Ames Housing Price Predictor

Jake Hogan

12/4/2020

Executive Summary

The Ames Housing dataset contains 79 variables that describe almost every feature of residential homes in Ames, Iowa. The variables describe anything from the number of bedrooms and bathrooms to whether or not the road leading up to the house is paved. The dataset is presented as part of a Kaggle competition (kaggle.com) in which the goal is to use the data set to predict the sale price of Ames homes as accurately as possible. Exploratory data analysis and visualization was used to clean the data and explore which variables would be the most valuable for predictions. Then random forests, gradient boosting, and a regularized generalized linear model were used in an ensemble to make final sale price predictions. The metric used to judge the sale price predictions is the RMSE of the log transformed sale prices. The final log RMSE of the predictions is 0.13972.

Data Analysis

The Ames Housing dataset can be downloaded from the Kaggle website linked above but requires an account. I've uploaded the raw data zip file to my GitHub account for easy access (github.com/hoganjr). The dataset comes split into a train file and test file. The train and test files contain 1460 and 1459 home sales, respectively. The test file has all sale prices removed. The zip file also contains a description for each of the 79 variables in the dataset.

Looking into the train data set shows that it has sale prices from 34,900 to 755,000 USD. The median sale price is 163,000 USD. Below is a distribution of sale prices. The distribution is right-skewed.

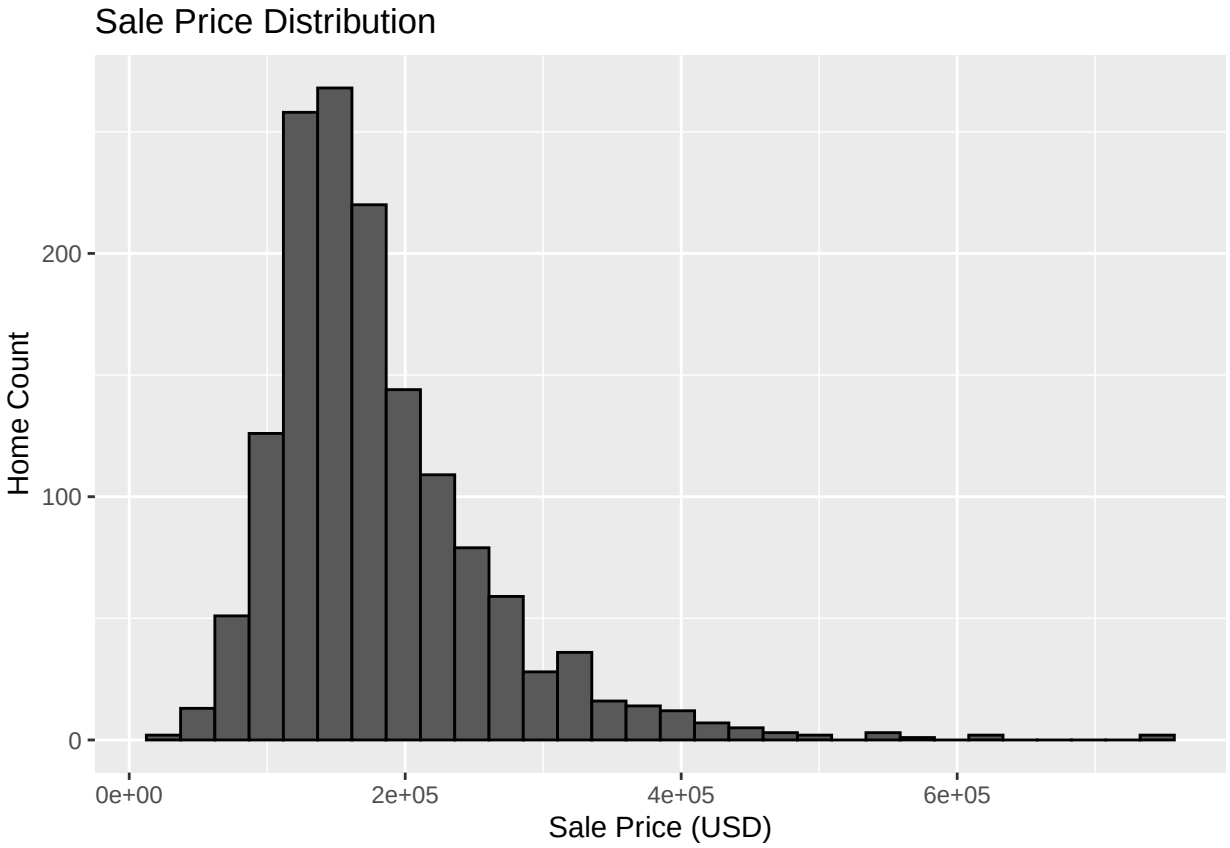

Figure 1: Distribution of Home Sale Prices

A log transformation of the sale prices creates a more normal distribution.

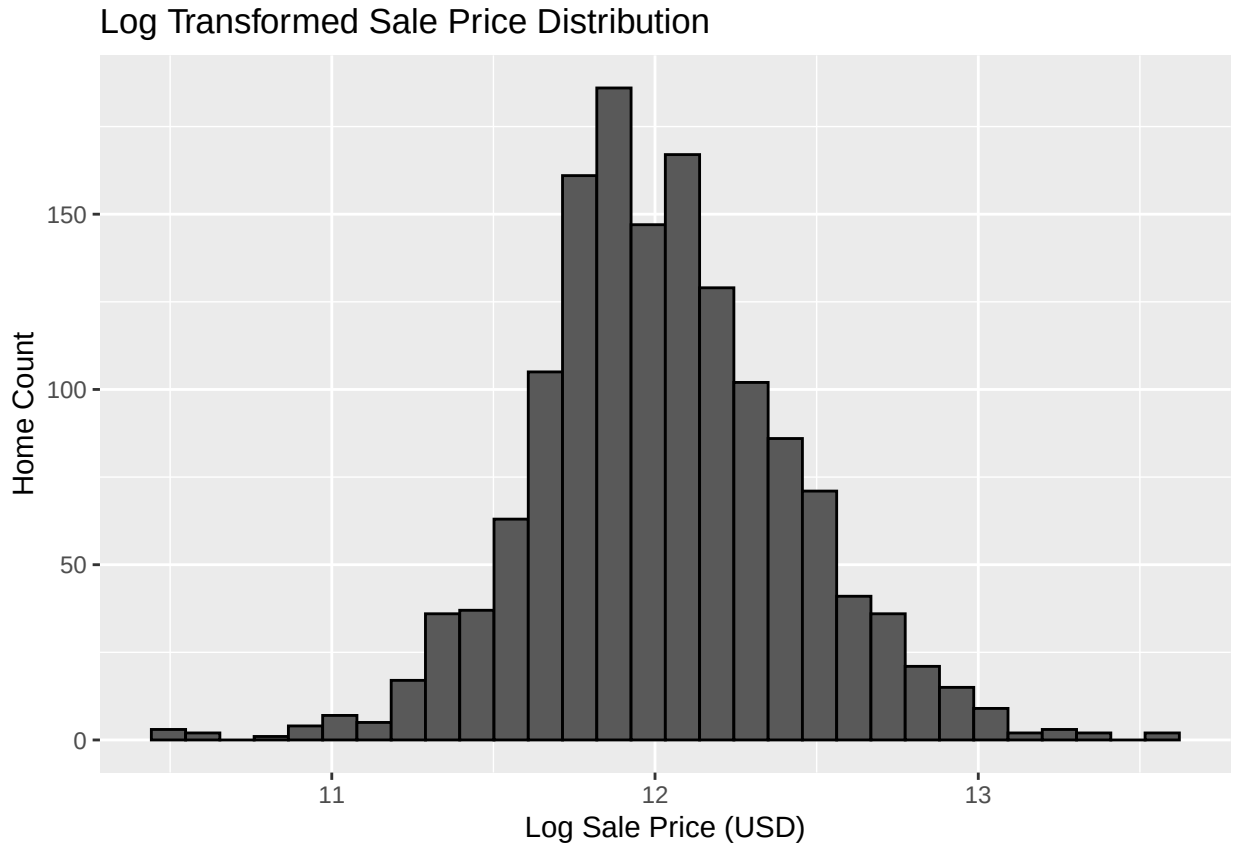

Figure 2: Distribution of Log Transformed Home Sale Prices

From intuition and personal experience I know that location (neighborhood), square footage, number of bedrooms and bathrooms, as well as age can all impact the sale price of a house. Here's the mean sale price by neighborhood.

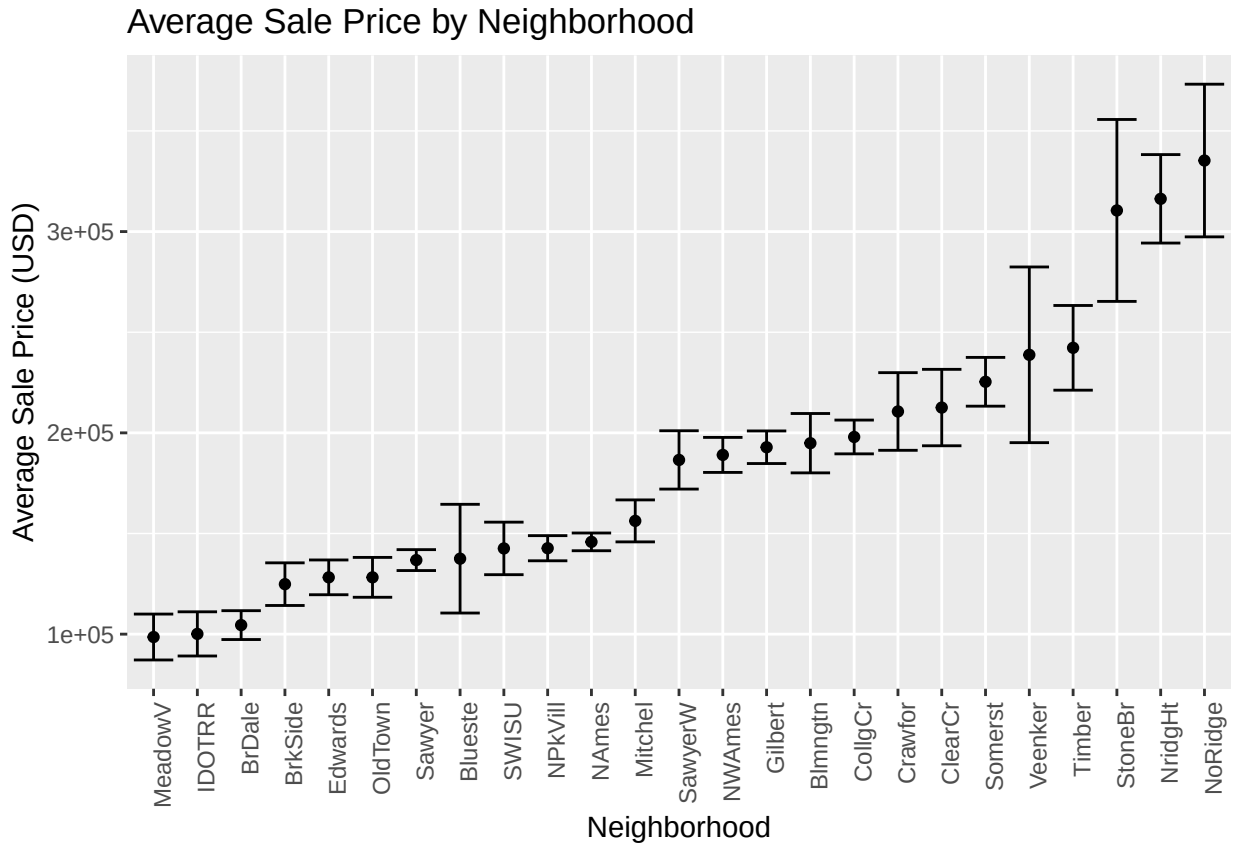

Figure 3: Mean Sale Price by Neighborhood

Clearly some neighborhoods are more valuable than others. The square footage data is provided in multiple variables which makes it less clear whether or not square footage matters for sale price. However if we combine them to get the total square footage a trend emerges.

Figure 4: Sale Price by Square Footage

There appears to be 2 outliers with square footages over 7,500 but with low sale prices. As for bathrooms, the dataset again splits these up into four categories (Basement Full Baths, Basement Half Baths, Above Ground Full Baths and Above Ground Half Baths). The four plots below show how quantities of each relate to sale price.

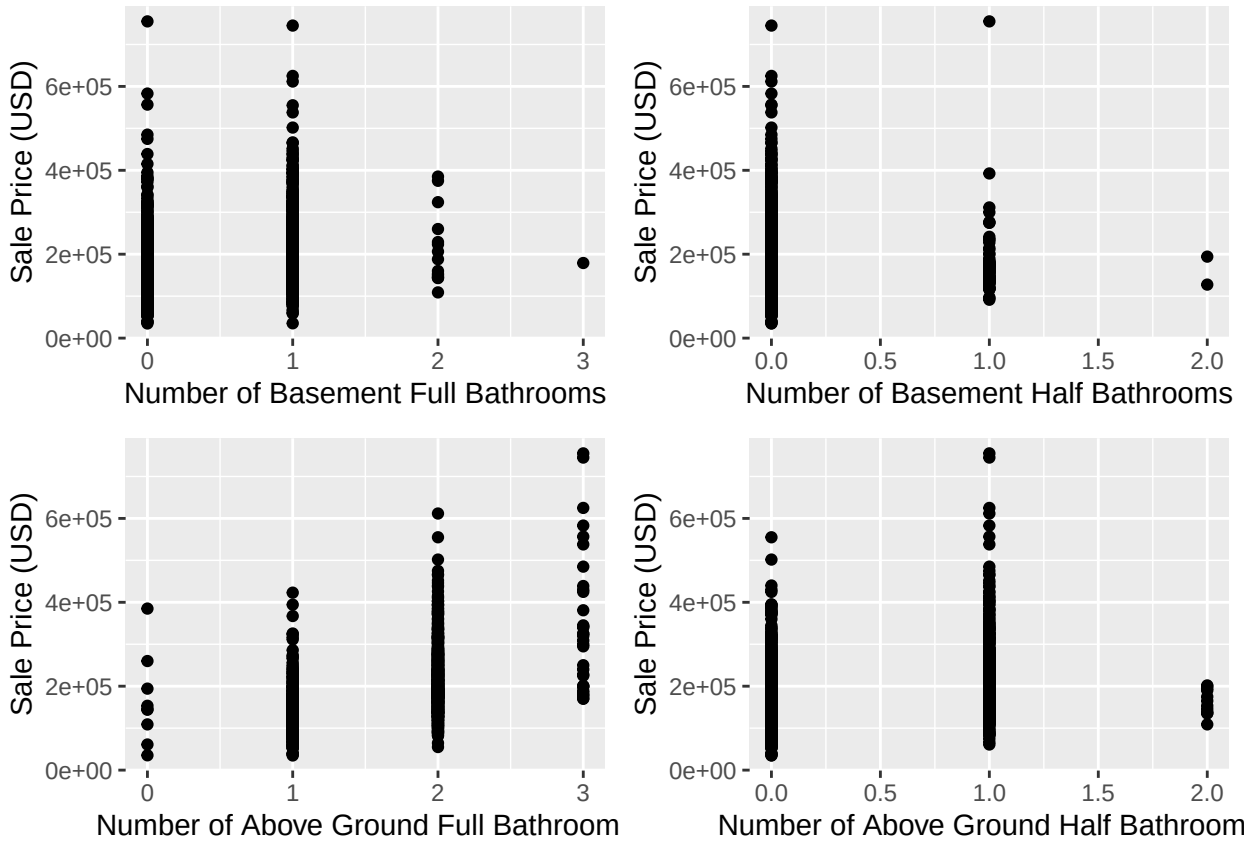

Figure 5: Sale Price by Bathroom Type

It appears above ground full baths may have a relationship with sale price but the rest are unclear. Let's combine them all to create a total bathrooms variable.

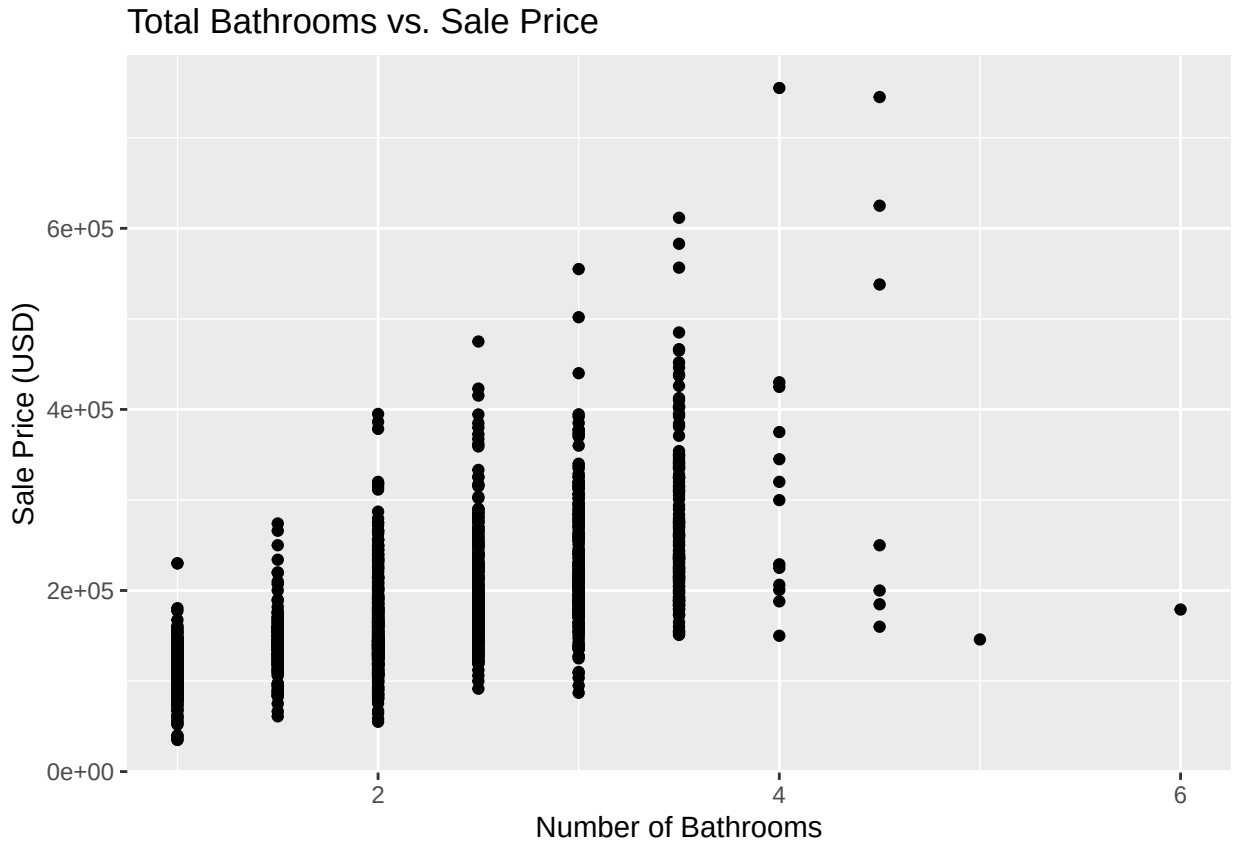

Figure 6: Sale Price by Total Bathrooms

There is a clearer trend now with a couple of outliers above 4.5 total baths. Now let's look at bedrooms. The only data available is bedrooms above ground. There doesn't appear to be a strong relationship with sale price.

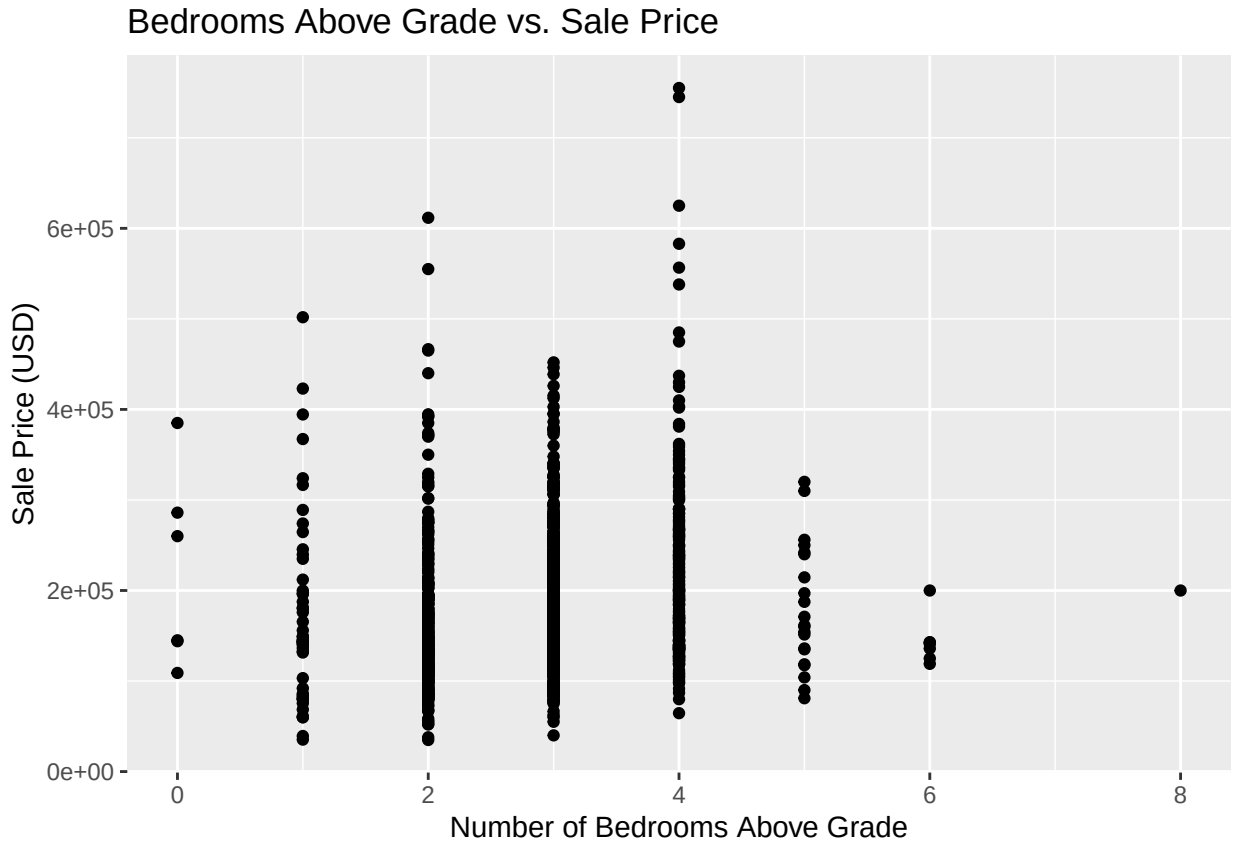

Figure 7: Sale Price by Total Bedrooms

There appears to be two variables relating to the age of the house, the year it was built and the year it was remodeled. If it hasn't been remodeled then the year it was remodeled matches the year it was built. Here are both variables plotted against sale price.

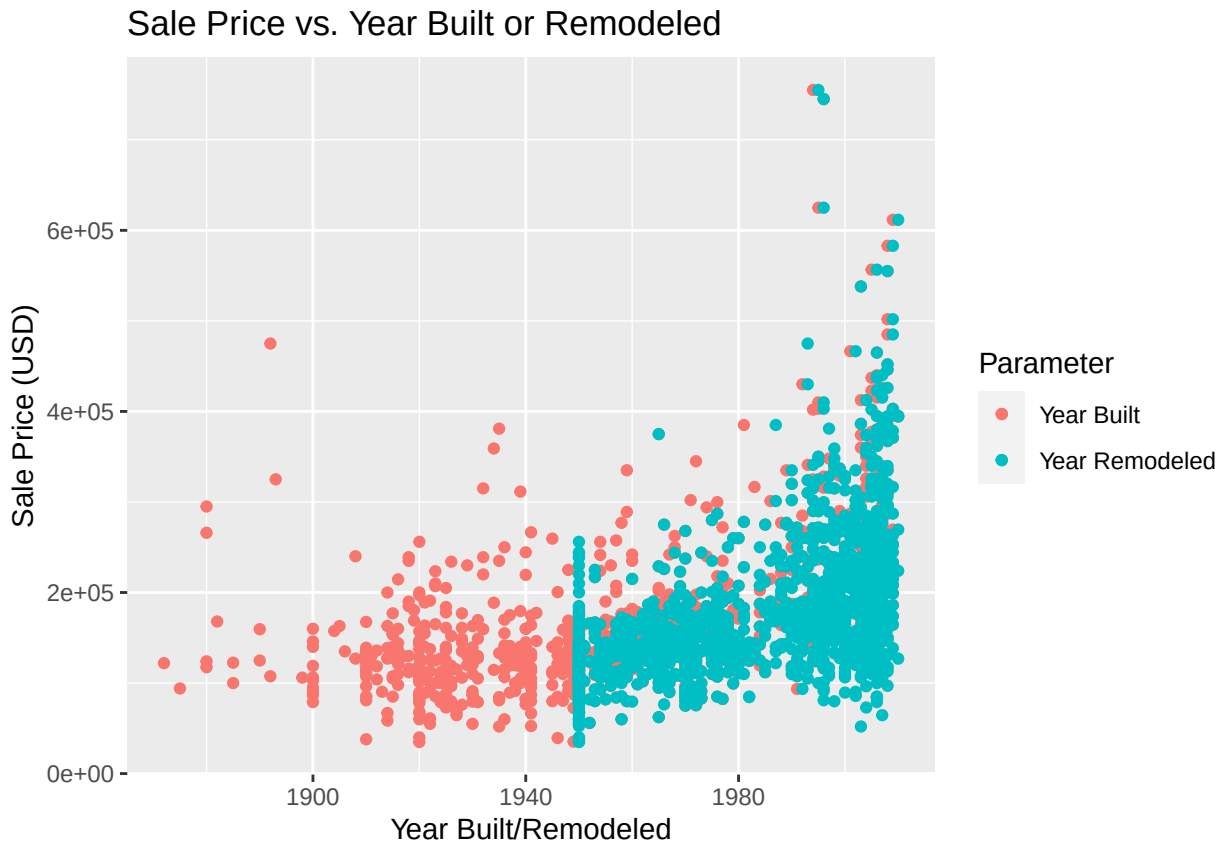

Figure 8: Sale Price vs. Year Built and Year Remodeled

There's no obvious distinction between the two parameters but there appears to be something odd with the year remodeled data. It appears that no remodeling was done prior to 1950 and that a spike in renovations occurred in the early 1950s.

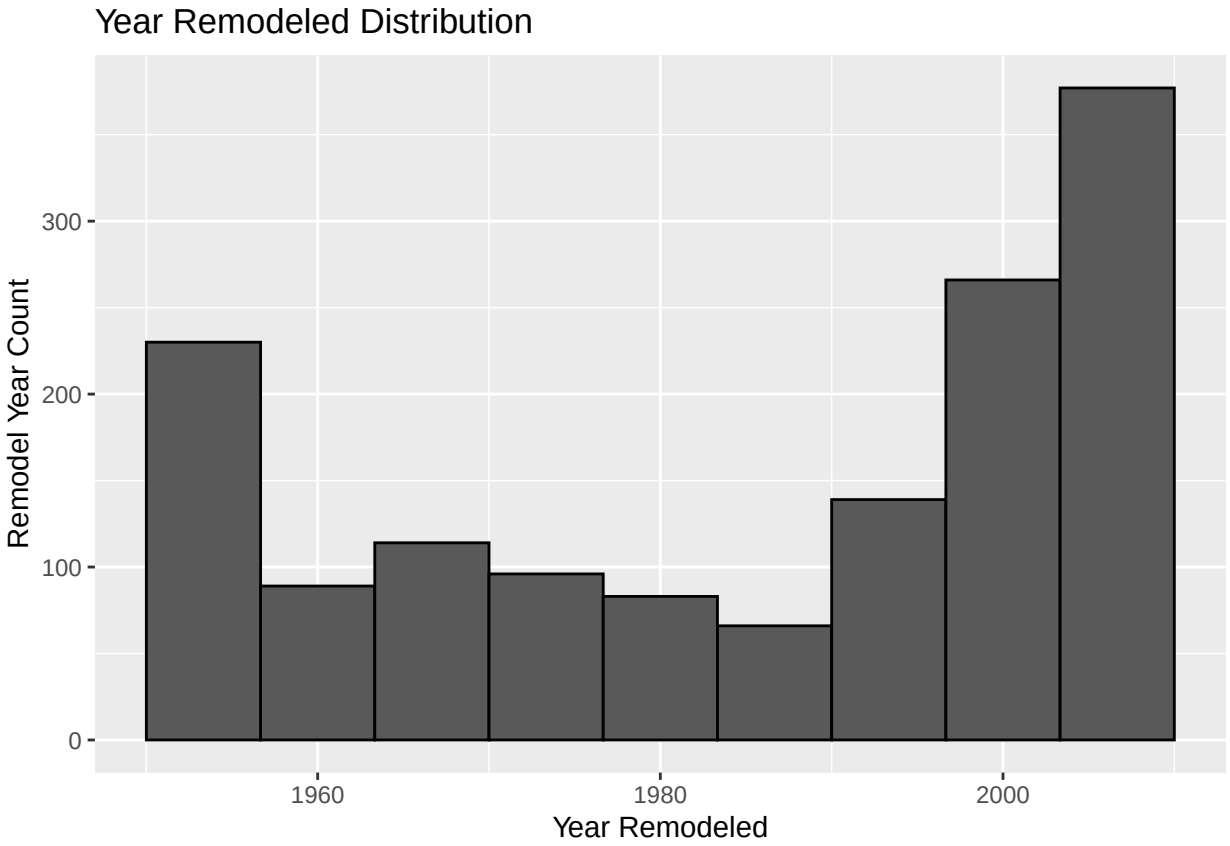

Figure 9: Distribution of Year Remodeled

It's possible the higher number of renovations in the early 1950s represents the homes being modernized with things like indoor plumbing or electricity but it's unclear. Due to this let's create new variables for whether or not the house was remodeled (a yes/no variable) and the relative age of the house (how long between the year it was sold and the year it was renovated). Homes that haven't been renovated or look dated can negatively impact the sale price.

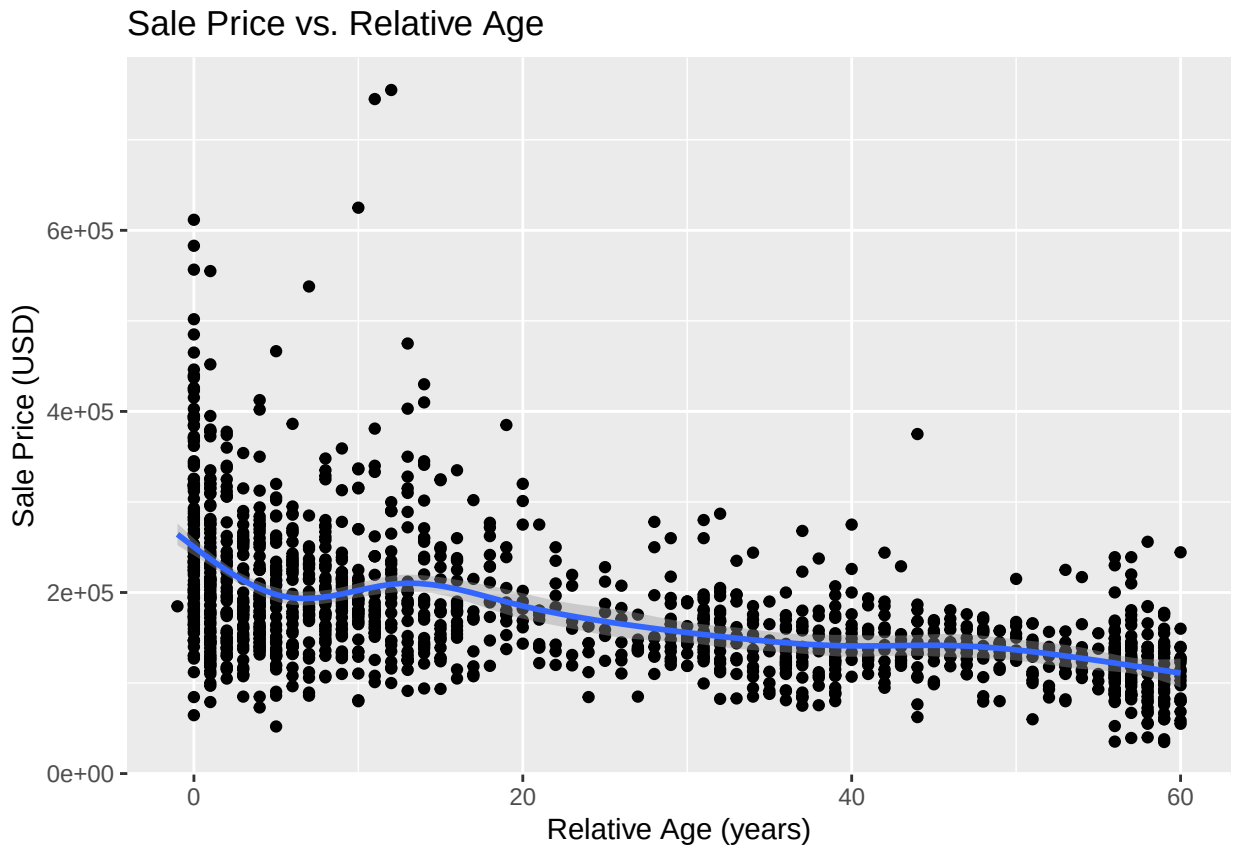

Figure 10: Relative Age vs. Sale Price

The rest of the parameters are more obscure and more difficult to determine the relationship with sale price. Before exploring some of these parameters we'll clean up some of the data by replacing any NAs with either a 0 for numeric data or None for categorical data. These operations are performed on both the train and test dataset.

There are 42 numeric variables and 43 categorical variables in the dataset. Comparing the lists of numeric and categorical variables shows that there is some overlap between the two. A major one is the "Quality" variables. Some use 1 through 10 and some use five abbreviations that are ordinal. The categorical variables this applies to are: Exterior Quality, Exterior Condition, Kitchen Quality, Fireplace Quality, Garage Quality, Garage Condition, Pool Quality, Heating Quality, Basement Quality, and Basement Condition. Fence Quality doesn't appear to have any sort of ranking for the quality abbreviations so that will be ignored. Below is an example of how Exterior Quality was modified. All others were modified in this fashion.

```
#all quality params use Po = Poor, Fa = Fair, TA = Typicla/Average, Gd = Good,
#Ex = Excellent
#I have added "None" above to replace NAs.
qual_vals <- c('None' = 0, 'Po' = 1, 'Fa' = 2, 'TA' = 3, 'Gd' = 4, 'Ex' = 5)

#Exterior Quality. Fixing train and test sets
train_raw$ExterQual <- as.integer(plyr::revalue(train_raw$ExterQual, qual_vals))
test_raw$ExterQual <- as.integer(plyr::revalue(test_raw$ExterQual, qual_vals))
```

The MSSubClasses variable, which identifies the type of dwelling, is numeric but not ordinal so that variable is converted to a factor.

Separating the numeric variables from the categorical variables allows us to create a correlation matrix. Since

there are a large number of variables the matrix is filtered to show only the variables that have a greater than 0.5 correlation coefficient with sale price.

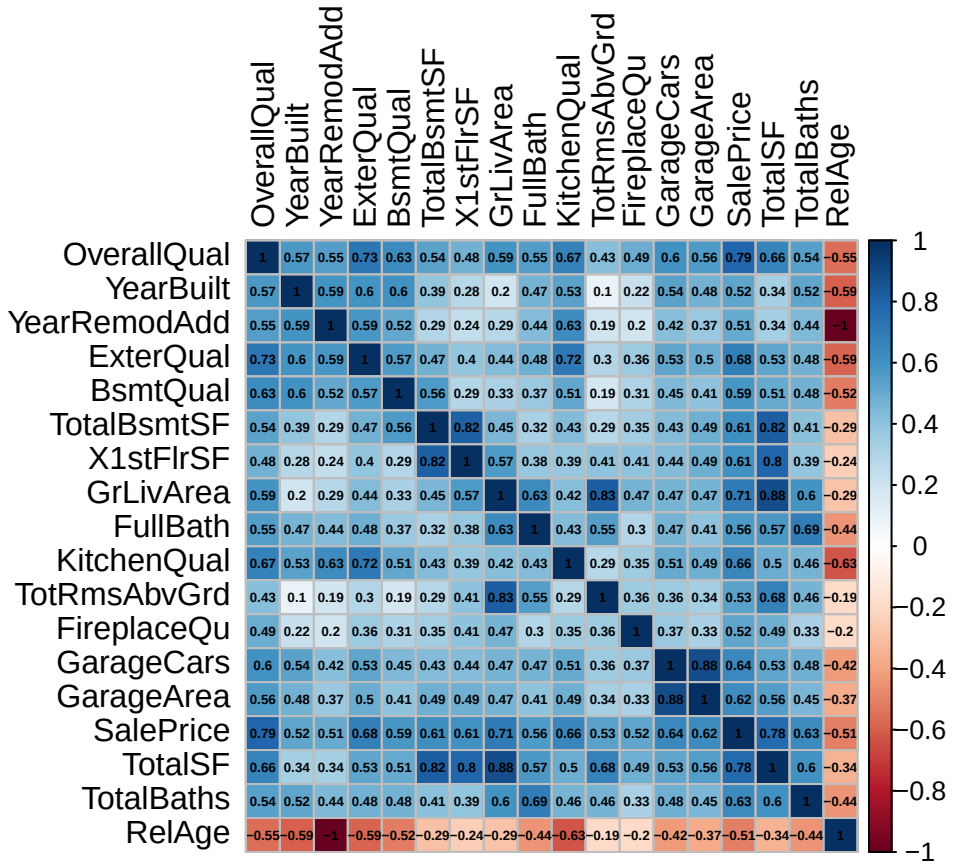

Figure 11: Numeric Variable Correlation

From this chart a list of variables to remove for prediction purposes can be created. We'll remove variables with poor correlation to sale price (less than or equal to 0.4). Also we'll remove "FullBath" due to high correlation with "TotalBaths". We'll remove "GrLivArea", "TotalBsmtSF", "1stFlrSF" due to high correlation with each other and "TotalSF". Then "YearBuilt" and "YearRemodAdd" are removed due to correlation with "RelAge".

To look more closely at the categorical data the variables are factorized and then an Analysis of Variance (ANOVA) test is used to get an impression of how dependent sale price is on each variable. The ANOVA test (using the aov() function) provides the F statistic and p-values for each variable. Higher F statistics correspond to smaller p-values. The smaller the p-value the more confident we can be that the variable does impact the sale price (confidence that we can reject the null hypothesis that it does not affect sale price). The chart below shows the log transformation of the inverse p-value to show how much sale price depends on each variable. Clearly neighborhood dominates in value with MSZoning and lot shape also showing importance. Based on these results we'll remove any variables that have a log transformed inverse p-value that is less than 0.01 times the neighborhood value. Additionally, the near zero variance (nearZeroVar()) function is run on the entire train set. The variables recommended are also removed. Finally, the outliers previously discussed are removed.

```
#use regression to look at categorical variables
#lets look at the variables that remain
char_names <- names(which(sapply(train_raw, is.character)))
```

```r
str(train_raw[,char_names]) #doesn't appear to be any kind of ranking with the
```

```
## 'data.frame':    1460 obs. of  33 variables:
##  $ MSZoning     : chr  "RL" "RL" "RL" "RL" ...
##  $ Street       : chr  "Pave" "Pave" "Pave" "Pave" ...
##  $ Alley        : chr  "None" "None" "None" "None" ...
##  $ LotShape     : chr  "Reg" "Reg" "IR1" "IR1" ...
##  $ LandContour  : chr  "Lvl" "Lvl" "Lvl" "Lvl" ...
##  $ Utilities    : chr  "AllPub" "AllPub" "AllPub" "AllPub" ...
##  $ LotConfig    : chr  "Inside" "FR2" "Inside" "Corner" ...
##  $ LandSlope    : chr  "Gtl" "Gtl" "Gtl" "Gtl" ...
##  $ Neighborhood : chr  "CollgCr" "Veenker" "CollgCr" "Crawfor" ...
##  $ Condition1   : chr  "Norm" "Feedr" "Norm" "Norm" ...
##  $ Condition2   : chr  "Norm" "Norm" "Norm" "Norm" ...
##  $ BldgType     : chr  "1Fam" "1Fam" "1Fam" "1Fam" ...
##  $ HouseStyle   : chr  "2Story" "1Story" "2Story" "2Story" ...
##  $ RoofStyle    : chr  "Gable" "Gable" "Gable" "Gable" ...
##  $ RoofMatl     : chr  "CompShg" "CompShg" "CompShg" "CompShg" ...
##  $ Exterior1st  : chr  "VinylSd" "MetalSd" "VinylSd" "Wd Sdng" ...
##  $ Exterior2nd  : chr  "VinylSd" "MetalSd" "VinylSd" "Wd Shng" ...
##  $ MasVnrType   : chr  "BrkFace" "None" "BrkFace" "None" ...
##  $ Foundation   : chr  "PConc" "CBlock" "PConc" "BrkTil" ...
##  $ BsmtExposure : chr  "No" "Gd" "Mn" "No" ...
##  $ BsmtFinType1 : chr  "GLQ" "ALQ" "GLQ" "ALQ" ...
##  $ BsmtFinType2 : chr  "Unf" "Unf" "Unf" "Unf" ...
##  $ Heating      : chr  "GasA" "GasA" "GasA" "GasA" ...
##  $ CentralAir   : chr  "Y" "Y" "Y" "Y" ...
##  $ Electrical   : chr  "SBrkr" "SBrkr" "SBrkr" "SBrkr" ...
##  $ Functional   : chr  "Typ" "Typ" "Typ" "Typ" ...
##  $ GarageType   : chr  "Attchd" "Attchd" "Attchd" "Detchd" ...
##  $ GarageFinish : chr  "RFn" "RFn" "RFn" "Unf" ...
##  $ PavedDrive   : chr  "Y" "Y" "Y" "Y" ...
##  $ Fence        : chr  "None" "None" "None" "None" ...
##  $ MiscFeature  : chr  "None" "None" "None" "None" ...
##  $ SaleType     : chr  "WD" "WD" "WD" "WD" ...
##  $ SaleCondition: chr  "Normal" "Normal" "Normal" "Abnorml" ...
```

```r
#remaining variables.
#convert remaining variables to factors.
train_raw[char_names] <- lapply(train_raw[char_names], factor)
#repeat for test data
char_names_test <- names(which(sapply(test_raw, is.character)))
test_raw[char_names_test] <- lapply(test_raw[char_names_test], factor)

#use ANOVA (aov()) to look at relationships between sale price and categorical variables
char_cor_mat <- train_raw[append(char_names,
                                 c("SalePrice","MSSubClass"))] #create categorical matrix
aov_cat_var <- broom::tidy(aov(SalePrice ~.,
                               char_cor_mat )) #run aov on the categorical matrix

aov_sum <- aov_cat_var %>% filter(!is.na(statistic)) %>%
  arrange(p.value) #summarize the aov test with ascending p-value

#plot by log(1/p.value) to show dependence
```

```r
aov_sum  %>% mutate(term = reorder(term, p.value)) %>%
  ggplot(aes(term,log(1/p.value))) + geom_col() +
  ggtitle("Variable vs Log of inverse p-value") + xlab("Variable") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Log of inverse p-value")
```

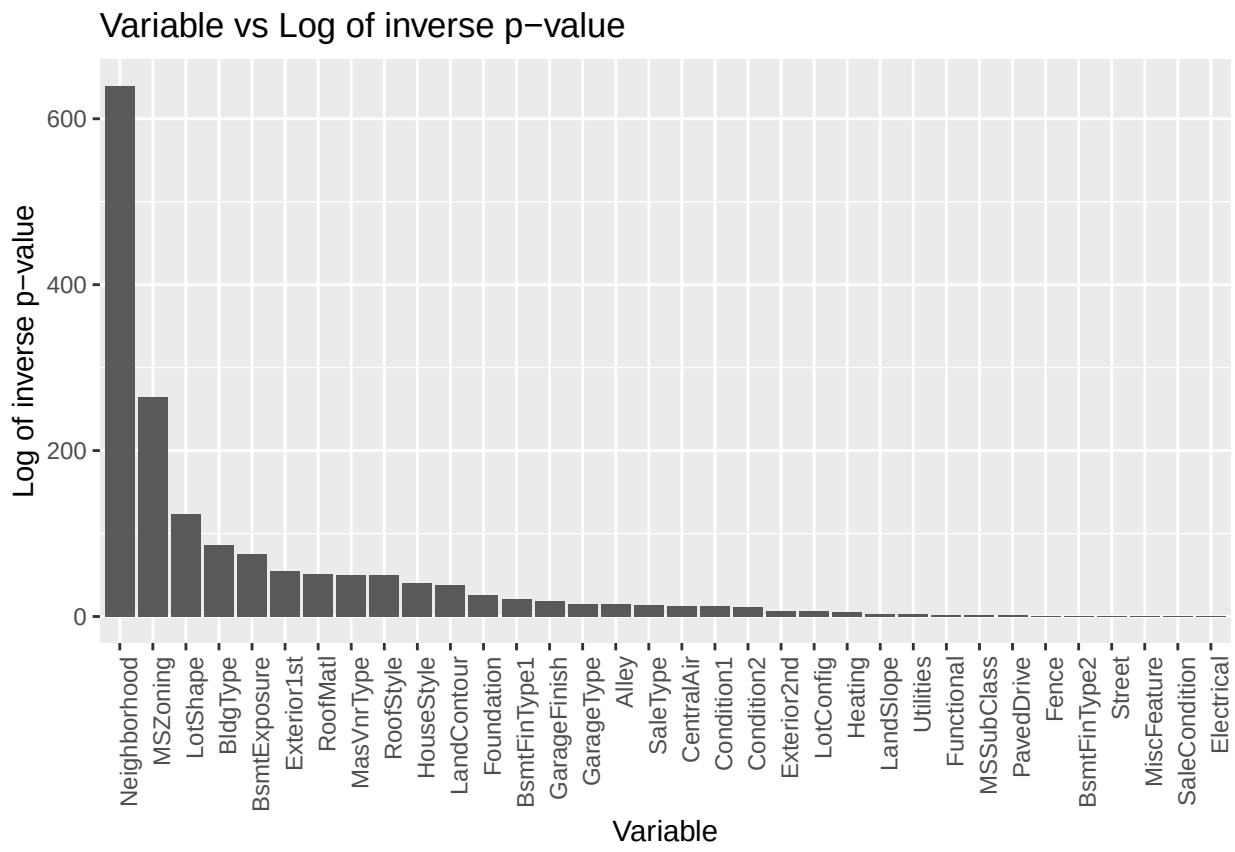

Figure 12: Categorical Variable Dependence

```r
#looks like neighborhood, zoning, lot shape, bldg type, and basement exposure
#might be valuable

#pick variables to remove
remove_aov <- aov_sum %>%
  filter(log(1/p.value) <= 0.01*max(log(1/p.value))) %>% pull(term)

#use nearzerovariance for pulling params out of model
nzv <- nearZeroVar(train_raw)
names(train_raw[,nzv]) #all these either had low F stat in anova or low cor
```

```
##  [1] "Street"        "Alley"         "LandContour"   "Utilities"
##  [5] "LandSlope"     "Condition2"    "RoofMatl"      "BsmtCond"
##  [9] "BsmtFinType2"  "BsmtFinSF2"    "Heating"       "LowQualFinSF"
## [13] "KitchenAbvGr"  "Functional"    "EnclosedPorch" "X3SsnPorch"
## [17] "ScreenPorch"   "PoolArea"      "PoolQC"        "MiscFeature"
## [21] "MiscVal"
```

```
nzv_remove <- names(train_raw[,nzv]) #create list of cols to remove
remove_combined <-unique(c(corr_remove,
                           remove_aov,nzv_remove)) #combine all variables to be removed

#removing all previously discussed parameters
train_simplified <- train_raw[, !names(train_raw) %in%
                                 remove_combined] #remove all unused params

#removing square footage and bathroom outliers
train_simplified <- train_simplified[-append(outliers_sf,outliers_baths),]
```

Model Development Methods

The performance of the prediction model will be judged on the log RMSE defined here:

```
#Define log RMSE function on which the competition is judged.
Log_RMSE <- function(actual_prices, pred_prices){
  sqrt(mean((log(actual_prices) - log(pred_prices))^2))
}
```

Since all training and testing of the models will have to occur on the provided train data set we need to partition the train dataset into train and test sets. A partition of 90% for training and 10% for testing was selected since the train set is fairly small overall.

```
#split train set for train/test of algos
#first we need to divide the train set in to train/test sets.  Giving 10% to test
set.seed(1, sample.kind = "Rounding")
train_split_index <- createDataPartition(y = train_simplified$SalePrice, times = 1,
                                         p = 0.1,
                                       list = FALSE)
split_train <- train_simplified[-train_split_index,]
split_test <- train_simplified[train_split_index,]
```

Due to the nature of the dataset having a mix a numeric and categorical data the types of algorithms that can be applied are limited to those that can handle both. Random forests, gradient boosting, and regularized generalized linear models are all common algorithms that can be applied to this data and can all be found in the caret package documentation[1,2].

Random Forests

Random forests improves upon decision trees by averaging multiple decision trees. The decision trees are assembled randomly so they are unique. To train the random forest model in caret I ran the Rborist method with a 10 fold cross validation. Rborist can be tuned on the minimum node size (minNode) and the number of trial predictors for a split (predFixed). For calculation time the number of trees is fixed to 50. The chart below shows the best tune created by the model. The best tune is then fit with 1000 trees.

```
#starting with Random Forests (aka Rborist for tuning)
library(Rborist)
control <- trainControl(method="cv", number = 10, p = 0.8) #cross validation
grid <- expand.grid(minNode = c(1,5) , predFixed = seq(1,15,2)) #tuning params
train_rf <-  train(split_train[, !names(split_train) %in% c("Id","SalePrice")],
                   log(split_train$SalePrice),
                   method = "Rborist",
                   nTree = 50,
                   trControl = control,
```

```
                  tuneGrid = grid,
                  nSamp = 5000)
ggplot(train_rf) + ggtitle("Random Forests Tuning Results") #plut tuning results
```

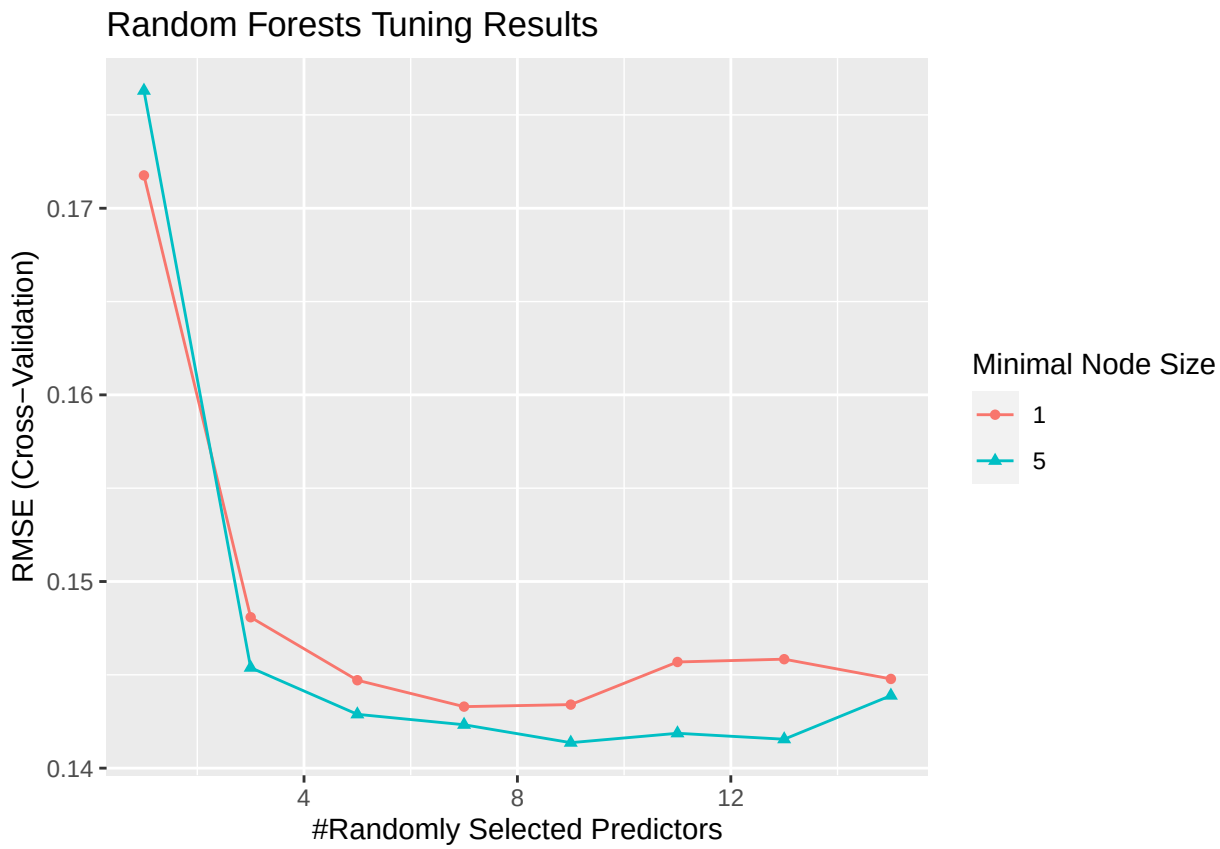

Figure 13: Random Forests Tuning

```
train_rf$bestTune #best  min node size and variables sampled at each split
```

```
##    predFixed minNode
## 13         9       5
```

```
#now fit the rf model with the best tune params
fit_rf <- train(split_train[, !names(split_train) %in% c("Id","SalePrice")],
                log(split_train$SalePrice),
                  method = "Rborist",
                  nTree = 1000,
                  tuneGrid = expand.grid(minNode = train_rf$bestTune$minNode,
                                         predFixed =train_rf$bestTune$predFixed))

pred_rf <- predict(fit_rf,
                   split_test) #predict using best fit

rmse_rf <- Log_RMSE(split_test$SalePrice,
                    exp(pred_rf)) #reverse the log transformation for real

rmse_results <- tibble(Method = "Random Forests", LogRMSE = rmse_rf)
```

```
#make table
if(knitr::is_html_output()){
  knitr::kable(rmse_results, "html", caption = "RMSE Results") %>%
    kableExtra::kable_styling(bootstrap_options = "striped", full_width = FALSE)
} else{
  knitr::kable(rmse_results, "latex", booktabs = TRUE, caption = "RMSE Results") %>%
    kableExtra::kable_styling(font_size = 8, latex_options = "HOLD_position")
}
```

Table 1: RMSE Results

Method	LogRMSE
Random Forests	0.131804

Making a prediction on the split test set shows a log RMSE of 0.131804. Let's try to improve.

Gradient Boosting

Gradient boosting produces a prediction model in the form of an ensemble of weaker prediction models. A tree is fit to the dataset with all observations getting equal weight. Once the first tree is evaluated the weights of values are changed based on how easy or hard they are to predict and a new tree is fit. Subsequent trees are all built off of the previous which becomes an ensemble of weighted trees. Here the xgboost method is used. I've tuned based on the learning rate (eta) which adjusts the weights of features between boosting steps to make the process more/less conservative. I've also tuned by max_depth which is the maximum depth of a tree and I've tuned on min_child_weight which sets a minimum weight for the sum of instance in a leaf node.

The gradient boosting does not accept categorical variables. Categorical variables must be converted to a numeric representation. This is accomplished by one-hot encoding the categorical variables. Basically each category gets a column with 1 or 0.

```
#use dummyVars function to generate one-hot df of categorical variables.
split_train_num <- split_train[,names(which(sapply(split_train,
                                                   is.numeric)))] #create numeric df
split_train_cat <- split_train[,names(which(sapply(split_train,
                                                   is.factor)))] #create categorical df
dummy <- dummyVars("~.", data = split_train_cat) #create dummy data frame
dummy_df <- data.frame(predict(dummy,
                               newdata = split_train_cat)) #fill dummy df with categorical data

oh_split_train <- cbind(dummy_df,
                        split_train_num) #combine numeric and one-hot cat df into single df

xgb_control <- trainControl(method="cv", number = 5, p = 0.8) #cross validation
train_xgb <- train(x = oh_split_train[, !names(oh_split_train) %in% c("Id","SalePrice")],
                   y = log(oh_split_train$SalePrice),
                   method = "xgbTree", #initially tried xgbLinear but got high RMSEs
                   trControl = xgb_control,
                   tuneGrid = expand.grid(nrounds = 1000 , #tuning params
                                          eta = c(0.01, 0.05, 0.1),
                                          max_depth = c(2,3,4,5,6),
                                          gamma = 0,
                                          colsample_bytree = 1,
                                          min_child_weight = c(1,2,3,4,5),
```

```
                                          subsample = 1)
                    )
ggplot(train_xgb) + ggtitle("Gradient Boost Tuning Results") #plot tuning results
```

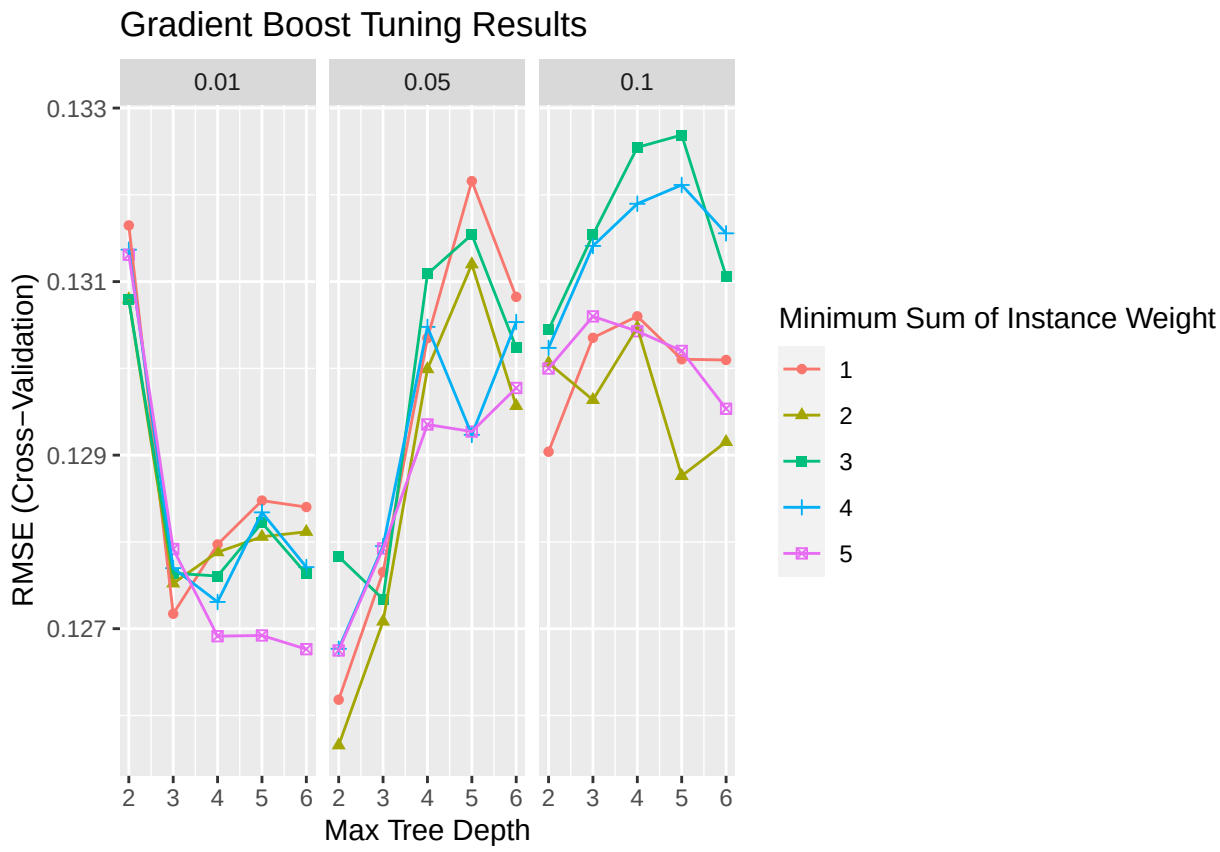

Figure 14: Gradient Boosting Tuning

```
train_xgb$bestTune #best  min node size and variables sampled at each split
```

```
##    nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 27    1000         2 0.05     0                1                2         1
```

```
#now fit to best tune params
fit_xgb <- train(x = oh_split_train[, !names(oh_split_train) %in% c("Id","SalePrice")],
                 y = log(oh_split_train$SalePrice),
                 method = "xgbTree",
                 tuneGrid = expand.grid(nrounds = 1000 ,
                                        eta = train_xgb$bestTune$eta,
                                        max_depth = train_xgb$bestTune$max_depth,
                                        gamma = 0,
                                        colsample_bytree = 1,
                                        min_child_weight = train_xgb$bestTune$min_child_weight,
                                        subsample = 1))

#create one-hot encoded split_test for prediction
split_test_num <- split_test[,names(which(sapply(split_test,
                                                 is.numeric)))] #create numeric df
```

```
split_test_cat <- split_test[,names(which(sapply(split_test,
                                                 is.factor)))] #create categorical df
dummy <- dummyVars("~.", data = split_test_cat) #create dummy data frame
dummy_df <- data.frame(predict(dummy,
                               newdata = split_test_cat)) #fill dummy df with categorical data

oh_split_test <- cbind(dummy_df,
                       split_test_num) #combine numeric and one-hot cat df into single df
pred_xgb <- predict(fit_xgb,
                    oh_split_test)#make prediction on one-hot split test set
```

```
rmse_xgb <- Log_RMSE(oh_split_test$SalePrice,exp(pred_xgb)) #calc log RMSE
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Gradient Boosting",
                                 LogRMSE = rmse_xgb)) #add to the table
#make table
if(knitr::is_html_output()){
  knitr::kable(rmse_results, "html", caption = "RMSE Results") %>%
    kableExtra::kable_styling(bootstrap_options = "striped", full_width = FALSE)
} else{
  knitr::kable(rmse_results, "latex", booktabs = TRUE, caption = "RMSE Results") %>%
    kableExtra::kable_styling(font_size = 8, latex_options = "HOLD_position")
}
```

Table 2: RMSE Results

Method	LogRMSE
Random Forests	0.1318040
Gradient Boosting	0.1160198

Here the log RMSE has improved over random forest to 0.1160198.

Regularized Generalized Linear Model

This is essentially a penalized glm that allows for control of mixing percentage between lasso and ridge regression (alpha between 0 and 1) as well as lambda (regularization parameter) that defines the amount of shrinkage. This is controlled via the tuneLength parameter (in my case try 6 alphas and 6 lambdas). The training set is also preprocessed to center (subtract the mean), scale (divide by standard deviation) and eliminate zero variance parameters.

```
#glmnet implements a penalized glm model. Using one-hot encoded params
glm_control <- trainControl(method="cv", number = 10) #cross validation
train_glm <- train(x = oh_split_train[, !names(oh_split_train) %in% c("Id","SalePrice")],
                   y = log(oh_split_train$SalePrice),
                   method = "glmnet",
                   preProcess = c("zv","center","scale"), #preprocess zv = zero variance
                   trControl = glm_control,
                   tuneLength = 6
)

ggplot(train_glm) + ggtitle("GLM Tuning Results") #plot tuning results
```

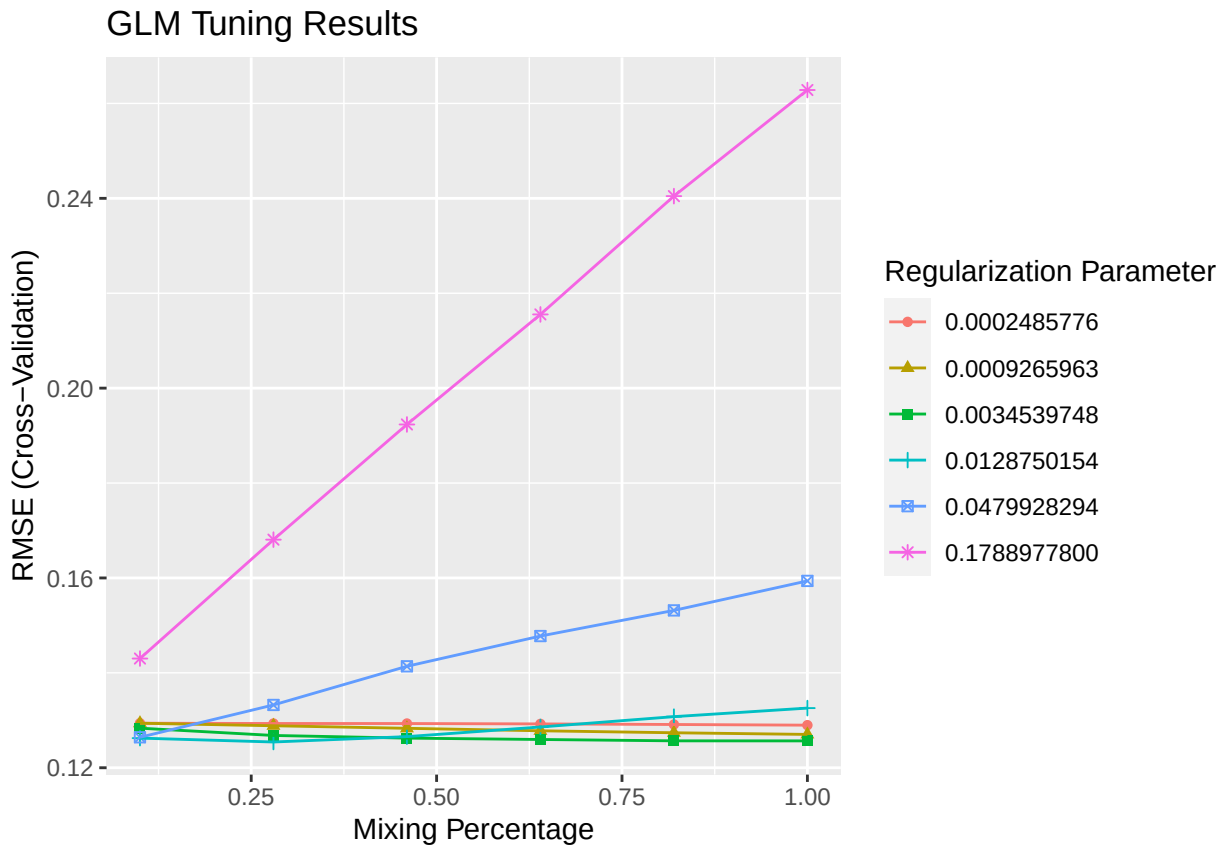

Figure 15: GLM Tuning

```
train_glm$bestTune #print best tune data
```

```
##    alpha     lambda
## 10  0.28 0.01287502
```

```
#now fit the best tune params
fit_glm <- train(x = oh_split_train[, !names(oh_split_train) %in% c("Id","SalePrice")],
                 y = log(oh_split_train$SalePrice),
                 method = "glmnet",
                 tuneGrid = expand.grid(
                   alpha = train_glm$bestTune$alpha, #mix between Lasso/Ridge regression
                                      lambda = train_glm$bestTune$lambda)
)
pred_glm <- predict(fit_glm,
                    oh_split_test) #make prediction on the one-hot split test set
rmse_glm <- Log_RMSE(oh_split_test$SalePrice,exp(pred_glm)) #calculate the log RMSE

rmse_results <- bind_rows(rmse_results,
                          tibble(Method="GLM",
                                 LogRMSE = rmse_glm)) #add to the table
#make table
if(knitr::is_html_output()){
  knitr::kable(rmse_results, "html", caption = "RMSE Results") %>%
    kableExtra::kable_styling(bootstrap_options = "striped", full_width = FALSE)
} else{
```

```
  knitr::kable(rmse_results, "latex", booktabs = TRUE, caption = "RMSE Results") %>%
    kableExtra::kable_styling(font_size = 8, latex_options = "HOLD_position")
}
```

Table 3: RMSE Results

Method	LogRMSE
Random Forests	0.1318040
Gradient Boosting	0.1160198
GLM	0.1187019

Here we see that the glm model has not necessarily improved on the gradient boosting model. Finally, let's create an ensemble of the average of all three models and compare. The ensemble prediction appears to slightly improve over the gradient boosted prediction.

Table 4: RMSE Results

Method	LogRMSE
Random Forests	0.1318040
Gradient Boosting	0.1160198
GLM	0.1187019
All Combined	0.1154840

Results

Now that we have tuned all three models and determined that the ensemble is the best predictor for sale price it's time to make final predictions on the validation set (the test.csv file). Before predicting on those a final training of each model is performed on the entire (unsplit) training set using the tuning parameters determined previously. The validation data is manipulated to match the training parameters. This includes removing NAs, reformatting the "Quality" parameters, factorizing parameters, removing unused parameters, and one-hot encoding the test data. With one-hot encoding you end up creating a new column for each level of a factor parameter. In some cases there are levels that are only seen in one data set or the other (e.g. "None" for MSZoning is only seen in the validation set). These columns are removed since these algorithms can't predict on parameters they've never seen.

Once each algorithm is trained on the entire training set the final predictions are made on the validation set. A csv file is created for submission to Kaggle. The resulting predictions came out to a log RMSE of 0.13972.

Conclusion

Data analysis and visualization showed the relationship between many of the parameters and sale price. Three algorithms were combined in an ensemble to predict the sale price of 1459 homes. The log residual mean squared error (RMSE) was used to validate the model with a final log RMSE of 0.13972. This model could be potentially be improved further by expanding the data analysis and cleaning. It's possible there are other features that could be eliminated or combined. There is a large library of algorithms that could be applied to this data so it's possible there are better ones out there as well or that a better, larger ensemble could be created.

References

1) topepo.github.io/caret/available-models.html
2) topepo.github.io/caret/train-models-by-tag.html