**SENG 474**

**Project report**

**House price prediction**

**Abstract**

**Our project intends to predict the house price by using a data set describing the sale of individual residential property in Ames, Iowa from 2006 to 2010. We will build one small data set which will pick from original data set by our understanding of attributes. Then, we will do visualization analysis and feature engineering to both small data set and original one. Eventually, we will put them into different models and to compare the difference which will give us better understanding of each model we choose. Additionally, let us to have some experience with tuning model’s parameter.**

***Keywords-----house; price prediction data mining; feature engineering; programming; python; sklean***

Xianhui Chen (V00803475) and Shiyi Huo (V00832071)

Department of Computer Science, University of Victoria

Table of Contents

[1 Introduction 2](#_Toc531285605)

[2 Related Work 2](#_Toc531285606)

[3 Data Preprocessing 2](#_Toc531285607)

[3.1 Data Exploration 2](#_Toc531285608)

[3.1.1 Overview 2](#_Toc531285609)

[3.1.2 Heatmap 3](#_Toc531285610)

[3.1.3 Scatterplot Matrix and Scatter Plot 4](#_Toc531285611)

[3.1.4 Wrap Up 5](#_Toc531285612)

[3.2 Dependent Variable Analysis (SalePrice) 5](#_Toc531285613)

[3.3 Independent Variable Analysis 7](#_Toc531285614)

[3.3.1 Missing Data 7](#_Toc531285615)

[3.3.2 Transform “Numerical” Variables into Categorical [4] 8](#_Toc531285616)

[3.3.3 Label Encoding Categorical Text Attributes [6] 8](#_Toc531285617)

[3.3.4 Transform Skewed Attributes 8](#_Toc531285618)

[3.3.5 Convert Categorical Data into One-hot Encoding [6] 9](#_Toc531285619)

[3.3.6 Data Cleaning for Small Dataset 9](#_Toc531285620)

[4 Data Mining 9](#_Toc531285621)

[4.1 10-fold Cross Validation 9](#_Toc531285622)

[4.2 Algorithms 9](#_Toc531285623)

[4.3 Metrics 10](#_Toc531285624)

[5 Observations 10](#_Toc531285625)

[5.1 Basic Models 10](#_Toc531285626)

[5.1.1 Ordinary Least Square Regression 10](#_Toc531285627)

[5.1.2 Ridge Regression 11](#_Toc531285628)

[5.1.3 Lasso Regression 11](#_Toc531285629)

[5.1.4 Elastic Net 12](#_Toc531285630)

[5.1.5 SVR (Support Vector Regression) 12](#_Toc531285631)

[5.2 Ensemble Models 13](#_Toc531285632)

[5.2.1 Gradient Boosting Regression and XGBoost 13](#_Toc531285633)

[5.2.2 Random Forest 13](#_Toc531285634)

[6 Conclusion 14](#_Toc531285635)

[References 14](#_Toc531285636)

# 1 Introduction

This project is adapted from “House Prices: Advanced Regression Techniques” on Kaggle [1]. The training dataset is of dimension 1461 x 81, which contains 1461 instances. With one dependent variable (sale price of a house) and 80 independent variables, we try to build models to predict sale price. The challenge is to deal with enormous amount of independent variables and select proper ones to build models. We will analyze dependent and independent variables, explore more about the relationship between them using visualization, apply data mining algorithms on preprocessed data, and evaluate performance of different models. Our goal is to explore which model has better performance.

# 2 Related Work

The data preprocessing phase is built on kernels posted for this regression problem [2].

Dependent variable analysis and data exploration are adapted from “Comprehensive data exploration with Python [3]”. This kernel presents a statistical way to solve the problem, which involves variable analysis, variable selection, simple and multiple linear regression model construction and assumption checking. Since there are many independent variables, we do not attempt to build a simple linear regression model, and thus we do not check assumptions for our multiple linear regression model since it is complex and requires a lot of statistical background.

Independent variable analysis is adapted from “Stacked Regression to Predict House Prices [4]”. This kernel describes how to impute missing data and transform skewed features for data preprocessing, and how to apply stacked regression models to achieve competitive performance.

# 3 Data Preprocessing

## 3.1 Data Exploration

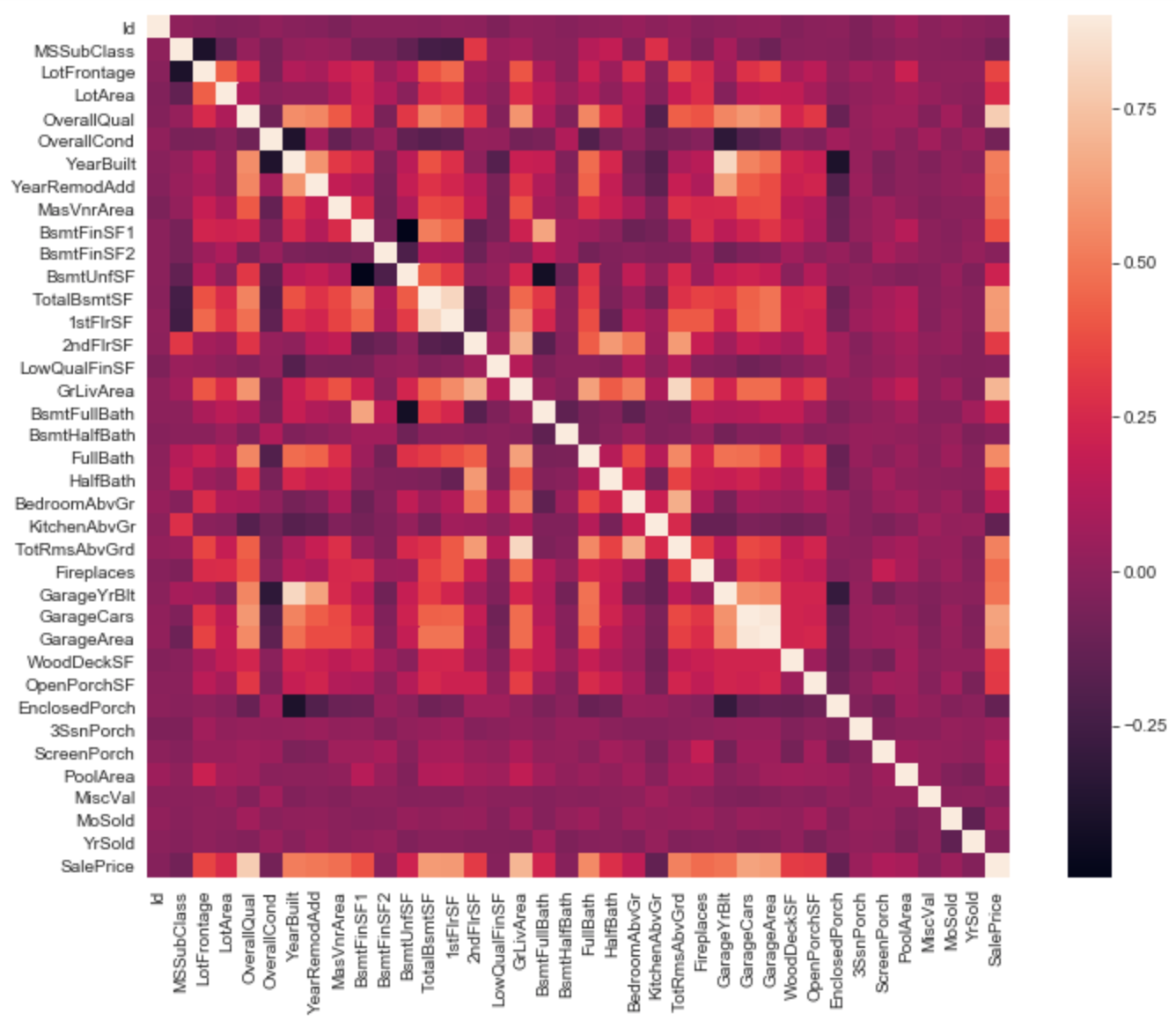
### 3.1.1 Overview

Before starting actual data preprocessing, we use visualization to explore dataset and get a taste of it. Since it is a regression problem, we need to take a look at relationship between dependent variable (house price) and independent variables, and we also want to see if there are correlations among independent variables.

We will use visualization to pick several most significant attributes to house price and we will take a look at those and to see if there are any outliers for those attributes. Since there are too many attributes, it is not realistic to go through all of them and eliminate outliers, we will just focus on outliers in the most correlated attributes since those attributes contribute most to the regression model. We will then build several regression models, and one of them is a multiple linear regression model using the most significant attributes selected during this phase. Also note that for other models, we will feed in all cleaned data and let the model to determine what variables to select.

### 3.1.2 Heatmap

Heatmap is used to visualize correlation coefficient (r) between a great amount of variables. Correlation coefficient used in linear regression indicates linear relationship between two variables. Recall r = 0 means no linear relationship, r = 1 means perfect positive linear relationship, and r = -1 means negative linear relationship.



As we can see in the heatmap, there are some white to salmon colored cells on the bottom, which indicates that OverallQual, GrLivArea, GarageCars, GarageArea, TotalBsmtSF and 1stFlrSF are the most significant variables (has correlation > 0.5) to house price. We can then take a look at a zoomed-in heatmap, which shows the top 9 variables correlated with house price. Since all attributes shown in this heatmap has high correlation with house price, we want to eliminate independent variables highly correlated with other independent variables.

Those correlations between independent variables are shown in salmon cells in heatmap. As we can see, GarageArea has high positive correlation (r = 0.88) with GarageCars. It makes sense that if the Garage is large, more cars could be parked. So we will drop GarageArea since GarageCars has higher correlation coefficient. And another salmon square indicates high positive correlation (r = 0.82) between 1stFlrSF and TotalBsmtSF. Again we drop 1stFlrSF since it has lower correlation coefficient compared to TotalBsmtSF. The last salmon square with high correlation (0.83) shows high correaltion between TotRmsAbvGrd and GrLivArea, so we eliminate TotRmsAbvGrd.

Therefore, now we only have 6 independent variables: OverallQual, GrLivArea, GarageCars, TotalBsmtSF, FullBath, YearBuilt. We will take a closer look at the relationship between house price and these independent variables using Scatterplot Matrix.

### 3.1.3 Scatterplot Matrix and Scatter Plot

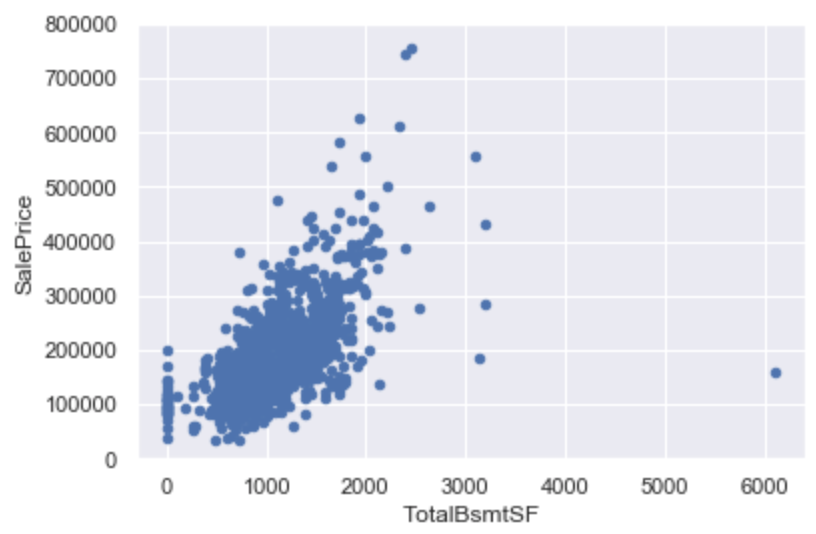
Scatterplot Matrix is a even detailed view of zoomed-in heatmap since it shows all data points so we can detect outliers. The variables from top to bottom are SalePrice, OverallQual, GrLivArea, GarageCars, TotalBsmtSF, FullBath and YearBuilt. We can see there are outliers in scatter plot of SalePrice vs GrLivArea and SalePrice vs TotalBsmtSF. So we can then zoom in again and generate scatter plot of SalePrice against those two independent variables.



In the scatter plot of SalePrice vs GrLivArea, we can see two obvious outlier in the lower right corner, these two houses have large GrLivArea but very low SalePrice. We need to delete them since they will harm the model. There are two more outliers on the upper right corner, but since they follow the trend of estimated regression line, we will keep them as they are.



In the scatter plot of SalePrice vs TotalBsmtSF, we can see one obvious outlier at TotalBsmtSF 6000. Since that instance is far away from the majority of data, we need to delete it. And other scatters generally follow linear trend, so we are good with that.



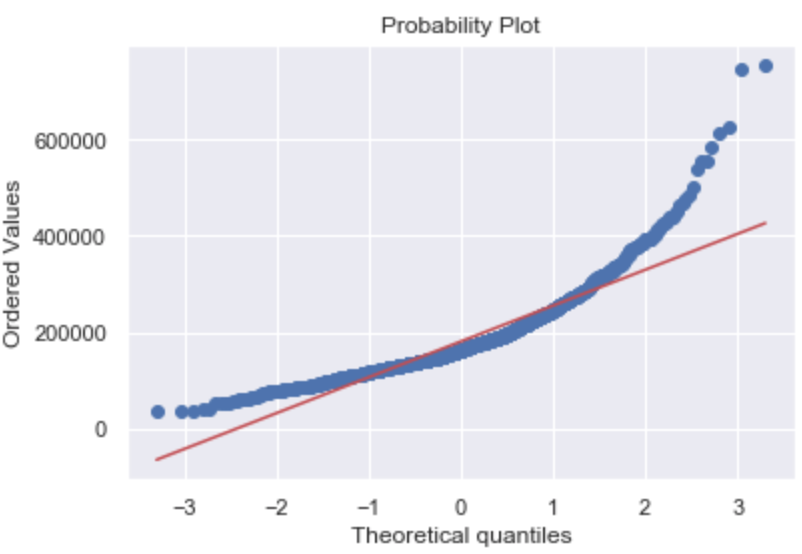
### 3.1.4 Wrap Up

To wrap up, in data exploration phase, we can generate another dataset with only 6 independent variables (OverallQual, GrLivArea, GarageCars, TotalBsmtSF, FullBath, YearBuilt) and 1459 iinstances since we deleted 2 outliers. This subdataset, selected using principles in statistics, will be used to build a multiple linear regression model, and this model is considered as human-selected and we will see if models built by different algorithms will have better performance than that. Note that we need to perform all data cleaning steps under section 3.3 for this subdataset before fitting it to the model as well.

## 3.2 Dependent Variable Analysis (SalePrice)

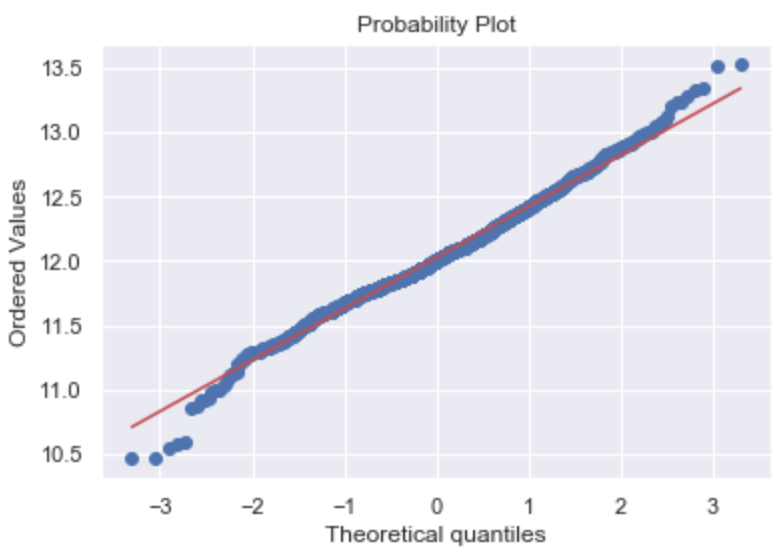
Since house price is the target variable, we need to take a look at it first. By looking at the histogram of house price, we can see the data is not normally distributed. Also the QQ plot does not look like a straight line, so the underlying distribution of house price is not normal.





In order to make models more accurate, we need to transform house price so that it follows normal distribution. Since the data is positively skewed, we try to apply log transformation to it [5]. After applying log transformation, histogram is closer to normal distribution and the QQ plot shows an approximately straight line.





## 3.3 Independent Variable Analysis

In section 3.3.1 to 3.3.5, we discuss data cleaning process for original training dataset, and in section 3.3.6, we talk about data cleaning process for small dataset which is selected in 3.1.4.

Since the training set is large, there might be a lot of missing values for independent variables. So we need to deal with missing data first before we step into independent variable analysis.

### 3.3.1 Missing Data

At the very beginning, without much investigation of original dataset, we delete all attributes with missing rate over 15%. And for attributes with missing rate lower than 15%, we impute them with the mode if they are categorical, and with the average if they are numerical. But it turns out the mean squared error of models are too low (about 1/10 of error we get now). It draws our attention so we find that we have to assess each attribute individually to impute it since we need to interpret missing values within the context of each attribute. Refined missing data imputing process is discussed as below.

We sort all attributes according to data missing rate and set the threshold to 15%, and we delete attributes with missing rate over 15%. So now we have 75 attributes. Note that data missing rate is computed as number of instances with NA over total number of instances.

All attributes with missing rate >15%:

PoolQC 99.588759

MiscFeature 96.298835

Alley 93.762851

Fence 80.740233

FireplaceQu 47.292666

LotFrontage 17.751885

For attributes with data missing rate greater than 0 but less than 15%, we need to process it one by one and to see how to impute them.

All attributes with missing rate < 15%:

GarageYrBlt 5.551748

GarageType 5.551748

GarageFinish 5.551748

GarageQual 5.551748

GarageCond 5.551748

BsmtExposure 2.604524

BsmtFinType2 2.604524

BsmtFinType1 2.535984

BsmtCond 2.535984

BsmtQual 2.535984

MasVnrArea 0.548321

MasVnrType 0.548321

Electrical 0.068540

For GarageYrBlt, though it looks like a numerical attribute, it is actually categorical (we will convert it to categorical attribute later), we try to impute missing values with the mode. An interesting finding is that the mode of GarageYrBlt is NA (which is not really surprising), so we impute NA with the most frequent non-null value.

For other GarageX attributes, since “NA” means “No Garage”, we impute missing values with None.

For BsmtX attributes, since “NA” mean “No Basement”, we impute missing values with None.

For MasVnrArea, since it is a numerical attribute, we impute missing values with the average.

For MasVnrType, since “None” means no masonry veneer, we impute missing values with None.

For Electrical, since it is a categorical attribute and there is no specific meaning for NA, so we impute missing values with the mode.

### 3.3.2 Transform “Numerical” Variables into Categorical [4]

We transform numerical variables that are actually categorical back into categorical, since they are not actually numerical, treating them as numerical values could harm the model. We decide to perform this before imputing missing data because we apply different strategies on imputing categorical and numerical data. There are 8 such variables: MSSubClass, OverallCond, OverallQual, YrSold, YearBuilt, YearRemodAdd, MoSold, GarageYrBlt.

### 3.3.3 Label Encoding Categorical Text Attributes [6]

Regression models does not accept text as input, so we need to encode text into numbers. Attributes to be converted are listed as follows:

'FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',   
 'ExterQual', 'ExterCond','HeatingQC', 'KitchenQual', 'BsmtFinType1',   
 'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlope',  
 'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClass', 'OverallCond',

'YrSold', 'MoSold'

### 3.3.4 Transform Skewed Attributes

We need to check if there are skewed features as we check skewness of house price. If there are skewed features, we apply Box-Cox transformation on them. This helps to build a more robust model.

### 3.3.5 Convert Categorical Data into One-hot Encoding [6]

We have already done label encoding for categorical data, but converting a text categorical attribute with label encoding makes that attribute looks ordinal, and that will mislead the regressor. So we need to convert those attributes into one-hot encoding.

After all data preprocessing steps, the dataset (of dimension 1459 x 392) is ready to be fed into models.

### 3.3.6 Data Cleaning for Small Dataset

Note that we also need to apply independent variable analysis for small dataset, which contains 6 independent variables selected in section 3.1.4. There is no missing data in this dataset, so we do not need to impute data. There is two numerical attributes that is actually categorical (OverallQual, YearBuilt), so we need to convert then to categorical. There is no text attribute so there is no need for label encoding, and there is no skewed attributes for transformation. There are two categorical attributes now (OverallQual, YearBuilt), so we need to apply one-hot encoding.

# 4 Data Mining

## 4.1 10-fold Cross Validation

Since this dataset is not really large (with about 1500 instances), and we cannot simply split it up into training and test set. Thus, 10-fold cross validation will be used to avoid overfitting.

## 4.2 Algorithms

There are 8 algorithms used in total and we apply them on both small dataset (of dimension 1459 x 6) and large dataset (1459, 392 ). The algorithms are ordinary least square regression [8], ridge regression, lasso regression, elastic net, SVR (Support Vector Regression [9]), XGBoost [10], gradient boosting regression and random forest. The first 5 algorithms are basic algorithms and the last 3 algorithms are ensemble algorithms.

Recall that we apply ordinary least square regression on small dataset and consider that as a baseline model (human-selected model), and we want to know if the models built with large dataset and other algorithms can beat it. The answer should be yes.

Algorithms used with tuned parameters are listed as follows:

* Ordinary least square regression:

regression linear\_model.LinearRegression(fit\_intercept=True, normalize=False, copy\_X=True,n\_jobs=None)

* Ridge regression:

linear\_model.Ridge(alpha=5)

* Lasso regression:

linear\_model.Lasso(alpha =0.0005)

* Elastic net:

Elastic Net(alpha=0.0004, l1\_ratio=.9, random\_state=2)

* SVR:

SVR(degree =11,gamma='scale', C=1.0, epsilon=0.1,shrinking=False)

* XGBoost:

xgb.XGBRegressor()

* Gradient boosting regression:

GradientBoostingRegressor(learning\_rate=0.05)

* Random forest:

RandomForestRegressor(max\_depth=12, random\_state=0,n\_estimators=100)

## 4.3 Metrics

A model is the combination of one algorithm and one dataset. Mean squared error and r^2 (R squared) are used to evaluate model performance. A model is considered having better performance if it has lower mean squared error and higher r^2.

# 5 Observations

Elastic net and lasso regression has the best performance among all models for both datasets. Other than ordinary least square regression, all models demonstrate better performance with the large dataset. Ordinary least square regression, lasso regression, ridge regression and elastic net all produce very similar results for small dataset. Due to computational power limit, the performance of ensemble models is not as good as basic models.

Parameters are tuned to achieve better performance. For simplicity, parameters used are shared by both datasets for each algorithm as given in section 4.2. One difficulty of tuning parameter, especially for ensemble models, is that tuning parameter requires a lot of time and, in most cases the performance does not have a linear relationship between parameters. So it is really hard for us to determine proper parameter values.

## 5.1 Basic Models

### 5.1.1 Ordinary Least Square Regression

Ordinary least square is used to fit a multiple regression model. Output generated are as follows. Note that ordinary least square regression fails to handle large dataset and gives absurd result since it does not involve regularization.

Applying ordinary least square regression on small dataset gives a reasonable output. Then we tried to tune model’s parameters (copy\_X, fit\_intercept, n\_jobs and normalize) , but it did not have much effect on output, so we simply use the model with default parameters. This model is considered as the baseline, with mean squared error 0.162.

**Ordinary least square regression score (large dataset):** mean squared error: 509779708.5001   
 R^2: -3448658133122633216.0000

**Ordinary least square regression score (small dataset):** mean squared error: 0.1620   
 R^2: 0.8325

### 5.1.2 Ridge Regression

Ridge regression is able to handle large dataset since it has regularization, and the performance of large dataset is better than that of small dataset. The performance on small dataset is very similar to the performance using ordinary least square, with tiny improve in . When we tune parameter, it turns out that parameter “alpha” has some effect on the output, and setting alpha = 5 gives better result than the default value of alpha = 0. When alpha increases, performance of both models increases, but the model with large dataset always perform better than the model with small dataset. Output of model with alpha = 5 is as follows:

**Ridge regression score (large dataset):** mean squared error: 0.1189   
 R^2: 0.9091  
  
**Ridge regression score (small dataset):** mean squared error: 0.1620   
 R^2: 0.8326

### 5.1.3 Lasso Regression

Compared to ridge regression, lasso regression gives very close result for large dataset, and it provides a little improvement for small dataset. Same as ridge regression, lasso gives better performance on large dataset. Similarly, we tuned the parameter “alpha” in order to have better results. Interestingly, for this model, output of two datasets do not perform the same during parameter tuning. As we change the alpha, performance of large dataset is enhanced while performance of small dataset remains unchanged. For large dataset, alpha of 0.0005 gives the optimal result while other alphas give less competitive results. Output of alpha = 0.0005 is given as follows:

**lasso regression score (large dataset):** mean squared error: 0.1171   
 R^2: 0.9120  
  
**lasso regression score (small dataset):** mean squared error: 0.1620   
 R^2: 0.8325

### 5.1.4 Elastic Net

Elastic net gives very similar output as ridge and lasso regression. We also tuned parameters for it and find it has similar behaviour during tuning as lasso regression. That is, performance of small dataset is not affected by parameters and the model with large dataset performs better than the model with small dataset.

However, elastic net has more parameters to tune compared to ridge and lasso regression. Other than ‘alpha’, it has one more parameter called ‘l1\_ratio’**.** After a few attempts, we find alpha = 0.004 and l1\_ratio = 0.9 gives fairly good result. Note that other l1\_ratio values may give better outputs, but the difference is very minor. The parameters which we did not mention has minor impact on the outputs or no effect at all. Again, outputs are given for tuned parameters:

**Elastic net score (large dataset)**:  
 mean squared error: 0.1171   
 R^2: 0.9117  
  
**Elastic net score (small dataset):**  
 mean squared error: 0.1620   
 R^2: 0.8325

### 5.1.5 SVR (Support Vector Regression)

SVR is very different from previous regression models. For large dataset, the mean squared error of SVR (0.13) is larger than mean squared error of models discussed above (0.12). Besides, the performance is very different between large dataset and small dataset. The output from large dataset is significantly better than the small dataset.

The parameter tuned here is “degree”. We increase degree from default value of 0 to 11 to get better output. For other parameter like “kernel”, changing it to other functions (e.g. polynomial or linear) requires large amount of calculation time and the output did not change significantly for large dataset. And tuning kernel for small dataset freezes the program and gives no output. It could be due to lack of computational power or SVR with kernel is not really applicable for small dataset. Since the computation requires large amount of time, we simply left “kernel” as default. Outputs from tuned parameters are given as below:

**SVR score (large dataset):** mean squared error: 0.1382   
 R^2: 0.8777  
  
**SVR score (small dataset):** mean squared error: 0.2400   
 R^2: 0.6339

## 5.2 Ensemble Models

It is interesting to find the difference in performance of two datasets are smaller in ensemble models than that in basic models. But the output of ensemble models are not as good as basic models due to lack of computational power.

### 5.2.1 Gradient Boosting Regression and XGBoost

For Gradient boosting regression and XGBoost, both of them requires very large amount of time (about 10 to 20 minutes) to run during parameter tuning since these two models are comp. So we only picked some parameters which require less time to produce output. However, the output does not tend to be much better or worse for specific parameters, and actually it tends to be quite random.

With regard of current computational power, tuning parameters does not have much effect on model performance for gradient boosting regression and XGBoost and the performance of these two algorithms is even worse compared to almost all other algorithms. Though we expect better performance with ensemble models, we do not have such strong computational power. With proper parameters set, gradient boosting and XGBoost gives quite similar result as other models [4]. Though data preprocessing steps are a bit different between this project and referenced kernel, it is enough to illustrate the output of these two algorithms.

**Gboost regression score (large dataset):**  
 mean squared error: 0.1393   
 R^2: 0.8764

**Gboost regression score (small dataset):** mean squared error: 0.1588   
 R^2: 0.8395

**XGBoost score (large dataset):** mean squared error: 0.1327   
 R^2: 0.8878  
  
**XGBoost score (small dataset):**  
 mean squared error: 0.1592   
 R^2: 0.8382

### 5.2.2 Random Forest

Random forest is the most valuable model because it has magnificent improvement after parameter tuning. First we increased the max depth, which has good impact on both small and large dataset. Like what we observed on other models, when the parameter is greater than a specific value (max\_depth = 12 in this case), the improvement becomes minor and uncertain. So we set max depth = 12. Another parameter to tune is “n\_estimators”. There is a positive relationship between performance and n\_estimators, i.e. performance will increase with larger n\_estimators. Unfortunately, as we raise the number of estimators , the computation time increases significantly as well. So we set it as 100 which ensures that the result is given within reasonable running time.

Performance with default parameter

**Random forest score (large dataset):**  
 mean squared error: 0.2386   
 R^2: 0.6373  
  
**Random forest score (small dataset):**  
 mean squared error: 0.2330   
 R^2: 0.6553

Performance with tuned parameter

**Random forest score (large dataset):** mean squared error: 0.1448   
 R^2: 0.8669  
  
**Random forest score (small dataset):** mean squared error: 0.1640   
 R^2: 0.8286

# 6 Conclusion

Through this project, we practiced common practice of data exploration with visualization, target variable analysis, independent variable analysis, data cleaning and model fitting. Most time and effort are spent during data preprocessing phase. It is desired since data cleaning is a very important step in data mining and models built with dirty data gives misleading results. If there are missing values in dataset, models cannot be built and errors will be returned. If the data preprocessing is improper, models built are not meaningful or accurate.

From the observations above, we can conclude that other than ordinary least square regression, all models have better performance on the large dataset. Elastic net and lasso regression has the best performance among all models for both datasets. Due to computational power and time limit, the performance of ensemble models is not as good as basic models.

Another lesson learned is that parameter tuning is very time-consuming, especially for complex models. For this project, most tuned parameters has very small or no impact on small dataset. For ensemble models, there are more parameters to tune and outputs rely on parameters. However, increase in parameters does not lead to steady increase in performance. So it is hard to determine parameter values.

# References

[1] <https://www.kaggle.com/c/house-prices-advanced-regression-techniques>

[2] <https://www.kaggle.com/c/house-prices-advanced-regression-techniques/kernels>

[3] <https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python>

[4] <https://www.kaggle.com/serigne/stacked-regressions-top-4-on-leaderboard>

[5] <http://desktop.arcgis.com/en/arcmap/latest/extensions/geostatistical-analyst/box-cox-arcsine-and-log-transformations.htm>

[6] <https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621>

[7]

<https://jorisvandenbossche.github.io/blog/2018/05/28/scikit-learn-columntransformer/>

[8] <https://scikit-learn.org/stable/modules/linear_model.html>

[9] <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>

[10] <https://scikit-learn.org/stable/modules/ensemble.html>