ShengYa Mei GB 656 Final Project 12/19/2022

Disclosure: This is a team submission by ShengYa Mei and Binhao Chen

Problem Statement

The Wisconsin housing market has been unsettling in the year 2022. The median home price sold in Wisconsin had an increase of 9.8% compared to last year and the number of homes sold was down 32.3% year over year (redfin.com). As a result, **Zillow's** real estate market in the Wisconsin region suffered from the impact and experienced a plunge in houses sold.

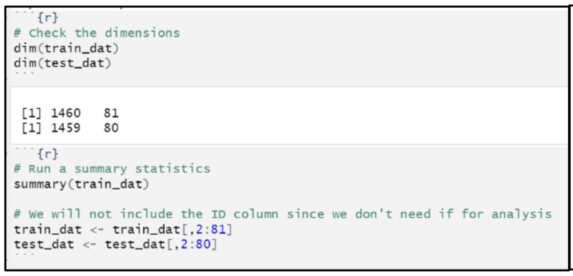
Business Application

The machine learning model constructed in this project aims to provide an accurate prediction of housing prices to be used by **Zillow Real Estate** in optimizing their real-estate marketplace. This model will benefit **Zillow** directly in their house pricing decisions as well as customers of **Zillow** in offering them a price that is fair and based. **Zillow** seeks to improve their housing sales in the upcoming year by setting prices that can accurately reflect the predicted housing market. To do this, **Zillow** has gathered house sales data in the year 2022 with specific details (features) on the houses sold and the sale price for each of the house sold. This data can be found in Excel file '**train.csv**'. **Zillow** has also collected information from the houses they will be putting on their marketplace in the year 2023 without sale prices set. This data can be found in Excel file '**test.csv**'. The goal for **Zillow** is to build a machine learning model based on the complete data with house sale prices in '**train.csv**', then, use this model to predict the price for houses in found in '**test.csv**'.

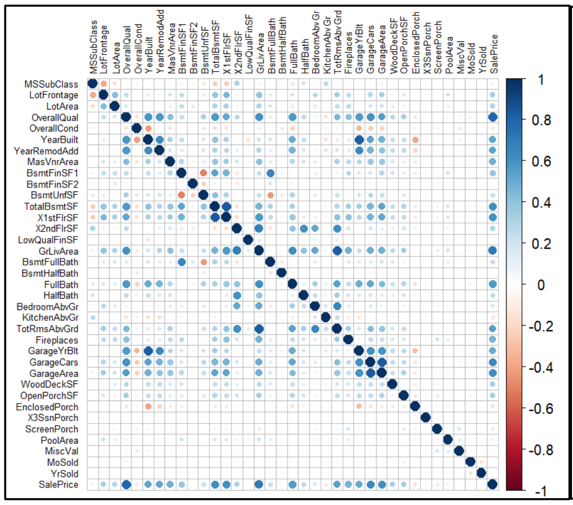
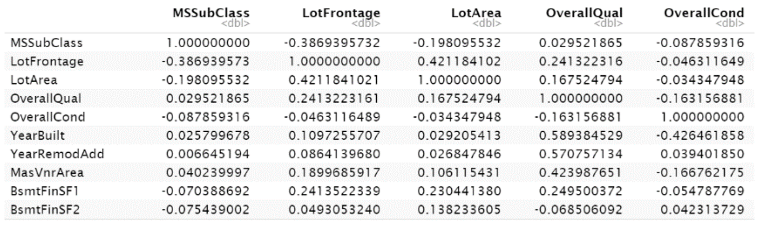
Project Approach

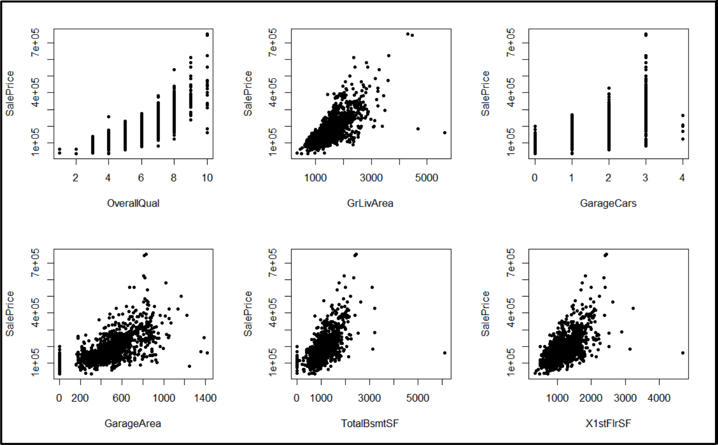
1. **Import required data sets, packages, and libraries**
   We will read in train and test data sets into RStudio as well as installing and calling packages and libraries that are needed for this model construction.

2. **Exploratory data analysis (EDA)**
   We begin by checking the dimensions of our data sets and running a simple summary statistic to view the min, max and mean of each feature.

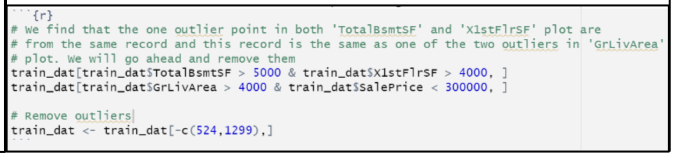


~~~
```{r}
# Check the dimensions
dim(train_dat)
dim(test_dat)
~~~

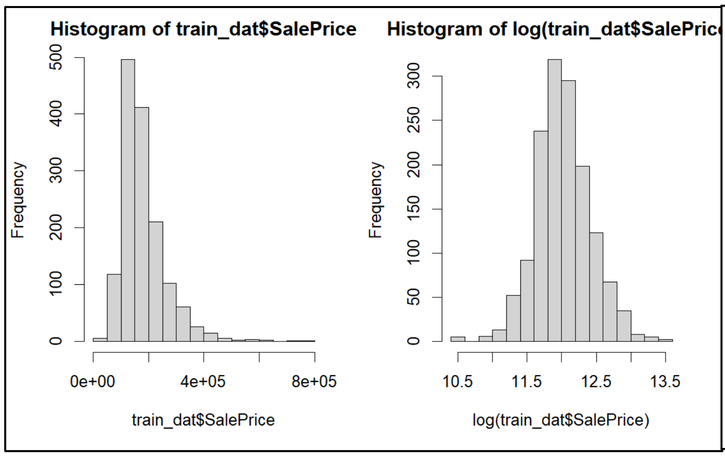
 [1] 1460   81
 [1] 1459   80
```{r}
# Run a summary statistics
summary(train_dat)

# We will not include the ID column since we don't need if for analysis
train_dat <- train_dat[,2:81]
test_dat <- test_dat[,2:80]
```
```

Our train data holds 1460 records for 81 different features whereas test data holds 1459 records for 80 different features. The Id column isn't necessary in data analysis so we will remove it from both data sets.

A correlation plot + matrix help us to understand the relationship between each of the features. We are particularly interested in understanding how each feature relates to 'SalePrice' of a house since that is the feature we want to predict.



We see that features 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea', 'TotalBsmtSF' and 'X1stFlrSF' have a good positive correlation of over 0.6 with 'SalePrice'. None of the features seems to have a strong negative correlation with 'SalePrice'. We will follow this up with scatterplot to visual their relationships. Note: Please refer to Rmarkdown file for complete correlation matrix



	MSSubClass <dbl>	LotFrontage <dbl>	LotArea <dbl>	OverallQual <dbl>	OverallCond <dbl>
MSSubClass	1.000000000	-0.3869395732	-0.198095532	0.029521865	-0.087859316
LotFrontage	-0.386939573	1.0000000000	0.421184102	0.241322316	-0.046311649
LotArea	-0.198095532	0.4211841021	1.000000000	0.167524794	-0.034347948
OverallQual	0.029521865	0.2413223161	0.167524794	1.000000000	-0.163156881
OverallCond	-0.087859316	-0.0463116489	-0.034347948	-0.163156881	1.000000000
YearBuilt	0.025799678	0.1097255707	0.029205413	0.589384529	-0.426461858
YearRemodAdd	0.006645194	0.0864139680	0.026847846	0.570757134	0.039401850
MasVnrArea	0.040239997	0.1899685917	0.106115431	0.423987651	-0.166762175
BsmtFinSF1	-0.070388692	0.2413522339	0.230441380	0.249500372	-0.054787769
BsmtFinSF2	-0.075439002	0.0493053240	0.138233605	-0.068506092	0.042313729
```



We see some prominent outliers in 'GrLivArea', 'TotalBsmtSF' and 'X1stFlrSF'. We will go ahead and remove them from our train data to avoid skewed results and under-performing models.

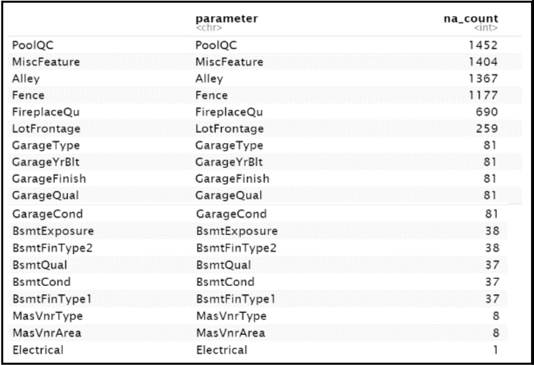


~~~
```{r}
# We find that the one outlier point in both 'TotalBsmtSF' and 'X1stFlrSF' plot are
# from the same record and this record is the same as one of the two outliers in 'GrLivArea'
# plot. We will go ahead and remove them
train_dat[train_dat$TotalBsmtSF > 5000 & train_dat$X1stFlrSF > 4000, ]
train_dat[train_dat$GrLivArea > 4000 & train_dat$SalePrice < 300000, ]

# Remove outliers
train_dat <- train_dat[-c(524,1299),]
```
~~~



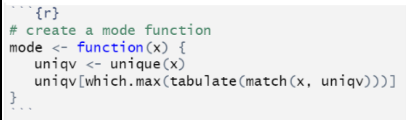
Histogram on sale price shows a right-skewed distribution. We see that most sale prices are clustered around 150K with some exception that are around 800K. We can perform a log transformation to visualize a normal distribution of sale price.

3. Data cleaning and transformation

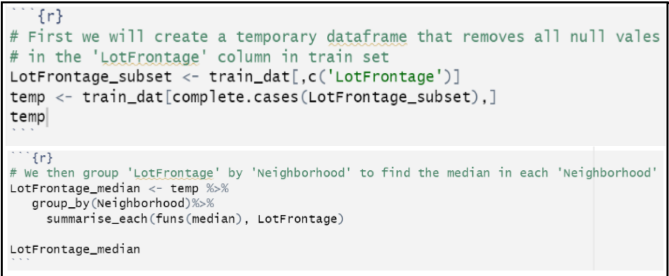


	parameter <chr>	na_count <int>
PoolQC	PoolQC	1452
MiscFeature	MiscFeature	1404
Alley	Alley	1367
Fence	Fence	1177
FireplaceQu	FireplaceQu	690
LotFrontage	LotFrontage	259
GarageType	GarageType	81
GarageYrBlt	GarageYrBlt	81
GarageFinish	GarageFinish	81
GarageQual	GarageQual	81
GarageCond	GarageCond	81
BsmtExposure	BsmtExposure	38
BsmtFinType2	BsmtFinType2	38
BsmtQual	BsmtQual	37
BsmtCond	BsmtCond	37
BsmtFinType1	BsmtFinType1	37
MasVnrType	MasVnrType	8
MasVnrArea	MasVnrArea	8
Electrical	Electrical	1

We checked for null value counts for each feature and found a total missing value of 6958 in our train data set. Most of the NA values are explained in the data set description where NA means there are none of the feature present at that house. For these NA values, we will replace them with 'none'. For missing categorical features, we will replace the NA value with the mode. Created mode function to calculate the mode.



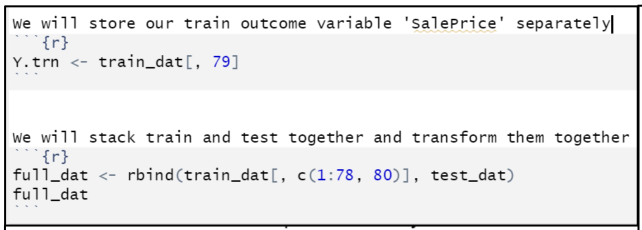
~~~
```{r}
# create a mode function
mode <- function(x) {
  uniqv <- unique(x)
  uniqv[which.max(tabulate(match(x, uniqv)))]
}
```
~~~



~~~
```{r}
# First we will create a temporary dataframe that removes all null vales
# in the 'LotFrontage' column in train set
LotFrontage_subset <- train_dat[,c('LotFrontage')]
temp <- train_dat[complete.cases(LotFrontage_subset),]
temp
~~~

```{r}
# We then group 'LotFrontage' by 'Neighborhood' to find the median in each 'Neighborhood'
LotFrontage_median <- temp %>%
   group_by(Neighborhood)%>%
     summarise_each(funs(median), LotFrontage)

LotFrontage_median
```
```

For feature 'LotFrontage', we see that it takes numerical values and that there is no specified values for NA. In this case, we will find the median of 'LotFrontage' after grouping by feature 'Neighborhood' then we will use this median to fill out missing 'LotFrontage' based on the 'Neighborhood' they are in. This can be accomplished through a left join on Neighborhood. We will perform the same cleaning process for our test data
```

Some of the numerical variables needs to be in categorical type otherwise our model won't be able to interpret it correctly.

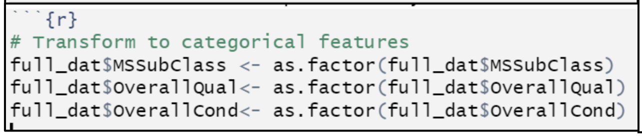


We will store our train outcome variable 'SalePrice' separately

~~~
```{r}
Y.trn <- train_dat[, 79]
```
~~~

We will stack train and test together and transform them together

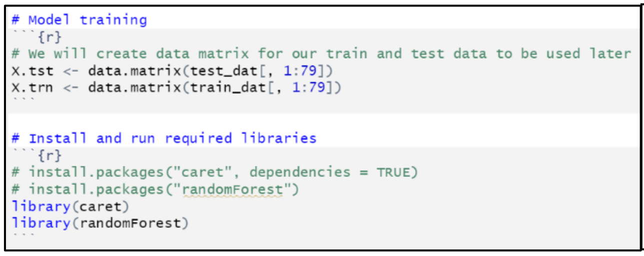
~~~
```{r}
full_dat <- rbind(train_dat[, c(1:78, 80)], test_dat)
full_dat
```
~~~



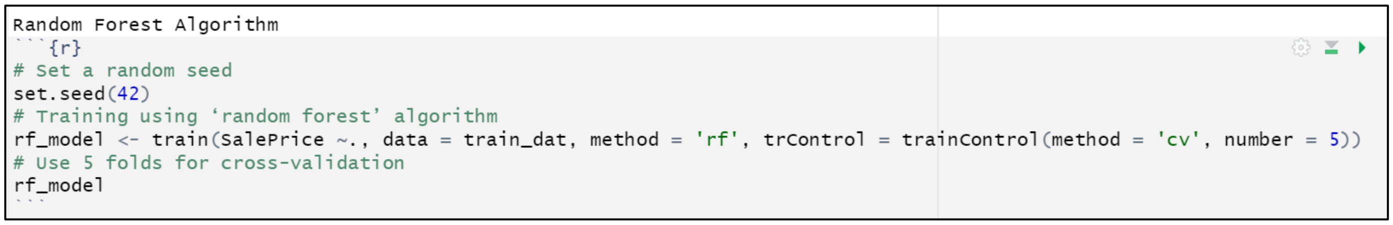
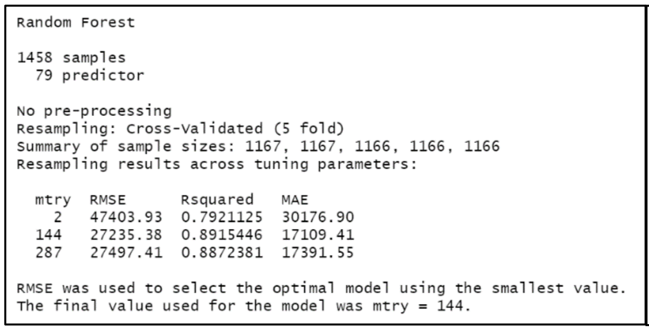
~~~
```{r}
# Transform to categorical features
full_dat$MSSubClass <- as.factor(full_dat$MSSubClass)
full_dat$OverallQual<- as.factor(full_dat$OverallQual)
full_dat$OverallCond<- as.factor(full_dat$OverallCond)
~~~

Before we feed in our train data into ML models, we first need to transform our categorical variables into numerical attributes which can be processed by the models. We perform label encoding using as.factor() for both train and test data set together. Then, we split them back to their train and test data sets.

4. Model training



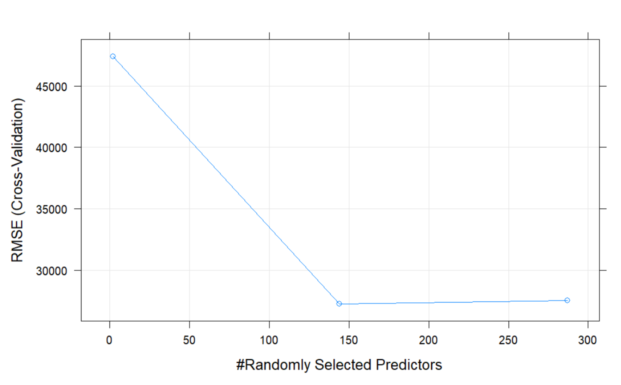
~~~
# Model training
```{r}
# We will create data matrix for our train and test data to be used later
X.tst <- data.matrix(test_dat[, 1:79])
X.trn <- data.matrix(train_dat[, 1:79])
~~~

Install and run required libraries
```{r}
# install.packages("caret", dependencies = TRUE)
# install.packages("randomForest")
library(caret)
library(randomForest)
```
```

We will store features in train and test data sets into variables to be used later in model building. We will import required libraries to run a Random Forest algorithm first.



Random Forest Algorithm

```
```{r}
# Set a random seed
set.seed(42)
# Training using 'random forest' algorithm
rf_model <- train(SalePrice ~., data = train_dat, method = 'rf', trControl = trainControl(method = 'cv', number = 5))
# Use 5 folds for cross-validation
rf_model
```
```



```
Random Forest

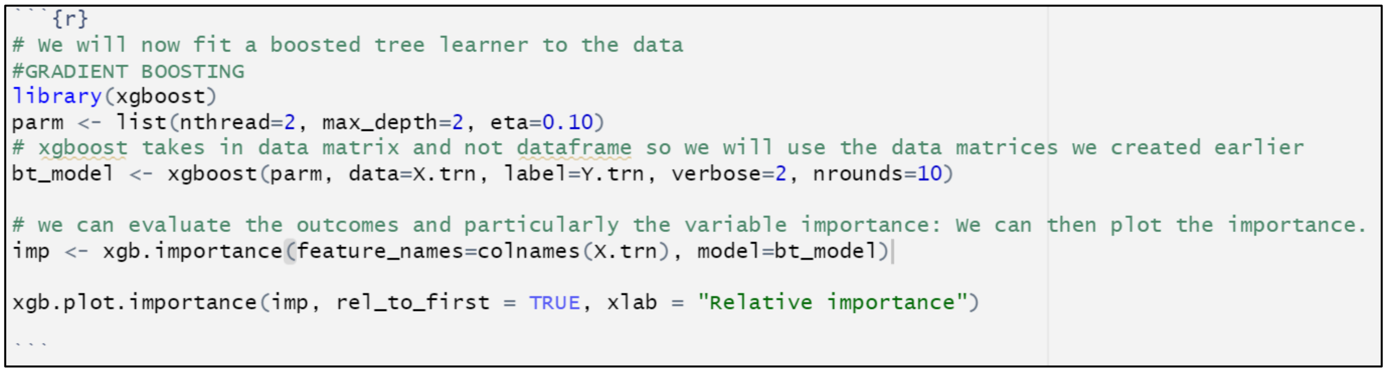
1458 samples
  79 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 1167, 1167, 1166, 1166, 1166
Resampling results across tuning parameters:

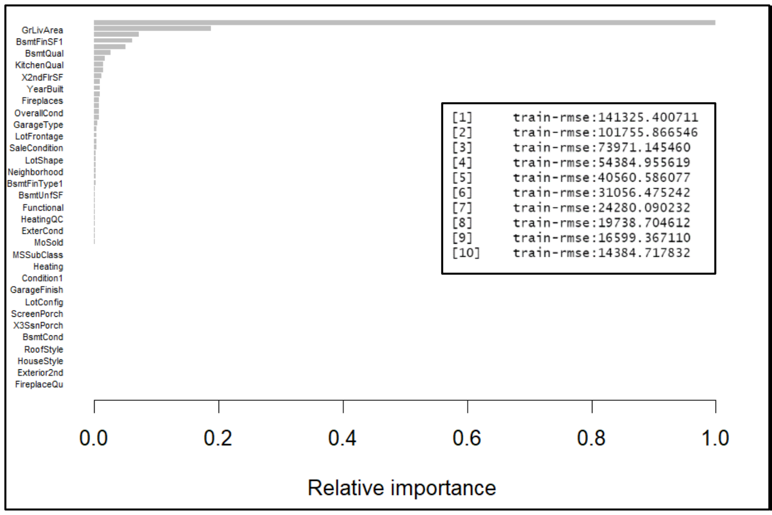
  mtry  RMSE      Rsquared   MAE
    2   47403.93  0.7921125  30176.90
  144   27235.38  0.8915446  17109.41
  287   27497.41  0.8872381  17391.55

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 144.
```

The best model generated has a r^2 value of 0.8915 and RMSE of 27235.38 and MAE of 17109.41. An r^2 value of 0.8915 is not bad but we will see if other algorithm approach can yield a lower RMSE. First, let us plot our rf_model.



We can see the point where the machine chose to be the best selection of predictors with the least RMSE (mtry = 144). This shows the bias and variance trade-off. We want a model that is as simple as possible and as complex as necessary.
```

Now, we will see if we can improve our model further with lower RMSE by running a Gradient Boost (Boosted Tree)



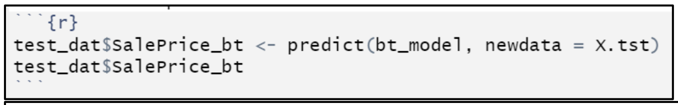
~~~
```{r}
# We will now fit a boosted tree learner to the data
#GRADIENT BOOSTING
library(xgboost)
parm <- list(nthread=2, max_depth=2, eta=0.10)
# xgboost takes in data matrix and not dataframe so we will use the data matrices we created earlier
bt_model <- xgboost(parm, data=X.trn, label=Y.trn, verbose=2, nrounds=10)

# we can evaluate the outcomes and particularly the variable importance: We can then plot the importance.
imp <- xgb.importance(feature_names=colnames(X.trn), model=bt_model)

xgb.plot.importance(imp, rel_to_first = TRUE, xlab = "Relative importance")

```
~~~



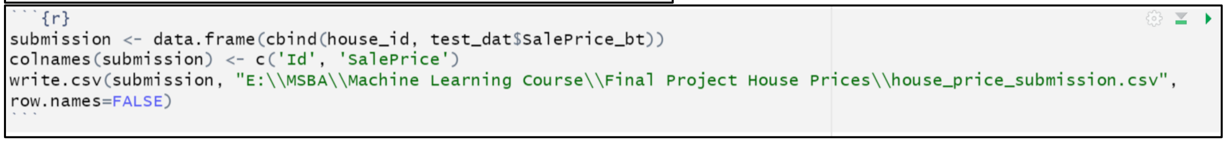
```
[1]	train-rmse:141325.400711
[2]	train-rmse:101755.866546
[3]	train-rmse:73971.145460
[4]	train-rmse:54384.955619
[5]	train-rmse:40560.586077
[6]	train-rmse:31056.475242
[7]	train-rmse:24280.090232
[8]	train-rmse:19738.704612
[9]	train-rmse:16599.367110
[10]	train-rmse:14384.717832
```

We did get a significantly lower train-RMSE of 14384.717832 after running through 10 iterations. By plot feature important from our gradient boost model, we can clearly see which features are the strongest determinant for 'SalePrice'.

We will now use this boosted tree model to predict our test data and create a submission file that lists out all house Ids in the test data set and each of their predicted sale price.



~~~
```{r}
test_dat$SalePrice_bt <- predict(bt_model, newdata = X.tst)
test_dat$SalePrice_bt
```
~~~



~~~
```{r}
submission <- data.frame(cbind(house_id, test_dat$SalePrice_bt))
colnames(submission) <- c('Id', 'SalePrice')
write.csv(submission, "E:\\MSBA\\Machine Learning Course\\Final Project House Prices\\house_price_submission.csv",
row.names=FALSE)
```
~~~

Project Summary

Machine learning techniques used in this project follows the recommended approach stated in the **House Prices – Advanced Regression Techniques** Kaggle competition. Random forest and gradient boost techniques were used to predict house prices. Results from gradient boost model was used in final Kaggle submission.