STT 481 PROJECT

Derien Weatherspoon

2023-03-22

Project Overview and Goals

Goal: Predict the sales price for each house. For each Id in the test set, you must predict the value of the SalePrice variable. So, **SalePrice** is the **Response** variable.

Metric: Submissions are evaluated on Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed sales price. (Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.)

Evaluation: Class evaluation is based on the following methodologies: KNN Methods, Linear Regression, Subset Selection, Shrinkage Methods, Generalized Additive Models, Regression Trees, Bagging, Random Forest, and Boosting methods. Extracted from these models will be the Estimated Test Errors (CV Estimate) and True Test Errors.

Introduce the Data

The data set used for this project is the Ames(Iowa) Housing Data. The unprocessed data set includes 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa.

Loading Packages

```
library(FNN)
library(class)
library(dplyr)
library(MASS)
library(corrplot)
library(ggcorrplot)
library(ggplot2)
library(glmnet)
library(gpairs)
library(leaps)
library(boot)
library(tree)
library(gbm)
library(splines)
library(gam)
library(randomForest)
library(car)
library(rpart)
library(rpart.plot)
```

Exploratory Data Analysis

I am using the pre-processed data, which is an already cleaned data set, that has 24 columns, and around 2900 rows combined.

Load Preprocessed Data

```
train <- read.csv("train_new.csv")
test <- read.csv("test_new.csv")
index <- rbind(train,test) # for KNN method
```

Because this data is already processed, I will not be doing any cleaning or checking of NAs. Creating a combination of the 2 datasets just in case.

Data dimensions and structure

```
dim(train)
```

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

```
dim(test)
```

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

```
dim(index)
```

```
## [1] 2919   24
```

The pre-processed data has a lot of columns removed.

Glimpse on data

```
glimpse(train)
```

```
## Rows: 1,460
## Columns: 24
## $ LotArea      <int> 8450, 9600, 11250, 9550, 14260, 14115, 10084, 10382, 6120~
## $ OverallQual  <int> 7, 6, 7, 7, 8, 5, 8, 7, 7, 5, 5, 9, 5, 7, 6, 7, 6, 4, 5, ~
## $ OverallCond  <int> 5, 8, 5, 5, 5, 5, 5, 6, 5, 6, 5, 5, 6, 5, 5, 8, 7, 5, 5, ~
## $ YearBuilt    <int> 2003, 1976, 2001, 1915, 2000, 1993, 2004, 1973, 1931, 193~
## $ YearRemodAdd <int> 2003, 1976, 2002, 1970, 2000, 1995, 2005, 1973, 1950, 195~
## $ BsmtFinSF1   <int> 706, 978, 486, 216, 655, 732, 1369, 859, 0, 851, 906, 998~
## $ BsmtFinSF2   <int> 0, 0, 0, 0, 0, 0, 0, 32, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ BsmtUnfSF    <int> 150, 284, 434, 540, 490, 64, 317, 216, 952, 140, 134, 177~
## $ X1stFlrSF    <int> 856, 1262, 920, 961, 1145, 796, 1694, 1107, 1022, 1077, 1~
## $ X2ndFlrSF    <int> 854, 0, 866, 756, 1053, 566, 0, 983, 752, 0, 0, 1142, 0, ~
## $ LowQualFinSF <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ BsmtFullBath <int> 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, ~
## $ BsmtHalfBath <int> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ FullBath     <int> 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 1, 3, 1, 2, 1, 1, 1, 2, 1, ~
## $ HalfBath     <int> 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, ~
## $ BedroomAbvGr <int> 3, 3, 3, 3, 4, 1, 3, 3, 2, 2, 3, 4, 2, 3, 2, 2, 2, 2, 3, ~
## $ KitchenAbvGr <int> 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, ~
## $ TotRmsAbvGrd <int> 8, 6, 6, 7, 9, 5, 7, 7, 8, 5, 5, 11, 4, 7, 5, 5, 5, 6, 6,~
## $ Fireplaces   <int> 0, 1, 1, 1, 1, 0, 1, 2, 2, 2, 0, 2, 0, 1, 1, 0, 1, 0, 0, ~
## $ GarageCars   <int> 2, 2, 2, 3, 3, 2, 2, 2, 2, 1, 1, 3, 1, 3, 1, 2, 2, 2, 2, ~
## $ GarageArea   <int> 548, 460, 608, 642, 836, 480, 636, 484, 468, 205, 384, 73~
## $ WoodDeckSF   <int> 0, 298, 0, 0, 192, 40, 255, 235, 90, 0, 0, 147, 140, 160,~
## $ MoSold       <int> 2, 5, 9, 2, 12, 10, 8, 11, 4, 1, 2, 7, 9, 8, 5, 7, 3, 10,~
## $ SalePrice    <dbl> 208500, 181500, 223500, 140000, 250000, 143000, 307000, 2~
```

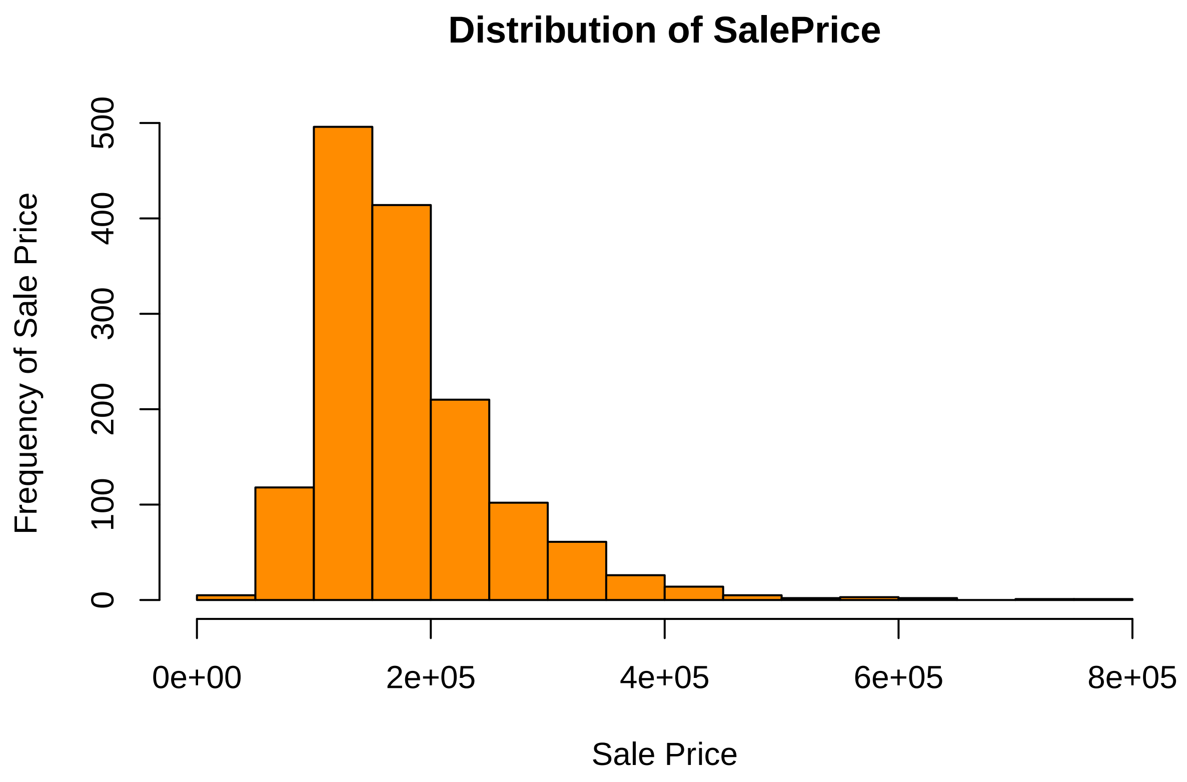
```
glimpse(test)
```

```
## Rows: 1,459
## Columns: 24
## $ LotArea      <int> 11622, 14267, 13830, 9978, 5005, 10000, 7980, 8402, 10176~
## $ OverallQual  <int> 5, 6, 5, 6, 8, 6, 6, 6, 7, 4, 7, 6, 5, 6, 7, 9, 8, 9, 8, ~
## $ OverallCond  <int> 6, 6, 5, 6, 5, 5, 7, 5, 5, 5, 5, 5, 5, 6, 6, 5, 5, 5, 5, ~
## $ YearBuilt    <int> 1961, 1958, 1997, 1998, 1992, 1993, 1992, 1998, 1990, 197~
## $ YearRemodAdd <int> 1961, 1958, 1998, 1998, 1992, 1994, 2007, 1998, 1990, 197~
## $ BsmtFinSF1   <dbl> 468, 923, 791, 602, 263, 0, 935, 0, 637, 804, 1051, 156, ~
## $ BsmtFinSF2   <dbl> 144, 0, 0, 0, 0, 0, 0, 0, 0, 78, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ BsmtUnfSF    <dbl> 270, 406, 137, 324, 1017, 763, 233, 789, 663, 0, 354, 327~
## $ X1stFlrSF    <int> 896, 1329, 928, 926, 1280, 763, 1187, 789, 1341, 882, 133~
## $ X2ndFlrSF    <int> 0, 0, 701, 678, 0, 892, 0, 676, 0, 0, 0, 504, 567, 601, 0~
## $ LowQualFinSF <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ BsmtFullBath <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ BsmtHalfBath <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ FullBath     <int> 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 2, ~
## $ HalfBath     <int> 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, ~
## $ BedroomAbvGr <int> 2, 3, 3, 3, 2, 3, 3, 3, 2, 2, 2, 2, 3, 3, 2, 3, 3, 3, 3, ~
## $ KitchenAbvGr <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ TotRmsAbvGrd <int> 5, 6, 6, 7, 5, 7, 6, 7, 5, 4, 5, 5, 6, 6, 4, 10, 7, 7, 8,~
## $ Fireplaces   <int> 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, ~
## $ GarageCars   <dbl> 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 3, 3, 3, 3, ~
## $ GarageArea   <dbl> 730, 312, 482, 470, 506, 440, 420, 393, 506, 525, 511, 26~
## $ WoodDeckSF   <int> 140, 393, 212, 360, 0, 157, 483, 0, 192, 240, 203, 275, 0~
## $ MoSold       <int> 6, 6, 3, 6, 1, 4, 3, 5, 2, 4, 6, 2, 3, 6, 6, 1, 6, 6, 2, ~
## $ SalePrice    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

Some data types vary from train and test. Such as SalePrice and GarageCars variables. Notice how SalePrice in the testing set is all zeros.

Show the distribution of SalePrice

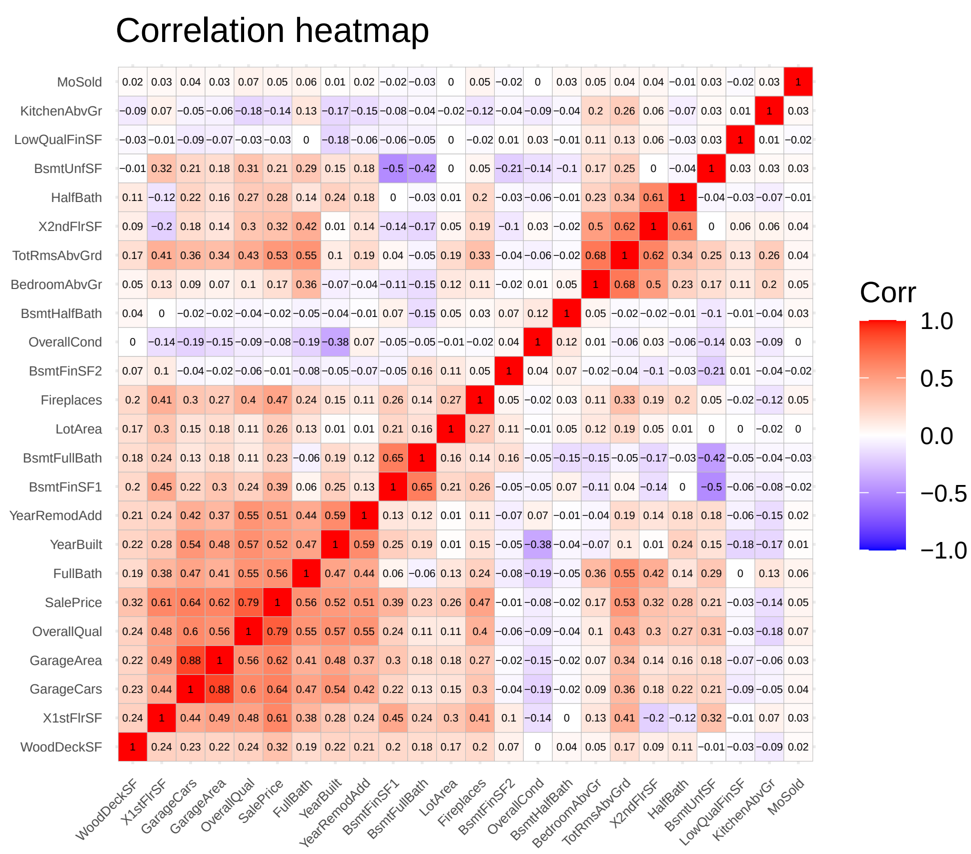
```
hist(x = train$SalePrice, main = "Distribution of SalePrice",
     xlab = "Sale Price", ylab = "Frequency of Sale Price", col = "darkorange")
```



The distribution of the housing prices looks to have the highest frequency around 130,000 to 180,000 dollars.

Correlation of all variables

```
ggcorrplot(cor(train), lab_size = 1.5, tl.cex = 5, lab = T,
           title = "Correlation heatmap",  hc.order = TRUE)
```



```
#correlation heatmap

round(cor(train),
  digits = 2 # rounded to 2 decimals
)
```

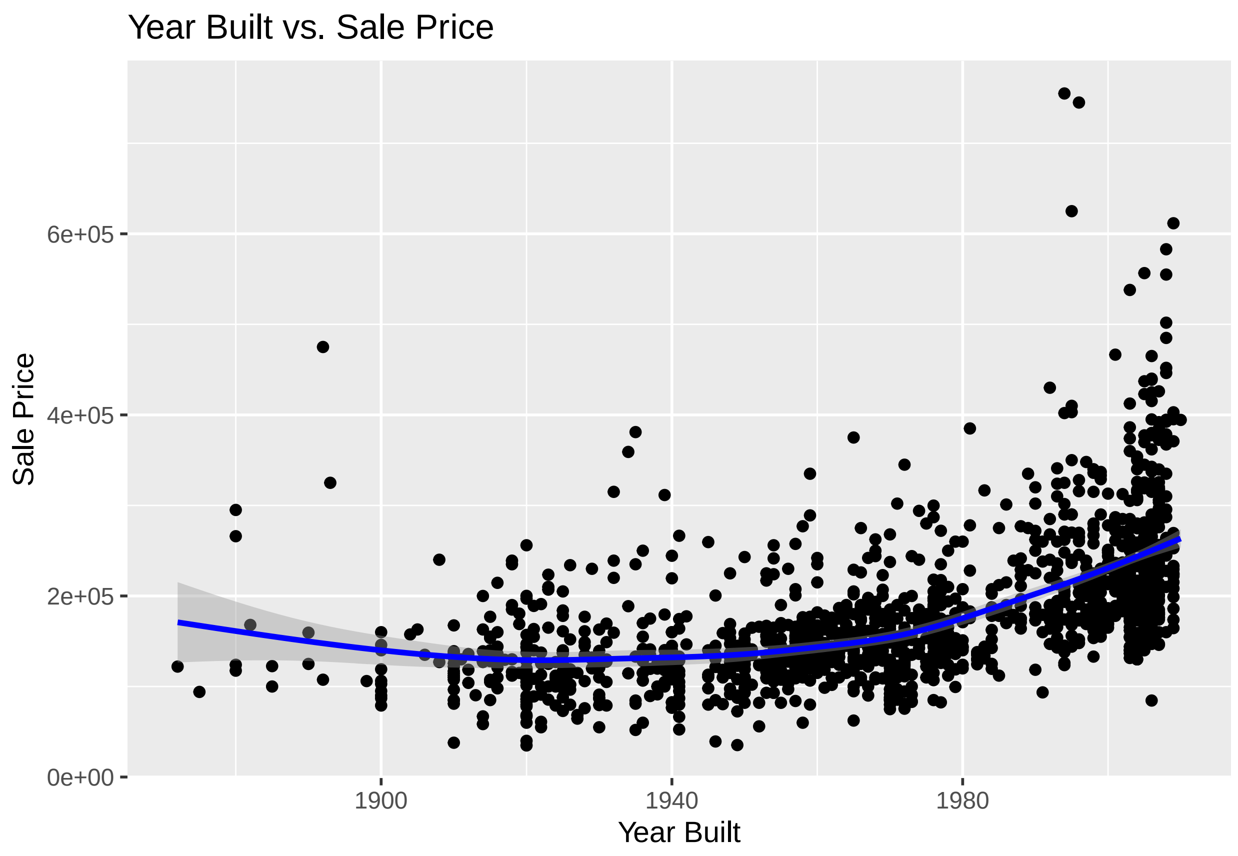
```
##               LotArea OverallQual OverallCond YearBuilt YearRemodAdd BsmtFinSF1
## LotArea          1.00        0.11       -0.01      0.01         0.01       0.21
## OverallQual      0.11        1.00       -0.09      0.57         0.55       0.24
## OverallCond     -0.01       -0.09        1.00     -0.38         0.07      -0.05
## YearBuilt        0.01        0.57       -0.38      1.00         0.59       0.25
## YearRemodAdd     0.01        0.55        0.07      0.59         1.00       0.13
## BsmtFinSF1       0.21        0.24       -0.05      0.25         0.13       1.00
## BsmtFinSF2       0.11       -0.06        0.04     -0.05        -0.07      -0.05
## BsmtUnfSF        0.00        0.31       -0.14      0.15         0.18      -0.50
## X1stFlrSF        0.30        0.48       -0.14      0.28         0.24       0.45
## X2ndFlrSF        0.05        0.30        0.03      0.01         0.14      -0.14
## LowQualFinSF     0.00       -0.03        0.03     -0.18        -0.06      -0.06
## BsmtFullBath     0.16        0.11       -0.05      0.19         0.12       0.65
## BsmtHalfBath     0.05       -0.04        0.12     -0.04        -0.01       0.07
## FullBath         0.13        0.55       -0.19      0.47         0.44       0.06
## HalfBath         0.01        0.27       -0.06      0.24         0.18       0.00
## BedroomAbvGr     0.12        0.10        0.01     -0.07        -0.04      -0.11
## KitchenAbvGr    -0.02       -0.18       -0.09     -0.17        -0.15      -0.08
## TotRmsAbvGrd     0.19        0.43       -0.06      0.10         0.19       0.04
## Fireplaces       0.27        0.40       -0.02      0.15         0.11       0.26
## GarageCars      0.15         0.60         -0.19        0.54           0.42        0.22
## GarageArea      0.18         0.56         -0.15        0.48           0.37        0.30
## WoodDeckSF      0.17         0.24          0.00        0.22           0.21        0.20
## MoSold          0.00         0.07          0.00        0.01           0.02       -0.02
## SalePrice       0.26         0.79         -0.08        0.52           0.51        0.39
##             BsmtFinSF2 BsmtUnfSF X1stFlrSF X2ndFlrSF LowQualFinSF BsmtFullBath
## LotArea           0.11      0.00      0.30      0.05         0.00         0.16
## OverallQual      -0.06      0.31      0.48      0.30        -0.03         0.11
## OverallCond       0.04     -0.14     -0.14      0.03         0.03        -0.05
## YearBuilt        -0.05      0.15      0.28      0.01        -0.18         0.19
## YearRemodAdd     -0.07      0.18      0.24      0.14        -0.06         0.12
## BsmtFinSF1       -0.05     -0.50      0.45     -0.14        -0.06         0.65
## BsmtFinSF2        1.00     -0.21      0.10     -0.10         0.01         0.16
## BsmtUnfSF        -0.21      1.00      0.32      0.00         0.03        -0.42
## X1stFlrSF         0.10      0.32      1.00     -0.20        -0.01         0.24
## X2ndFlrSF        -0.10      0.00     -0.20      1.00         0.06        -0.17
## LowQualFinSF      0.01      0.03     -0.01      0.06         1.00        -0.05
## BsmtFullBath      0.16     -0.42      0.24     -0.17        -0.05         1.00
## BsmtHalfBath      0.07     -0.10      0.00     -0.02        -0.01        -0.15
## FullBath         -0.08      0.29      0.38      0.42         0.00        -0.06
## HalfBath         -0.03     -0.04     -0.12      0.61        -0.03        -0.03
## BedroomAbvGr     -0.02      0.17      0.13      0.50         0.11        -0.15
## KitchenAbvGr     -0.04      0.03      0.07      0.06         0.01        -0.04
## TotRmsAbvGrd     -0.04      0.25      0.41      0.62         0.13        -0.05
## Fireplaces        0.05      0.05      0.41      0.19        -0.02         0.14
## GarageCars       -0.04      0.21      0.44      0.18        -0.09         0.13
## GarageArea       -0.02      0.18      0.49      0.14        -0.07         0.18
## WoodDeckSF        0.07     -0.01      0.24      0.09        -0.03         0.18
## MoSold           -0.02      0.03      0.03      0.04        -0.02        -0.03
## SalePrice        -0.01      0.21      0.61      0.32        -0.03         0.23
##             BsmtHalfBath FullBath HalfBath BedroomAbvGr KitchenAbvGr
## LotArea             0.05     0.13     0.01         0.12        -0.02
## OverallQual        -0.04     0.55     0.27         0.10        -0.18
## OverallCond         0.12    -0.19    -0.06         0.01        -0.09
## YearBuilt          -0.04     0.47     0.24        -0.07        -0.17
## YearRemodAdd       -0.01     0.44     0.18        -0.04        -0.15
## BsmtFinSF1          0.07     0.06     0.00        -0.11        -0.08
## BsmtFinSF2          0.07    -0.08    -0.03        -0.02        -0.04
## BsmtUnfSF          -0.10     0.29    -0.04         0.17         0.03
## X1stFlrSF           0.00     0.38    -0.12         0.13         0.07
## X2ndFlrSF          -0.02     0.42     0.61         0.50         0.06
## LowQualFinSF       -0.01     0.00    -0.03         0.11         0.01
## BsmtFullBath       -0.15    -0.06    -0.03        -0.15        -0.04
## BsmtHalfBath        1.00    -0.05    -0.01         0.05        -0.04
## FullBath           -0.05     1.00     0.14         0.36         0.13
## HalfBath           -0.01     0.14     1.00         0.23        -0.07
## BedroomAbvGr        0.05     0.36     0.23         1.00         0.20
## KitchenAbvGr       -0.04     0.13    -0.07         0.20         1.00
## TotRmsAbvGrd       -0.02     0.55     0.34         0.68         0.26
## Fireplaces          0.03     0.24     0.20         0.11        -0.12
## GarageCars         -0.02     0.47     0.22         0.09        -0.05
## GarageArea         -0.02     0.41     0.16         0.07        -0.06
## WoodDeckSF          0.04     0.19     0.11         0.05        -0.09
## MoSold              0.03     0.06    -0.01         0.05         0.03
## SalePrice           -0.02     0.56     0.28         0.17        -0.14
##              TotRmsAbvGrd Fireplaces GarageCars GarageArea WoodDeckSF MoSold
## LotArea              0.19       0.27       0.15       0.18       0.17   0.00
## OverallQual          0.43       0.40       0.60       0.56       0.24   0.07
## OverallCond         -0.06      -0.02      -0.19      -0.15       0.00   0.00
## YearBuilt            0.10       0.15       0.54       0.48       0.22   0.01
## YearRemodAdd         0.19       0.11       0.42       0.37       0.21   0.02
## BsmtFinSF1           0.04       0.26       0.22       0.30       0.20  -0.02
## BsmtFinSF2          -0.04       0.05      -0.04      -0.02       0.07  -0.02
## BsmtUnfSF            0.25       0.05       0.21       0.18      -0.01   0.03
## X1stFlrSF            0.41       0.41       0.44       0.49       0.24   0.03
## X2ndFlrSF            0.62       0.19       0.18       0.14       0.09   0.04
## LowQualFinSF         0.13      -0.02      -0.09      -0.07      -0.03  -0.02
## BsmtFullBath        -0.05       0.14       0.13       0.18       0.18  -0.03
## BsmtHalfBath        -0.02       0.03      -0.02      -0.02       0.04   0.03
## FullBath             0.55       0.24       0.47       0.41       0.19   0.06
## HalfBath             0.34       0.20       0.22       0.16       0.11  -0.01
## BedroomAbvGr         0.68       0.11       0.09       0.07       0.05   0.05
## KitchenAbvGr         0.26      -0.12      -0.05      -0.06      -0.09   0.03
## TotRmsAbvGrd         1.00       0.33       0.36       0.34       0.17   0.04
## Fireplaces           0.33       1.00       0.30       0.27       0.20   0.05
## GarageCars           0.36       0.30       1.00       0.88       0.23   0.04
## GarageArea           0.34       0.27       0.88       1.00       0.22   0.03
## WoodDeckSF           0.17       0.20       0.23       0.22       1.00   0.02
## MoSold               0.04       0.05       0.04       0.03       0.02   1.00
## SalePrice            0.53       0.47       0.64       0.62       0.32   0.05
##              SalePrice
## LotArea           0.26
## OverallQual       0.79
## OverallCond      -0.08
## YearBuilt         0.52
## YearRemodAdd      0.51
## BsmtFinSF1        0.39
## BsmtFinSF2       -0.01
## BsmtUnfSF         0.21
## X1stFlrSF         0.61
## X2ndFlrSF         0.32
## LowQualFinSF     -0.03
## BsmtFullBath      0.23
## BsmtHalfBath     -0.02
## FullBath          0.56
## HalfBath          0.28
## BedroomAbvGr      0.17
## KitchenAbvGr     -0.14
## TotRmsAbvGrd      0.53
## Fireplaces        0.47
## GarageCars        0.64
## GarageArea        0.62
## WoodDeckSF        0.32
## MoSold            0.05
## SalePrice         1.00
```

OverallQual seems to have strongest correlation with SalePrice.

Plotting year build vs sale price.

As you see in today's market, older homes usually that are kept in good condition can be a bit pricier than newer homes, because it is seen as "vintage". So I want to see what happens between the two variables.

```
ggplot(data = index[index$SalePrice > 0,], aes(x = YearBuilt, y = SalePrice)) +
  geom_point() +
  geom_smooth(method = NULL, se = T, colour = "blue", linetype = "solid") +
  labs(x = "Year Built",
       y = "Sale Price",
       title = "Year Built vs. Sale Price"
       )
```



# KNN Method

Indexing and Scaling

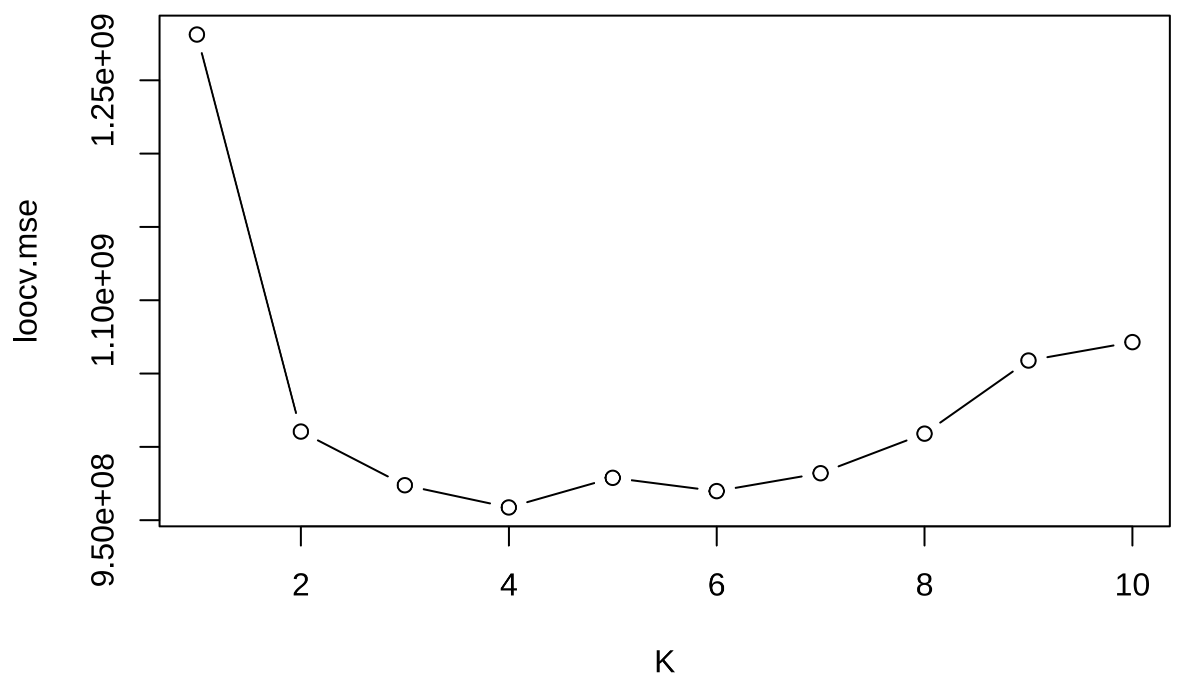
```
index <- rbind(train, test) #binding train and test
index.model <- scale(index) #scaled indexed data set

index.train.x <- index.model[1:nrow(train),]
index.test.x <- index.model[-(1:nrow(train)),]
```

First, to use KNN method I have to bind the train and test data together so I can then scale it, creating an indexed training and test data set.

Find tuning parameter K and error rate

```r
set.seed(1)
loocv.mse <- rep(0,10)
counter <- 0
for(k in 1:10){
counter <- counter + 1 # counter for k
cvknn <- knn.reg(index.train.x, NULL, train$SalePrice, k = k)
## the little k here is the number of nearest neighbors not k-fold
## X.train is the training input
## y.train is the training output
loocv.mse[counter] <- mean(cvknn$residuals^2)
}
plot(1:10, loocv.mse, type="b", xlab="K")
```



```r
which.min(loocv.mse) # gives us the K in which has the best error rate
```

```
## [1] 4
```

LOOCV method tells me that K = 4 is the best K value to go with. So this is how why I choose the tuning parameter of K = 4

Perform prediction with the tuning parameter

```
# Use K = 4, since the LOOCV method says K = 4.
knn.pred <- knn.reg(train = index.train.x, test = index.test.x,
                    y = train$SalePrice, k = 4)$pred
knn.pred[1:25]
```

```
##  [1] 111350.0 150750.0 174875.0 173000.0 157260.0 173875.0 160125.0 165730.5
##  [9] 145562.5 123100.0 207587.5  91875.0  89875.0 152150.0 109375.0 273428.2
## [17] 221182.0 215625.0 239250.0 292125.0 283062.5 175596.2 189083.0 166758.8
## [25] 184091.2
```

Show the first 25 predictions of the SalePrice. This looks accurate considering how much a home in Iowa would vary depending on the area.

KNN MSE

```
knn.error <- mean((knn.pred-test$SalePrice)^2)
```

Linear Regression Analysis

Fit a linear regression model with all predictors

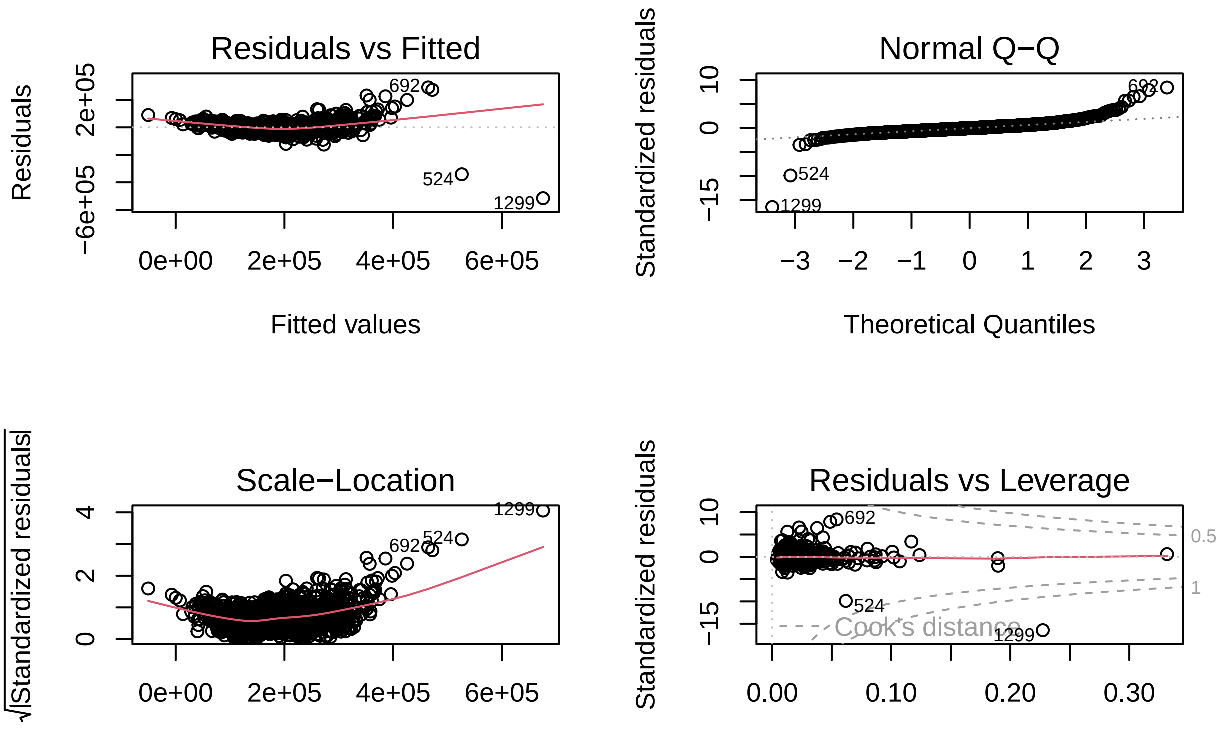
```
train.fit <- lm(SalePrice ~., data = train)
summary(train.fit)
```

```
##
## 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
```

Plenty of variables that are significant with the response variable. Almost all of them have some significance.

Perform residual diagnostics

```
par(mfrow=c(2,2))
plot(train.fit)
```



```r
ncvTest(train.fit) # non-constant variance testing
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 4084.926, Df = 1, p = < 2.22e-16
```

There are visible outliers in the residual diagnostics. I will remove them for the next plot. NCV test to check for heteroscedasticity, the p-value is very low which means indicates some significance in the model.

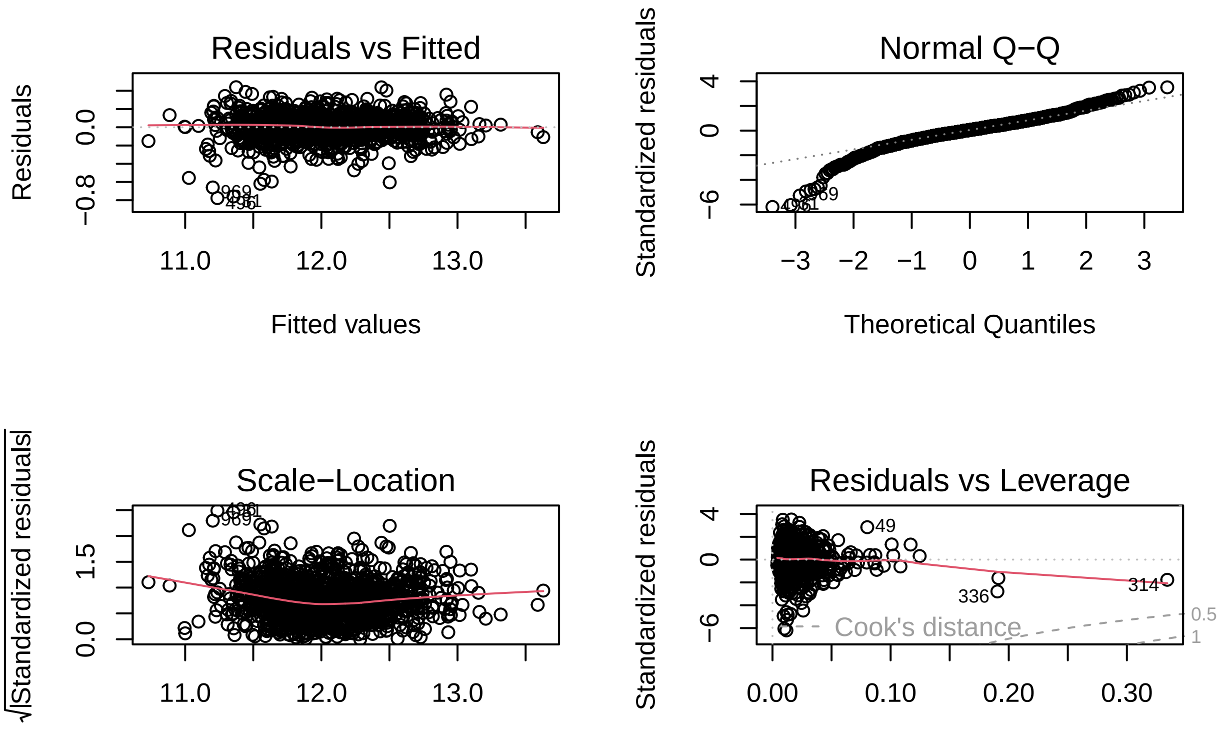
Checking other remedy

```r
train.fit_2 <- lm(log(SalePrice) ~., data = train[-c(524, 633, 1299),]) # removing the outliers
summary(train.fit_2)
```

```
##
## Call:
## lm(formula = log(SalePrice) ~ ., data = train[-c(524, 633, 1299),
##     ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.77648 -0.06026  0.00591  0.07206  0.43888
```

```
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)    
## (Intercept)    2.878e+00  4.711e-01   6.110 1.28e-09 ***
## LotArea        2.543e-06  3.635e-07   6.996 4.02e-12 ***
## OverallQual    7.289e-02  4.268e-03  17.078  < 2e-16 ***
## OverallCond    5.087e-02  3.667e-03  13.874  < 2e-16 ***
## YearBuilt      2.805e-03  2.069e-04  13.554  < 2e-16 ***
## YearRemodAdd   1.094e-03  2.373e-04   4.610 4.38e-06 ***
## BsmtFinSF1     1.997e-04  1.748e-05  11.427  < 2e-16 ***
## BsmtFinSF2     1.374e-04  2.552e-05   5.384 8.51e-08 ***
## BsmtUnfSF      1.211e-04  1.530e-05   7.913 4.99e-15 ***
## X1stFlrSF      2.680e-04  2.093e-05  12.803  < 2e-16 ***
## X2ndFlrSF      2.376e-04  1.778e-05  13.366  < 2e-16 ***
## LowQualFinSF   1.178e-04  7.090e-05   1.662 0.096767 .  
## BsmtFullBath   2.115e-02  9.500e-03   2.226 0.026167 *  
## BsmtHalfBath  -5.711e-03  1.479e-02  -0.386 0.699505    
## FullBath       1.512e-02  1.018e-02   1.485 0.137867    
## HalfBath       1.816e-02  9.585e-03   1.895 0.058277 .  
## BedroomAbvGr  -7.819e-03  6.095e-03  -1.283 0.199720    
## KitchenAbvGr  -9.748e-02  1.753e-02  -5.561 3.19e-08 ***
## TotRmsAbvGrd   1.156e-02  4.437e-03   2.606 0.009268 ** 
## Fireplaces     3.804e-02  6.379e-03   5.963 3.12e-09 ***
## GarageCars     3.575e-02  1.044e-02   3.425 0.000633 ***
## GarageArea     1.078e-04  3.515e-05   3.067 0.002205 ** 
## WoodDeckSF     4.828e-05  2.848e-05   1.695 0.090258 .  
## MoSold         6.576e-04  1.233e-03   0.533 0.593816    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.126 on 1433 degrees of freedom
## Multiple R-squared:  0.902,  Adjusted R-squared:  0.9005 
## F-statistic: 573.7 on 23 and 1433 DF,  p-value: < 2.2e-16
```

```
par(mfrow=c(2,2))
plot(train.fit_2)
```



The diagnostics look better, especially for the normal Q-Q graph. The remedy used is a log transformation for the SalePrice variable. Furthermore, removing outliers should decrease the error.

Explaining some significant coefficients from the linear model

Some important variables worth noting (highly significant) are OverallQual, OverallCond, YearBuilt and LotArea. Obviously, these are all factors you would expect to be important in regards to housing prices. Interesting as well are BsmtFinSF1 and BsmtFinSF2, which do not share the same significance. Why could this be?

Perform prediction

```
linear.pred <- exp(predict(train.fit_2, test))
head(linear.pred)
```

```
##        1        2        3        4        5        6
## 119792.5 155447.4 176137.0 197615.4 182525.0 171205.9
```

Predicting a few housing prices given the data. I apply exp() to the predict function because I'm using a log transformation on train.fit_2.

Error Rate for Linear Regression

```
set.seed(1)
glm.train.fit_2 <- glm(log(SalePrice) ~., data = train)
cv.glm.train.fit_2 <- cv.glm(train, glm.train.fit_2, K = 10)
cvmse.linear <- cv.glm.train.fit_2$delta[2]
cvmse.linear
```

```
## [1] 0.02508612
```

0.02508612 is the cv error rate I get for Linear Regression.

```
lin.error <- mean((linear.pred-test$SalePrice)^2)
```

```
mean((train.fit_2$residuals)^2)
```

```
## [1] 0.01561881
```

Subset Selection Method

Initial fitting

```
subset.fit <- regsubsets(log(SalePrice) ~ ., data = train, nvmax = ncol(train))
names(subset.fit)
```

```
##  [1] "np"        "nrbar"     "d"         "rbar"      "thetab"    "first"
##  [7] "last"      "vorder"    "tol"       "rss"       "bound"     "nvmax"
## [13] "ress"      "ir"        "nbest"     "lopt"      "il"        "ier"
## [19] "xnames"    "method"    "force.in"  "force.out" "sserr"     "intercept"
## [25] "lindep"    "nullrss"   "nn"        "call"
```

```
subsum <- summary(subset.fit)
subsum$rsq
```

```
##  [1] 0.6677904 0.7241287 0.7617143 0.7972648 0.8205223 0.8338734 0.8429616
##  [8] 0.8487207 0.8515875 0.8534964 0.8554266 0.8568384 0.8575458 0.8580521
## [15] 0.8586758 0.8593499 0.8599193 0.8603954 0.8605692 0.8606782 0.8607702
## [22] 0.8608385 0.8608596
```

Best Subset

Which subset selection method do you use? Best subset, forward stepwise, or backward stepwise? And explain why you choose this method.

```
par(mfrow=c(2,2))
plot(subsum$rss ,xlab="Number of Variables ", ylab="RSS", type="l")
plot(subsum$adjr2 ,xlab="Number of Variables ", ylab="Adjusted RSq", type="l")
print(c('Adjusted RSquared', which.max(subsum$adjr2)))
```

```
## [1] "Adjusted RSquared" "20"
```

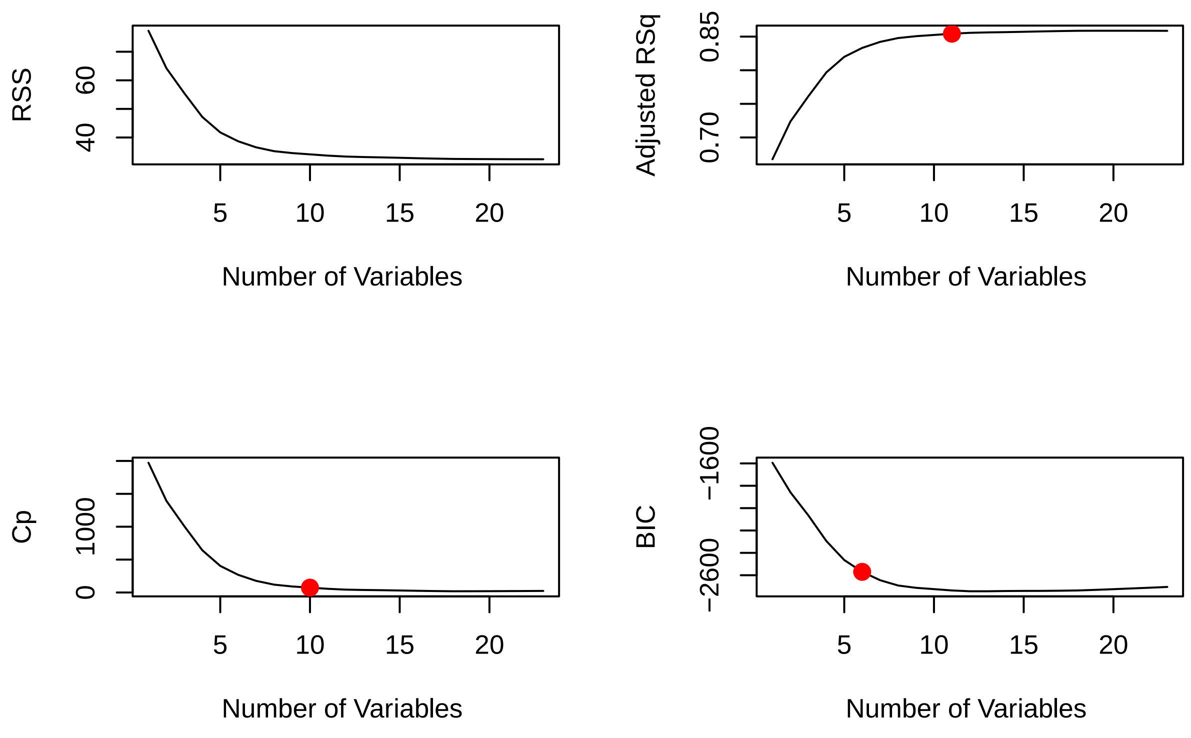
```
points(11,subsum$adjr2[11], col = "red", cex = 2, pch = 20)
plot(subsum$cp, xlab = "Number of Variables ", ylab = "Cp", type = "l")
print(c('Cp',which.min(subsum$cp)))
```

```
## [1] "Cp" "18"
```

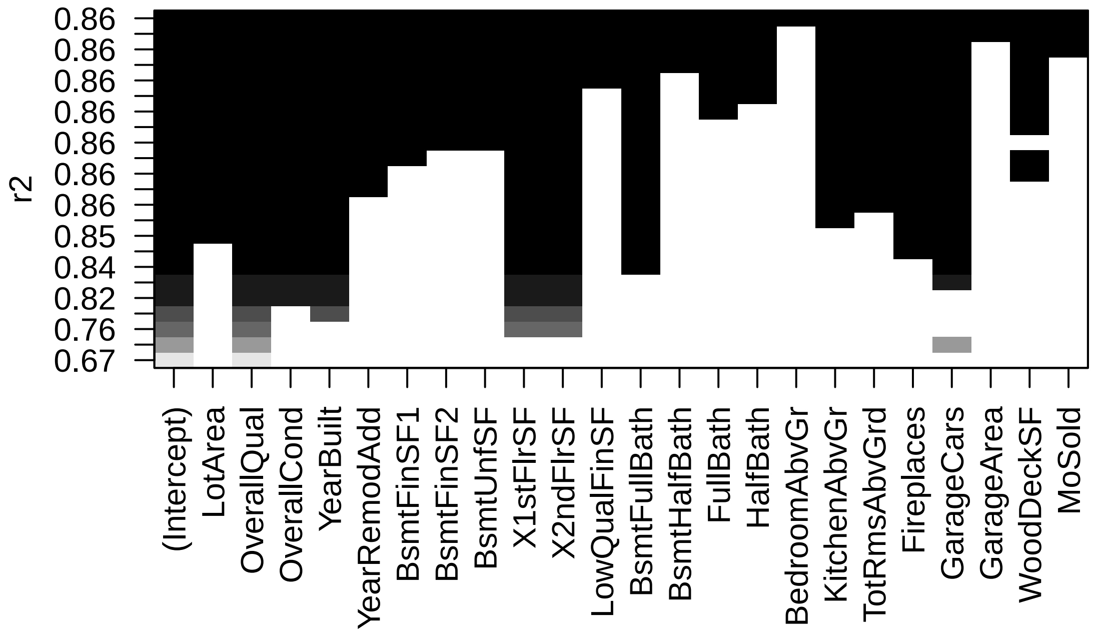
```
points(10,subsum$cp[10], col = "red", cex = 2, pch = 20)
print(c('BIC',which.min(subsum$bic)))
```

```
## [1] "BIC" "12"
```

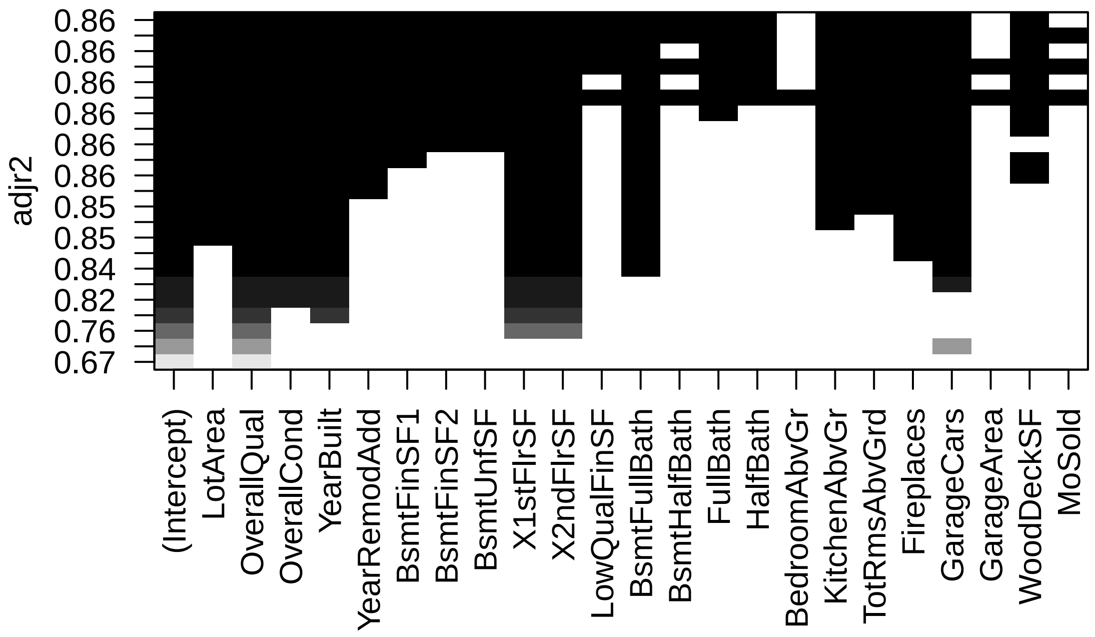
```
plot(subsum$bic, xlab = "Number of Variables ", ylab = "BIC", type = "l")
points(6,subsum$bic[6], col = "red", cex = 2, pch = 20)
```



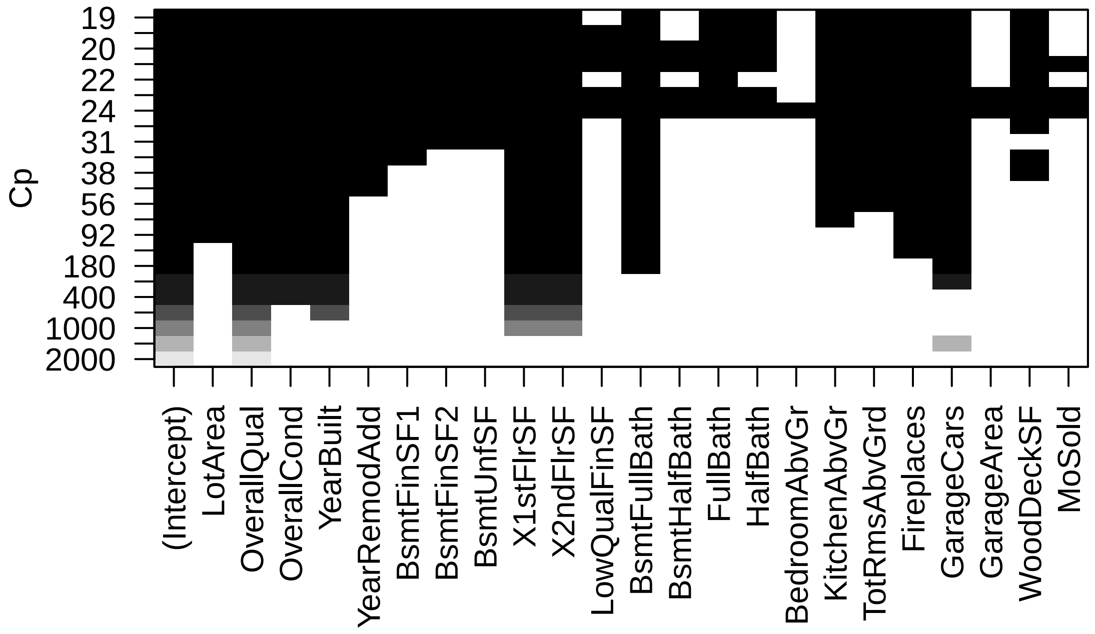
```r
plot(subset.fit,scale="r2")
```



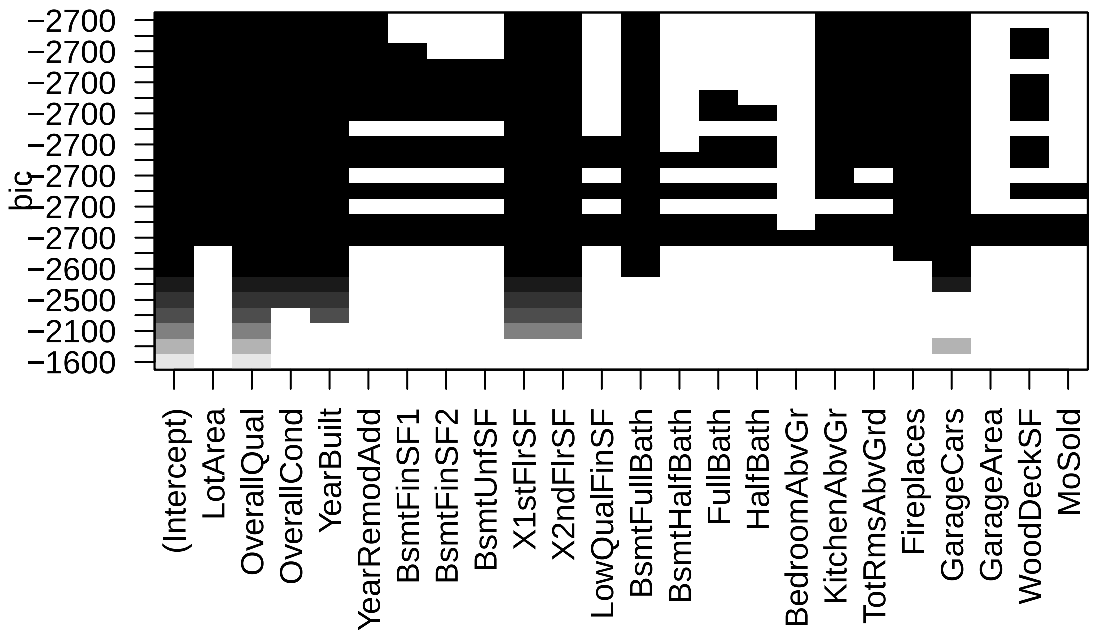
```r
plot(subset.fit,scale="adjr2")
```



```
plot(subset.fit,scale="Cp")
```



```r
plot(subset.fit,scale="bic")
```



Forward Stepwise

```
fit.fwd <- regsubsets(log(SalePrice) ~., data = train, nvmax = ncol(train), method = "forward")
par(mfrow=c(2,2))
fit.fwd.summary <- summary(fit.fwd)
plot(fit.fwd.summary$rss ,xlab="Number of Variables ", ylab="RSS", type="l")
plot(fit.fwd.summary$adjr2 ,xlab="Number of Variables ", ylab="Adjusted RSq", type="l")
print(c('Adjusted RSquared', which.max(fit.fwd.summary$adjr2)))
```

```
## [1] "Adjusted RSquared" "20"
```

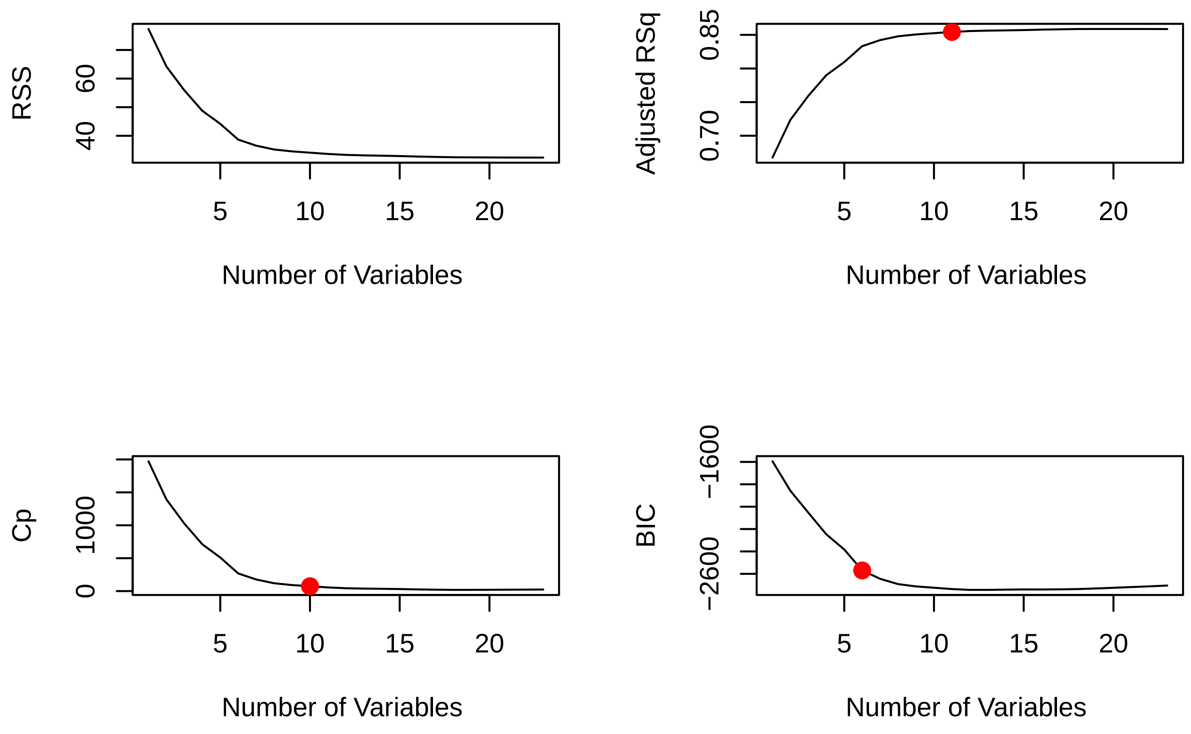
```
points(11, fit.fwd.summary$adjr2[11], col = "red", cex = 2, pch = 20)
plot(fit.fwd.summary$cp, xlab = "Number of Variables ", ylab = "Cp", type = "l")
print(c('Cp', which.min(fit.fwd.summary$cp)))
```

```
## [1] "Cp" "18"
```

```
points(10,fit.fwd.summary$cp[10], col = "red", cex = 2, pch = 20)
print(c('BIC', which.min(fit.fwd.summary$bic)))
```

```
## [1] "BIC" "12"
```

```
plot(fit.fwd.summary$bic, xlab = "Number of Variables ", ylab = "BIC", type = "l")
points(6,fit.fwd.summary$bic[6], col = "red", cex = 2, pch = 20)
```



Backward Stepwise

```
fit.bwd <- regsubsets(log(SalePrice) ~., data = train, nvmax = ncol(train), method = "backward")
par(mfrow=c(2,2))
fit.bwd.summary <- summary(fit.bwd)
plot(fit.bwd.summary$rss ,xlab="Number of Variables ", ylab="RSS", type="l")
plot(fit.bwd.summary$adjr2 ,xlab="Number of Variables ", ylab="Adjusted RSq", type="l")
print(c('Adjusted RSquared', which.max(fit.bwd.summary$adjr2)))
```

```
## [1] "Adjusted RSquared" "20"
```

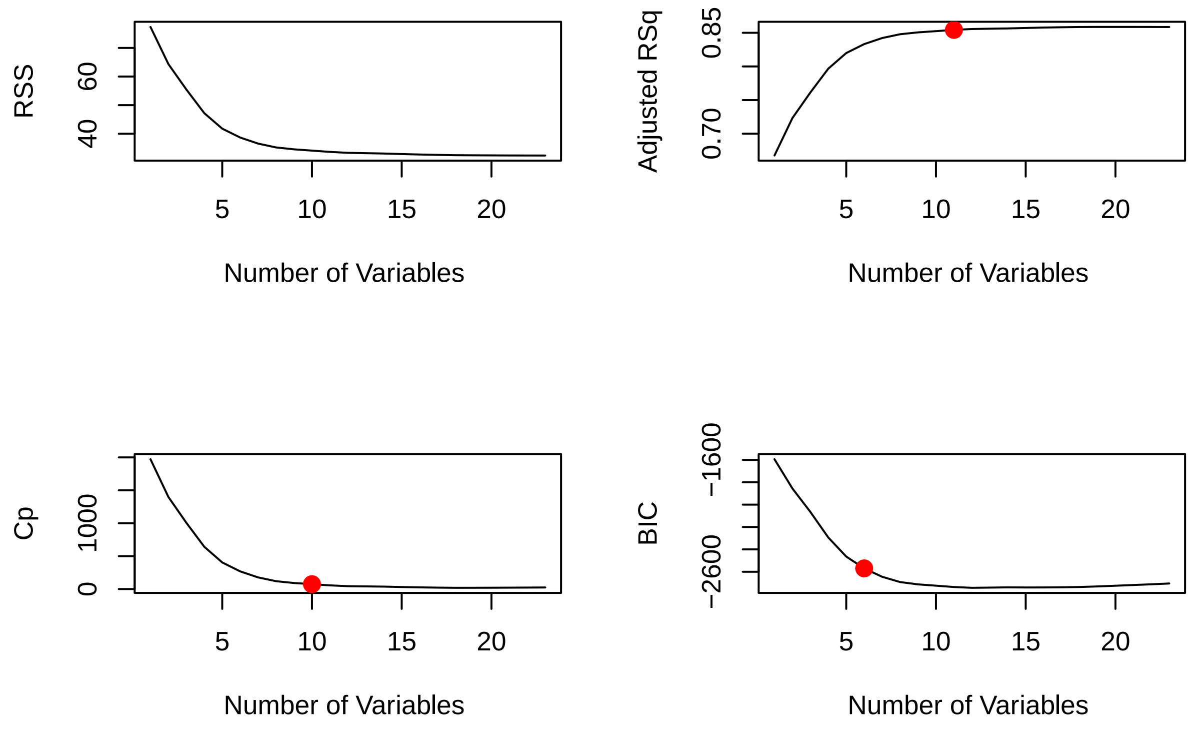
```
points(11, fit.bwd.summary$adjr2[11], col = "red", cex = 2, pch = 20)
plot(fit.bwd.summary$cp, xlab = "Number of Variables ", ylab = "Cp", type = "l")
print(c('Cp', which.min(fit.bwd.summary$cp)))
```

```
## [1] "Cp" "18"
```

```
points(10,fit.bwd.summary$cp[10], col = "red", cex = 2, pch = 20)
print(c('BIC', which.min(fit.bwd.summary$bic)))
```

```
## [1] "BIC" "12"
```

```
plot(fit.bwd.summary$bic, xlab = "Number of Variables ", ylab = "BIC", type = "l")
points(6,fit.bwd.summary$bic[6], col = "red", cex = 2, pch = 20)
```



The forward and backward step selection plots are the same. This could be due to them picking the best variables for both selections. I will choose the best subset selection model though, since the number of variables is small.

Test Error for Subset Selection

Get initial function

```
predict.regsubsets <- function (object, newdata , id, ...){
  form <- as.formula(object$call[[2]])  # formula of null model
  mat <- model.matrix(form, newdata)    # building an "X" matrix from newdata
  coefi <- coef(object, id = id)        # coefficient estimates associated with the object model contai
  xvars <- names(coefi)            # names of the non-zero coefficient estimates
  return(mat[,xvars] %*% coefi)    # X[,non-zero variables] %*% Coefficients[non-zero variables]
}
```

Perform cv on all selections

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

cv.error.sub.fit <- rep(0, ncol(train[,-1]))
cv.error.fwd.fit <- rep(0, ncol(train[,-1]))
cv.error.bwd.fit <- rep(0, ncol(train[,-1]))

for (i in 1:ncol(train[,-ncol(train)])) {
  sub.error <- rep(0,10)
  fwd.error <- rep(0,10)
  bwd.error <- rep(0,10)

  for (k in 1:10){
    train.f <- train[fold.index != k,]
    test.f <- train[fold.index == k,]
    true <- test.f[,"SalePrice"]
    sub.fit <- regsubsets(log(SalePrice) ~., data = train.f, nvmax = ncol(train[,-1]))
    fwd.fit <- regsubsets(log(SalePrice) ~., data = train.f, nvmax = ncol(train[,-1]), method='forward')
    bwd.fit <- regsubsets(log(SalePrice) ~., data = train.f, nvmax = ncol(train[,-1]), method='backward
    sub.predictions <- exp(predict.regsubsets(sub.fit, test.f, id = i))
    fwd.predictions <- exp(predict.regsubsets(fwd.fit, test.f, id = i))
    bwd.predictions <- exp(predict.regsubsets(bwd.fit, test.f, id = i))

    sub.error[k] <- mean((sub.predictions - true)^2)
    fwd.error[k] <- mean((fwd.predictions - true)^2)
    bwd.error[k] <- mean((bwd.predictions - true)^2)
  }
  cv.error.sub.fit[i] <- mean(sub.error)
  cv.error.fwd.fit[i] <- mean(fwd.error)
  cv.error.bwd.fit[i] <- mean(bwd.error)
}
```

Subset Selection Errors

```
print(c(cv.error.sub.fit, which.min(cv.error.sub.fit))) # Best Subset + Coefficient numbers
```

```
##  [1] 2091250881 2528087973 3499983801 3339565791 3600723003 7561874432
##  [7] 5375697637 6108855075 6482499785 6511944925 8156248100 8111577026
## [13] 8431144590 9348596878 8733051398 8015092426 8380618992 8366329719
## [19] 8404912009 8305670485 8180221683 8037973005 8045746486          1
```

```
print(c(cv.error.fwd.fit, which.min(cv.error.fwd.fit))) # Forward Stepwise + Coefficient numbers
```

```
##  [1] 2091250881 2528087973 3566863710 3364471746 3630483113 7561874432
##  [7] 5375697637 6108855075 6482499785 6510127217 5958522461 7458008315
## [13] 8431144590 9346073775 8737465631 8028602631 8389409813 8371880120
## [19] 8404912009 8305670485 8180221683 8037973005 8045746486          1
```

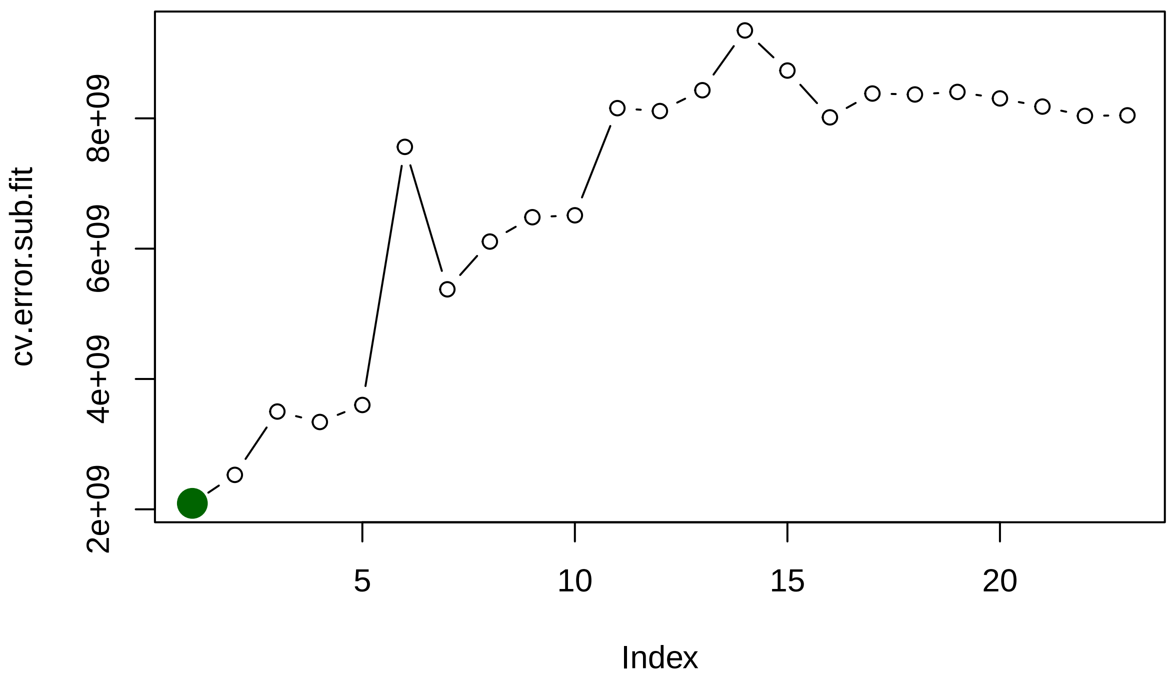
```
print(c(cv.error.bwd.fit, which.min(cv.error.bwd.fit))) # Backward Stepwise + Coefficient numbers
```

```
##  [1] 2091250881 2358213472 3334518306 3339565791 3600723003 7561874432
##  [7] 5375697637 6108855075 7628639853 8716816044 8156248100 8115650310
## [13] 8424942746 9332947788 8711534004 8012220553 8380618992 8366329719
## [19] 8404912009 8305670485 8180221683 8037973005 8045746486          1
```

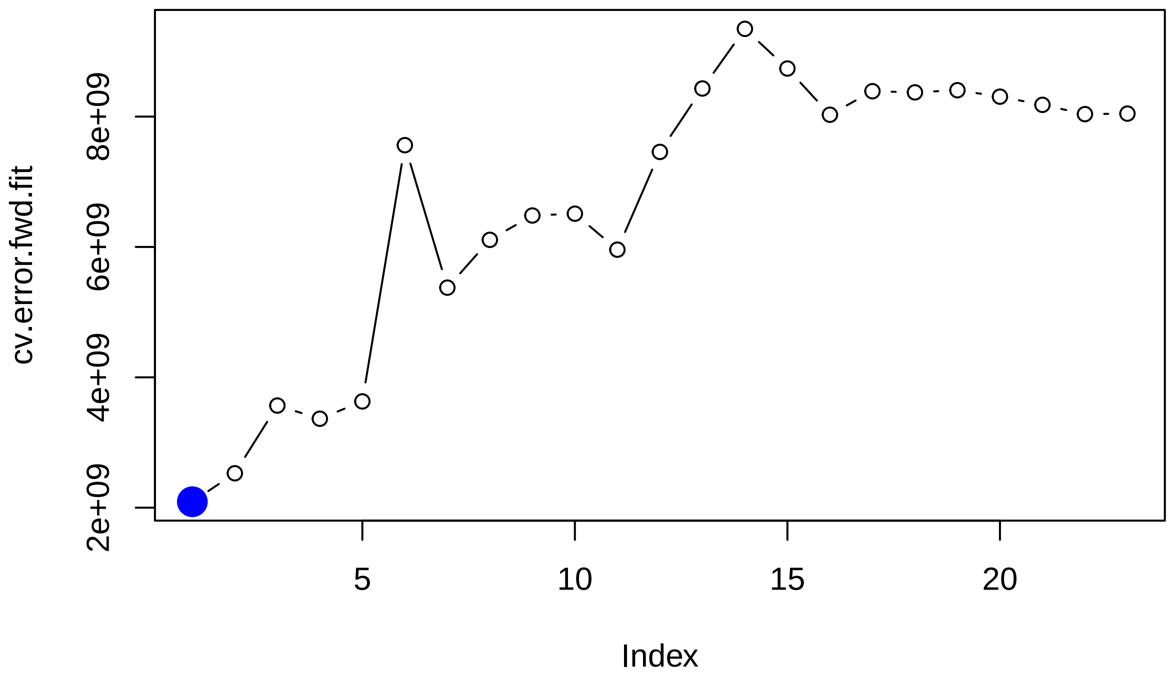
The minimum coefficient number for each of the three models is 1, leading me to believe all selection models are equally the same.

Plot each selection error

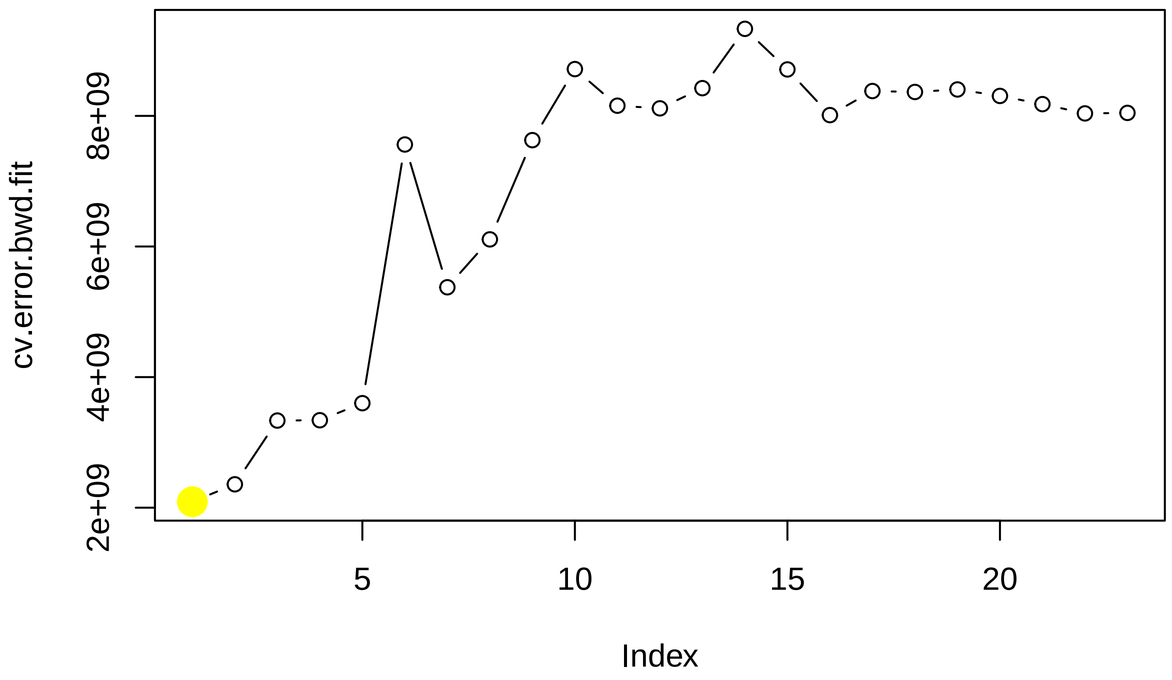
```
par(mfrow=c(1,1))
plot(cv.error.sub.fit, type = "b")
points(which.min(cv.error.sub.fit), cv.error.sub.fit[which.min(cv.error.sub.fit)],
       col = "darkgreen", cex = 3, pch = 20)
```



```
par(mfrow=c(1,1))
plot(cv.error.fwd.fit, type = "b")
points(which.min(cv.error.fwd.fit), cv.error.fwd.fit[which.min(cv.error.fwd.fit)],
       col = "blue", cex = 3, pch = 20)
```



```
par(mfrow=c(1,1))
plot(cv.error.bwd.fit, type = "b")
points(which.min(cv.error.bwd.fit), cv.error.bwd.fit[which.min(cv.error.bwd.fit)],
       col = "yellow", cex = 3, pch = 20)
```



Again, all the same. Each plot is showing the minimum coefficient value is 1, which is what my other data shows.

Which criterion do you use for selecting your model? Cp, BIC, Adjusted R-squared, or CV? And explain why you choose this criterion.

When selecting my model, I am taking the BIC into consideration most. This is because it takes in the least amount of variables to have a successful output.

Explain some of the coefficients in the models

```
coef(subset.fit, which.min(cv.error.sub.fit))
```

```
## (Intercept) OverallQual
##   10.584442    0.236028
```

```
coef(fit.fwd, which.min(cv.error.fwd.fit))
```

```
## (Intercept) OverallQual
##   10.584442    0.236028
```

```
coef(fit.bwd, which.min(cv.error.bwd.fit))
```

```
## (Intercept) OverallQual
##   10.584442    0.236028
```

In this case, I find the coefficients from all models because they are all selecting the same variable. It is interesting to note that the intercept for each model, they are all the same.

Perform prediction

```
subset.predictions <- exp(predict.regsubsets(subset.fit, newdata = test, id = which.min(cv.error.sub.fi
head(sub.predictions)
```

```
##        [,1]
## 9  152017.2
## 25 160297.4
## 28 280318.2
## 44 127302.0
## 51 170002.8
## 54 328670.4
```

```
fwd.predictions <- exp(predict.regsubsets(subset.fit, newdata = test, id = which.min(cv.error.fwd.fit))
head(fwd.predictions)
```

```
##       [,1]
## 1 128615.4
## 2 162854.1
## 3 128615.4
## 4 162854.1
## 5 261101.8
## 6 162854.1
```

```
bwd.predictions <- exp(predict.regsubsets(subset.fit, newdata = test, id = which.min(cv.error.bwd.fit))
head(bwd.predictions)
```

```
##       [,1]
## 1 128615.4
## 2 162854.1
## 3 128615.4
## 4 162854.1
## 5 261101.8
## 6 162854.1
```

The predictions for all are the same as expected, since all the models are the same too.

Stepwise Selection MSE's

```
forward.error <- mean((fwd.predictions - test$SalePrice)^2)
backward.error <- mean((bwd.predictions - test$SalePrice)^2)
subset.error <- mean((subset.predictions - test$SalePrice)^2)
```

Shrinkage Methods

Initial fitting

```
x.fit <- model.matrix(log(SalePrice) ~., data = train)[,-1]
head(x.fit)
```

```
##   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
## 1        548          0      2
## 2        460        298      5
## 3        608          0      9
## 4        642          0      2
## 5        836        192     12
## 6        480         40     10
```

```
trsp <- log(train$SalePrice) # make the response variable
```

Tuning parameters for Ridge and Lasso

To find the best tuning parameters for Ridge and Lasso, I will perform cross-validation on the model, and this choose the lowest lambda value, which should provide me with the best parameters.

Ridge Regression

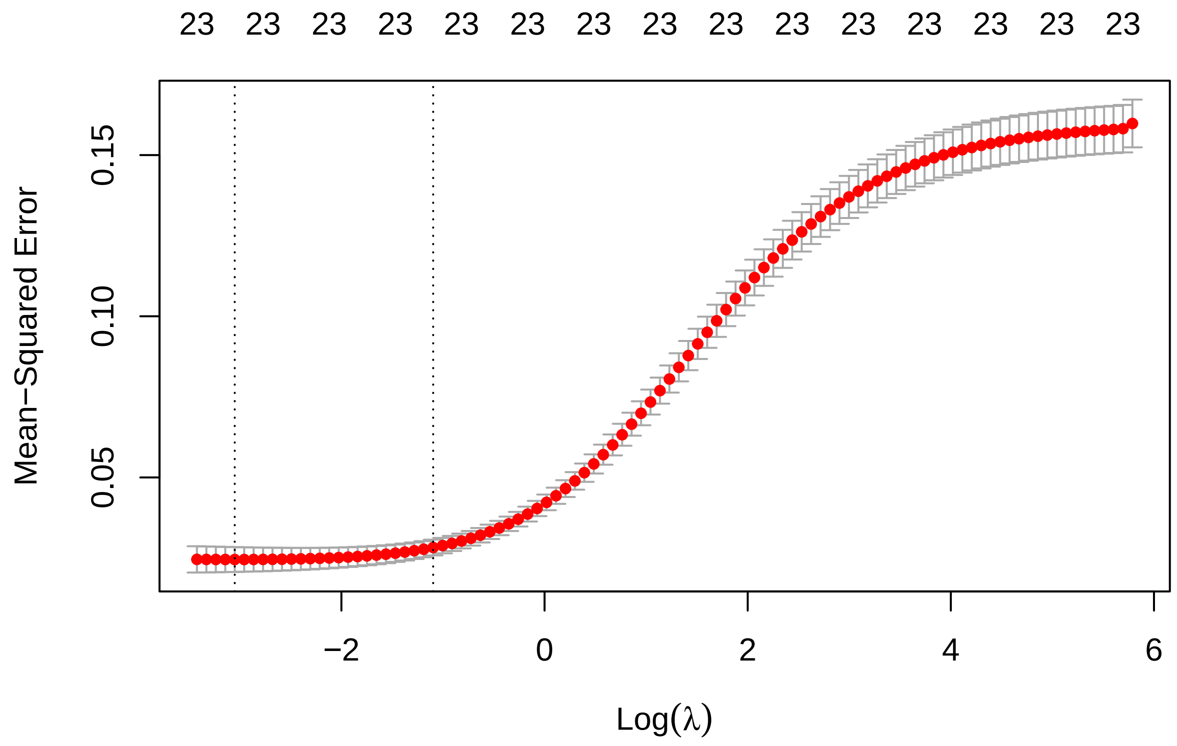
```
set.seed(1)
ridge.fit <- glmnet(x.fit, trsp, alpha = 0)
names(ridge.fit)
```

```
##  [1] "a0"        "beta"      "df"        "dim"       "lambda"    "dev.ratio"
##  [7] "nulldev"   "npasses"   "jerr"      "offset"    "call"      "nobs"
```

```
cv.ridge <- cv.glmnet(x.fit, trsp, alpha = 0, nfolds = 10)
bestridge_lambda <- cv.ridge$lambda.min # best tuning parameter
bestridge_lambda
```

```
## [1] 0.04734258
```

```
plot(cv.ridge)
```



The best lambda value corresponds with 0.04734258 for seed of 1 for Ridge regression.

Lasso Regression

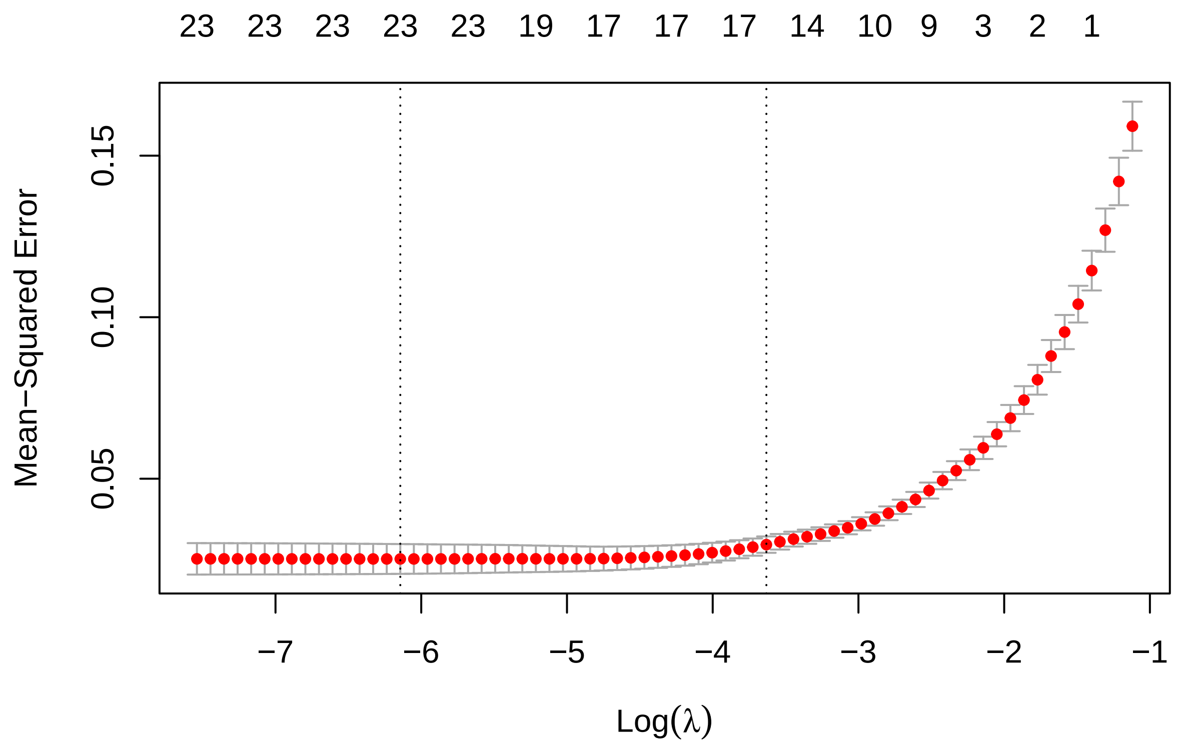
```r
set.seed(1)
lasso.fit <- glmnet(x.fit, trsp, alpha = 1)
names(lasso.fit)
```

```
##  [1] "a0"        "beta"      "df"        "dim"       "lambda"    "dev.ratio"
##  [7] "nulldev"   "npasses"   "jerr"      "offset"    "call"      "nobs"
```

```r
cv.lasso <- cv.glmnet(x.fit, trsp, alpha = 1, nfolds = 10)
bestlasso_lambda <- cv.lasso$lambda.min # best tuning parameter
bestlasso_lambda
```

```
## [1] 0.002146928
```

```r
plot(cv.lasso)
```



The best lambda value corresponds with 0.002146928. Lasso does better at interpretation because it turns more variables into zero, so there will be fewer variables to deal with during interpretation.

Explain the coefficients for Ridge and Lasso

```r
coef(ridge.fit, s = bestridge_lambda)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)   3.641225e+00
## LotArea       1.919329e-06
## OverallQual   7.537097e-02
## OverallCond   3.870735e-02
## YearBuilt     1.977977e-03
## YearRemodAdd  1.568023e-03
## BsmtFinSF1    8.237522e-05
## BsmtFinSF2    7.080739e-05
## BsmtUnfSF     5.729941e-05
## X1stFlrSF     1.553614e-04
## X2ndFlrSF     8.835735e-05
## LowQualFinSF  6.256883e-05
## BsmtFullBath  5.363994e-02
## BsmtHalfBath  1.592436e-02
## FullBath      5.538422e-02
## HalfBath      4.038365e-02
## BedroomAbvGr  6.859483e-03
## KitchenAbvGr -1.080762e-01
## TotRmsAbvGrd  2.269215e-02
## Fireplaces    5.555231e-02
## GarageCars    5.846278e-02
## GarageArea    1.144670e-04
## WoodDeckSF    1.089014e-04
## MoSold        1.517166e-03
```

These are the coefficient values for ridge regression when the best lambda value is chosen. All variables are included.

```r
coef(lasso.fit, s = bestlasso_lambda)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)   3.570992e+00
## LotArea       1.940513e-06
## OverallQual   8.699803e-02
## OverallCond   4.589673e-02
## YearBuilt     2.456467e-03
## YearRemodAdd  1.089413e-03
## BsmtFinSF1    5.945677e-05
## BsmtFinSF2    4.694423e-05
## BsmtUnfSF     3.420263e-05
## X1stFlrSF     2.086299e-04
## X2ndFlrSF     1.268396e-04
## LowQualFinSF  5.637944e-05
## BsmtFullBath  5.362777e-02
## BsmtHalfBath  9.003998e-03
## FullBath      3.504515e-02
## HalfBath      2.529157e-02
## BedroomAbvGr  2.591955e-03
## KitchenAbvGr -9.706370e-02
## TotRmsAbvGrd  1.975700e-02
## Fireplaces    4.851250e-02
## GarageCars    6.960237e-02
## GarageArea    4.181887e-05
## WoodDeckSF    8.554423e-05
## MoSold        5.572703e-04
```

These are the coefficient values for lasso regression when the best lambda value is chosen. Normally, some coefficients shouldn't have a value, but then because of the transformations all coefficients are included in this lasso regression.

Perform Predictions on Ridge and Lasso

```
ridge.pred <- exp(predict(ridge.fit, s = bestridge_lambda, newx = model.matrix(SalePrice ~., data = tes
head(ridge.pred)
```

```
##         s1
## 1 117080.7
## 2 146154.2
## 3 172010.4
## 4 198543.5
## 5 179154.4
## 6 177107.2
```

```
lasso.pred <- exp(predict(lasso.fit, s = bestlasso_lambda, newx = model.matrix(SalePrice ~., data = tes
head(lasso.pred)
```

```
##         s1
## 1 114180.7
## 2 146402.5
## 3 167890.9
## 4 195703.8
## 5 184415.9
## 6 175341.0
```

The predictions look on par for houses in Iowa, so I think it is safe to say that these are accurately predicted.

Error Rate for Ridge and Lasso

```
ridge.cvmse <- ridge.fit$dev.ratio[which.min(cv.ridge$lambda)]
lasso.cvmse <- lasso.fit$dev.ratio[which.min(cv.lasso$lambda)]
ridge.cvmse
```

```
## [1] 0.8593565
```

```
lasso.cvmse
```

```
## [1] 0.8608202
```

Lasso and Ridge MSE

```
lasso.error <- mean((lasso.pred-test$SalePrice)^2)
ridge.error <- mean((ridge.pred-test$SalePrice)^2)
```

Generalized Additive Model

```
train.gam <- gam(log(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) + s(KitchenAbvGr)
s(TotRmsAbvGrd) + s(Fireplaces) + s(GarageCars) + s(GarageArea) + s(WoodDeckSF) + s(MoSold), df = 4, da

train.gam
```

```
## Call:
## gam(formula = log(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) + s(KitchenAbvGr) + s(TotRmsAbvGrd) +
##     s(Fireplaces) + s(GarageCars) + s(GarageArea) + s(WoodDeckSF) +
##     s(MoSold), data = train, df = 4)
##
## Degrees of Freedom: 1459 total; 1381 Residual
## Residual Deviance: 20.99259
```

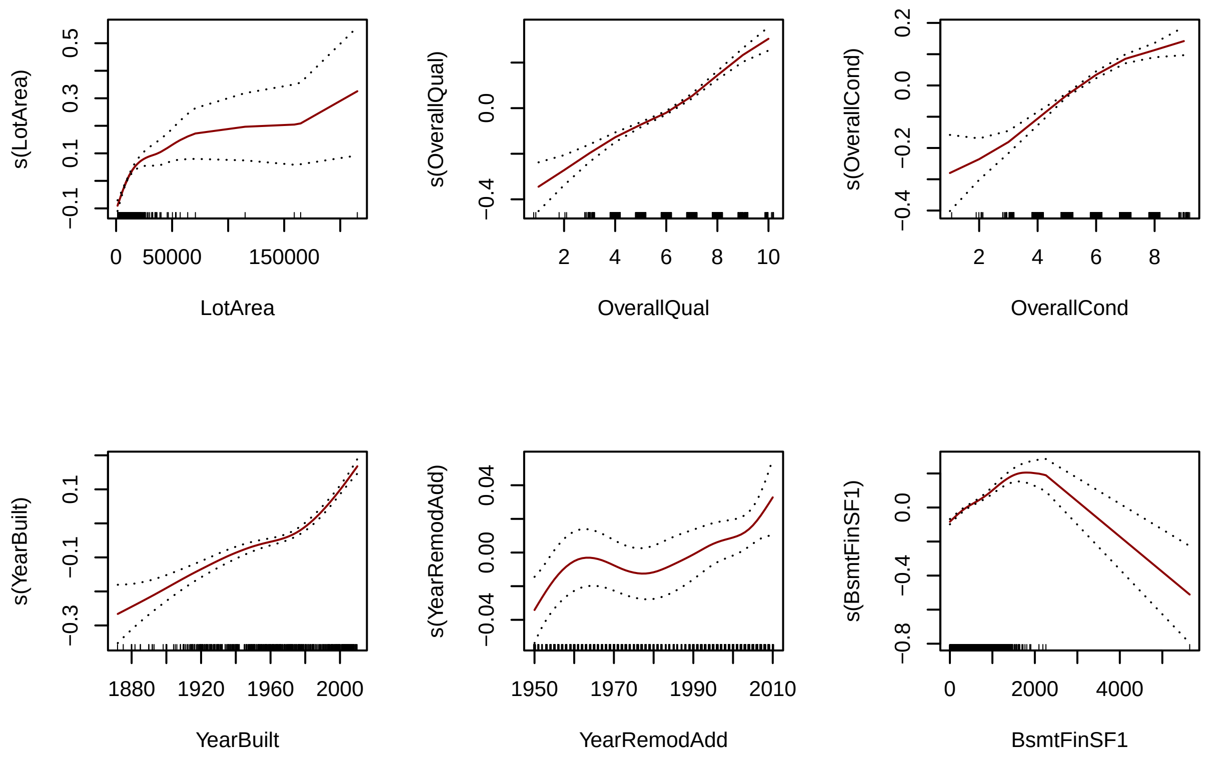
```
summary(train.gam)
```

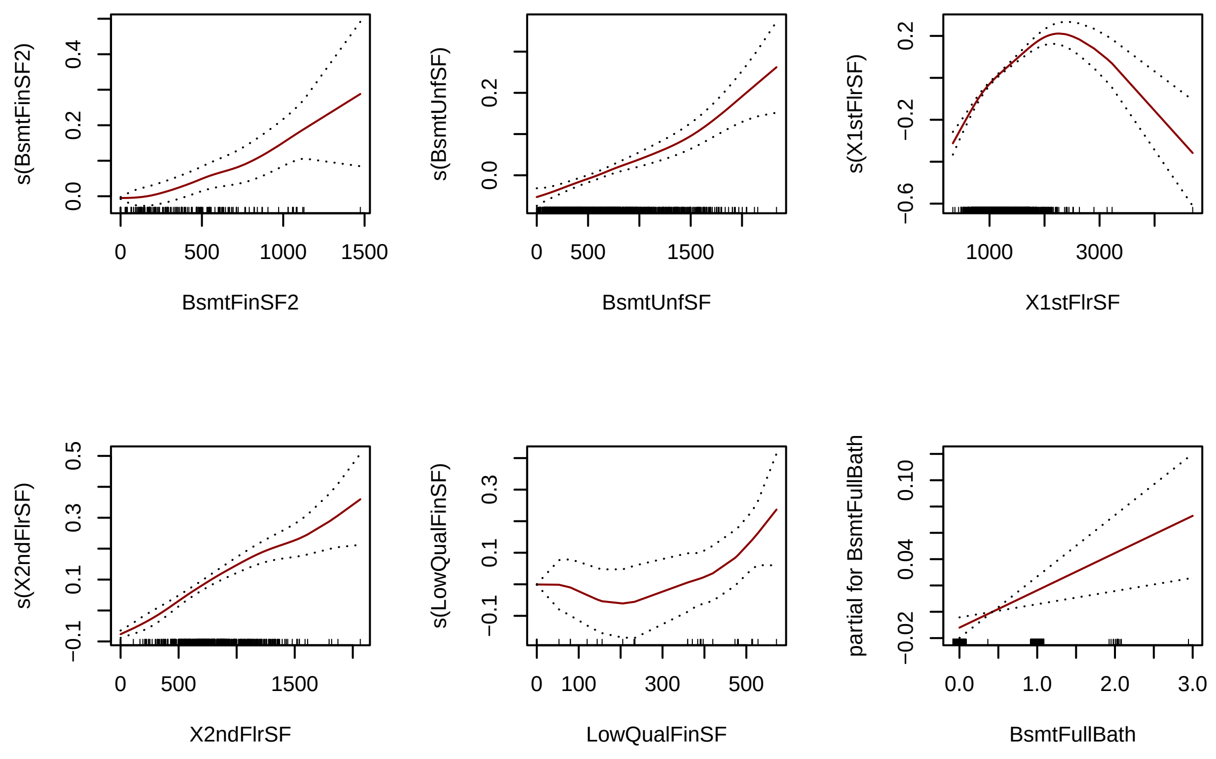
```
##
## Call: gam(formula = log(SalePrice) ~ s(LotArea) + s(OverallQual) +
##     s(OverallCond) + s(YearBuilt) + s(YearRemodAdd) + s(BsmtFinSF1) +
##     s(BsmtFinSF2) + s(BsmtUnfSF) + s(X1stFlrSF) + s(X2ndFlrSF) +
##     s(LowQualFinSF) + BsmtFullBath + BsmtHalfBath + FullBath +
##     HalfBath + s(BedroomAbvGr) + s(KitchenAbvGr) + s(TotRmsAbvGrd) +
##     s(Fireplaces) + s(GarageCars) + s(GarageArea) + s(WoodDeckSF) +
##     s(MoSold), data = train, df = 4)
## Deviance Residuals:
##       Min        1Q    Median        3Q       Max
## -1.154959 -0.048750  0.004168  0.063624  0.466136
##
## (Dispersion Parameter for gaussian family taken to be 0.0152)
##
##     Null Deviance: 232.8007 on 1459 degrees of freedom
## Residual Deviance: 20.9926 on 1381 degrees of freedom
## AIC: -1890.052
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                    Df  Sum Sq Mean Sq   F value    Pr(>F)
## s(LotArea)          1  17.353  17.353 1141.5498 < 2.2e-16 ***
## s(OverallQual)      1 135.032 135.032 8883.0832 < 2.2e-16 ***
## s(OverallCond)      1   0.428   0.428   28.1463 1.309e-07 ***
## s(YearBuilt)        1   8.975   8.975  590.4429 < 2.2e-16 ***
## s(YearRemodAdd)     1   0.507   0.507   33.3830 9.340e-09 ***
## s(BsmtFinSF1)       1   5.143   5.143  338.3097 < 2.2e-16 ***
## s(BsmtFinSF2)       1   0.410   0.410   26.9630 2.384e-07 ***
## s(BsmtUnfSF)        1   3.031   3.031  199.3638 < 2.2e-16 ***
## s(X1stFlrSF)        1   2.238   2.238  147.2006 < 2.2e-16 ***
## s(X2ndFlrSF)        1  15.393  15.393 1012.6055 < 2.2e-16 ***
## s(LowQualFinSF)     1   0.071   0.071    4.6686  0.030890 *
## BsmtFullBath        1   0.117   0.117    7.7187  0.005539 **
## BsmtHalfBath        1   0.017   0.017    1.0989  0.294699
## FullBath            1   0.016   0.016    1.0814  0.298567
## HalfBath            1   0.270   0.270   17.7887 2.629e-05 ***
## s(BedroomAbvGr)     1   0.064   0.064    4.1893  0.040869 *
## s(KitchenAbvGr)     1   0.498   0.498   32.7895 1.258e-08 ***
## s(TotRmsAbvGrd)     1   0.066   0.066    4.3136  0.037994 *
## s(Fireplaces)       1   0.781   0.781   51.3975 1.225e-12 ***
## s(GarageCars)       1   1.117   1.117   73.4616 < 2.2e-16 ***
## s(GarageArea)       1   0.004   0.004    0.2809  0.596195
## s(WoodDeckSF)       1   0.073   0.073    4.8131  0.028410 *
## s(MoSold)           1   0.000   0.000    0.0079  0.929188
## Residuals        1381  20.993   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 19.619 1.860e-12 ***
## s(OverallQual)         3  6.688 0.0001757 ***
## s(OverallCond)         3  4.820 0.0024198 **
## s(YearBuilt)           3 16.158 2.516e-10 ***
## s(YearRemodAdd)        3  3.335 0.0187915 *
## s(BsmtFinSF1)          3 42.986 < 2.2e-16 ***
## s(BsmtFinSF2)          3  1.339 0.2600753
## s(BsmtUnfSF)           3  1.662 0.1733706
## s(X1stFlrSF)           3 56.077 < 2.2e-16 ***
## s(X2ndFlrSF)           3  1.342 0.2590501
## s(LowQualFinSF)        3  2.470 0.0603542 .
## BsmtFullBath
## BsmtHalfBath
## FullBath
## HalfBath
## s(BedroomAbvGr)        3  2.163 0.0906425 .
## s(KitchenAbvGr)        2  1.789 0.1674941
## s(TotRmsAbvGrd)        3  1.069 0.3611188
## s(Fireplaces)          2  0.823 0.4394966
## s(GarageCars)          3  7.495 5.615e-05 ***
## s(GarageArea)          3  4.628 0.0031602 **
## s(WoodDeckSF)          3  0.714 0.5438097
## s(MoSold)              3  3.055 0.0274937 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

When choosing the best tuning parameter, I am using the default degree of freedom, 4, and mention this as the tuning parameter. From the above model, there are some clear variables that are not significant for this model, such as BsmtHalfBath and FullBath.

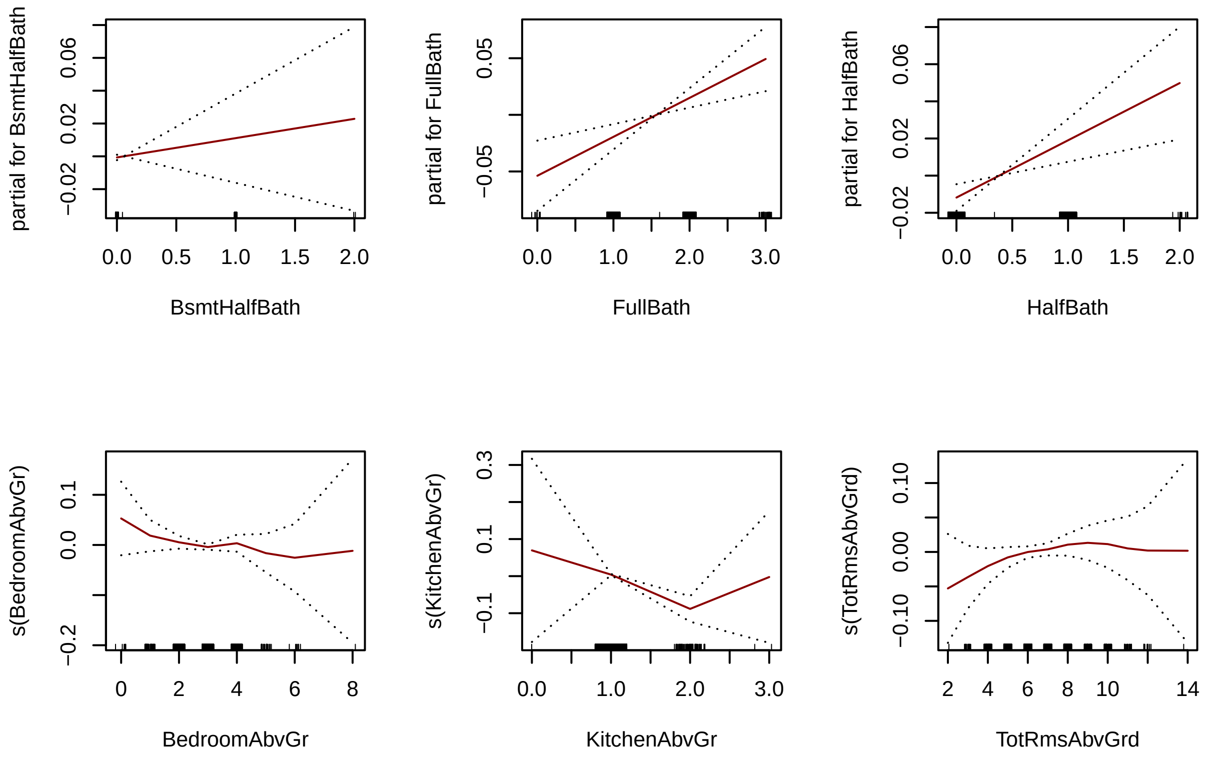
Plotting the GAM

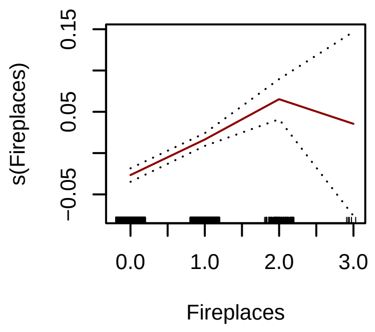
```r
par(mfrow = c(2,3))
plot(train.gam, se = T, col = "darkred")
```

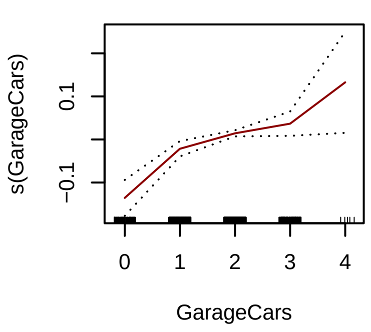


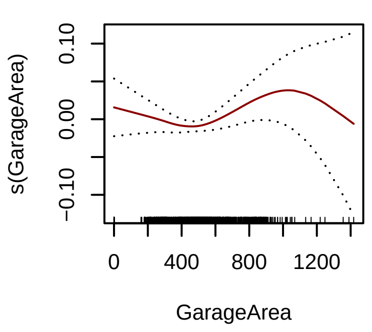


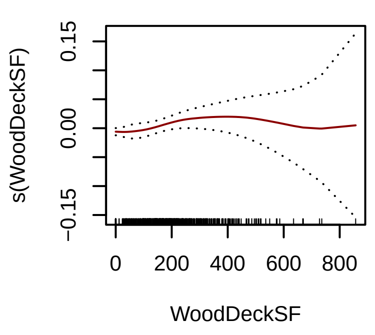
s(BsmtFinSF2)
0.0
0.2
0.4
0
500
1000
1500
BsmtFinSF2
s(BsmtUnfSF)
0.0
0.2
0
500
1500
BsmtUnfSF
s(X1stFlrSF)
−0.6
−0.2
0.2
1000
3000
X1stFlrSF
s(X2ndFlrSF)
−0.1
0.1
0.3
0.5
0
500
1500
X2ndFlrSF
s(LowQualFinSF)
−0.1
0.1
0.3
0
100
300
500
LowQualFinSF
partial for BsmtFullBath
−0.02
0.04
0.10
0.0
1.0
2.0
3.0
BsmtFullBath

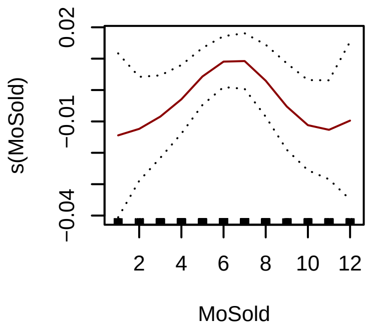

partial for BsmtHalfBath
BsmtHalfBath
partial for FullBath
FullBath
partial for HalfBath
HalfBath
s(BedroomAbvGr)
BedroomAbvGr
s(KitchenAbvGr)
KitchenAbvGr
s(TotRmsAbvGrd)
TotRmsAbvGrd











## Perform predictions on GAM

```
gam.pred <- exp(predict(train.gam, newdata = test))
head(gam.pred)
```

```
##        1        2        3        4        5        6
## 123398.0 164943.5 185457.0 206090.1 174268.7 166702.1
```

GAM MSE

```
gam.error <- mean((gam.pred-test$SalePrice)^2)
```

Checking coefficients for GAM

```
coef(train.gam, complete = T)
```

```
##     (Intercept)      s(LotArea)  s(OverallQual)  s(OverallCond)     s(YearBuilt)
##    2.938359e+00    2.800644e-06    6.960676e-02    5.543003e-02     3.122358e-03
## s(YearRemodAdd)   s(BsmtFinSF1)   s(BsmtFinSF2)    s(BsmtUnfSF)     s(X1stFlrSF)
##    7.883509e-04    1.550240e-04    1.309136e-04    1.018974e-04     2.248740e-04
##    s(X2ndFlrSF) s(LowQualFinSF)    BsmtFullBath    BsmtHalfBath         FullBath
##    2.197160e-04    1.617756e-04    2.832847e-02    1.176769e-02    3.436877e-02
##        HalfBath s(BedroomAbvGr) s(KitchenAbvGr) s(TotRmsAbvGrd)   s(Fireplaces)
##    3.081861e-02   -5.421726e-03   -8.229685e-02    4.972428e-03    4.325328e-02
##   s(GarageCars)   s(GarageArea)   s(WoodDeckSF)       s(MoSold)
##    4.656661e-02    1.826115e-05    6.101402e-05   -1.069907e-04
```

Decision Tree Model

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

```
##
## 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
```

There are only 6 variables the decision tree decided to use. Also, the tree has 12 terminal nodes, which is relatively large, but not for this data set.

Glimpse at the tree

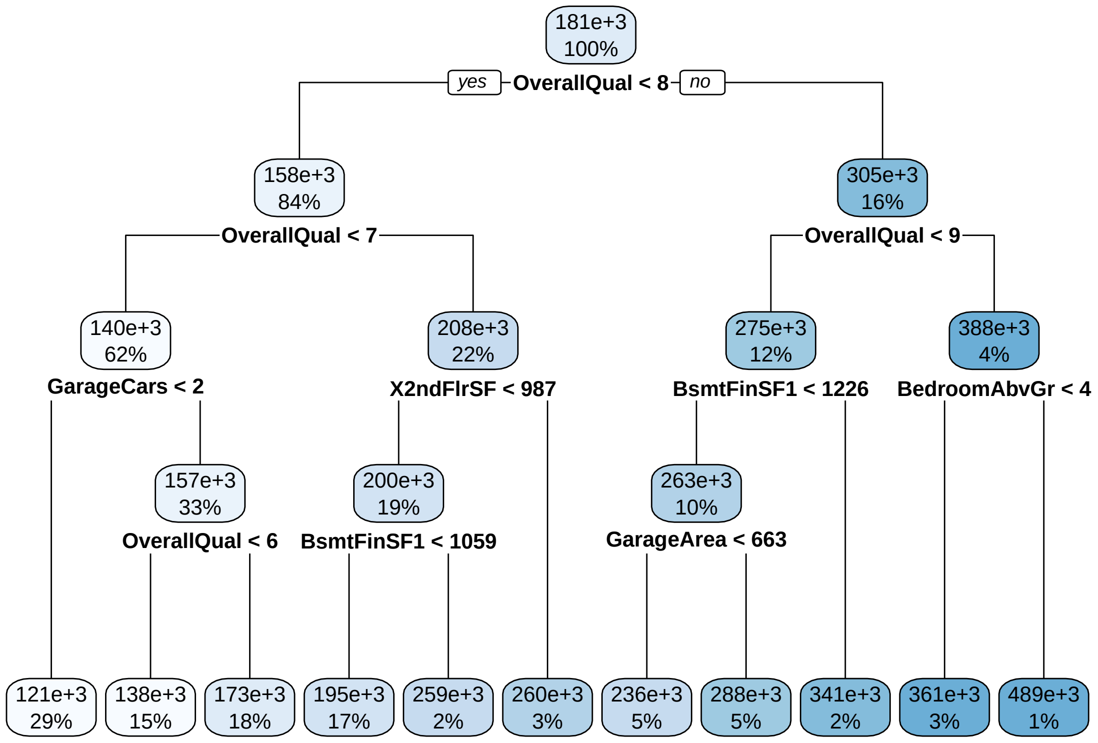
```
train.tree
```

```
## 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 *
```

Choosing node 7 here. This node asked whether or not OverallQual was greater than 8.5. If OverallQual was in this threshold, the tree split the average, and if the house had an OverallQual which was greater, then the price of the house is predicted to be higher than 388500.

Plot the tree

```r
r.tree <- rpart(SalePrice ~ ., data = train)
rpart.plot(r.tree)
```



Instead of plotting the tree regularly, use *rpart* to make the tree easily readable. The tree still gives the same results.

Decision Tree Prediction

```r
tree.pred <- predict(train.tree, newdata = test)
head(tree.pred)
```

```
##        1        2        3        4        5        6
## 121039.8 121039.8 138068.2 173210.9 236143.5 173210.9
```

Decision Tree Prediction Error

```r
tree.error <- mean((tree.pred-test$SalePrice)^2)
```

Decision Tree CV

```r
cv.train <- cv.tree(train.tree, FUN = prune.tree)
best.size <- cv.train$size[which.min(cv.train$dev)]
best.size
```

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

The best size for the tree appears to be K = 12, now I will prune a tree with the above tree size.

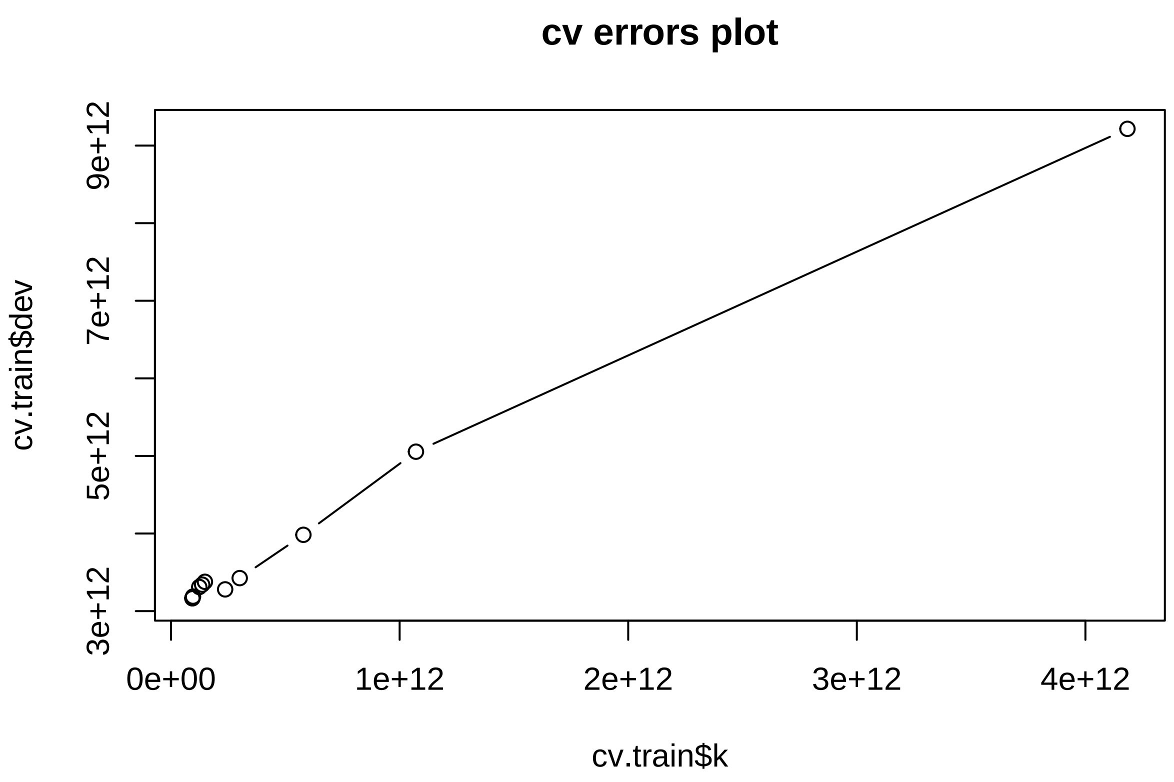
Looking at CV values

```r
cv.train
```

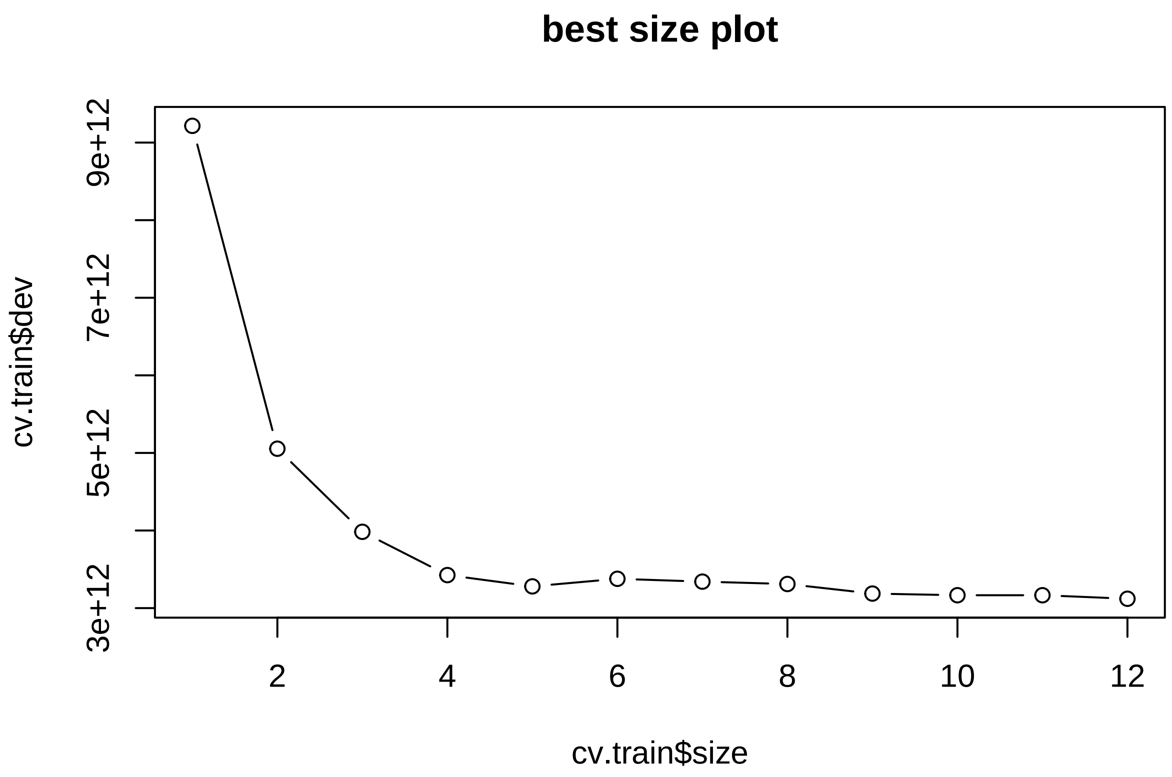
```
## $size
##  [1] 12 11 10  9  8  7  6  5  4  3  2  1
##
## $dev
##  [1] 3.120752e+12 3.166104e+12 3.166104e+12 3.187404e+12 3.310683e+12
##  [6] 3.341139e+12 3.377725e+12 3.280254e+12 3.425571e+12 3.983335e+12
## [11] 5.054478e+12 9.215596e+12
##
## $k
##  [1]         -Inf 9.324800e+10 9.391657e+10 9.625920e+10 1.235916e+11
##  [6] 1.362664e+11 1.482161e+11 2.368001e+11 3.004554e+11 5.790427e+11
## [11] 1.071461e+12 4.183856e+12
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Plotting CV size

```
plot(cv.train$k, cv.train$dev, type = "b", main = "cv errors plot")
```



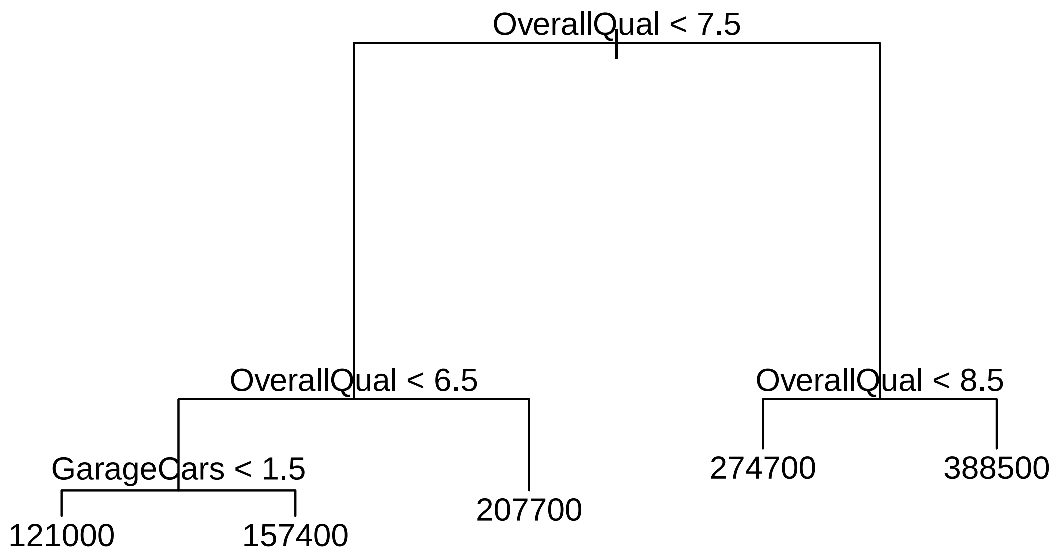
```
plot(cv.train$size, cv.train$dev, type = "b", main = "best size plot")
```



Here it is verified that 12 is the best size for the tree, visually.

Pruning the Tree

```
pruned.tree <- prune.tree(train.tree, best = 5)
plot(pruned.tree)
text(pruned.tree, pretty = 1)
```



```r
pruned.tree
```

```
## 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 *
##     5) OverallQual > 6.5 319 6.288e+11 207700 *
##   3) OverallQual > 7.5 229 2.037e+12 305000
##     6) OverallQual < 8.5 168 6.819e+11 274700 *
##     7) OverallQual > 8.5 61 7.756e+11 388500 *
```

Looking at the differences between the pruned tree and the unpruned tree, there is no difference at all in the chosen node of 7 with the best size of 11. In this case, I am changing the best size down to 5 as a standard, just to make the plotted tree look readable, and to still include node 7.

Pruned Tree Predictions

```r
pruned.pred <- predict(pruned.tree, newdata = test)
head(pruned.pred)
```

```
##        1        2        3        4        5        6 
## 121039.8 121039.8 157414.8 157414.8 274735.5 157414.8
```

Since the pruned and unpruned tree are the same, I am seeing the same predictions as well.

Pruned Tree Errors

```
pruned.error <- mean((pruned.pred-test$SalePrice)^2)
```

Random Forest

```
set.seed(1)
sqrt(ncol(train) - 1) # 4.79... ~ 5
```

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

```
rf.train <- randomForest(SalePrice ~ ., data = train, mtry = 5, importance = T, ntree = 1000)
rf.train
```

```
## 
## Call:
##  randomForest(formula = SalePrice ~ ., data = train, mtry = 5,      importance = T, ntree = 1000) 
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 5
## 
##           Mean of squared residuals: 866190267
##                     % Var explained: 86.27
```

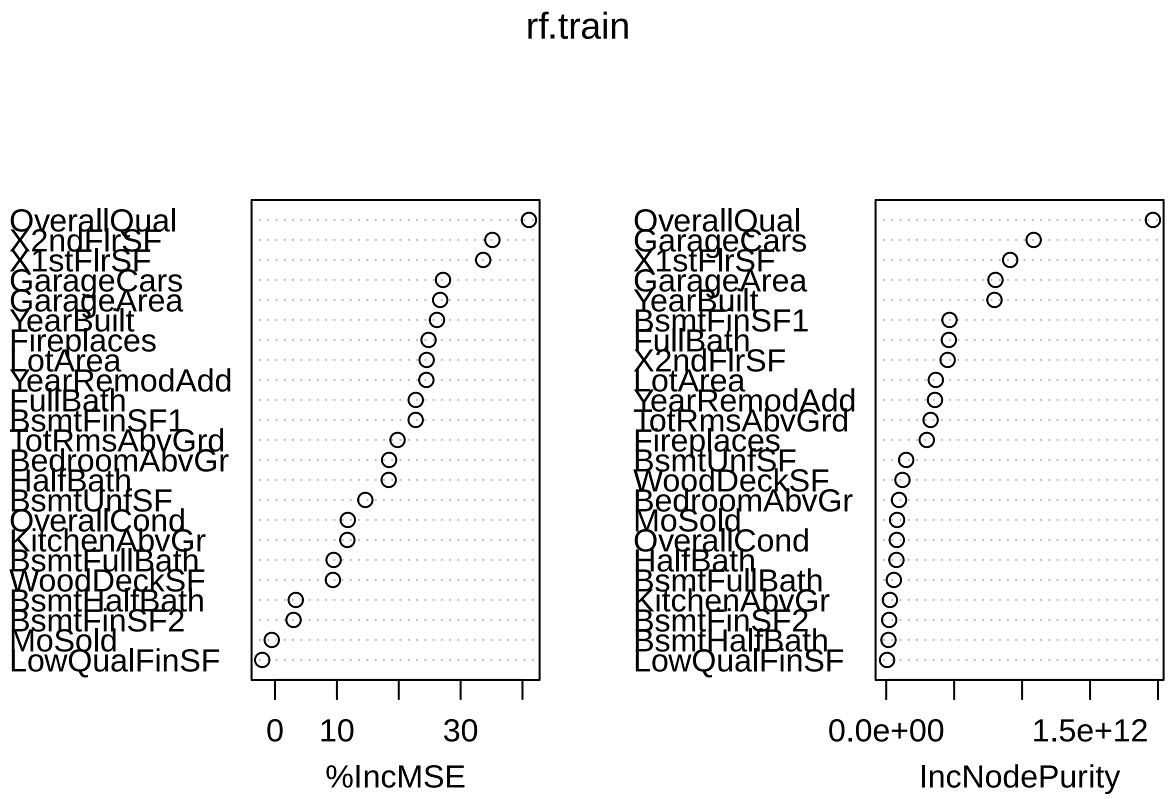
Here I found the value of mtry to be 5, then fit a randomForest model with mtry of 5, and number of trees to be 1000.

Checking Importance

```
importance(rf.train)
```

```
##                 %IncMSE IncNodePurity
## LotArea      24.5068740  3.647543e+11
## OverallQual  41.0418662  1.962346e+12
## OverallCond  11.7642790  7.750099e+10
## YearBuilt    26.1742824  7.965272e+11
## YearRemodAdd 24.4661176  3.587975e+11
## BsmtFinSF1   22.7372525  4.653549e+11
## BsmtFinSF2    2.9954919  2.170142e+10
## BsmtUnfSF    14.6001203  1.468819e+11
## X1stFlrSF    33.6476588  9.120381e+11
## X2ndFlrSF    35.1343935  4.519929e+11
## LowQualFinSF -2.0650265  6.218316e+09
## BsmtFullBath  9.4690048  5.609880e+10
## BsmtHalfBath  3.3585672  1.665339e+10
## FullBath     22.7455080  4.607885e+11
## HalfBath     18.3792520  7.526331e+10
## BedroomAbvGr 18.4402959  9.396537e+10
## KitchenAbvGr 11.6929644  2.720804e+10
## TotRmsAbvGrd 19.8063715  3.265131e+11
## Fireplaces   24.8177548  2.983067e+11
## GarageCars   27.1588484  1.084640e+12
## GarageArea   26.6968485  8.037361e+11
## WoodDeckSF    9.3609823  1.193962e+11
## MoSold       -0.5252677  7.971486e+10
```

```
varImpPlot(rf.train)
```



From the plot, it looks like OverallQual, X2ndFlrSF and X1stFlrSF are the most important when it comes to MSE. MoSold and LowQualFinSF have almost no importance.

Random Forest Predictions

```
rf.pred <- predict(rf.train, newdata = test)
head(rf.pred)
```

```
##        1        2        3        4        5        6
## 129205.6 152902.0 181602.9 186100.2 192022.3 190525.9
```

Random Forest Errors

```
rf.error <- mean((rf.pred - test$SalePrice)^2)
```

Boosting

Boosting CV

```
#First, I need to find the best number of trees. 10-fold CV.
train.boost.cv <- gbm(SalePrice ~ ., data = train,
                      distribution = "gaussian",
                      shrinkage = 0.01,
                      cv.folds = 10) # leave n.tree out, it'll use default
which.min(train.boost.cv$cv.error)
```

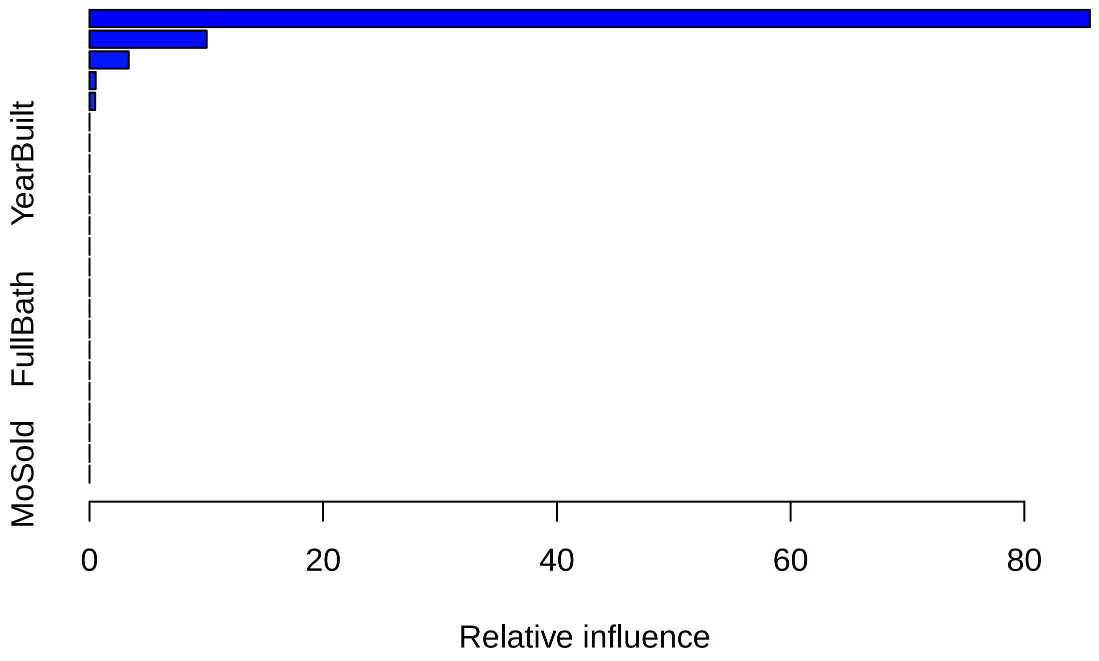
```
## [1] 100
```

Through cross-validation, 100 trees is the best number to iterate through.

```
train.boost <- gbm(SalePrice ~., data = train, distribution = "gaussian", n.trees = 100, shrinkage = 0.
train.boost
```

```
## gbm(formula = SalePrice ~ ., distribution = "gaussian", data = train,
##     n.trees = 100, shrinkage = 0.01)
## A gradient boosted model with gaussian loss function.
## 100 iterations were performed.
## There were 23 predictors of which 5 had non-zero influence.
```

```
summary(train.boost)
```



```
##                       var    rel.inf
## OverallQual   OverallQual 85.6046840
## GarageCars     GarageCars 10.0231177
## X1stFlrSF       X1stFlrSF  3.3527617
## X2ndFlrSF       X2ndFlrSF  0.5282840
## Fireplaces     Fireplaces  0.4911526
## LotArea           LotArea  0.0000000
## OverallCond   OverallCond  0.0000000
## YearBuilt       YearBuilt  0.0000000
## YearRemodAdd YearRemodAdd  0.0000000
## BsmtFinSF1     BsmtFinSF1  0.0000000
## BsmtFinSF2     BsmtFinSF2  0.0000000
## BsmtUnfSF       BsmtUnfSF  0.0000000
## LowQualFinSF LowQualFinSF  0.0000000
## BsmtFullBath BsmtFullBath  0.0000000
## BsmtHalfBath BsmtHalfBath  0.0000000
## FullBath         FullBath  0.0000000
## HalfBath         HalfBath  0.0000000
## BedroomAbvGr BedroomAbvGr  0.0000000
## KitchenAbvGr KitchenAbvGr  0.0000000
## TotRmsAbvGrd TotRmsAbvGrd  0.0000000
## GarageArea     GarageArea  0.0000000
## WoodDeckSF     WoodDeckSF  0.0000000
## MoSold             MoSold  0.0000000
```

23 predictors were used in the boosting model, 5 of them having non-zero influence. The 5 predictors that

some non-zero influence are: OverallQual, GarageCars, X1stFlrSF, FullBath, and YearBuilt. OverallQual has the highest relative influence in the boosting model.

Boosting Predictions

```
boosting.pred <- predict(train.boost, newdata = test, n.trees = 100)
head(boosting.pred)
```

```
## [1] 160400.5 160998.9 160948.9 161547.3 227369.6 161547.3
```

Boosting MSE

```
boosting.error <- mean((boosting.pred - test$SalePrice)^2)
```

Bagging

```
ncol(train) - 1 # number of predictors
```

```
## [1] 23
```

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

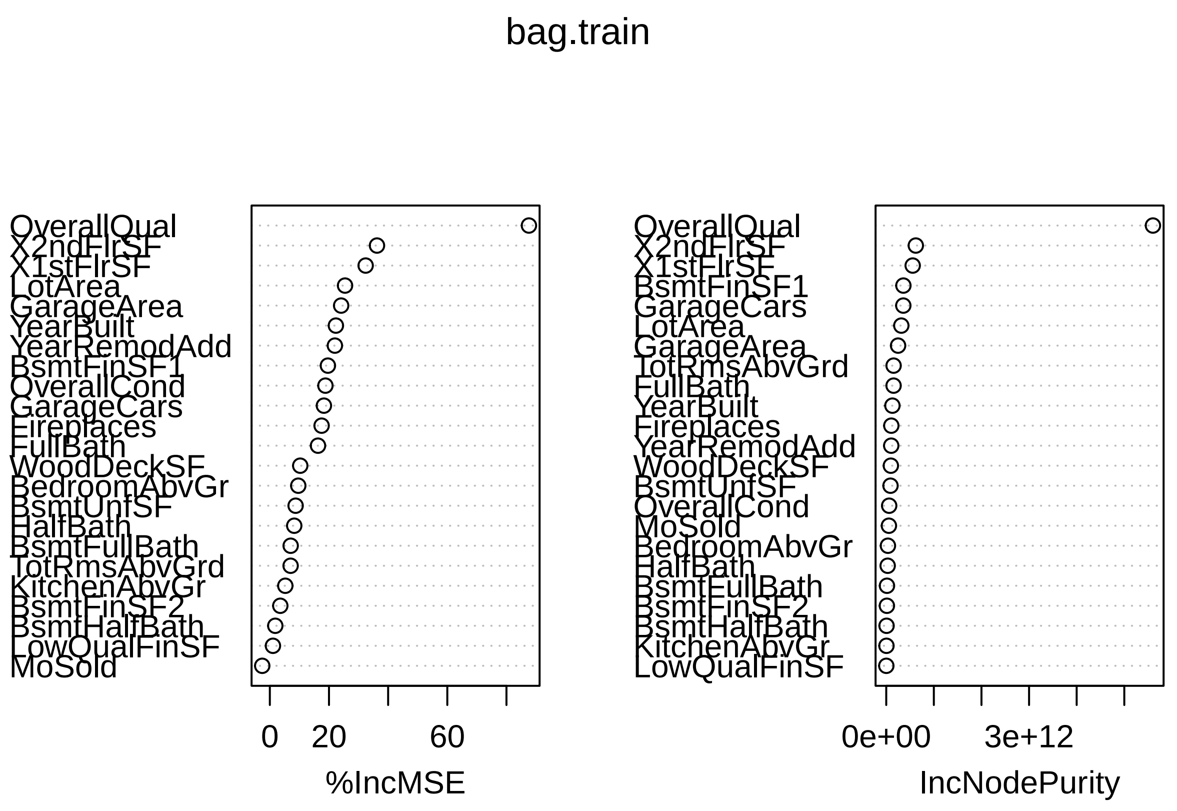
Setting mtry to 23, I determined in the same chunk that the number of predictors should be 23.

Bagging Importance

```
importance(bag.train)
```

```
##                %IncMSE IncNodePurity
## LotArea      25.425392  3.167909e+11
## OverallQual  87.604864  5.601815e+12
## OverallCond  18.789715  6.156961e+10
## YearBuilt    22.330285  1.292153e+11
## YearRemodAdd 21.993253  1.049823e+11
## BsmtFinSF1   19.624942  3.600306e+11
## BsmtFinSF2    3.532667  1.336753e+10
## BsmtUnfSF     8.719354  8.968906e+10
## X1stFlrSF    32.398636  5.558395e+11
## X2ndFlrSF    36.230317  6.211324e+11
## LowQualFinSF  1.046433  1.768555e+09
## BsmtFullBath  7.041587  1.546736e+10
## BsmtHalfBath  1.834591  7.071876e+09
## FullBath     16.277303  1.542048e+11
## HalfBath       8.246419  3.039662e+10
## BedroomAbvGr   9.612041  3.562535e+10
## KitchenAbvGr   5.225612  4.694575e+09
## TotRmsAbvGrd   7.017686  1.544150e+11
## Fireplaces    17.499771  1.063818e+11
## GarageCars    18.274517  3.583694e+11
## GarageArea    24.103794  2.490526e+11
## WoodDeckSF    10.271406  9.684320e+10
## MoSold        -2.574815  5.613963e+10
```

```
varImpPlot(bag.train)
```



Just like before, the most important predictors are the same as the other methods. OverallQual, X1stFlrSF, X2ndFlrSF.

Bagging Predictions

```
bag.pred <- predict(bag.train, newdata = test)
head(bag.pred)
```

```
##        1        2        3        4        5        6
## 129962.7 157217.6 164797.6 179996.8 194523.0 188260.0
```

Predictions line up for housing prices.

Bagging Errors

```
bag.error <- mean((bag.pred - test$SalePrice)^2)
```

All MSE

```
## [1] "KNN MSE:"         "28330849558.0679"
```

```
## [1] "Linear Regression MSE:" "40083608206.0172"
```

```
## [1] "Forward Stepwise MSE:" "34915186258.8137"
```

```
## [1] "Backward Stepwise MSE:" "34915186258.8137"
```

```
## [1] "Best Subset MSE:" "34915186258.8137"
```

```
## [1] "Ridge Regression MSE:" "36199635351.5297"
```

```
## [1] "Lasso Regression MSE:" "36678884418.507"
```

```
## [1] "Generalized Additive Model MSE:" "36987402565.3387"
```

```
## [1] "Decision Tree MSE:" "37959706612.8671"
```

```
## [1] "Pruned Tree MSE:" "37861815318.0981"
```

```
## [1] "Random Forest MSE:" "36951305829.6866"
```

```
## [1] "Boosting MSE:"    "33957213364.6433"
```

```
## [1] "Bagging MSE:"     "37031868843.6184"
```

KNN has the lowest MSE, while Linear Regression has the highest. My question is, will these play out the same way using the Kaggle score?

Which method do I think will perform the best?

I think that the Linear models will perform the best. My reason is because this model was the easiest to tamper with for me to get values that were desirable. If not for this reason, I think KNN would perform the best because it has the lowest MSE of all methods.

Writing CSVs

```r
Id <- seq(1461,2919)
knn.df <- data.frame(Id, knn.pred)
colnames(knn.df) <- c('Id', 'SalePrice')
write.csv(knn.df, file='TestPredictions_KNN.csv', row.names = FALSE)

# Kaggle SCORE: 0.20829
```

```r
linear.df <- data.frame(Id, linear.pred)
colnames(linear.df) <- c('Id', 'SalePrice')
write.csv(linear.df, file='TestPredictions_LR.csv', row.names = FALSE)

# Kaggle SCORE: 0.14867
```

```r
forward.df <- data.frame(Id, fwd.predictions)
colnames(forward.df) <- c('Id', 'SalePrice')
write.csv(forward.df, file='TestPredictions_Forward.csv', row.names = FALSE)

# Kaggle Score: 0.22896
```

```r
backward.df <- data.frame(Id, bwd.predictions)
colnames(backward.df) <- c('Id', 'SalePrice')
write.csv(backward.df, file='TestPredictions_Backward.csv', row.names = FALSE)

# Kaggle Score: 0.22896
```

```r
subset.df <- data.frame(Id, subset.predictions)
colnames(subset.df) <- c('Id', 'SalePrice')
write.csv(subset.df, file='TestPredictions_Subset.csv', row.names = FALSE)

# Kaggle SCORE: 0.22896
```

```r
lasso.df <- data.frame(Id, lasso.pred)
colnames(lasso.df) <- c('Id', 'SalePrice')
write.csv(lasso.df, file='TestPredictions_Lasso.csv', row.names = FALSE)

# Kaggle SCORE: 0.15165
```

```r
ridge.df <- data.frame(Id, ridge.pred)
colnames(ridge.df) <- c('Id', 'SalePrice')
write.csv(ridge.df, file='TestPredictions_Ridge.csv', row.names = FALSE)

# Kaggle SCORE: 0.15508
```

```r
gam.df <- data.frame(Id, gam.pred)
colnames(gam.df) <- c('Id', 'SalePrice')
write.csv(gam.df, file='TestPredictions_GAM.csv', row.names = FALSE)

# Kaggle Score: 0.13368
```

```r
dt.df <- data.frame(Id, tree.pred)
colnames(dt.df) <- c('Id', 'SalePrice')
write.csv(dt.df, file='TestPredictions_DT.csv', row.names = FALSE)

# Kaggle Score: 0.24341
```

```r
pruned.df <- data.frame(Id, pruned.pred)
colnames(pruned.df) <- c('Id', 'SalePrice')
write.csv(pruned.df, file='TestPredictions_Pruned.csv', row.names = FALSE)

# Kaggle Score: 0.26001
```

```r
rf.df <- data.frame(Id, rf.pred)
colnames(rf.df) <- c('Id', 'SalePrice')
write.csv(rf.df, file='TestPredictions_RF.csv', row.names = FALSE)

# Kaggle Score: 0.1528
```

```r
boosting.df <- data.frame(Id, boosting.pred)
colnames(boosting.df) <- c('Id', 'SalePrice')
write.csv(boosting.df, file='TestPredictions_Boosting.csv', row.names = FALSE)

# Kaggle Score: 0.32367
```

```r
bag.df <- data.frame(Id, bag.pred)
colnames(bag.df) <- c('Id', 'SalePrice')
write.csv(bag.df, file='TestPredictions_Bagging.csv', row.names = FALSE)

# Kaggle Score: 0.15406
```

Discussion on why some methods performed better or worst

The first method I want to mention is the Generalized Additive Model. As my best model, I was not expecting this model to perform the way it did. Though, I think this could attributed to carefully looking at the data and choosing variables to put a spline on based on if it is linear or not, and how easy it was to tell if I was overfitting or not.

Next, I want to talk about how the decision tree performed better than the pruned tree. Initially, I was expecting the pruned tree to perform better just because it is a more specified version of the decision tree, but I think this ultimately led to its downfall.

As for the worst method, I am not too surprised by it being the boosting model. I found the number of predictors to try to be 23, and the model suggested that 100 trees was optimal still. I feel like this is the reason the model did the worst, but even so, I tried 1000 trees and the model actually performed worse than with 100 trees. Potentially, I could change the number of predictors used, instead of the optimal amount.

Conclusion / Summary

Overall, I am happy that I was able to get at least model under a score of 0.14. Using a variety of methods on this data set showed me how different outcomes can be, and the importance of being able to test out many different methods, instead of just going with one I am comfortable with. Though, every model seemed to give good predictions when it came to housing prices, even the boosting model and that performed the worst. I think I can say overall that the models made in this project can somewhat accurately depict housing prices in Iowa.

Discussion for the future

In the future, I think that there are a number of different things I could look into for this data specifically. With more time, I could investigate the variables a bit more indepth, and try to apply the investigations to the models built. I could look into things like does having more full baths over half baths affect things like house price, even if the number of bedrooms are not ideal.