Predicting House Prices on Kaggle Competition

Mengqi Fan
M.S. in Business Analytics

Abstract

In this data science project, the house prices on Kaggle competition is selected and the main task is to use training data set provided by Kaggle to train and optimize our model, and then use the model constructed and the testing data set to make prediction for the final house prices. We went through the whole process from preprocessing and exploratory data analysis to modeling and finally, make predictions. Meanwhile, three useful kernels on this topic have been chosen and evaluated.

Table of Contents

1. Executive Summary

1.1 Problem Statement

In this data science project, the "House Prices: Advanced Regression Techniques" in Kaggle competition is selected. The objective of this contest is to use data that contains 79 features describing almost all aspect of a house to train a model and predict the final house price for each home as accurate as possible. The related task for this project include:

- ♦ Preprocessing and exploratory data analysis — This is a process of organizing, plotting, and summarizing a data set. The process of EDA includes: examine the data generally, examine each field/column individually, address anomalies in data, transform data to make it more useful.
- ♦ Evaluation of 3 other solutions — Three Kaggle solutions completed by others for this contest are chosen. Key features used, modeling approach and performance in each solution are summarized and evaluated.
- ♦ Modeling — Creating models to examine the relevance of different independent variables and different algorithms.

1.2 Datasets

There are four files provided on Kaggle and we mainly focus on the train.csv file (the training set) and test.csv file (the testing set). The training set contains 1460 observations with 79 variables including the sale price of each house and other 78 variables that help describe nearly all aspects of a house (Lot size, Type of utilities, Physical locations, etc.) in Ames, Iowa, for each observation. The testing set contains 1459 observations with 78 explanatory variables. It can be seen from the training set and testing set that both of them have a large number of missing values. Besides, both categorical and numerical variables exist in the datasets. The main goal for this project is to use the model trained by using the training set to predict the sale price for each observation in the testing set.

1.3 Metric

The Root Mean Squared Logarithmic Error (RMSE) will be used to as the evaluation metrics for this project. The RMSE can be expressed as (image from https://medium.com/human-in-a-machine-world/mae-and-RMSE-which-metric-is-better-e60ac3bde13d):

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

The logarithm of the predicted sales price and the logarithm of the observed sales price will be used for evaluation. This measurement is useful when there is a wide range in the target variable, and we do not necessarily want to penalize large errors when the predicted and target values are themselves high. Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.

1.4 Findings

In the first section, by doing the basic summary and looking at samples of data, we find that there are both numerical and categorical data in the training and testing data set. Meanwhile, there are large numbers of missing values in many features. Thus for the data cleaning process we replaced variables with 0, None, mode and median according different meaning in each features. Correlation coefficients are also calculated in order to find the variables that are highly related with the target variable —sale price. The top 5 features are OverallQual, GrLivArea, GarageCars, GarageArea and TotalBsmtSF. We detected outliers by making scatter plot between these variables and the target variable and then we found there are some outliers in GrLivArea and TotalBsmtSF.The outliers are deleted as they does not follow the overall trend of graph. Furthermore, we did some feature engineering such as creating new features, transferring feature types, label encoding some features and finally, get dummies. For the modeling process, 10 different algorithms are used for the training data set and we finally constructed a stacking model as our final model. The details will be discussed in the following subsections.

2. Data description and initial processing

2.1 Basic Summary of Data

The first step is to import the training and testing dataset. From the two dataset, we can see that the train file contains a wide variety of information that might be useful in understanding the sale price. It also includes a record of sale price for each house. The test file contains all of the columns of the first file except sale price for each house. Our goal is to predict sale price.

After the importing process, we use head( ) and tail( ) function to look at a sample of training data and testing data — the first few rows and last few rows. By using train.shape and test.shape we know that there are 1460 rows and 81 columns in the training dataset and there are 1459 rows and 80 columns in the testing dataset. Then we use train.index and test.index to get the row information. From the result, we know that the training set has 1460 rows and testing set has 1459 rows, which means the training dataset has 1460 different house information (1459 for the testing set). Then use train.columns and test.columns to check the column names. There are information from 78 different aspects to help us assess the house price. In order to identify numerical and categorical variables, the train.select_dtypes and test.select_dtypes are used. The results from training set are shown below (Figure 1 & Figure 2):

```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF',
       'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
       'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

Figure 1. Numerical Variables in Training Set

```
Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

Figure 2. categorical variables in Training Set

Figure 1 shows the 36 numeric variables that describe different aspects of house features (exclude Id and SalePrice column); Figure 2 shows 43 categorical variables describing house features. Finally, to get overall description of my data, use the describe() function to get various summary statistics on numeric columns that exclude NaN values.

2.2 Missing Values

First, use train.info( ) and test.info( ) to briefly look at the total number of non-nulls in each column from both training and testing set. From the result we can see that there are great amount of missing values in both dataset. Then use train.isnull( ).sum( ) and test.isnull( ).sum( ) to show the number of missing variables. For example, by sorting missing values, it can be seen that missing values in training set exist in these columns (Table 1).

Based on the result and null value description above, we find that for these features: PoolQC, MiscFeature, Alley, Fence, FireplaceQu, GarageType, GarageFinish, GarageQual, GarageCond, BsmtFinType2, BsmtExposure, BsmtFinType1, BsmtCond, BsmtQual, MasVnrType, the null value means that the house does not have these parts. Therefore, we can replace the missing values with "None".

Feature Name	Number of NAs	Description	Feature Name	Number of NAs	Description
PoolQC	1453	No Pool	*GarageCond*	81	No Garage
MiscFeature	1406	None	*BsmtFinType2*	38	No Basement
Alley	1369	No alley access	*BsmtExposure*	38	No Basement
Fence	1179	No Fence	*BsmtFinType1*	37	No Basement
FireplaceQu	690	No Fireplace	*BsmtCond*	37	No Basement
LotFrontage	259	NA	*BsmtQual*	37	No Basement
GarageYrBlt	81	NA	*MasVnrArea*	8	NA
GarageType	81	No Garage	*MasVnrType*	8	None
GarageFinish	81	No Garage	*Electrical*	1	NA
GarageQual	81	No Garage			

Table 1. Summary of Missing Values Training Set

Next, consider the feature GarageYrBlt (Year garage was built) and MasVnrArea (Masonry veneer area in square feet). From the table we found that the number of missing values is equal to the number of missing values in "garage" (81) and "MasVnrType"(1). Thus we can conclude that the missing values in these two features are the result from the situation that the house does not have garage or masonry veneer area. Therefore, we can replace missing values in GarageYrBlt with "None" and replace missing values in MasVnrArea with 0.

For LotFrontage (Linear feet of street connected to property), since it is likely to have a similar area to other houses in its neighborhood, replace missing values with the median LotFrontage of the neighborhood. For Electrical (Electrical system), since this is categorical feature, we can replace it with the value that has most frequency in this column. By using pd.value_counts( ) we found that the value with most frequency is "SBrkr".

Missing Values in testing set are shown below (Table 2). For variables with missing values the same as in training set, we use the same method to deal with them. Besides, there are some other variables with missing values and the description of them are not provided by Kaggle.

Feature Name	Number of NAs	Description	Feature Name	Number of NAs	Description
PoolQC	1456	No Pool	*GarageCond*	78	No Garage
MiscFeature	1408	None	*BsmtFinType2*	42	No Basement
Alley	1352	No alley access	*BsmtExposure*	44	No Basement
Fence	1169	No Fence	*BsmtFinType1*	42	No Basement
FireplaceQu	730	No Fireplace	*BsmtCond*	45	No Basement
LotFrontage	227	NA	*BsmtQual*	44	No Basement
GarageYrBlt	78	NA	*MasVnrArea*	15	NA
GarageType	76	No Garage	*MasVnrType*	16	None
GarageFinish	78	No Garage	*MSZoning*	4	NA
GarageQual	78	No Garage	*BsmtFullBath*	2	NA
BsmtHalfBath	2	NA	*Utilities*	2	NA
Functional	2	NA	*Exterior2nd*	1	NA
Exterior1st	1	NA	*SaleType*	1	NA
BsmtFinSF1	1	NA	*BsmtFinSF2*	1	NA
BsmtUnfSF	1	NA	*KitchenQual*	1	NA
GarageCars	1	NA	*GarageArea*	1	NA
TotalBsmtSF	1	NA			

Table 2. Summary of Missing Values Testing Set

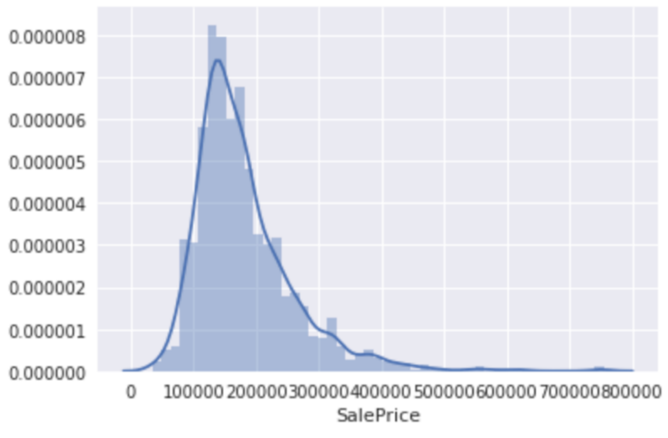
As a result, for numeric variables (BsmtFullBath, BsmtHalfBath, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, GarageCars, GarageArea, TotalBsmtSF), replace the missing values with 0; For categorical variables (MSZoning, Functional, Exterior2nd, Exterior1st, SaleType, KitchenQual), replace them with the mode value. As for "Utilities", the missing value may mean "no utilities", thus replace the missing value with None.

Finally, run the code again and check if there are still missing values left. The result shows that we now have no missing values in training and testing set.

2.3 Visualization and Correlation Analysis

First, plot histogram for the dependent variable in this dataset ——SalePrice. The histogram of the sale price is shown in Figure 3 on the left. It can be seen that the plot is right skewed. In order to build a linear model, it is necessary to transform it into normally distributed. Here we use log transform and the result is shown in Figure 3 on the right. Meanwhile, from calculation we found that the skewness for the sale price data is 1.8829 and Kurtosis is 6.5363.

Second, calculate correlation coefficient between independent variables and SalePrice in training data set and choose the top 5 variables that have the largest pairwise correlation with sale price. A correlation matrix (Figure 4) is also created to get a quick overview of the variables and its relationships. From the result, we find these 5 variables are OverallQual (0.790982), GrLivArea (0.708624), GarageCars (0.640409), GarageArea (0.623431) and TotalBsmtSF (0.613581). Then, create scatter plots between 'SalePrice' and the 5 correlated variables.



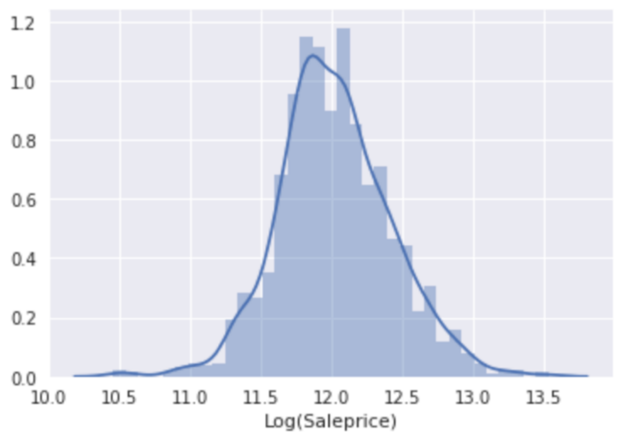


Figure 3. Histograms of House Sale Price

The scatter plots are shown in notebook. From the results, we can get some ideas about relationships between variables. For example, it seems like that there are not obvious relationships between most of the independent variables. Also, for the scatter plot between GrLiveArea vs TotalBsmtSF, there is a line constructed by the dots and most of the data are above this line. In reality, this should be reasonable. Since most of the house has larger above ground living area than basement area. The next step is to visualize the relationship between the 5 variables we picked and the target variable — SalePrice. We created boxplot for OverallQual, GarageCars features and scatter plot for other three features. The result is shown in Figure 5 and Figure 6.

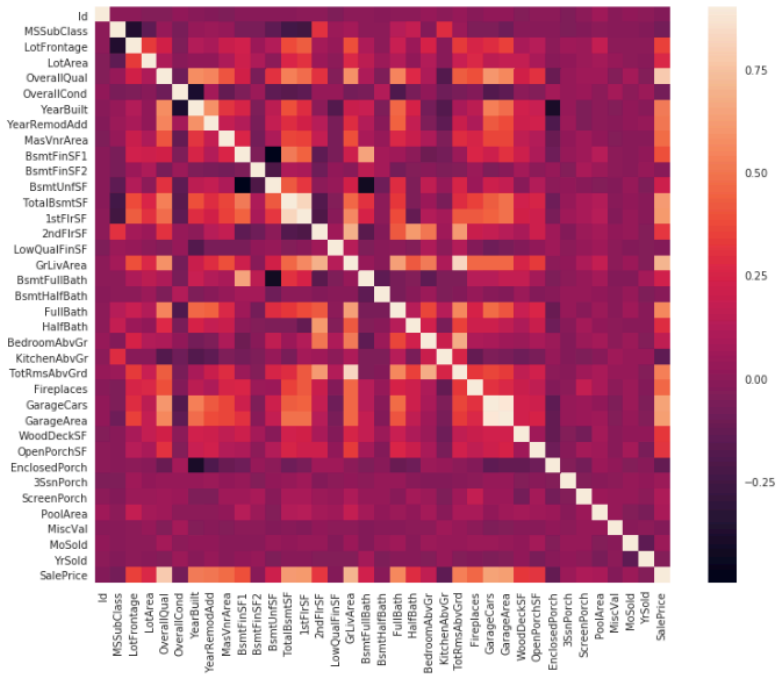


Figure 4. Correlation Matrix

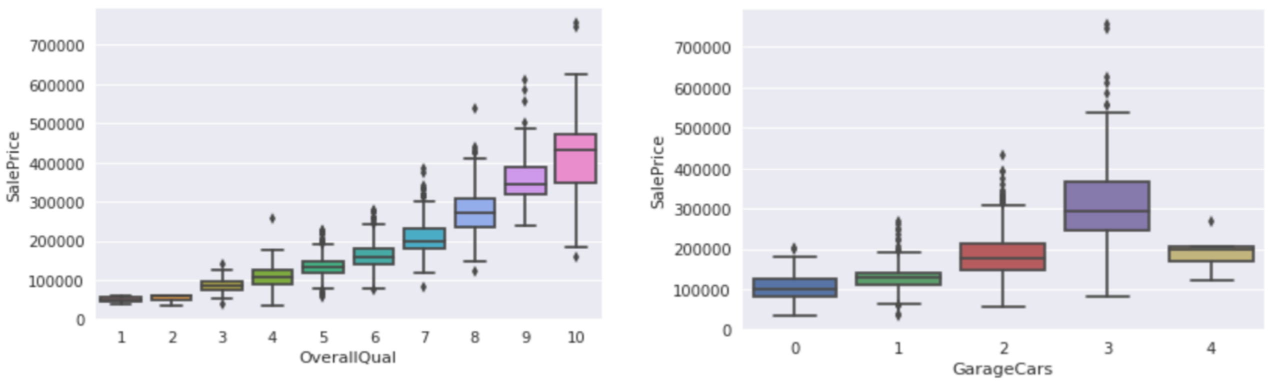


Figure 5: Boxplots of SalePrice vs OverallQuall (Left) and SalePrice vs GarageCars (Right)

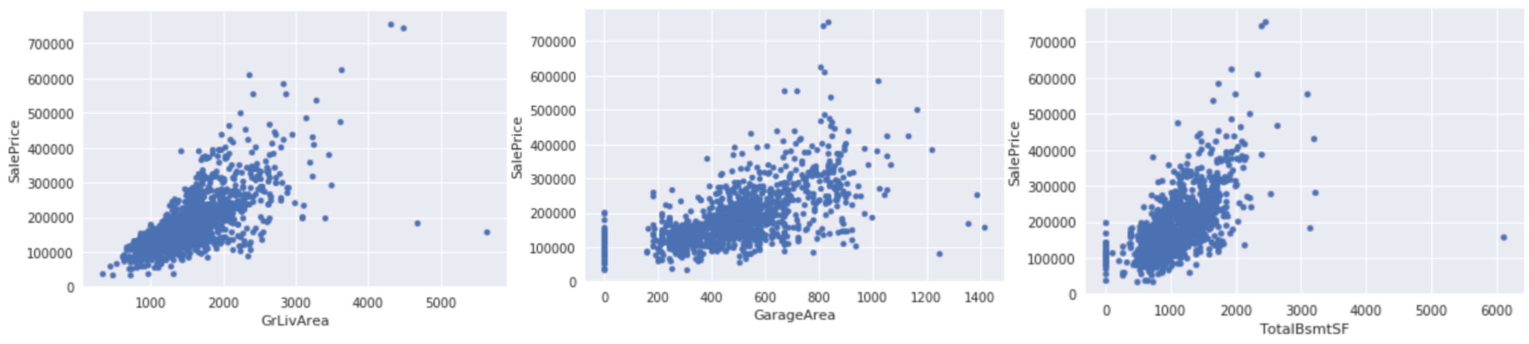


Figure 6: Scatter Plots of SalePrice with Numeric Features (GrLivArea, GarageArea, TotalBsmtSF)

From the boxplots, the sale price increases with the increase of rates of the overall material and finish of the house. As for size of garage in car capacity, the most expensive houses have the garage that has space for 3 cars and the cheapest houses no garage space for cars. From the scatterplot, it can be seen that the sale price increases with the increase of these three types of area.

3. Modeling and Evaluation of Other Solutions

In this section, 3 solutions completed by others on Kaggle have been chosen and identified based on the key features selected, modeling approach and performance.

3.1 Solution 1: Predicting House Prices with Regression
(https://www.kaggle.com/samsonqian/predicting-house-prices-with-regression)

In this solution, there are mainly six parts: Importing packages, loading and inspecting data, imputing null values, feature engineering, creating, training, evaluating, validating, and testing machine learning models and finally, submission.

In the first part, pandas and numpy are imported to deal with numbers and data; seaborn and matplotlib are imported to visualize data. In the second part, training and testing sets are loaded and some description functions are used at first to look at the sample of the data with shape, feature names and statistical information. Then, the author calculated the correlation coefficients for each feature with the sale price to prepare for the modeling part.

In the third part, data cleaning is processed. It can be found from the data set that both categorical variables and numeric variables have large amount of missing values. For the columns that "NaN actually means something", missing values are replaced by None; For the numeric feature "LotFrontage", the missing values are dropped. For the left features that have large amount of missing values (GarageYrBlt, MasVnrArea, and MasVnrType), the author replaces the NAs in categorical features with "None" and replaces NAs in other two features with median value in the column. Last, for features with fewer missing values, the NAs are replaced with medians (for numeric features) or modes (for categorical features). After all these process done, the training and testing sets are checked again to make sure there are no missing values. The next part is feature engineering. The first step is to show the distribution of sale price of different houses and after the log transformation, the distribution becomes more normalized. Then the author deals with categorical data by changing all of them into representative numbers. Finally, the data set is ready to build models.

The next section is mainly about training and evaluating models. The target variable is sale price. However, the value need to be log transformed to be normalized (TransformedPrice). Then, make train-test split to prepare for modeling process. In the modeling process, 5 different regression models (Linear Regression, Ridge, Lasso, Decision Tree Regressor, Random Forest Regressor) are built and trained. For each model, four types of scores are evaluated: best score, R-square score, RMSE score and cross validation score. By sorting with different types of scores, the author chooses the Random Forest Regressor as it has the highest R-square score and lowest RMSE score. The final section is to use the Random Forest Regressor chosen by the author to make prediction on the testing set. The performance for this model is 0.14783.

Further research : For the data cleaning part, I find that the author droped all the missing values in the LotFrontage column. This method may not be very good since there are still large amount of missing values in this columns, which means we cannot generate predictions with missing, loss of much data. Thus for this column, I replace the missing values with mean value (70.0) in the column. Keep other code unchanged and calculate the scores for the Random Forest Regressor. The result shows that for my approach, the R squared score is the same with author's approach, while the best score of my approach is lower and so has a higher RMSE. Thus my method of replacing the missing value with mean value in LotFrontage column is worse than the authors approach of directly dropping NAs.

3.2 Solution 2: House Prices EDA to ML (Beginner)

(https://www.kaggle.com/dejavu23/house-prices-eda-to-ml-beginner/notebook)

This solution includes 4 parts: imports, settings and switches, global functions; exploratory data analysis; data wrangling; scikit-learn basic regression models and comparison of results.

In the first section, important packages such as numpy, pandas and seaborn. Especially, in this part, some settings for optimal performance and runtime are pre-defined for later process. Finally, three useful functions such as calculate the best score for the modeling part and print correlation coefficient are define as well. The training set and testing set are loaded. The second section is mainly for EDA. To start, the author use some basic summary functions such as shape, info, head and describe to take a look at the datasets and figure out how many features and missing values in training and testing sets. Second, the distribution plot is normalized by making a log transformation. Then, names of numeric and categorical features are checked. To deal with missing values, there are mainly two steps. The first step is replace "NAs that has some meaning" in columns with None; The second step is replace the remaining NAs with modes (for categorical variables) or mean values (for numeric variables). Apart from saleprice, the author finds that there are still other variables that are not normally distributed. Therefore, the log transformations are made with these variables to normalize the data. The next part is to calculate and check correlation of different features with the target variable ——SalePrice_Log. In this part, the correlation plot of different numeric variables with target variable are plotted and certain number of numeric variables with strong correlation with target variable are chosen for the data wrangling section and correlation matrix between these variables and target variable is created. Next, boxplot of different categorical variables with target variable are created and 10 categorical variables with strong correlation with target variable are chosen.

In the third section, columns with weak correlation with target variable are dropped. For categorical features that have strong correlation with target variable, the categorical value is represented and replaced by numeric value. Then re-calculate the correlation between all the features left with target variable and drop the features that have weak correlation. After this step, the correlation matrix with all features left is created. Finally, for the features that are similar, drop the one that has lower correlation coefficient and by now the data has been leaned and transformed. Then a new list of features is created for the final modeling part. The fourth part is the modeling part, in which the author uses 8 regressors (linear regression, ridge, lasso, elastic net, stochastic gradient descent, decision tree regressor, random forest regressor and SVR) from scikit-learn to train the model and compare the RMSE score for different model used. Finally, correlation of different model results are calculated and based on the result, the author chooses the two models (Random Forest and Stochastic Gradient Descent) and calculate the mean results based on these two model prediction results. Performance for this solution is 0.14857.

3.3 Solution 3 Simple Stacking Approach: Top 15% Score
(https://www.kaggle.com/josh24990/simple-stacking-approach-top-15-score)

This solution mainly includes 5 sections: introduction, data processing, feature engineering, machine learning and final predictions.

In the first section, key packages such as pandas, numpy, seaborn (for visualization), sklearn (for machine learning) are imported and the training and testing datasets are loaded. Then some basic functions such as info, head are used to inspect the two dataset. Especially, "Id" column is dropped in this section as it is not useful in the modeling part. The second section is data processing. The author first checks the outliers in the GrLivArea feature and then makes plots to show the outliers and deletes them. Second, histograms for the target variable (Saleprice) are made and for the same reason with other solutions, a log transformation is made on the target variable to prepare for modeling part.

The third section is feature engineering, which can be divided into two parts: dealing with missing data and converting variables. In the first part, percentage of NAs in each column is calculated and visualized by using a bar chart. Then, missing values in different columns are replaced with None, 0, mode, and median based on the meaning and class (numeric or categorical) of each feature. As for variable conversion part, several steps are processed: first, some numeric variables that should actually be categorical variables (MSSubClass, OverallCond, YrSold, MoSold) are converted into the right type; Second, variables with skewd data are log transformed; Third, variables with same meaning are combined and new variables are created (TotalSF). Fourth, features where a class is over 97% represented are dropped from dataset. Finally, features that will be used for modeling part are picked and show as below:

```
cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'KitchenQual', 'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlope',
        'LotShape', 'PavedDrive', 'Alley', 'CentralAir', 'MSSubClass', 'OverallCond',
        'YrSold', 'MoSold')
```

Figure 7: Features Used in Modeling

Then, the author uses LabelEncoder and get_dummies to deal with categorical data in these columns and finally data for modeling part is well prepared.

The fourth section is machine learning, which includes three phases: initiating algorithms, fitting algorithms and stacking algorithms. Another special part in this solution is to set key tuning parameter in generalized linear models by calculating the cross-validation 'RMSE' error, following 10-folds. The objective for this process is to ensure that all RMSE scores produced have been smoothed out across the entire dataset and are not a result of any irregularities, which otherwise would provide a misleading representation of model performance. In this process tuning parameter for Ridge regression, Lasso regression, ElasticNet regression and Kernel ridge regression are picked and models are fitted. Then several gradient boosting algorithm such as Gradient boosting, XGBoost, LightGBM, CatBoost are used and regressor models are initiated. In the second phase, all models are fitted and a plot is created to show the rank of performance in each model. From the result, it can be seen that all models performed very well. In the third phase, All the models are used to fit and predict the target variable in testing set. The final section is to create the stacked model. The model is created by the following method (CatBoost model not included):

stacked = (lasso_pred + elastic_pred + ridge_pred + xgb_pred + lgb_pred + krr_pred + gbr_pred) / 7

By running the codes, it can be seen that the performance for this stacked model is 0.11635.

3.4 Summary

In the first solution, all variables in training dataset (except Id and SalePrice column) are used for modeling and the target variable are log transformed to get better performance. During modeling process, 5 different regression models (Linear Regression, Ridge, Lasso, Decision Tree Regressor, Random Forest Regressor) are built and trained. By sorting performance scores, the author finally chooses the Random Forest Regressor to make prediction on the testing set. The performance for this model is 0.14783.

In the second solution, variables for modeling are picked by calculating correlation with target variables and choose features with most strong correlation. Features used for the Regressors are:

OverallQual, GrLivArea_Log, NbHd_num, GarageCars, ExtQ_num, KiQ_num, BsQ_num, TotalBsmtSF, FullBath, YearBuilt, YearRemodAdd, Fireplaces, MasVnrArea, MSZ_num.

In modeling part, linear regression, ridge, lasso, elastic net, stochastic gradient descent, decision tree regressor, random forest regressor and SVR from scikit-learn are used to train the model. Then the author choose model pairs by calculating correlation of different model result. Finally, the author chooses Random Forest and Stochastic Gradient Descent as stacked model and calculate the mean based on these two model to predict target value in testing set. Performance for this solution is 0.14857.

In the third solution, variables for modeling are picked mainly by feature engineering. In the training set, features where a class is over 97% represented are dropped form dataset. Meantime, new column is created by combine columns that have same meaning. Features that will be used for modeling part are picked and show as below:

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

The stacked model is used to predict target variable in testing dataset. The model is created by the following method:

stacked = (lasso_pred + elastic_pred + ridge_pred + xgb_pred + lgb_pred + krr_pred + gbr_pred) / 7

Performance for this solution is 0.11635, which means of all these three solutions, this is the best for the lowest RMSE score. Stacking is a good method since it combines models and thus allows the best elements of their predictive power on the given problem, thus smoothing over any gaps left from an individual model and increasing the likelihood of stronger overall model performance. Therefore, for my model I used the stacking method. Table below is a summary of the information for these three solutions.

Solution	Features	Modeling Approach	Performance
1	All Features (Except Id, SalePrice)	Random Forest	0.14783
2	OverallQual, GrLivArea_Log, NbHd_num, GarageCars, ExtQ_num, KiQ_num, BsQ_num, TotalBsmtSF, FullBath, YearBuilt, YearRemodAdd, Fireplaces, MasVnrArea, MSZ_num	Random Forest Stochastic Gradient Descent	0.14857
3	All Features (Except Street, Utilities, Condition2, RoofMatl, Heating, PoolQC)	Stacked Model (Lasso, Elastic Net, Ridge, XGBoost, LightGBM + Kernel Ridge + Gradient Boosting)	0.11635

Table 3: Summary of the Three Solutions

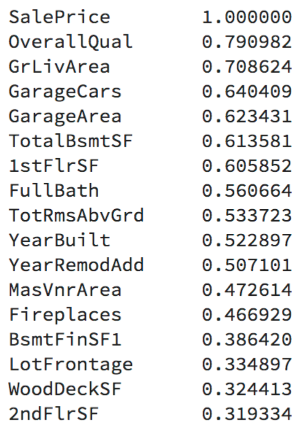
4. Feature Analysis and Engineering

4.1 Outliers

From the result in the visualization part, the top 5 variables that have the largest pairwise correlation with sale price are OverallQual, GrLivArea, GarageCars, GarageArea and TotalBsmtSF. From the scatter plot of SalePrice with GrLivArea and TotalBsmtSF, it is found that there are some outliers in these two plot. In the first plot, the two values with bigger 'GrLivArea' seem strange and they are not following the crowd; In the second plot, there is one value with bigger ‘TotalBsmtSF’ not following the crowd. As a result, these outliers in training set are deleted.

4.2 Feature Creation

From the result of correlation coefficient calculated between variables and sale price (Figure 8), it can be seen that features that are related to area are highly correlated with sale price. Thus a new feature named ‘TotalSF’ (measured as the sum of 'TotalBsmtSF', '1stFlrSF' and '2ndFlrSF') meaning “total square feet” is created. Meantime, a new feature named ‘Total_House_Quality’ (measured as sum of 'OverallQual' and 'OverallCond') is created.



```
SalePrice        1.000000
OverallQual      0.790982
GrLivArea        0.708624
GarageCars       0.640409
GarageArea       0.623431
TotalBsmtSF      0.613581
1stFlrSF         0.605852
FullBath         0.560664
TotRmsAbvGrd     0.533723
YearBuilt        0.522897
YearRemodAdd     0.507101
MasVnrArea       0.472614
Fireplaces       0.466929
BsmtFinSF1       0.386420
LotFrontage      0.334897
WoodDeckSF       0.324413
2ndFlrSF         0.319334
```

Figure 8: Top Correlation Coefficient between Features and SalePrice

4.3 Feature Justification

By analyzing the meaning and data types of the independent variables, it can be seen that some numeric variables that have special meaning should be transformed into categorical variables. The list of these variables are as below:

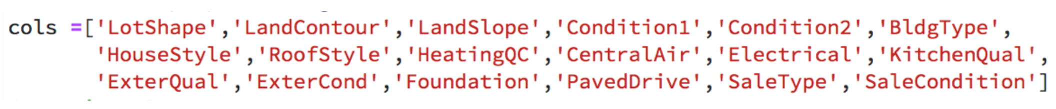
MSSubClass: Identifies the type of dwelling involved in the sale

YrSold: Year Sold

MoSold: Month Sold

4.4 Further Steps before Modeling

For real-world data, there are a lot of features that are heavily skewed. Transformation technique is useful to stabilize variance, make the data more normal distribution-like, improve the validity of measures of association. Hence we transformed the skewed numeric features by taking log(feature + 1). For some categorical variables that may contain information in their ordering set (Figure 9), Label Encoding (which is a utility class to help normalize labels such that they contain only values between 0 and n_classes-1) is used on these features.



```
cols =['LotShape','LandContour','LandSlope','Condition1','Condition2','BldgType',
       'HouseStyle','RoofStyle','HeatingQC','CentralAir','Electrical','KitchenQual',
       'ExterQual','ExterCond','Foundation','PavedDrive','SaleType','SaleCondition']
```

Figure 9: Categorical Variables Using Label Encoding

Finally, create dummy variables for all the categorical features in the dataset.

5. Modeling

5.1 Initial Attempts

Before modeling, separate y as the dependent variable and X as the matrix of independent variables. Use train_test_split (test_size = 0.3) to split the training set into training data and validation data. It can be see that the X_train has 1019 rows and 361 columns and X_test has 438 rows and 361 columns. Then, define a function (RMSE_cv) that returns the cross-validation RMSE error, following 10-folds.The goal is to ensure all RMSE scores produced are smoothed out across the entire dataset and are not a result of any irregularities, which would provide a misleading model performance.

First I used the Naïve Model and take average of sale price in training data (y_train) as the prediction value. Then use the prediction value and value from y_test to calculate R-square and RMSE, which is the evaluation metrics for this competition. From the result we found that the R-square score is negative and the RMSE score is big. To explain this result, it is possible that R-square can be negative. Since R-square is defined as the proportion of variance explained by the fit, it is likely to be negative if the chosen model fits worse than a horizontal line. R-square is negative only when the chosen model does not follow the trend of the data, so fits worse than a horizontal line.

Before using other models, the Boston Housing neural network model is used in my data (from the deep learning homework) and it shows that the model fits pretty good and the R-square score is about 0.932.

5.2 Generalized Linear Models

5.2.1 Ridge Regression & Lasso Regression

There are many similarities between ridge regression and lasso regression ("A Complete Tutorial on Ridge and Lasso Regression in Python"). Both of them shrink the regression coefficients so that variables with minor contribution to the outcome, have their coefficients close to zero. This work is done by penalizing the magnitude of variables' coefficients along with minimizing the error between predicted and actual observations. These are called 'regularization' techniques.

The main difference between lasso and ridge is how they assign penalty to the coefficients. For ridge regression, it performs L2 regularization, which means it adds penalty of sum of squares of coefficients in the optimization objective. The minimization objective can be described as equation below:

$$\text{Minimization Objective} = \text{LS Obj} + \alpha^*$$

In the equation, "LS Obj" refers to least squares objective and α* is sum of sum of square of coefficients.

For lasso regression, it performs L1 regularization, which means it adds penalty equivalent to absolute value of the magnitude of coefficients. The minimization objective equation is the same with ridge:

Minimization objective = LS Obj + α*

The difference here is the meaning of α*. In this equation, it is sum of absolute value of coefficients.

For ridge and lasso, the main tuning parameter is alpha - a regularization parameter that measures how flexible the model is. The higher the regularization the less prone our model will be to overfit. However it will also lose flexibility and might not capture all of the signal in the data. During the model tuning process, we set a list of alpha in ridge and then made plots of calculated RMSE vs alpha. The results are shown in figure 10. From this curve we notice that when alpha is too large, the regularization is too strong and the model cannot capture all the complexities in the data. If however we let the model be too flexible (alpha small) the model begins to overfit. A value of alpha = 10 is about right based on the this curve. The lowest RMSE is 0.11122696043712099. In lasso model, we do a slightly different approach here and use the built in LassoCV to figure out the best alpha. The lowest value of RMSE for lasso is 0.1093424704465464. By comparison the results, we found that the lasso model performs better.

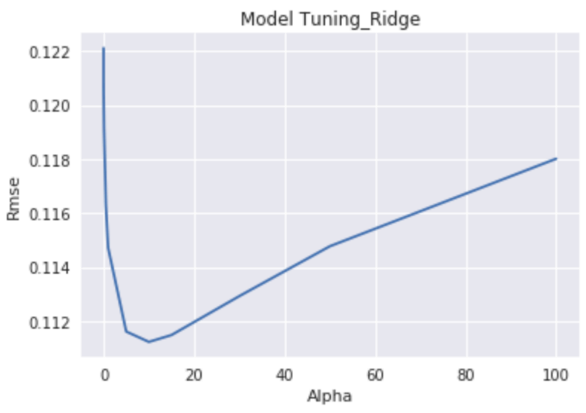


Figure 10. Relationship between RMSE and Alpha in Ridge Model

5.2.2 Elastic Net

Elastic Net is a linear regression model that is penalized with both the L1 regularization and L2 regularization. This combination allows for shrinking coefficients like ridge and setting some coefficients to zero like lasso.

Same with ridge and lasso, the key tuning parameter in elastic net is alpha. We use the same method and code with ridge and finally get the lowest RMSE value of 0.1095575263337879.

5.3 Decision Trees

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches and each represents values for the attribute tested. Leaf node represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data ("Regression with Decision Trees"). The core algorithm for building decision trees is called ID3, which employs a top-down, greedy search through the space of possible branches with no backtracking.

In this model, grid search (an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid) is used for parameter tuning. The RMSE value is 0.23825511743124933, which is higher than other models we developed before.

5.4 Ensemble methods

5.4.1 Random Forests

Random forests is an ensemble learning method that combines feature selection and decision trees. It develops lots of decision tree based on random selection of data and random selection of variables and many such random trees lead to a random forest. When splitting a node during construction of the tree, the split picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but its variance also decreases (because of averaging), usually more than compensating for the increase in bias, hence yielding an overall better model ("Ensemble methods").

In this model, we use grid search for parameter tuning and finally get the score 0.13428445796178295. It can be seen from this result that random forests has better performance than decision trees.

5.4.2 Gradient Tree Boosting

Gradient tree boosting involves three elements: loss function (based on type of problem being solved), weak learner (decision trees), additive model. Trees are added one at a time, and existing trees in the model are not changed. A gradient descent procedure is used to minimize the loss. To perform this procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient). We do this by parameterizing the tree, then modify parameters of the tree and move in the right direction by reducing residual. The output for the new tree is added to the output of the existing trees in order to correct or improve the final output of the model. A fixed number of trees are added or training stops when the loss reaches an acceptable level or no longer improves on an external validation dataset ("A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning").

For the Gradient Boosting algorithm, “huber” is used as the loss function since it is robust to outliers. In this model, I failed to use grid search for parameter tuning since it is extremely time consuming. As a result, other parameters are chosen based on past experience other kernels tackling this challenge, followed by trial and error to refine them to this specific dataset. The RMSE is 0.11390870391929091.

5.5 XGBoost

XGBoost is short for eXtreme gradient boosting. It is a library designed and optimized for boosted tree algorithms. It is highly efficient, flexible and portable, which implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that can solve data science problems in a accurate and fast way (“XGBoost Documentation”).

With the same reason as Gradient Tree Boosting, for this model some important parameters are set and we got the RMSE 0.11156401339952708.

5.6 LightGBM

Light GBM is a distributed, fast, high-performance gradient boosting framework based on decision tree algorithm It is used for classification, ranking, and many other machine learning tasks. As Light GBM is based on decision trees, instead of spliting the tree depth wise or level wise, it splits the tree leaf wise with the best fit. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than level-wise algorithm. Therefore, it has much better accuracy, which can rarely be achieved by any of the existing boosting algorithms. At the same time, it is very fast, hence the word “Light” (“Which algorithm takes the crown: Light GBM vs XGBOOST ?”).

Still, for this model, key parameters are set and we got the RMSE 0.11479546886018226.

5.7 Simple Stacking Model Construction

5.7.1 Comparison of Base Model Performance

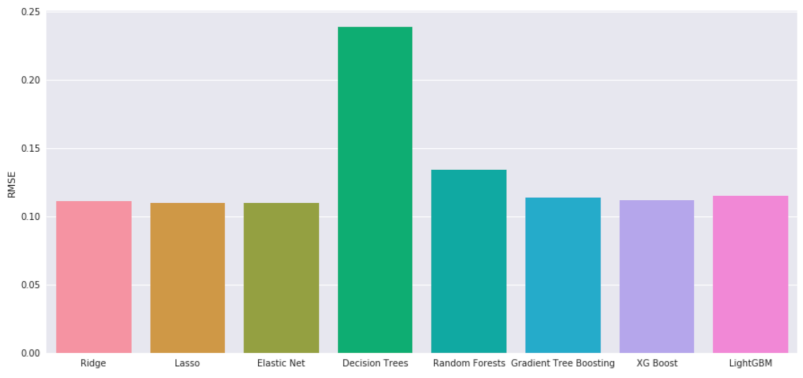
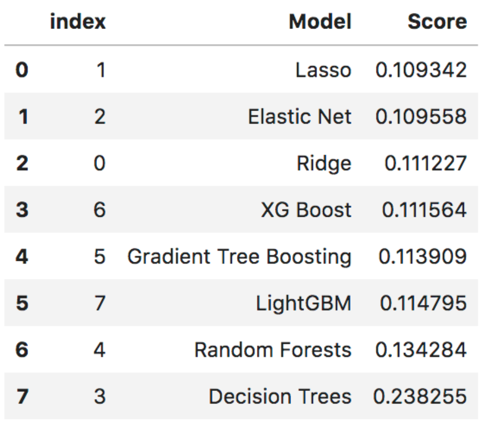


Figure 11. RMSE for the Eight Model

From the bar plot (Figure 11) it is obvious that the decision trees has the worst performance compared with other models, which is followed by random forests. Other 6 models have pretty similar performance for our dataset. From Figure 12 we can see Lasso performs the best among these 8 models with the lowest RMSE 0.109342. The next step is to build a stacking model by combine these 8 base models.



	index	Model	Score
0	1	Lasso	0.109342
1	2	Elastic Net	0.109558
2	0	Ridge	0.111227
3	6	XG Boost	0.111564
4	5	Gradient Tree Boosting	0.113909
5	7	LightGBM	0.114795
6	4	Random Forests	0.134284
7	3	Decision Trees	0.238255

Figure 12. RMSE Table for the Eight Model (Ranked from RMSE Low to High)

5.7.2 Stacking Model Construction and Final Results

Model stacking is an efficient ensemble method in which the predictions generated by using various machine learning algorithms, are used as inputs in a second-layer learning algorithm. Then the second-layer is trained to optimally combine the model predictions to form a new set of predictions. These can be used on additional layers, or the process can stop here with a final result (Figure 13). This method is commonly used to boost predictive accuracy ("Why do stacked ensemble models win data science competitions?"). In other words, ensemble stacking can be referred to as blending since all the numbers are blended to produce a prediction or classification.

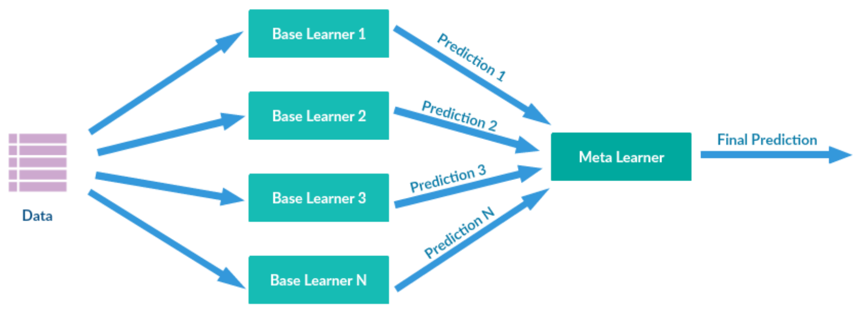


Figure 13. Stacking visualized (Image from http://supunsetunga.blogspot.com/)

The simplest stacking method is the unweighted average of the prediction of the base models, in which the sum of the predicted values of all base models are divided by number of base models. In an unweighted average, each model takes the same weight when an ensemble model is built. There is another method using weighted averages of the prediction of the base models. In my modeling process, I used the simple stacking method (using unweighted averages) to make final predictions.

My first trial is to use all the eight models as base model to build the stacking model by taking average of the prediction results：

stacked = (ridge_pred + lasso_pred + elasticnet_pred + rf_pred + gtb_pred + xgb_pred + lgb_pred + dtree_pred) / 8

In the first submission on Kaggle, the score is 0.14440. Then we dropped the decision trees since it has very bad performance for the training dataset. We use the same stacking method to get the final prediction and then the second submission gave us the core of 0.14287, which is better than the first submission. Then we keep dropping the random forests (as it did not perform well for training dataset) and used the left 6 base model to build the stacking model. This time the score on Kaggle became to 0.15035, which is worse than the second submission. The summary of these three submission is shown in table 4. Finally we choose the stacking model used in submission two as our final model, which are: Ridge, Lasso, Elastic Net, Random Forests, Gradient Tree Boosting, XGBoost, LightGBM. The final stacked model is described as:

stacked = (ridge_pred + lasso_pred + elasticnet_pred + rf_pred + gtb_pred + xgb_pred + lgb_pred) / 7

Submission	Base Models	Score
1	All the Eight Models	0.14440
2	Dropped Decision Trees	0.14287
3	Dropped Decision Trees & Random Forests	0.15035

Table 4. Submission Results on Kaggle

5.8 Summary

In the modeling part, there are 10 models in total have been used for the training dataset: Naïve Model, Neural Network, Ridge, Lasso, Elastic Net, Random Forests, Gradient Tree Boosting, XGBoost, LightGBM, Decision Trees. Overall the Naïve Model performs very bad , followed by Decision Trees and Random Forests. There is much possibility for Neural Network to get a great performance. The other 7 models have pretty same level of performance. Among all these models, Lasso is the best model for the training dataset and has the lowest RMSE. From the process of building the stacking model we can find that just by adding more models to the stacking algorithm, does not mean we will get a better predictor. There are no free lunches in machine learning. From several trails we find that the stacking model with 7 models (listed above) has the best performance for the testing dataset. Since there is a relatively large difference between performance on training and testing data set, it is possible that our model may have the problem of overfitting. Using grid search for parameter tuning may be one factor that can lead to overfitting. Therefore, in order to get a higher performance, there is still much work to do.

6. Conclusion

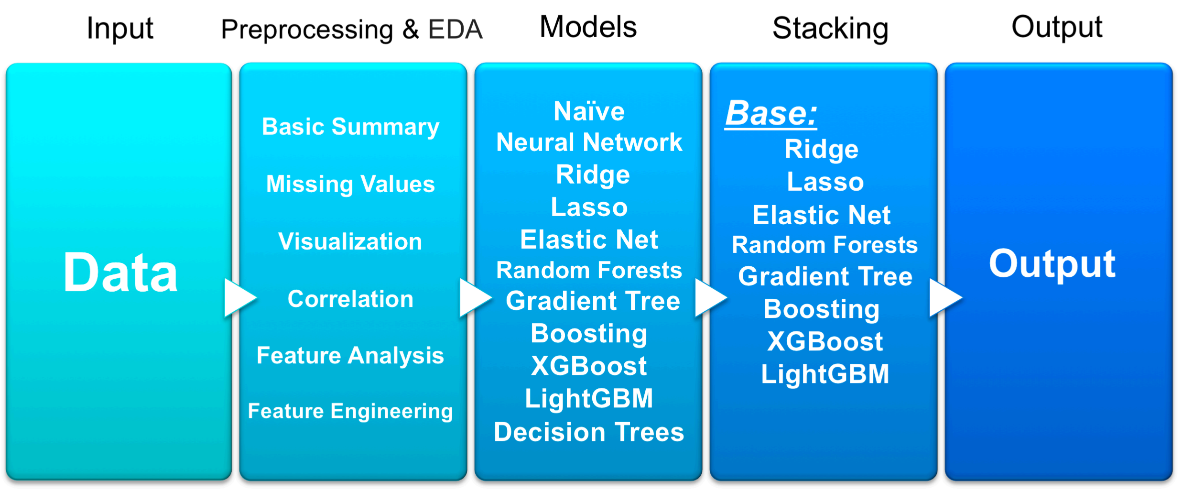


Figure 14. Flowchart of the Whole Process

The whole process of what we have done for this Kaggle competition task is shown in Figure 14. I started with preprocessing and exploratory data analysis, where steps such as basic summary, data cleaning, visualization and correlation analysis, feature analysis and engineering are included in this section and data are well prepared for modeling. Then I tried many different algorithms and used the RMSE as the main evaluation for model performance. Finally, some important base models are chosen based on their individual performance on the training data set and then are combined into the stacking model and get the final prediction results.

The performance of the final stacking model I constructed is not satisfying. However, there is still much room for improvement. This project opens several avenues for future work. Since data processing is a very important part before modeling process, more advanced technologies can be tried in this section such as in detecting outliers, feature selection. As for modeling section, it is necessary to try other methods for parameter tuning. We also need to check if there are any problems of overfitting. Different algorithms are also good choice for improving the performance and we may also try more complicated stacking model to improve the performance of our model.

Works Cited

"A Complete Tutorial on Ridge and Lasso Regression in Python." *Analytics Vidhya*. Analytics Vidhya, 28 Jan. 2016. Web. 8 Dec. 2018. < https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression python/>

"Regression with Decision Trees." *YouTube.* YouTube, 28 Aug. 2014. Web. 8 Dec. 2018. <https://www.youtube.com/watch?v=nSaOuPCNvlk>

"Ensemble methods." *scikit-learn.* scikit-learn developers, 2007. Web. 8 Dec. 2018. <https://scikit-learn.org/stable/modules/ensemble.html#random-forests>

"A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning." *Machine Learning Mastery*. Machine Learning Mastery, 09 Sep. 2016. Web. 8 Dec. 2018. <https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/>

"XGBoost Documentation." *XGBoost.* xgboost developers, 2016. Web. 8 Dec. 2018. < https://xgboost.readthedocs.io/en/latest/#>

"Which algorithm takes the crown: Light GBM vs XGBOOST ?" *Analytics Vidhya.* Analytics Vidhya, 12 Jun. 2017. Web. 8 Dec. 2018. <https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/>

“Why do stacked ensemble models win data science competitions?” *sas.* SAS Institute Inc., 18 May. 2017. Web. 8 Dec. 2018. <https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/>