IDV_Learners_Ames_Kaggle_Melvin

Melvin

May 20, 2019

Introduction

House prices are one of the most unpredictable asset valuations in the world of finance. They form a major part of a country's GDP and economic valuation as most loans that are issued are real estate and student loans. Commercial banks, merchant banks, credit unions, and development banks are deeply involved in this business in one way or another and rely a lot on real estate valuation in the market in order to competitively price their mortgage rates against the millions of applicants who are seeking financing to buy their dream property. Governments also seek to understand real estate valuation via market prices in order to estimate tax revenues from capital gains taxes. Regulatory agencies want to know how exposed is the finance sector against real estate in order to draft rules for fair game within the finance industry. A poor understanding of valuation, credit liquidity risk management, and lack of necessary oversight led to the Great Recession of 2008.

The topic for this study is to predict house prices using several models namely Multivariate GLM, Principal Component Regression (PCR), and Random Forest algorithms. I am using a cleaned up dataset of the Ames Housing Prices, which is different from the original.

Method

Note:: The cleaning of the dataset was done separately and since the dataset involved 82 variables, it would be too much to include in this report. In a nutshell, the cleaning was done as follows:

- Not all null values signified absence of data. Some null values represented a category, like NA for dataset$bsmt.exposure logically refers to the known feature of a home that has no basement, instead of a missing information on the existence of a basement. Null values are supposed to carry doubt of unknown, not certainty on the existence or absence of something. Categorical NAs have been changed to reflect this.
- The original dataset was partitioned in a 50:50 manner, with the SalesPrice column removed from the test dataset and put into the sample_submission file. I have reversed that and re-partitioned the dataset into 70:30 by rejoining the SalesPrice column in sample_submission back to the test dataset, merging test and train datasets, and creating data partition at p=0.7.
- Column headers have been converted to small letters instead of capitalized initials. Also, dot separators (.) have been introduced to separate the elements of the column headers to avoid confusion.

Loading the libraries and the datasets

So first, let us load the libraries and datasets that we will be using by running the code below.

```
library(tidyverse)
```

```
## -- Attaching packages ----------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.1.0     v purrr   0.3.1
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.4.0
## v readr   1.2.1     v forcats 0.3.0

## -- Conflicts -------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(scales)
```

```
##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```

```r
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```r
library(RColorBrewer)
library(pls)
```

```
##
## Attaching package: 'pls'

## The following object is masked from 'package:corrplot':
##
##     corrplot

## The following object is masked from 'package:caret':
##
##     R2

## The following object is masked from 'package:stats':
##
##     loadings
```

```r
library(car)
```

```
## Loading required package: carData

##
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following object is masked from 'package:purrr':
##
##     some
```

```
library(grid)
library(lattice)
library(e1071)
library(ranger)
library(extraTrees)
```

```
## Loading required package: rJava
```

```
library(RRF)
```

```
## RRF 1.9

## Type rrfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'RRF'

## The following object is masked from 'package:ranger':
##
##     importance

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(Rborist)
```

```
## Loading required package: Rcpp

## Rborist 0.1-8

## Type RboristNews() to see new features/changes/bug fixes.
```

```
library(Rcpp)
library(quantreg)
```

```
## Loading required package: SparseM

##
## Attaching package: 'SparseM'

## The following object is masked from 'package:base':
##
##     backsolve
```

```r
library(quantregForest)
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following objects are masked from 'package:RRF':
##
##     classCenter, combine, getTree, grow, importance, margin,
##     MDSplot, na.roughfix, outlier, partialPlot, treesize,
##     varImpPlot, varUsed
```

```
## The following object is masked from 'package:ranger':
##
##     importance
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(doParallel)
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loading required package: iterators
```

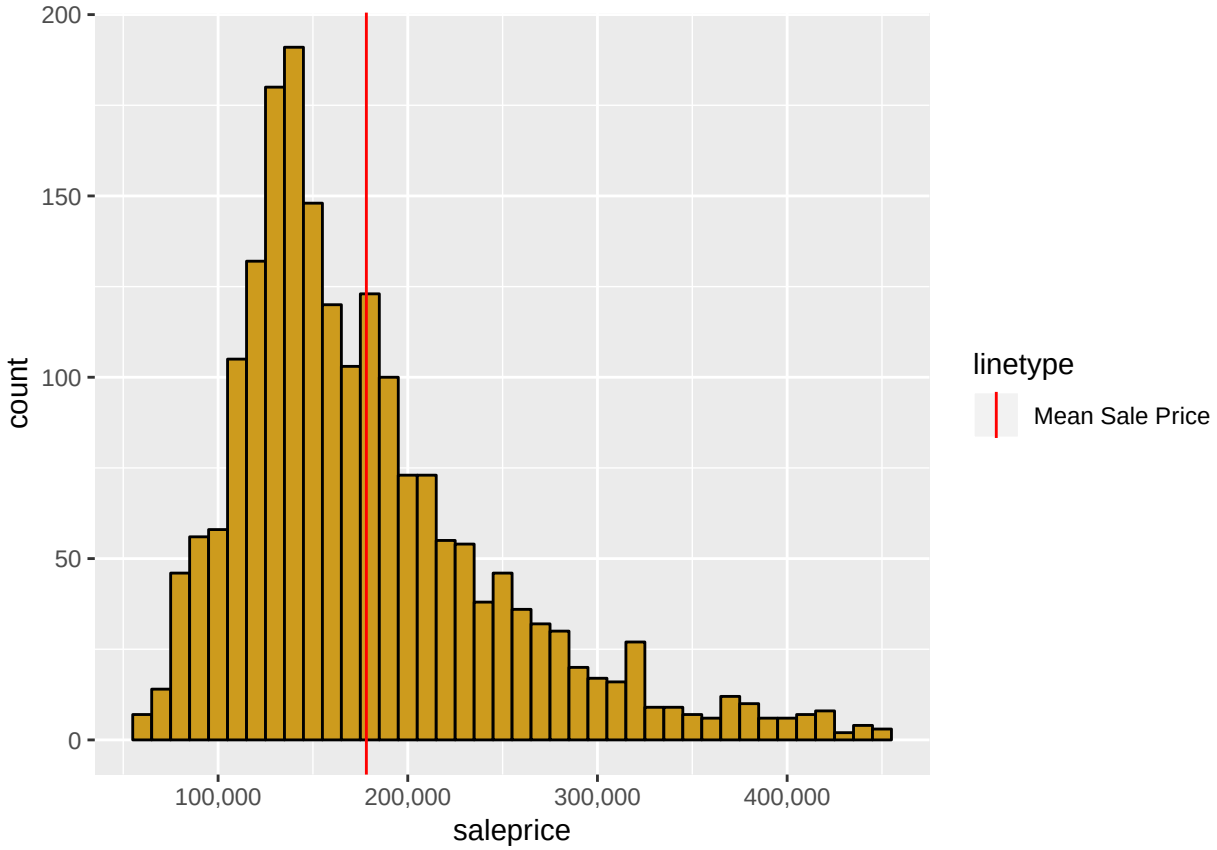
```
## Loading required package: parallel
```

```r
setwd("C:\\Users\\UserPC\\Documents\\Data Analyst Practice\\Ames_Kaggle\\Test_Run")
dataset <- read.csv("cleaned_housing.csv", header = TRUE)
dataset_train <- read.csv("train_housing.csv", header = TRUE)
dataset_test <- read.csv("test_housing.csv", header = TRUE)
```

Describing the dataset

Let us begin by running a few visualization studies on the data we have just imported. We shall start by understanding the number of houses sold according to their prices and where the mean sale price stands by coding the following below:

```
ggplot(data=dataset_train[!is.na(dataset_train$saleprice),], aes(x=saleprice)) +
  geom_histogram(color = "black", fill="goldenrod3", binwidth = 10000) +
  scale_x_continuous(breaks= seq(0, 800000, by=100000), labels = comma) +
  geom_vline(aes(xintercept = mean(saleprice), linetype = "Mean Sale Price"), color = 'red')
```



This is what we know from the mean sale price:

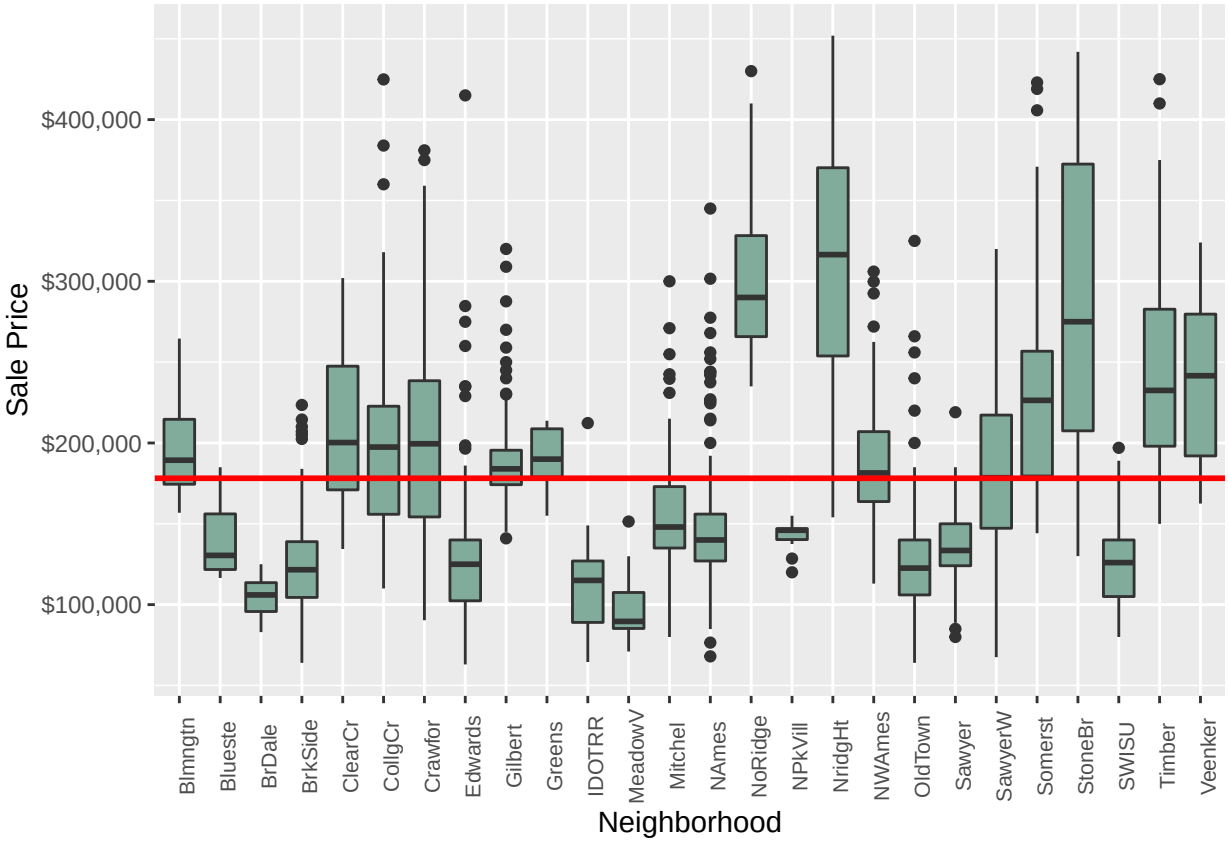
```
mean(dataset$saleprice)
```

```
## [1] 178085.6
```

Location, Location, Location

The common saying is "Location, Location, Location". This means that real estate sale prices are always dependent on the neighborhood. Bad neighborhood means bad selling prices. Let's take a deeper look at that:

```
#Sales Prices according to Neighborhood against mean prices
ggplot(dataset_train, aes(x = neighborhood, y = saleprice)) +
  geom_boxplot(fill = "#81AC9B") +
  theme(axis.text.x = element_text(angle = 90, size = 8), legend.position = "none") +
  scale_y_continuous("Sale Price", labels = dollar) +
  geom_hline(aes(yintercept=mean(saleprice)), colour='red', linetype='solid', lwd=1)+
scale_x_discrete("Neighborhood")
```



From the boxplot chart above, we see that 12 neighborhoods performed below the overall mean. Three of these neighborhoods had far reaching outliers way above the mean.

We also see that 14 neighborhoods performed above the overall mean. 6 of these neighborhoods had their first quartile below the mean.

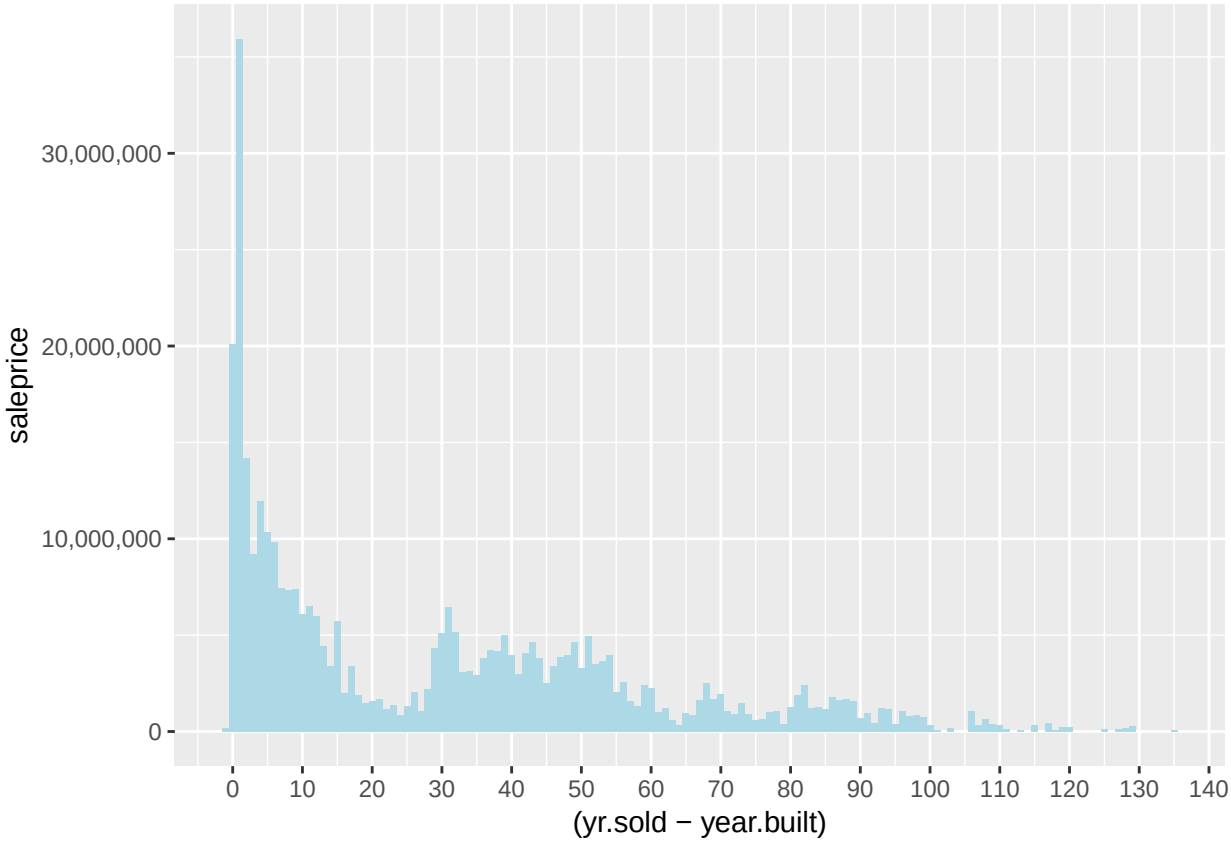
This should tell us that the idea of a neighborhood alone influencing selling prices is not well-founded. Rather, plenty of other factors contribute to the selling price of a real estate property. It also contributes to the reasons why some neighborhoods have a larger spread than others.

Besides, neighborhoods are categorical in nature. Categorical variables are meant for us to group properties together as a class of properties rather than actually using them to evaluate worth of property. Valuations exist within a given category of properties.

Age of property on year sold.

How about age of property? Supposedly, the longer you hold a property, the more valuable it is. Let's examine this paradigm:

```
#Age of homes that are being sold

ggplot(dataset_train, aes(x = (yr.sold - year.built), y=saleprice))+
  geom_bar(stat="identity", fill="light blue")+
  scale_x_continuous(breaks=seq(0,150, 10))+
  scale_y_continuous(labels = comma)
```



It appears that most of the value of the real estate that was transacted was coming from homes that were aged less than 10 years old. There is also a market for homes that were between 30 years to 50 years old.

Correlation spread

So let's take a look at what really influences the real estate market. Let's do a correlation plot. Let us use numeric variables only.

```
numericVars <- which(sapply(dataset_train, is.numeric)) #index vector numeric variables
numericVars
```

```
##    lot.frontage        lot.area    overall.qual    overall.cond
##               2               3              15              16
##      year.built  year.remod.add    mas.vnr.area    bsmtfin.sf.1
##              17              18              24              32
##    bsmtfin.sf.2     bsmt.unf.sf   total.bsmt.sf     x1st.flr.sf
##              34              35              36              41
##     x2nd.flr.sf low.qual.fin.sf     gr.liv.area  bsmt.full.bath
##              42              43              44              45
##  bsmt.half.bath       full.bath       half.bath   bedroom.abvgr
##              46              47              48              49
##   kitchen.abvgr   totrms.abvgrd      fireplaces   garage.yr.blt
##              50              52              54              57
##     garage.cars     garage.area    wood.deck.sf   open.porch.sf
##              59              60              64              65
##  enclosed.porch     x3ssn.porch    screen.porch       pool.area
##              66              67              68              69
##         mo.sold         yr.sold       saleprice ln.lot.frontage
##              73              74              77              78
##     ln.lot.area       house.age yrs.since.remod   has.2nd.floor
##              79              80              81              82
```

```r
numericVarNames <- names(numericVars) #saving names vector for use later on
numericVarNames
```

```
##  [1] "lot.frontage"    "lot.area"        "overall.qual"
##  [4] "overall.cond"    "year.built"      "year.remod.add"
##  [7] "mas.vnr.area"    "bsmtfin.sf.1"    "bsmtfin.sf.2"
## [10] "bsmt.unf.sf"     "total.bsmt.sf"   "x1st.flr.sf"
## [13] "x2nd.flr.sf"     "low.qual.fin.sf" "gr.liv.area"
## [16] "bsmt.full.bath"  "bsmt.half.bath"  "full.bath"
## [19] "half.bath"       "bedroom.abvgr"   "kitchen.abvgr"
## [22] "totrms.abvgrd"   "fireplaces"      "garage.yr.blt"
## [25] "garage.cars"     "garage.area"     "wood.deck.sf"
## [28] "open.porch.sf"   "enclosed.porch"  "x3ssn.porch"
## [31] "screen.porch"    "pool.area"       "mo.sold"
## [34] "yr.sold"         "saleprice"       "ln.lot.frontage"
## [37] "ln.lot.area"     "house.age"       "yrs.since.remod"
## [40] "has.2nd.floor"
```

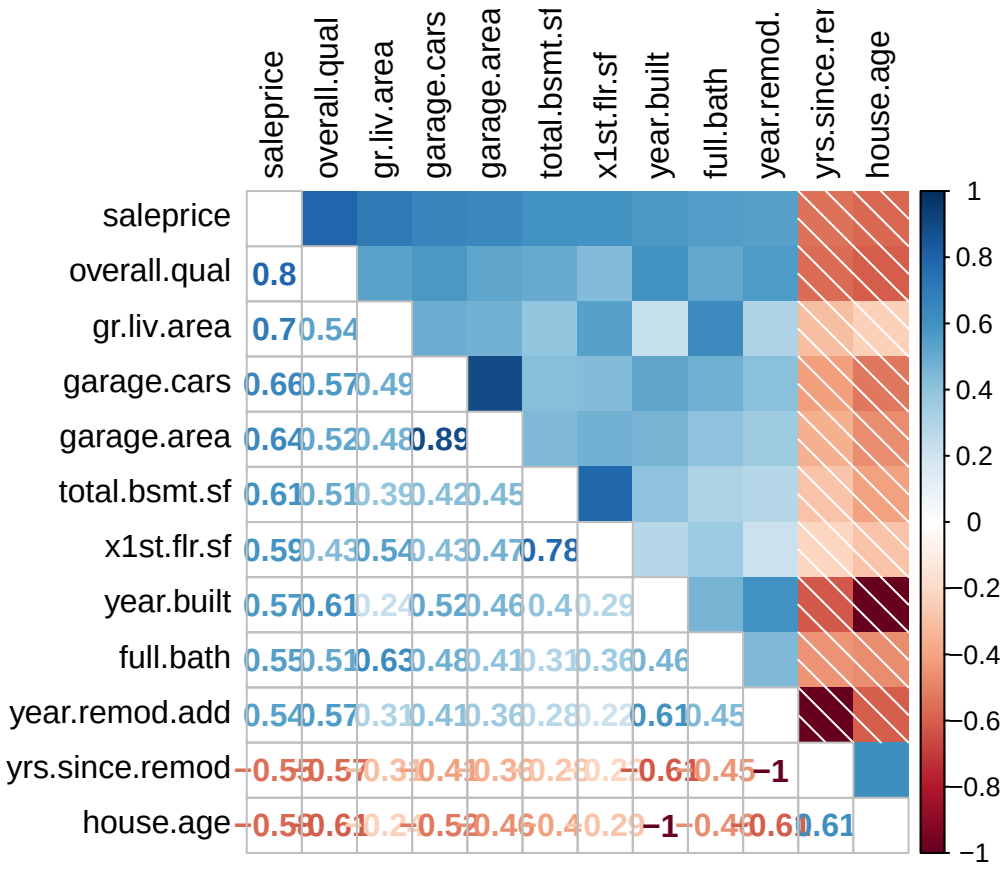
So let us now plot the correlation plot for variables that have more than corr = +/- 0.5 against a particular variable.

```r
#correlation plot
corr_train <- cor(dataset_train[,numericVars], use='pairwise.complete.obs')
cor_sorted <- as.matrix(sort(corr_train[,'saleprice'], decreasing = TRUE))
cor_sorted
```

```
##                       [,1]
## saleprice        1.00000000
## overall.qual     0.79841065
## gr.liv.area      0.70100157
## garage.cars      0.65559060
## garage.area      0.64131249
## total.bsmt.sf    0.60970494
## x1st.flr.sf      0.59335854
## year.built       0.57471600
## full.bath        0.55328647
## year.remod.add   0.54367191
## totrms.abvgrd    0.48541315
## mas.vnr.area     0.46065600
## fireplaces       0.45527662
## bsmtfin.sf.1     0.39448763
## ln.lot.area      0.36084929
## lot.frontage     0.35998774
## open.porch.sf    0.33917024
## ln.lot.frontage  0.33673860
## wood.deck.sf     0.32319478
## half.bath        0.28625346
## x2nd.flr.sf      0.28135434
## lot.area         0.26414389
## bsmt.full.bath   0.25273643
## garage.yr.blt    0.24714191
## bsmt.unf.sf      0.19266514
## bedroom.abvgr    0.14509858
## has.2nd.floor    0.11145833
## screen.porch     0.09037640
## mo.sold          0.04847734
## pool.area        0.03529182
## x3ssn.porch      0.02744831
## bsmtfin.sf.2    -0.02572072
## yr.sold         -0.03013161
## bsmt.half.bath  -0.04403115
```

```
## low.qual.fin.sf -0.05932947
## kitchen.abvgr   -0.11333687
## enclosed.porch  -0.11393768
## overall.cond    -0.13220252
## yrs.since.remod -0.54558446
## house.age       -0.57544129
```

```
#name of numeric variables above +/- 0.5 correlation
CorHigh <- names(which(apply(cor_sorted, 1, function(x) abs(x)>.5)))
cor_numVar <- corr_train[CorHigh, CorHigh]

corrplot.mixed(cor_numVar, tl.col="black", tl.pos = "lt", upper="shade")
```




```
#table of correlation name and value of correlation for those +/- 0.5 and above
trimmed_corr_train <- as.data.frame(apply(cor_sorted, 2, function(x) ifelse (abs(x) >=0.5,x,NA)))
```

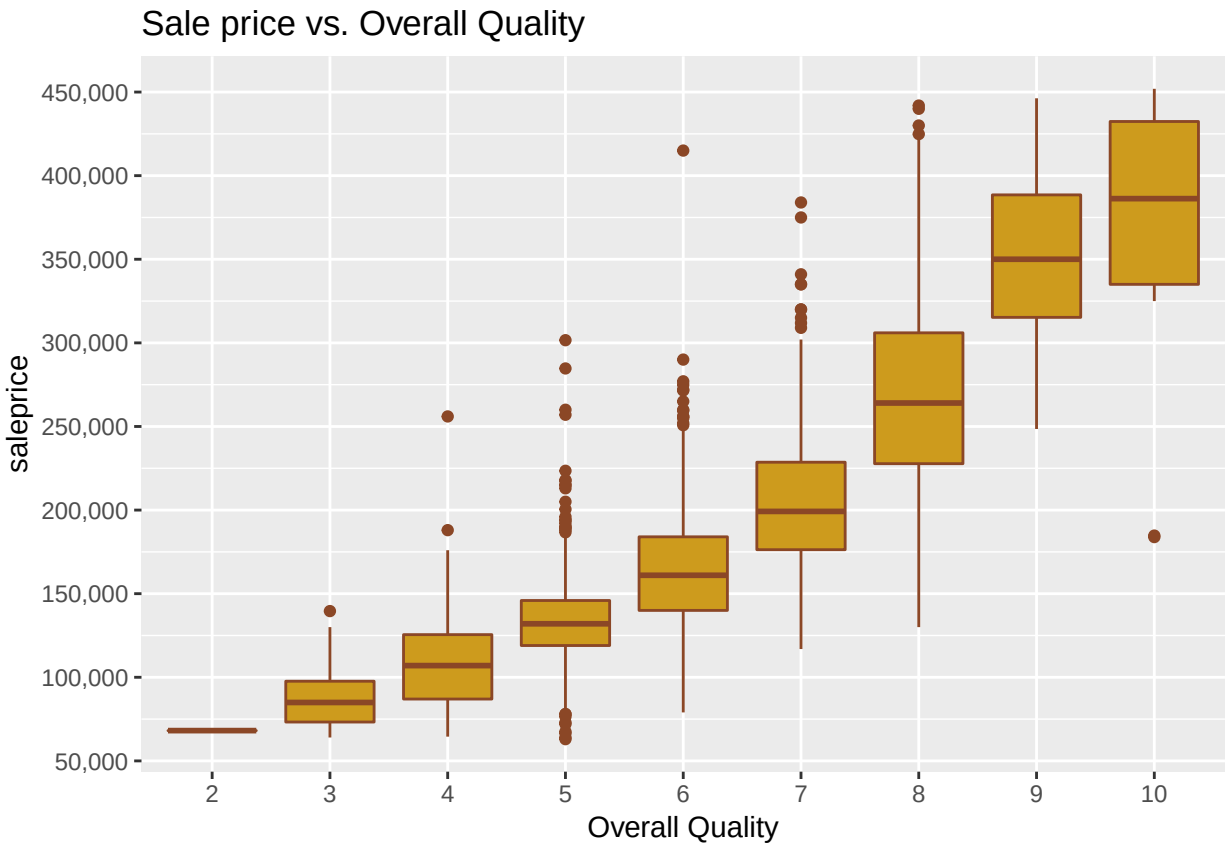
In the trimmed_corr_train table, we find that 12 elements contribute to more than a 0.5 correlation to the selling price:

```
na.omit(trimmed_corr_train)
```

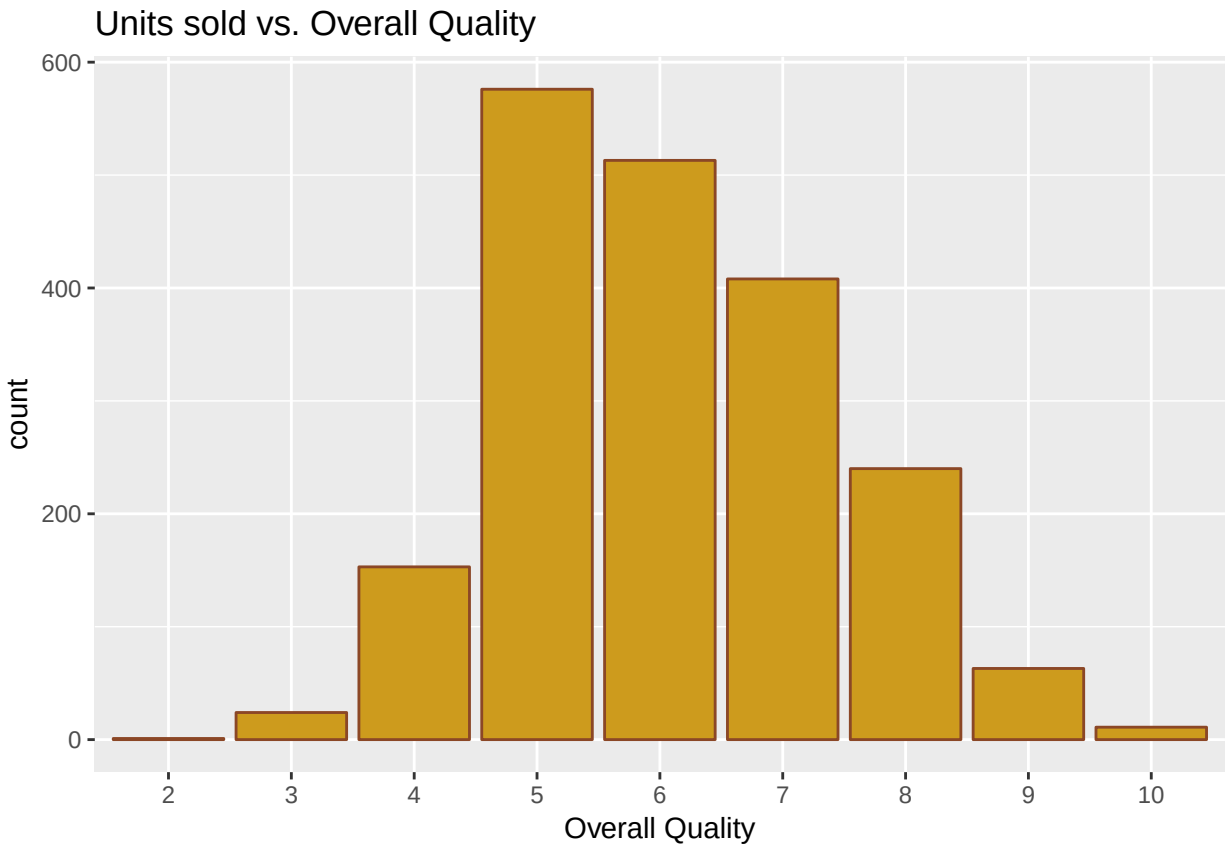
```
##                         V1
## saleprice        1.0000000
## overall.qual     0.7984107
## gr.liv.area      0.7010016
## garage.cars      0.6555906
## garage.area      0.6413125
## total.bsmt.sf    0.6097049
## x1st.flr.sf      0.5933585
## year.built       0.5747160
## full.bath        0.5532865
## year.remod.add   0.5436719
## yrs.since.remod -0.5455845
## house.age       -0.5754413
```

It looks like "overall.qual" and "gr.liv.area" are the top 2 influencers of the selling price. Let's look at a visualization of these two elements:

```
#Sale price vs. Overall Quality boxplot

ggplot(data=dataset_train[!is.na(dataset_train$saleprice),],
       aes(x=factor(overall.qual), y=saleprice)) +
  geom_boxplot(col='sienna4', fill='goldenrod3') + labs(x='Overall Quality') +
  scale_y_continuous(breaks= seq(0, 800000, by=50000), labels = comma)+
  ggtitle('Sale price vs. Overall Quality')
```



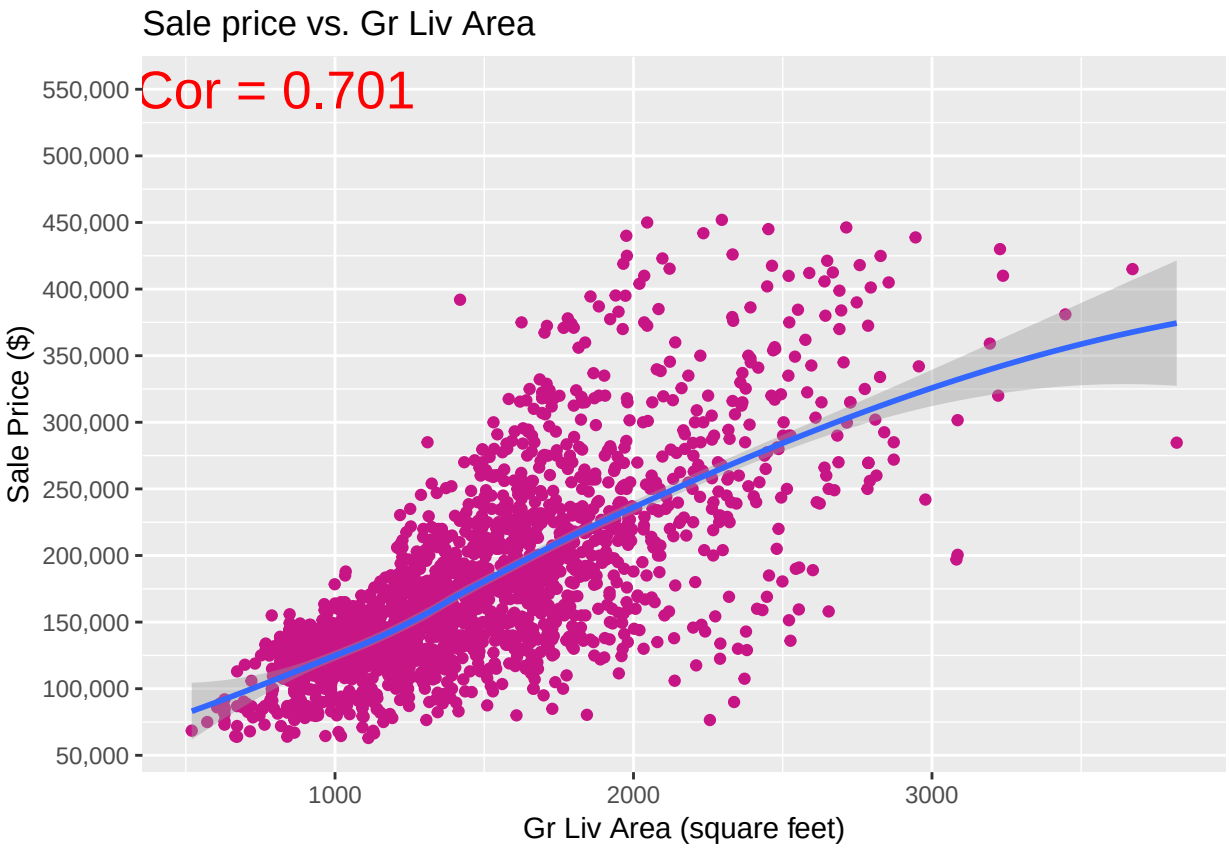

```
# Unit count vs. Overall Quality bar chart
ggplot(data=dataset_train[!is.na(dataset_train$saleprice),],
       aes(x=factor(overall.qual))) +
  geom_bar(col='sienna4', fill='goldenrod3') + labs(x='Overall Quality') +
  ggtitle('Units sold vs. Overall Quality')
```



It looks like Overall Quality really has a significant influence on the selling price as it is evident that the higher the overall quality, the higher is the command price. However, note that for mid-range quality (5-8) we see that the range is somewhat larger than the other properties sold at low-range or high-range overall quality categories. Mid-range properties demonstrate the highest amount of outliers.

The barchart "Units sold vs. Overall Quality" reflects another insight. Most of the properties transacted are mid-range and thus the existence of outliers is more likely in this range than the other ranges.

```
#Sale Price vs. Ground Living Area
ggplot(subset(dataset_train, gr.liv.area > 0 & gr.liv.area < 4000),
       aes(gr.liv.area, saleprice)) +
  geom_point(color = "mediumvioletred") + geom_smooth(method = "loess") +
  annotate("text", x = 800, y = 5.5e+05,
           label = paste('Cor =',
                         round(cor(dataset_train$gr.liv.area, dataset_train$saleprice),4)),
           size = 7, color = 'red') +
  scale_y_continuous(breaks= seq(0, 800000, by=50000), labels = comma)+
  xlab('Gr Liv Area (square feet)') +
  ylab('Sale Price ($)') +
  ggtitle('Sale price vs. Gr Liv Area')
```



The correlation point chart above demonstrates the sale price against the above ground living area. The chart shows that the greater the above ground living area, the higher is the selling price. This is especially true for homes with less than 3000 square feet of space.

Let us now move to creating predictive models for these.

Predictive Models

Generalized Linear Model (GLM)

For starters, let us work with generalized linear models against all variables first, and later against the 12 variables singled out in the trimmed_corr_train table.

Let us run the following code first to convert all numeric columns as integers:

```
# specify actual values in test set for model predictions
test_y <- dataset_test[, "saleprice"]
# convert factors to integers in train set
i <- sapply(dataset_train, is.factor)
dataset_train[i] <- lapply(dataset_train[i], as.integer)
# convert factors to integers in test set
i <- sapply(dataset_test, is.factor)
dataset_test[i] <- lapply(dataset_test[i], as.integer)
```

Now, we shall run the following code:

```
#GLM model against all variables
all.model <- glm(saleprice ~ ., data = dataset_train)
# predict on test set
pred.all <- predict(all.model, dataset_test)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
RMSE.all <- sqrt(mean((pred.all - test_y)^2))
RMSE.all
```

```
## [1] 30198.86
```

```
Results <- data_frame(Method="Multivariate GLM - All variables", RMSE = round(RMSE.all,2))
Results
```

```
## # A tibble: 1 x 2
##   Method                              RMSE
##   <chr>                              <dbl>
## 1 Multivariate GLM - All variables 30199.
```

Above is a Generalized Linear Model that is based on all the variables available in the test dataset. Before I move on, let's run a code against the 12 variables that have correlation of more than 0.5 as found in the earlier correlation plot:

```
##Linear Model for variables with correlation above 0.5
Linear_Model_Above_0.5 <- glm(data = dataset_train, saleprice ~
                                overall.qual+
                                neighborhood +
                                gr.liv.area +
                                garage.cars +
                                garage.area +
                                total.bsmt.sf +
                                x1st.flr.sf +
                                year.built +
                                full.bath +
                                year.remod.add +
                                yrs.since.remod +
                                house.age)



LM_pred <- predict(Linear_Model_Above_0.5, dataset_test)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
df_LM_pred <- data.frame(LM_pred, dataset_test$saleprice)
saleprice_MSE.1 <- mean((df_LM_pred$LM_pred - df_LM_pred$dataset_test.saleprice)^2, na.rm=TRUE)
RMSE_LM_pred <- sqrt(saleprice_MSE.1)

RMSE_LM_pred
```

```
## [1] 33967.12
```

```
Results <- bind_rows(Results,
                     data_frame(Method="Multivariate GLM - >0.5 corr variables",
                                RMSE = RMSE_LM_pred))
```

Not surprisingly, the RMSE exhibited by the two models are somewhat different.

```
Results
```

```
## # A tibble: 2 x 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Multivariate GLM - All variables     30199.
## 2 Multivariate GLM - >0.5 corr variables 33967.
```

It seems like many other factors that were seemingly having less correlational influence over the sales price were causing a reduction in the RMSE by $3,768.26. We may easily assume that the Multivariate GLM - All variables model would be a better predictor. However, let us examine the goodness of fit (GOF) in the code below:

```
#GOF for Multivariate GLM - All variables
a <- 1-(all.model$deviance/all.model$null.deviance)

#GOF for Multivariate GLM - >.05 corr variables
b <- 1-(Linear_Model_Above_0.5$deviance/Linear_Model_Above_0.5$null.deviance)

paste("Predictability Loss", " = ", round(((a-b)*100),2),"%", sep = "", collapse = NULL)
```

```
## [1] "Predictability Loss = 6.77%"
```

It appears that while our GOF fell from 0.881 to 0.813, we were not so much punished by the reduction of variables from all variables to the only 12 variables that concern us. We lost only 6.77% of predictability by reducing 70 other variables that do not contribute to the correlation.

Statistically speaking, the "Multivariate GLM - >0.5 corr variables" models make good sense.

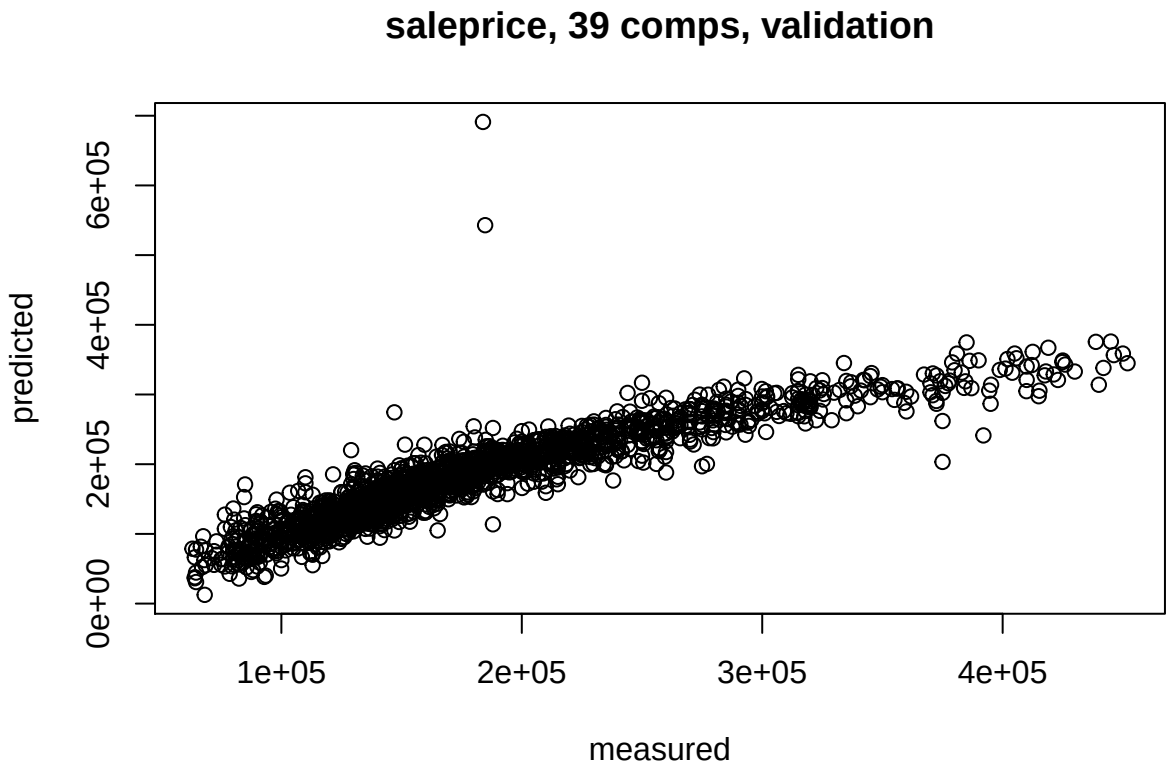
However lets explore further.

Principal Component Regression modeling

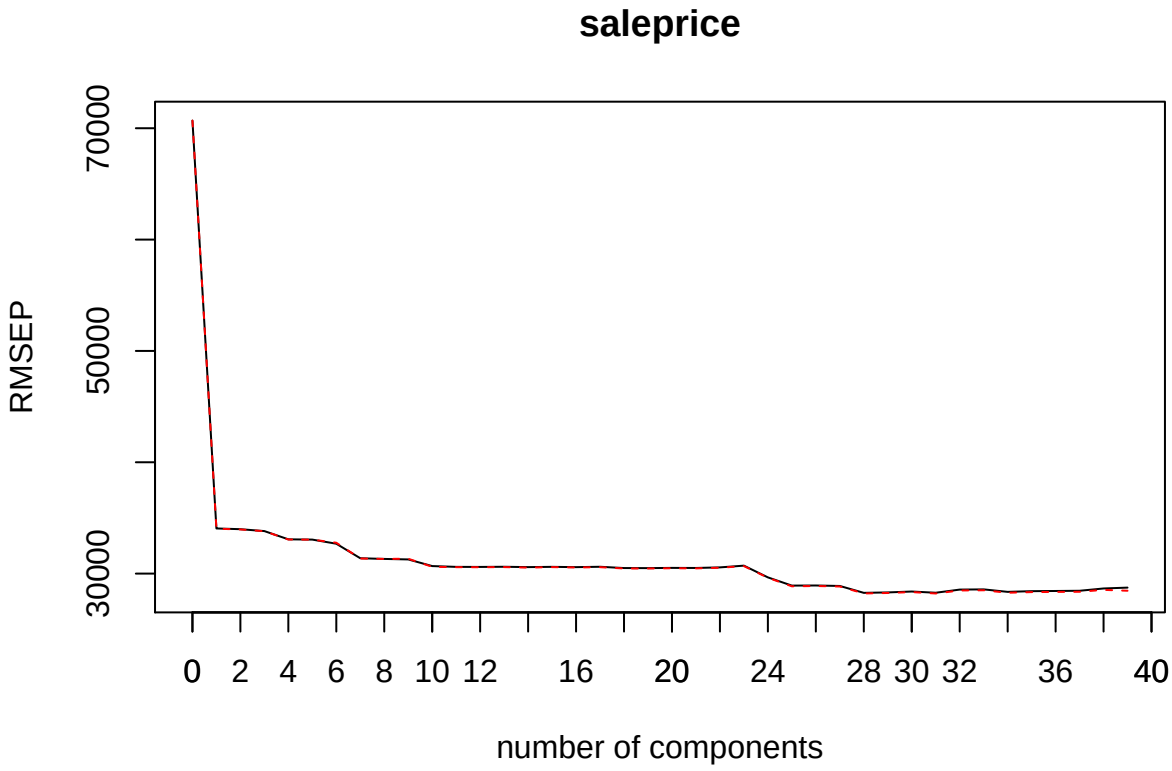
Principal Component Regression is an extension of Principal Component Analysis because it combines the concepts of linear regression to predict based on independent variables while using PCA to estimate the unknown regression coefficients.

Let us download the train and test dataset again under a different variable. Then we shall take only non-factor variables. Let us make a preliminary training to identify the number of critical components by running the codes below:

```
#Principal Component Regression
pcr_train <- read.csv("train_housing.csv", header = TRUE)
pcr_train_num <- pcr_train
pcr_test <- read.csv("test_housing.csv", header = TRUE)

# drop factors
for(i in colnames(pcr_train)){
  if(is.factor( pcr_train[[i]] )){
    pcr_train_num[[i]] <- NULL
  }
}


#General PCR modeling against all variables
pcr_saleprice <- pcr(saleprice ~ ., data=pcr_train_num, scale = TRUE, validation = "CV")

plot(pcr_saleprice)
```




```
#finding component values by plots
plot(pcr_saleprice,"validation")
axis(1, at=seq(0, 40, 2))
```



We see clearly that the training RMSE drops significantly at 1 component, 7 components, and 24 components. Let us use the following identified components as our PCR models.

PCR - 1 Component

```
# predict RMSE pcr1
pcr_predict_1 <- predict(pcr_saleprice, pcr_test, ncomp=1)
pcr_predict_1 <- data.frame(pcr_predict_1, pcr_test$saleprice)
saleprice_MSE_1 <- mean((pcr_predict_1$saleprice.1.comps - pcr_predict_1$pcr_test.saleprice)^2)
pcr_RMSE_1 <- sqrt(saleprice_MSE_1)
pcr_RMSE_1
```

```
## [1] 36420.37
```

```
Results <- bind_rows(Results,
                     data_frame(Method="PCR - 1 component",
                                RMSE = pcr_RMSE_1))

Results
```

```
## # A tibble: 3 x 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Multivariate GLM - All variables     30199.
## 2 Multivariate GLM - >0.5 corr variables 33967.
## 3 PCR - 1 component                    36420.
```

PCR - 7 Component

```
# predict RMSE pcr2
pcr_predict_2 <- predict(pcr_saleprice, pcr_test, ncomp=7)
pcr_predict_2 <- data.frame(pcr_predict_2, pcr_test$saleprice)
saleprice_MSE_2 <- mean((pcr_predict_2$saleprice.7.comps - pcr_predict_2$pcr_test.saleprice)^2)
pcr_RMSE_2 <- sqrt(saleprice_MSE_2)
pcr_RMSE_2
```

```
## [1] 33924.24
```

```
Results <- bind_rows(Results,
                     data_frame(Method="PCR - 7 component",
                                RMSE = pcr_RMSE_2))

Results
```

```
## # A tibble: 4 x 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Multivariate GLM - All variables     30199.
## 2 Multivariate GLM - >0.5 corr variables 33967.
## 3 PCR - 1 component                    36420.
## 4 PCR - 7 component                    33924.
```

PCR - 24 Component

```
# predict RMSE pcr3
pcr_predict_3 <- predict(pcr_saleprice, pcr_test, ncomp=24)
pcr_predict_3 <- data.frame(pcr_predict_3, pcr_test$saleprice)
saleprice_MSE_3 <- mean((pcr_predict_3$saleprice.24.comps - pcr_predict_3$pcr_test.saleprice)^2)
pcr_RMSE_3 <- sqrt(saleprice_MSE_3)
pcr_RMSE_3
```

```
## [1] 33344.96
```

```
Results <- bind_rows(Results,
                     data_frame(Method="PCR - 24 component",
                                RMSE = pcr_RMSE_3))
```

So let's take a look at the results of the RMSE:

```
Results
```

```
## # A tibble: 5 x 2
##   Method                                  RMSE
##   <chr>                                  <dbl>
## 1 Multivariate GLM - All variables       30199.
## 2 Multivariate GLM - >0.5 corr variables 33967.
## 3 PCR - 1 component                      36420.
## 4 PCR - 7 component                      33924.
## 5 PCR - 24 component                     33345.
```

It seems like between PCR - 7 component and PCR - 24 component, the difference is marginal. Rather, if we take PCR - 24 component, we actually risk overfitting the data with more than 3 times more component with less than 1.7% improvement.

However, with all these ambiguity, lets take a look at the new model, Random forest.

Random Forest Models

Until now, we have a concern of overfitting from the two models we have explored so far. We are not confident on the extent of the overfitting that we have because we are constantly fooled by the statement that the lower the test RMSE, the better it is.

Therefore, I will introduce Random Forest modeling, which is an ensemble of decision trees that reduces overfitting to a great extent. For most of the time, we do not need to worry about overfitting in such a situation.

The RF models that we are exploring are as follows:

1. Quantile Random Forest (QRF)
2. Random Forest (ranger, Rborist, rf)
3. Conditional Inference Random Forest (cforest)
4. Parallel Random Forest (parRF)

Parallel Processing

In this section, we will also introduce multicore processing, otherwise also known as parallel processing. I am using an Intel Coffee Lake i3-8100 quadcore processor, which means I have up to four cores on which I can delegate the computational workload to be worked on by the CPU. This has absolutely nothing to do with the Parallel Random Forest which I will be trying on.

To start, let us convert all factor variables as integers.

```
# make a dataframe of all numeric features
num_features = names(which(sapply(dataset_train, is.numeric)))
df_numeric = dataset_train[num_features]
# make a dataframe of all categorical features
cat_features = names(which(sapply(dataset_train, is.factor)))
# convert all categorical features to numeric variables for modeling
dataset_train[cat_features] <- sapply(dataset_train[cat_features], as.integer)
dataset_test[cat_features] <- sapply(dataset_test[cat_features], as.integer)
# split test into x and y
test_x <- dataset_test %>% dplyr::select(-saleprice)
test_y <- dataset_test %>% dplyr::select(saleprice)
```

Next, we will choose repeating cross-validation as our tuning control when we train our random forest. This is to reduce the likelihood of overfitting.

Then, we run the following training models consecutively in a chunk. This is because the chunk itself functions on parallel processing and should not be separated:

```
# set seed for reproducibility
set.seed(256)
##myFolds <- createFolds(dataset_train, k = 5)
control <- trainControl(method = "repeatedcv", repeats = 3, number = 10)
cl <- makePSOCKcluster(4)
registerDoParallel(cl)

#Conditional Inference Random Forest (cforest)
set.seed(123)

cforest_model <- train(saleprice ~ .,
        method = "cforest",
        data = dataset_train,
        trControl = control)

pred_cforest <- predict(cforest_model, dataset_test)
df_pred_cforest <- data.frame(pred_cforest, dataset_test$saleprice)
saleprice_cforest <- mean((df_pred_cforest$pred_cforest - df_pred_cforest$dataset_test.saleprice)^2)
cforest_RMSE <- sqrt(saleprice_cforest)
Results <- bind_rows(Results,
          data_frame(Method="cforest",
                     RMSE = cforest_RMSE))

#QRF
set.seed(123)

qrf_model <- train(saleprice ~ .,
                   method = "qrf",
                   data = dataset_train,
                   trControl = control)

pred_qrf <- predict(qrf_model, dataset_test)
df_pred_qrf <- data.frame(pred_qrf, dataset_test$saleprice)
saleprice_qrf <- mean((df_pred_qrf$pred_qrf - df_pred_qrf$dataset_test.saleprice)^2)
qrf_RMSE <- sqrt(saleprice_qrf)
Results <- bind_rows(Results,
                     data_frame(Method="Quantile Random Forest (QRF)",
                                RMSE = qrf_RMSE))


#ranger
set.seed(123)

ranger_model <- train(saleprice ~.,
                   method = "ranger",
                   data = dataset_train,
                   trControl = control)

pred_ranger <- predict(ranger_model, dataset_test)
df_pred_ranger <- data.frame(pred_ranger, dataset_test$saleprice)
saleprice_ranger <- mean((df_pred_ranger$pred_ranger - df_pred_ranger$dataset_test.saleprice)^2)
ranger_RMSE <- sqrt(saleprice_ranger)
Results <- bind_rows(Results,
          data_frame(Method="Random Forest (Ranger)",
                     RMSE = ranger_RMSE))
```

```
#Rborist
set.seed(123)

Rborist_model <- train(saleprice ~.,
                       method = "Rborist",
                       data = dataset_train,
                       trControl = control)

pred_Rborist <- predict(Rborist_model, dataset_test)
df_pred_Rborist <- data.frame(pred_Rborist, dataset_test$saleprice)
saleprice_Rborist <- mean((df_pred_Rborist$pred_Rborist - df_pred_Rborist$dataset_test.saleprice)^2)
Rborist_RMSE <- sqrt(saleprice_Rborist)
Results <- bind_rows(Results,
          data_frame(Method="Random Forest (Rborist)",
                     RMSE = Rborist_RMSE))

#RF
set.seed(123)

RF_model <- train(saleprice ~ .,
                  method = "rf",
                  data = dataset_train,
                  trControl = control)

pred_RF <- predict(RF_model, dataset_test)
df_pred_RF <- data.frame(pred_RF, dataset_test$saleprice)
saleprice_RF <- mean((df_pred_RF$pred_RF - df_pred_RF$dataset_test.saleprice)^2)
RF_RMSE <- sqrt(saleprice_RF)
Results <- bind_rows(Results,
          data_frame(Method="Random Forest (RF)",
                     RMSE = RF_RMSE))


#parRF
set.seed(123)

parRF_model <- train(saleprice ~ .,
                  method = "parRF",
                  data = dataset_train,
                  trControl = control)

pred_parRF <- predict(parRF_model, dataset_test)
df_pred_parRF <- data.frame(pred_parRF, dataset_test$saleprice)
saleprice_parRF <- mean((df_pred_parRF$pred_parRF - df_pred_parRF$dataset_test.saleprice)^2)
parRF_RMSE <- sqrt(saleprice_parRF)
Results <- bind_rows(Results,
          data_frame(Method="Parallel Random Forest (parRF)",
                     RMSE = parRF_RMSE))

stopCluster(cl)
```

Let us view the results:

```
Results
```

```
## # A tibble: 11 x 2
##    Method                                  RMSE
##    <chr>                                  <dbl>
##  1 Multivariate GLM - All variables       30199.
##  2 Multivariate GLM - >0.5 corr variables 33967.
```

```
##  3 PCR - 1 component                     36420.
##  4 PCR - 7 component                     33924.
##  5 PCR - 24 component                    33345.
##  6 cforest                               24007.
##  7 Quantile Random Forest (QRF)          22555.
##  8 Random Forest (Ranger)                22122.
##  9 Random Forest (Rborist)               22126.
## 10 Random Forest (RF)                    22192.
## 11 Parallel Random Forest (parRF)        22282.
```

Results

It appears evident at this point that random forests are producing the better of the three models that have been tested against the Ames Housing Prices dataset. The R packages of Ranger, Rborist, and RF are yielding competitively similar results while Conditional Inference RF, Parallel RF, and Quantile RF yield their own results.

Choosing Random Forest models seem to be the most suited for real estate price prediction.

Limitations

Bear in mind that the results of this prediction is valid only for this dataset. Datasets coming from other cities, states, or countries will yield different results and will require retraining of the dataset using other models of prediction. It may or may not arrive at the same conclusion as this one.

Conclusion

As a conclusion, we can see the flaws of several models in the beginnig of this study. Generalized Linear Models require a linear relationship between independent variables and the dependent variables. It also assumes that it is more reliable when a normal distribution exists. Neither of these two can be established regarding the distribution of sale prices or its relationship with other variables. More so, it is prone to overfitting.

PCR is not suitable because it is a far more complicated algorithm that combines both PCA and linear regression models, but it yields almost the same test RMSE as the Multivariate GLM for variables above 0.5 correlation.

Random Forest models remain the most suitable as they demonstrate approximately 33% improvement in the test RMSE results.