

**House prices: advanced regression techniques kaggle competition project – spring 2018**

Orange Analytics – Final Project Report – Team 11



**Team Members**

Fregenet Andya (A11516880)

Manideep Mellachervu (A20036160)

Trevor Behm (A10159191)

Ravi Josyula (A20090983)

**Table of Contents**

[**Introduction and Project Background** 2](#_Toc7200003)

[**Project Objectives & Scope** 3](#_Toc7200004)

[**Orange Analytics – Final Team Score and Ranking** 4](#_Toc7200005)

[**Project Timeline** 5](#_Toc7200006)

[**Data Exploration, Cleaning and Transformation** 6](#_Toc7200007)

[**Feature Engineering and Variable Selection** 13](#_Toc7200008)

[**Modelling and Methodology** 15](#_Toc7200009)

[**Appendix (Code and Scored data)** 20](#_Toc7200010)

# **Introduction and Project Background**

People dream about owning the ideal house that satisfies their desired needs while ensuring that they adhere to their budget constraints. Often times, finding the perfect balance between the two is not an easy task, and it demands a considerable amount of searching as well as compromises. This project aims at exploring the use of advanced regression and other analytical techniques, and feature engineering to predict the final price of residential homes in Ames, Iowa. With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges you to predict the final price of each home.

The prediction of the home prices is done as part of the fulfillment of competition requirement on Kaggle. The datasets used for the purpose of this project / competition were provided by Kaggle after registration for the competition. Although, initial data exploration, data assay, variable selections and preliminary model testing were conducted on SAS Enterprise Miner, SAS Enterprise Guide, and Python - feature engineering and various modeling techniques used for predictions and scoring were performed using Python. The final submissions made to the competition and ranking awarded to the team were based on the models, predictions and scoring submissions performed on Python.

Submissions are evaluated on [Root-Mean-Squared-Error (RMSE)](https://en.wikipedia.org/wiki/Root-mean-square_deviation) between the logarithm of the predicted value and the logarithm of the observed sales price.

# **Project Objectives & Scope**

Based on initial project meetings, the objectives and the scope of the project was narrowed down to the following deliverables:

1. Predict sales prices of houses using available data
2. Feature Engineering: to create new and useful features from existing variables
3. Build a host of high-performance models to see which gives us the closest predictions with the best model performance
4. Place as high as possible in the competition

# **Orange Analytics – Final Team Score and Ranking**

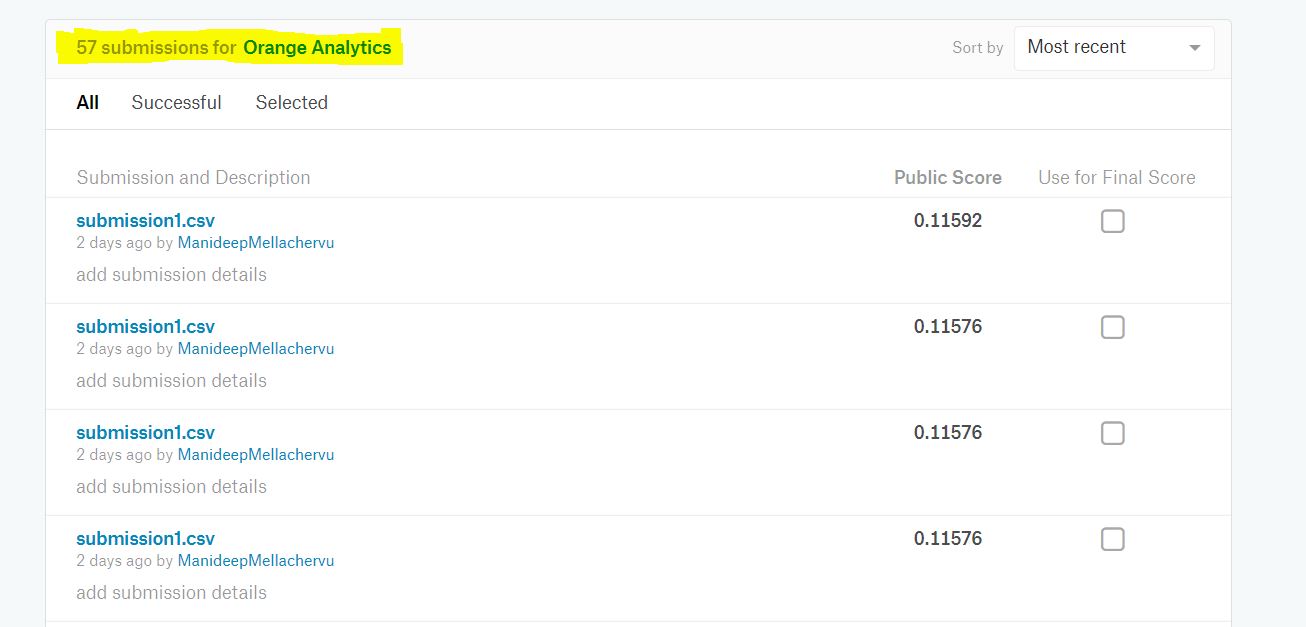
Team 11- Orange Analytics’ final rank is **582** out of the 4377 teams:





Team 11- Orange Analytics’ final/best score is **0.11576**. The closer the score is to zero, the better (more accurate) the predictions:

There were a total of 57 scoring submissions made by the team:



# **Project Timeline**

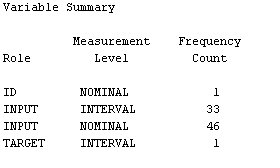
The timeline of the deliverables for this project is as follows:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Deliverable** | 02/18 | 02/25 | 03/04 | 03/11 | 03/18 | 04/01 | 04/08 | 04/15 | 04/22 | 04/29 |
| **Descriptive Analytics** |  |  |  |  |  |  |  |  |  |  |
| **Briefing 1** |  |  |  |  |  |  |  |  |  |  |
| **Data Assay** |  |  |  |  |  |  |  |  |  |  |
| **Feature Creation** |  |  |  |  |  |  |  |  |  |  |
| **Briefing 2** |  |  |  |  |  |  |  |  |  |  |
| **Data Preparation** |  |  |  |  |  |  |  |  |  |  |
| **Briefing 3** |  |  |  |  |  |  |  |  |  |  |
| **Model Building** |  |  |  |  |  |  |  |  |  |  |
| **Briefing 4** |  |  |  |  |  |  |  |  |  |  |
| **Draft Final Report** |  |  |  |  |  |  |  |  |  |  |
| **Briefing 5** |  |  |  |  |  |  |  |  |  |  |
| **Code/Report Submission** |  |  |  |  |  |  |  |  |  |  |

# **Data Exploration, Cleaning and Transformation**

Two data sets are available for the project. The first dataset is training dataset, which contains 1460 observations representing houses and 79 exploratory variables that describes most features of the homes along with the price of each house. The second dataset - the test dataset, contains 1459 observation with the same 79 house features.

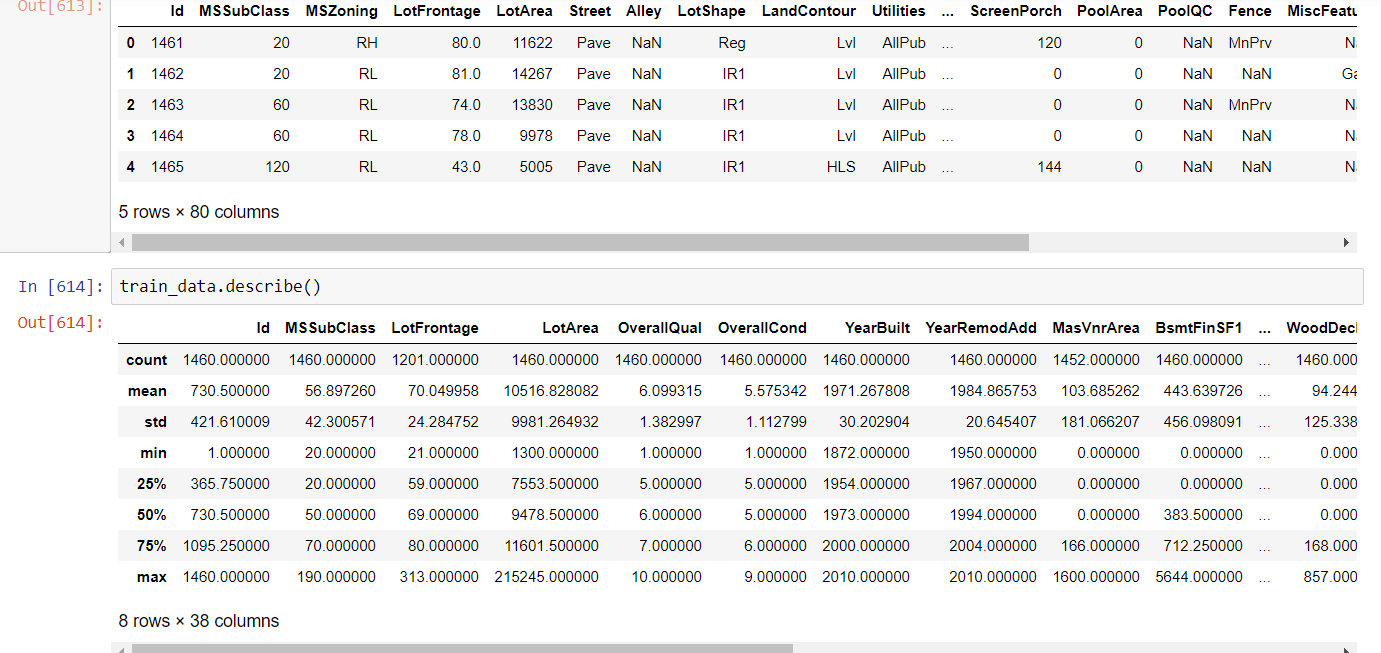
From the 79 variables, 33 are continuous variables & 46 are categorical variables. The target variable ‘Sales Price’ is a continuous variable.



Preliminary data exploration of the training dataset reveled value NA is used in multiple variables. However, the representation is not consistent across the data set. For three continues variables, it represents missing values. One of the continuous variables with value NA for missing values is MasVnrArea (Masonry veneer type). NA was inferred as missing because value zero is placed to indicate cases where there is no masonry veneer. Variable LotFrontage, has 259 observations with NA values. Given lot area data is available for these observations, these values are considered as missing values rather than observation with no lot frontage at all. Similarly, NA values for variable GarageYrBlt (Garage year built) is considered as missing value.

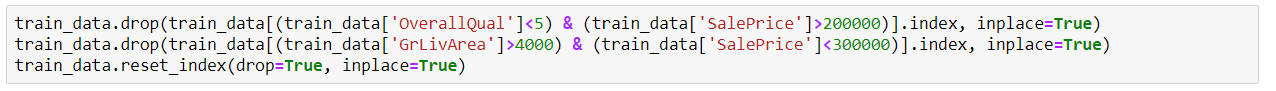
When it comes to categorical variables, NA represents missing for some variables such as MasVnrType (Masonry veneer type). However, for other variables such as GarageType, NA represents absence of the feature from the house. Therefore, a careful look of each variables is warranted.

**Description of the dataset – basic descriptive statistics**:

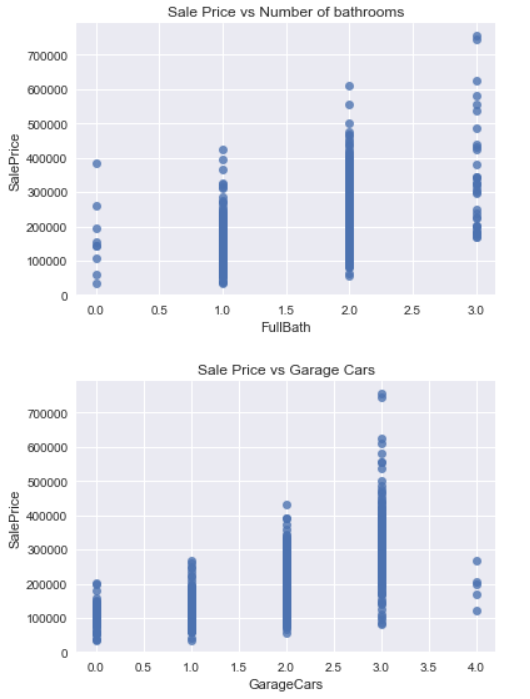
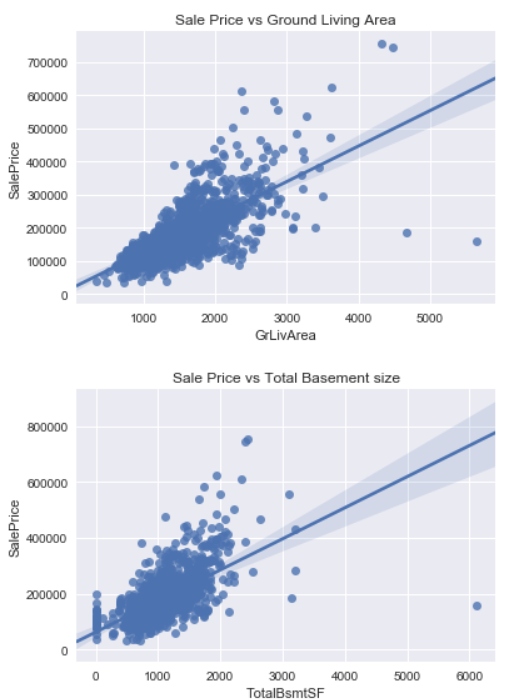


**Outlier Detection and Removal:**

If the overall quality was less than 5 and the Sale Price was greater than $200,000, or if the Living Area was greater than 4000 square feet or the Sale Price was lesser than $300,000 the values were dropped from the dataset since they were considered as outliers by the team.

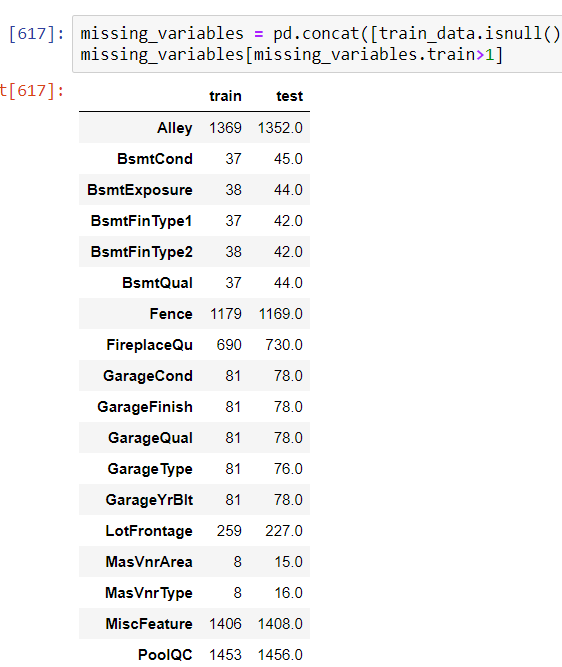


Outlier detection and removal was done by plotting the variable Sales Price was against various variables such as GrLivArea, Total Basement Size, Number of Bathrooms, and Garage Cars as can be seen from the outputs below:



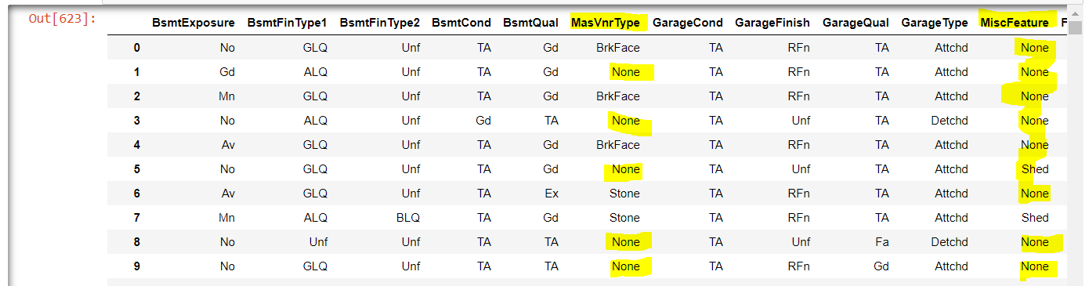
After the outliers were removed from the data, we started checking for missing values so that appropriate imputation techniques could be implemented.

**Checking for missing values:**

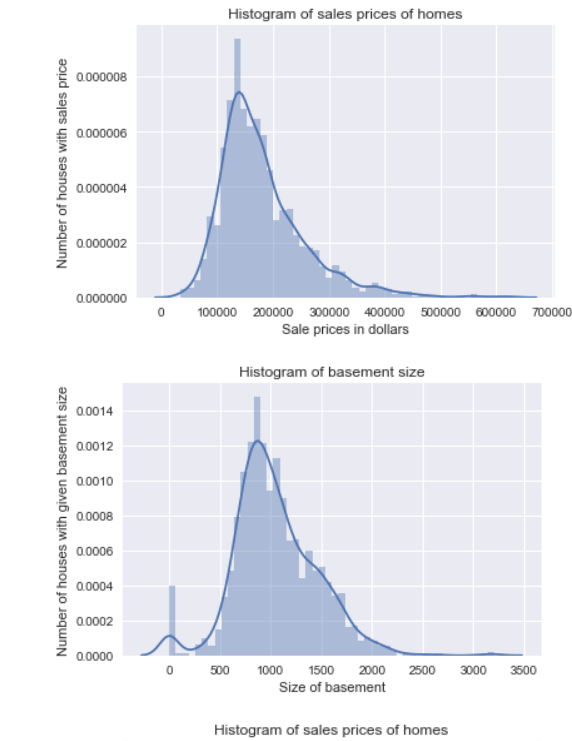


Upon closer observation of the dataset we can notice that the values are missing at random from the dataset. This is because of the way the survey data was collected in this case.

Next, we imputed the columns with ‘None’ / NA’s wherever appropriate on a case by case basis. A screenshot showing the same is attached below:

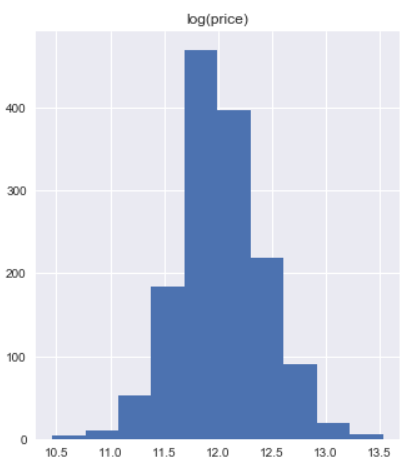
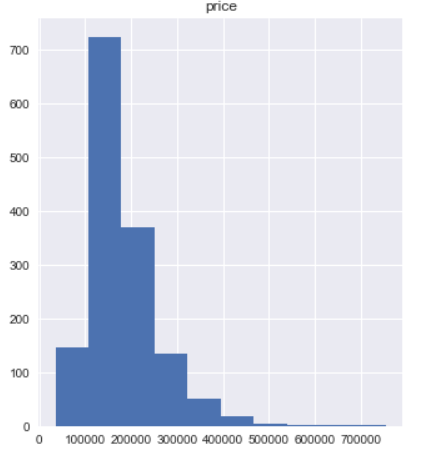


We also checked distributions across different variables. An example of the histograms of Sales Price and Size of Basement is shown in the figures below:



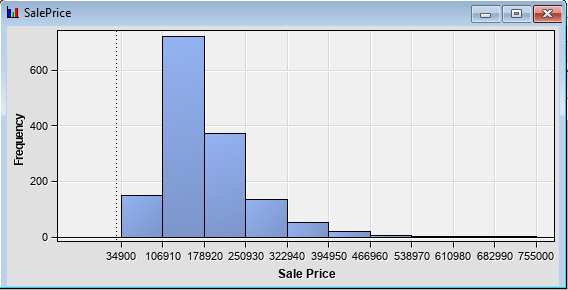
**Variable Transformations:**

After checking for distributions of the variables, the team handled skewness and kurtosis of variables. An example of how Sales Price was handled can be seen from the outputs below:

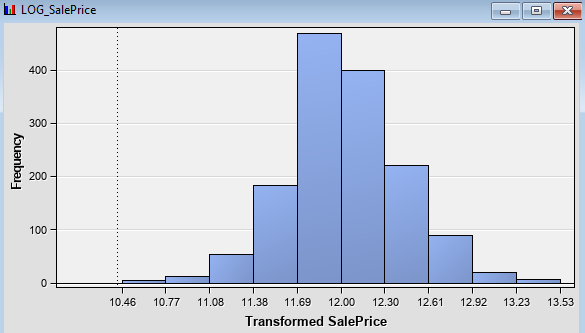


*Original Variable* *Log Transformed Variable*

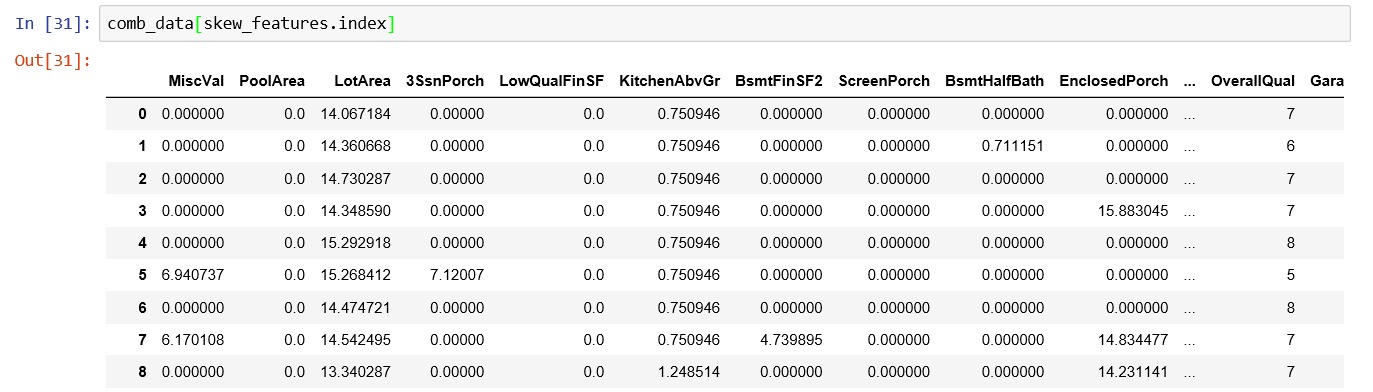
The distribution of target variable sales price is right skewed. With skewness value of 1.88 as can be seen from the figure below:



After log transformation, the skewness value dropped to 0.12 and the distribution of the sales price normalized as shown in the figure below:



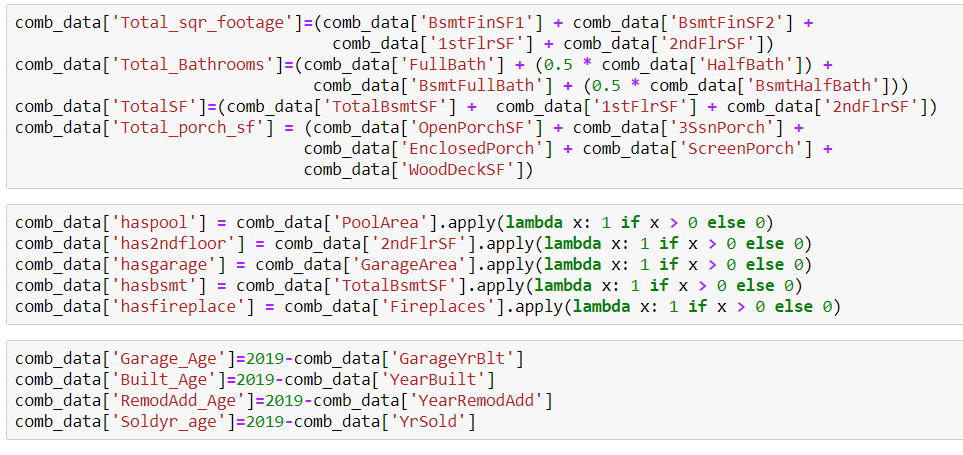
We used box-cos transformation for the variables in the dataset. The transformed values of the variables can be seen below:



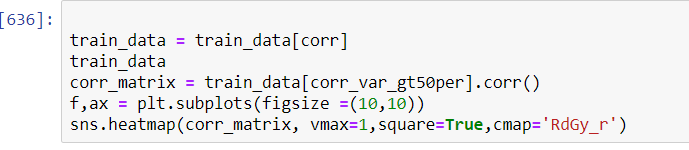
# **Feature Engineering and Variable Selection**

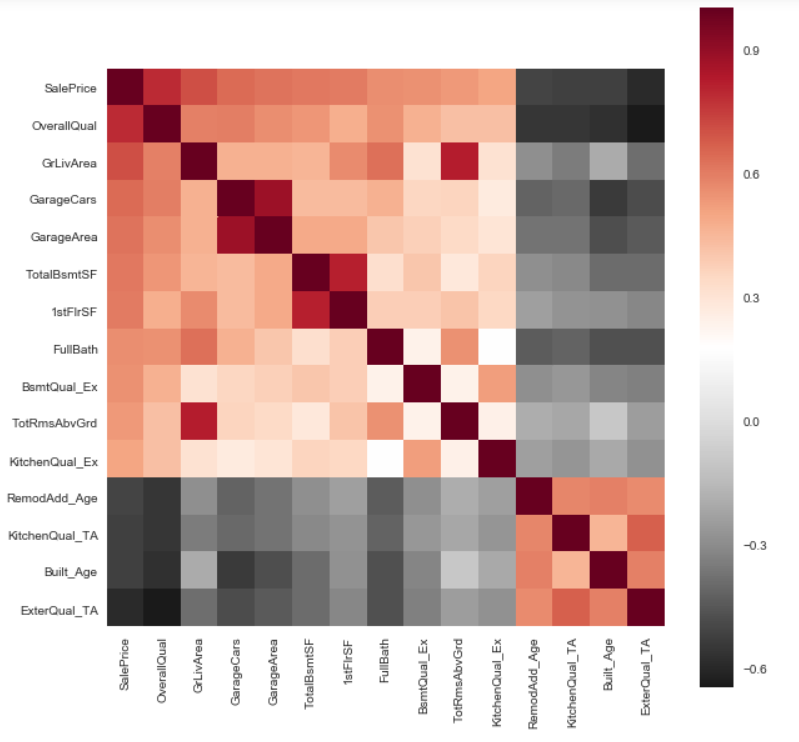
We used SAS and python to create various new features / variables from the existing cleaned, imputed and transformed variables, as described below:

* Square feet – Combined these into a total sqft value (totalsf) for the homes. Including unfinished basement area had a higher result in our model, but also tested only using finished area.
* Created a binary variable (unfinbase) to include with totalsf to denote unfinished square footage (unfinished area being understood not to be included in loan value of home).
* Ext1 and Ext 2 kept but binned all variables with less than 50 observations into an ‘other’ category. (Renamed siding1 and siding2)
* Other masonry information was removed due to less impact than and correlation to siding variables.
* Fireplace quality kept – other variables including number of fireplaces removed due decreasing predictive value of the variables above.
* Combined bathrooms variable performed better than the one separated for above ground.
* Summation variable created for total porch area.
* Created a variable (recentupdate) to combine minimum values of time since built or remodeled.
* Combined month and year variables to create a date variable.
* Created binary variable (postcrash) to signify sales occurring after the housing crash.



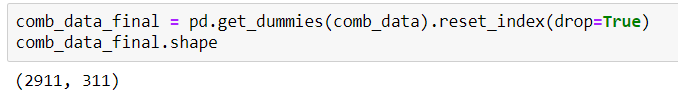
Next, we checked variable correlation using a heat map:





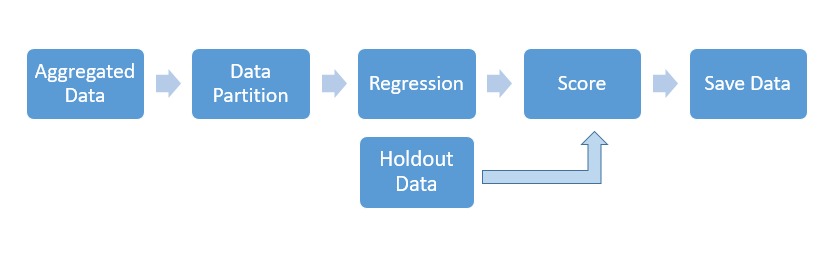
We can see from the above heat map that variables such as Living area & total # of rooms, garage cars & garage area, basement area & first floor area, total # of rooms & living area, Overall quality are highly correlated and important variables in the model with respect to the target variable Sales Price.

**Dummy Variable Creation:**



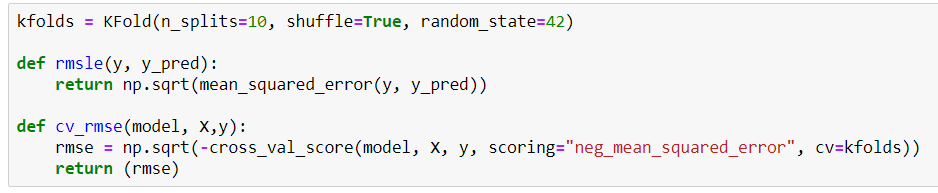
After creating dummy variables, we have 2911 rows and 311 columns in the dataset. Categorical variables with different levels were broken down into individual binary variables.

# **Modelling and Methodology**



**Splitting datasets into training and validation:**

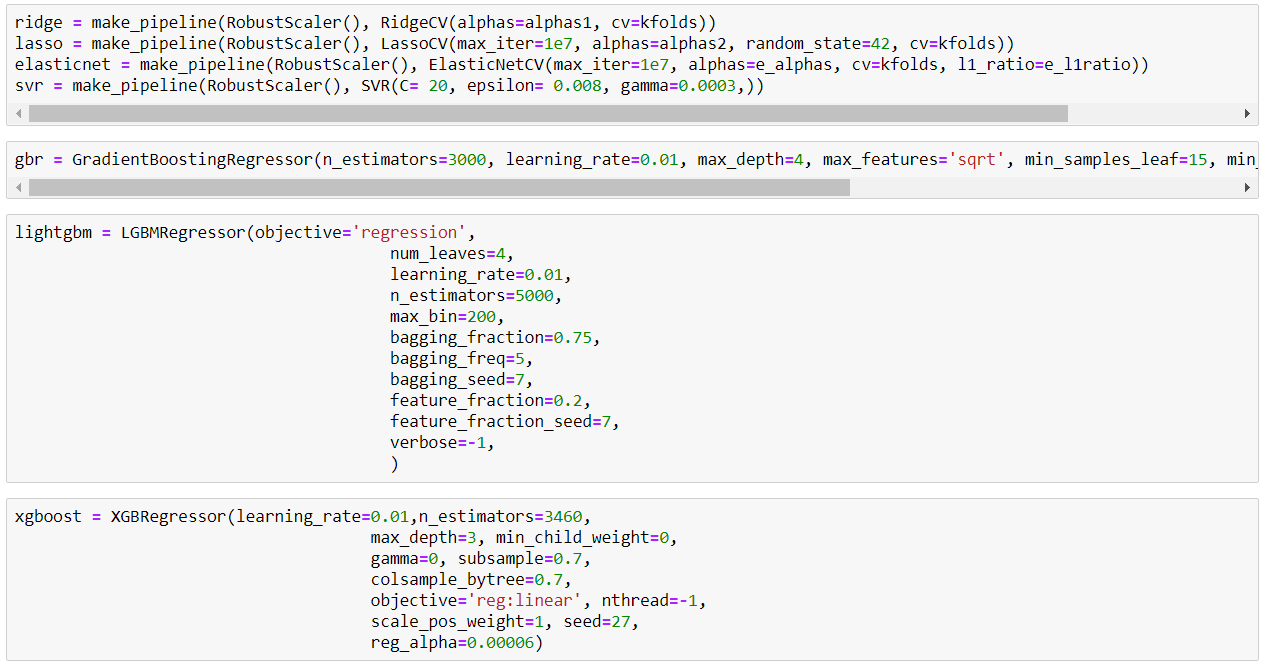
First, the final dataset with all the features (post cleaning, imputation, and transformations) was split into 10 smaller datasets for K-fold cross validation of the models. 9 datasets were used for training the models and the remaining 1 dataset was used for validation. This process is repeated until each sample was used as validation dataset. Hence, 10 models are built using single machine learning algorithm and average of rmse of all models built using 10 datasets is used as the final rmse for that model. This process of cross validation is used for all the regression techniques as shown in following pages.



**Modelling:**

The following analytical models were built and used in this project:

1. Ridge Regression
2. LASSO Regression
3. Elastic Net
4. Support Vector Regression
5. Gradient Boosting
6. Light Gradient Boosting
7. XG Boosting



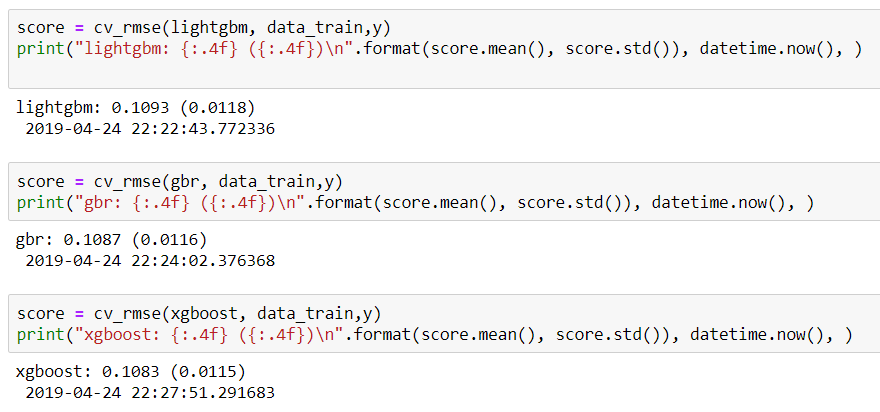
We built different models with Sales Price as the target and the 311 variables as described above as inputs, with the models themselves selecting the best variables for inputs.

**Calculating RMSE:**

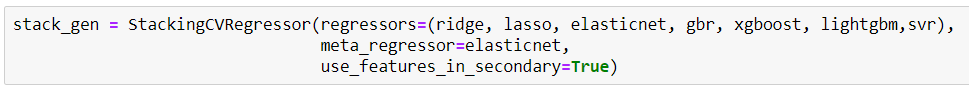
We can observe that the Root Mean Square Error is the least for LASSO and Elastic Net with values of 0.1029.

This means that these two models perform the best in terms of having the least error in predicting the Sales Price values versus the actual Sales Price Values.

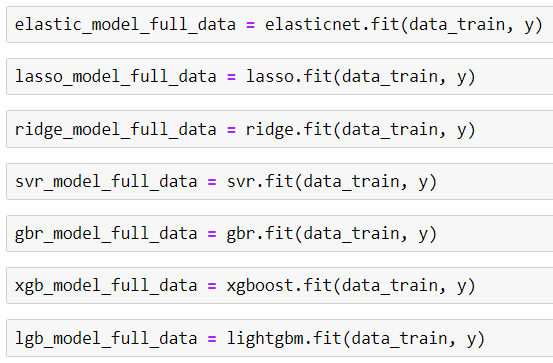




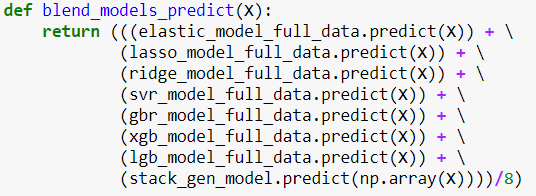
Next, we have combines all the seven models to create a new ‘Stacking Regressor’ model:



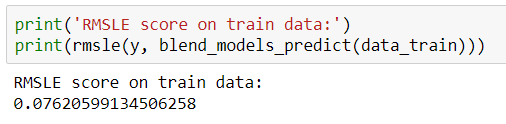
Next, we fit each and every individual model with the input variables in data\_train (training dataset) and the target y = Sales Price:



After this, we created a new function which calculates the average of all predictions (Predicted Values) from all the above models:



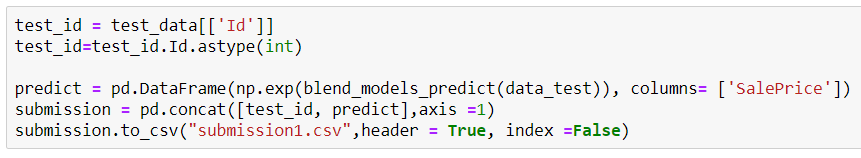
RMSE score of trained data came out as 0.076 as can be seen from the output below:



The following variables came out as the top most important variables from the gradient boosting regressor model.



Then, we have used the test data given by Kaggle and used the function ‘blend\_ models\_predict’ (shown above) to calculate the average of predictions from all the models.



After running the above function, we get a dataset with predicted Sales Price values. This scored dataset was then uploaded to Kaggle and we received a score of **0.11576**.





# **Appendix (Code and Scored data)**

**Python Code**

import os

import numpy as np

import pandas as pd

import matplotlib

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

from scipy.stats import skew

from scipy.special import boxcox1p

from scipy.stats import boxcox\_normmax

from datetime import datetime

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import ElasticNetCV, LassoCV, RidgeCV

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.svm import SVR

from sklearn.pipeline import make\_pipeline

from sklearn.preprocessing import RobustScaler

from sklearn.model\_selection import KFold, cross\_val\_score

from sklearn.metrics import mean\_squared\_error

from mlxtend.regressor import StackingCVRegressor

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

from sklearn import linear\_model, svm

from sklearn.ensemble import GradientBoostingRegressor as xgb

from sklearn.preprocessing import Imputer

os.chdir('E:\Sem4\Kaggle Project\house-prices-advanced-regression-techniques')

train\_data=pd.read\_csv('train.csv')

test\_data=pd.read\_csv('test.csv')

plt.figure(1)

sns.set(color\_codes=True)

ax = sns.regplot(x="GrLivArea", y="SalePrice", data=train\_data)

plt.title('Sale Price vs Ground Living Area')

plt.figure(2)

sns.set(color\_codes=True)

ax = sns.regplot(x="TotalBsmtSF", y="SalePrice", data=train\_data)

plt.title('Sale Price vs Total Basement size')

plt.figure(3)

sns.set(color\_codes=True)

ax = sns.regplot(x="FullBath", y="SalePrice", data=train\_data, fit\_reg = False)

plt.title('Sale Price vs Number of bathrooms')

plt.figure(4)

sns.set(color\_codes=True)

ax = sns.regplot(x="GarageCars", y="SalePrice", data=train\_data, fit\_reg = False)

plt.title('Sale Price vs Garage Cars')

plt.figure(5)

sns.set(color\_codes=True)

ax = sns.regplot(x="OverallQual", y="SalePrice", data=train\_data, fit\_reg = False)

plt.title('Sale Price vs OverallQual')

train\_data.sort\_values(by = 'SalePrice', ascending = False)[:10]['SalePrice']

train\_data.sort\_values(by = 'GrLivArea', ascending = False)[:10]['GrLivArea']

train\_data.sort\_values(by = 'TotalBsmtSF', ascending = False)[:10]['TotalBsmtSF']

outliers = [30, 88, 462, 631, 1322]

train\_data = train\_data.drop(train\_data.index[outliers])

# y = y.drop(y.index[outliers])

train\_data.drop(train\_data[(train\_data['OverallQual']<5) & (train\_data['SalePrice']>200000)].index, inplace=True)

train\_data.drop(train\_data[(train\_data['GrLivArea']>4000) & (train\_data['SalePrice']<300000)].index, inplace=True)

train\_data.reset\_index(drop=True, inplace=True)

comb\_data = pd.concat((train\_data.loc[:,'MSSubClass':'SaleCondition'],

test\_data.loc[:,'MSSubClass':'SaleCondition']))

num\_var = [f for f in train\_data.columns if train\_data.dtypes[f] != 'object']

num\_var.remove('SalePrice')

num\_var.remove('Id')

cat\_var = [f for f in train\_data.columns if train\_data.dtypes[f] == 'object']

matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)

prices = pd.DataFrame({"price":train\_data["SalePrice"], "log(price)":np.log1p(train\_data["SalePrice"])})

prices.hist()

train\_data["SalePrice"] = np.log1p(train\_data["SalePrice"])

missing\_variables = pd.DataFrame(comb\_data.isnull().sum())

missing\_variables = missing\_variables[missing\_variables[0]>0]

missing\_variables

for col in ('BsmtExposure','BsmtFinType1','BsmtFinType2','BsmtCond',

'BsmtQual','MasVnrType', 'GarageCond','GarageFinish','GarageQual',

'GarageType','MiscFeature','FireplaceQu','Fence','PoolQC','Alley'):

comb\_data[col]= comb\_data[col].fillna('None')

for col in ('Utilities','Functional','Exterior1st','Exterior2nd','MSZoning','KitchenQual','SaleType','Electrical'):

comb\_data[col]= comb\_data[col].fillna(train\_data[col].mode()[0])

miss\_num\_var = [f for f in missing\_variables.index.tolist() if f in num\_var]

miss\_cat\_var = [f for f in missing\_variables.index.tolist() if f in cat\_var]

miss\_num\_var

for col in miss\_num\_var:

comb\_data[col]= comb\_data[col].fillna(0)

skew\_features = train\_data[num\_var].apply(lambda x: skew(x)).sort\_values(ascending=False)

high\_skew = skew\_features[skew\_features > 0.75]

skew\_index = high\_skew.index

for i in skew\_index:

comb\_data[i] = boxcox1p(comb\_data[i], boxcox\_normmax(comb\_data[i] + 1))

comb\_data['Total\_sqr\_footage']=(comb\_data['BsmtFinSF1'] + comb\_data['BsmtFinSF2'] +

comb\_data['1stFlrSF'] + comb\_data['2ndFlrSF'])

comb\_data['Total\_Bathrooms']=(comb\_data['FullBath'] + (0.5 \* comb\_data['HalfBath']) +

comb\_data['BsmtFullBath'] + (0.5 \* comb\_data['BsmtHalfBath']))

comb\_data['TotalSF']=(comb\_data['TotalBsmtSF'] + comb\_data['1stFlrSF'] + comb\_data['2ndFlrSF'])

comb\_data['Total\_porch\_sf'] = (comb\_data['OpenPorchSF'] + comb\_data['3SsnPorch'] +

comb\_data['EnclosedPorch'] + comb\_data['ScreenPorch'] +

comb\_data['WoodDeckSF'])

comb\_data['haspool'] = comb\_data['PoolArea'].apply(lambda x: 1 if x > 0 else 0)

comb\_data['has2ndfloor'] = comb\_data['2ndFlrSF'].apply(lambda x: 1 if x > 0 else 0)

comb\_data['hasgarage'] = comb\_data['GarageArea'].apply(lambda x: 1 if x > 0 else 0)

comb\_data['hasbsmt'] = comb\_data['TotalBsmtSF'].apply(lambda x: 1 if x > 0 else 0)

comb\_data['hasfireplace'] = comb\_data['Fireplaces'].apply(lambda x: 1 if x > 0 else 0)

comb\_data['Garage\_Age']=2019-comb\_data['GarageYrBlt']

comb\_data['Built\_Age']=2019-comb\_data['YearBuilt']

comb\_data['RemodAdd\_Age']=2019-comb\_data['YearRemodAdd']

comb\_data['Soldyr\_age']=2019-comb\_data['YrSold']

comb\_data['PostCrash']=(comb\_data.YrSold>2008)

comb\_data=comb\_data.drop(columns=['GarageYrBlt','YearBuilt','YearRemodAdd','YrSold'])

comb\_data['PostCrash']=comb\_data['PostCrash'].astype('int')

comb\_data.shape

comb\_data\_final = pd.get\_dummies(comb\_data).reset\_index(drop=True)

comb\_data\_final.shape

overfit = []

for i in comb\_data\_final.columns:

counts = comb\_data\_final[i].value\_counts()

zeros = counts.iloc[0]

if zeros / len(comb\_data\_final) \* 100 > 99.94:

overfit.append(i)

overfit = list(overfit)

comb\_data\_final = comb\_data\_final.drop(overfit, axis=1)

# X\_sub = X\_sub.drop(overfit, axis=1)

Overfit

data\_train = comb\_data\_final[:train\_data.shape[0]]

data\_test = comb\_data\_final[train\_data.shape[0]:]

y = train\_data.SalePrice

kfolds = KFold(n\_splits=10, shuffle=True, random\_state=42)

def rmsle(y, y\_pred):

return np.sqrt(mean\_squared\_error(y, y\_pred))

def cv\_rmse(model, X,y):

rmse = np.sqrt(-cross\_val\_score(model, X, y, scoring="neg\_mean\_squared\_error", cv=kfolds))

return (rmse)

alphas1 = [14.5, 14.6, 14.7, 14.8, 14.9, 15, 15.1, 15.2, 15.3, 15.4, 15.5]

alphas2 = [5e-05, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008]

e\_alphas = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007]

e\_l1ratio = [0.8, 0.85, 0.9, 0.95, 0.99, 1]

ridge = make\_pipeline(RobustScaler(), RidgeCV(alphas=alphas1, cv=kfolds))

lasso = make\_pipeline(RobustScaler(), LassoCV(max\_iter=1e7, alphas=alphas2, random\_state=42, cv=kfolds))

elasticnet = make\_pipeline(RobustScaler(), ElasticNetCV(max\_iter=1e7, alphas=e\_alphas, cv=kfolds, l1\_ratio=e\_l1ratio))

svr = make\_pipeline(RobustScaler(), SVR(C= 20, epsilon= 0.008, gamma=0.0003,))

gbr = GradientBoostingRegressor(n\_estimators=3000, learning\_rate=0.01, max\_depth=4, max\_features='sqrt', min\_samples\_leaf=15, min\_samples\_split=10, loss='huber', random\_state =42)

lightgbm = LGBMRegressor(objective='regression',

num\_leaves=4,

learning\_rate=0.01,

n\_estimators=5000,

max\_bin=200,

bagging\_fraction=0.75,

bagging\_freq=5,

bagging\_seed=7,

feature\_fraction=0.2,

feature\_fraction\_seed=7,

verbose=-1,

)

xgboost = XGBRegressor(learning\_rate=0.01,n\_estimators=3460,

max\_depth=3, min\_child\_weight=0,

gamma=0, subsample=0.7,

colsample\_bytree=0.7,

objective='reg:linear', nthread=-1,

scale\_pos\_weight=1, seed=27,

reg\_alpha=0.00006) score = cv\_rmse(ridge , data\_train,y)

print("Ridge: {:.4f} ({:.4f})\n".format(score.mean(), score.std()), datetime.now(), )

score = cv\_rmse(lasso , data\_train,y)

print("LASSO: {:.4f} ({:.4f})\n".format(score.mean(), score.std()), datetime.now(), )

score = cv\_rmse(elasticnet, data\_train,y)

print("elastic net: {:.4f} ({:.4f})\n".format(score.mean(), score.std()), datetime.now(), )

score = cv\_rmse(svr, data\_train,y)

print("SVR: {:.4f} ({:.4f})\n".format(score.mean(), score.std()), datetime.now(), )

score = cv\_rmse(lightgbm, data\_train,y)

print("lightgbm: {:.4f} ({:.4f})\n".format(score.mean(), score.std()), datetime.now(), )

score = cv\_rmse(gbr, data\_train,y)

print("gbr: {:.4f} ({:.4f})\n".format(score.mean(), score.std()), datetime.now(), )

score = cv\_rmse(xgboost, data\_train,y)

print("xgboost: {:.4f} ({:.4f})\n".format(score.mean(), score.std()), datetime.now(), )

stack\_gen = StackingCVRegressor(regressors=(ridge, lasso, elasticnet, gbr, xgboost, lightgbm,svr),

meta\_regressor=elasticnet,

use\_features\_in\_secondary=True)

stack\_gen\_model = stack\_gen.fit(np.array(data\_train), np.array(y))

elastic\_model\_full\_data = elasticnet.fit(data\_train, y)

lasso\_model\_full\_data = lasso.fit(data\_train, y)

ridge\_model\_full\_data = ridge.fit(data\_train, y)

svr\_model\_full\_data = svr.fit(data\_train, y)

gbr\_model\_full\_data = gbr.fit(data\_train, y)

xgb\_model\_full\_data = xgboost.fit(data\_train, y)

lgb\_model\_full\_data = lightgbm.fit(data\_train, y)

def blend\_models\_predict(X):

return (((elastic\_model\_full\_data.predict(X)) + \

(lasso\_model\_full\_data.predict(X)) + \

(ridge\_model\_full\_data.predict(X)) + \

(svr\_model\_full\_data.predict(X)) + \

(gbr\_model\_full\_data.predict(X)) + \

(xgb\_model\_full\_data.predict(X)) + \

(lgb\_model\_full\_data.predict(X)) + \

(stack\_gen\_model.predict(np.array(X))))/8)

print('RMSLE score on train data:')

print(rmsle(y, blend\_models\_predict(data\_train)))

test\_id = test\_data[['Id']]

test\_id=test\_id.Id.astype(int)

predict = pd.DataFrame(np.exp(blend\_models\_predict(data\_test)), columns= ['SalePrice'])

submission = pd.concat([test\_id, predict],axis =1)

submission.to\_csv("submission1.csv",header = True, index =False)

feats = {} # a dict to hold feature\_name: feature\_importance

for feature, importance in zip(data\_train.columns, gbr.feature\_importances\_):

feats[feature] = importance #add the name/value pair

importances = pd.DataFrame.from\_dict(feats, orient='index').rename(columns={0: 'Gini-importance'})

importances[importances['Gini-importance']>0.005]

importances.sort\_values(by='Gini-importance',ascending=False)

**Scored and Submitted Data:**

****

**Python Notebook Hyperlink:**

****