Kaggle Ames Housing Dataset Problem

Luke Kim

1. Introduction

Kaggle is a competitive platform for people interested in data science, and according to my experience it works as follows: For a particular problem, we download its dataset (we have training and test data sets) and construct a model which we then train on the training data set. We then make our predictions using the test data set. We then submit our prediction as a .csv file and the kaggle computes the rank of your prediction by using some statistical metric. In this particular, they seem to be using the root mean squared error (i.e. the smaller our error, the higher our rank).

The following analysis has been written in R markdown, and you are viewing the knitted pdf version. The Rmd file is also posted under the same Github repository.

This is the link to the competition: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview

2. First Attempt using Linear Regression

The first thing I instinctively tried was to fit a multiple linear regression model on the training data. After looking over the data set, I identified a set of variables such as YearBuilt, YearRemodAdd, MSSubClass, OverallQual, OverallCond, GrLivArea, LotArea, BedroomAbvGr, GarageArea, MasVnrArea, TotRmsAbvGrd, Neighborhood, KitchenQual as predictors to predict the response variable, which is SalePrice. These predictor variables were chosen after careful trial and error as well as using common-sense (for instance, we think it is reasonable that the Neighborhood information would be important in determining house sales prices). Also, during the trial and error step, there seemed to be some variables that are strongly correlated, such as X1stFlrSf (square feet area of first floor) and X2ndFlrSf (square feet area of second floor) which actually affected the result of the prediction.

```r
library(tidyr)
house_train <- read.csv('./train.csv')
lm.fit <- lm(SalePrice~YearBuilt+YearRemodAdd+MSSubClass+
                    OverallQual+OverallCond+GrLivArea+LotArea+
                    BedroomAbvGr+GarageArea+MasVnrArea+TotRmsAbvGrd+
                    Neighborhood+KitchenQual,data = house_train)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = SalePrice ~ YearBuilt + YearRemodAdd + MSSubClass +
##     OverallQual + OverallCond + GrLivArea + LotArea + BedroomAbvGr +
##     GarageArea + MasVnrArea + TotRmsAbvGrd + Neighborhood + KitchenQual,
##     data = house_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -406902  -13376    -978   11446  252150
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -9.824e+05  1.611e+05  -6.097 1.39e-09 ***
## YearBuilt            4.183e+02  6.927e+01   6.039 1.98e-09 ***
## YearRemodAdd         9.335e+01  6.520e+01   1.432 0.152447
## MSSubClass          -2.161e+02  2.397e+01  -9.016  < 2e-16 ***
## OverallQual          1.267e+04  1.125e+03  11.265  < 2e-16 ***
## OverallCond          5.704e+03  9.590e+02   5.948 3.42e-09 ***
## GrLivArea            5.115e+01  3.683e+00  13.886  < 2e-16 ***
## LotArea              6.465e-01  9.657e-02   6.695 3.10e-11 ***
## BedroomAbvGr        -6.055e+03  1.590e+03  -3.809 0.000145 ***
## GarageArea           2.786e+01  5.451e+00   5.112 3.62e-07 ***
## MasVnrArea           2.096e+01  5.791e+00   3.619 0.000307 ***
## TotRmsAbvGrd         2.430e+03  1.116e+03   2.177 0.029629 *
## NeighborhoodBlueste -8.983e+03  2.414e+04  -0.372 0.709910
## NeighborhoodBrDale  -2.039e+04  1.174e+04  -1.737 0.082684 .
## NeighborhoodBrkSide -3.886e+03  1.020e+04  -0.381 0.703359
## NeighborhoodClearCr  4.954e+03  1.064e+04   0.466 0.641611
## NeighborhoodCollgCr -5.142e+03  8.554e+03  -0.601 0.547832
## NeighborhoodCrawfor  2.085e+04  1.004e+04   2.077 0.037944 *
## NeighborhoodEdwards -1.742e+04  9.305e+03  -1.872 0.061346 .
## NeighborhoodGilbert -9.297e+03  8.912e+03  -1.043 0.297008
## NeighborhoodIDOTRR  -1.696e+04  1.070e+04  -1.584 0.113352
## NeighborhoodMeadowV -1.373e+03  1.162e+04  -0.118 0.906015
## NeighborhoodMitchel -8.407e+03  9.539e+03  -0.881 0.378259
## NeighborhoodNAmes   -1.107e+04  9.036e+03  -1.225 0.220912
## NeighborhoodNoRidge  4.996e+04  9.959e+03   5.016 5.94e-07 ***
## NeighborhoodNPkVill  2.664e+03  1.351e+04   0.197 0.843667
## NeighborhoodNridgHt  3.654e+04  9.011e+03   4.055 5.28e-05 ***
## NeighborhoodNWAmes  -1.559e+04  9.326e+03  -1.672 0.094777 .
## NeighborhoodOldTown -1.854e+04  9.875e+03  -1.877 0.060672 .
## NeighborhoodSawyer  -9.767e+03  9.493e+03  -1.029 0.303726
## NeighborhoodSawyerW -5.818e+03  9.173e+03  -0.634 0.525985
## NeighborhoodSomerst  5.860e+03  8.727e+03   0.671 0.502050
## NeighborhoodStoneBr  5.523e+04  1.025e+04   5.390 8.27e-08 ***
## NeighborhoodSWISU   -1.023e+04  1.148e+04  -0.891 0.373087
## NeighborhoodTimber   4.395e+03  9.804e+03   0.448 0.654014
## NeighborhoodVeenker  2.656e+04  1.267e+04   2.096 0.036231 *
## KitchenQualFa       -4.014e+04  7.364e+03  -5.451 5.90e-08 ***
## KitchenQualGd       -4.070e+04  4.050e+03 -10.049  < 2e-16 ***
## KitchenQualTA       -4.311e+04  4.703e+03  -9.168  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 31960 on 1413 degrees of freedom
##   (8 observations deleted due to missingness)
## Multiple R-squared:  0.8418, Adjusted R-squared:  0.8375
## F-statistic: 197.9 on 38 and 1413 DF,  p-value: < 2.2e-16
```

Most of these predictors were statistically significant with very small p-values, and the Multiple R-squared value is 0.8418. Furthermore, the p-value for the F-statistic is < 2.2e-16 which suggests that the predictors are good at explaining the variability in our response.

After training the model, we have to fit this model on our test data to get predictions. I encountered a major problem though - there was some missing data provided by Kaggle and they were denoted as NA. Kaggle does not accept a solution with missing rows (it checks if the number of rows of our prediction is equal to the number of rows in the test data table). This means that we cannot just simply get rid of rows with NA values. Fortunately, there was only one missing value for the qualitative variable KitchenQual at row 96, and because KitchenGrade (which is a predictor we did not use in our model yet) had a value of 1 in that row and most of the houses with KitchenGrade equal to 1 had a KitchenQual level of TA of GD, I decided to assign TA to this value. There were some quantitative values missing too, and to get around this I decided to assign some negative value (chose -1) to fill all NA values because the quantitative variables all had a positive value, so it would be possible to differentiate the NAs with some negative value. Below is an R code I wrote to deal with the present NA values and to obtain my prediction.

```r
library(tidyverse) # utility functions
house_test <- read.csv('./test.csv')
v <- c('YearBuilt','YearRemodAdd','MSSubClass','OverallQual',
              'OverallCond','GrLivArea','LotArea','BedroomAbvGr',
              'GarageArea','MasVnrArea','TotRmsAbvGrd')
temp <- house_test
for (i in v){
  temp[,i][is.na(temp[,i])] <- -1
}
temp[96,'KitchenQual'] <- factor("TA") # randomly decided to assign TA
lm.prediction <- predict(lm.fit,temp)
sum(is.na(lm.prediction)) # check if there are any NA valued rows in our prediction
```

```
## [1] 0
```

```r
submission <- data_frame('Id' = house_test$Id, 'SalePrice' = lm.prediction)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```r
write_csv(submission,'submission.csv')
```

To improve the model, I could examine adding more predictors, checking for more colinearity between predictors (or use penalizing models with L1 and L2 regularization such as Ridge Regression or Lasso), or changing the way I deal with missing values. Another option would be to test if other forms of regression may work, such as polynomial regression. My current error is pretty low at around 0.16 (definitely not bad at all), but we have much more room for improvement. The first thing to do is to conduct data imputation i.e. deal with all the missing values. Then we'll see if we will stick to linear regression or use a different model.

3. Data imputation

The first thing I realised is that we have to drop the Id column, because that column is also being used for training our model, and the Id column is just continuous observation number and we don't need it for prediction. In fact, I'm certain that the Id column, when not removed, will make our prediction worse. Let's write code to get rid of the Id columns.

```r
library(tidyr)
library(tidyverse)
library(gbm)
library(glmnet)
library(rpart)
library(caret)

house_train <- read.csv('./train.csv')
house_test <- read.csv('./test.csv')
house_train$Id <- NULL
IdList <- house_test$Id # save Id List for submission.
house_test$Id <- NULL
```

Note that we could merge the train and test data set for the data cleaning process, but let's just deal with each of them seperately for now.

We then convert some integer encoded values to factors. Using factors for categorical variables may be more useful for modeling later on. This is because categorical variables enter into statistical models differently than continuous variables, and storing data as factors ensures that the modeling functions will treat such data correctly.

```
house_train$OverallQual<-as.factor(house_train$OverallQual)
house_train$OverallCond<-as.factor(house_train$OverallCond)

house_test$OverallQual<-as.factor(house_test$OverallQual)
house_test$OverallCond<-as.factor(house_test$OverallCond)
```

We will now fill in missing data for both the train and test datasets. There are actually many missing data, and we will be using various methods (wild guesses, using statistical model to predict missing data, and general common sense) to deal with the NA issue.

Data imputation for train set

We fill up the missing information for the train data. Let's first examine what missing data we have.

```
train_na_list <- colnames(house_train)[colSums(is.na(house_train)) > 0]
train_na_list
```

```
##  [1] "LotFrontage"  "Alley"        "MasVnrType"   "MasVnrArea"   "BsmtQual"
##  [6] "BsmtCond"     "BsmtExposure" "BsmtFinType1" "BsmtFinType2" "Electrical"
## [11] "FireplaceQu"  "GarageType"   "GarageYrBlt"  "GarageFinish" "GarageQual"
## [16] "GarageCond"   "PoolQC"       "Fence"        "MiscFeature"
```

We use the tree model to predict the missing values for LotFrontage

```
LotFrontage.tree <- rpart(LotFrontage~LotArea+BldgType+Neighborhood+LotConfig,data=house_train)

house_train[is.na(house_train$LotFrontage),]$LotFrontage <- predict(LotFrontage.tree,newdata=house_trai
```

We fill up the missing values in Alley with "Not Available"

```
house_train$Alley <- as.character(house_train$Alley)
house_train$Alley[is.na(house_train$Alley)] <- "Not Available"
house_train$Alley <- as.factor(house_train$Alley)
```

For Fence, NA simply means no fence.

```
house_train$Fence <- as.character(house_train$Fence)
house_train$Fence[is.na(house_train$Fence)] <- "No Fence"
house_train$Fence <- as.factor(house_train$Fence)
```

For MasVnrType and MasVnrArea, we impute the missing values. It is weird that even if MasVnrType is None, MasVnrArea > 0. But because this is how the data is prepared, let's accept this fact. I'll try to fill in missing MasVnrType as None and missing MasVnrArea simply to zero.

```
house_train$MasVnrType <- as.character(house_train$MasVnrType)
house_train$MasVnrType[is.na(house_train$MasVnrType)] <- "None"
house_train$MasVnrType <- as.factor(house_train$MasVnrType)

house_train$MasVnrArea[is.na(house_train$MasVnrArea)] <- 0
```

For BsmtQual and BsmtCond, missing values simply mean that the basement does not exist.

```r
house_train$BsmtQual <- as.character(house_train$BsmtQual)
house_train$BsmtQual[is.na(house_train$BsmtQual)] <- "NA"
house_train$BsmtQual <- as.factor(house_train$BsmtQual)

house_train$BsmtCond <- as.character(house_train$BsmtCond)
house_train$BsmtCond[is.na(house_train$BsmtCond)] <- "NA"
house_train$BsmtCond <- as.factor(house_train$BsmtCond)
```

All the missing values in BsmtExposure, BsmtFinType1, BsmtFinType2 are equivalent to “No Basement”. So we can impute the missing values as NA.

```r
house_train$BsmtExposure <- as.character(house_train$BsmtExposure)
house_train$BsmtExposure[is.na(house_train$BsmtExposure)] <- "NA"
house_train$BsmtExposure <- as.factor(house_train$BsmtExposure)

house_train$BsmtFinType1 <- as.character(house_train$BsmtFinType1)
house_train$BsmtFinType1[is.na(house_train$BsmtFinType1)] <- "NA"
house_train$BsmtFinType1 <- as.factor(house_train$BsmtFinType1)

house_train$BsmtFinType2 <- as.character(house_train$BsmtFinType2)
house_train$BsmtFinType2[is.na(house_train$BsmtFinType2)] <- "NA"
house_train$BsmtFinType2 <- as.factor(house_train$BsmtFinType2)
```

```r
sum(is.na(house_train$Electrical))
```

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

```r
summary(house_train$Electrical)
```

```
## FuseA FuseF FuseP   Mix SBrkr  NA's
##    94    27     3     1  1334     1
```

There appears to be only one Electrical with NA. We could either get rid of this data point, or assign it to something else. My choice would be to assign it to SBrkr, because most of the Electricals are assigned to the SBrkr category.

```r
house_train$Electrical <- as.character(house_train$Electrical)
house_train$Electrical[is.na(house_train$Electrical)] <- "SBrkr"
house_train$Electrical <- as.factor(house_train$Electrical)
```

NA for Fireplace simply means no fireplace. So we just replace the missing values to the categorical variable NA

```r
house_train$FireplaceQu <- as.character(house_train$FireplaceQu)
house_train$FireplaceQu[is.na(house_train$FireplaceQu)] <- "NA"
house_train$FireplaceQu <- as.factor(house_train$FireplaceQu)
```

Replace missing values to categorical NA for GarageType, GarageCond, GarageQual, GarageFinish

```r
house_train$GarageType <- as.character(house_train$GarageType)
house_train$GarageType[is.na(house_train$GarageType)] <- "NA"
house_train$GarageType <- as.factor(house_train$GarageType)

house_train$GarageCond <- as.character(house_train$GarageCond)
house_train$GarageCond[is.na(house_train$GarageCond)] <- "NA"
house_train$GarageCond <- as.factor(house_train$GarageCond)

house_train$GarageQual <- as.character(house_train$GarageQual)
```

```
house_train$GarageQual[is.na(house_train$GarageQual)] <- "NA"
house_train$GarageQual <- as.factor(house_train$GarageQual)

house_train$GarageFinish <- as.character(house_train$GarageFinish)
house_train$GarageFinish[is.na(house_train$GarageFinish)] <- "NA"
house_train$GarageFinish <- as.factor(house_train$GarageFinish)
```

Next is the GarageYrBlt. We could approximate the GarageYrBlt to be simlar to the YearBuilt (the year the house was built). This is not always the case (the house could have first been built without a garage then later on a garage was added), but this is a reasonable assumption to make.

```
for (i in 1:nrow(house_train)){
  if (is.na(house_train[i,'GarageYrBlt'])){
    house_train[i,'GarageYrBlt'] <- house_train[i,'YearBuilt']
  }
}
```

Lastly, replace the NA values of PoolQC and MiscFeature to a categorical NA

```
house_train$PoolQC <- as.character(house_train$PoolQC)
house_train$PoolQC[is.na(house_train$PoolQC)] <- "NA"
house_train$PoolQC <- as.factor(house_train$PoolQC)

house_train$MiscFeature <- as.character(house_train$MiscFeature)
house_train$MiscFeature[is.na(house_train$MiscFeature)] <- "NA"
house_train$MiscFeature <- as.factor(house_train$MiscFeature)
```

Data imputation for test set

We fill up the missing information for the test data. Let's first examine what missing data we have. It is quite similar to imputing missing values we did in the training data set.

```
test_na_list <- colnames(house_test)[colSums(is.na(house_test)) > 0]
test_na_list
```

```
##  [1] "MSZoning"     "LotFrontage"  "Alley"        "Utilities"    "Exterior1st"
##  [6] "Exterior2nd"  "MasVnrType"   "MasVnrArea"   "BsmtQual"     "BsmtCond"
## [11] "BsmtExposure" "BsmtFinType1" "BsmtFinSF1"   "BsmtFinType2" "BsmtFinSF2"
## [16] "BsmtUnfSF"    "TotalBsmtSF"  "BsmtFullBath" "BsmtHalfBath" "KitchenQual"
## [21] "Functional"   "FireplaceQu"  "GarageType"   "GarageYrBlt"  "GarageFinish"
## [26] "GarageCars"   "GarageArea"   "GarageQual"   "GarageCond"   "PoolQC"
## [31] "Fence"        "MiscFeature"  "SaleType"
```

Fill up NA values for MSZoning as "None". As this literally means there is no information for zoning.

```
house_test$MSZoning <- as.character(house_test$MSZoning)
house_test$MSZoning[is.na(house_test$MSZoning)] <- "None"
house_test$MSZoning <- as.factor(house_test$MSZoning)
```

LotFrontage is defined as the linear feet of street connected to property. So values with NA are probably properties that are not connected to the street. We can impute the values of LotFrontage using a tree model. We use features such as LotArea and LotConfig for prediction, because these features appear to be correlated with each other according to a pairwise correlation graph of the features.

```
LotFrontage.tree <- rpart(LotFrontage~LotArea+BldgType+Neighborhood+LotConfig,data=house_test)

house_test[is.na(house_test$LotFrontage),]$LotFrontage <- predict(LotFrontage.tree,newdata=house_test[i
```

For Alley, NA literally means no alley access according to the data description. We thus fill up the NA values to the string value NA.

```r
house_test$Alley <- as.character(house_test$Alley)
house_test$Alley[is.na(house_test$Alley)] <- "Not Available"
house_test$Alley <- as.factor(house_test$Alley)
```

For Utilities, upon inspection there only seems to be 2 missing data, and the rest of the data are labeled as AllPub. It is quite difficult to infer the type of utility, so let's just use "Not Available"

```r
house_test$Utilities <- as.character(house_test$Utilities)
house_test$Utilities[is.na(house_test$Utilities)] <- "Not Available"
house_test$Utilities <- as.factor(house_test$Utilities)
```

For Fence, NA simply means no fence.

```r
house_test$Fence <- as.character(house_test$Fence)
house_test$Fence[is.na(house_test$Fence)] <- "No Fence"
house_test$Fence <- as.factor(house_test$Fence)
```

We perform similar imputation (compared to training set) to MasVnrType and MasVnrArea

```r
house_test$MasVnrType <- as.character(house_test$MasVnrType)
house_test$MasVnrType[is.na(house_test$MasVnrType)] <- "None"
house_test$MasVnrType <- as.factor(house_test$MasVnrType)

house_test$MasVnrArea[is.na(house_test$MasVnrArea)] <- 0
```

```r
house_test$BsmtQual <- as.character(house_test$BsmtQual)
house_test$BsmtQual[is.na(house_test$BsmtQual)] <- "NA"
house_test$BsmtQual <- as.factor(house_test$BsmtQual)

house_test$BsmtCond <- as.character(house_test$BsmtCond)
house_test$BsmtCond[is.na(house_test$BsmtCond)] <- "NA"
house_test$BsmtCond <- as.factor(house_test$BsmtCond)
```

```r
house_test$BsmtExposure <- as.character(house_test$BsmtExposure)
house_test$BsmtExposure[is.na(house_test$BsmtExposure)] <- "NA"
house_test$BsmtExposure <- as.factor(house_test$BsmtExposure)

house_test$BsmtFinType1 <- as.character(house_test$BsmtFinType1)
house_test$BsmtFinType1[is.na(house_test$BsmtFinType1)] <- "NA"
house_test$BsmtFinType1 <- as.factor(house_test$BsmtFinType1)

house_test$BsmtFinType2 <- as.character(house_test$BsmtFinType2)
house_test$BsmtFinType2[is.na(house_test$BsmtFinType2)] <- "NA"
house_test$BsmtFinType2 <- as.factor(house_test$BsmtFinType2)
```

```r
house_test$FireplaceQu <- as.character(house_test$FireplaceQu)
house_test$FireplaceQu[is.na(house_test$FireplaceQu)] <- "NA"
house_test$FireplaceQu <- as.factor(house_test$FireplaceQu)
```

```r
house_test$PoolQC <- as.character(house_test$PoolQC)
house_test$PoolQC[is.na(house_test$PoolQC)] <- "NA"
house_test$PoolQC <- as.factor(house_test$PoolQC)

house_test$MiscFeature <- as.character(house_test$MiscFeature)
house_test$MiscFeature[is.na(house_test$MiscFeature)] <- "NA"
```

```r
house_test$MiscFeature <- as.factor(house_test$MiscFeature)

house_test$GarageType <- as.character(house_test$GarageType)
house_test$GarageType[is.na(house_test$GarageType)] <- "NA"
house_test$GarageType <- as.factor(house_test$GarageType)

house_test$GarageCond <- as.character(house_test$GarageCond)
house_test$GarageCond[is.na(house_test$GarageCond)] <- "NA"
house_test$GarageCond <- as.factor(house_test$GarageCond)

house_test$GarageQual <- as.character(house_test$GarageQual)
house_test$GarageQual[is.na(house_test$GarageQual)] <- "NA"
house_test$GarageQual <- as.factor(house_test$GarageQual)

house_test$GarageFinish <- as.character(house_test$GarageFinish)
house_test$GarageFinish[is.na(house_test$GarageFinish)] <- "NA"
house_test$GarageFinish <- as.factor(house_test$GarageFinish)
```

Let's look at the number of missing data for Exterior1st and Exterior2nd.

```r
summary(house_test$Exterior1st)
```

```
## AsbShng AsphShn BrkComm BrkFace  CBlock CemntBd HdBoard MetalSd Plywood  Stucco
##      24       1       4      37       1      65     220     230     113      18
## VinylSd Wd Sdng WdShing    NA's
##     510     205      30       1
```

```r
summary(house_test$Exterior2nd)
```

```
## AsbShng AsphShn Brk Cmn BrkFace  CBlock CmentBd HdBoard ImStucc MetalSd Plywood
##      18       1      15      22       2      66     199       5     233     128
##   Stone  Stucco VinylSd Wd Sdng Wd Shng    NA's
##       1      21     510     194      43       1
```

There are only 1 NA's for each. After reading the data_description.txt, it appears that we can assign NAs to other.

```r
house_test$Exterior1st <- as.character(house_test$Exterior1st)
house_test$Exterior1st[is.na(house_test$Exterior1st)] <- "Other"
house_test$Exterior1st <- as.factor(house_test$Exterior1st)

house_test$Exterior2nd <- as.character(house_test$Exterior2nd)
house_test$Exterior2nd[is.na(house_test$Exterior2nd)] <- "Other"
house_test$Exterior2nd <- as.factor(house_test$Exterior2nd)
```

Let us assign missing values for BsmtFullBath and BsmtHalfBath to zero.

```r
house_test$BsmtFullBath[is.na(house_test$BsmtFullBath)] <- 0
house_test$BsmtHalfBath[is.na(house_test$BsmtHalfBath)] <- 0
```

Note that there are only 1 NA values for BsmtFinSF1,BSmtFinSF2,BsmtUnfSF and TotalBsmtSF. We assign these to both to zero.

```r
house_test$BsmtFinSF1[is.na(house_test$BsmtFinSF1)] <- 0
house_test$BsmtFinSF2[is.na(house_test$BsmtFinSF2)] <- 0
house_test$BsmtUnfSF[is.na(house_test$BsmtUnfSF)] <- 0
house_test$TotalBsmtSF[is.na(house_test$TotalBsmtSF)] <- 0
```

```
for (i in 1:nrow(house_test)){
  if (is.na(house_test[i,'GarageYrBlt'])){
    house_test[i,'GarageYrBlt'] <- house_test[i,'YearBuilt']
  }
}
```

Let us assign the missing values for KitchenQual,Functional and SaleType to the most frequent category.

```
summary(house_test$KitchenQual)
```

```
##   Ex   Fa   Gd   TA NA's
##  105   31  565  757    1
```

```
house_test$KitchenQual <- as.character(house_test$KitchenQual)
house_test$KitchenQual[is.na(house_test$KitchenQual)] <- "TA"
house_test$KitchenQual <- as.factor(house_test$KitchenQual)
```

```
summary(house_test$Functional)
```

```
## Maj1 Maj2 Min1 Min2  Mod  Sev  Typ NA's
##    5    4   34   36   20    1 1357    2
```

```
house_test$Functional <- as.character(house_test$Functional)
house_test$Functional[is.na(house_test$Functional)] <- "Typ"
house_test$Functional <- as.factor(house_test$Functional)
```

```
summary(house_test$SaleType)
```

```
##   COD   Con ConLD ConLI ConLw   CWD   New   Oth    WD  NA's
##    44     3    17     4     3     8   117     4  1258     1
```

```
house_test$SaleType <- as.character(house_test$SaleType)
house_test$SaleType[is.na(house_test$SaleType)] <- "WD"
house_test$SaleType <- as.factor(house_test$SaleType)
```

There are only one missing values for GarageCars and GarageArea, so let's just assign 0 to them.

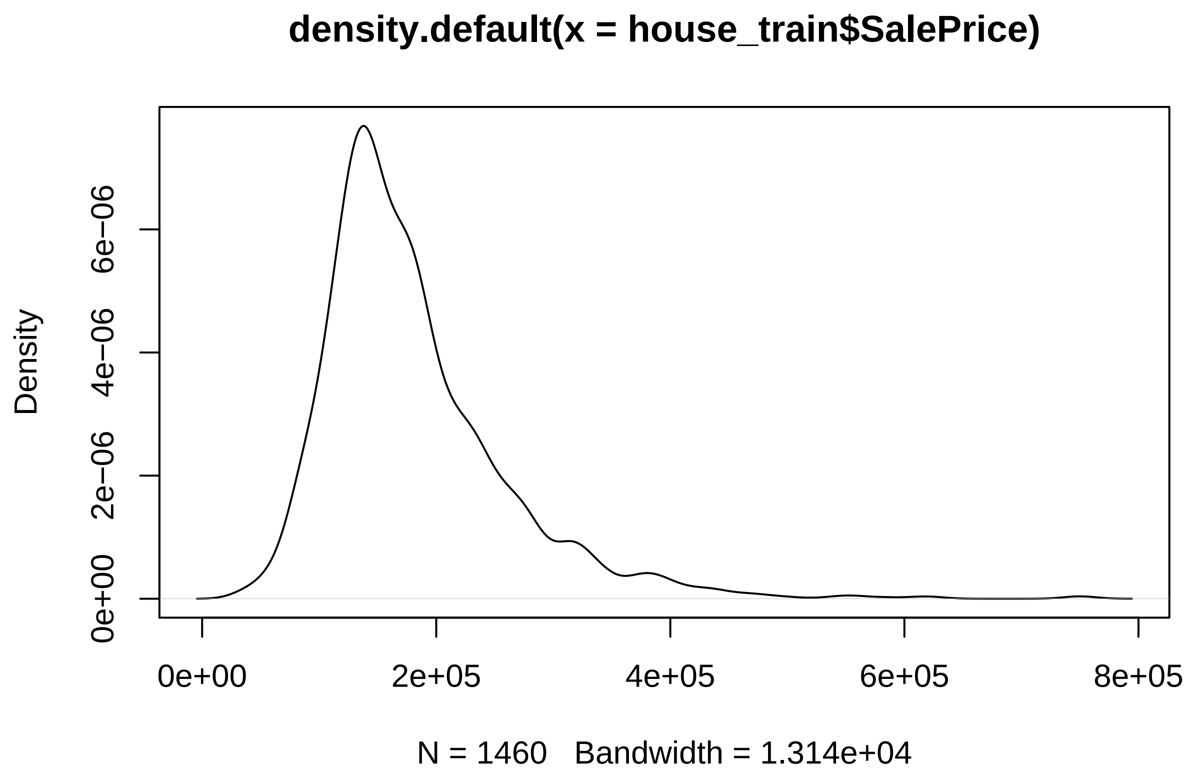
```
house_test$GarageCars[is.na(house_test$GarageCars)] <- 0
house_test$GarageArea[is.na(house_test$GarageArea)] <- 0
```

Now, we are officially done with filling in the missing data part. This process may be improved by using other prediction models to estimate missing data (such as what we did for LotArea), but what we have so far looks good for now.

4. Data transformation

Let us examine the distribution of SalePrice

```
plot(density(house_train$SalePrice))
```



```r
library(moments)
skewness(house_train$SalePrice)
```

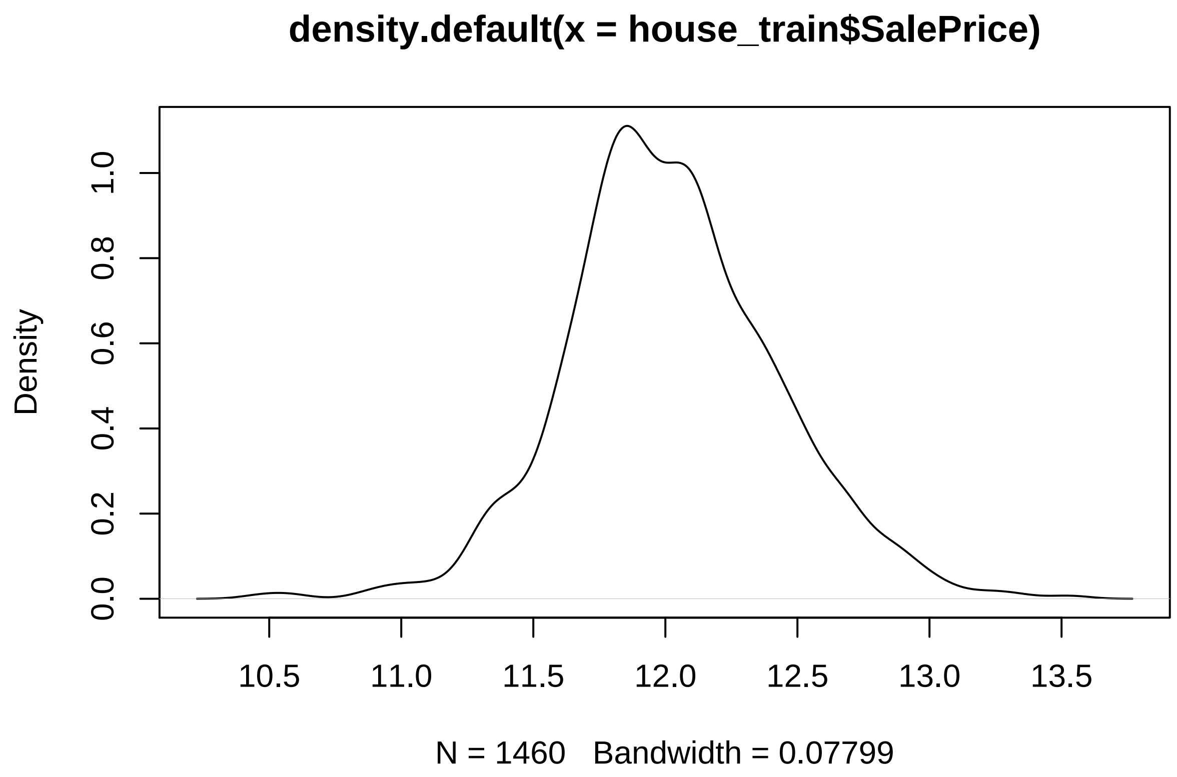
```
## [1] 1.880941
```

```r
kurtosis(house_train$SalePrice)
```

```
## [1] 9.509812
```

It appears that the data is quite skewed. We wish to normalize our target data. Let's try log transforming SalePrice.

```r
house_train$SalePrice <- log(house_train$SalePrice)
plot(density(house_train$SalePrice))
```



```
skewness(house_train$SalePrice)
```

```
## [1] 0.1212104
```

```
kurtosis(house_train$SalePrice)
```

```
## [1] 3.802656
```

The data appears to be normalized from the log transformation. Much better. We just need to exponentiate the final result (our prediction) before submission.

5. Feature Engineering

Combining features to create other features may actually improve the accuracy of our model. Let's just add a few features for now.

First, the age of the house, which can be calculated from the difference of the year built and year sold.

```
house_train$age <- as.integer(as.character(house_train$YrSold)) - as.integer(as.character(house_train$Ye

house_test$age <- as.integer(as.character(house_test$YrSold)) - as.integer(as.character(house_test$YearB
```

Second, the total square feet (area) of the house

```
house_train$TotalSF <- house_train$GrLivArea + house_train$TotalBsmtSF + house_train$X1stFlrSF + house_t

house_test$TotalSF <- house_test$GrLivArea + house_test$TotalBsmtSF + house_test$X1stFlrSF + house_test$
```

Third, the total square feet of the porch

```
house_train$TotalPorchSF <- house_train$OpenPorchSF + house_train$EnclosedPorch + house_train$X3SsnPorc

house_test$TotalPorchSF <-  house_test$OpenPorchSF + house_test$EnclosedPorch + house_test$X3SsnPorch +
```

We can also define a variable for overall Garage quality. We would need to conver the factor variables to numeric for this.

```
GarageQualNumeric <- c()
GarageQualChar <- as.character(house_train$GarageQual)
for (i in 1:length(GarageQualChar)){
  if (GarageQualChar[i] == "Ex"){
    GarageQualNumeric <- c(GarageQualNumeric,100)
  }else if (GarageQualChar[i] == "Gd"){
    GarageQualNumeric <- c(GarageQualNumeric,90)
  }else if (GarageQualChar[i] == "TA"){
    GarageQualNumeric <- c(GarageQualNumeric,80)
  }else if (GarageQualChar[i] == "Fa"){
    GarageQualNumeric <- c(GarageQualNumeric,70)
  }else if (GarageQualChar[i] == "Po"){
    GarageQualNumeric <- c(GarageQualNumeric,60)
  }else if (GarageQualChar[i] == "NA"){
    GarageQualNumeric <- c(GarageQualNumeric,50)
  }
}
GarageCondNumeric <- c()
GarageCondChar <- as.character(house_train$GarageCond)
for (i in 1:length(GarageCondChar)){
  if (GarageCondChar[i] == "Ex"){
    GarageCondNumeric <- c(GarageCondNumeric,100)
  }else if (GarageCondChar[i] == "Gd"){
    GarageCondNumeric <- c(GarageCondNumeric,90)
  }else if (GarageCondChar[i] == "TA"){
    GarageCondNumeric <- c(GarageCondNumeric,80)
  }else if (GarageCondChar[i] == "Fa"){
    GarageCondNumeric <- c(GarageCondNumeric,70)
  }else if (GarageCondChar[i] == "Po"){
    GarageCondNumeric <- c(GarageCondNumeric,60)
  }else if (GarageCondChar[i] == "NA"){
    GarageCondNumeric <- c(GarageCondNumeric,50)
  }
}
house_train$OverallGarageScore <- house_train$GarageCars * house_train$GarageArea * GarageQualNumeric *


GarageQualNumeric <- c()
GarageQualChar <- as.character(house_test$GarageQual)
for (i in 1:length(GarageQualChar)){
  if (GarageQualChar[i] == "Ex"){
    GarageQualNumeric <- c(GarageQualNumeric,100)
  }else if (GarageQualChar[i] == "Gd"){
    GarageQualNumeric <- c(GarageQualNumeric,90)
  }else if (GarageQualChar[i] == "TA"){
    GarageQualNumeric <- c(GarageQualNumeric,80)
  }else if (GarageQualChar[i] == "Fa"){
    GarageQualNumeric <- c(GarageQualNumeric,70)
  }else if (GarageQualChar[i] == "Po"){
    GarageQualNumeric <- c(GarageQualNumeric,60)
  }else if (GarageQualChar[i] == "NA"){
    GarageQualNumeric <- c(GarageQualNumeric,50)
  }
}
GarageCondNumeric <- c()
GarageCondChar <- as.character(house_test$GarageCond)
for (i in 1:length(GarageCondChar)){
  if (GarageCondChar[i] == "Ex"){
    GarageCondNumeric <- c(GarageCondNumeric,100)
  }else if (GarageCondChar[i] == "Gd"){
    GarageCondNumeric <- c(GarageCondNumeric,90)
  }else if (GarageCondChar[i] == "TA"){
    GarageCondNumeric <- c(GarageCondNumeric,80)
  }else if (GarageCondChar[i] == "Fa"){
    GarageCondNumeric <- c(GarageCondNumeric,70)
  }else if (GarageCondChar[i] == "Po"){
    GarageCondNumeric <- c(GarageCondNumeric,60)
  }else if (GarageCondChar[i] == "NA"){
    GarageCondNumeric <- c(GarageCondNumeric,50)
  }
}
house_test$OverallGarageScore <- house_test$GarageCars * house_test$GarageArea * GarageQualNumeric * Ga
```

We could create a weighted score for the total bath as well

```
house_train$BathScore <- house_train$BsmtFullBath + 0.5*house_train$BsmtHalfBath + house_train$FullBath

house_test$BathScore <- house_test$BsmtFullBath + 0.5*house_test$BsmtHalfBath + house_test$FullBath + 0
```

We can also add some variable transformation as a new feature. For instance, log of the general living area.

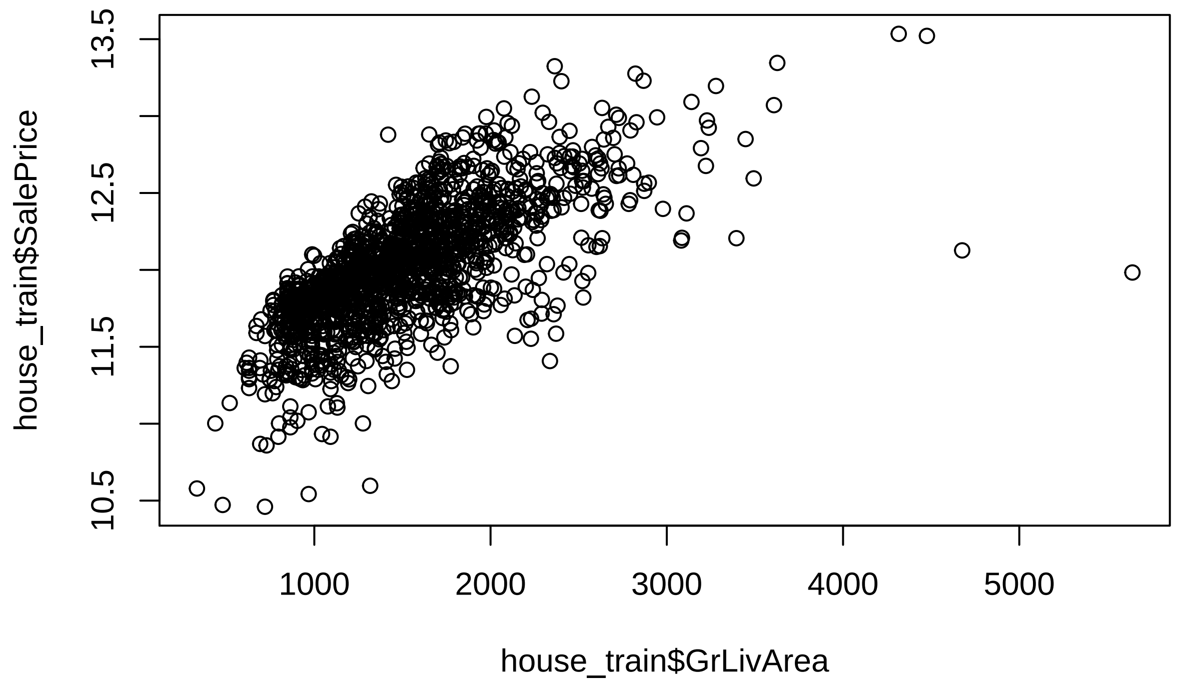
```
house_train$logGrLivArea <- log(house_train$GrLivArea)
house_test$logGrLivArea <- log(house_test$GrLivArea)
```

we can go on but let's stop here for now. Feature engineering requires creativity and one can come up with many interesting features by combining given/existing features. Sometimes I use a platform called h2o.ai (https://www.h2o.ai/) as running a training model on this platform generates new features that may help minimize RMSE or MSE.

6. Removing Outliers

One of the most correlated variables with SalePrice is the GrLivArea (they appear to have a strong linear relationship). Let's plot their relationship.

```
plot(house_train$GrLivArea,house_train$SalePrice)
```



```
cor(house_train$GrLivArea,house_train$SalePrice)
```

```
## [1] 0.7009267
```

It appears that we have a pretty high positive correlation between the two. The two points with GrLivArea > 4000 appear to be outliers. Let's get rid of them from the training set.

```
house_train[which(house_train$GrLivArea > 4000),c("GrLivArea","SalePrice")]
```

```
##      GrLivArea SalePrice
## 524       4676  12.12676
## 692       4316  13.53447
## 1183      4476  13.52114
## 1299      5642  11.98293
```

It must be the two rows with the highest and the next highest values for GrLivArea. They are rows 524 and 1299. Let's get rid of them.

```
house_train <- house_train[-c(524,1299)]
```

In this case, we decided to get rid of the outlier because the above two points appear to violate the highly positive correlation between SalePrice and GrLivArea.

7. Using GBM

After doing researches here and there, it appears that gradient boosting machines (GBM) is the way to go. From what I understand, random forest models build an ensemble of deep independent trees, whereas the GBM builds an ensemble of shallow and weak successive trees with each tree learning and improving on the previous, and when these trees are combined, they produce a powerful committee that outperforms many other models.

Basically, GBM relies on the intuition that the best possible next model (when combined with the previous model), minimizes the overall prediction error. We set the target outcomes for this next model in order to minimize the error and the target outcome for each case in the data depends on how much changing that case's prediction impacts the overall prediction error. For instance, if a small change in the prediction for a case causes a large decrease in error, then next target outcome of the case is a high value. Then predictions from the new model that are close to its target will reduce the error. More about GBM can be found here: https://cran.r-project.org/web/packages/gbm/gbm.pdf

8. Tuning the hyperparameters and cross validation

When using the GBM provided by R, there are a few parameters we have to decide. First is the number of trees. This literally means the total number of trees to fit. GBMs often require many trees, but too many trees may lead to overfitting so our goal is to find the optimal number of tress that minimize the loss function (in our case RMSE). Second is the depth of the tree. This means the number d of splits in each tree, which controls the complexity of the boosted ensemble. If $d = 1$, this means that the tree is a stump consisting of a single split. Usually we will have values of $d < 10$. Third is the learning rate (or shrinkage). This is a constant which controls how quickly the algorithm proceeds down the gradient descent. If the learning rate is too small, it may reduce chances of overfitting but may take a long time to reach optimal fit. Fourth is the option of subsampling (i.e. what percentage of the sample are we going to use), but let's not worry about this parameter in this analysis.

One way to test for the various parameter values is to use the caret package in R. We use 5-fold cross-validation to select the best values among the range of selected values for the hyperparameters. If the number of data points is very large for our training dataset (house_train), we could instead do a train-validation-test set split, but we only have 1460 data for our training set. Also, note that the range of values we selected for hyperparameter tuning were chosen at random (values that usually work well from past experience).

```r
library(caret)
library(parallel)
library(doParallel)
# for parallel computing
cluster <- makeCluster(detectCores()-1)
registerDoParallel(cluster)

gbmGrid <- expand.grid(interaction.depth = c(3,4,6,10),
                       n.trees = c(100,500,1000),
                       shrinkage = c(0.1,0.01),
                       n.minobsinnode = 10)

# for 5-fold cross-validation
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,
                           repeats = 1)

gbmFit <- train(SalePrice~.,data = house_train,
                method = "gbm",
                trControl = fitControl,
                verbose = FALSE,
                tuneGrid = gbmGrid)

gbmFit
```

9. Submitting prediction results from the GBM model

It took a very very long time (many hours) for the Caret hyperparameter optimization to run, but it seems that among the values we considered, n.trees = 1000, interaction.depth = 4 and shrinkage = 0.1 seems to produce the least RMSE from 5-fold cross-validation. Let's use these values for the hyperparameters for tuning our GBM model.

```
set.seed(1)
boost.model <- gbm(SalePrice~.,data=house_train,distribution="gaussian",n.trees=1000,interaction.depth=4
predicted_prices <- exp(predict(boost.model,newdata=house_test,n.trees=100))
head(predicted_prices) # printing the first few values
submission <- data_frame('Id' = IdList, 'SalePrice' = predicted_prices)
write_csv(submission,'submission.csv')
```

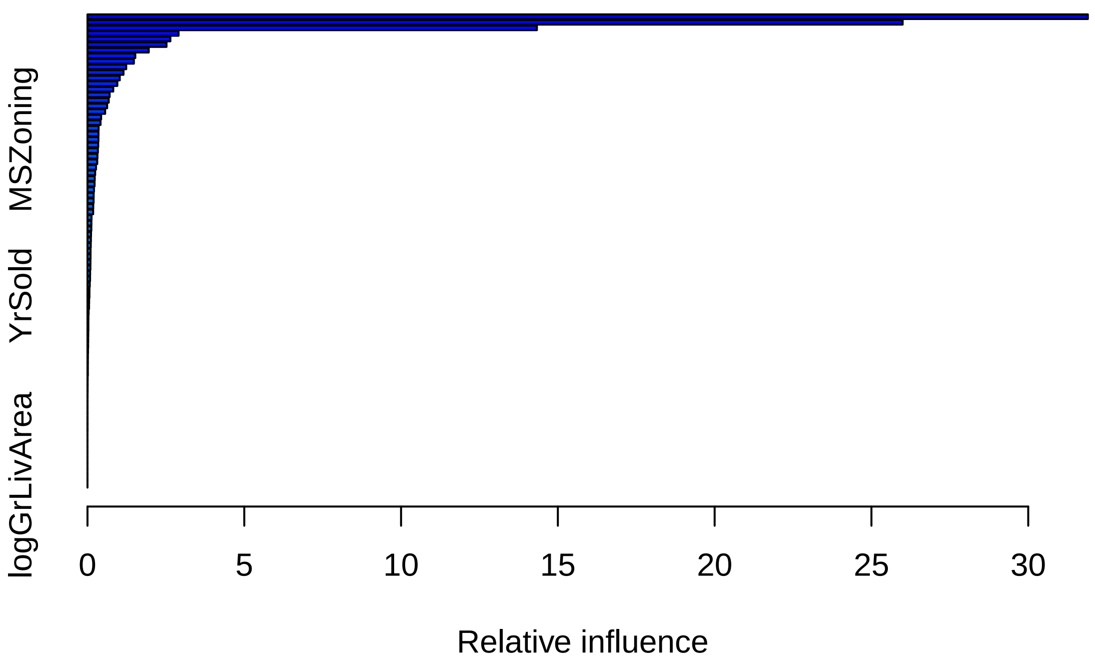
Upon submitting the result file submission.csv, I get a public leaderboard score of 0.13269. This is better than the result obtained from linear regression which was around 0.16. Note that we only tried a small subset of possible values for the hyperparameters in our caret train function. Why don't we try fitting a GBM model with larger number of trees (that was untested on caret) and see how it performs on the public leaderboard?

```
set.seed(1)
boost.model <- gbm(SalePrice~.,data=house_train,distribution="gaussian",n.trees=10000,interaction.depth=
predicted_prices <- exp(predict(boost.model,newdata=house_test,n.trees=10000))
head(predicted_prices) # printing the first few values
```

```
## [1] 118241.2 159895.1 180415.1 187808.2 193852.1 175250.1
```

```
submission <- data_frame('Id' = IdList, 'SalePrice' = predicted_prices)
write_csv(submission,'submission_gbm_large.csv')
```

```
summary(boost.model)
```



```
##                                     var      rel.inf
## TotalSF                         TotalSF 3.190135e+01
## OverallQual                 OverallQual 2.599556e+01
## Neighborhood               Neighborhood 1.433188e+01
## BathScore                     BathScore 2.904954e+00
## OverallGarageScore OverallGarageScore 2.643739e+00
## KitchenQual                 KitchenQual 2.520291e+00
## OverallCond                 OverallCond 1.953962e+00
## CentralAir                   CentralAir 1.524829e+00
## ExterQual                     ExterQual 1.479198e+00
## BsmtFinSF1                   BsmtFinSF1 1.234927e+00
## YearRemodAdd               YearRemodAdd 1.148614e+00
## X1stFlrSF                     X1stFlrSF 1.029248e+00
## LotArea                         LotArea 9.515975e-01
## GarageFinish               GarageFinish 8.216160e-01
## GarageType                   GarageType 7.069139e-01
## SaleCondition             SaleCondition 6.780230e-01
## FireplaceQu                 FireplaceQu 6.268852e-01
## GrLivArea                     GrLivArea 5.635533e-01
## TotalBsmtSF                 TotalBsmtSF 4.356798e-01
## BsmtQual                       BsmtQual 4.163487e-01
## age                                 age 3.545052e-01
## GarageCond                   GarageCond 3.506119e-01
## MSZoning                       MSZoning 3.492189e-01
## GarageCars                   GarageCars 3.422894e-01
## BsmtFinType1               BsmtFinType1 3.317679e-01
## Functional                    Functional 3.185893e-01
## Exterior1st                  Exterior1st 3.109883e-01
## BsmtExposure                BsmtExposure 2.711663e-01
## OpenPorchSF                  OpenPorchSF 2.449393e-01
## GarageYrBlt                  GarageYrBlt 2.313923e-01
## TotalPorchSF                TotalPorchSF 2.261025e-01
## Condition1                    Condition1 2.064657e-01
## Exterior2nd                  Exterior2nd 1.969863e-01
## YearBuilt                      YearBuilt 1.952451e-01
## GarageQual                    GarageQual 1.840635e-01
## X2ndFlrSF                      X2ndFlrSF 1.808400e-01
## GarageArea                    GarageArea 1.287766e-01
## Fireplaces                    Fireplaces 1.253176e-01
## BsmtUnfSF                      BsmtUnfSF 1.228686e-01
## WoodDeckSF                    WoodDeckSF 1.134217e-01
## ExterCond                      ExterCond 1.088267e-01
## ScreenPorch                  ScreenPorch 1.050061e-01
## HeatingQC                      HeatingQC 9.766255e-02
## KitchenAbvGr                KitchenAbvGr 9.667885e-02
## SaleType                        SaleType 9.459711e-02
## LotFrontage                  LotFrontage 9.285020e-02
## RoofMatl                        RoofMatl 8.381883e-02
## PavedDrive                    PavedDrive 8.227817e-02
## BsmtCond                        BsmtCond 7.314285e-02
## FullBath                        FullBath 6.288260e-02
## YrSold                            YrSold 6.190483e-02
## LandContour                  LandContour 5.285616e-02
## LotConfig                      LotConfig 5.043052e-02
## TotRmsAbvGrd                TotRmsAbvGrd 3.537849e-02
## MoSold                            MoSold 2.977254e-02
## BldgType                        BldgType 2.815373e-02
## MSSubClass                    MSSubClass 2.797538e-02
## MasVnrArea                    MasVnrArea 2.468783e-02
## EnclosedPorch              EnclosedPorch 2.309203e-02
## Fence                              Fence 2.121818e-02
## Foundation                    Foundation 1.666080e-02
## RoofStyle                      RoofStyle 1.326262e-02
## BedroomAbvGr                BedroomAbvGr 1.275603e-02
## LotShape                        LotShape 1.217478e-02
## HouseStyle                    HouseStyle 1.215338e-02
## BsmtFinType2                BsmtFinType2 6.259876e-03
## Electrical                    Electrical 5.272224e-03
## BsmtFullBath                BsmtFullBath 4.146874e-03
## Heating                          Heating 2.905734e-03
## Alley                              Alley 1.650243e-03
## Condition2                    Condition2 1.631218e-03
## MasVnrType                    MasVnrType 1.135248e-03
## BsmtFinSF2                    BsmtFinSF2 6.827777e-04
## LandSlope                      LandSlope 5.287907e-04
## MiscFeature                  MiscFeature 5.163220e-04
## HalfBath                        HalfBath 1.714050e-04
## LowQualFinSF                LowQualFinSF 1.501217e-04
## Street                            Street 0.000000e+00
## Utilities                      Utilities 0.000000e+00
## BsmtHalfBath              BsmtHalfBath 0.000000e+00
## X3SsnPorch                  X3SsnPorch 0.000000e+00
## PoolArea                      PoolArea 0.000000e+00
## PoolQC                          PoolQC 0.000000e+00
## MiscVal                        MiscVal 0.000000e+00
## logGrLivArea              logGrLivArea 0.000000e+00
```

From the summary, we see that the features TotalSF, OverallQual and Neighborhood are the 3 most influential features.

Upon submitting the result to Kaggle, I saw that the RMSE was reduced down to 0.12752. How about we go one step further and increase the number of trees to 50000?

```r
set.seed(1)
boost.model <- gbm(SalePrice~.,data=house_train,distribution="gaussian",n.trees=50000,interaction.depth
predicted_prices <- exp(predict(boost.model,newdata=house_test,n.trees=50000))
head(predicted_prices) # printing the first few values
submission <- data_frame('Id' = IdList, 'SalePrice' = predicted_prices)
write_csv(submission,'submission_gbm_larger.csv')
```

With many trees and a low learning rate, the computation took many minutes to run. Upon submitting the result on Kaggle, I ended up with an RMSE of 0.12793 (a slightly lower RMSE than before). This is probably due to overfitting with a very large number of trees. For our GBM model, let's just stick to the hyperparameters of 10000 trees, depth 4 and shrinkage 0.001 for now.

10. Lasso Regression and the Ensemble

Note that above, we have been using GBM in order to predict SalePrices. Now let's take a different turn and use a different model for prediction and see how it compares to GBM. We will be using the Lasso.The Lasso is defined as follows: $L + \alpha \sum_{i=1}^{n} |w_i|$, where α is the penalty. The greater the penalty, it forces the weights w to be smaller and vice versa. Some of the weights may be reduced down to zero. This is actually called L1 regularization.

To implement the lasso, I have referred to the code in this github repo: https://github.com/praneethkvs/house-prices

What is interesting to note is that unlike me who went to impute data per feature, the owner of the repo used the missForest package to predict the missing NA values using the tree model (I used this method too - using a tree for prediction for only a small number of features though, not all of them).

The owner of this repo also provides the csv file for the results of the lasso. Let's download it and use the results. According to the owner of the github repo, this Lasso output yielded a score of 0.125, so slightly better than my GBM.

```r
library(tidyr)
lasso_pred <- read.csv('./lasso_preds.csv')
head(lasso_pred)
```

```
##     Id SalePrice
## 1 1461  122949.0
## 2 1462  151048.8
## 3 1463  181028.6
## 4 1464  191707.2
## 5 1465  203731.7
## 6 1466  171616.2
```

We try to average the results from my best GBM and the Lasso. This is called an ensemble, and it may reduce the variance or the bias of the output and hence increase prediction accuracy.

```r
ensemble <- (predicted_prices + lasso_pred$SalePrice)/2
head(ensemble)
```

```
## [1] 120595.1 155472.0 180721.9 189757.7 198791.9 173433.1
```

```r
submission <- data_frame('Id' = IdList, 'SalePrice' = ensemble)
write_csv(submission,'submission_ensemble.csv')
```

Upon submitting on Kaggle, I got a result of 0.12196, which is the best score obtained so far! This currently puts me in the top 28%.

11. Conclusion and possible next steps

The best achieved score so far is the top 28% score from combining the results of GBM and Lasso.

We could improve our RMSE results as follows

- Try a greater range of values for caret for hyperparameter optimization. If caret takes to long for computation, we can use other packages such as h2o.
- Consider trying other tree-based models such as random forest and cubist. Furthermore, try to ensemble these results with the previously achieved results (i.e. stacking).
- Try to do more feature engineering to add more features and figure out some features that may be important in predicting SalePrices.
- Try to run PCA or PCA regression for dimensionality reduction. Try to reduce the number of features for our GBM model and see how it performs.
- Try to run various models after normalizing data (to make sure they are in the same scale if appropriate).
- Try to obtain more datapoints and perhaps attempt neural nets?
- Try LightGBM and XGBoost and again combine results.