Examining predictive models for housing prices

2022-05-01

```
#read in data
House_train<- read.csv("train_new.csv")
House_test<- read.csv("test_new.csv")
```

Introduction To Data

In this problem we are tasked with predicting the Sale Price of over a thousand different houses. This raw data has 79 different predictors relating to a plethora of housing characteristics of residential homes in Ames Iowa. We must use statistical and machine learning models to accomplish this feat of prediction.

Explanation of the raw data

As said above this data compiles 79 different predictors for 2919 houses. With the actual sale prices for half of them included to be used as the training data set. These predictors range from quantitative variables such as LotArea (size of the lot in square feet), BsmtQual (height of the basement), and TotalBsmtSF (square feet of the basement) to qualitative variables such as BsmtFinType1 (Quality of basement finished area), HeatingQC (Heating quality and condition), and Street (type of road access). A lot of the qualitative variables are rated on a scale of 1-10 with 10 being the best and 1 being the worst.

Explanation of how the data was cleaned

In this altered data there are no missing values and all categorical variables are removed. This leaves 23 predictive variables with the qualitative ones being replaced with dummy variables.

KNN

```
set.seed(10)
House_train_no_prices<- House_train[,-24]

#test different k values to see how they perform
House.mse <- rep(0,300)
counter <- 0
for(k in 1:300){
counter <- counter + 1 # counter for k
knn <- knn.reg(House_train_no_prices, NULL, House_train$SalePrice, k = k)
# the little k here is the number of nearest neighbors
House.mse[counter] <- mean(knn$residuals^2)
}

#find our best performing k value in terms of MSE
which.min(House.mse)
```

```
[1] 4
```

How the tuning parameter was chosen

In the code above, 300 values of k were tested and the one with the lowest MSE value was chosen, that being k=4

```
#predict using our best performing k value
knn_predict<- knn.reg(House_train_no_prices, House_test[,-24], House_train$SalePrice, k=4)
head(knn_predict$pred)
```

```
[1] 143825.0 175225.0 165250.0 220697.5 139970.0 188668.8
```

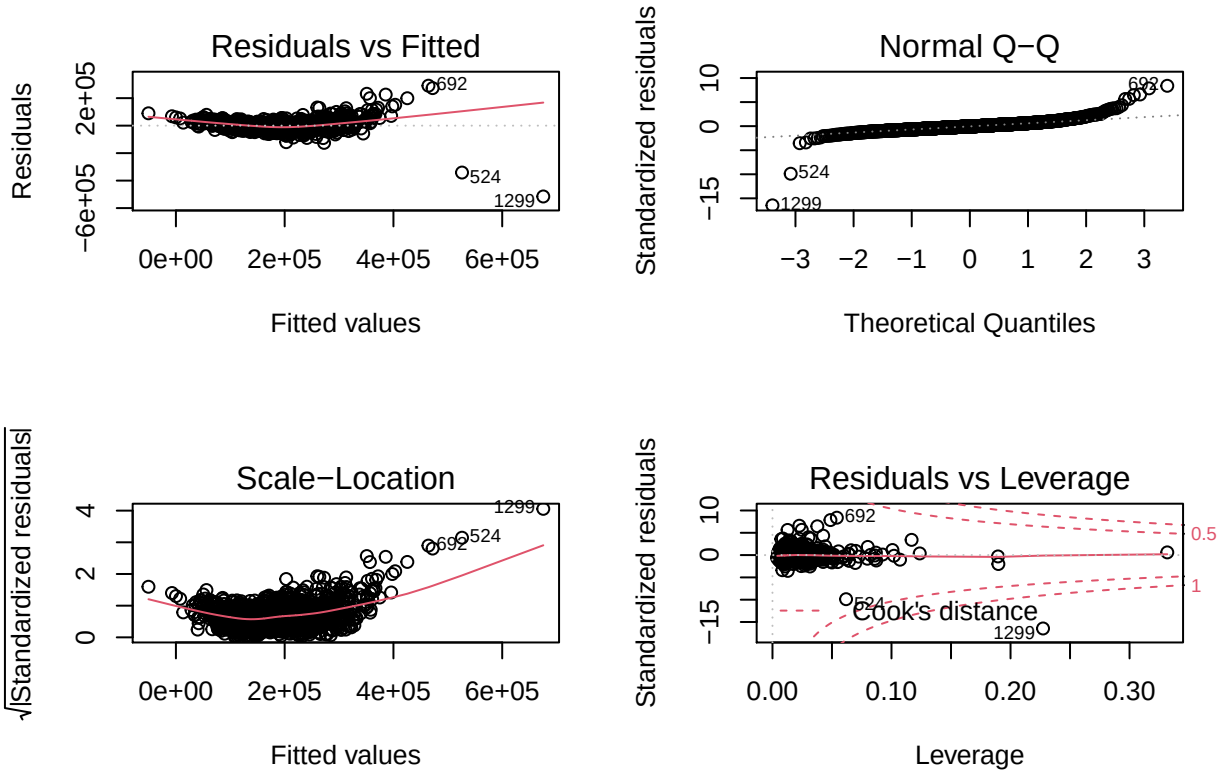
Linear Regression

```
House_lm<- lm(SalePrice~ ., data = House_train )
summary(House_lm)
```

```
Call:
lm(formula = SalePrice ~ ., data = House_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 ***
```

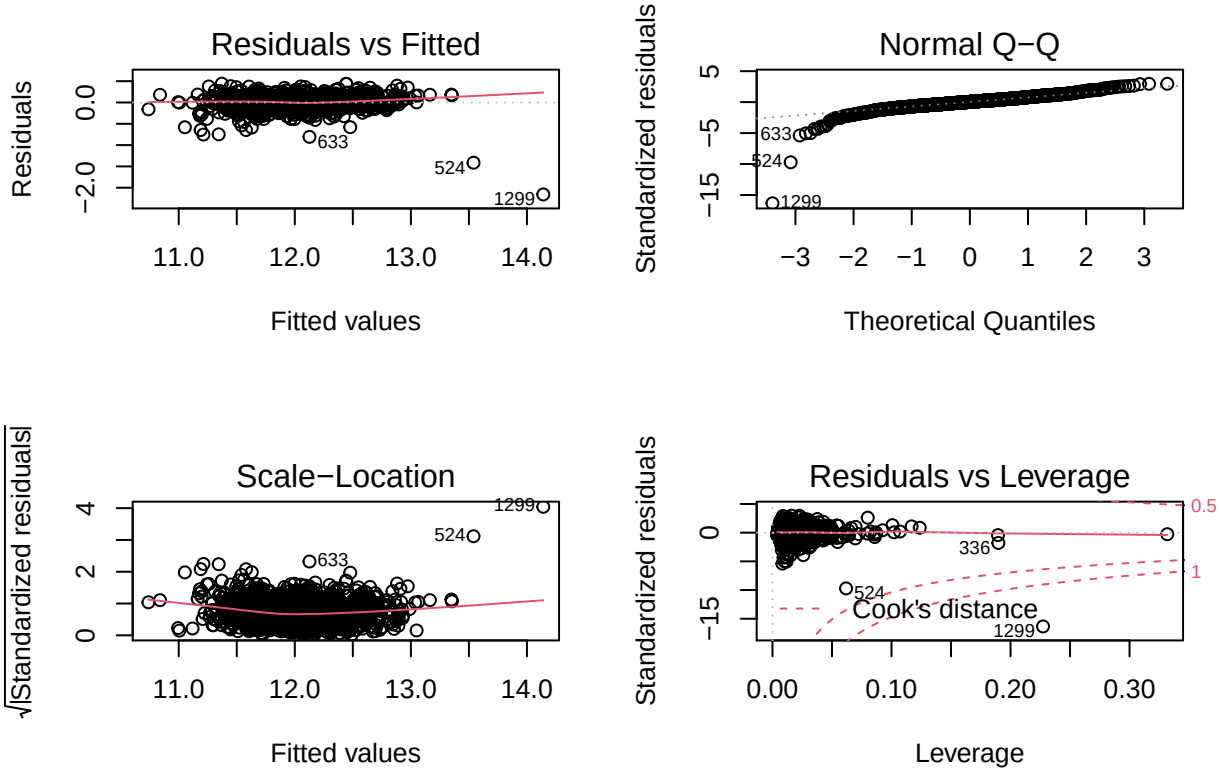
```
GarageArea    5.430e+00  9.861e+00   0.551 0.581942
WoodDeckSF    2.125e+01  8.036e+00   2.644 0.008278 **
MoSold        5.284e+01  3.480e+02   0.152 0.879328
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

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
```

```
#do residual diagnostics
par(mfrow=c(2,2))
plot(House_lm)
```

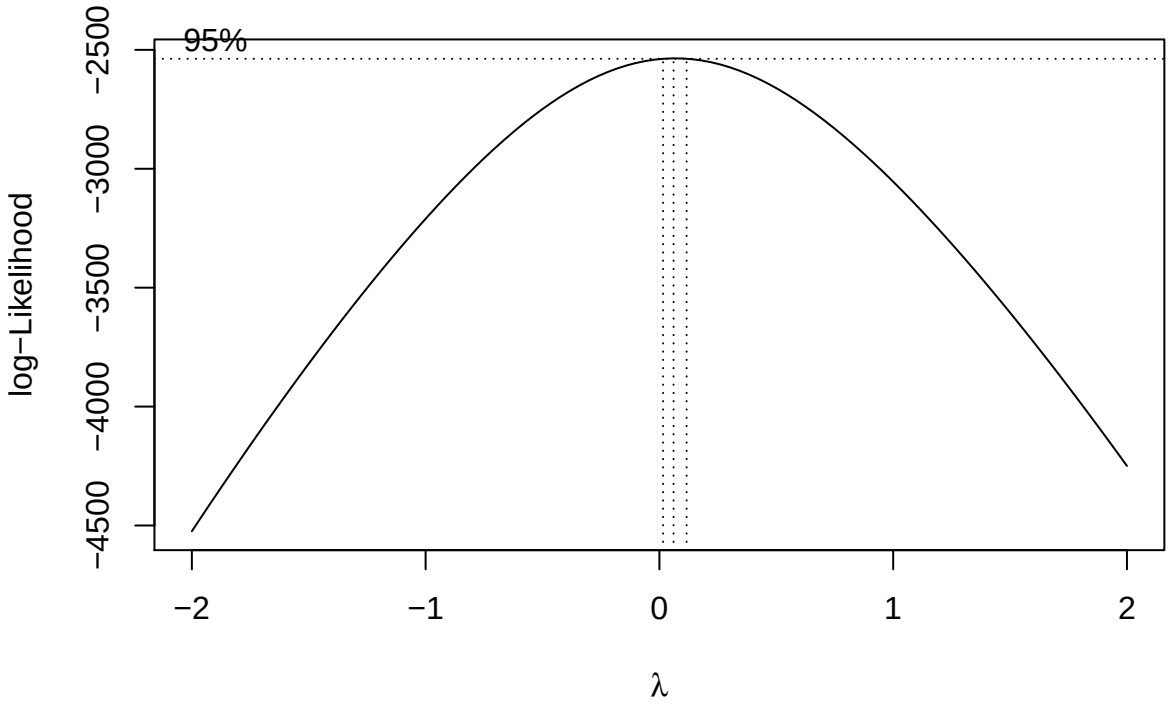


First off we want to check out the normality of the residuals. Ideally we want the residuals on the QQplot to fall along the line. In this model we can see that near the far right of the graph the residuals start to stray away from the line. There are multiple ways we can fix this, though in terms of this data, I believe we should perform a Y transformation.

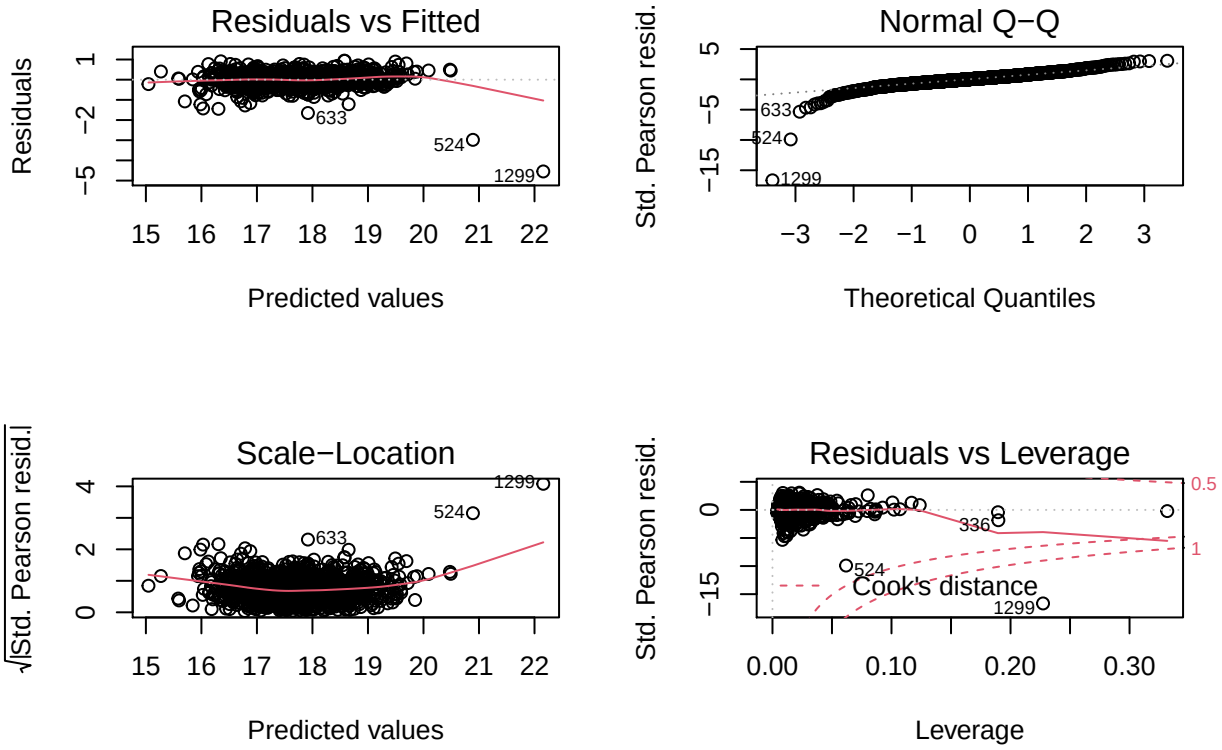
```
#log transform of Y
House_lm_log_y<- lm(log(SalePrice)~ ., data = House_train )


#do residual diagnostics
par(mfrow=c(2,2))
plot(House_lm_log_y)
```




```
#boxcox transform of Y
boxcox<- boxcox(House_lm)
```




```
#want to find the lambda value that gives us the max y value
lambda<- boxcox$x[which.max(boxcox$y)]

#now we can apply the optimal lambda to our model and transform Y
House_lm_boxcox<- glm(((SalePrice^lambda-1)/lambda)~ ., data = House_train )
par(mfrow=c(2,2))
plot(House_lm_boxcox)
```



We can see both these Y transformations have a much better fit on the Normal Q-Q line. For each of these models we can we can check the other residual diagnostics. We can see in each model point 1299 is considered a high leverage point with a cooks distance of > 1. Though unless we thought that this outlier was from a sampling error we wouldn't remove it despite its high impact on the regression line. Lastly we want to see if the residuals vs fitted values resemble a line close to the ideal line of y=0. Both these models seem to have a fairly straight line close to y = 0's line.

```
#looking for the most significant coefficients
summary(House_lm_boxcox)
```

```
Call:
glm(formula = ((SalePrice^lambda - 1)/lambda) ~ ., data = House_train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.5475  -0.1444   0.0108   0.1609   0.9422

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.452e-01  1.158e+00   0.125 0.900293
LotArea       4.178e-06  8.940e-07   4.674 3.24e-06 ***
OverallQual   1.736e-01  1.045e-02  16.608  < 2e-16 ***
OverallCond   9.951e-02  9.034e-03  11.015  < 2e-16 ***
YearBuilt     5.217e-03  5.091e-04  10.248  < 2e-16 ***
YearRemodAdd  2.141e-03  5.844e-04   3.664 0.000258 ***
```

```
BsmtFinSF1    1.737e-04  4.115e-05   4.221 2.58e-05 ***
BsmtFinSF2    1.606e-04  6.255e-05   2.568 0.010334 *
BsmtUnfSF     1.312e-04  3.714e-05   3.531 0.000427 ***
X1stFlrSF     4.175e-04  5.087e-05   8.209 4.92e-16 ***
X2ndFlrSF     2.950e-04  4.241e-05   6.954 5.36e-12 ***
LowQualFinSF  2.267e-04  1.747e-04   1.297 0.194806
BsmtFullBath  1.199e-01  2.308e-02   5.194 2.36e-07 ***
BsmtHalfBath  3.466e-02  3.638e-02   0.953 0.340761
FullBath      7.474e-02  2.496e-02   2.994 0.002801 **
HalfBath      5.176e-02  2.361e-02   2.193 0.028489 *
BedroomAbvGr  2.575e-04  1.496e-02   0.017 0.986264
KitchenAbvGr -2.191e-01  4.318e-02  -5.075 4.37e-07 ***
TotRmsAbvGrd  4.103e-02  1.091e-02   3.760 0.000177 ***
Fireplaces    9.942e-02  1.564e-02   6.359 2.72e-10 ***
GarageCars    1.468e-01  2.535e-02   5.793 8.51e-09 ***
GarageArea    7.500e-05  8.593e-05   0.873 0.382930
WoodDeckSF    1.847e-04  7.003e-05   2.637 0.008455 **
MoSold        2.894e-03  3.032e-03   0.954 0.340125
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 0.09646879)

    Null deviance: 1003.97  on 1459  degrees of freedom
Residual deviance:  138.53  on 1436  degrees of freedom
AIC: 754.84

Number of Fisher Scoring iterations: 2
```

Model Interpretation

As we can see from the summary, some of the most significant coefficients are Fireplaces, X1stFlrSF, X2ndFlrSF , and GarageCars. Fire places represents the amount of fire places in the house, X1stFlrSF and X2ndFlrSF represent the square ft of the first and second floors respectively and GarageCars represents the size of the garage in car capacity. For example this model believes for each fireplace the house has its value goes up by the inverse of the boxcox transformation 9.942e-02 dollars and for each square foot on the first floor the listed price goes up the inverse boxcox of 4.175e-04 dollars.

```
#predict using this model
House_lm_predict<- predict(House_lm_boxcox, House_test)
#undo boxcox to understand predictions
lr_pred_values<- (House_lm_predict*lambda+1)^(1/lambda)
head(lr_pred_values)
```

```
       1        2        3        4        5        6
113547.3 146898.7 168796.4 197042.7 183458.7 176347.0
```

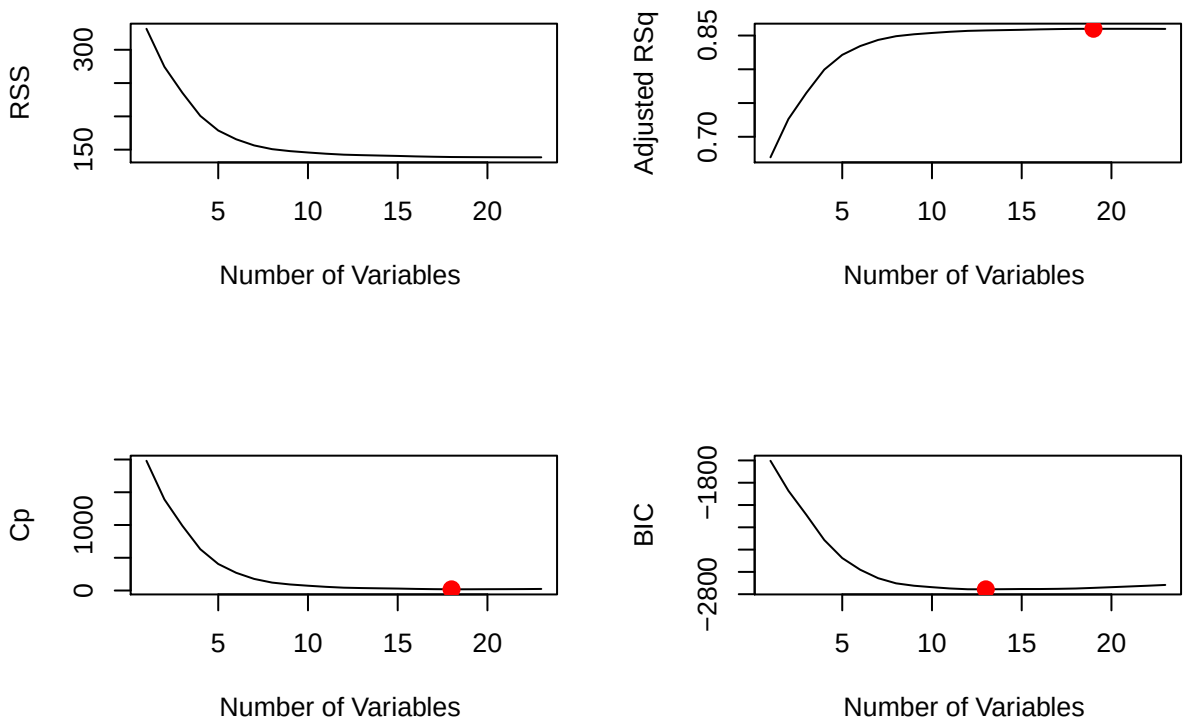
Subset Selection

```
#when doing subset selection we must choose our method of either Best subset, forward stepwise, or back
ncol(House_train)
```

```
[1] 24
```

We can see our data has 23 predictors for predicting Saleprice. Since we can expend the extra computing time used for best subset method given our lower number of predictors, that is the method we should choose for the best accuracy.

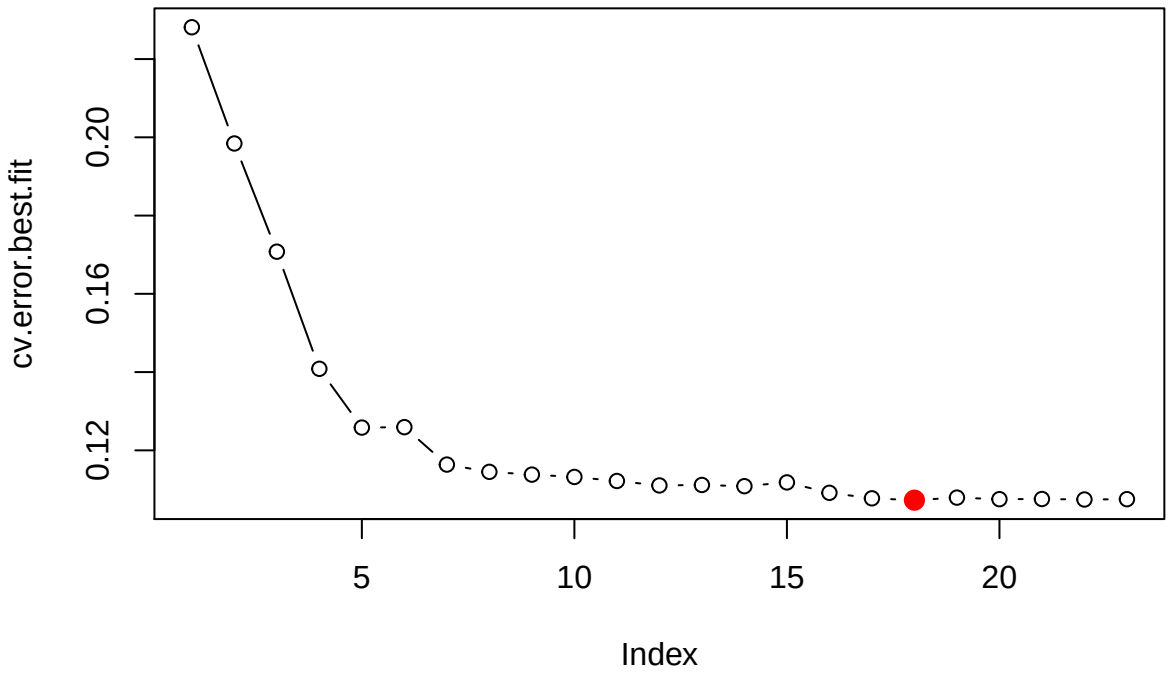
```
library(leaps)
regfit.full <- regsubsets(((SalePrice^lambda-1)/lambda) ~ . , data = House_train, nvmax = 23)
reg.summary<- summary(regfit.full)

#RSS graph
par(mfrow=c(2,2))
plot(reg.summary$rss ,xlab="Number of Variables ", ylab="RSS", type="l")
plot(reg.summary$adjr2 ,xlab="Number of Variables ", ylab="Adjusted RSq", type="l")

#Adj RSq Plot
points(which.max(reg.summary$adjr2),reg.summary$adjr2[which.max(reg.summary$adjr2)], col = "red", cex =
plot(reg.summary$cp, xlab = "Number of Variables ", ylab = "Cp", type = "l")

#CP Plot
points(which.min(reg.summary$cp),reg.summary$cp[which.min(reg.summary$cp)], col = "red", cex = 2, pch =

#BIC Plot
plot(reg.summary$bic, xlab = "Number of Variables ", ylab = "BIC", type = "l")
points(which.min(reg.summary$bic),reg.summary$bic[which.min(reg.summary$bic)], col = "red", cex = 2, pc
```



While we have all these criteria, our obvious end goal is to predict house prices accurately, and the method that best simulates this would be cross validation as that's what its best suited towards (simulating the test data when we don't have the real test data).

```
#using cross validation to choose a model

#first we must make a predict 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 containing id n
xvars <- names(coefi) # names of the non-zero coefficient estimates
return(mat[,xvars] %*% coefi) # X[,non-zero variables] %*% Coefficients[non-zero variables]
}


#prepare and use 10-fold cross validation

fold.index <- cut(sample(1:nrow(House_train)), breaks=10, labels=FALSE)

cv.error.best.fit <- rep(0,23)
for(i in 1:23){ # try different numbers of variables
error <- rep(0, 10)

for (k in 1:10){
```

```
House.train <- House_train[fold.index != k,]
House.test <- House_train[fold.index == k,]
true.y <- ((House.test[,"SalePrice"]^lambda-1)/lambda)
best.fit <- regsubsets(((SalePrice^lambda-1)/lambda) ~ ., data = House.train, nvmax = 23)
pred <- predict(best.fit, House.test, id = i)
error[k] <- mean((pred - true.y)^2)
}
cv.error.best.fit[i] <- mean(error)
}


par(mfrow=c(1,1))
plot(cv.error.best.fit, type = "b")
points(which.min(cv.error.best.fit), cv.error.best.fit[which.min(cv.error.best.fit)],
col = "red", cex = 2, pch = 20)
```



How the tuning parameter was chosen

Comparing all the cv errors of the different model sizes allows us to choose the model size that gives us the lowest cv error. In this case it being 19 variables.

```
#view the best model according to cross validation
cv.House.best <- regsubsets(((SalePrice^lambda-1)/lambda) ~ ., data = House_train, nvmax = 23)
coef(cv.House.best, which.min(cv.error.best.fit))
```

```
  (Intercept)       LotArea   OverallQual   OverallCond     YearBuilt
 2.774034e-01  4.219215e-06  1.745226e-01  9.960421e-02  5.132339e-03
 YearRemodAdd    BsmtFinSF1    BsmtFinSF2     BsmtUnfSF     X1stFlrSF
 2.166775e-03  1.819035e-04  1.692164e-04  1.313691e-04  4.196173e-04
    X2ndFlrSF  BsmtFullBath      FullBath      HalfBath  KitchenAbvGr
 2.958002e-04  1.135597e-01  7.287194e-02  4.914904e-02 -2.239024e-01
 TotRmsAbvGrd    Fireplaces    GarageCars    WoodDeckSF
 4.266053e-02  9.785981e-02  1.638806e-01  1.888466e-04
```

Model Interpretation

Our model uses 19 predictors with BsmtFullBath, GarageCars, and OverallQual having some of the largest impact within the model.

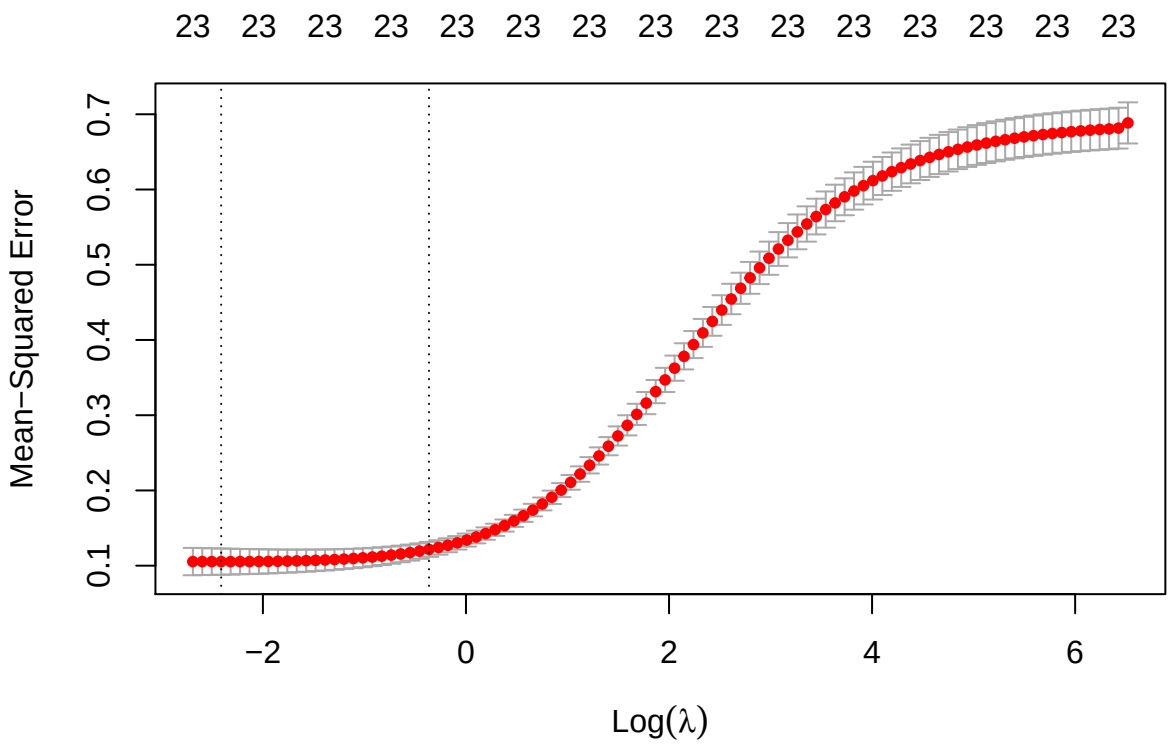
```
#predict using the best model
best_lm_pred<-predict.regsubsets(cv.House.best,House_test, id = which.min(cv.error.best.fit))

#undo boxcox to understand predictions
best_lm_pred<- (best_lm_pred*lambda+1)^(1/lambda)
head(best_lm_pred)
```

```
      [,1]
1 112163.0
2 147685.7
3 170025.4
4 197872.2
5 185203.0
6 177301.1
```

Shrinkage Methods

Ridge Regression

```
#setup for ridge regression and lasso
X <- model.matrix(((SalePrice^lambda-1)/lambda) ~., House_train)[,-1]
y <- ((House_train$SalePrice^lambda-1)/lambda)

#make model
ridge<- glmnet(X, y, alpha = 0)

#use cross validation to find the lambda value
cv.ridge <- cv.glmnet(X, y, alpha = 0, nfolds = 10)
plot(cv.ridge)
```




```
#finding the best lambda to us in the model
bestlam <- cv.ridge$lambda.min
bestlam
```

```
[1] 0.08971838
```

```
#looking at the coefficients of our best lambda's model
coef(ridge, s = bestlam)
```

```
24 x 1 sparse Matrix of class "dgCMatrix"
                         s1
(Intercept)   5.512896e-01
LotArea       4.059283e-06
OverallQual   1.587592e-01
OverallCond   7.873651e-02
YearBuilt     4.067045e-03
YearRemodAdd  3.169250e-03
BsmtFinSF1    1.749540e-04
BsmtFinSF2    1.449132e-04
BsmtUnfSF     1.208573e-04
X1stFlrSF     3.317487e-04
X2ndFlrSF     1.944098e-04
LowQualFinSF  1.315230e-04
BsmtFullBath  1.107334e-01
BsmtHalfBath  3.134695e-02
```

```
FullBath      1.123442e-01
HalfBath      8.056268e-02
BedroomAbvGr  8.693009e-03
KitchenAbvGr -2.281894e-01
TotRmsAbvGrd  4.832578e-02
Fireplaces    1.128660e-01
GarageCars    1.206827e-01
GarageArea    2.313515e-04
WoodDeckSF    2.266854e-04
MoSold        3.103354e-03
```

How the tuning parameter was chosen

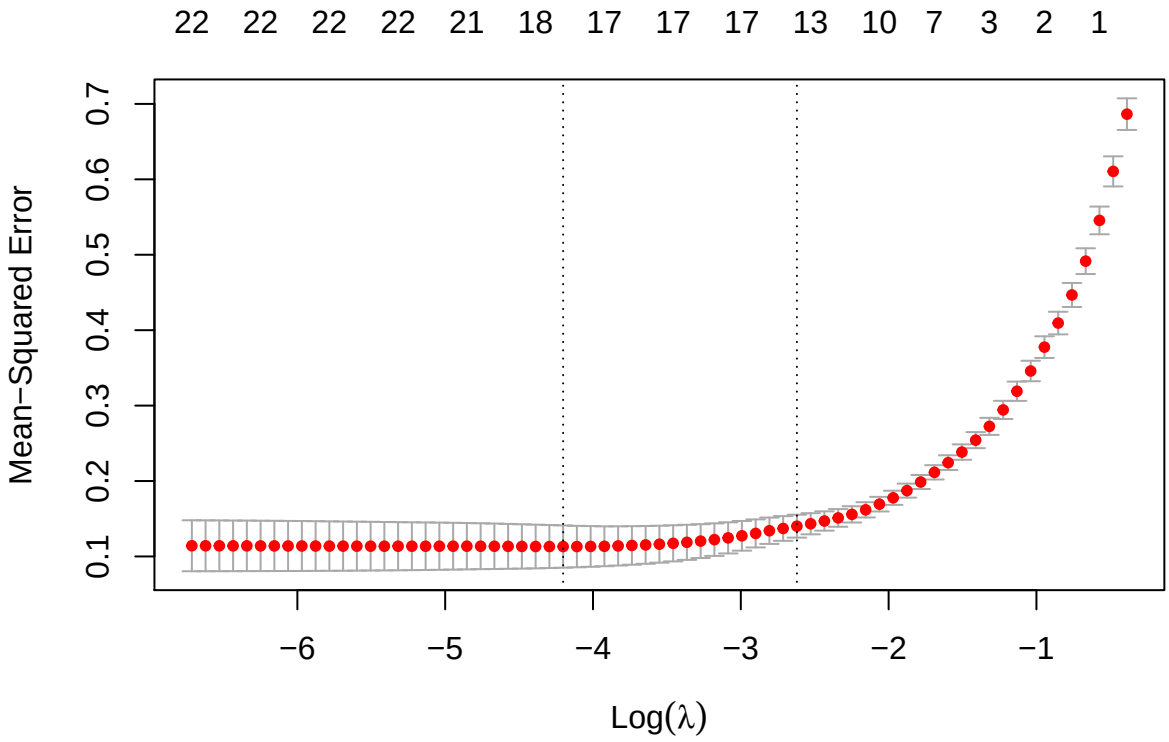
For Ridge regression we choose the tuning parameter (lambda) through means of cross validation. Cross validation compares each lambda values Mean-Squared Error allowing us to choose the lambda that gives us the lowest MSE for our model.

Model Interpretation

Looking at ridges's coefficients we can see it includes all 23 of the predictors, although they are extremely varying in terms of significance and impact on the model. With overallqual holding the greatest impact.

Lasso

```
#alpha = 1 signifies lasso
lasso<- glmnet(X, y, alpha = 1)

#use cross validation
cv.lasso <- cv.glmnet(X, y, alpha = 1)
plot(cv.lasso)
```




```
#find best lambda for cross validation
bestlam_lasso <- cv.lasso$lambda.min
coef(lasso, s = bestlam_lasso)
```

```
24 x 1 sparse Matrix of class "dgCMatrix"
                      s1
(Intercept)   1.169001e+00
LotArea       3.788439e-06
OverallQual   1.963604e-01
OverallCond   7.368894e-02
YearBuilt     4.400753e-03
YearRemodAdd  2.493157e-03
BsmtFinSF1    7.394645e-05
BsmtFinSF2    .
BsmtUnfSF     .
X1stFlrSF     4.478584e-04
X2ndFlrSF     2.268861e-04
LowQualFinSF  .
BsmtFullBath  8.918035e-02
BsmtHalfBath  .
FullBath      6.220700e-02
HalfBath      4.373271e-02
BedroomAbvGr  .
KitchenAbvGr -1.678906e-01
TotRmsAbvGrd  4.421252e-02
Fireplaces    9.678418e-02
```

```
GarageCars    1.364570e-01
GarageArea    1.081157e-04
WoodDeckSF    1.707942e-04
MoSold        .
```

How the tuning parameter was chosen

similarly to ridge regression for lasso we choose the tuning parameter (lambda) through means of cross validation. Cross validation compares each lambda values Mean-Squared Error allowing us to choose the lambda that gives us the lowest MSE for our model.

Model Interpretation

Looking at Lasso's coefficients we can see it includes all predictors except MoSold, BedroomAbvGr, BsmtHalfBath, BsmtUnfSF, BsmtFinSF2 as these predictors input reach 0 in the Lasso model. With OverallQual having the greatest impact.

Lasso does better in terms of model interpretation, as the way the formula is set up the more insignificant predictors fall truly to 0 allowing us to leave them out of our model. While in ridge regression the predictors may get extremely close to 0 but they never fall to true 0 so we must keep them in the model.

```
#Ridge predictions
test.X <- model.matrix(((SalePrice^lambda-1)/lambda) ~., House_test)[,-1]

ridge.pred <- predict(ridge, s = bestlam, newx = test.X)

#undo boxcox to understand predictions
ridge.pred<- (ridge.pred*lambda+1)^(1/lambda)
head(ridge.pred)
```

```
        s1
1 117084.7
2 147005.5
3 172344.0
4 198884.6
5 180522.8
6 177662.5
```

```
#lasso predictions

lasso.pred <- predict(lasso, s = bestlam_lasso, newx = test.X)


#undo boxcox to understand predictions
lasso.pred<- (lasso.pred*lambda+1)^(1/lambda)
head(lasso.pred)
```

```
        s1
1 115690.5
2 147729.4
```

```
3 166925.0
4 193536.4
5 189136.2
6 174785.2
```

GAM

```
# we use the model obtained from best subset selection as the skeleton for our GAM
House.GAM<- gam(((SalePrice^lambda-1)/lambda)~ s(LotArea,3) + OverallQual + OverallCond + YearBuilt + Y

summary(House.GAM)
```

```
Call: gam(formula = ((SalePrice^lambda - 1)/lambda) ~ s(LotArea, 3) +
    OverallQual + OverallCond + YearBuilt + YearRemodAdd + s(BsmtFinSF1,
    3) + s(BsmtFinSF2, 3) + s(BsmtUnfSF, 3) + s(X1stFlrSF, 3) +
    s(X2ndFlrSF, 3) + s(LowQualFinSF, 3) + BsmtFullBath + BsmtHalfBath +
    FullBath + HalfBath + KitchenAbvGr + TotRmsAbvGrd + Fireplaces +
    GarageCars + s(WoodDeckSF, 3) + MoSold, data = House_train)
Deviance Residuals:
     Min       1Q   Median       3Q      Max
-2.59320 -0.12717  0.01031  0.14227  1.02160

(Dispersion Parameter for gaussian family taken to be 0.0714)

    Null Deviance: 1003.974 on 1459 degrees of freedom
Residual Deviance: 101.5417 on 1422 degrees of freedom
AIC: 329.3468

Number of Local Scoring Iterations: NA

Anova for Parametric Effects
                    Df Sum Sq Mean Sq   F value    Pr(>F)
s(LotArea, 3)        1  73.33   73.33 1026.9034 < 2.2e-16 ***
OverallQual          1 610.44  610.44 8548.6078 < 2.2e-16 ***
OverallCond          1   1.42    1.42   19.9235 8.697e-06 ***
YearBuilt            1  29.61   29.61  414.7019 < 2.2e-16 ***
YearRemodAdd         1   3.62    3.62   50.7480 1.663e-12 ***
s(BsmtFinSF1, 3)     1  19.52   19.52  273.2910 < 2.2e-16 ***
s(BsmtFinSF2, 3)     1   1.20    1.20   16.8341 4.311e-05 ***
s(BsmtUnfSF, 3)      1  14.65   14.65  205.2074 < 2.2e-16 ***
s(X1stFlrSF, 3)      1   8.69    8.69  121.7266 < 2.2e-16 ***
s(X2ndFlrSF, 3)      1  63.05   63.05  882.9166 < 2.2e-16 ***
s(LowQualFinSF, 3)   1   0.17    0.17    2.4060  0.121097
BsmtFullBath         1   0.64    0.64    9.0234  0.002712 **
BsmtHalfBath         1   0.00    0.00    0.0000  0.995930
FullBath             1   0.00    0.00    0.0076  0.930588
HalfBath             1   0.59    0.59    8.3244  0.003971 **
KitchenAbvGr         1   2.50    2.50   35.0164 4.091e-09 ***
TotRmsAbvGrd         1   0.35    0.35    4.9376  0.026435 *
Fireplaces           1   2.71    2.71   37.9886 9.250e-10 ***
GarageCars           1   6.20    6.20   86.7821 < 2.2e-16 ***
```

```
s(WoodDeckSF, 3)      1   0.21    0.21    2.9734  0.084859 .
MoSold                1   0.00    0.00    0.0090  0.924273
Residuals          1422 101.54    0.07
---
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)            2 23.401 1.001e-10 ***
OverallQual
OverallCond
YearBuilt
YearRemodAdd
s(BsmtFinSF1, 3)         2 73.405 < 2.2e-16 ***
s(BsmtFinSF2, 3)         2  2.144   0.11752
s(BsmtUnfSF, 3)          2  3.767   0.02334 *
s(X1stFlrSF, 3)          2 70.405 < 2.2e-16 ***
s(X2ndFlrSF, 3)          2  0.637   0.52881
s(LowQualFinSF, 3)       2  2.701   0.06747 .
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageCars
s(WoodDeckSF, 3)         2  0.997   0.36920
MoSold
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Looking at the summary results we can see if the predictors relationships are linear or non-linear based on significance values. Now we update our model removing the splines from our linear predictors

```
# now we perform cross validation to find the best number of splines for the non-linear predictors



cv.error.best.splines <- rep(0,8)
for(i in 1:8){ # try different numbers of splines
error <- rep(0, 10)

for (k in 1:10){
House.train <- House_train[fold.index != k,]
House.test <- House_train[fold.index == k,]
true.y <- ((House.test[,"SalePrice"]^lambda-1)/lambda)

fits<- gam(((SalePrice^lambda-1)/lambda)~ s(LotArea,i) + OverallQual + OverallCond + YearBuilt + YearRe

pred <- predict(fits, House.test)
error[k] <- mean((pred - true.y)^2)
}
```

```
print(mean(error))
cv.error.best.splines[i] <- mean(error)
}
```

```
[1] 0.1066104
[1] 0.08700601
[1] 0.07687214
[1] 0.07775526
[1] 0.09293656
[1] 0.1215192
[1] 0.1618796
[1] 0.2137267
```

```
#best amount of splines according to cv
which.min(cv.error.best.splines)
```

```
[1] 3
```

How the tuning parameter was chosen

In the code chunk above we calculated the cv test error for each of the degree of freedom options and used the df that gave us the lowest cv test error as our parameter.

```
updated.House.GAM<- gam(((SalePrice^lambda-1)/lambda)~ s(LotArea,3) + OverallQual + OverallCond + YearBu

summary(updated.House.GAM)
```

```
Call: gam(formula = ((SalePrice^lambda - 1)/lambda) ~ s(LotArea, 3) +
    OverallQual + OverallCond + YearBuilt + YearRemodAdd + s(BsmtFinSF1,
    3) + BsmtFinSF2 + s(BsmtUnfSF, 3) + s(X1stFlrSF, 3) + X2ndFlrSF +
    LowQualFinSF + BsmtFullBath + BsmtHalfBath + FullBath + HalfBath +
    KitchenAbvGr + TotRmsAbvGrd + Fireplaces + GarageCars + WoodDeckSF +
    MoSold, data = House_train)
Deviance Residuals:
     Min       1Q   Median       3Q      Max
-2.59649 -0.12711  0.01033  0.14334  1.01939

(Dispersion Parameter for gaussian family taken to be 0.0716)

    Null Deviance: 1003.974 on 1459 degrees of freedom
Residual Deviance: 102.4147 on 1430 degrees of freedom
AIC: 325.8458

Number of Local Scoring Iterations: NA

Anova for Parametric Effects
                  Df Sum Sq Mean Sq   F value    Pr(>F)
s(LotArea, 3)      1  73.10   73.10 1020.6429 < 2.2e-16 ***
OverallQual        1 615.87  615.87 8599.2484 < 2.2e-16 ***
```

```
OverallCond         1   1.32    1.32   18.4562 1.856e-05 ***
YearBuilt           1  30.25   30.25  422.4064 < 2.2e-16 ***
YearRemodAdd        1   3.77    3.77   52.6431 6.535e-13 ***
s(BsmtFinSF1, 3)    1  19.46   19.46  271.6712 < 2.2e-16 ***
BsmtFinSF2          1   1.24    1.24   17.3636 3.272e-05 ***
s(BsmtUnfSF, 3)     1  15.11   15.11  211.0201 < 2.2e-16 ***
s(X1stFlrSF, 3)     1   8.81    8.81  123.0616 < 2.2e-16 ***
X2ndFlrSF           1  63.37   63.37  884.7636 < 2.2e-16 ***
LowQualFinSF        1   0.17    0.17    2.3662  0.124207
BsmtFullBath        1   0.66    0.66    9.2530  0.002394 **
BsmtHalfBath        1   0.00    0.00    0.0166  0.897351
FullBath            1   0.00    0.00    0.0028  0.958085
HalfBath            1   0.61    0.61    8.5099  0.003587 **
KitchenAbvGr        1   2.39    2.39   33.3858 9.261e-09 ***
TotRmsAbvGrd        1   0.33    0.33    4.6294  0.031596 *
Fireplaces          1   2.76    2.76   38.5399 7.015e-10 ***
GarageCars          1   6.14    6.14   85.7611 < 2.2e-16 ***
WoodDeckSF          1   0.18    0.18    2.5197  0.112650
MoSold              1   0.00    0.00    0.0009  0.975646
Residuals        1430 102.41    0.07
---
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)          2 24.689 2.875e-11 ***
OverallQual
OverallCond
YearBuilt
YearRemodAdd
s(BsmtFinSF1, 3)       2 72.667 < 2.2e-16 ***
BsmtFinSF2
s(BsmtUnfSF, 3)        2  3.430   0.03264 *
s(X1stFlrSF, 3)        2 71.424 < 2.2e-16 ***
X2ndFlrSF
LowQualFinSF
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageCars
WoodDeckSF
MoSold
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#predict using this best model
gam.pred<- data.frame(predict(updated.House.GAM, newdata = House_test))

#undo boxcox to understand predictions
```

```
gam.pred<- (gam.pred*lambda+1)^(1/lambda)
head(gam.pred)
```

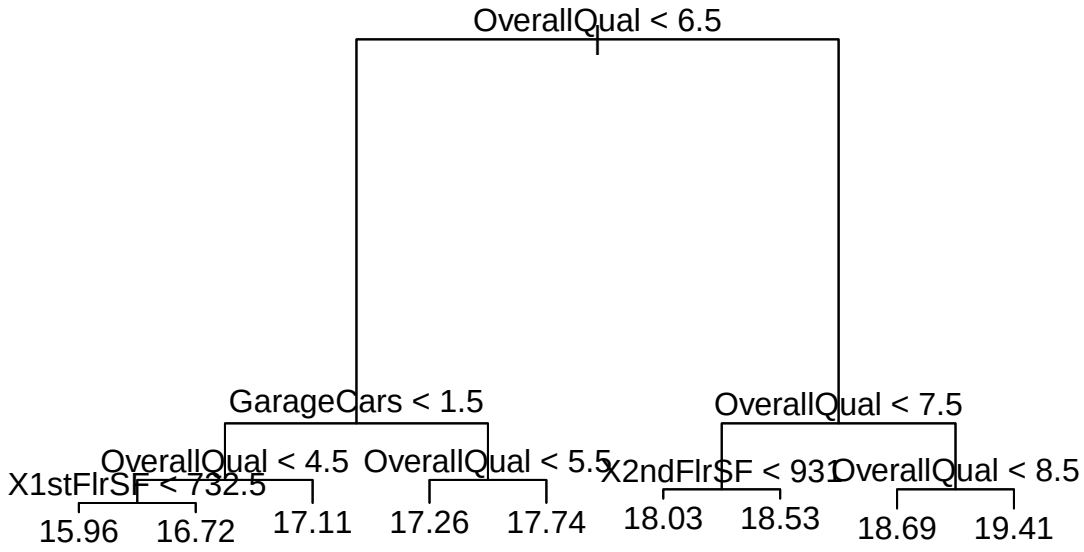
```
  predict.updated.House.GAM..newdata...House_test.
1                                         116113.1
2                                         160375.5
3                                         181136.5
4                                         199484.4
5                                         182744.8
6                                         172249.5
```

Model Interpretation

Our model uses 19 predictors as found in the best model selection we previously ran. Out of the 19 predictors 4 are non-linear, fit with splines that have 3 degrees of freedom.

Regression Trees

```
#make the tree
tree.House <- tree(((SalePrice^lambda-1)/lambda) ~ ., House_train)

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




```
summary(tree.House)
```

```
Regression tree:
tree(formula = ((SalePrice^lambda - 1)/lambda) ~ ., data = House_train)
Variables actually used in tree construction:
[1] "OverallQual" "GarageCars"  "X1stFlrSF"   "X2ndFlrSF"
Number of terminal nodes:  9
Residual mean deviance:  0.1981 = 287.4 / 1451
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.80400 -0.25240  0.02607  0.00000  0.26440  1.55900
```

```
#now we use cross validation to find the optimal tree size
cv.tree.House <- cv.tree(tree.House, FUN = prune.tree, K = 10)

optimal.size <- cv.tree.House$size[which.min(cv.tree.House$dev)]
optimal.size
```

```
[1] 9
```

Since this optimal size is identical to the amount of terminal nodes our tree initially had, we don't need to prune the tree before performing prediction

How the tuning parameter was chosen

In the code chunk above we calculated the cv error for each of the tree sizes and used the tree size that yielded us the lowest error for our tree size of the model.

Model Interpretation

Our tree has a total of 9 terminal nodes with the variables being used in tree construction: "OverallQual" "GarageCars" "X1stFlrSF" "X2ndFlrSF".

```
# make predictions with our tree
tree.House.pred<- data.frame(predict(tree.House,House_test))

#undo boxcox to understand predictions
tree.House.pred<- (tree.House.pred*lambda+1)^(1/lambda)
head(tree.House.pred)
```

```
  predict.tree.House..House_test.
1                        125385.6
2                        125385.6
3                        134935.6
4                        170450.6
5                        268016.0
6                        170450.6
```

Bagging

```
set.seed(10)
bag.House <- randomForest(((SalePrice^lambda-1)/lambda) ~ ., data = House_train,
mtry = ncol(House_train) - 1, importance = TRUE, ntree = 1000)
#see importance in bagging model
importance(bag.House)
```

```
                %IncMSE IncNodePurity
LotArea       53.9873027    37.5067282
OverallQual  124.4198638   591.5164268
OverallCond   36.6910819    15.0064655
YearBuilt     37.0866006    21.6290824
YearRemodAdd  29.3952384    16.0306617
BsmtFinSF1    47.8889883    32.7106357
BsmtFinSF2     7.7613674     1.7263626
BsmtUnfSF     21.5151285    10.3247427
X1stFlrSF     70.5563090    68.2720262
X2ndFlrSF     70.0336558    31.5901252
LowQualFinSF  -0.7486209     0.3215774
BsmtFullBath  12.4584045     1.8334848
BsmtHalfBath   3.1014955     0.4705249
FullBath      32.3529738    14.8606305
HalfBath      23.0610760     2.6183478
```

```
BedroomAbvGr  21.2356756     5.3765477
KitchenAbvGr  12.3681863     1.4071729
TotRmsAbvGrd  23.1003963    10.1560522
Fireplaces    29.7027510    15.6119203
GarageCars    37.2012724    65.0915143
GarageArea    40.2161187    39.2187025
WoodDeckSF    16.2994694     7.6930799
MoSold        -1.3903463     6.7316437
```

Model Interpretation

In our bagging model the predictors that have the most impact are OverallQual being the uncontested first while X1stFlrSF and X2ndFlrSF are nearly tied for second.

How the tuning parameter was chosen

For bagging we must make mtry = the number of columns -1 as this is what makes the selection method bagging. In addition we want to use a larger number for ntree so that every input row gets predicted a few times.

```
#predict for bagging
bag.House.pred<- data.frame(predict(bag.House,House_test))

#undo boxcox to understand predictions
bag.House.pred<- (bag.House.pred*lambda+1)^(1/lambda)
head(bag.House.pred)
```

```
  predict.bag.House..House_test.
1                       129084.2
2                       156679.0
3                       168863.5
4                       179913.9
5                       193277.3
6                       186880.3
```

Random Forest

```
set.seed(10)
randomforest.House <- randomForest(((SalePrice^lambda-1)/lambda) ~ ., data = House_train,
mtry = (ncol(House_train) - 1)/3, importance = TRUE, ntree = 1000)

importance(randomforest.House)
```

```
               %IncMSE IncNodePurity
LotArea      46.0790217    41.1123781
OverallQual  53.7752226   294.8341036
OverallCond  33.3947472    16.2939444
YearBuilt    31.3487974   116.9050330
```

```
YearRemodAdd 24.8569654    34.1331256
BsmtFinSF1   44.4372891    34.9643995
BsmtFinSF2    5.6700286     1.9531366
BsmtUnfSF    21.5828969    12.8745533
X1stFlrSF    50.7335877    81.9467705
X2ndFlrSF    50.3530989    34.8703695
LowQualFinSF -1.8022202     0.4980327
BsmtFullBath 17.9561936     4.0627127
BsmtHalfBath  3.0728409     0.6816807
FullBath     25.5895839    60.1439674
HalfBath     24.4058351     5.3519172
BedroomAbvGr 23.2375192     9.5776136
KitchenAbvGr 12.1286256     2.9756515
TotRmsAbvGrd 27.1924053    21.2221725
Fireplaces   33.6178334    35.7638795
GarageCars   24.2002860    93.1735051
GarageArea   31.3793711    72.0105447
WoodDeckSF   15.2001889     8.2868943
MoSold        0.6527922     7.6301633
```

Model Interpretation

Similar to bagging, in our random forest model the predictors that have the most impact are OverallQual being the uncontested first while X1stFlrSF and X2ndFlrSF are nearly tied for second.

How the tuning parameter was chosen

For randomforest we must make mtry = the number of columns -1 *1/3 as this is what makes the selection method randomforest for regression. In addition we want to use a larger number for ntree so that every input row gets predicted a few times.

```
#random forest predictions
randomforest.House.pred<- data.frame(predict(randomforest.House,House_test))

#undo boxcox to understand predictions
randomforest.House.pred<- (randomforest.House.pred*lambda+1)^(1/lambda)
head(randomforest.House.pred)
```

```
  predict.randomforest.House..House_test.
1                                128577.1
2                                153363.6
3                                176272.8
4                                183201.6
5                                191586.6
6                                187623.7
```

Boosting

```
set.seed(10)
boost.House <- gbm(((SalePrice^lambda-1)/lambda) ~ . , data = House_train, shrinkage = 0.01, n.tree = 5
```

```
Distribution not specified, assuming gaussian ...
```

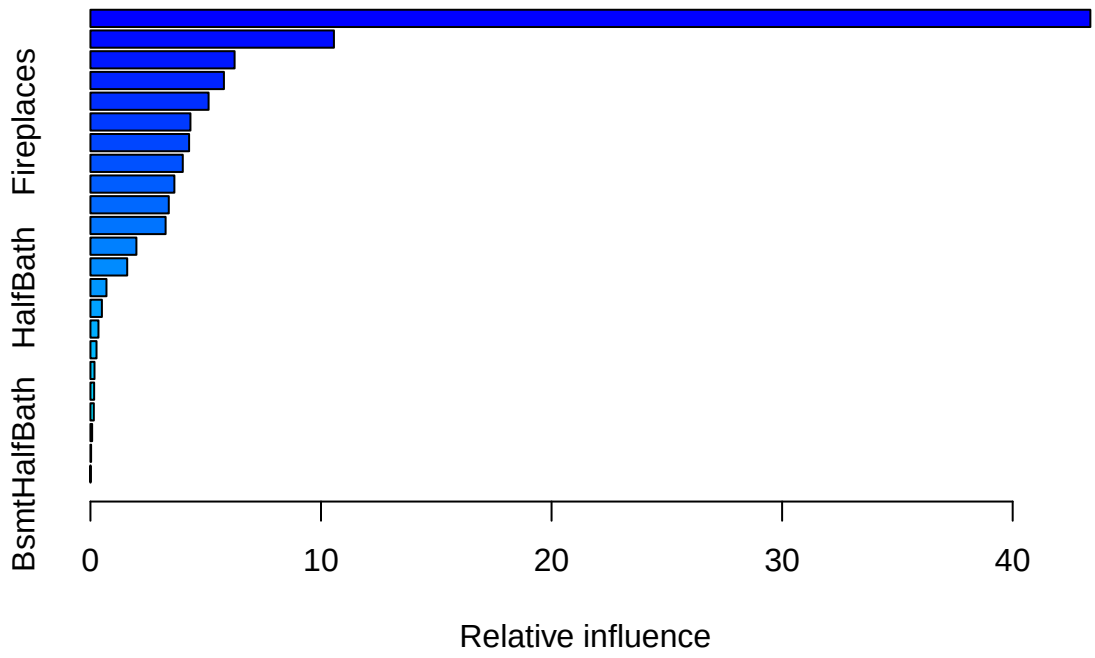
```
#use 10 fold cv to find best amount of trees

which.min(boost.House$cv.error)
```

```
[1] 4996
```

```
# now we use this number of trees for our optimal model
boost.House <- gbm(((SalePrice^lambda-1)/lambda) ~ . , data = House_train, shrinkage = 0.01, n.tree = w
```

```
Distribution not specified, assuming gaussian ...
```

```
summary(boost.House)
```




```
                    var      rel.inf
OverallQual   OverallQual 43.369304242
X1stFlrSF       X1stFlrSF 10.561209366
GarageCars     GarageCars  6.252653019
```

```
LotArea           LotArea  5.789735471
BsmtFinSF1     BsmtFinSF1  5.124771240
Fireplaces     Fireplaces  4.336757370
YearBuilt       YearBuilt  4.278880146
GarageArea     GarageArea  4.005385920
X2ndFlrSF       X2ndFlrSF  3.640804594
FullBath         FullBath  3.398093610
YearRemodAdd YearRemodAdd  3.260383037
OverallCond   OverallCond  1.993689291
TotRmsAbvGrd TotRmsAbvGrd  1.595231844
HalfBath         HalfBath  0.694582144
BsmtUnfSF       BsmtUnfSF  0.497807071
WoodDeckSF     WoodDeckSF  0.347939666
MoSold             MoSold  0.263401355
BsmtFullBath BsmtFullBath  0.175550679
KitchenAbvGr KitchenAbvGr  0.161164175
BsmtFinSF2     BsmtFinSF2  0.146404696
BedroomAbvGr BedroomAbvGr  0.074285659
LowQualFinSF LowQualFinSF  0.028673361
BsmtHalfBath BsmtHalfBath  0.003292043
```

Model Interpretation

In our boosting model OverallQual holds the highest relative influence with a value of 43.369304242 while X1stFlrSF has 10.561209366 and GarageCars has 6.252653019 .

How the tuning parameter was chosen

In the coding chunk above we used 10 fold cross validation to find the best number of trees for ntree. Ending up selecting the number of trees that yielded us the lowest cv test error.

```
#predictions

boost.House.pred<- data.frame(predict(boost.House,House_test))

#undo boxcox to understand predictions
boost.House.pred<- (boost.House.pred*lambda+1)^(1/lambda)
head(boost.House.pred)
```

```
  predict.boost.House..House_test.
1                         125609.8
2                         158803.1
3                         181750.8
4                         195708.4
5                         187813.0
6                         174792.5
```

CV test errors

```
set.seed(10)
#knn
cv.knn <- knn.reg(House_train_no_prices, NULL, House_train$SalePrice, k = 4)
# the little k here is the number of nearest neighbors
knn.cv.House.mse <- mean(cv.knn$residuals^2)
knn.cv.House.mse
```

```
[1] 2320056528
```

```
#lm
cv.lm <- cv.glm(House_train, House_lm_boxcox, K = 10)$delta[1]
cv.lm
```

```
[1] 0.1062286
```

```
#best lm
best.lm.mse<-cv.error.best.fit[which.min(cv.error.best.fit)]
best.lm.mse
```

```
[1] 0.1072685
```

```
#ridge
ridge.cv.mse<- ((cv.ridge$cvup[96]+cv.ridge$cvlo[96])/2)
ridge.cv.mse
```

```
[1] 0.1053981
```

```
#lasso
#take upper and lower estimates of bestlam MSE to find MSE
Lasso.cv.mse<- ((cv.lasso$cvup[49]+cv.lasso$cvlo[49])/2)
Lasso.cv.mse
```

```
[1] 0.1136002
```

```
#GAM
cv.error.best.splines[3]
```

```
[1] 0.07687214
```

```
#Regression Trees
for (k in 1:10){
House.train <- House_train[fold.index != k,]
House.test <- House_train[fold.index == k,]
true.y <- ((House.test[,"SalePrice"]^lambda-1)/lambda)

fits<- tree(((SalePrice^lambda-1)/lambda) ~ ., House.train)

pred <- predict(fits, House.test)
error[k] <- mean((pred - true.y)^2)
}
print(mean(error))
```

```
[1] 0.2150121
```

```
#Bagging
for (k in 1:10){
House.train <- House_train[fold.index != k,]
House.test <- House_train[fold.index == k,]
true.y <- ((House.test[,"SalePrice"]^lambda-1)/lambda)

fits<- randomForest(((SalePrice^lambda-1)/lambda) ~ ., data = House.train,
mtry = ncol(House_train) - 1, importance = TRUE, ntree = 100)

pred <- predict(fits, House.test)
error[k] <- mean((pred - true.y)^2)
}
print(mean(error))
```

```
[1] 0.09525418
```

```
#Random Forest
for (k in 1:10){
House.train <- House_train[fold.index != k,]
House.test <- House_train[fold.index == k,]
true.y <- ((House.test[,"SalePrice"]^lambda-1)/lambda)

fits<-  randomForest(((SalePrice^lambda-1)/lambda) ~ ., data = House.train,
mtry = (ncol(House_train) - 1)/3, importance = TRUE, ntree = 100)

pred <- predict(fits, House.test)
error[k] <- mean((pred - true.y)^2)
}
print(mean(error))
```

```
[1] 0.08772727
```

```
#Boosting
for (k in 1:10){
House.train <- House_train[fold.index != k,]
House.test <- House_train[fold.index == k,]
true.y <- ((House.test[,"SalePrice"]^lambda-1)/lambda)

fits<- gbm(((SalePrice^lambda-1)/lambda) ~ . , data = House.train, shrinkage = 0.01, n.tree = which.min

pred <- predict(fits, House.test)
error[k] <- mean((pred - true.y)^2)
}
```

```
Distribution not specified, assuming gaussian ...
Distribution not specified, assuming gaussian ...
Distribution not specified, assuming gaussian ...
Distribution not specified, assuming gaussian ...
Distribution not specified, assuming gaussian ...
Distribution not specified, assuming gaussian ...
```

```
Distribution not specified, assuming gaussian ...
Distribution not specified, assuming gaussian ...
Distribution not specified, assuming gaussian ...
Distribution not specified, assuming gaussian ...
```

```
print(mean(error))
```

```
[1] 0.08484271
```

As we can see from each methods respective CV error above. Based on these values we would predict that GAM will perform the best.

Comparison of predicted prices with real house prices

After submitting to Kaggle, the method that performed the best was GAM with a log based RMSE of 0.13521, the second best was Boosting with 0.14351 and the methods that performed the worst were KNN with 0.25007 and Regression Tree with 0.23063.

Discussion on why the methods perform the best and the worst

When looking at why our methods performed the way they did, it was no surprise that Regression Tree performed one of the worst. This is because tree based method typically trade accuracy for interpretability. Meaning that while the regression tree is easy to interpret especially for someone not familiar with statistics it has lower prediction accuracy. As for KNN it has difficulty in some regression models and often does better in classification problems. Looking at GAM, which performed the best, this is most likely because implementing the flexible splines allowed our model to better interpret the intricacies that come within this housing data set. Boosting surprisingly did very well showing the large amount of trees we use resulted in a fairly interpretable and accurate model.

Conclusion and summary

To conclude, after running 10 different models using cross validation to select the parameters for each. GAM resulted in our most accurate predictions. Each of these models has it own strengths when it comes to interpretability and accuracy trade offs and could be selected differently when taking in to account ones goals.

Discussion of further questions raised by this study

As noted initially, this housing data was extracted from Ames, Iowa. This makes us question if the factors that were the most impactful in our models here would as be as impactful in other parts of the United States or even the rest of the World. It would be very interesting to see if there are universal things that increase the predicted value of a house no matter the location. When looking to further applications of this data, we could strive to include a much larger scale of data including multiple states housing information, to see if we can apply what we have discovered here more generally.