Kaggle Competition - Iowa Housing Prices

Maxwell Waun

Introduction to the data

The data used in the following report is from the Housing Dataset from Kaggle competitions. The goal of this project is to create different models in order to most accurately predict the sale prices of the houses given. Nine models will be built in order to accomplish this These methods include: K-Nearest Neighbor, Linear Regression, Backwards Step-Wise Selection, Shrinkage Methods, Regression Trees, Bagging, Random Forest, Boosting and Generalized Additive Methods.

Explaining the Raw Data

The raw data consists of 81 columns in the training data, and 80 columns in the test data. The missing column from the test data is the `SalePrice` column. This is because the test values are hidden, only to be used to test model accuracy once submitted back into the Kaggle competition. Of these 81 columns in the training data, there are 1460 observations for each. For the test data, there are 1459 observations that correspond to it's 80 columns.

How the data was cleaned

I used the pre-processed data provided by the professor this project was assigned by. Therefore, I had nothing to do with cleaning the data.

Importing Libraries

```
library(readr)
library(class)
library(FNN)
library(MASS)
library(faraway)
library(glmnet)
library(leaps)
library(tree)
library(randomForest)
library(gam)
library(caret)
library(gbm)
```

Importing the Data

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

set.seed(123)
```

Create necessary functions

```
rmse <- function(actual, predicted)
  sqrt(mean(actual-predicted)^2)

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

---

1 KNN Method

Scale the Data

We do this because distance based algorithms are affected by the scale of the data.

```
train.scale <- as.data.frame(scale(train[,1:23]))
test.scale <- as.data.frame(scale(test[,1:23]))
train.scale$SalePrice <- train$SalePrice

SalePrice <- train[,24]
```

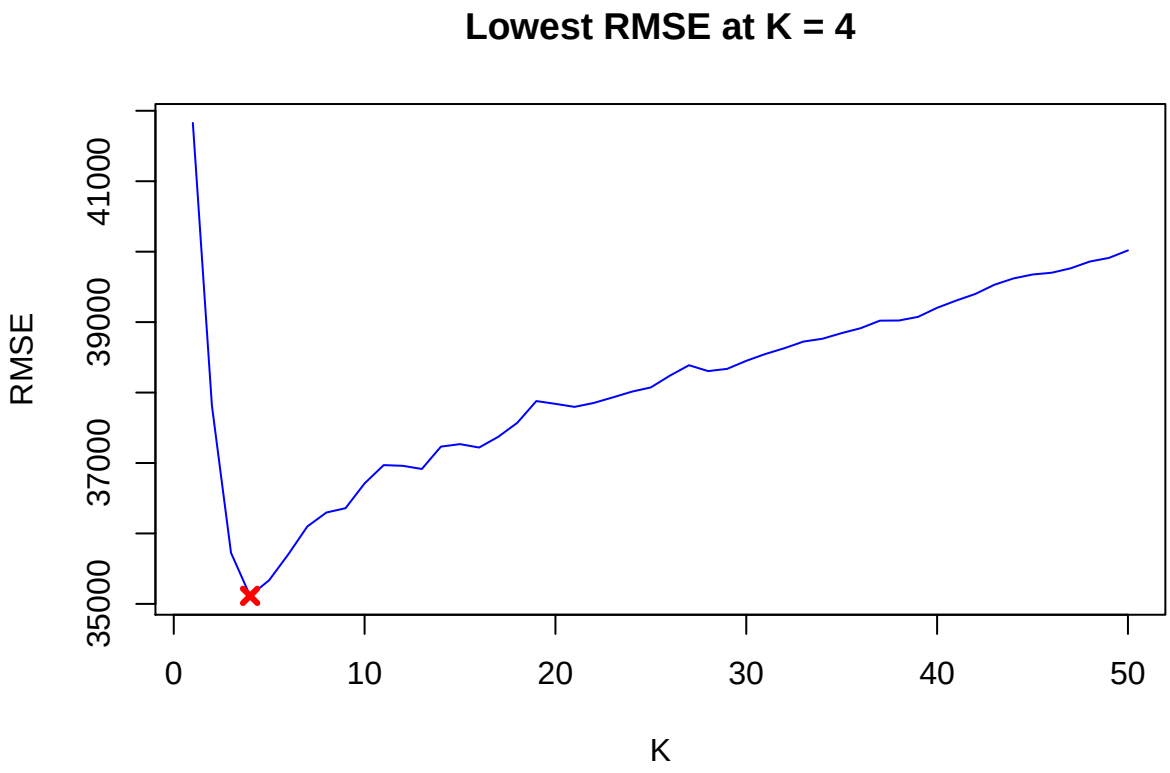
Perform 10 fold cross validation on training set to obtain lowest RMSE and it's corresponding K value in order to tune the model.

```
ctrl <- trainControl(method = "cv", number = 10)
knn.cv <- train(SalePrice ~.,
                    data = train.scale,
                    trControl = ctrl,
                    tuneGrid   = expand.grid(k = 1:50),
                    method = "knn")

knn.rmse <- round(knn.cv$results$RMSE[which.min(knn.cv$results$RMSE)])
best.k <- knn.cv$results$k[which.min(knn.cv$results$RMSE)]

plot(1:50, knn.cv$results$RMSE,
     col = "blue",
```

```
     type = 'l',
     main = paste0("Lowest RMSE at K = ",best.k),
     ylab = "RMSE",
     xlab= "K")
points(best.k,
       knn.rmse,
       pch = 4,
       col = "red",
       lwd = 3)
```




```
train.scale <- train.scale[,1:23]
```

The RMSE obtained by using the *KNN* method on the training set is 35114 at K = 4.

I chose the tuning parameter `K` by obtaining the root mean squared error for KNN predictions for K = [1, . . . , 200] on the training set. As you can see, the larger K becomes, the larger the RMSE becomes after K = 4. Therefore, we will choose K to be equal to 4, which corresponds to the lowest RMSE

Predicting with KNN when K = 4

```
knn.pred <- knn.reg(train.scale, test = test.scale, y = train$SalePrice, k = best.k)
```

2 Linear Regression

Perform 10 fold cross validation on training set to obtain RMSE

```
ctrl <- trainControl(method = "cv", number = 10)
lm.cv <- train(SalePrice ~.,
                       data = train,
                       trControl = ctrl,
                       method = "lm")

lm.rmse <- round(lm.cv$results$RMSE)
```

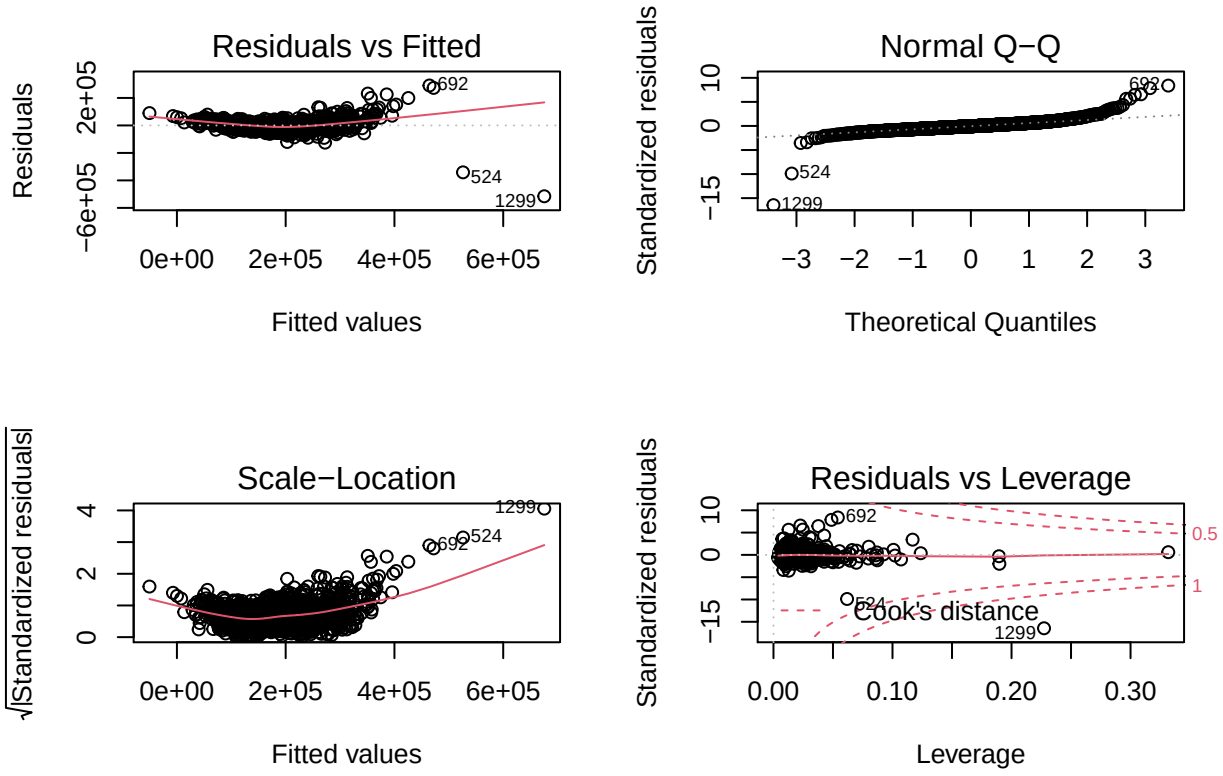
The RMSE obtained by using the *Linear Regression* method on the training set is 36551

Fit the linear regression model

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

Residuals Plot

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



It appears that the normal Q-Q plot is slightly non linear as well as the scale-location graph. The normal Q-Q plot can be fixed by finding the largest leverage points and removing them. The non-linear relationship

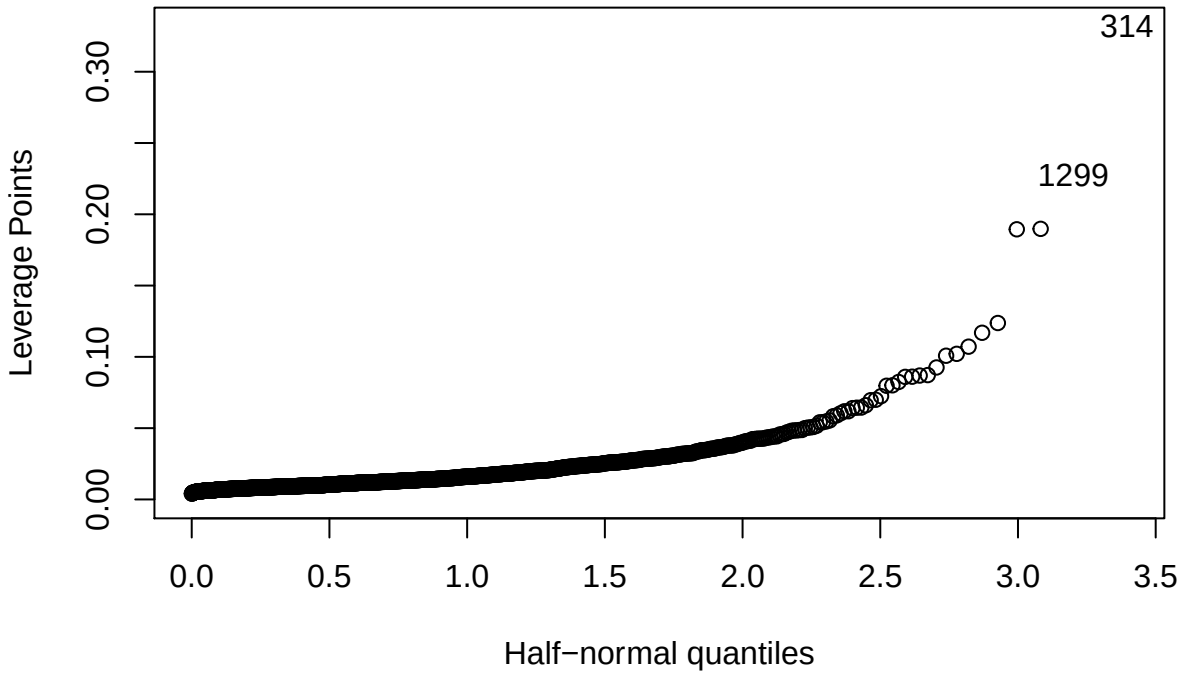
in the Scale-Location plot suggests heteroskedasticity. To remedy the normality assumption, the largest leverage point is to be removed.

Remedies if the model assumptions are violated

- Removing the largest leverage points is a remedy in case model assumptions are violated

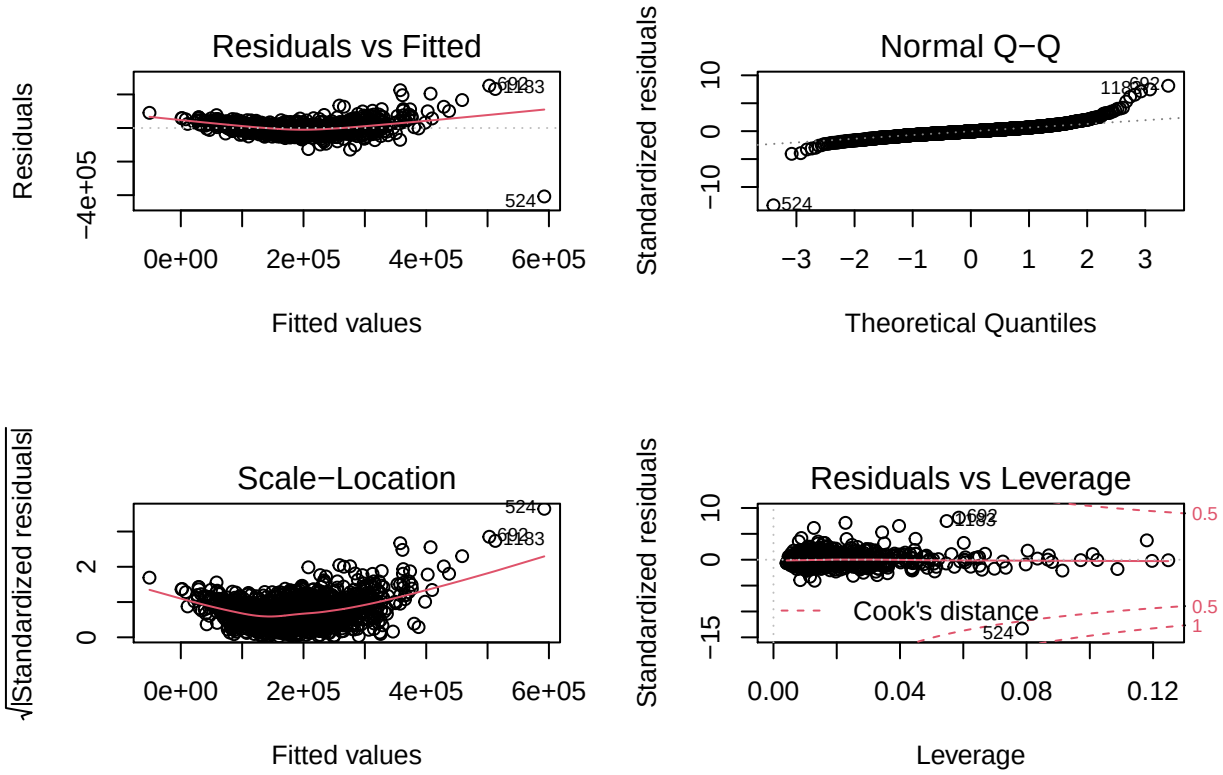
Plot halfnorm to find highest leverage points

```
halfnorm(influence(lm.fit)$hat, labs = row.names(train), ylab = "Leverage Points")
```



Remove top leverage points and plot residuals

```
lm.train <- train[-c(250,314,336,707,1299),]
lm.fit2 <- lm(SalePrice~.-SalePrice, data=lm.train)
par(mfrow=c(2,2))
plot(lm.fit2)
```



The normal Q-Q plot looks more linear than it did before removing the large leverage points

Significant Coefficients

```
summary(lm.fit2)
```

```
##
## Call:
## lm(formula = SalePrice ~ . - SalePrice, data = lm.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -407676  -15184   -1558   13230  252132
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.098e+06  1.199e+05  -9.160  < 2e-16 ***
## LotArea       1.089e+00  1.783e-01   6.110 1.28e-09 ***
## OverallQual   1.553e+04  1.086e+03  14.296  < 2e-16 ***
## OverallCond   4.896e+03  9.309e+02   5.259 1.67e-07 ***
## YearBuilt     3.564e+02  5.251e+01   6.786 1.68e-11 ***
## YearRemodAdd  1.659e+02  6.037e+01   2.748 0.006076 **
## BsmtFinSF1    4.598e+01  4.422e+00  10.399  < 2e-16 ***
## BsmtFinSF2    2.177e+01  6.499e+00   3.350 0.000829 ***
## BsmtUnfSF     2.505e+01  3.881e+00   6.454 1.49e-10 ***
## X1stFlrSF     5.676e+01  5.309e+00  10.691  < 2e-16 ***
```

```
## X2ndFlrSF     5.719e+01  4.458e+00  12.830  < 2e-16 ***
## LowQualFinSF  1.512e+01  1.799e+01   0.841 0.400711
## BsmtFullBath  3.785e+02  2.409e+03   0.157 0.875170
## BsmtHalfBath -2.746e+03  3.764e+03  -0.729 0.465875
## FullBath     -1.429e+03  2.586e+03  -0.553 0.580666
## HalfBath     -1.915e+03  2.433e+03  -0.787 0.431341
## BedroomAbvGr -1.071e+04  1.544e+03  -6.938 6.00e-12 ***
## KitchenAbvGr -2.214e+04  4.456e+03  -4.968 7.56e-07 ***
## TotRmsAbvGrd  4.917e+03  1.128e+03   4.359 1.40e-05 ***
## Fireplaces    3.128e+03  1.614e+03   1.938 0.052824 .
## GarageCars    2.619e+03  2.653e+03   0.987 0.323792
## GarageArea    2.365e+01  8.954e+00   2.642 0.008344 **
## WoodDeckSF    1.289e+01  7.313e+00   1.762 0.078208 .
## MoSold       -1.887e+02  3.126e+02  -0.604 0.546225
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 31980 on 1431 degrees of freedom
## Multiple R-squared:   0.84,  Adjusted R-squared:  0.8374
## F-statistic: 326.6 on 23 and 1431 DF,  p-value: < 2.2e-16
```

The significant statistics according to our model summary are:

- **- LotArea**:*Lot size in square feet*
- **- OverallQual**:*Rates the overall material and finish of the house*
- **- OverallCond**:*Rates the overall condition of the house*
- **- YearBuilt**:*Original construction date*
- **- YearRemodAdd**:*Remodel date (same as construction date if no remodeling or additions)*
- **- BsmtFinSF1**:*finished square feet*
- **- BsmtFinSF2**:*Type 2 finished square feet*
- **- BsmtUnfSF**:*Unfinished square feet of basement area*
- **- X1stFlrSF**:*First Floor square feet*
- **- X2ndFlrSF**:*Second floor square feet*
- **- BedroomAbvGr**:*Bedrooms above grade*
- **- KitchenAbvGr**:*Kitchen above grade*
- **- TotRmsAbvGrd**:*Total rooms above grade (does not include bathrooms)*
- **- GarageArea**:*Size of garage in square feet*

Predicting with Linear Regression

```
lm.pred <- predict(lm.fit, test, type="response")
```

---

3 Subset Selection

At first glance, I'm worried that best subset is the *worst* method to use in this scenario, as going through each subset of the 23 predictors will be computationally exhaustive (8,388,608 models). Because of this, I'm left with choosing between forward and backward step-wise selection.

Forward step-wise is a good choice for when our number of predictors is larger than our number of observations (p > n). Additionally, it's good to start off with all the predictors in the case of there being collinearity, and therefore using backwards step-wise selection will ensure we at least start with those variables, unlike forward step-wise. With that said, I decided to go with *backwards step-wise selection*

Perform 10 fold cross validation on training set to obtain RMSE

```
cv.errors <- matrix(data = 0, nrow = 10, ncol = 23)
folds=sample(rep(1:10, length=nrow(train)))
for(j in 1:10){
  # Fit the model with each subset of predictors on the training part of the fold
  best.fit=regsubsets(SalePrice~.,data=train[folds!=j,], method = "backward", nvmax=23)
  # For each subset
  for(i in 1:23){
    # Predict on the hold out part of the fold for that subset
    pred=predict(best.fit, train[folds==j,],id=i)
    # Get the mean squared error for the model trained on the fold with the subset
    cv.errors[j,i] = mean((train$SalePrice[folds==j]-pred)^2)
  }
}
cv.errors <- as.data.frame(cv.errors)
cv.errors <- apply(cv.errors,2,mean)
cv.errors <- sqrt(cv.errors)
cv.errors <- as.numeric(as.character(cv.errors))
back.rmse <- round(cv.errors[which.min(cv.errors)])
```

Backwards step-wise regression on the training data obtains an RMSE of 37805

I chose nvmax = 23 since there could be collinearity present that would possibly be missed if we didn't start with all the predictors

```
back.fit <- regsubsets(SalePrice ~ ., nvmax = 23, data = train, method = "backward")
back.summary <- summary(back.fit)
```

Compute Cp, BIC, RSS and Adjusted R^2

```
cp.min <- which.min(back.summary$cp)
bic.min <- which.min(back.summary$bic)
adjr2.max <- which.max(back.summary$adjr2)
rss.min <- which.min(back.summary$rss)
```

Plot Accuracy Metrics

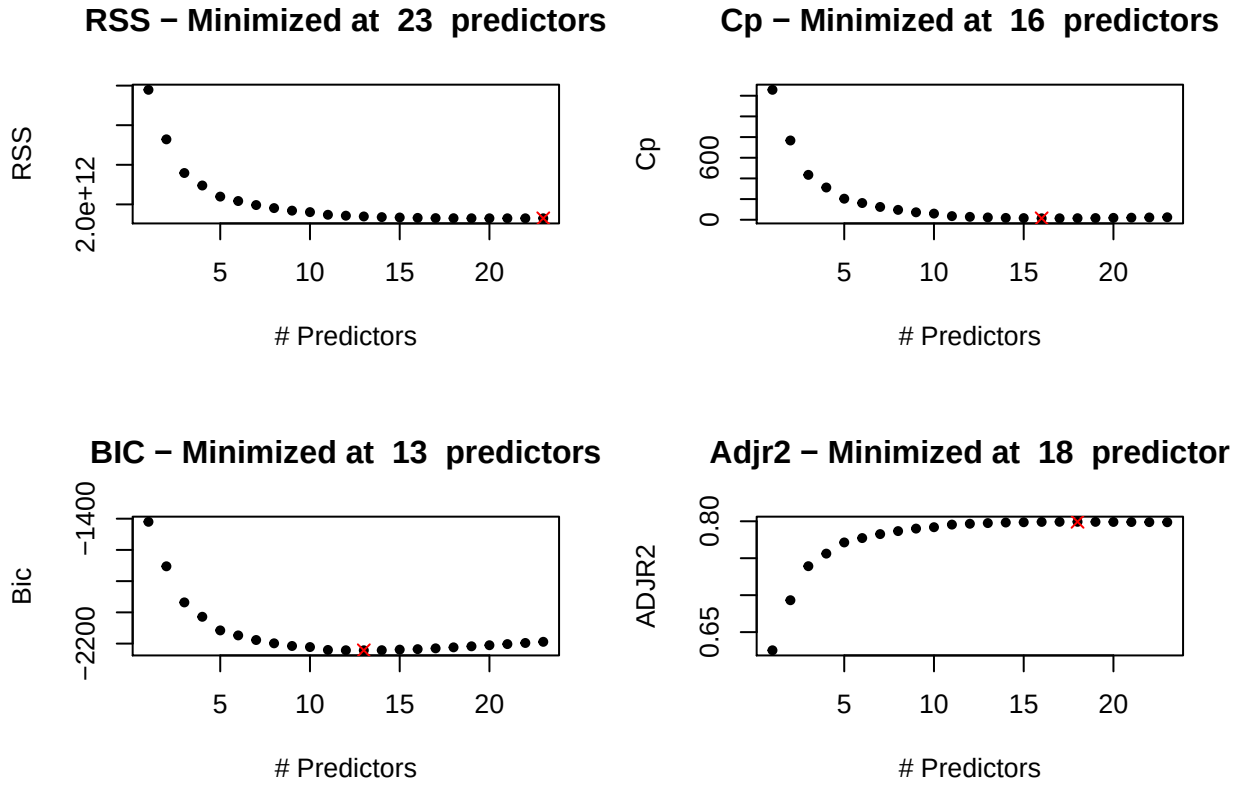
```
par(mfrow=c(2,2))

plot(back.summary$rss,
     main = paste("RSS - Minimized at ",rss.min," predictors"),
     xlab = "# Predictors",
     ylab = "RSS", pch = 20)
points(rss.min,
       back.summary$rss[rss.min],
       col = "red",
       pch = 4,
       lwd= 1)

plot(back.summary$cp,
     main = paste("Cp - Minimized at ",cp.min," predictors"),
     xlab = "# Predictors",
     ylab = "Cp",
     pch = 20)
points(cp.min,
       back.summary$cp[cp.min],
       col = "red",
       pch = 4,
       lwd= 1)

plot(back.summary$bic,
     main = paste("BIC - Minimized at ",bic.min," predictors"),
     xlab = "# Predictors",
     ylab = "Bic", pch = 20)
points(bic.min,
       back.summary$bic[bic.min],
       col = "red",
       pch = 4,
       lwd= 1)

plot(back.summary$adjr2,
     main = paste("Adjr2 - Minimized at ",adjr2.max," predictor"),
     xlab = "# Predictors",
     ylab = "ADJR2",
     pch = 20)
points(adjr2.max,
       back.summary$adjr2[adjr2.max],
       col = "red",
       pch = 4,
       lwd= 1)
```



I will choose Adjusted R^2 to determine which model to use in backwards step-wise selection. The adjusted R^2 value gives the proportion of variation in the `SalePrice` that is explained by the variation in the features.

```
print(coefficients(back.fit, id = adjr2.max))
```

```
##   (Intercept)       LotArea   OverallQual   OverallCond     YearBuilt
## -9.233942e+05  4.442802e-01  1.735440e+04  4.725902e+03  3.018044e+02
##  YearRemodAdd    BsmtFinSF1    BsmtFinSF2     BsmtUnfSF     X1stFlrSF
##  1.304796e+02  2.361949e+01  1.031296e+01  1.284729e+01  5.185799e+01
##    X2ndFlrSF  BsmtFullBath      FullBath  BedroomAbvGr  KitchenAbvGr
##  4.259458e+01  6.888193e+03  2.656044e+03 -9.020249e+03 -2.565867e+04
##  TotRmsAbvGrd    Fireplaces    GarageCars    WoodDeckSF
##  6.101927e+03  3.946090e+03  1.223549e+04  2.139239e+01
```

The coefficients used in the model selected by Adjusted R^2 include

- **1 LotArea**:*Lot size in square feet*
- **2 OverallQual**:*Rates the overall material and finish of the house*
- **3 OverallCond**:*Rates the overall condition of the house*
- **4 YearBuilt**:*Original construction date*

- **5 BsmtFinSF1**:*finished square feet*
- **6 X1stFlrSF**:*First Floor square feet*
- **7 X2ndFlrSF**:*Second floor square feet*
- **8 BsmtFullBath**:*Basement full bathrooms*
- **9 FullBath**:*Full bathrooms*
- **10 BedroomAbvGr**:*Bedrooms above grade*
- **11 KitchenAbvGr**:*Kitchen above grade*
- **12 TotRmsAbvGrd**:*Total rooms above grade (does not include bathrooms)*
- **13 GarageCars**:*Size of garage in square feet*
- **14 WoodDeckSF**:*Size of wood deck in square feet*

Predicting with backwards step-wise

```
back.pred <- predict.regsubsets(back.fit, test, id = adjr2.max)
```

---

4 Shrinkage

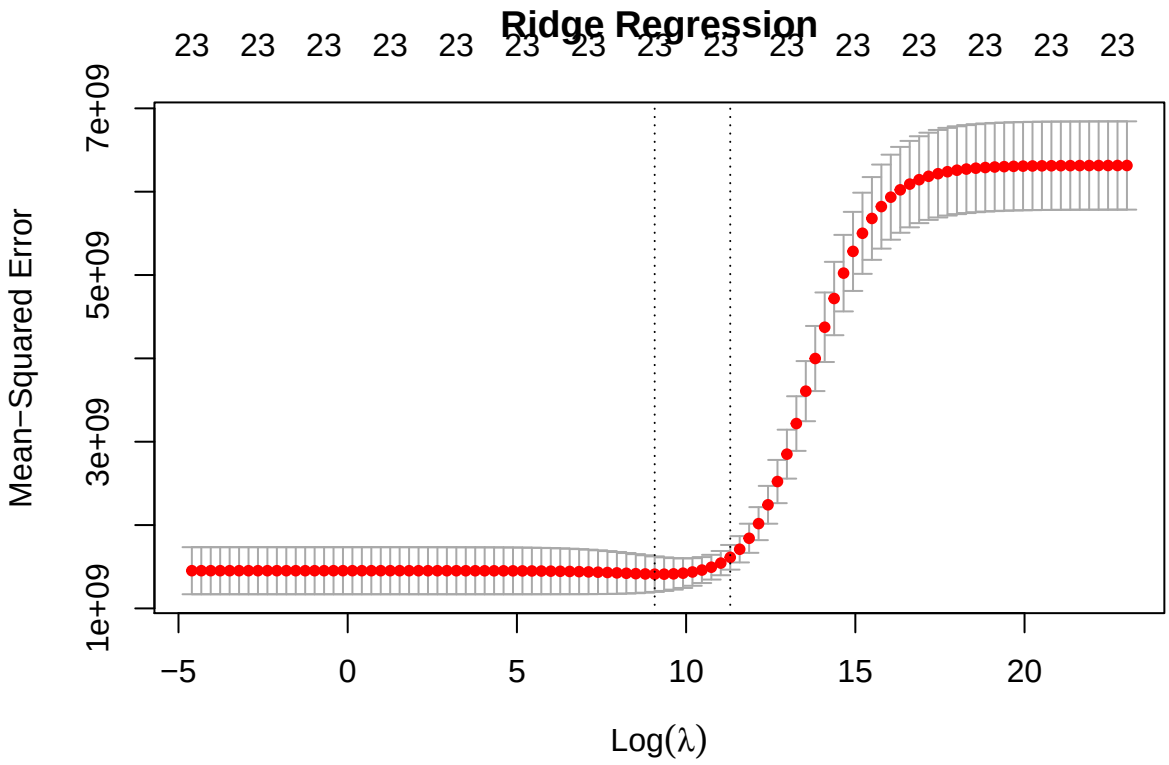
Initialize lambda grid and train/test matrices using scaled data

```
grid <- 10^seq(10, -2, length = 100)
train.matrix <- model.matrix(SalePrice ~ ., data = train.scale)[,-1]
test.matrix <- model.matrix(rep(0, nrow(test.scale)) ~., data = test.scale)[,-1]
```

Ridge Regression

Using cross-validation to find the best lambda

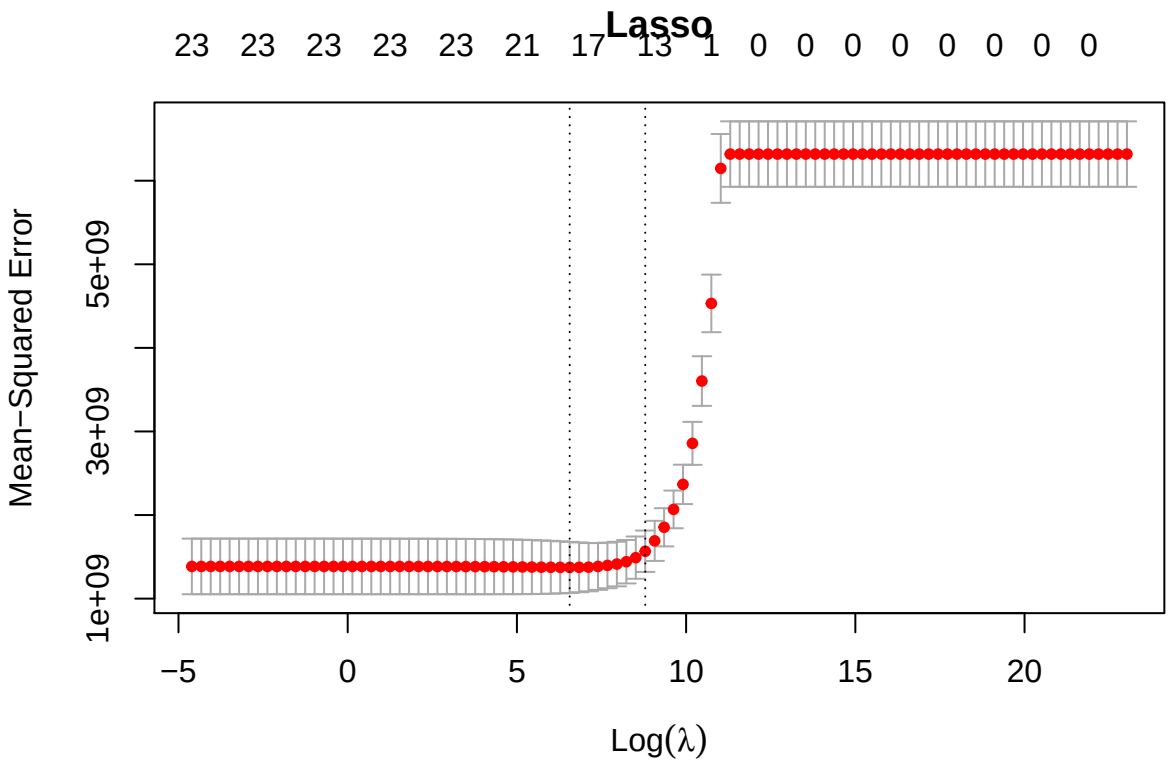
```
ridge.fit <- cv.glmnet(train.matrix, SalePrice, alpha = 0, lambda = grid, thresh = 1e-12)
ridge.rmse <- round(sqrt(min(ridge.fit$cvm)))
plot(ridge.fit, main = "Ridge Regression")
```



Lasso

Using cross-validation to find the best lambda

```
lasso.fit <- cv.glmnet(train.matrix, SalePrice, alpha = 1, lambda = grid, thresh = 1e-12)
lasso.rmse <- round(sqrt(min(lasso.fit$cvm)))
plot(lasso.fit, main = "Lasso")
```



Ridge regression on the split training data obtains an RMSE of 37562 Lasso regression on the split training data obtains an RMSE of 37062

When lambda is very large, the penalty will increase as well which will bring the coefficients close to zero, but not zero. When is smaller, the coefficients aren't shrunk as much.

When talking about lasso, the penalty will instead shrink the coefficients all the way towards zero, potentially leaving us with less predictors than we started with. This is advantageous in this scenario as we have more predictors than most likely needed. Therefore I believe that lasso can do a better job in terms of model interpretation.

Coefficients

Ridge Regression

```
predict(ridge.fit, s = ridge.fit$lambda.min, type = "coefficients")
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  180921.1959
## LotArea        4330.4541
## OverallQual   21872.7268
## OverallCond    3994.7000
## YearBuilt      6878.2056
## YearRemodAdd   4515.1292
```

```
## BsmtFinSF1    10034.6846
## BsmtFinSF2     1307.0287
## BsmtUnfSF      5177.7146
## X1stFlrSF     15306.5170
## X2ndFlrSF     12193.3625
## LowQualFinSF    507.5363
## BsmtFullBath   3823.8258
## BsmtHalfBath    225.2931
## FullBath       4274.8106
## HalfBath       2013.7941
## BedroomAbvGr  -4896.0069
## KitchenAbvGr  -5388.0735
## TotRmsAbvGrd  10182.2105
## Fireplaces     4426.9463
## GarageCars     6775.3516
## GarageArea     4458.0835
## WoodDeckSF     3102.9866
## MoSold          234.5500
```

Based on the coefficients, it appears that `OverallQual` and `X1stFlrSF` are the largest coefficients. Note the predictors for ridge regression and lasso are different.

Lasso

```
predict(lasso.fit, s = lasso.fit$lambda.min, type = "coefficients")
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  180921.1959
## LotArea        4180.7542
## OverallQual   26236.9474
## OverallCond    3580.9725
## YearBuilt      7837.1987
## YearRemodAdd   3515.7753
## BsmtFinSF1     7417.3673
## BsmtFinSF2        .
## BsmtUnfSF      1539.3021
## X1stFlrSF     21249.2755
## X2ndFlrSF     16620.5785
## LowQualFinSF      .
## BsmtFullBath   2940.4198
## BsmtHalfBath      .
## FullBath        849.4802
## HalfBath          .
## BedroomAbvGr  -4524.4934
## KitchenAbvGr  -4945.2272
## TotRmsAbvGrd   8079.6523
## Fireplaces     2687.9509
## GarageCars     7766.8598
## GarageArea     1823.9780
## WoodDeckSF     2608.7090
## MoSold            .
```

Based on the coefficients, it appears that `OverallQual`, `X1stFlrSF`, and `X2stFlrSF` are the largest

coefficients, and therefore are going to determine the majority of the outcome.

I tuned **s** by finding the minimum lambda on the fit for both lasso and ridge regression

Predicting with Ridge and Lasso

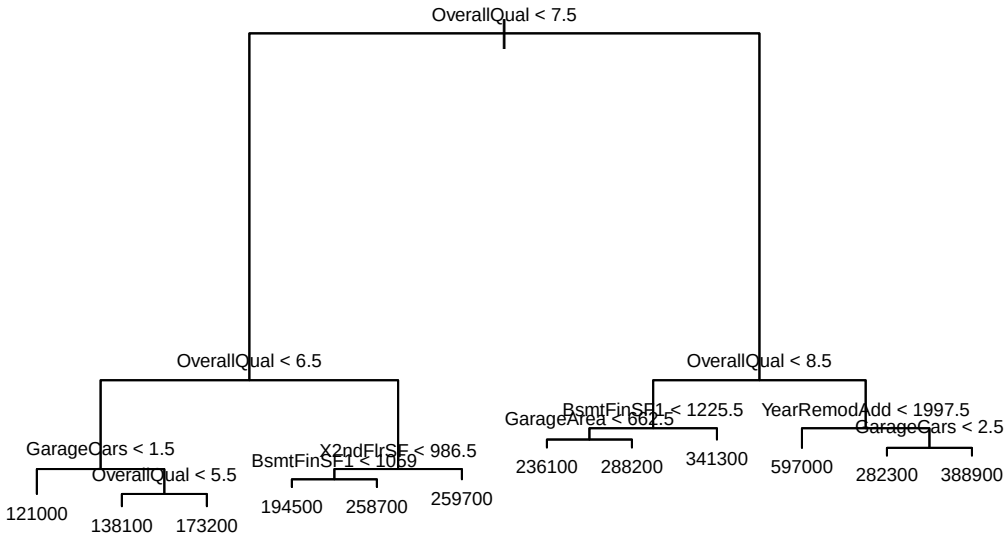
```
ridge.pred <- predict(ridge.fit, newx = test.matrix, s = ridge.fit$lambda.min)
lasso.pred <- predict(lasso.fit, newx = test.matrix, s =lasso.fit$lambda.min)
```

---

5. Regression Trees

```
ind <- sample(1:nrow(train), nrow(train)/4)
tree.train <- train[-ind,]
tree.test <- train[ind,]
tree.temp <- tree(SalePrice~., tree.train)
cv.data <- cv.tree(tree.temp, K = 10)
best.size <- cv.data$size[which.min(cv.data$dev)]
prune.data <- prune.tree(tree.temp, best = best.size)
tree.pred.temp <- predict(prune.data, tree.test)
tree.rmse2 <- mean((tree.test$SalePrice- tree.pred.temp)^2)
tree.rmse <- round(sqrt(tree.rmse2))
```

```
tree.data <- tree(SalePrice ~ ., train)
plot(tree.data)
text(tree.data,pretty=0, cex = 0.6)
```

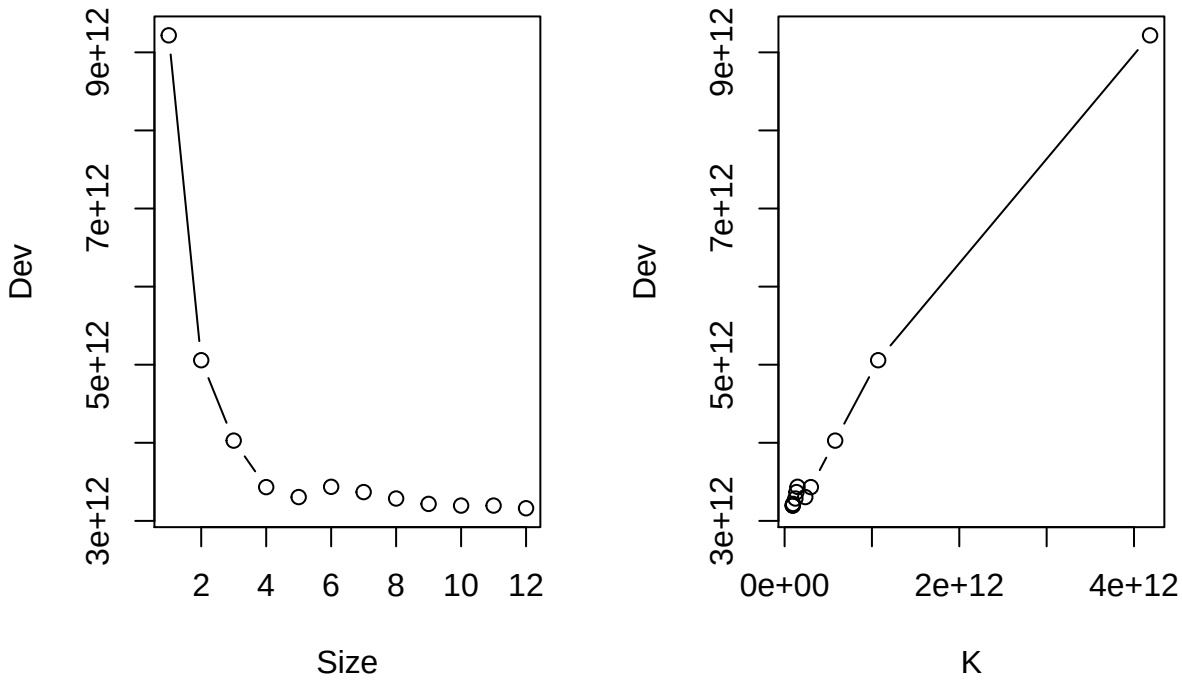


Now we perform cross-validation in order to determine the optimal level of tree complexity

```
cv.data <- cv.tree(tree.data, K = 10)
par(mfrow = c(1,2))

plot(cv.data$size,
     cv.data$dev,
     type = "b",
     xlab="Size",
     ylab = "Dev")

plot(cv.data$k,
     cv.data$dev,
     type = "b",
     xlab = "K",
     ylab = "Dev")
```



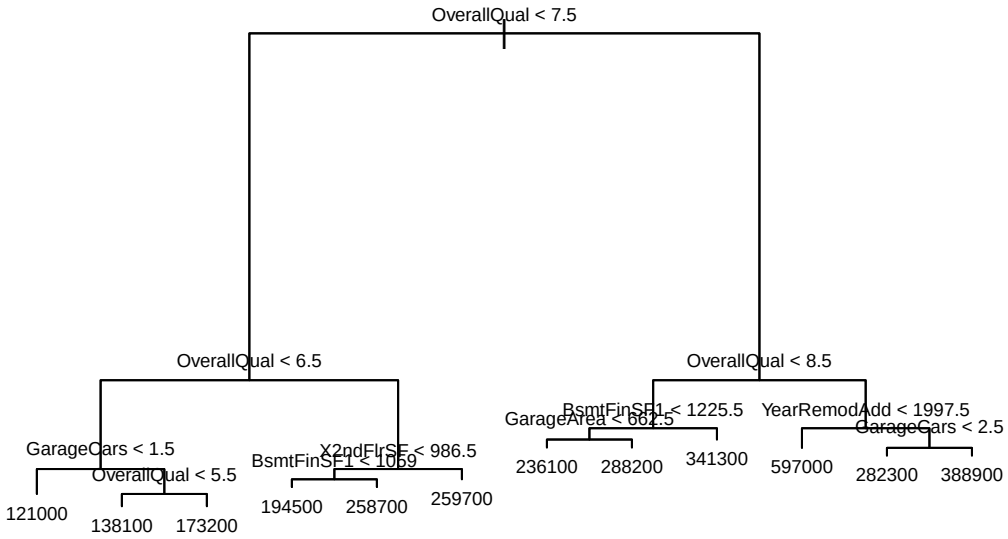
We obtain the best tree size by tuning the size before we prune. The best size is the minimum size obtained from the cross validation.

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

The best size tree according to cross validation is 12

Pruning Tree

```
prune.data <- prune.tree(tree.data, best = best.size)
plot(prune.data)
text(prune.data, pretty = 0, cex = 0.6)
```



Predicting with pruned tree

```
tree.pred <- predict(prune.data, test)
```

Pruned Regression Trees gave RMSE of 48721

---

6 Bagging

When perfoming Bagging, calculating mtry as $ncol(tree_train) - 1$ will yield the best results. Therefore, that's how I chose my tuning parameter for Bagging

```
bag.data <- randomForest(SalePrice ~ ., data = train, mtry = ncol(train)-1, importance = TRUE)
bag.rmse <- round(sqrt(mean(bag.data$mse)))
```

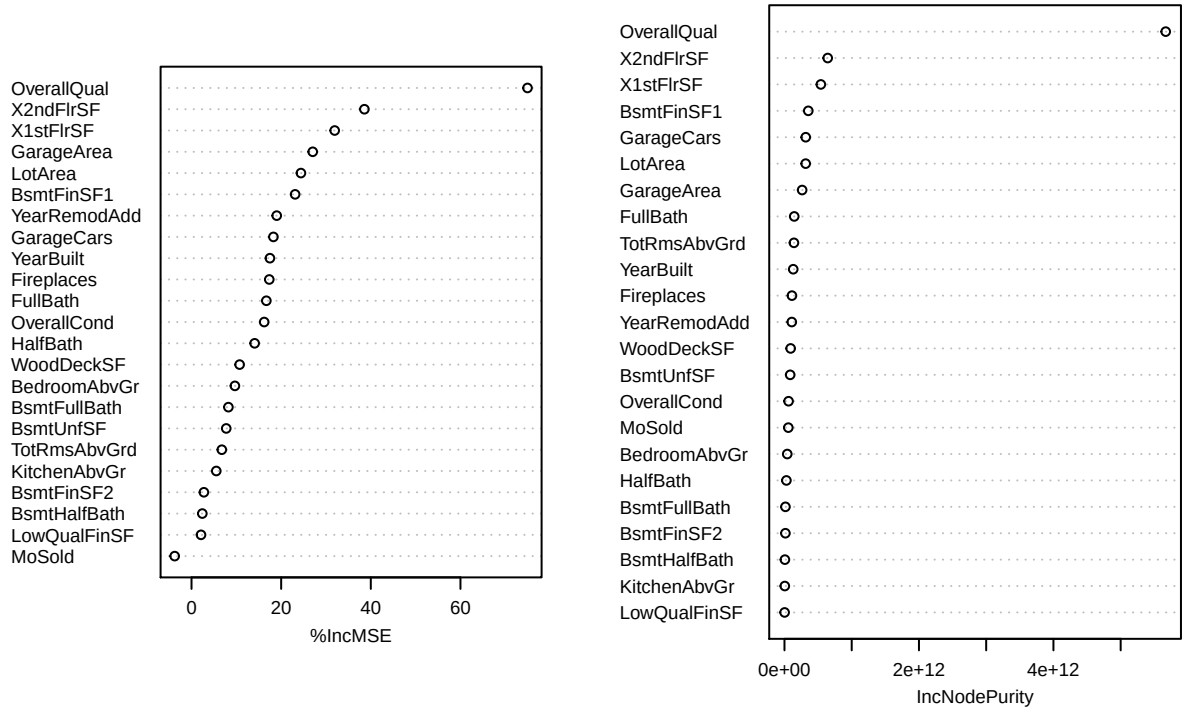
Bagging gave RMSE of 30188

Model Interpretation

```r
importance(bag.data)
```

```
##                %IncMSE IncNodePurity
## LotArea      24.407460  3.153609e+11
## OverallQual  74.959677  5.669740e+12
## OverallCond  16.209643  6.093807e+10
## YearBuilt    17.491047  1.300776e+11
## YearRemodAdd 19.008281  1.086239e+11
## BsmtFinSF1   23.111007  3.542122e+11
## BsmtFinSF2    2.758378  1.384929e+10
## BsmtUnfSF     7.743722  8.492850e+10
## X1stFlrSF    31.942039  5.411740e+11
## X2ndFlrSF    38.568874  6.430279e+11
## LowQualFinSF  2.117662  2.204917e+09
## BsmtFullBath  8.226356  1.450661e+10
## BsmtHalfBath  2.409065  6.607248e+09
## FullBath     16.692898  1.450892e+11
## HalfBath     14.083417  2.860342e+10
## BedroomAbvGr  9.679325  4.120499e+10
## KitchenAbvGr  5.520534  4.767069e+09
## TotRmsAbvGrd  6.755581  1.410330e+11
## Fireplaces   17.347169  1.110113e+11
## GarageCars   18.269244  3.160314e+11
## GarageArea   27.037898  2.627677e+11
## WoodDeckSF   10.730333  9.157761e+10
## MoSold       -3.757409  5.727091e+10
```

```r
varImpPlot(bag.data, cex = 0.6)
```



From this, `OverallQual` seems to be the most important feature to accurately predict `SalePrice`

Predicting with Bagging

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

---

7 Random Forest

When perfoming RandomForest, similar to Bagging, calculating mtry as $sqrt(ncol(tree.train) - 1)$ will yield the best results. Therefore, that's how I chose my tuning parameter for RandomForest

```
rf.data <- randomForest(SalePrice ~ .,
                        data = train,
                        mtry = round(sqrt(ncol(train)-1)),
                        importance = TRUE)
rf.rmse <- round(sqrt(mean(rf.data$mse)))
```

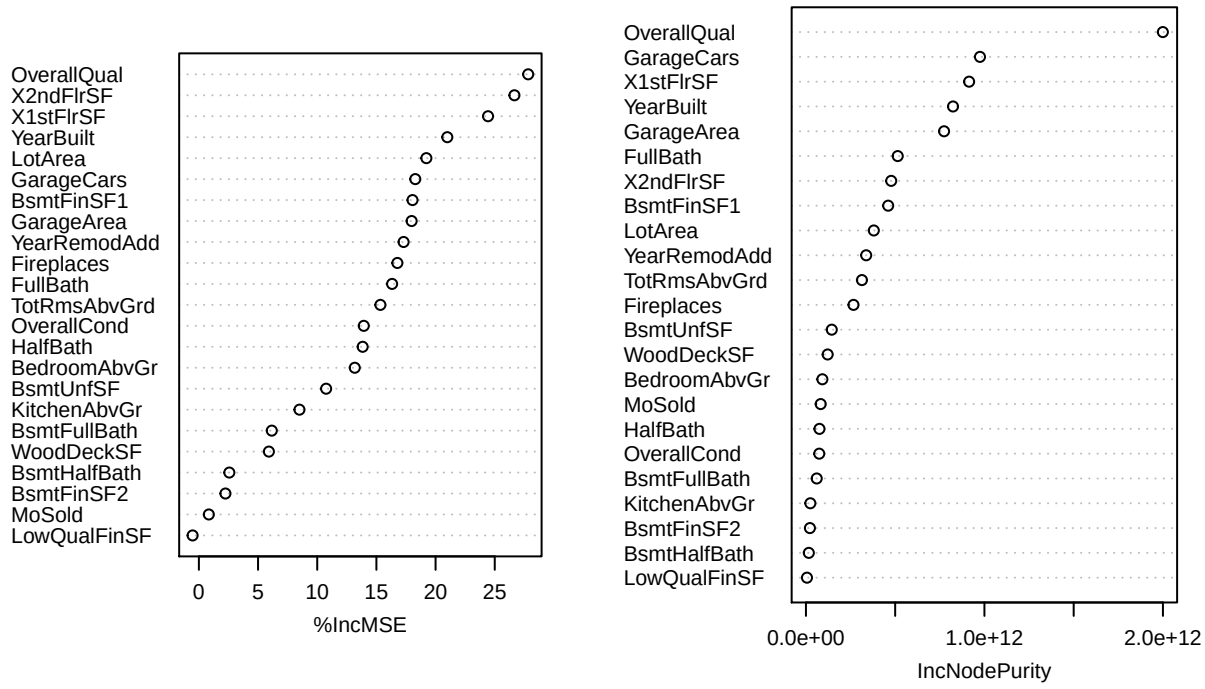
RandomForest gave RMSE of 30211

Model Interpretation

```
importance(rf.data)
```

```
##                 %IncMSE IncNodePurity
## LotArea      19.2236847  3.802865e+11
## OverallQual  27.8258959  1.999512e+12
## OverallCond  13.9350102  7.454177e+10
## YearBuilt    20.9916262  8.239701e+11
## YearRemodAdd 17.2994356  3.375848e+11
## BsmtFinSF1   18.0515354  4.607579e+11
## BsmtFinSF2    2.2434287  2.225378e+10
## BsmtUnfSF    10.7534886  1.446332e+11
## X1stFlrSF    24.4350019  9.142690e+11
## X2ndFlrSF    26.6645999  4.776203e+11
## LowQualFinSF -0.5310949  5.836264e+09
## BsmtFullBath  6.1591956  6.037035e+10
## BsmtHalfBath  2.5731757  1.626695e+10
## FullBath     16.3276987  5.143408e+11
## HalfBath     13.8387235  7.580380e+10
## BedroomAbvGr 13.1730422  9.214797e+10
## KitchenAbvGr  8.5001373  2.465682e+10
## TotRmsAbvGrd 15.3384950  3.138156e+11
## Fireplaces   16.7736107  2.664227e+11
## GarageCars   18.2865368  9.750733e+11
## GarageArea   17.9771493  7.739927e+11
## WoodDeckSF    5.9148006  1.217418e+11
## MoSold        0.8406090  8.272976e+10
```

```
varImpPlot(rf.data, cex = 0.7)
```



From the plot, we can see that `OverallQual`, `GarageCars`, `GarageArea`, `X1stFlrSF` and `YearBuilt` are the most important features when trying to accurately predict `SalePrice`

Predicting with RandomForest

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

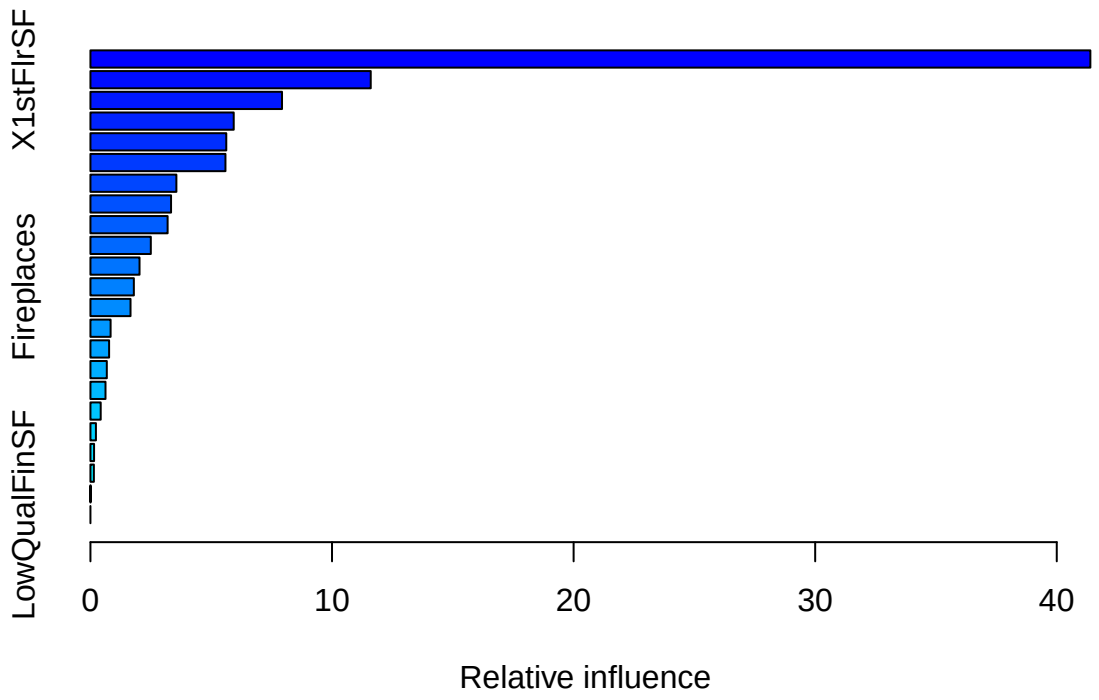
---

8 Boosting

10 fold cross validation to select the best number of trees

I chose shrinkage to be 0.01 and n.tree to be 5000 as this is the examples most widely used in class and on online resources.

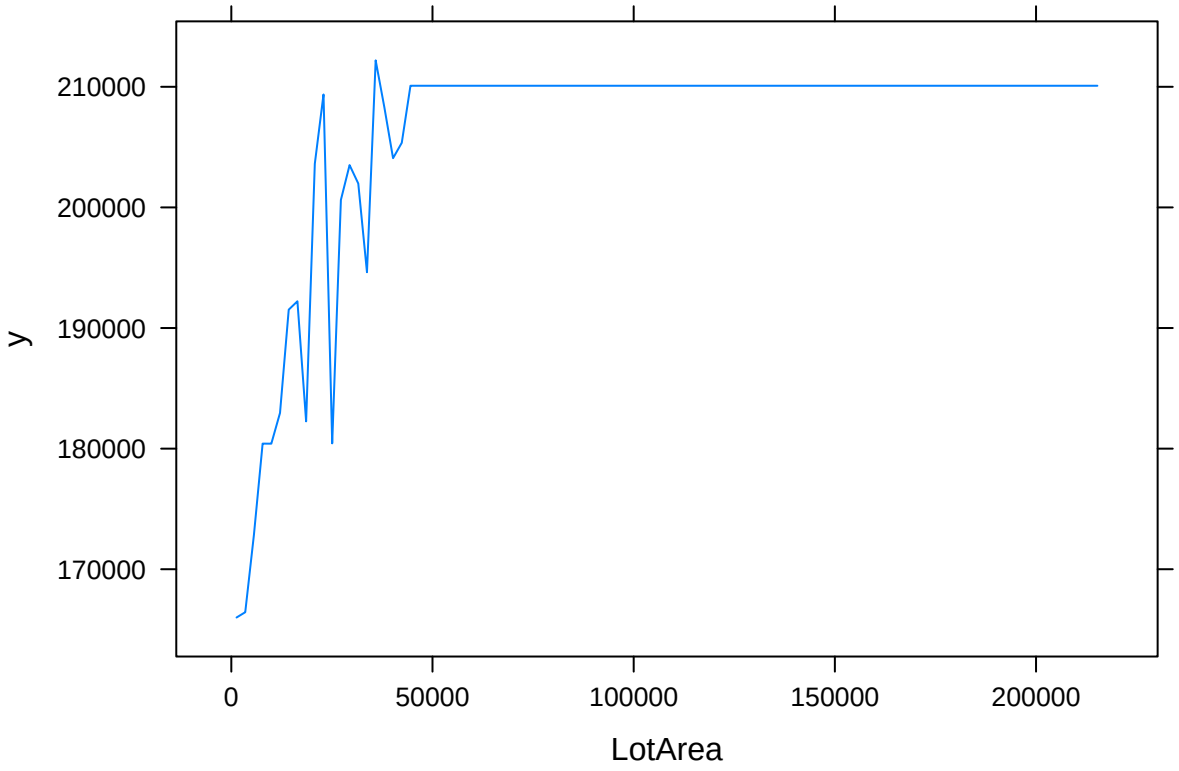
```
boost.data <- gbm(SalePrice ~ .,
                          data = train[,colnames(train) != "SalePrice"],
                          distribution = "gaussian", shrinkage = 0.01,
                          n.tree = 5000, interaction.depth = 4, cv.folds = 10)
boost.rmse <- round(sqrt(mean(boost.data$cv.error)))
summary(boost.data)
```



Boosting gave RMSE of 32020

Model Interpretation

```
plot(boost.data)
```



We can see that as `OverallQual` increments up, the value of the property is estimated to increase almost quadrratically.

Predicting with Boosting

```
boost.pred <- predict(boost.data, newdata = test,
                      n.trees = which.min(boost.data$cv.error),
                      type = "response")
```

---

9 General Additive Models

Obtaining RMSE using 10-fold cross-validation.

```
ctrl <- trainControl(method = "cv", number = 10)
gam.cv <- train(SalePrice ~.,
                      data = train,
                      trControl = ctrl,
                      method = "gamSpline")

gam.rmse <- round(gam.cv$results$RMSE[which.min(gam.cv$results$RMSE)])
```

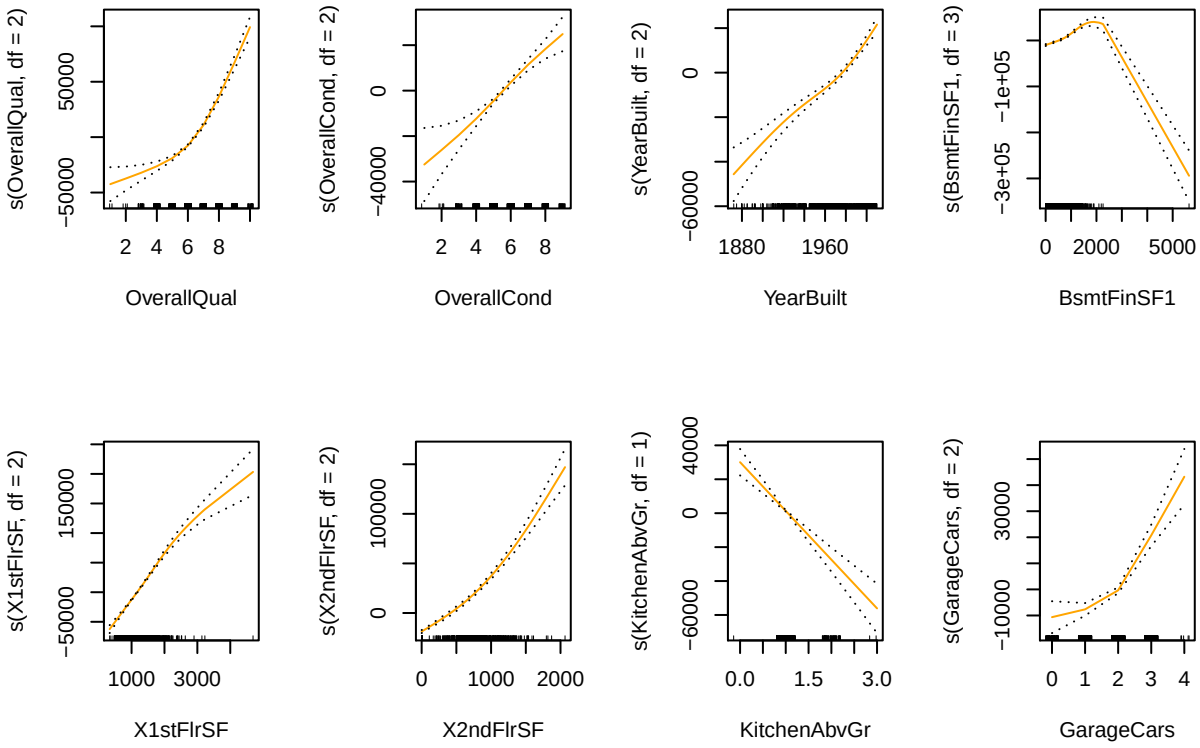
GAM method gave RMSE of 30943

```
gam.data <- gam(SalePrice ~ s(OverallQual, df=2)+ s(OverallCond, df =2)+ s(YearBuilt, df =2)+ s(BsmtFinS
```

I tuned the degrees of freedom by analyzing the plots below and determining based on that.

Model interpretation

```
par(mfrow = c(2,4))
plot(gam.data, se = TRUE, col = "orange")
```



Predicting with GAM

```
gam.pred <- predict(gam.data, newdata = test)
```

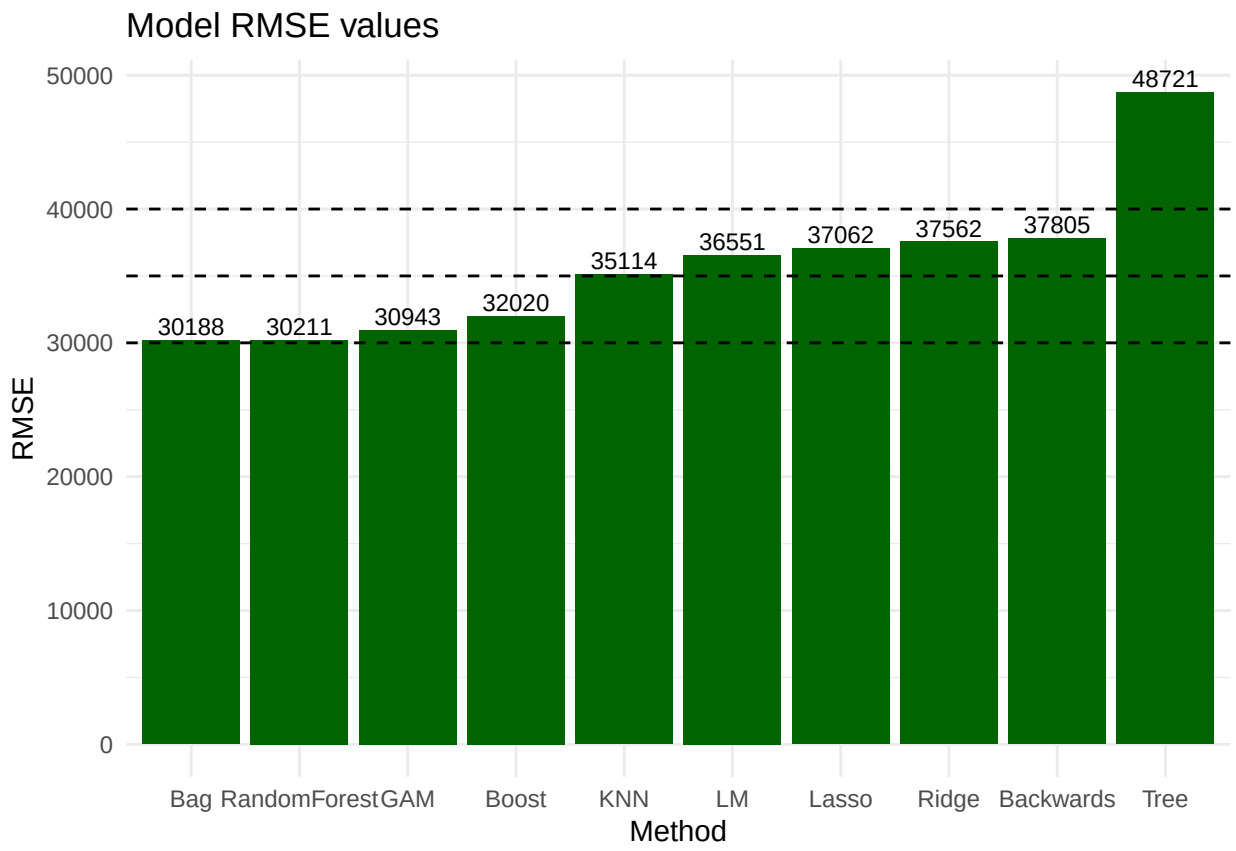
Estimating Test Errors and True Test Errors

```
RMSE.list <- c(knn.rmse, lm.rmse, back.rmse, ridge.rmse, lasso.rmse, tree.rmse, bag.rmse, rf.rmse, boos
RMSE.df <- data.frame(Model=c("KNN", "Linear Regression", "Backwards Step-wise", "Ridge", "Lasso", "Tre
                      RMSE = RMSE.list)
RMSE.df
```

```
##                   Model  RMSE
## 1                   KNN 35114
## 2    Linear Regression 36551
## 3  Backwards Step-wise 37805
## 4                 Ridge 37562
## 5                 Lasso 37062
## 6                  Tree 48721
## 7               Bagging 30188
## 8          RandomForest 30211
## 9              Boosting 32020
## 10                  GAM 30943
```

```
methods <- c("KNN", "LM", "Backwards", "Ridge", "Lasso", "Tree", "Bag", "RandomForest", "Boost", "GAM")
results <- data.frame(matrix(data=NA, nrow=length(methods), ncol=0))
results$Method <- methods
results$rmse <- RMSE.df$RMSE
```

```
results$Test.Error.Rate<- as.numeric(as.character(results$rmse))
results$labs <- methods
results <- results[order(results$rmse),]
rownames(results) <- 1:10
ggplot(results, aes(x=reorder(labs, rmse), y = rmse)) +
  geom_bar(stat="identity", fill = "darkgreen") +
  labs(title = "Model RMSE values", x = "Method", y = "RMSE") +
  geom_hline(yintercept=c(30000,35000,40000), linetype = "dashed", color = "black") +
  geom_text(aes(label=round(rmse,2)), vjust = -0.3,color="black", size=3.0) +
  theme_minimal()
```



Predicting which models will succeed

From this graph, I have to assume that RandomForest, Bagging, Boosting and GAM methods will perform very similarly. This is because of their respective CV errors. On the other hand, Regression Trees is likely to be the worst performing method.

Results after submitting

After submitting, my *Boosted* method performed the best, with a score of 0.14. Conversely, my worst model was Backwards Step-Wise. This is almost what I predicted through obtaining the RMSE for each model.

Why did some models do better than others?

This is because the relationship between the features and `SalePrice` wasn't always linear. Therefore, methods like linear regression obviously performed the worst. The Gradient Boosted method performed the best because it uses an additive model to add weak learners to minimize the loss function. This is similar to GAM, which is why GAM method performed nearly as well.

Conclusion and Summary

In the end, all 4 of my models with the lowest RMSE values had the lowest error after submitting to the Kaggle competition. I was surprised to see that Regression Trees did as poorly as it did given that Random Forest, Bagging and Boosting did so well. The models that worked the best are the models that didn't

assume normality in the data. In conclusion, this project has helped compile many different important regression topics into one cohesive project that can be referenced for future work if needed.

Discussion of Further Questions

Future work could include pre-processing the data and potentially using features not present in the pre-cleaned data. Additionally, more work could be done surrounding the analysis of the features, with more visualization to help guide the next person towards seeing the most important variables from the beginning in a more clean fashion rather than solving for significance at each step. Finally, more work could be done to create additional models such as Support Vector Machines and possibly even Neural Networks.

Create Predictions for Kaggle Competition

KNN Method

```
knn.kaggle <- data.frame("Id" = 1461:2919, "SalePrice"= knn.pred$pred[1:1459])
write.csv(knn.kaggle, "KNN_Kaggle.csv", row.names = FALSE)
```

Linear Regression

```
lm.kaggle <- data.frame("Id" = 1461:2919, "SalePrice"= lm.pred)
write.csv(lm.kaggle, "Linear_Regression_Kaggle.csv", row.names = FALSE)
```

Backwards Step-wise

```
back.kaggle <- data.frame("Id" = 1461:2919, "SalePrice"= back.pred)
write.csv(back.kaggle, "Backwards_Stepwise_Kaggle.csv", row.names = FALSE)
```

Ridge Regression

```
ridge.kaggle <- data.frame("Id" = 1461:2919, "SalePrice"= ridge.pred)
write.csv(ridge.kaggle, "Ridge_Kaggle.csv", row.names = FALSE)
```

Lasso

```
lasso.kaggle <- data.frame("Id" = 1461:2919, "SalePrice" = lasso.pred)
write.csv(lasso.kaggle, "Lasso_Kaggle.csv", row.names = FALSE)
```

Regression Tree

```
tree.kaggle <- data.frame("Id" = 1461:2919, "SalePrice" = tree.pred)
write.csv(tree.kaggle, "Tree_Kaggle.csv", row.names = FALSE)
```

Bagging

```
bag.kaggle <- data.frame("Id" = 1461:2919, "SalePrice" = bag.pred)
write.csv(bag.kaggle, "Bag_Kaggle.csv", row.names = FALSE)
```

RandomForest

```
rf.kaggle <- data.frame("Id" = 1461:2919, "SalePrice" = rf.pred)
write.csv(rf.kaggle, "RF_Kaggle.csv", row.names = FALSE)
```

Boosting

```
boost.kaggle <- data.frame("Id" = 1461:2919, "SalePrice" = boost.pred)
write.csv(boost.kaggle, "Boost_Kaggle.csv", row.names = FALSE)
```

General Additive Models

```
gam.kaggle <- data.frame("Id" = 1461:2919, "SalePrice" = gam.pred)
write.csv(gam.kaggle, "GAM_Kaggle.csv", row.names = FALSE)
```

Results Below

Tree_Kaggle.csv 35 minutes ago by Maxwell Waun Regression Tree	0.24341
Bag_Kaggle.csv 36 minutes ago by Maxwell Waun Bagging Model	0.15384
RF_Kaggle.csv 36 minutes ago by Maxwell Waun Random Forest Model	0.15315
Boost_Kaggle.csv 37 minutes ago by Maxwell Waun Boosted Model	0.14078
GAM_Kaggle.csv 38 minutes ago by Maxwell Waun GAM Model	0.15498
KNN_Kaggle.csv a month ago by Maxwell Waun using KNN	0.55223
Lasso_Kaggle.csv a month ago by Maxwell Waun using Lasso	0.45489
Ridge_Kaggle.csv a month ago by Maxwell Waun using Ridge Regression	0.23164
Backwards_Stepwise_Kaggle.csv a month ago by Maxwell Waun Using Backwards Step-wise selection	0.45533
Linear_Regression_Kaggle.csv a month ago by Maxwell Waun Using Linear Regression	0.45105