Predicting Housing Prices - Exploring Regression ML Techniques

Pan Si Cheng Steven, 5355516

June 26, 2020

Introduction

At some point in life, almost everyone will face the decision of whether to invest in a home and ponder over what factors matter in influencing this purchase. This project uses Housing data from the Ames Housing dataset compiled by Dean De Cock available on Kaggle to try and help answer this age-old question.

Given the dataset's wide and diverse coverage on factors that affect the sale price of a residential home, the research question this project seeks to explore is naturally related to predicting sale prices of residential homes. We are interested in developing a predictive supervised learning model to predict housing prices based on the given availability and features of various rooms and amenities in a residential home, and studying how these variables affect the response.

We begin our initial exploration via exploratory graphics and an initial model within the Data and Data Overview sections. After cleaning up our data we then build features for our possible models implementing Random Forest/Lasso Regularized Regression methods and compare their errors before drawing conclusions on a final model to adopt.

Data

We begin our project by loading the data provided from Kaggle and make initial observations on the dataset.

We first begin by finding the dimensions of the training and test sets respectively.

Since our data is verified for use from a live Kaggle competition, it comes with an extra label column 'ID' provided by Kaggle to submit this project for competition, as well as a separate set of testing data without a column for the response variable.

We take note that the training set contains 1460 observations and 81 features whilst the test set contains 1459 observations and 80 features.

```
## [1] 2919   80
```

In addition, we merged our training and datasets after clearing the Kaggle 'ID' column and the 'SalePrice' response variable from the training set to find 2919 observations and 80 features in our combined dataset. (Appendix 1.1)

	MSSubCl…	MSZon…	LotFrontage	LotArea	Street	Alley	LotSha…	LandContour	Utilities
	<int>	<chr>	<int>	<int>	<chr>	<chr>	<chr>	<chr>	<chr>
1	60	RL	65	8450	Pave	*NA*	Reg	Lvl	AllPub

	MSSubCl… <int>	MSZon… <chr>	LotFrontage <int>	LotArea <int>	Street <chr>	Alley <chr>	LotSha… <chr>	LandContour <chr>	Utilities <chr>
2	20	RL	80	9600	Pave	*NA*	Reg	Lvl	AllPub
3	60	RL	68	11250	Pave	*NA*	IR1	Lvl	AllPub
4	70	RL	60	9550	Pave	*NA*	IR1	Lvl	AllPub
5	60	RL	84	14260	Pave	*NA*	IR1	Lvl	AllPub
6	50	RL	85	14115	Pave	*NA*	IR1	Lvl	AllPub

6 rows | 1-10 of 81 columns

Initially exploring the dataset, we find 80 variables in the data generalized into 4 categories; '# of Rooms', 'Square Footage', 'Value' and 'Ratings':

- There are numeric '# of Rooms' variables that count the number of each type of room and various amenities avaiable within each residential property.
- There are numeric 'Square Footage' variables that provide specifics on the size of these various rooms and amenities.
- There are numeric 'Value' variables that provide the various prices paid for amenities within the property and of the property itself.
- There are character 'Ratings' variables that provide qualities and ratings of the overall property, its amenities, neighborhood, zoning.
- There are character Categorical variables that represent conditions and provide details of amenities and of the residential property.
- There remain additional misc. numerical variables that are used as time markers or factors (categorical by nature).
- In total, there are 37 numerical variables and 43 character variables.

From studying the dataset, we note a list of general recurrence of the type of amenities/rooms/ratings within the data such as: Neighbourhood, Garage, Bathrooms, Pools, etc…. Thus, it seems wise to consider implementing feature engineering for this wide variety of variables via the type of room/amenity they are generally sorted into after cleaning up the data to form a single strong predictor variable and study its correlation.

Finally, before we begin data cleaning and preprocessing we will explore our response variable.

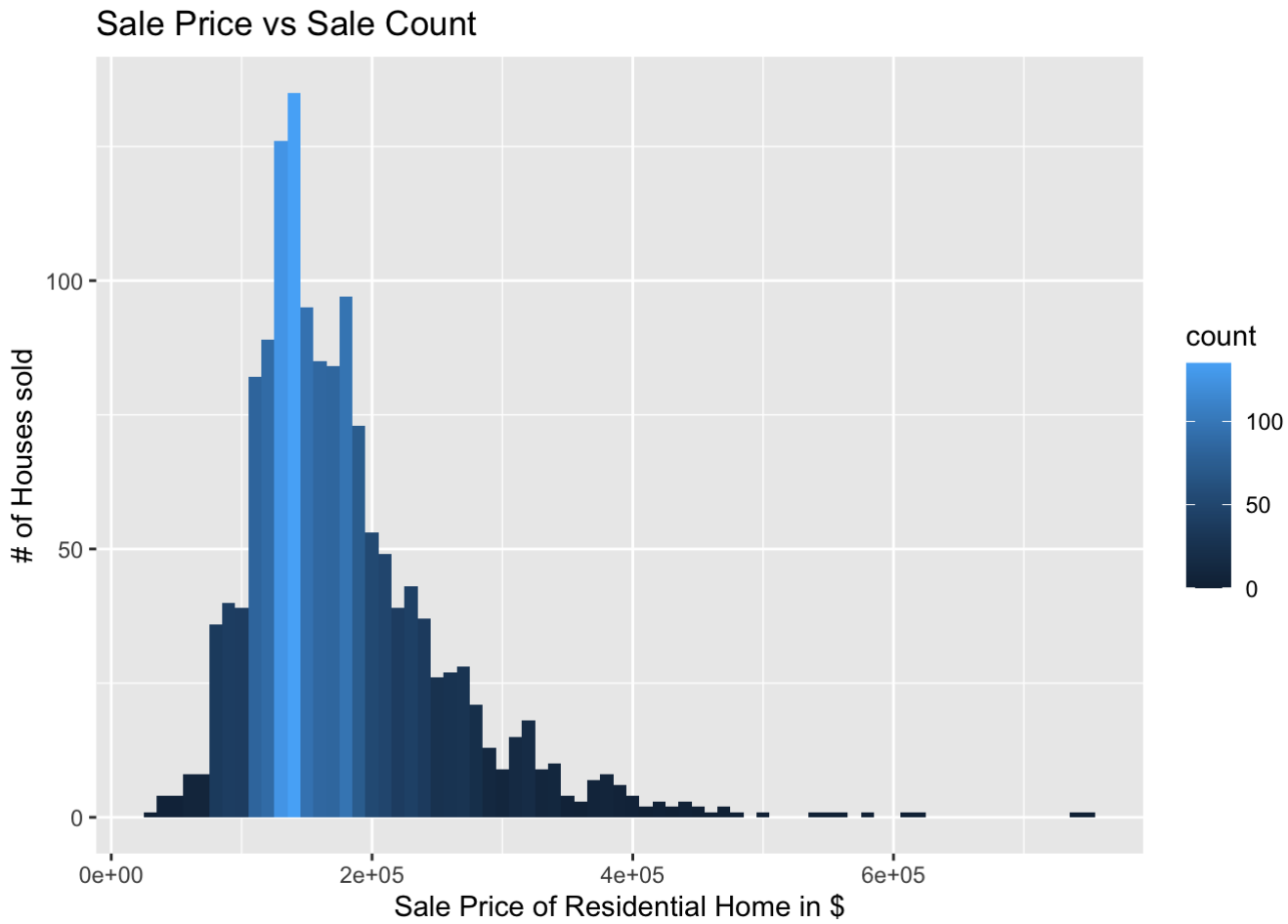
The response variable we are interested in is the numeric variables 'SalePrice', and it measures the Sale Price of a House within the dataset.

```
summary(train$SalePrice)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   34900  129975  163000  180921  214000  755000
```

From a quick glance, we see that the lowest sale price for a property to be 34,900 whilst the highest sale price to be 755,000.

Plotting our response variable to see how housing prices are distributed across the all homes in the dataset:



Histogram of Response Variable (Appendix 1.2)

From the graph above, we find the sale prices from our dataset to be right skewed. We log transform our response variable and explore further.

Histogram of Log Transformed Response Variable (Appendix 1.2)

We find an approximately normally distributed graph and decide to log transform our numeric variables to address skew issues when we begin data pre-processing.

Now we take a look at the missing values within the dataset:

```
##  Exterior1st  Exterior2nd   BsmtFinSF1   BsmtFinSF2    BsmtUnfSF
##            1            1            1            1            1
##  TotalBsmtSF   Electrical  KitchenQual   GarageCars   GarageArea
##            1            1            1            1            1
##     SaleType    Utilities BsmtFullBath BsmtHalfBath   Functional
##            1            2            2            2            2
##     MSZoning   MasVnrArea   MasVnrType BsmtFinType1 BsmtFinType2
##            4           23           24           79           80
##     BsmtQual     BsmtCond BsmtExposure   GarageType  GarageYrBlt
##           81           82           82          157          159
## GarageFinish   GarageQual   GarageCond  LotFrontage  FireplaceQu
##          159          159          159          486         1420
##    SalePrice        Fence        Alley  MiscFeature       PoolQC
##         1459         2348         2721         2814         2909
```

```
## [1] 35
```

Further exploration of our data shows us that a large number (35) of variables contain missing values. We decide to continue with the dataset as is to clean our data for feature building. (Appendix 1.4)

Methods

We begin by cleaning and processing our data based on type. We focus on imputing all the missing values and coerce categorical variables into ordinal variables or factorse. After processing our character variables, we build numerical features that appear obviously correlated via room type or other details. We normalize our numerical data to account for the right skew we plotted earlier, and hot encode our character variables to prepare for model training. We filter out unbalanced predicrtors to produce a final trimmed dataset to build our working data sets.

After implementing our training, test, and validation sets, we train our Random Forest, Ridge, and Lasso regression models against the validation set to find CV test errors for our models. We compare these results and apply them in our model building section.

1. Cleaning & Processing Data:

We know there are 35 features that contain missing values and begin cleaning the data to account for their missing values by type of amenity/room.

- For our numeric '# of Rooms' and 'Square Footage' variables, we know that 'NA' values mean the house does not have that specified amenity and can begin replacing them with 0 values to signify the lack of it within the house. We similarly conduct this replacement on any categorical character variables that indicate the presence of an amenity and replace the 'NA' values with 'None'.
- For our character 'Ratings' variables, we process their custom ratings into an ordered vector and assign them as ordinal labels for the variable. We assign ordinal labels as the case of order within our qualities matter, but not the specific number used to represent them.
- For our categorical character variables that provide general classification details, we process them as factored variables and replace any relevant missing data with their respective modes to complete our data. Order in this case does not matter as they simply describe a detail of the property.

As such, we begin by clearing missing values and processing our variables for model building.

The code provided below is a small portion of a large section of the code used to clean and process our data. It is included to match up with the explanations above on how we decided to clean our data. (Appendix 2.1)

```
#Replacing 'NA' values within our numeric '# of Rooms' and 'Square Footage' variables
 (e.g. Basement Full Bathrooms, Basement Square Footage)
full$BsmtFullBath[is.na(full$BsmtFullBath)] <- 0
full$BsmtFinSF1[is.na(full$BsmtFinSF1)] <- 0

#Replacing 'NA' values within categorical character variables that indicate the presence
of an amenity (e.g. Alley, Fence)
full$Alley[is.na(full$Alley)] <- 'None'
full$Fence[is.na(full$Fence)] <- 'None'

#Replacing 'NA' values within our character 'Ratings' variables with 'None' (e.g.Garage)
full$GarageQual[is.na(full$GarageQual)] <- 'None'
full$GarageCond[is.na(full$GarageCond)] <- 'None'
full$GarageFinish[is.na(full$GarageFinish)] <- 'None'

#Processing custom ratings into an vector to assign ordered labels and convert into a nu
meric ordinal variable.
Qualities = c('None' = 0, 'Po' = 1, 'Fa' = 2, 'TA' = 3, 'Gd' = 4, 'Ex' = 5)
Finishes = c('None' = 0, 'Unf' = 1, 'RFn' = 2, 'Fin' = 3)

full <- full %>%
  mutate('GarageQual'=as.integer(revalue(full$GarageQual, Qualities))) %>%
  mutate('GarageCond'=as.integer(revalue(full$GarageCond, Qualities))) %>%
  mutate('GarageFinish'= as.integer(revalue(full$GarageFinish, Finishes)))

#Replacing 'NA' values within our categorical chracter variables with their mode (e.g. Z
oning, Exteriors)
full$MSZoning[is.na(full$MSZoning)] <- mode(full$MSZoning)
full$Exterior1st[is.na(full$Exterior1st)] <- mode(full$Exterior1st)
full$Exterior2nd[is.na(full$Exterior2nd)] <- mode(full$Exterior2nd)

#Factorizing categorical variables for model building later on
full <- full %>%
  mutate(Alley = as.factor(full$Alley)) %>%
  mutate(Fence = as.factor(full$Fence)) %>%
  mutate(MSZoning = as.factor(full$MSZoning)) %>%
  mutate(Exterior1st = as.factor(full$Exterior1st)) %>%
  mutate(Exterior2nd = as.factor(full$Exterior2nd))
```

After replacing all our missing values, we must now process the remaining variables and coerce the character variables into factors in order to implement our ML methods.

```
##  [1] "Street"        "LandContour"   "LandSlope"     "Neighborhood"
##  [5] "Condition1"    "Condition2"    "BldgType"      "HouseStyle"
##  [9] "RoofStyle"     "RoofMatl"      "Foundation"    "Heating"
## [13] "HeatingQC"     "CentralAir"    "GarageFinish"  "GarageQual"
## [17] "GarageCond"    "PavedDrive"    "SaleCondition"
```

Verifying there are 19 character variables in the dataset, we coerce them into factors as we did for our previous imputed variables to implement them in our methods. We also note the existence of 3 numeric variables ‘MoSold’, ‘YrSold’, and ‘MSSubClass’ that have categorical values and coerce them into factors as well. (Appendix 2.3)

2. Feature Creation

After familiarizing ourselves with the large number of variables through conducting initial analysis and cleanup, we revisit a point brought up earlier regarding many variables corresponding to similar rooms or similar data types. We decide to create features based on these sub-groups and similarities to consolidate our variables:

- Number of Room/Amenity Type (Bathrooms)
- Neighborhoods (By determining economic class through median sale prices)
- Total Square Footage of living areas (Above ground, basement, porch)

Total Square Footage of Living Areas

We have two variables 'GrLivArea' and 'TotalBsmtSF' that represent the total above ground and below ground square footage of a residential home. Thus, we begin by engineering a new feature 'TotalSF' that represents the sum of living areas within a residential home. (Appendix 2.4)

Bathrooms Binned

We notice we have four variables 'FullBath', 'HalfBath', 'BsmtHalfBath', 'BsmtFullBath' that measure the number of bathrooms/half bathrooms in a house. Since these are similar we decide to group them as a single count of total number of bathrooms in a house. We sum these four variables to create a new feature 'numBR'. (Appendix 2.4)

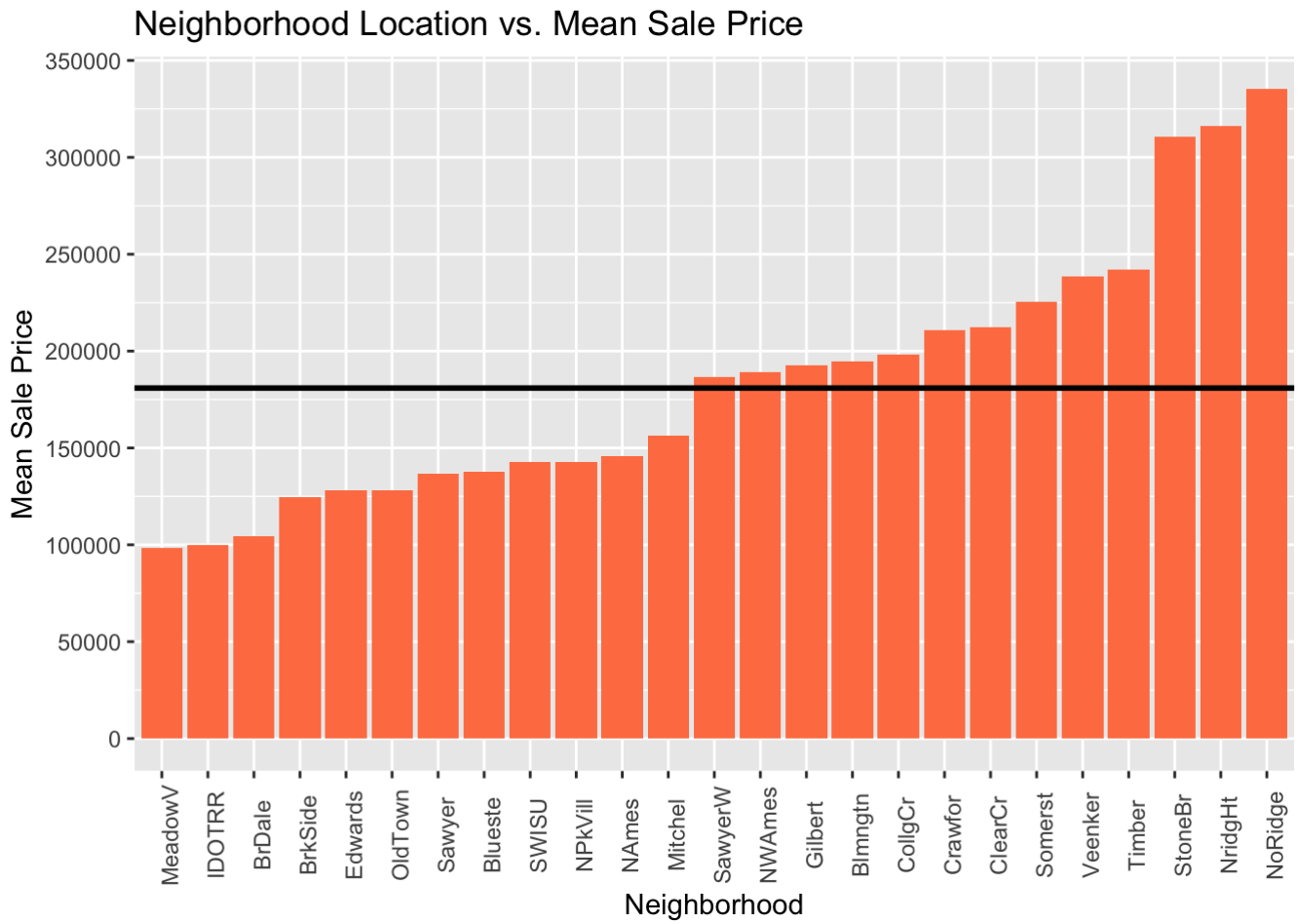
Porches Binned

We notice that there are four variables 'OpenPorchSF', 'EnclosedPorch', 'X3SsnPorch', 'ScreenPorch' that measure the square footage of various porches of a house an decide to similarly bin them as a single value of total square footage of a houses porch. We sum these variables to create a new feature 'TotalPorchSF'. (Appendix 2.4)

Neighborhoods Binned

We notice that there are 25 different possible values for 'Neighborhood'. To find an orderly way of categorizing the various neighborhoods, we plot the mean sale prices of homes against each neighborhood, and plot the mean horizontal line across the barplot. We choose to have 0 represent the lower half of the neighborhoods mean sale prices whilst 1 the upper half of the mean sale prices plot. We create this feature 'NBClass' and assign the respective values. (Appendix 2.5)

```
## No summary function supplied, defaulting to `mean_se()`
```



After creating our features from the numerical variables, we must address possible colinearity issues in our dataset from feature engineering and remove one variable used in each of the features (GrLivArea,FullBath,OpenPorchSF,Neighborhood) built from the main dataset. (Appendix 2.6)

Finally, we process and transform our “true”/original numerical variables. We know from graphing the response variable in our initial exploration that the data is right skewed. After log transforming the dataset, we found a graph of the response that followed the normal distribution much closer. We decide to normalize our true numeric variables in our dataset before proceeding with model building.

Since we know a variable’s skew ranges from -1 to 1 with 0 skew being a normally distributed variable, we log transform our variables if their absolute skew is greater than 0.8 to normalize our data. (Appendix 2.7)

```
#Numeric variables other than Sale Price
numerical <- full[,names(full) %in% truenumericalnames]
skew <- apply(numerical, 2, skewness)
skew <- skew[(skew > 0.8) | (skew < -0.8)]

for(col in names(skew)){
    numerical[,col] <- log(numerical[,col]+1)
  }
```

Now that we’ve constructed our new features from the myriad of variables provided in the dataset and cleaned up our numerical variables, we must create dummy binary variables for our multi leveled categorical features in order to implement them in the models later on. We accomplish this via caret’s dummyVars function where we hot one encode our categorical features. Finally, we bind our variables together to form a complete preprocessed data set to begin building training and test set. (Appendix 2.8)

```
# use caret dummyVars function for hot one encoding for categorical features, automatica
lly sorts through categorical and numeric
dummies <- dummyVars("~ .", data = full[,cnames])
categorical <- data.frame(predict(dummies,newdata=full[,cnames]))

full<- cbind(normalized,categorical)
dim(full)
```

```
## [1] 2919  245
```

We find our new combined dataset to have 2919 observations of 245 variables.

Since we're going to train our models with this dataset, we must also consider reducing the variance and bias within to prevent a high MSE. We use caret's built in Near Zero Variance function to remove any unbalanced predictors which have near zero variance and high frequency. (Appendix 2.8)

```
#Find all the predictors with TRUE 'nzv' and store their names into a vector
zerovarpredictors <- nearZeroVar(full, saveMetrics = TRUE)
filtered <- rownames(zerovarpredictors)[zerovarpredictors$nzv == TRUE]

full <- full[,!names(full) %in% filtered]
dim(full)
```

```
## [1] 2919  106
```

We find our final dataset to contain 2919 observations of 106 variables.

3. Training/Testing Set:

Finally, we split our now organized data back into its training and test sets as predetermined from Kaggle. The full training set contains 1460 observations of 106 variables. The test set contains 1459 observations of 106 variables. We create a separate validation set using 40% of our training set data with 582 observations of 106 variables, leaving a working training set with 878 observations of 106 variables. (Appendix 2.9)

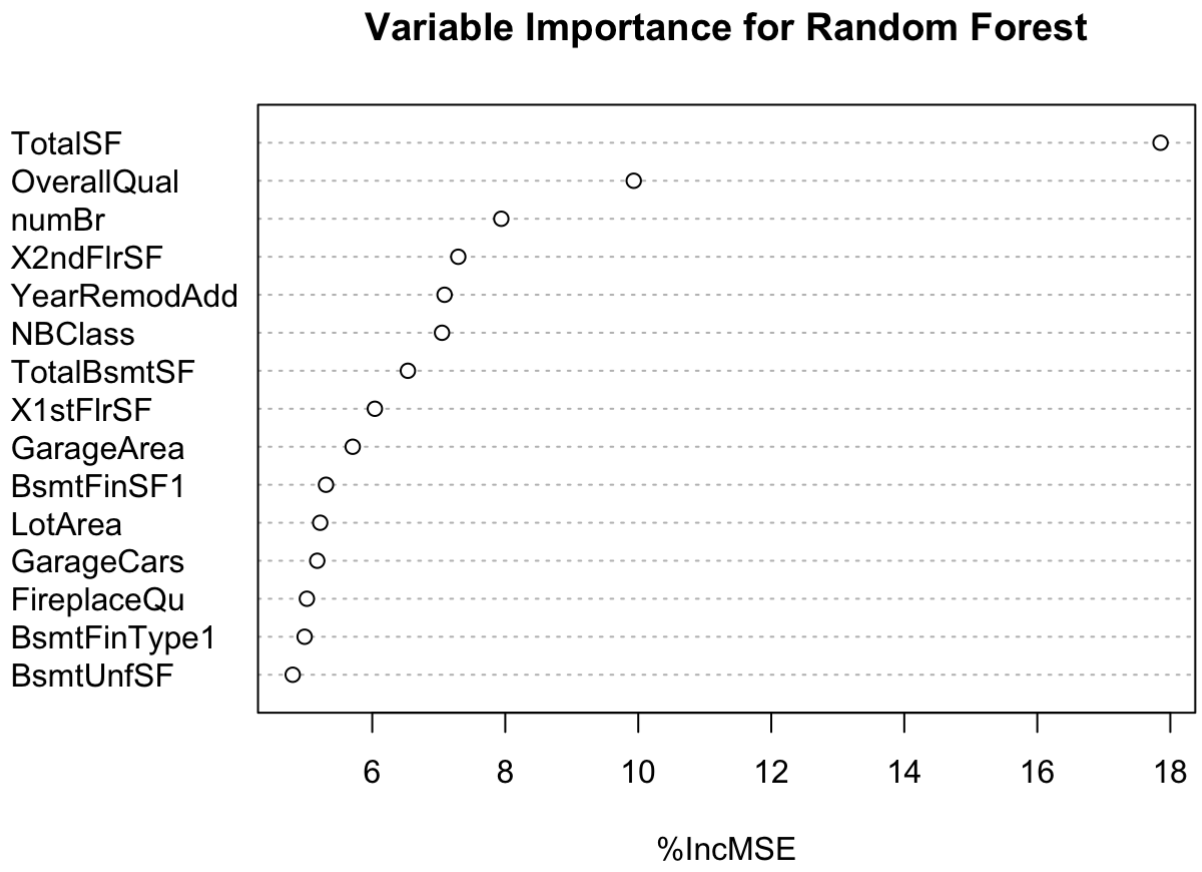
```
set.seed(369)
full.train <- full[1:1460, ]
full.test <- full[1461:2919,]

# We log transform our response variable since it is also a skewed numerical variable.
full.train$SalePrice = log(full.train$SalePrice+1)

#40% CV Set
part <- createDataPartition(y = full.train$SalePrice, p = 0.6, list = FALSE)
working.train <- full.train[part,]
validation.train <- full.train[-part,]
```

4. Random Forest:

We begin by training our Random Forest model on the test set and graph its variable importance plot to interpret the results.

```
set.seed(369)
rf.train = randomForest(x = working.train[,-106], y = working.train$SalePrice, ntree=100
, importance=TRUE)
rf.train
```

```
##
## Call:
##  randomForest(x = working.train[, -106], y = working.train$SalePrice,      ntree = 10
0, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 35
##
##           Mean of squared residuals: 0.01994836
##                     % Var explained: 87.75
```




```
yhat.rf = predict(rf.train, newdata = validation.train)
mean((yhat.rf-validation.train$SalePrice)^2)
```

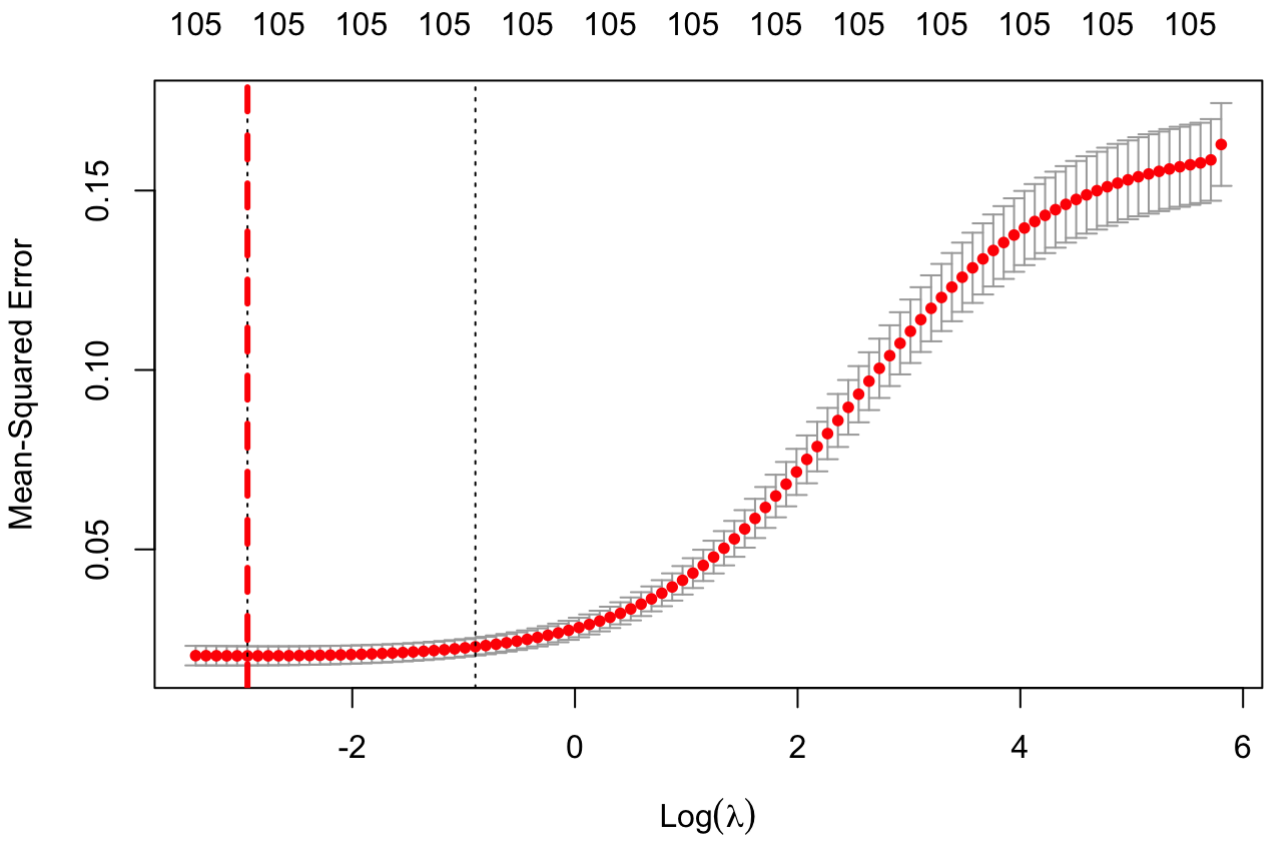
```
## [1] 0.01808418
```

Using the working training set we created, we find an MSE of 0.01995. We also find the features ‘TotalSF’, ‘OverallQual’, and ‘numBR’ to be the most important and that our model has a CV testing error of .0181

5. Ridge Regression

Now, we implement Ridge Regression using the working training set and use cross-validation to select our best tuning parameter lambda.

```
set.seed(120)

cv.ridge = cv.glmnet(as.matrix(working.train[,-106]), working.train$SalePrice, alpha = 0
)
```




```
bestlam.ridge = cv.ridge$lambda.min
bestlam.ridge
```

```
## [1] 0.05280697
```

After finding our best lambda = 0.0528, we retrain our Ridge regression model with s = 0.0528 to calculate our CV test error.

```
## [1] 0.01709381
```

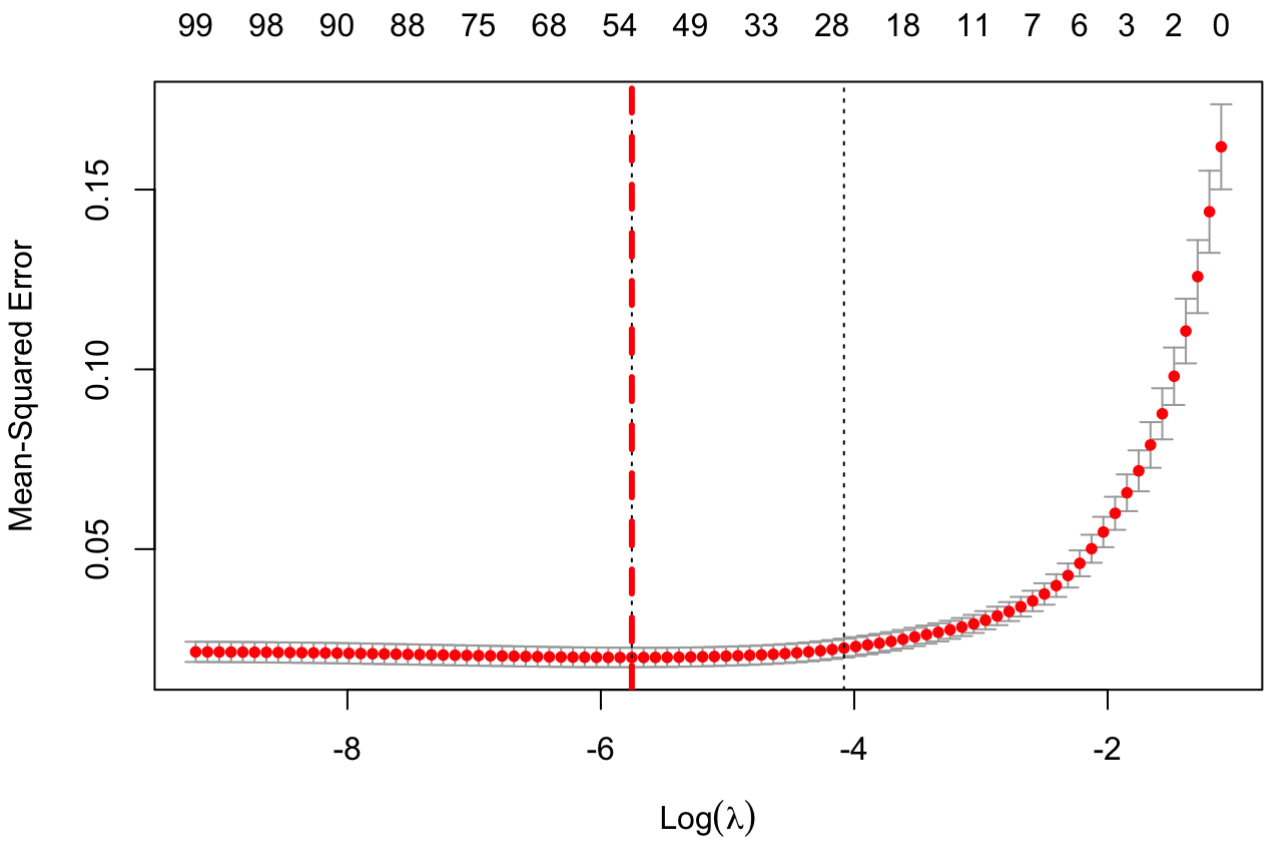
We find that our ridge regression model has a CV testing error of .0171

(Appendix 2.11)

6. Lasso Regression:

Now, we implement Lasso Regression using the working training set and use cross-validation to select our best tuning parameter lambda.

```
set.seed(120)
cv.lasso = cv.glmnet(as.matrix(working.train[,-106]), working.train$SalePrice, alpha = 1
)
```




```
bestlam.lasso = cv.lasso$lambda.min
bestlam.lasso
```

```
## [1] 0.003165695
```

After finding our best lambda = 0.00317, we retrain our Lasso regression model with s = 0.00317 to calculate our CV test error.

```
## [1] 0.01581656
```

We find that our lasso regression model has a CV testing error of .0158

(Appendix 2.12)

Since all 3 models display similar CV test errors, we decide to adopt all 3 and retrain our models with the full testing set to see if we can find further nuance.

Model Building

We now retrain all 3 of our models on the true test set. Since we previously log transformed our response variable to address right skew, we reverse the log transformation on our predicted values before calculating our true test error. (Appendix 3.1)

1. Final Random Forest

```
set.seed(120)

predict <- as.data.frame(full.test)

rf.train = randomForest(x = full.train[,-106], y = full.train$SalePrice, ntree=100, impo
rtance=TRUE)
yhat.rf = predict(rf.train, newdata = full.test[,-106])
yhat.rf = as.double(exp(yhat.rf) - 1)
```

2. Final Ridge Regression

```
set.seed(120)

cv.ridge = cv.glmnet(as.matrix(full.train[,-106]), full.train$SalePrice, alpha = 0)
bestlam.ridge = cv.ridge$lambda.min
yhat.ridge = as.double(predict(cv.ridge, s = bestlam.ridge, newx = as.matrix(full.test[,
-106])))
yhat.ridge = as.double(exp(yhat.ridge)-1)
```

3. Final Lasso Regression

```
set.seed(120)

cv.lasso = cv.glmnet(as.matrix(full.train[,-106]), full.train$SalePrice, alpha = 1)
bestlam.lasso = cv.lasso$lambda.min
yhat.lasso = as.double(predict(cv.lasso, s = bestlam.lasso, newx = as.matrix(full.test[,
-106])))
yhat.lasso = as.double(exp(yhat.lasso)-1)
```

4. Error Testing

We now compute the true test error rate of our various models.

Since this Dataset comes from Kaggle, the following code calculates the true test error rate if we had created the test set from a complete dataset including the response variable).The true test error rate is computed by submitting a submission to Kaggle, an example of my submission code is provided in the .RMD but will be suppressed)

Predicting on the test set, we find the following error rates for our models:

- Random Forest = 13.725%
- Ridge Regression = 13.077%

- Lasso Regression = 12.954%

(Appendix 3.2)

Since all 3 test error rates are similar, we decide to combine all 3 models' predictions and take their average to predict the response variable.

```
#Combined Model error (mean commented out so Rmd can knit)
combined.pred <- (yhat.rf + yhat.ridge + yhat.lasso) / 3.0
#mean((lasso.pred-full.test$SalePrice)^2)
```

We find that when we train our combined model on the full set we yield an error rate to be 12.737%. (Appendix 3.3)

Conclusion

We choose our final model to be the combined model averaging the predicted values that come from Random Forest, Ridge Regression, and Lasso Regression. We find the performance of our final model to be the best, with a test error rate of 12.737%. Possible ways of improving our final model could relate to building more features that capture unaddressed interactions, or with a further implementation of a Boosting model to join the combined average. Furthermore, we could consider manually removing low variance variables instead of implementing caret's near zero variance function or alter the parameters within to reduce the amount of predictors removed from the final dataset.

Since this project comes from an ongoing Kaggle competition, my future research direction will likely revolve around implementing these methods and also considering other ways of weighing our combined model to improve my final score (Rank 1599, .12737, Top 30%.)

Appendix

Data Overview:

(1.1 Data Loading)

```
train <- read.csv('train.csv', header = TRUE, stringsAsFactors = F)
test <- read.csv('test.csv', header = TRUE, stringsAsFactors = F)

dim(train)
```

```
## [1] 1460   81
```

```
dim(test)
```

```
## [1] 1459   80
```

```
test$Id <- NULL
train$Id <- NULL
test$SalePrice <- NA

full <- rbind(train, test)
dim(full)
```

```
## [1] 2919   80
```

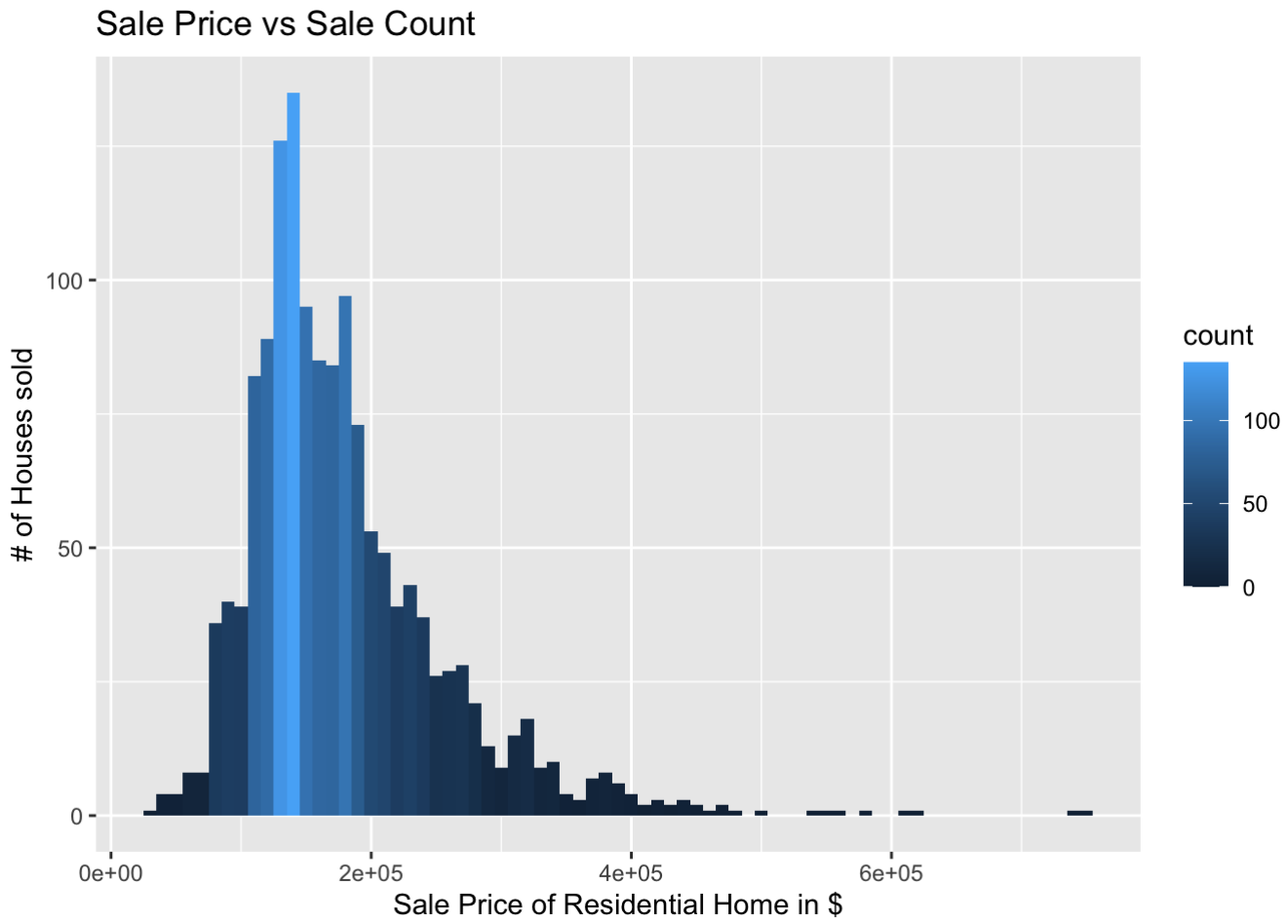
```
#Used later on to preprocess data, removed false numericals and also removed variables u
sed in building features later on
truenumerical <-  select_if(full, is.numeric)
truenumericalnames <- names(truenumerical)
length(truenumericalnames) #This is the length used to find # numerical variables = 37
```

```
## [1] 37
```

```
truenumerical <- subset(truenumerical, select = -c(GrLivArea,FullBath,OpenPorchSF,MSSubC
lass,MoSold,YrSold,OverallQual,OverallCond,SalePrice))
truenumericalnames <- names(truenumerical)
```

(1.2) Plotting Response Variable

```
#Plotting response variable vs. count
ggplot(data = train, aes(x=SalePrice, fill = ..count..)) +
  geom_histogram(binwidth = 10000) +
  ggtitle('Sale Price vs Sale Count') +
  xlab('Sale Price of Residential Home in $')  +
  ylab('# of Houses sold')
```




```
#Plotting log response variable vs. count

train$logSalePrice <- log(train$SalePrice)
ggplot(data = train, aes(x=logSalePrice, fill = ..count..)) +
  geom_histogram(binwidth = 0.1) +
  ggtitle('Log Sale Price vs Sale Count') +
  xlab('Log Sale Price of Residential Home') +
  ylab('# of Houses sold')
```

(1.3) Initial Exploratory Model

```
#Initial exploratory model using log transformed response variable
lm.fit = lm(logSalePrice ~ OverallQual + GrLivArea + GarageCars, data = train)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = logSalePrice ~ OverallQual + GrLivArea + GarageCars,
##     data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.54542 -0.09069  0.01367  0.11488  0.66654
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.058e+01  2.248e-02  470.64   <2e-16 ***
## OverallQual 1.434e-01  4.977e-03   28.81   <2e-16 ***
## GrLivArea   2.216e-04  1.185e-05   18.70   <2e-16 ***
## GarageCars  1.316e-01  8.389e-03   15.69   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1886 on 1456 degrees of freedom
## Multiple R-squared:  0.7776, Adjusted R-squared:  0.7771
## F-statistic:  1697 on 3 and 1456 DF,  p-value: < 2.2e-16
```

(1.4) Missing Data

```
missing <- which(colSums(is.na(full)) > 0)
sort(colSums(sapply(full[missing], is.na)))
```

```
##  Exterior1st  Exterior2nd   BsmtFinSF1   BsmtFinSF2    BsmtUnfSF
##            1            1            1            1            1
##  TotalBsmtSF   Electrical  KitchenQual   GarageCars   GarageArea
##            1            1            1            1            1
##     SaleType    Utilities BsmtFullBath BsmtHalfBath   Functional
##            1            2            2            2            2
##     MSZoning   MasVnrArea   MasVnrType BsmtFinType1 BsmtFinType2
##            4           23           24           79           80
##     BsmtQual     BsmtCond BsmtExposure   GarageType  GarageYrBlt
##           81           82           82          157          159
## GarageFinish   GarageQual   GarageCond  LotFrontage  FireplaceQu
##          159          159          159          486         1420
##    SalePrice        Fence        Alley  MiscFeature       PoolQC
##         1459         2348         2721         2814         2909
```

```
length(missing)
```

```
## [1] 35
```

Methods

(2.1) Cleaning up Variables

```
#Replacing 'NA' values within our numeric '# of Rooms' and 'Square Footage' variables
 (e.g. Basement Full Bathrooms, Basement Square Footage)
full$BsmtFullBath[is.na(full$BsmtFullBath)] <- 0
full$BsmtFinSF1[is.na(full$BsmtFinSF1)] <- 0

#Replacing 'NA' values within categorical character variables that indicate the presence
of an amenity (e.g. Alley, Fence)
full$Alley[is.na(full$Alley)] <- 'None'
full$Fence[is.na(full$Fence)] <- 'None'

#Replacing 'NA' values within our character 'Ratings' variables with 'None' (e.g.Garage)
full$GarageQual[is.na(full$GarageQual)] <- 'None'
full$GarageCond[is.na(full$GarageCond)] <- 'None'
full$GarageFinish[is.na(full$GarageFinish)] <- 'None'

#Processing custom ratings into an vector to assign ordered labels and convert into a nu
meric ordinal variable.
Qualities = c('None' = 0, 'Po' = 1, 'Fa' = 2, 'TA' = 3, 'Gd' = 4, 'Ex' = 5)
Finishes = c('None' = 0, 'Unf' = 1, 'RFn' = 2, 'Fin' = 3)

full <- full %>%
  mutate('GarageQual'=as.integer(revalue(full$GarageQual, Qualities))) %>%
  mutate('GarageCond'=as.integer(revalue(full$GarageCond, Qualities))) %>%
  mutate('GarageFinish'= as.integer(revalue(full$GarageFinish, Finishes)))

#Replacing 'NA' values within our categorical chracter variables with their mode (e.g. Z
oning, Exteriors)
full$MSZoning[is.na(full$MSZoning)] <- mode(full$MSZoning)
full$Exterior1st[is.na(full$Exterior1st)] <- mode(full$Exterior1st)
full$Exterior2nd[is.na(full$Exterior2nd)] <- mode(full$Exterior2nd)

#Factorizing categorical variables for model building later on
full <- full %>%
  mutate(Alley = as.factor(full$Alley)) %>%
  mutate(Fence = as.factor(full$Fence)) %>%
  mutate(MSZoning = as.factor(full$MSZoning)) %>%
  mutate(Exterior1st = as.factor(full$Exterior1st)) %>%
  mutate(Exterior2nd = as.factor(full$Exterior2nd))
```

```
#Rest of cleanup data with output suppressed

#We note from our missing values that the 'NA' values in the garage variables do not cor
respond in the case of House 2127 and 2577. The following code resolves these difference
s. (Found via comparing the 157 NA's in GarageType with 159 NA's in the other numerical
 variables)

full$GarageCond[2127] <- names(sort(-table(full$GarageCond)))[1]
full$GarageQual[2127] <- names(sort(-table(full$GarageQual)))[1]
full$GarageFinish[2127] <- names(sort(-table(full$GarageFinish)))[1]
full$GarageCars[2577] <- 0
full$GarageArea[2577] <- 0
full$GarageType[2577] <- NA


#Ratings based ordinal variables
full$PoolQC[is.na(full$PoolQC)] <- 'None'
full$FireplaceQu[is.na(full$FireplaceQu)] <- 'None'
full$BsmtQual[is.na(full$BsmtQual)] <- 'None'
full$BsmtCond[is.na(full$BsmtCond)] <- 'None'
full$KitchenQual[is.na(full$KitchenQual)] <- 'TA'

full <- full %>%
  mutate('PoolQC' = as.integer(revalue(full$PoolQC, Qualities))) %>%
  mutate(FireplaceQu = as.integer(revalue(full$FireplaceQu, Qualities))) %>%
  mutate(BsmtQual = as.integer(revalue(full$BsmtQual, Qualities))) %>%
  mutate(BsmtCond = as.integer(revalue(full$BsmtCond, Qualities))) %>%
  mutate(KitchenQual = as.integer(revalue(full$KitchenQual, Qualities))) %>%
  mutate(ExterQual = as.integer(revalue(full$ExterQual, Qualities))) %>%
  mutate(ExterCond = as.integer(revalue(full$ExterCond, Qualities)))
```

```
## The following `from` values were not present in `x`: Po, TA
```

```
## The following `from` values were not present in `x`: Po
```

```
## The following `from` values were not present in `x`: Ex
```

```
## The following `from` values were not present in `x`: None, Po
## The following `from` values were not present in `x`: None, Po
```

```
## The following `from` values were not present in `x`: None
```

```
#Factored Categorical variables
full$GarageType[is.na(full$GarageType)] <- 'None'
full$MiscFeature[is.na(full$MiscFeature)] <- 'None'
full$Electrical[is.na(full$Electrical)] <- mode(full$Electrical)
full$SaleType[is.na(full$SaleType)] <- mode(full$SaleType)
full$Utilities[is.na(full$Utilities)] <- mode(full$Utilities)

full <- full %>%
  mutate(MiscFeature = as.factor(full$MiscFeature)) %>%
  mutate(Electrical = as.factor(full$Electrical)) %>%
  mutate(SaleType = as.factor(full$SaleType)) %>%
  mutate(LotConfig = as.factor(full$LotConfig)) %>%
  mutate(Utilities = as.factor(full$Utilities)) %>%
  mutate(GarageType = as.factor(full$GarageType))

#BsmtExposure
exposure <- c('None'=0, 'No'=1, 'Mn'=2, 'Av'=3, 'Gd'=4)
full$BsmtExposure[is.na(full$BsmtExposure)] <- 'None'
full <- full %>%
 mutate(BsmtExposure = as.integer(revalue(full$BsmtExposure, exposure)))

#BSmtFintype1 and 2
finish <- c('None'=0, 'Unf'=1, 'LwQ'=2, 'Rec'=3, 'BLQ'=4, 'ALQ'=5, 'GLQ'=6)
full$BsmtFinType1[is.na(full$BsmtFinType1)] <- 'None'
full$BsmtFinType2[is.na(full$BsmtFinType2)] <- 'None'
full <- full %>%
 mutate(BsmtFinType1 = as.integer(revalue(full$BsmtFinType1, finish)))  %>%
 mutate(BsmtFinType2 = as.integer(revalue(full$BsmtFinType2, finish)))

#Masonry
mason <- c('None'=0, 'BrkCmn'=0, 'BrkFace'=1, 'Stone'=2)
full$MasVnrType[is.na(full$MasVnrType)] <- 'None'
full <- full %>%
 mutate(MasVnrType = as.integer(revalue(full$MasVnrType, mason)))

#Integer replacements
full$GarageYrBlt[is.na(full$GarageYrBlt)] <- 0
full$BsmtHalfBath[is.na(full$BsmtHalfBath)] <- 0
full$BsmtFinSF2[is.na(full$BsmtFinSF2)] <- 0
full$BsmtUnfSF[is.na(full$BsmtUnfSF)] <- 0
full$TotalBsmtSF[is.na(full$TotalBsmtSF)] <- 0
full$MasVnrArea[is.na(full$MasVnrArea)] <-0

#Functionality
func = c('Sal'=0, 'Sev'=1, 'Maj2'=2, 'Maj1'=3, 'Mod'=4, 'Min2'=5, 'Min1'=6, 'Typ'=7)
full$Functional[is.na(full$Functional)] <- mode(full$Functional)

full <- full %>%
 mutate(Functional = as.integer(revalue(full$Functional, func)))
```

```
## The following `from` values were not present in `x`: Sal
```

```
#Lot via median since each corresponding neighborhood has limited lots

for (i in 1:nrow(full)){
        if(is.na(full$LotFrontage[i])){
               full$LotFrontage[i] <- as.integer(median(full$LotFrontage[full$Neighborho
od==full$Neighborhood[i]], na.rm=TRUE))
        }
}

shape <- c('IR3'=0, 'IR2'=1, 'IR1'=2, 'Reg'=3)

full <- full %>%
  mutate(LotShape = as.integer(revalue(full$LotShape, shape)))
```

(2.2) Identifying character variables and coercing into factors

```
#Coercing character variables
Chars <- names(full[,sapply(full, is.character)])
Chars
```

```
##  [1] "Street"        "LandContour"   "LandSlope"     "Neighborhood"
##  [5] "Condition1"    "Condition2"    "BldgType"      "HouseStyle"
##  [9] "RoofStyle"     "RoofMatl"      "Foundation"    "Heating"
## [13] "HeatingQC"     "CentralAir"    "GarageFinish"  "GarageQual"
## [17] "GarageCond"    "PavedDrive"    "SaleCondition"
```

(2.3) Cleaning remaining variables that were not already processed from imputing missing values

```
#Ratings Ordinal Variable
full$HeatingQC<-as.integer(revalue(full$HeatingQC, Qualities))
```

```
## The following `from` values were not present in `x`: None
```

```
full <- full %>%
  mutate(Street = as.integer(revalue(full$Street, c('Grvl'=0, 'Pave'=1)))) %>%
  mutate(PavedDrive = as.integer(revalue(full$PavedDrive, c('N'=0, 'P'=1, 'Y'=2)))) %>%
  mutate(LandSlope = as.integer(revalue(full$LandSlope, c('Sev'=0, 'Mod'=1, 'Gtl'=2))))
 %>%
  mutate(CentralAir = as.integer(revalue(full$CentralAir, c('N'=0, 'Y'=1))))

#Factored Categorical Variables
full <- full %>%
  mutate(Foundation = as.factor(full$Foundation)) %>%
  mutate(RoofStyle = as.factor(full$RoofStyle)) %>%
  mutate(RoofMatl = as.factor(full$RoofMatl)) %>%
  mutate(BldgType = as.factor(full$BldgType)) %>%
  mutate(Neighborhood = as.factor(full$Neighborhood)) %>%
  mutate(Condition1 = as.factor(full$Condition1))  %>%
  mutate(Condition2 = as.factor(full$Condition2)) %>%
  mutate(SaleCondition = as.factor(full$SaleCondition)) %>%
  mutate(HouseStyle = as.factor(full$HouseStyle)) %>%
  mutate(Heating = as.factor(full$Heating)) %>%
  mutate(LandContour = as.factor(full$LandContour)) %>%
  mutate(MoSold = as.factor(full$MoSold)) %>%
  mutate(YrSold = as.factor(full$YrSold)) %>%
  mutate(MSSubClass = as.factor(full$MSSubClass))
```

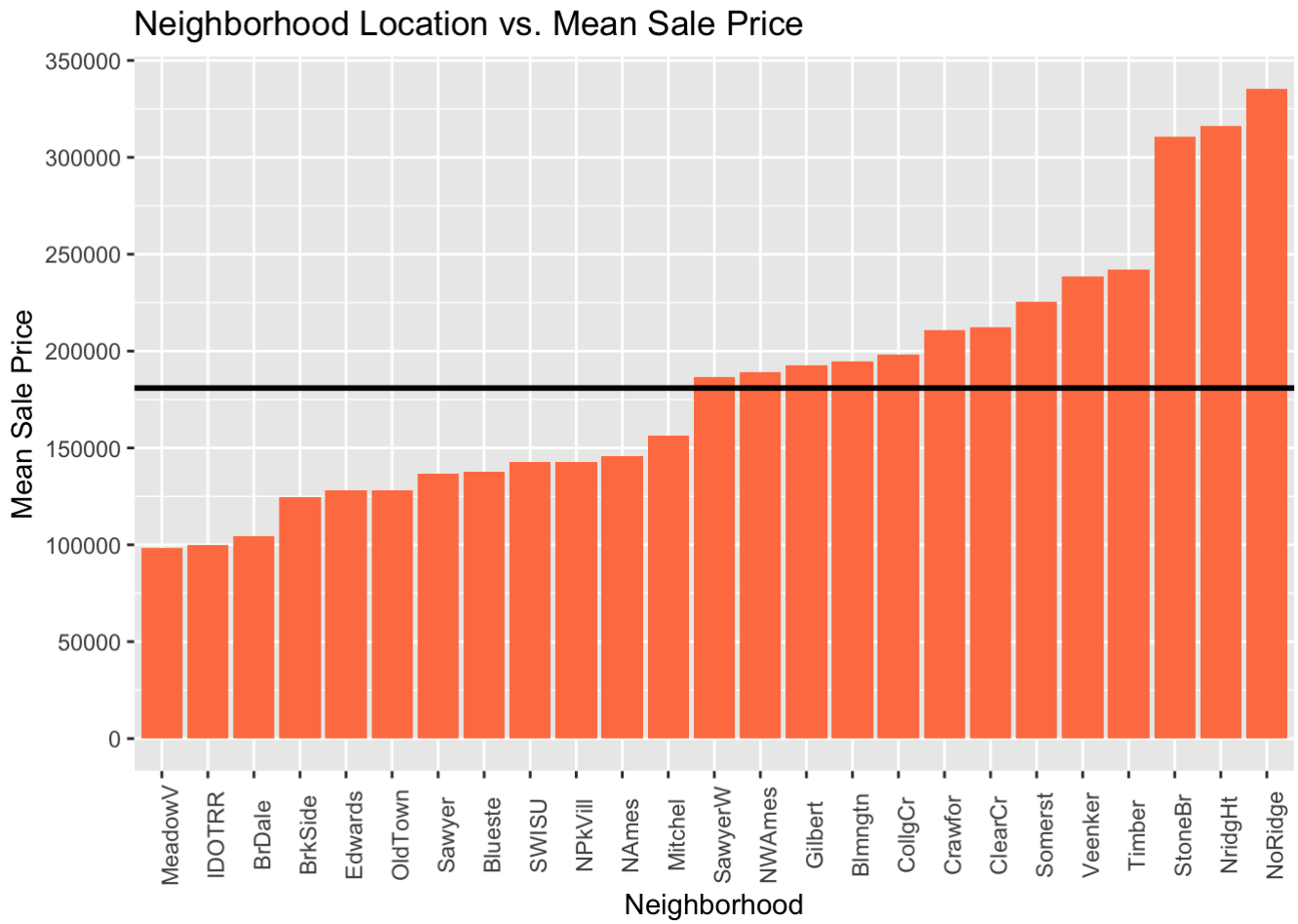
(2.4) Numerical Feature Engineering

```
full <- full %>%
  mutate(TotalSF = GrLivArea + TotalBsmtSF) %>%
  mutate(numBr = FullBath + HalfBath + BsmtHalfBath + BsmtFullBath) %>%
  mutate(TotalPorchSF = OpenPorchSF + EnclosedPorch + X3SsnPorch + ScreenPorch)
```

(2.5) Neighborhood Plotting & Binning

```
#Plotting mean sale price vs. neighborhood
ggplot(train, aes(x=reorder(Neighborhood, SalePrice, FUN=mean), y=SalePrice)) +
  geom_bar(stat='summary',fill='coral') + labs(x='Neighborhood', y="Mean Sale Price") +
    theme(axis.text.x = element_text(angle = 90)) +
    scale_y_continuous(breaks= seq(0, 500000, by = 50000)) +
    geom_hline(yintercept= mean(train$SalePrice), size = 1, color = "black") +
    ggtitle('Neighborhood Location vs. Mean Sale Price')
```

```
## No summary function supplied, defaulting to `mean_se()`
```




```
low <- c('MeadowV','IDOTRR','BrDale', 'BrkSide', 'Edwards', 'OldTown', 'Sawyer', 'Bluest
e', 'SWISU','NPkVill','NAmes','Mitchel')
high <- c('SawyerW','NWAmes','Gilbert','CollgrCr','Crawfor','Somerst','Veenker','Timber'
,'StoneBr','NridgeHt','Noridge')

full <- full %>%
  mutate(NBClass = as.numeric(ifelse(Neighborhood %in% low, 0 , 1)))
```

(2.6) Colinearity Cleaning

```
full <- subset(full, select = -c(GrLivArea,FullBath,OpenPorchSF,Neighborhood))
```

(2.7) Normalizing Data

```
#Since we added 4 numerical features
truenumericalnames <- append(truenumericalnames, c('TotalSF', 'numBr', 'TotalPorchSF','N
BClass'))
```

```
#Numeric variables other than Sale Price
numerical <- full[,names(full) %in% truenumericalnames]
skew <- apply(numerical, 2, skewness)
skew <- skew[(skew > 0.8) | (skew < -0.8)]

for(col in names(skew)){
    numerical[,col] <- log(numerical[,col]+1)
  }
```

```
preprocess <- preProcess(numerical, method=c("center", "scale"))
normalized <- predict(preprocess, numerical)
```

(2.8) Full Buildout

```
categorical <- full[, !(names(full) %in% truenumericalnames)]
cnames <- names(categorical)
```

```
# use caret dummyVars function for hot one encoding for categorical features, automatica
lly sorts through categorical and numeric
dummies <- dummyVars("~ .", data = full[,cnames])
categorical <- data.frame(predict(dummies,newdata=full[,cnames]))

full<- cbind(normalized,categorical)
dim(full)
```

```
## [1] 2919  245
```

```
#Find all the predictors with TRUE 'nzv' and store their names into a vector
zerovarpredictors <- nearZeroVar(full, saveMetrics = TRUE)
filtered <- rownames(zerovarpredictors)[zerovarpredictors$nzv == TRUE]

full <- full[,!names(full) %in% filtered]
dim(full)
```

```
## [1] 2919  106
```

(2.9) Training/Test/Validation Set

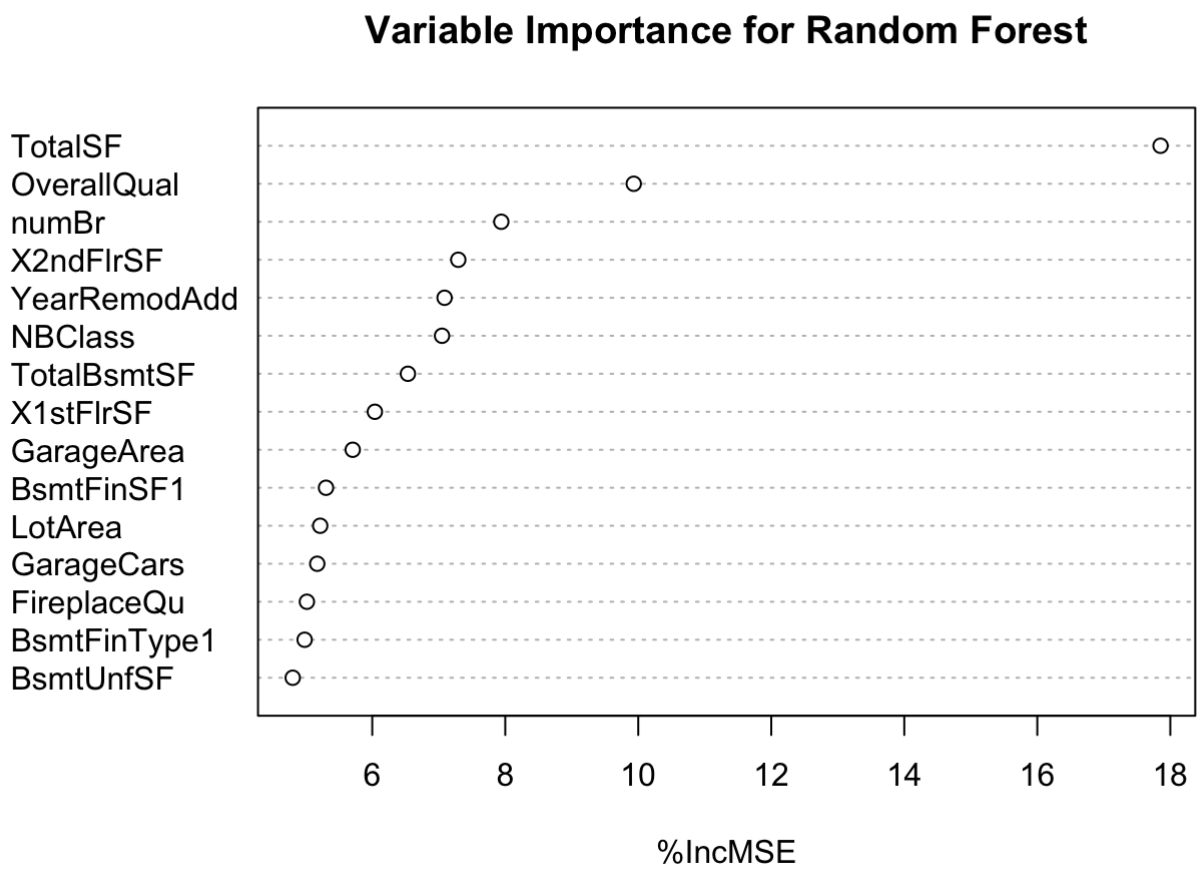
```
set.seed(369)
full.train <- full[1:1460, ]
full.test <- full[1461:2919,]

# We log transform our response variable since it is also a skewed numerical variable.
full.train$SalePrice = log(full.train$SalePrice+1)

#40% CV Set
part <- createDataPartition(y = full.train$SalePrice, p = 0.6, list = FALSE)
working.train <- full.train[part,]
validation.train <- full.train[-part,]
```

(2.10) Random Forest

```
set.seed(369)
rf.train = randomForest(x = working.train[,-106], y = working.train$SalePrice, ntree=100
, importance=TRUE)
rf.train
```

```
##
## Call:
##  randomForest(x = working.train[, -106], y = working.train$SalePrice,      ntree = 10
0, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 35
##
##           Mean of squared residuals: 0.01994836
##                     % Var explained: 87.75
```

```
#Importance function and Plot
rf.imp = importance(rf.train)
varImpPlot(rf.train, n.var = 15, type = 1, sort = T, main = "Variable Importance for Ran
dom Forest" )
```




```
#+xlab does not work on varImpPlot due to a dotchart overlap issue, it should have been
 xlab= "% Increase of MSE if Variable is randomly) permuted"
```

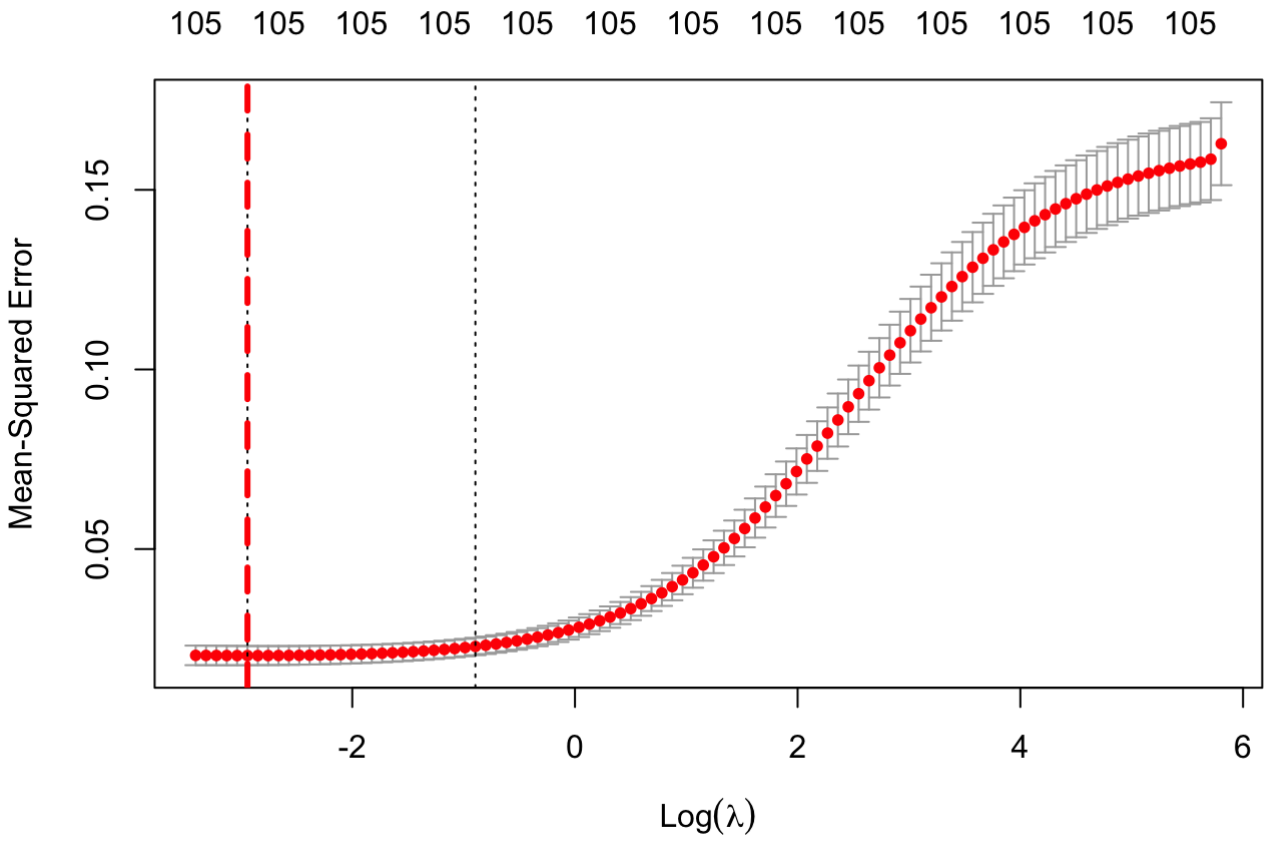
```
yhat.rf = predict(rf.train, newdata = validation.train)
mean((yhat.rf-validation.train$SalePrice)^2)
```

```
## [1] 0.01808418
```

(2.11) Ridge Regression

```
set.seed(120)

cv.ridge = cv.glmnet(as.matrix(working.train[,-106]), working.train$SalePrice, alpha = 0
)
```

```
plot(cv.ridge)
abline(v = log(cv.ridge$lambda.min), col="red", lwd=3, lty=2)
```




```
bestlam.ridge = cv.ridge$lambda.min
bestlam.ridge
```

```
## [1] 0.05280697
```

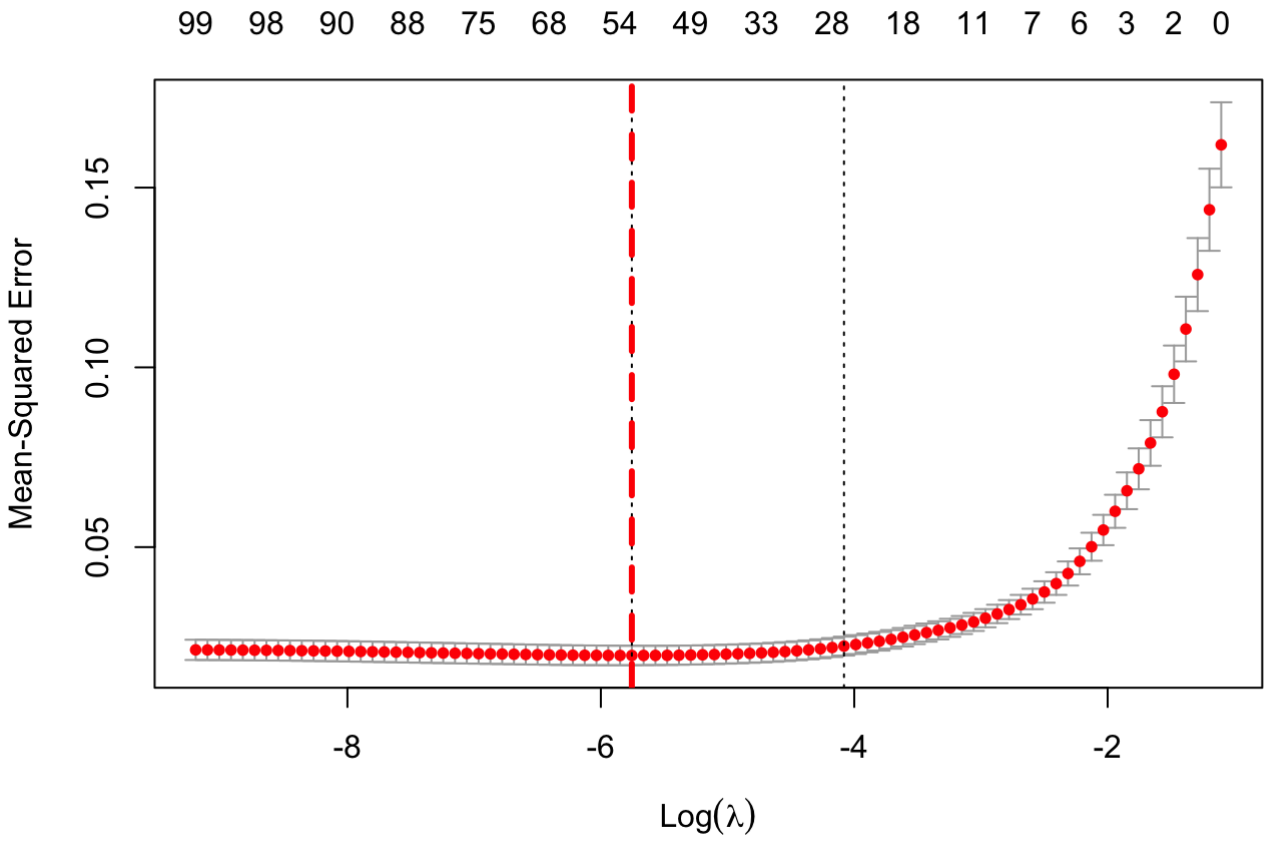
```
ridge.pred = predict(cv.ridge, s = bestlam.ridge, newx = as.matrix(validation.train[,-10
6]))
mean((ridge.pred-validation.train$SalePrice)^2)
```

```
## [1] 0.01709381
```

(2.12) Lasso Regression

```
set.seed(120)
cv.lasso = cv.glmnet(as.matrix(working.train[,-106]), working.train$SalePrice, alpha = 1
)
```

```
plot(cv.lasso)
abline(v = log(cv.lasso$lambda.min), col="red", lwd=3, lty=2)
```




```
bestlam.lasso = cv.lasso$lambda.min
bestlam.lasso
```

```
## [1] 0.003165695
```

```
lasso.pred = predict(cv.lasso, s = bestlam.lasso, newx = as.matrix(validation.train[,-10
6]))
mean((lasso.pred-validation.train$SalePrice)^2)
```

```
## [1] 0.01581656
```

Model Building

(3.1) Final Random Forest

```
set.seed(120)

predict <- as.data.frame(full.test)

rf.train = randomForest(x = full.train[,-106], y = full.train$SalePrice, ntree=100, impo
rtance=TRUE)
yhat.rf = predict(rf.train, newdata = full.test[,-106])
yhat.rf = as.double(exp(yhat.rf) - 1)
```

```
set.seed(120)

cv.ridge = cv.glmnet(as.matrix(full.train[,-106]), full.train$SalePrice, alpha = 0)
bestlam.ridge = cv.ridge$lambda.min
yhat.ridge = as.double(predict(cv.ridge, s = bestlam.ridge, newx = as.matrix(full.test[,
-106])))
yhat.ridge = as.double(exp(yhat.ridge)-1)
```

```
set.seed(120)

cv.lasso = cv.glmnet(as.matrix(full.train[,-106]), full.train$SalePrice, alpha = 1)
bestlam.lasso = cv.lasso$lambda.min
yhat.lasso = as.double(predict(cv.lasso, s = bestlam.lasso, newx = as.matrix(full.test[,
-106])))
yhat.lasso = as.double(exp(yhat.lasso)-1)
```

(3.2) True Test Error

```
#RF Error
#mean((yhat.rf-full.test$SalePrice)^2)

#Ridge Error
#mean((yhat.ridge-full.test$SalePrice)^2)

#Lasso Error
#mean((yhat.lasso-full.test$SalePrice)^2)
```

(3.3) Combined Model True Test Error

```
#Combined Model error (mean commented out so Rmd can knit)
combined.pred <- (yhat.rf + yhat.ridge + yhat.lasso) / 3.0
#mean((lasso.pred-full.test$SalePrice)^2)
```

```
#sub <- data.frame(Id = test_ID, SalePrice = yhat.lasso)
#write.csv(sub, "submissionlasso.csv", row.names=FALSE)

#sub <- data.frame(Id = test_ID, SalePrice = combined.pred)
#write.csv(sub, "submissioncombined.csv", row.names=FALSE)
```

Reference

1. Data Source: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview (https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview)
2. GGplot Design essentials: http://www.sthda.com/english/wiki/ggplot2-essentials (http://www.sthda.com/english/wiki/ggplot2-essentials)
3. Pre-Processing ML Data in R https://machinelearningmastery.com/pre-process-your-dataset-in-r/ (https://machinelearningmastery.com/pre-process-your-dataset-in-r/)
4. Implementing dummy variables for categorical variables via caret https://amunategui.github.io/dummyVar-Walkthrough/ (https://amunategui.github.io/dummyVar-Walkthrough/)

5. Implementing caret for Ridge/Lasso: https://stackoverflow.com/questions/48179423/error-error-in-lognetx-is-sparse-ix-jx-y-weights-offset-alpha-nobs (https://stackoverflow.com/questions/48179423/error-error-in-lognetx-is-sparse-ix-jx-y-weights-offset-alpha-nobs)
6. Implementing Near Zero Variance on Predictors via caret: https://www.rdocumentation.org/packages/caret/versions/6.0-76/topics/nearZeroVar (https://www.rdocumentation.org/packages/caret/versions/6.0-76/topics/nearZeroVar)
7. Labs 4,5,6