

# MMA867 Kaggle Competition

## Housing Prices Prediction



By:

Gopala Goyal

20254605

Competition Rank: 1356

## Overview

Buying a house is one big milestone for a person and is not an easy decision. The person buying a home must think from multiple perspectives while choosing a home. For example, how many rooms are there in the house, number of floors, locality, quality of the house, year it was built, and the list is endless. Each feature of the house and its surrounding impacts the price of the house. As a data-scientist, if we have the right data and knowledge of the algorithms, we can help the consumer to choose a right home and focus on the features that should be the key focus area.

As a part of a Kaggle competition, we were provided with the data of residential homes in Ames, Iowa, and we have a challenge to predict the final price of each home. The below report provides the insights about the dataset, steps followed to perform data-wrangling, pre-processing and the process used to apply predictive modelling techniques to estimate the price of the houses.

## Datasets

As per the Kaggle competition, the dataset has been provided to us where we have many factors (about 80) including the number of rooms, area of the house, whether or not it has a swimming pool, quality, year built, modified and so on. All these variables or factors are used to determine the price of the house. The dataset contains a training dataset as well a separate test dataset that can be used to see how efficient the model is.

The data comprises of both categorical as well as numerical variables that would be used to create a regression model which in-turn would be used to predict the house prices for the test dataset.

The data provided is not clean, i.e. it contains a lot of missing values for both categorical as well as numerical data. Since we have a data description file provided to us by Kaggle, we would be utilizing the file to understand more about the data and use it to complete/fill the missing values.

```
3 | hp_train_df.head()
```

:

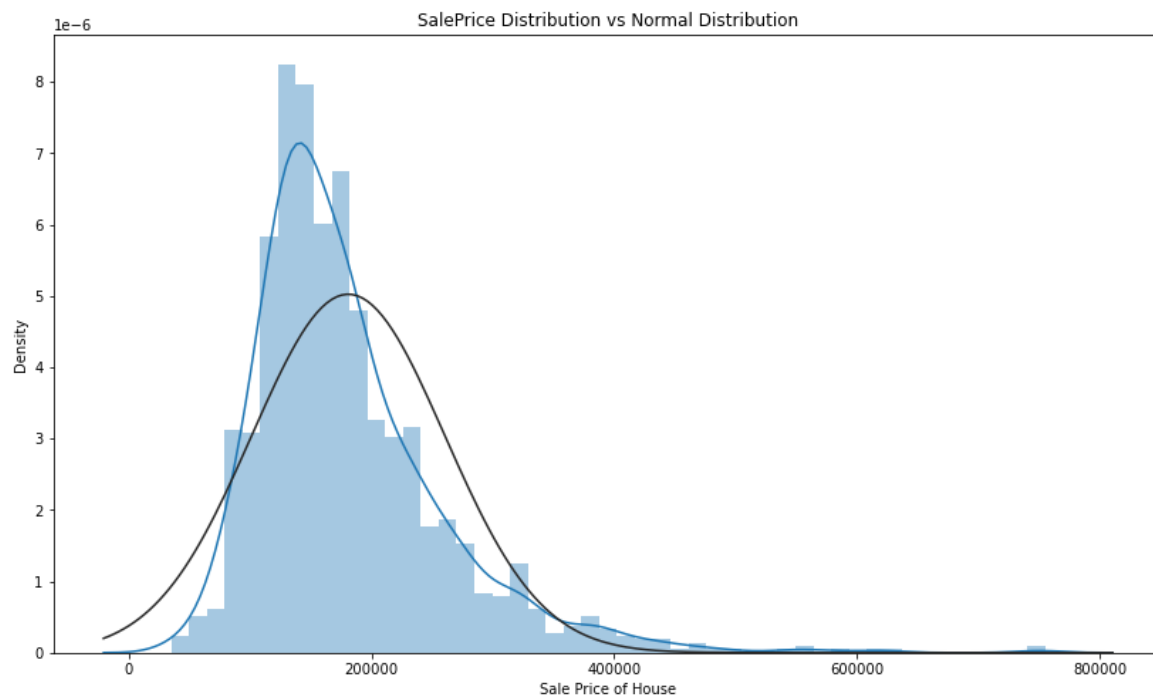
	Id	MSSubClass	MSZoning	LotFror
0	1	60	RL	
1	2	20	RL	
2	3	60	RL	
3	4	70	RL	
4	5	60	RL	

5 rows x 81 columns

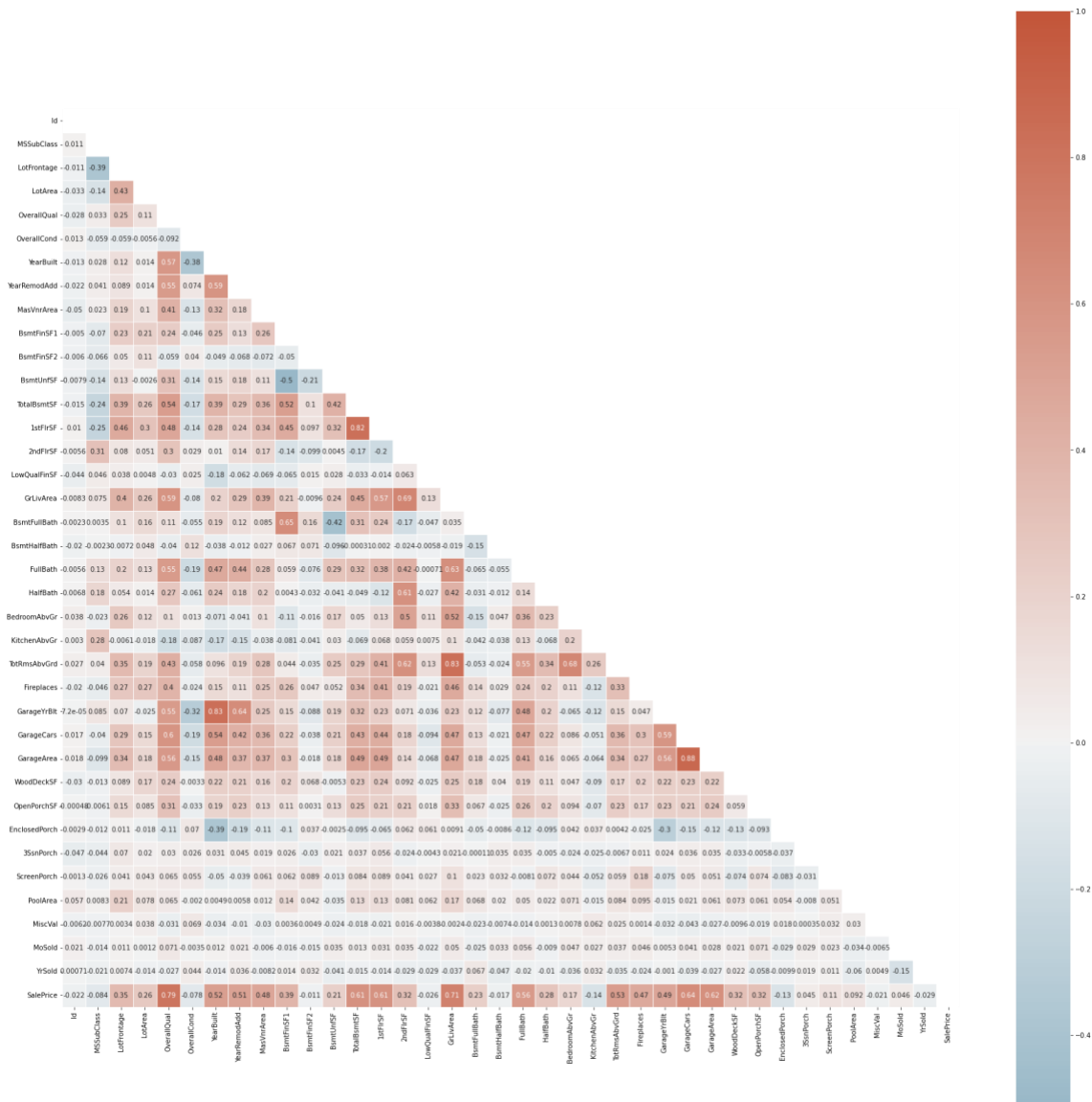
## Exploratory Data Analysis – (EDA)

As a part of the EDA, we analyzed the data by validating the structure, sanity and exploring the variables by comparing them with the Sale Price variable.

We now plotted a histogram to check the normality of the data; if our data is not normally distributed then we will try to transform our data, since the data model requires data to be normally distributed. We tried to plot a normal distribution graph for the dependent variable Sale price and found that it is skewed towards the right. We would be performing log transformation to scale the data and the target variable and normalize it.



The correlation matrix that has been generated, shows that there are around 11 independent variables that are correlated with more than 50% correlation factor with the Sale Price dependent variable. To continue the exploration, we picked the individual variables and generated different graphs and plots to see how the relation between it and the Sale Price variable is. As per the observations, the following variables seem to have a positive relation with Sale Price. In case the independent variable increases, the Sale Price increases and is positively inclined. Also, we found that there are many variables that have empty or NaN values. Also, there are a few outliers that have been observed in the dataset.

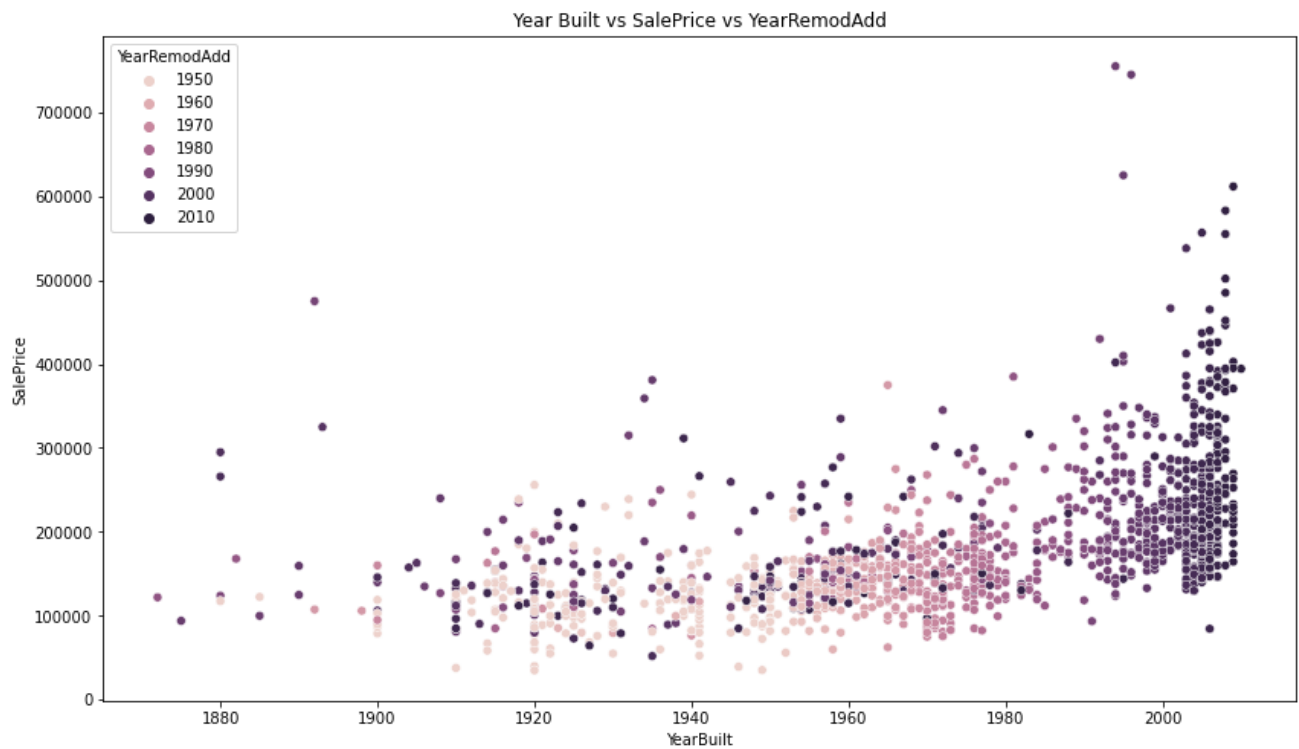
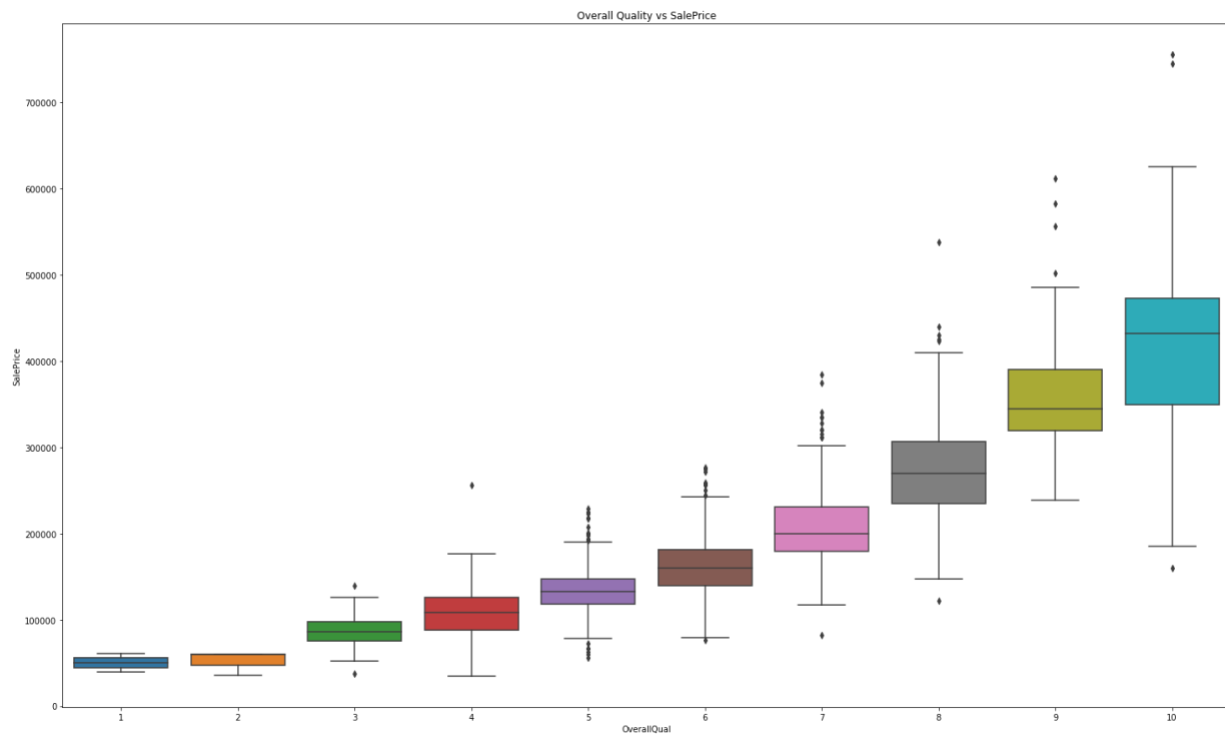


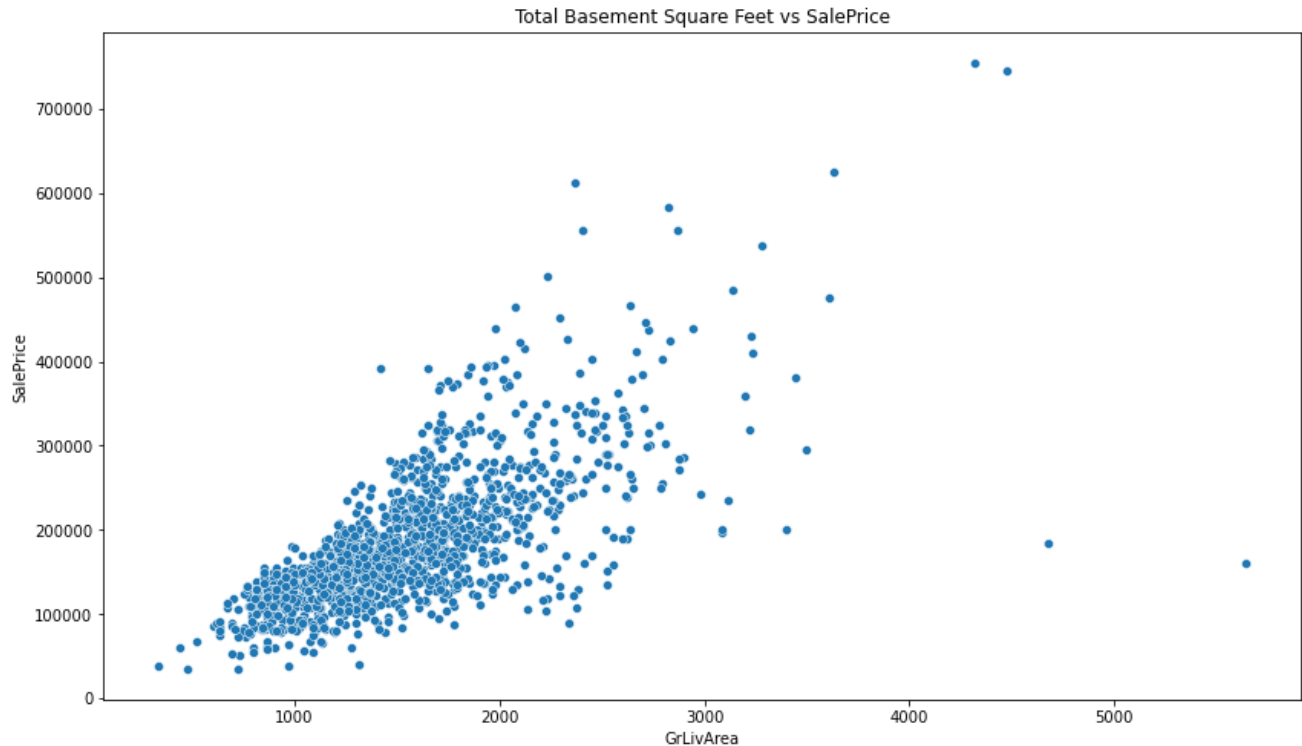
Post the EDA, we would proceed with the data wrangling process where we would pre-process the data and prepare it for modelling and running predictive analysis.

As a summary, we did the following EDA:

1. Studying the dependency of a single dependent variable i.e Sales Price (Univariate Analysis)
2. Studying the effect of dependent variable on independent variable (Multivariate Analysis)

The below visualizations show the relations between SalePrice variable and some highly correlated variables:





## Data Wrangling

Data wrangling process is concerned with cleaning the dataset that has been provided to us. Here, our target is to make the data free from any discrepancy, including missing data, invalid values, and inconsistencies.

For categorical variables, there are a couple of ways that we can fill the null/NA values, that include filling it with mode or removing the values that have significantly high number of missing values. For numeric/non-categorical variables, we can clean and fill-in the missing values using the multiple imputations method. Multiple imputations can be applied using different machine learning algorithms like linear regression, K nearest neighbors, Random Forest and so on.

The following process was followed in-order to perform the data wrangling –

Since we are provided with the test and the training data, we merged both the datasets and performed data wrangling. The reason to merge the datasets is that while we do the wrangling, there could be some independent variables that we remove or add, and hence this should be consistent between the training and the testing dataset.

After merging, we calculated the percentage of missing values in the dataset for each column. Using these percentages, we found that there are many categorical as well as numerical columns that have missing/NA values.

```

5 null_v = null_v.sort_values(by = ['Nan_Sum'])
6 null_v['Disc_Keep'] = np.where(null_v['Perc%']>30, 'Discard', 'Keep')
7 null_v['DataType'] = hp_tr_te.dtypes
8 null_v

```

	Nan_Sum	Perc%	Disc_Keep	DataType
TotalBsmtSF	1	0.034258	Keep	float64
GarageArea	1	0.034258	Keep	float64
GarageCars	1	0.034258	Keep	float64
KitchenQual	1	0.034258	Keep	object
Electrical	1	0.034258	Keep	object

1. To clean the categorical variables, we have filled the NA values with the mode of the columns.
2. To clean the numerical values, we have used sklearn's impute package and used KNNImputer to calculate the numeric values on the basis of their neighbors and input the missing values.
3. We had to separate out the numeric and the categorical variables from the dataset because KNNImputer gives error in case there are categorical variables present in the dataset.

```

1 #As an assumption, we fill the NA values for the qualitative variables to the Mode(value that occurs the most)
2 hp_tr_te['KitchenQual'] = hp_tr_te['KitchenQual'].fillna('TA')
3 hp_tr_te['Electrical'] = hp_tr_te['Electrical'].fillna(hp_tr_te['Electrical'].mode()[0])
4 hp_tr_te['SaleType'] = hp_tr_te['SaleType'].fillna(hp_tr_te['SaleType'].mode()[0])
5 hp_tr_te['Exterior1st'] = hp_tr_te['Exterior1st'].fillna(hp_tr_te['Exterior1st'].mode()[0])
6 hp_tr_te['Exterior2nd'] = hp_tr_te['Exterior2nd'].fillna(hp_tr_te['Exterior2nd'].mode()[0])
7 hp_tr_te['Functional'] = hp_tr_te['Functional'].fillna(hp_tr_te['Functional'].mode()[0])
8 hp_tr_te['Utilities'] = hp_tr_te['Utilities'].fillna(hp_tr_te['Utilities'].mode()[0])
9 hp_tr_te['MSZoning'] = hp_tr_te['MSZoning'].fillna(hp_tr_te['MSZoning'].mode()[0])
10
11 #For the variables that have significantly high missing values, on the basis of the data description file, we can s
12 for col in ['MasVnrType', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'FireplaceQu', 'Fence', 'Alley', 'Mi
13     hp_tr_te[col] = hp_tr_te[col].fillna('None')
14

```

```

1 #To Handle missing values in the numeric variables, we would use multiple imputations
2 from sklearn.impute import KNNImputer
3 ttn = hp_tr_te.select_dtypes(include=[np.number])
4 ttc = hp_tr_te.select_dtypes(exclude=[np.number])
5 imputer = KNNImputer(n_neighbors=5, weights='uniform', metric='nan_euclidean')
6 ttn = pd.DataFrame(imputer.fit_transform(ttn), columns = ttn.columns)
7
8 #Reset ttc index
9 ttc = ttc.reset_index(drop=True)
10
11 #Concatenate TTN & TTC
12 hp_tr_te = pd.concat([ttn, ttc], axis=1)

```

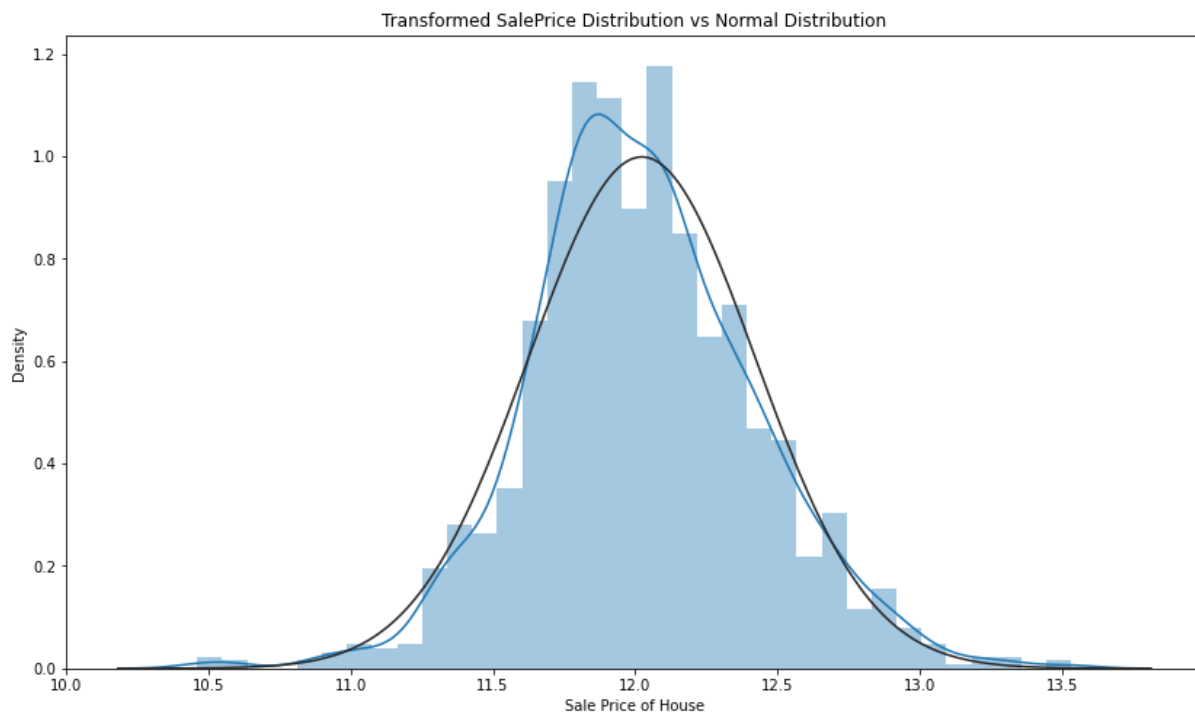
## Data Transformation and Feature Engineering

The process of data transformation is a key aspect of the data analysis and predictive modelling process. This ensures that the data has the following attributes:

1. Smoothness
2. Low noise
3. Aggregated
4. Normalized

Log transformation is a process of data transformation where the values in the dataset are replaced with logarithmic values. The choice of the logarithm base is usually left up to the analyst and it would depend on the purposes of statistical modeling. In this project, we would be using the natural log transformation.

We used logarithmic transformation because the dataset has skewed values. We can transform the data to make it asymptotically normal so that the analysis results from this data are valid. Also, because our target variable is skewed towards the right, we applied logarithmic transformations so that the skewness is removed. As a process, this logarithmic transformation would be removed at the end of the modelling process to estimate the housing prices.



As a part of the Feature engineering process, we create different new independent variables that could help in creating a robust regression model. This also ensures that there is more understanding between the variables and hence can help in running more hypothesis tests. We then split the data back (that we initially merged) into test and training dataset. Furthermore, we



split the dataset into training and validation dataset using the Sklearn's `test_train_split()` method that splits the data randomly.

The following feature engineering process was followed in the current data:

1. We created three interaction variables, `TotalSquareFeetArea`, `Total_Home_Quality` and `Total_Bathrooms`. According to the business understanding, these are the 3 important factors that one looks for once the person is trying to buy a home.
2. To use the category variables in the regression model, we converted all the categorical variables into dummy variables using the `get_dummies()` function.
3. We also separated out the numeric variables to check the skewness and apply logarithmic transformation to correct the skewness for each of the skewed variables.

We would use these features as a part of our model creation and applying the regularization models.

```
#Split Train and test data
training_data = hp_tr_te_dum[:1460]
testing_data = hp_tr_te_dum[1460:]
testing_data.reset_index(drop=True)
#testing_data['Id'] = range(1460, 1460+len(testing_data))

#Splitting training data into training and validation set
from sklearn.model_selection import train_test_split
X_train, X_val, Y_train, Y_val = train_test_split(training_data, dependent_log, test_size=0.25, random_state=42)
```

## Modelling Process

For any Linear Regression model the algorithm tries to minimize the cost function by trying to achieve the minimum mean square error (MSE) of the predicted vs the training data. Adding too many variables into a linear model can cause the model to overfit the training data and hence can lead to higher MSE for the model.

In order to prevent the model from overfitting, we apply the concept of regularization. Using the regularization technique, we introduce a controlling parameter in the cost function for a model that helps to regulate the independent variables and their effect on the cost function. The magnitude of the regularization variable is controlled with a coefficient known as  $\lambda$ . In other words, it is the technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values.

For this dataset, we tried to model and apply regularization technique using the following 2 techniques:

1. LASSO Cross Validation Regression
2. Ridge Regression & Cross Validation

Both regression models select the entire dataset as an input with all the columns/features as well as the features created in the feature engineering process. The model is created and is fit to the training dataset, validated using the validation dataset that is split from the training dataset, and the corresponding score and mean square error are calculated. If we are satisfied with the MSE,

or in other words, the MSE is low enough, we can proceed to test the model on the test data provided. Else, we would have to re-engineer and fine tune the model on the basis of business understanding and domain knowledge. If the mean squared error in the validation dataset is not low or needs improvement, the above process is then repeated until we achieve a low mean squared error for the validation dataset.

### LASSO Cross Validation Regression

For this dataset, we created a few interactions as discussed in the Feature Engineering section and executed the LASSO cross validation regression. Using this we add a regularization term that is the absolute sum of the magnitude of the coefficients of each of the features of the dataset. This regularization term is known as L1 penalty as it penalizes the model and prevents it from overfitting. Using LASSO helps in parameter shrinkage and variable selection due to the ability to zero-down the coefficient of features that are not significant and have low predictive power. We applied k-fold cross validation in addition to the LASSO regression that helped us choose the value of lambda which gives the lowest error rate/MSE.

```
LassoCV(alphas=[0.1, 0.3, 0.5, 0.8, 1], cv=5, verbose=True)
Alpha:0.10, R2:0.694, MSE:0.06, RMSE:0.23
```

---

### Ridge Regression

To apply and explore a different regularized regression technique for the housing price dataset, we tried to implement Ridge regression along with k-fold cross validation to predict the housing prices with a low mean square error. By using ridge regression, a regularization term is added to the cost function that is a square of the coefficients of each of the features in the dataset. Addition of this regularization term penalizes the model and prevents it from being an overfit model. This penalty term is known as L2 penalty. Here, all the coefficients get reduced/shrunk by the same factor unlike the LASSO regression. When combined with K-fold cross validation, we randomly assign samples to one of  $k$  folds, then estimate the error statistic in the usual  $k$ -fold cross validation way. This helps in getting the estimates for the regularization term coefficient (Lambda) that helps in controlling the impact of regularization term on the model.

```
Ridge(alpha=0.14)
R2:0.948, MSE:0.02, RMSE:0.13
```

---

## Conclusion

By applying both regularized regression modelling algorithms on the housing prices dataset, developing interactions and features that could be considered as significant in predicting the prices of houses, we conclude that Ridge regression seems more accurate. The mean squared error for LASSO is being estimated to be about 0.06 however for Ridge, it is being estimated to be about 0.015 which is comparatively lower than LASSO.

The performance of the Ridge regression model is better than LASSO is because of the feature engineering process. While applying linear modelling, the art and science of creating features and choosing the right combination of independent variables is the key factor that aids in creating significantly accurate predictive models.

Even though the mean squared errors are low, there is still a lot of improvements that can be done in modelling and feature engineering. There can be a greater number of interactions that provide better relations and combinations between the features in dataset. This can help in high accuracy and models with higher predictive power and estimation capabilities.

## Technical Appendix

### 1. Feature Engineering

```
#Created 3 features that I find are one of the most important as a combination
hp_tr_te['TotalSquareFeetArea'] = hp_tr_te['GrLivArea'] + hp_tr_te['1stFlrSF'] + hp_tr_te['2ndFlrSF'] + hp_tr_te['TotalBs
hp_tr_te['Total_Home_Quality'] = hp_tr_te['OverallQual'] + hp_tr_te['OverallCond']
hp_tr_te['Total_Bathrooms'] = (hp_tr_te['FullBath'] + (0.5 * hp_tr_te['HalfBath']) +
                             hp_tr_te['BsmtFullBath'] + (0.5 * hp_tr_te['BsmtHalfBath']))

#Dummy Variable creation
hp_tr_te_dum = pd.get_dummies(hp_tr_te)

#Separate out the numeric variables to check the skewness and apply logarithmic transformation to correct the skew
from scipy.stats import skew
hp_tr_te_num = hp_tr_te_dum.dtypes[hp_tr_te_dum.dtypes != object].index
skewed_features = hp_tr_te_dum[hp_tr_te_num].apply(lambda x: skew(x)).sort_values(ascending=False)
high_skew = skewed_features[skewed_features > 0.5]
skew_index = high_skew.index

# Normalize skewed features using log transformation
for i in skew_index:
    hp_tr_te_dum[i] = np.log1p(hp_tr_te_dum[i])
```

### 2. LASSO Regression

```
1 #LASSO
2
3 from sklearn.linear_model import Lasso, LassoCV
4 from sklearn.metrics import mean_squared_error
5 alphas = [0.1, 0.3, 0.5, 0.8, 1]
6 lassocv = LassoCV(alphas=alphas, cv=5, verbose=True).fit(X_train, Y_train)
7 print(lassocv)
8
9 score = lassocv.score(X_train, Y_train)
10 ypred = lassocv.predict(X_val)
11 mse = mean_squared_error(Y_val, ypred)
12
13 print("Alpha:{0:.2f}, R2:{1:.3f}, MSE:{2:.2f}, RMSE:{3:.2f}"
14       .format(lassocv.alpha_, score, mse, np.sqrt(mse)))

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
.....

LassoCV(alphas=[0.1, 0.3, 0.5, 0.8, 1], cv=5, verbose=True)
Alpha:0.10, R2:0.694, MSE:0.06, RMSE:0.23

.....[Parallel(n_jobs=1)]: Done 5 out of 5 | elapsed: 0.2s finished
```



### 3. Ridge Regression

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge, RidgeCV
def rmse_cv(model):
    rmse = np.sqrt(-cross_val_score(model, X_train, Y_train, scoring = "neg_mean_squared_error", cv = 5))
    return(rmse)

ridge = RidgeCV(alphas = np.arange(0.01, 1, 0.01)).fit(X_train, Y_train)
print(rmse_cv(ridge).mean())

#Ridge Regression
model = Ridge(alpha=0.14)
model.fit(X_train, Y_train)
score = model.score(X_train, Y_train)
ypred = model.predict(X_val)
mse = mean_squared_error(Y_val, ypred)
print(model)
print("R2:{0:.3f}, MSE:{1:.2f}, RMSE:{2:.2f}".format(score, mse, np.sqrt(mse)))
```

#### 4. Competition Rank

1356	Gopala Goyal		0.13318	4	2d
<b>Your Best Entry</b> 					
Your submission scored 0.14138, which is not an improvement of your best score. Keep trying!					