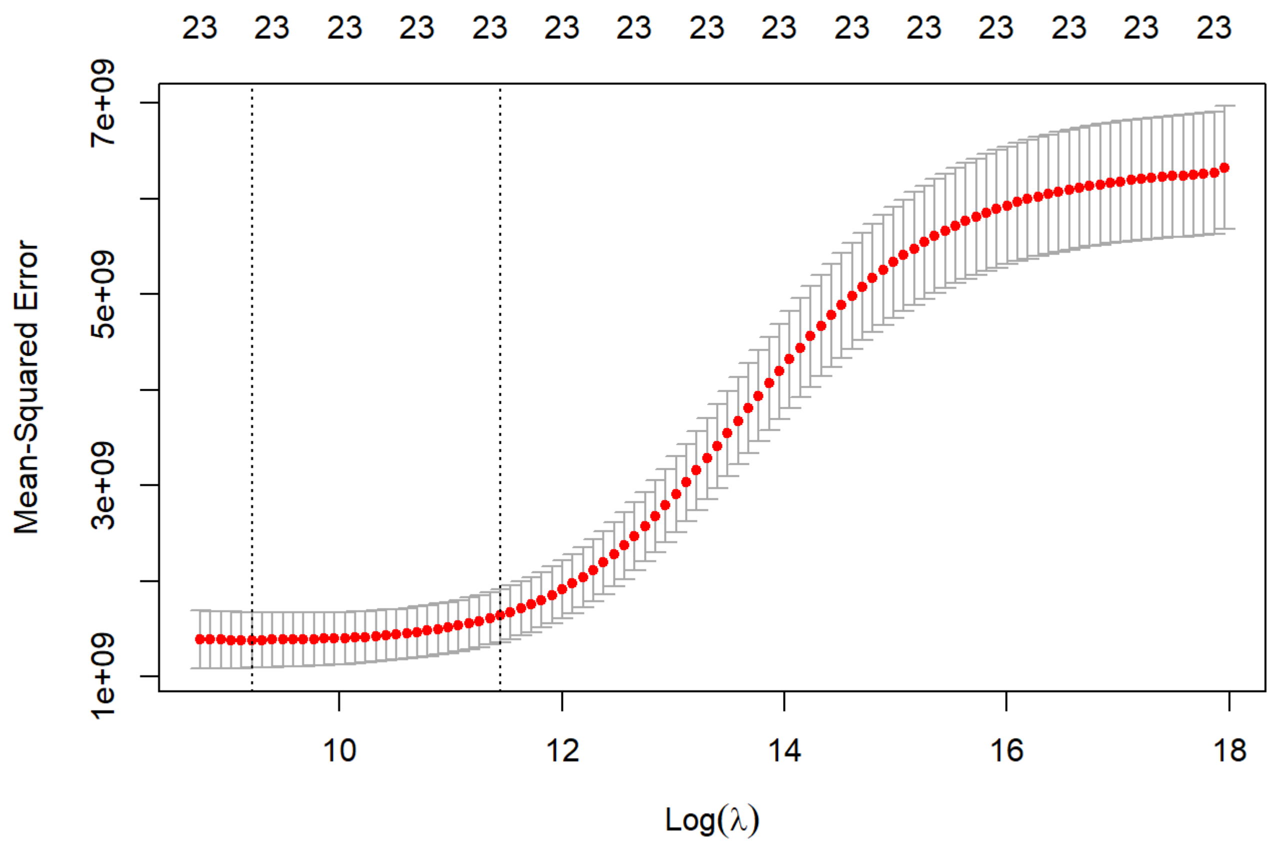
```
## Loaded glmnet 4.1-4
```

```
shrink.x <- model.matrix(SalePrice~.,data=train)[,-1]
shrink.y <- train$SalePrice
```

```
ridge.mod <- glmnet(shrink.x, shrink.y, alpha=0)
```

I now use cross-validation to choose the tuning parameter λ.

```
cv.out.ridge <- cv.glmnet(shrink.x, shrink.y, alpha=0, nfolds=10)
plot(cv.out.ridge)
```



```
bestlam.ridge <- cv.out.ridge$lambda.min
bestlam.ridge
```

```
## [1] 10002.09
```

I examine the coefficient estimates when λ = bestlam

```
coef(ridge.mod, s=bestlam.ridge)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  -9.490476e+05
## LotArea       4.326608e-01
## OverallQual   1.560618e+04
## OverallCond   3.480872e+03
## YearBuilt     2.231208e+02
## YearRemodAdd  2.267367e+02
## BsmtFinSF1    2.177777e+01
## BsmtFinSF2    7.842628e+00
## BsmtUnfSF     1.155595e+01
## X1stFlrSF     3.870415e+01
## X2ndFlrSF     2.682269e+01
## LowQualFinSF  1.025446e+01
## BsmtFullBath  7.380379e+03
## BsmtHalfBath  8.963956e+02
## FullBath      8.103571e+03
## HalfBath      4.299686e+03
## BedroomAbvGr -5.681446e+03
## KitchenAbvGr -2.424981e+04
## TotRmsAbvGrd  6.229496e+03
## Fireplaces    7.113715e+03
## GarageCars    8.993603e+03
## GarageArea    2.190673e+01
## WoodDeckSF    2.507095e+01
## MoSold        9.003405e+01
```

As expected, none of the coefficients are zero—ridge regression does not perform variable selection. Few examples of some explanations of the coefficients: An increase of 1 square foot in lot size indicates an increase in SalePrice of $0.43 An increase of 1 in overall quality rating indicates an increase in SalePrice of $15,606.18 An increase of 1 in overall condition rating indicates an increase in SalePrice of $3,480.87 An increase of 1 year in year built indicates an increase in SalePrice of $223.12

```
test.shrink.x <- model.matrix(SalePrice ~., test)[,-1]
ridge.predict <- predict(ridge.mod, s=bestlam.ridge, newx=test.shrink.x)
ridge.final <- cbind(ID, ridge.predict)
head(ridge.final)
```

```
##     ID       s1
## 1 1461 118861.8
## 2 1462 163406.5
## 3 1463 182327.8
## 4 1464 207269.2
## 5 1465 199413.0
## 6 1466 187242.7
```

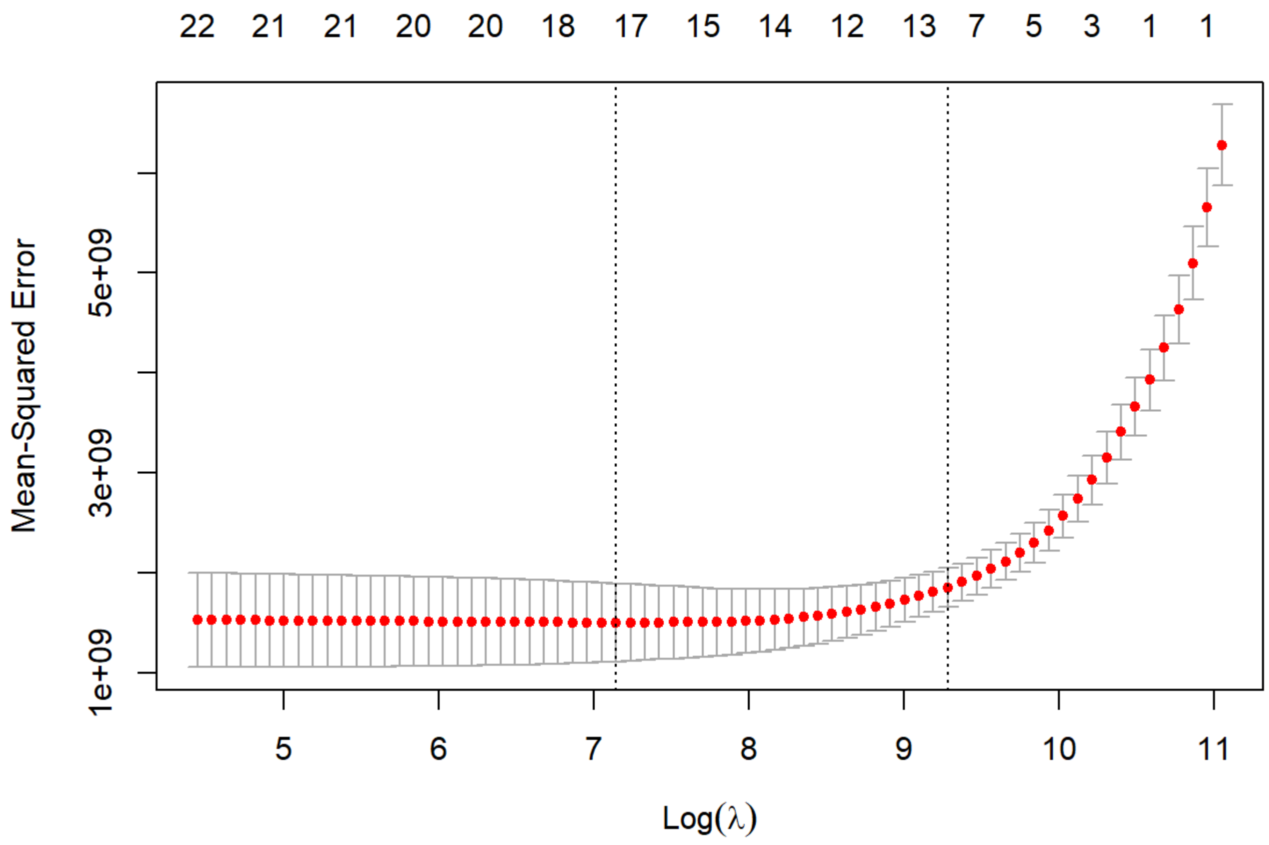
Lasso

For lasso, it’s the same idea as ridge regression. I fit a lasso model on the training set and predict the salaries using the fitted model. Except this time i will use the argument alpha=1

```
lasso.mod <- glmnet(shrink.x, shrink.y, alpha=1)
```

Again, I use cross-validation to choose the tuning parameter λ.

```
cv.out.lasso <- cv.glmnet(shrink.x, shrink.y, alpha=1)
plot(cv.out.lasso)
```



```
bestlam.lasso <- cv.out.lasso$lambda.min
bestlam.lasso
```

```
## [1] 1262.12
```

I examine the coefficient estimates when λ = bestlam

```
coef(lasso.mod, s = bestlam.lasso)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  -9.072808e+05
## LotArea       3.908083e-01
## OverallQual   2.005457e+04
## OverallCond   2.053073e+03
## YearBuilt     2.203162e+02
## YearRemodAdd  2.043323e+02
## BsmtFinSF1    1.445949e+01
## BsmtFinSF2    .
## BsmtUnfSF     .
## X1stFlrSF     5.425823e+01
## X2ndFlrSF     3.463841e+01
## LowQualFinSF  .
## BsmtFullBath  4.287642e+03
## BsmtHalfBath  .
## FullBath      8.135727e+02
## HalfBath      .
## BedroomAbvGr -2.942177e+03
## KitchenAbvGr -1.944368e+04
## TotRmsAbvGrd  4.169650e+03
## Fireplaces    4.362425e+03
## GarageCars    1.014158e+04
## GarageArea    1.046881e+01
## WoodDeckSF    2.013069e+01
## MoSold        .
```

Many of the coefficient estimates with the best λ are exactly zero because they've been forced to 0 by the model. An increase of 1 square foot in lot size indicates an increase in SalePrice of $0.39 An increase of 1 in overall quality rating indicates an increase in SalePrice of $20,054.57 An increase of 1 in overall condition rating indicates an increase in SalePrice of $2,053.07 An increase of 1 year in year built indicates an increase in SalePrice of $220.31

```
lasso.predict <- predict(lasso.mod, s=bestlam.lasso, newx=test.shrink.x)
lasso.final <- cbind(ID, lasso.predict)
head(lasso.final)
```

```
##     ID       s1
## 1 1461 114906.6
## 2 1462 166738.4
## 3 1463 175294.5
## 4 1464 199502.4
## 5 1465 207258.2
## 6 1466 181002.3
```

Generalized Additive Models (GAM)

I now fit a GAM on the training set. I initially tried to use smooth.spine() to figure out the best degrees of freedom for each parameter. It came up with wonky results, so I decided to use use the default df, which is 4. Now I can create my GAM model. The predictors that I chose for my GAM model come from the predictors that resulted from my Forward Stepwise model.

```
library(gam)
```

```
## Warning: package 'gam' was built under R version 4.2.2
```

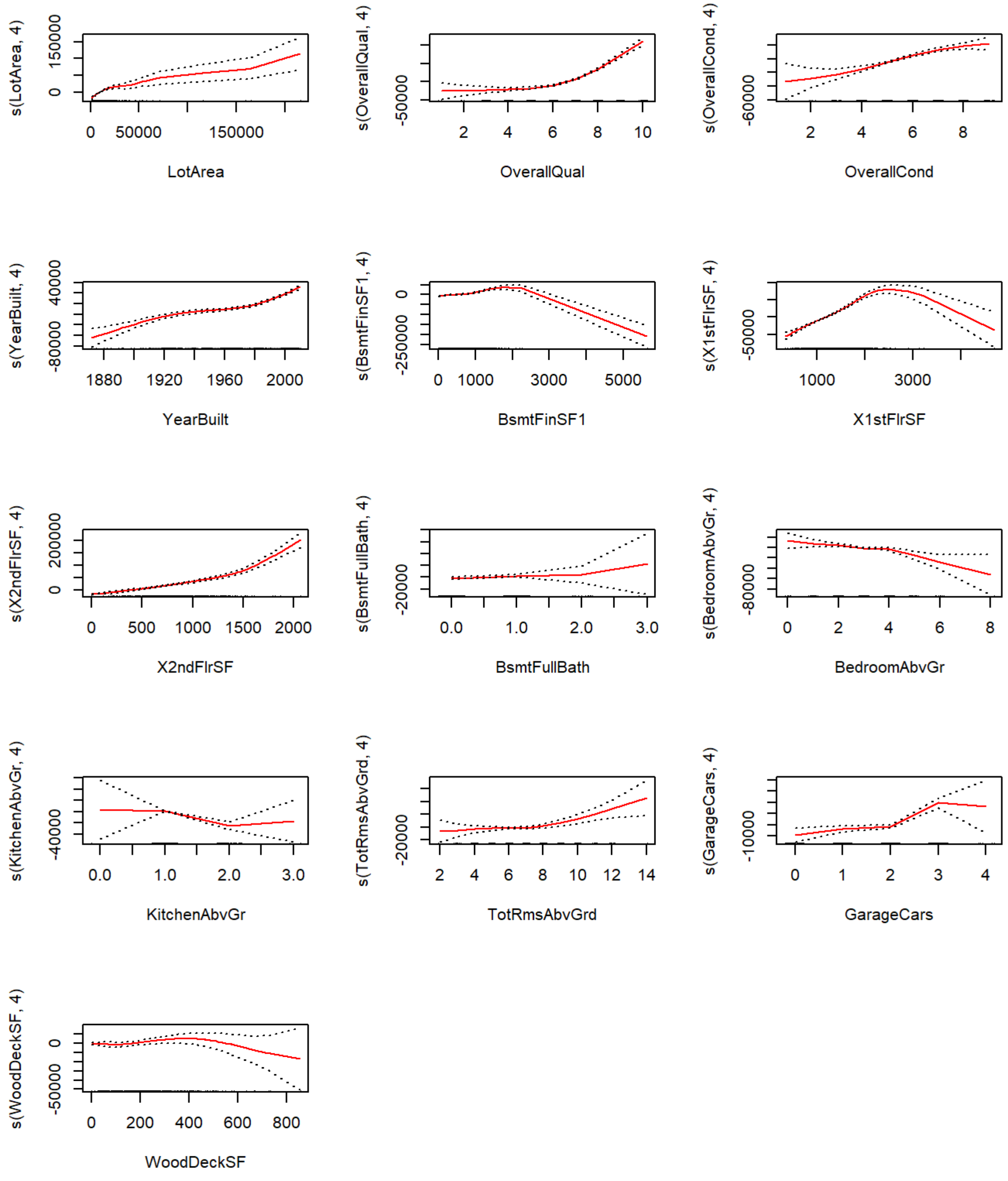
```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 4.2.2
```

```
## Loaded gam 1.22
```

```
gam.fit <- gam(SalePrice ~ s(LotArea, 4) + s(OverallQual, 4) + s(OverallCond, 4) + s(YearBuilt,
4) + s(BsmtFinSF1, 4) + s(X1stFlrSF, 4) + s(X2ndFlrSF, 4) + s(BsmtFullBath, 4) + s(BedroomAbvGr,
4) + s(KitchenAbvGr, 4) + s(TotRmsAbvGrd, 4) + s(GarageCars, 4) + s(WoodDeckSF, 4), data=train)
par(mfrow = c(3, 3))
plot(gam.fit, se = T, col = "red")
```



LotArea, OverallCond and YearBuilt seem to have a linear effect. As those variables increase, so does SalePRice.

X1stFlrSF seems to have a quadratic effect. As 1st floor square feet increases, so does SalePrice. As it continues to increase though, SalePrice starts decreasing. OverallQual and X2ndFlrSF seem to have an exponential effect. The other variables appear to have non-linear effects.

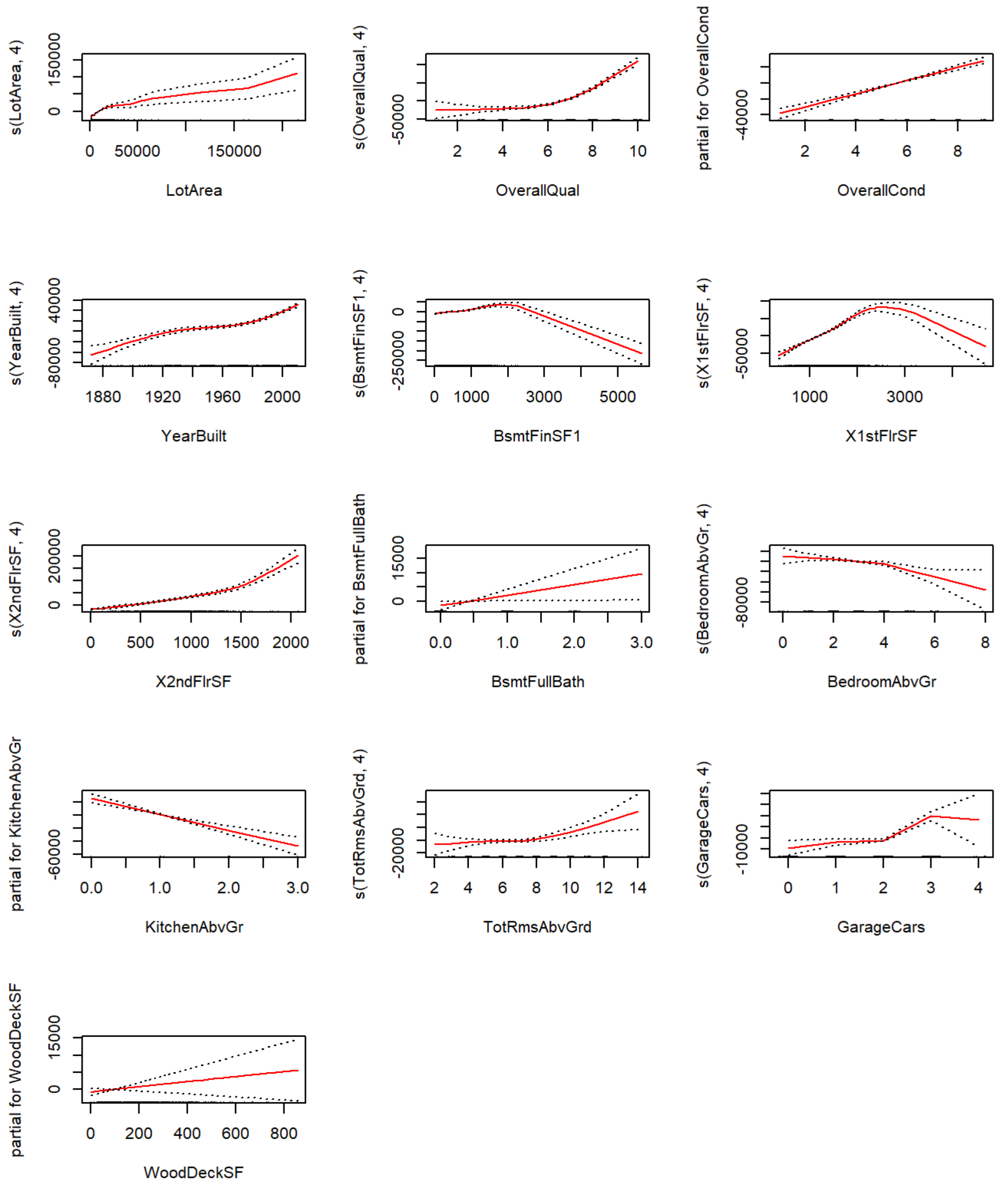
I use summary to see whether the non-linear effects are significant or not.

```
summary(gam.fit)
```

```
##
## Call: gam(formula = SalePrice ~ s(LotArea, 4) + s(OverallQual, 4) +
##     s(OverallCond, 4) + s(YearBuilt, 4) + s(BsmtFinSF1, 4) +
##     s(X1stFlrSF, 4) + s(X2ndFlrSF, 4) + s(BsmtFullBath, 4) +
##     s(BedroomAbvGr, 4) + s(KitchenAbvGr, 4) + s(TotRmsAbvGrd,
##     4) + s(GarageCars, 4) + s(WoodDeckSF, 4), data = train)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -374571  -10953    -571   10421  185803
##
## (Dispersion Parameter for gaussian family taken to be 687793329)
##
##     Null Deviance: 9.207911e+12 on 1459 degrees of freedom
## Residual Deviance: 969100912572 on 1409 degrees of freedom
## AIC: 33904.93
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                     Df     Sum Sq    Mean Sq   F value    Pr(>F)
## s(LotArea, 4)        1 6.5966e+11 6.5966e+11  959.1014 < 2.2e-16 ***
## s(OverallQual, 4)    1 4.7100e+12 4.7100e+12 6847.9468 < 2.2e-16 ***
## s(OverallCond, 4)    1 4.2580e+09 4.2580e+09    6.1907  0.012957 *
## s(YearBuilt, 4)      1 2.1409e+11 2.1409e+11  311.2742 < 2.2e-16 ***
## s(BsmtFinSF1, 4)     1 1.9139e+11 1.9139e+11  278.2696 < 2.2e-16 ***
## s(X1stFlrSF, 4)      1 1.4845e+11 1.4845e+11  215.8324 < 2.2e-16 ***
## s(X2ndFlrSF, 4)      1 5.6129e+11 5.6129e+11  816.0755 < 2.2e-16 ***
## s(BsmtFullBath, 4)   1 3.6618e+09 3.6618e+09    5.3240  0.021178 *
## s(BedroomAbvGr, 4)   1 1.1232e+10 1.1232e+10   16.3298 5.609e-05 ***
## s(KitchenAbvGr, 4)   1 2.7799e+10 2.7799e+10   40.4173 2.764e-10 ***
## s(TotRmsAbvGrd, 4)   1 7.2617e+09 7.2617e+09   10.5579  0.001184 **
## s(GarageCars, 4)     1 2.8155e+10 2.8155e+10   40.9346 2.137e-10 ***
## s(WoodDeckSF, 4)     1 9.8141e+08 9.8141e+08    1.4269  0.232472
## Residuals         1409 9.6910e+11 6.8779e+08
## ---
## 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, 4)           3 13.402 1.259e-08 ***
## s(OverallQual, 4)       3 77.019 < 2.2e-16 ***
## s(OverallCond, 4)       3  1.569   0.19494
## s(YearBuilt, 4)         3 17.782 2.485e-11 ***
## s(BsmtFinSF1, 4)        3 56.428 < 2.2e-16 ***
## s(X1stFlrSF, 4)         3 54.365 < 2.2e-16 ***
## s(X2ndFlrSF, 4)         3 25.124 7.772e-16 ***
## s(BsmtFullBath, 4)      2  0.142   0.86796
## s(BedroomAbvGr, 4)      3  3.896   0.00871 **
## s(KitchenAbvGr, 4)      2  1.895   0.15069
## s(TotRmsAbvGrd, 4)      3  5.417   0.00105 **
## s(GarageCars, 4)        3 14.368 3.183e-09 ***
## s(WoodDeckSF, 4)          3  2.065   0.10301
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It appears that OverallCond, BsmtFullBath, KitchenAbvGr, and WoodDeckSF non-linear effects are very insignificant. I will remove the non-linear effects of those variables.

```
gam.fit2 <- gam(SalePrice ~ s(LotArea, 4) + s(OverallQual, 4) + OverallCond + s(YearBuilt, 4) +
s(BsmtFinSF1, 4) + s(X1stFlrSF, 4) + s(X2ndFlrSF, 4) + BsmtFullBath + s(BedroomAbvGr, 4) + Kitch
enAbvGr + s(TotRmsAbvGrd, 4) + s(GarageCars, 4) + WoodDeckSF, data=train)
par(mfrow = c(3, 3))
plot(gam.fit2, se = T, col = "red")
```



You can see that OverallCond, BsmtFullBath, KitchenAbvGr, and WoodDeckSF have linear effects now. OverallCond and WoodDeckSF having positive effects, KitchenAbvGr having a negative effect.

```
summary(gam.fit2)
```

```
##
## Call: gam(formula = SalePrice ~ s(LotArea, 4) + s(OverallQual, 4) +
##     OverallCond + s(YearBuilt, 4) + s(BsmtFinSF1, 4) + s(X1stFlrSF,
##     4) + s(X2ndFlrSF, 4) + BsmtFullBath + s(BedroomAbvGr, 4) +
##     KitchenAbvGr + s(TotRmsAbvGrd, 4) + s(GarageCars, 4) + WoodDeckSF,
##     data = train)
## Deviance Residuals:
##        Min         1Q     Median         3Q        Max
## -375090.02  -11010.28     -98.81   10867.46  186349.35
##
## (Dispersion Parameter for gaussian family taken to be 689467248)
##
##     Null Deviance: 9.207911e+12 on 1459 degrees of freedom
## Residual Deviance: 978353855315 on 1419 degrees of freedom
## AIC: 33898.8
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                     Df     Sum Sq    Mean Sq   F value    Pr(>F)
## s(LotArea, 4)        1 6.5360e+11 6.5360e+11  947.9827 < 2.2e-16 ***
## s(OverallQual, 4)    1 4.6738e+12 4.6738e+12 6778.9266 < 2.2e-16 ***
## OverallCond          1 4.4757e+09 4.4757e+09    6.4915  0.010944 *
## s(YearBuilt, 4)      1 2.0943e+11 2.0943e+11  303.7507 < 2.2e-16 ***
## s(BsmtFinSF1, 4)     1 1.9602e+11 1.9602e+11  284.3075 < 2.2e-16 ***
## s(X1stFlrSF, 4)      1 1.5039e+11 1.5039e+11  218.1236 < 2.2e-16 ***
## s(X2ndFlrSF, 4)      1 5.6556e+11 5.6556e+11  820.2878 < 2.2e-16 ***
## BsmtFullBath         1 3.4473e+09 3.4473e+09    4.9999  0.025504 *
## s(BedroomAbvGr, 4)   1 1.3645e+10 1.3645e+10   19.7913 9.312e-06 ***
## KitchenAbvGr         1 2.7880e+10 2.7880e+10   40.4365 2.732e-10 ***
## s(TotRmsAbvGrd, 4)   1 6.9798e+09 6.9798e+09   10.1234  0.001496 **
## s(GarageCars, 4)     1 2.7268e+10 2.7268e+10   39.5490 4.251e-10 ***
## WoodDeckSF           1 1.1118e+09 1.1118e+09    1.6125  0.204343
## Residuals         1419 9.7835e+11 6.8947e+08
## ---
## 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, 4)           3 14.443 2.854e-09 ***
## s(OverallQual, 4)       3 75.966 < 2.2e-16 ***
## OverallCond
## s(YearBuilt, 4)         3 17.683 2.851e-11 ***
## s(BsmtFinSF1, 4)        3 57.988 < 2.2e-16 ***
## s(X1stFlrSF, 4)         3 53.344 < 2.2e-16 ***
## s(X2ndFlrSF, 4)         3 25.008 8.882e-16 ***
## BsmtFullBath
## s(BedroomAbvGr, 4)      3  4.072  0.006834 **
## KitchenAbvGr
## s(TotRmsAbvGrd, 4)      3  5.251  0.001325 **
## s(GarageCars, 4)        3 15.122 1.086e-09 ***
## WoodDeckSF
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
gam.predict <- predict(gam.fit2, newdata=test)
gam.final <- cbind(ID, gam.predict)
head(gam.final)
```

```
##     ID gam.predict
## 1 1461    124953.2
## 2 1462    167798.7
## 3 1463    182562.4
## 4 1464    190681.0
## 5 1465    196788.4
## 6 1466    166937.8
```

Regression Trees

I will now use the tree() function to fit a regression tree in order to predict SalePrice using all variables

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.2.2
```

```
tree.sales <- tree(SalePrice ~ ., data=train)
```

The summary() function lists the variables that are used as internal nodes in the tree, the number of terminal nodes, and the training MSE.

```
summary(tree.sales)
```

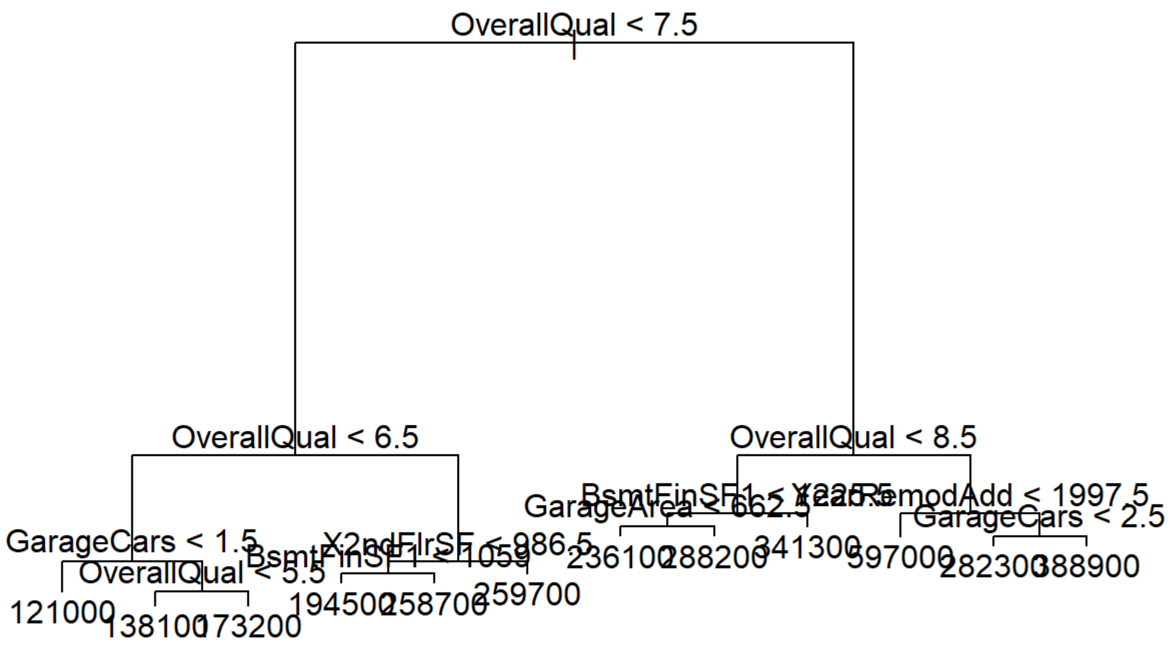
```
##
## Regression tree:
## tree(formula = SalePrice ~ ., data = train)
## Variables actually used in tree construction:
## [1] "OverallQual"  "GarageCars"   "X2ndFlrSF"    "BsmtFinSF1"   "GarageArea"
## [6] "YearRemodAdd"
## Number of terminal nodes:  12
## Residual mean deviance:  1.481e+09 = 2.145e+12 / 1448
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -212000  -22210   -1040       0   18940  222800
```

Out of all 23 predictors, only 6 of them are being used for the regression tree, with 12 terminal nodes. We see that the training MSE is 1.481e+09

```
tree.sales
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 1460 9.208e+12 180900
##    2) OverallQual < 7.5 1231 2.988e+12 157800
##      4) OverallQual < 6.5 912 1.287e+12 140400
##        8) GarageCars < 1.5 427 3.672e+11 121000 *
##        9) GarageCars > 1.5 485 6.196e+11 157400
##         18) OverallQual < 5.5 218 1.975e+11 138100 *
##         19) OverallQual > 5.5 267 2.739e+11 173200 *
##      5) OverallQual > 6.5 319 6.288e+11 207700
##       10) X2ndFlrSF < 986.5 279 4.335e+11 200300
##         20) BsmtFinSF1 < 1059 254 2.899e+11 194500 *
##         21) BsmtFinSF1 > 1059 25 4.972e+10 258700 *
##       11) X2ndFlrSF > 986.5 40 7.169e+10 259700 *
##    3) OverallQual > 7.5 229 2.037e+12 305000
##      6) OverallQual < 8.5 168 6.819e+11 274700
##       12) BsmtFinSF1 < 1225.5 142 4.290e+11 262500
##         24) GarageArea < 662.5 70 1.562e+11 236100 *
##         25) GarageArea > 662.5 72 1.765e+11 288200 *
##       13) BsmtFinSF1 > 1225.5 26 1.167e+11 341300 *
##      7) OverallQual > 8.5 61 7.756e+11 388500
##       14) YearRemodAdd < 1997.5 5 1.075e+11 597000 *
##       15) YearRemodAdd > 1997.5 56 4.313e+11 369900
##         30) GarageCars < 2.5 10 2.596e+10 282300 *
##         31) GarageCars > 2.5 46 3.121e+11 388900 *
```

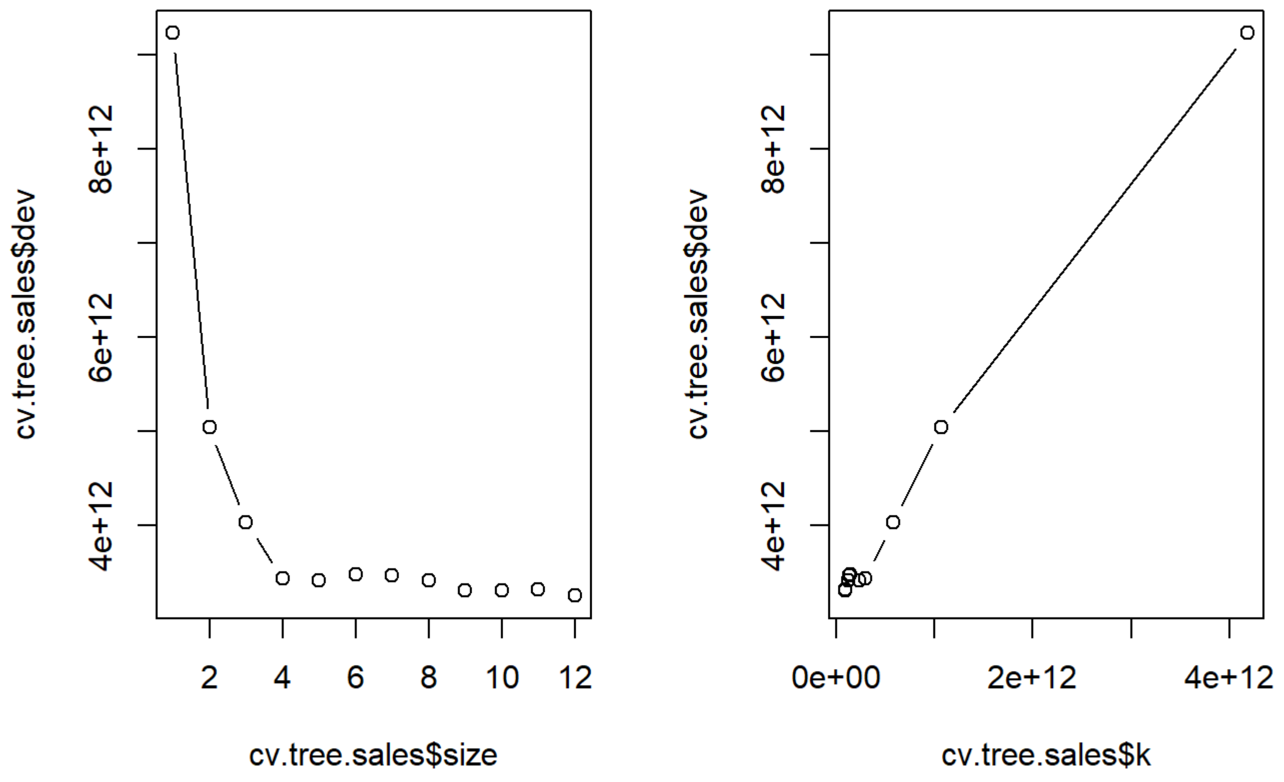
```
plot(tree.sales)
text(tree.sales, pretty=0)
```



It looks like OverallQual is the most important indicator of SalePrice, since the first few branches differentiates from high to low overall quality. Example: If OverallQual of house is less than 7.5, also less than 6.5, size of garage in car capacity is less than 1.5, the predicted price of the house is $121,000.

Now I will use the cv.tree() function to see whether pruning the tree will improve performance.

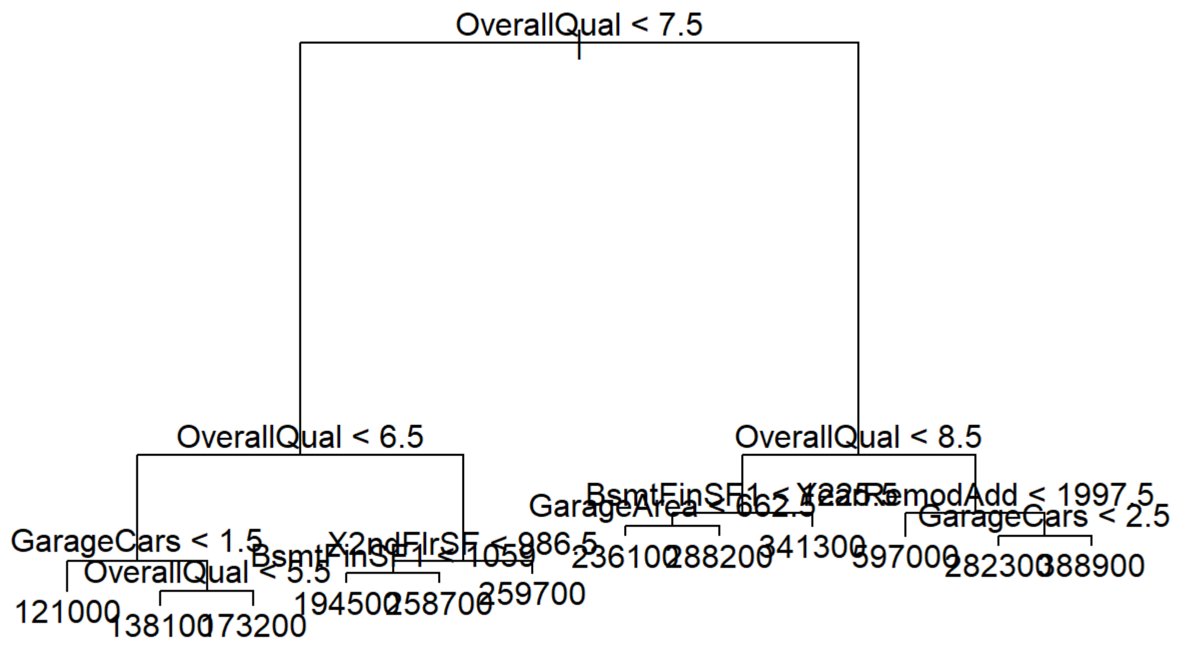
```
cv.tree.sales <- cv.tree(tree.sales, K = 10)
par(mfrow = c(1,2))
plot(cv.tree.sales$size, cv.tree.sales$dev, type = "b")
plot(cv.tree.sales$k, cv.tree.sales$dev, type = "b")
```



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

```
## [1] 12
```

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



This tree is exactly the same as the initial tree. This makes sense because the initial tree already had 12 leaves, which is the best.size amount.

```
tree.predict <- predict(prune.sales, newdata=test)
tree.final <- cbind(ID, tree.predict)
head(tree.final)
```

```
##     ID tree.predict
## 1 1461     121039.8
## 2 1462     121039.8
## 3 1463     138068.2
## 4 1464     173210.9
## 5 1465     236143.5
## 6 1466     173210.9
```

Bagging

Bagging is simply a special case of a random forest with m = p. Therefore, the randomForest() function can randomly be used to perform bagging.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.2.2
## randomForest 4.7-1.1
```

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

```
bag.sales <- randomForest(SalePrice ~ ., data=train, mtry=23, importance=TRUE)
bag.sales
```

```
##
## Call:
##  randomForest(formula = SalePrice ~ ., data = train, mtry = 23,      importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 23
##
##           Mean of squared residuals: 902471996
##                     % Var explained: 85.69
```

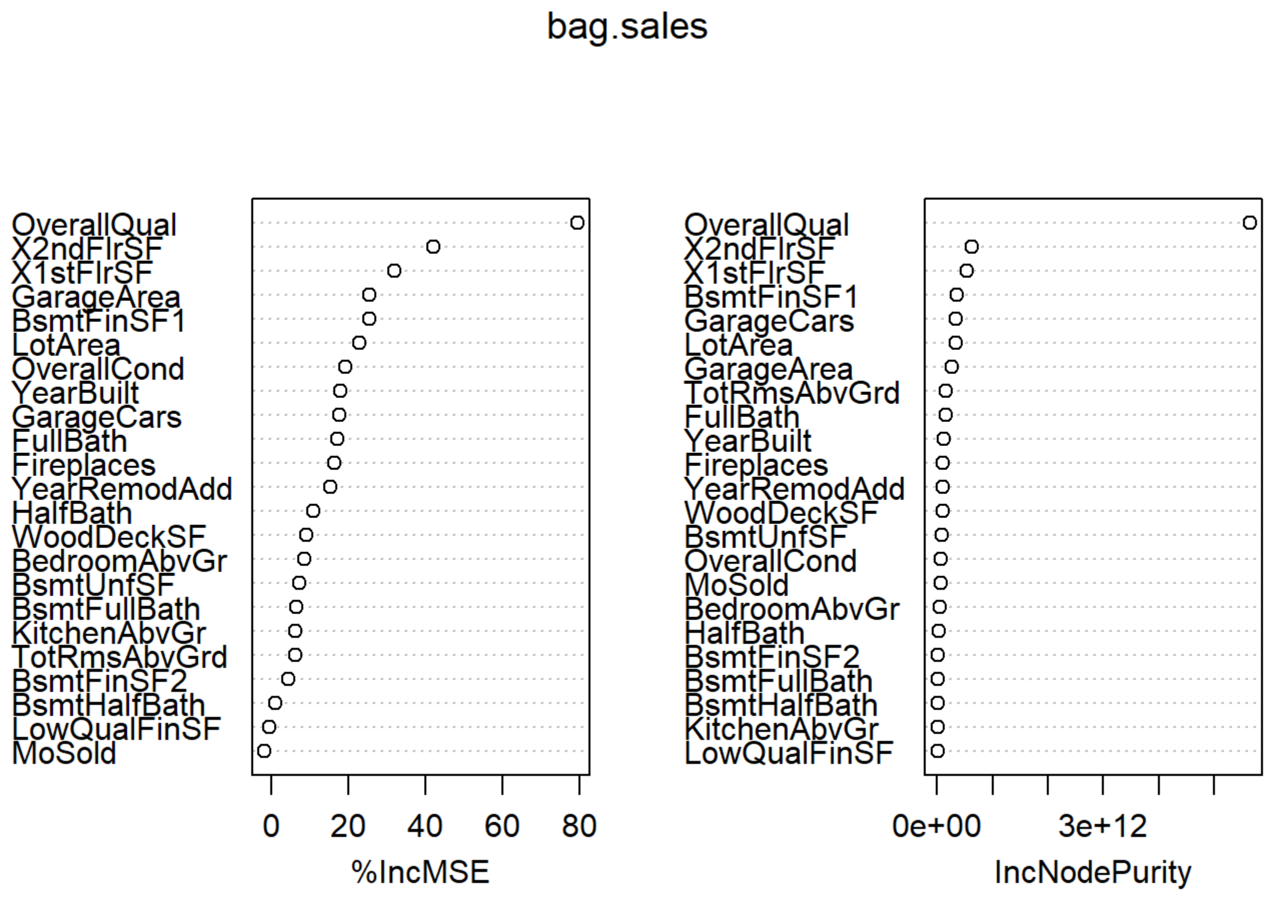
The argument mtry = 23 indicates that all 23 predictors should be considered for each split of the tree - in other words, that bagging should be done. We can see that the training MSE is 902471996

Using the importance() and varImpPlot functions, I can view the importance of each variable

```
importance(bag.sales)
```

```
##                 %IncMSE IncNodePurity
## LotArea      22.9635522  3.328721e+11
## OverallQual  79.4951504  5.645180e+12
## OverallCond  19.2609647  6.639033e+10
## YearBuilt    18.0405439  1.251395e+11
## YearRemodAdd 15.4596274  1.042377e+11
## BsmtFinSF1   25.5161291  3.593947e+11
## BsmtFinSF2    4.4243968  1.421850e+10
## BsmtUnfSF     7.4228215  8.553482e+10
## X1stFlrSF    31.9645754  5.304877e+11
## X2ndFlrSF    42.2950222  6.311489e+11
## LowQualFinSF -0.5175281  2.320130e+09
## BsmtFullBath  6.6407757  1.365103e+10
## BsmtHalfBath  1.2589859  8.031081e+09
## FullBath     17.3138915  1.555652e+11
## HalfBath     11.0309369  2.768467e+10
## BedroomAbvGr  8.6504540  3.889651e+10
## KitchenAbvGr  6.4362821  6.183108e+09
## TotRmsAbvGrd  6.2563136  1.584231e+11
## Fireplaces   16.5040616  1.056219e+11
## GarageCars   17.8393495  3.386597e+11
## GarageArea   25.6048513  2.560071e+11
## WoodDeckSF    9.2186019  9.261536e+10
## MoSold       -1.5888044  5.721967e+10
```

```
varImpPlot(bag.sales)
```



We can see that OveralQual is by far the most important variable when predicting SalePrice. X2ndFlrSF and X1stFlrSF come in 2nd and 3rd. It looks like MoSold and LowQualFInSF are the least important.

```
bag.predict <- predict(bag.sales, newdata=test)
bag.final <- cbind(ID, bag.predict)
head(bag.final)
```

```
##     ID bag.predict
## 1 1461    129291.7
## 2 1462    157233.8
## 3 1463    166728.8
## 4 1464    180281.2
## 5 1465    194826.4
## 6 1466    187895.5
```

Random Forest

Growing a random forest proceeds in exactly the same way, except that I will use a smaller value of the mtry argument.

```
sqrt(ncol(train) - 1)
```

```
## [1] 4.795832
```

```
rf.sales <- randomForest(SalePrice ~ ., data=train, mtry = 5, importance = TRUE)
rf.sales
```

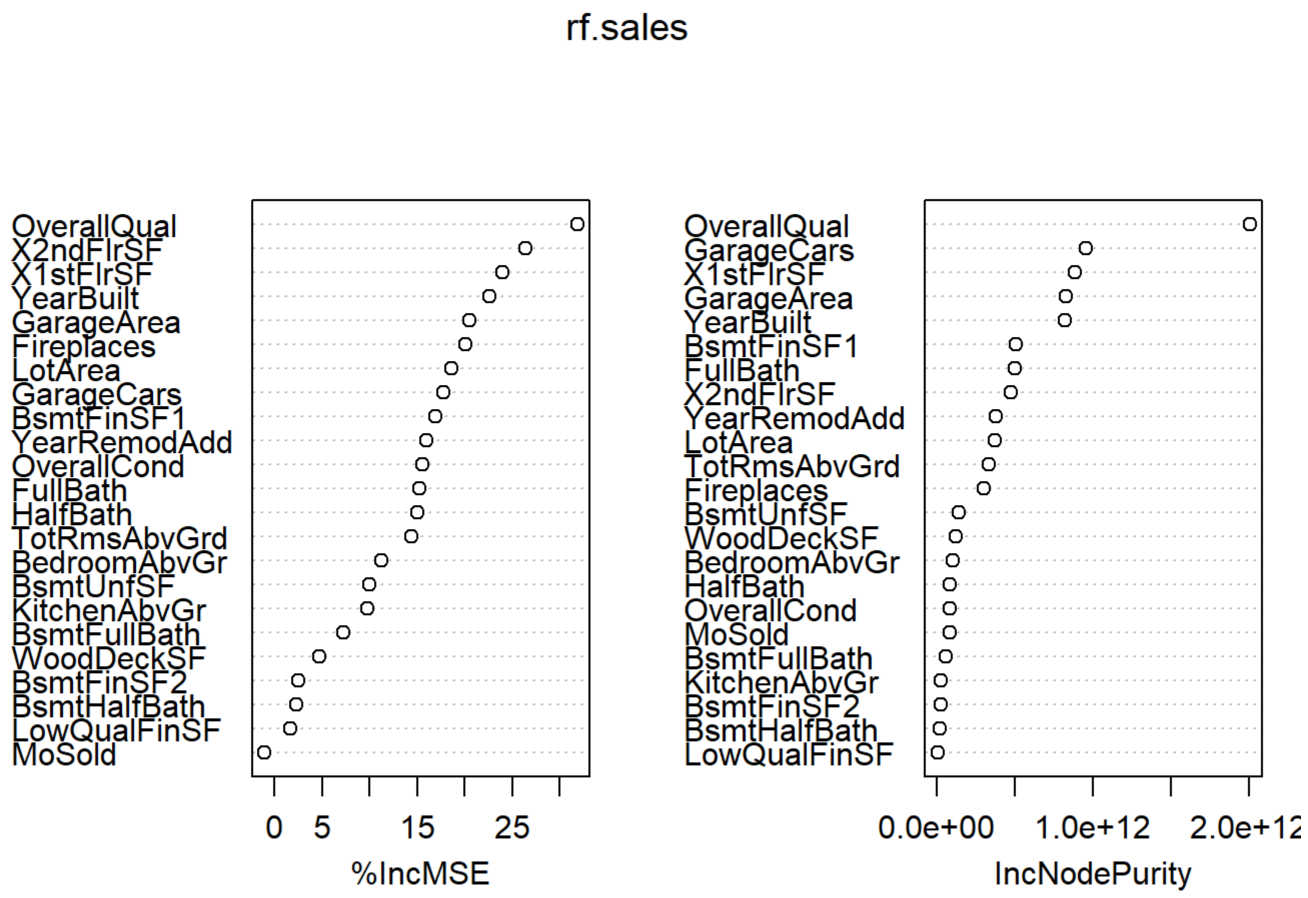
```
##
## Call:
##  randomForest(formula = SalePrice ~ ., data = train, mtry = 5,      importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 882192035
##                     % Var explained: 86.01
```

We can see that the training MSE is 882192035

```
importance(rf.sales)
```

```
##                %IncMSE IncNodePurity
## LotArea      18.597129  3.717714e+11
## OverallQual  31.841200  2.004339e+12
## OverallCond  15.593727  8.002951e+10
## YearBuilt    22.650362  8.169148e+11
## YearRemodAdd 16.001012  3.785947e+11
## BsmtFinSF1   16.970082  5.056234e+11
## BsmtFinSF2    2.481850  2.118702e+10
## BsmtUnfSF     9.951936  1.417402e+11
## X1stFlrSF    23.974318  8.813758e+11
## X2ndFlrSF    26.419822  4.718049e+11
## LowQualFinSF  1.639970  6.961889e+09
## BsmtFullBath  7.245137  5.335587e+10
## BsmtHalfBath  2.272958  1.641376e+10
## FullBath     15.239652  4.975732e+11
## HalfBath     15.023012  8.294988e+10
## BedroomAbvGr 11.257233  9.839582e+10
## KitchenAbvGr  9.705609  2.426613e+10
## TotRmsAbvGrd 14.384192  3.287979e+11
## Fireplaces   20.060170  2.968040e+11
## GarageCars   17.808502  9.551369e+11
## GarageArea   20.480446  8.230763e+11
## WoodDeckSF    4.713439  1.205809e+11
## MoSold       -1.052917  7.870304e+10
```

```
varImpPlot(rf.sales)
```



We can still see that OverallQual is the most important predictor, with X2ndFlrSF and X1stFlrSF not falling too far behind. MoSold and LowQualFinSF are still in last.

```
rf.predict <- predict(rf.sales, newdata=test)
rf.final <- cbind(ID, rf.predict)
head(rf.final)
```

```
##     ID rf.predict
## 1 1461   130137.5
## 2 1462   155048.7
## 3 1463   180998.3
## 4 1464   185649.0
## 5 1465   191148.4
## 6 1466   190874.7
```

Boosting

Here I use the gbm package, and within it the gbm() function, to fit boosted classification trees to the same data set. I run gbm() with the option distribution="gaussian" since this is a regression problem. The option interaction.depth=4 limits the depth of each tree.

```
library(gbm)
```

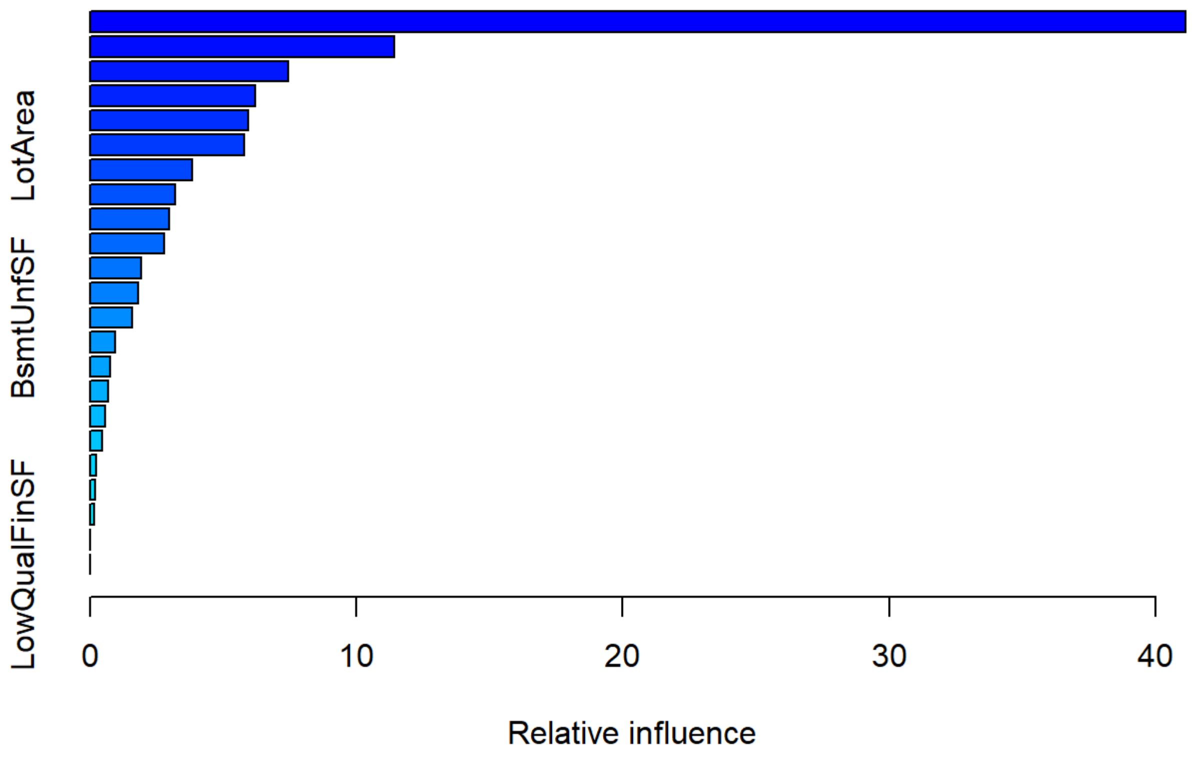
```
## Warning: package 'gbm' was built under R version 4.2.2
```

```
## Loaded gbm 2.1.8.1
```

```
boost.sales <- gbm(SalePrice ~ . , data=train, distribution="gaussian", shrinkage=0.01, n.tree=5
000, interaction.depth=4)
boost.sales
```

```
## gbm(formula = SalePrice ~ ., distribution = "gaussian", data = train,
##     n.trees = 5000, interaction.depth = 4, shrinkage = 0.01)
## A gradient boosted model with gaussian loss function.
## 5000 iterations were performed.
## There were 23 predictors of which 23 had non-zero influence.
```

```
summary(boost.sales)
```



```
##                       var      rel.inf
## OverallQual   OverallQual 4.113273e+01
## X1stFlrSF       X1stFlrSF 1.143481e+01
## BsmtFinSF1     BsmtFinSF1 7.439741e+00
## GarageCars     GarageCars 6.198381e+00
## X2ndFlrSF       X2ndFlrSF 5.934699e+00
## LotArea           LotArea 5.778989e+00
## GarageArea     GarageArea 3.830956e+00
## FullBath         FullBath 3.192297e+00
## TotRmsAbvGrd TotRmsAbvGrd 2.958153e+00
## YearBuilt       YearBuilt 2.780928e+00
## YearRemodAdd YearRemodAdd 1.915019e+00
## Fireplaces     Fireplaces 1.812283e+00
## BsmtUnfSF       BsmtUnfSF 1.574319e+00
## WoodDeckSF     WoodDeckSF 9.322491e-01
## OverallCond   OverallCond 7.594377e-01
## MoSold             MoSold 6.910096e-01
## HalfBath         HalfBath 5.788245e-01
## BedroomAbvGr BedroomAbvGr 4.680950e-01
## BsmtFinSF2     BsmtFinSF2 2.112614e-01
## BsmtFullBath BsmtFullBath 2.081271e-01
## KitchenAbvGr KitchenAbvGr 1.554598e-01
## BsmtHalfBath BsmtHalfBath 1.168905e-02
## LowQualFinSF LowQualFinSF 5.420322e-04
```

Again, OverallQual comes in 1st for most important predictor, with X1stFlrSF coming in 2nd. BsmtHalfBath and LowQualFinSF are the least important

I will now use the option cv.folds = 10 to perform 10-fold cross-validation to select the best number of trees.

```
boost.cv.sales <- gbm(SalePrice ~ . , data=train, distribution="gaussian", shrinkage=0.01, n.tre
e=5000, interaction.depth=4, cv.folds=10)
which.min(boost.cv.sales$cv.error)
```

```
## [1] 4798
```

```
boost.cv.sales
```

```
## gbm(formula = SalePrice ~ ., distribution = "gaussian", data = train,
##     n.trees = 5000, interaction.depth = 4, shrinkage = 0.01,
##     cv.folds = 10)
## A gradient boosted model with gaussian loss function.
## 5000 iterations were performed.
## The best cross-validation iteration was 4798.
## There were 23 predictors of which 23 had non-zero influence.
```

The cross-validation approach indicates that the best number of trees is 4798.

```
boost.predict <- predict(boost.cv.sales, newdata=test, n.trees=which.min(boost.cv.sales$cv.erro
r))
boost.final <- cbind(ID, boost.predict)
head(boost.final)
```

```
##        ID boost.predict
## [1,] 1461      125606.6
## [2,] 1462      161232.9
## [3,] 1463      185439.1
## [4,] 1464      190875.9
## [5,] 1465      185651.7
## [6,] 1466      184270.5
```

CV MSE

KNN Regression

```
set.seed(100)
fold.index <- cut(sample(1:nrow(train)), breaks=10, labels=FALSE)

mse <- rep(0,10)
for(i in 1:10){
  pred.val <- knn.reg(train.x.scale[fold.index!=i,], train.x.scale[fold.index==i,], train$SalePr
ice[fold.index!=i], k=5)
  mse[i] <- mean((pred.out$pred - train$SalePrice[fold.index==i])^2)
}
knn.cv.mse <- sum(mse)/10
knn.cv.mse
```

```
## [1] 7985721096
```

Linear Regression

```
set.seed(100)
fold.index <- cut(sample(1:nrow(train)), breaks=10, labels=FALSE)

mse <- rep(0,10)
for(i in 1:10){
  lm <- lm(SalePrice~., data=train[fold.index!=i,])
  lm.pred <- predict(lm, newdata=train[fold.index==i,])
  mse[i] <- mean((lm.pred - train$SalePrice[fold.index==i])^2)
}
lm.cv.mse <- sum(mse)/10
lm.cv.mse
```

```
## [1] 1440282364
```

Forward Stepwise Selection

```
set.seed(100)
fold.index <- cut(sample(1:nrow(train)), breaks=10, labels=FALSE)

mse <- rep(0,10)
for(i in 1:10){
  fwd.train <- train[fold.index!=i,]
  fwd.test <- train[fold.index==i,]
  true.y <- fwd.test[,'SalePrice']
  best.fit <- regsubsets(SalePrice ~., data=fwd.train, nvmax=23)
  fwd.pred <- predict.regsubsets(best.fit, fwd.test, id=13)
  mse[i] <- mean((fwd.pred-true.y)^2)
}
fwd.cv.mse <- sum(mse)/10
fwd.cv.mse
```

```
## [1] 1459598233
```

Ridge Regression

```
set.seed(100)
fold.index <- cut(sample(1:nrow(train)), breaks=10, labels=FALSE)

mse <- rep(0,10)
for(i in 1:10){
  train.x <- shrink.x[fold.index!=i,]
  train.y <- shrink.y[fold.index!=i]
  test.x <- shrink.x[fold.index==i,]
  test.y <- shrink.y[fold.index==i]
  ridge <- glmnet(train.x, train.y, alpha=0)
  ridge.pred <- predict(ridge, s=bestlam.ridge, newx=test.x)
  mse[i] <- mean((ridge.pred - test.y)^2)
}
ridge.cv.mse <- sum(mse)/10
ridge.cv.mse
```

```
## [1] 1395113514
```

Lasso

```
set.seed(100)
fold.index <- cut(sample(1:nrow(train)), breaks=10, labels=FALSE)

mse <- rep(0,10)
for(i in 1:10){
  train.x <- shrink.x[fold.index!=i,]
  train.y <- shrink.y[fold.index!=i]
  test.x <- shrink.x[fold.index==i,]
  test.y <- shrink.y[fold.index==i]
  lasso <- glmnet(train.x, train.y, alpha=1)
  lasso.pred <- predict(lasso, s=bestlam.lasso, newx=test.x)
  mse[i] <- mean((lasso.pred - test.y)^2)
}
lasso.cv.mse <- sum(mse)/10
lasso.cv.mse
```

```
## [1] 1425907868
```

Generalized Additive Models (GAM)

```
set.seed(100)
fold.index <- cut(sample(1:nrow(train)), breaks=10, labels=FALSE)

mse <- rep(0,10)
for(i in 1:10){
  gam <- gam(SalePrice ~ s(LotArea, 4) + s(OverallQual, 4) + OverallCond + s(YearBuilt, 4) + s(B
smtFinSF1, 4) + s(X1stFlrSF, 4) + s(X2ndFlrSF, 4) + BsmtFullBath + s(BedroomAbvGr, 4) + KitchenA
bvGr + s(TotRmsAbvGrd, 4) + s(GarageCars, 4) + WoodDeckSF, data=train[fold.index!=i,])
  gam.pred <- predict(gam, newdata=train[fold.index==i,])
  mse[i] <- mean((gam.pred - train$SalePrice[fold.index==i])^2)
}
gam.cv.mse <- sum(mse)/10
gam.cv.mse
```

```
## [1] 835879482
```

Regression Tree

```
tree.cv.mse <- min(cv.tree.sales$dev)/nrow(train)
tree.cv.mse
```

```
## [1] 2226328639
```

Bagging

```
set.seed(100)
fold.index <- cut(sample(1:nrow(train)), breaks=10, labels=FALSE)

mse <- rep(0,10)
for(i in 1:10){
  bag <- randomForest(SalePrice ~ ., mtry=23, data=train[fold.index!=i,])
  bag.pred <- predict(bag, newdata=train[fold.index==i,])
  mse[i] <- mean((bag.pred - train$SalePrice[fold.index==i])^2)
}
bag.cv.mse <- sum(mse)/10
bag.cv.mse
```

```
## [1] 928324226
```

Random Forest

```
set.seed(100)
fold.index <- cut(sample(1:nrow(train)), breaks=10, labels=FALSE)

mse <- rep(0,10)
for(i in 1:10){
  rf <- randomForest(SalePrice ~ ., mtry=5, data=train[fold.index!=i,])
  rf.pred <- predict(rf, newdata=train[fold.index==i,])
  mse[i] <- mean((rf.pred - train$SalePrice[fold.index==i])^2)
}
rf.cv.mse <- sum(mse)/10
rf.cv.mse
```

```
## [1] 878634096
```

Boosting

```
boost.cv.mse <- min(boost.cv.sales$cv.error)
boost.cv.mse
```

```
## [1] 834426735
```

CV MSE from best to worst: Boosting: 834,426,735 GAM: 835,879,482 Random Forest: 878,634,096 Bagging: 928,324,226 Ridge Regression: 1,395,113,514 Lasso: 1,425,907,868 Linear Regression: 1,440,282,364 Forward Stepwise Selection: 1,459,598,233 Regression Tree: 2,226,328,639 KNN Method: 7,985,721,096

True Test Errors from best to worst: Boosting: 0.14034 GAM: 0.14327 Linear Regression: 0.15083 Random Forest: 0.1521 Bagging: 0.15356 KNN Method: 0.17816 Regression Tree: 0.24341 Ridge Regression: 0.34976 Forward Stepwise Selection: 0.45533 Lasso: 0.4991

Boosting and GAM had the lowest errors for each error method, meaning those 2 methods were the most accurate. Since Lasso did much better for CV then True Test, you can say that this method over fit the data. Since Linear Regression and KNN Method did worse for CV than True Test, you can say that this method under fit the data. I am surprised GAM performed so well considering I only used the default df of 4 for each variable. Overall, it looks like Forward Stepwise selection performed the worst. Using Best Subset would probably have been better. Using a KNN parameter of 5 resulted in a inflexible model performed on the training data, so it makes sense to have a high CV MSE. It also makes sense KNN CV MSE was high because KNN does not work well for high denominational data.

Conclusion and Summary

From many of the different statistical methods I used, it can be concluded that OverallQual (Overall material and finish quality), X1ndFlrSF (First Floor square feet), and X2stFlrSF (Second floor square feet) are the most important variables in predicting the sale price of a house. We can also conclude that Boosting and GAM were the best techniques for accurately predicting sale price.

After using all these different regression techniques, it makes me wonder how many other techniques are out there that could perform even better on this data set.