**HOUSING: PRICE PREDICTION PROJECT**

**By SREEKARI. I**

**INTRODUCTION**

Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world’s economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases.

Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company.

A US-based housing company named Surprise Housing has decided to enter the Australian market. The company is looking at prospective properties to buy houses to enter the market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price.

**OBJECTIVE**

We are going to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not. This model will then be used by the management to understand how exactly the prices vary with the variables. They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns. Further, the model will be a good way for the management to understand the pricing dynamics of a new market.

**DATA SOURCE**

Data is the a crucial element in data analysis and machine learning. In this project, we are using the data collected by the company. The data set consists of the details of sale of houses in Australia.

**DATA ANALYSIS**

As we have the required data, let us look into it and know its attributes. We can see that the data has 1168 rows and 81 columns.

The basic steps involved in the data analysis are

**Cleaning**:

It is the first module called to clean the item and verify that all the information in it correspond to the pattern used to extract it. The cleaning module removes the noise, and check that all the values are not empty, otherwise the item is dropped. This is done for simplicity, indeed, it could be better to try to inference them later. After the cleaning part done, the item is sent to the formatting module.

**Formatting**:

The second module is used to format the item’s values as we want. A basic example could be for the price, initially got being string type, is converted as float. This is done for every numeric values.

**Model creation:**

Once the data is cleaned and formatted, the predictive model can be created. After this, the model is saved and used for prediction.

Let us perform these steps on the data we have.

**DATA CLEANING**

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Id 1168 non-null int64

1 MSSubClass 1168 non-null int64

2 MSZoning 1168 non-null object

3 LotFrontage 954 non-null float64

4 LotArea 1168 non-null int64

5 Street 1168 non-null object

6 Alley 77 non-null object

7 LotShape 1168 non-null object

8 LandContour 1168 non-null object

9 Utilities 1168 non-null object

10 LotConfig 1168 non-null object

11 LandSlope 1168 non-null object

12 Neighborhood 1168 non-null object

13 Condition1 1168 non-null object

14 Condition2 1168 non-null object

15 BldgType 1168 non-null object

16 HouseStyle 1168 non-null object

17 OverallQual 1168 non-null int64

18 OverallCond 1168 non-null int64

19 YearBuilt 1168 non-null int64

20 YearRemodAdd 1168 non-null int64

21 RoofStyle 1168 non-null object

22 RoofMatl 1168 non-null object

23 Exterior1st 1168 non-null object

24 Exterior2nd 1168 non-null object

25 MasVnrType 1161 non-null object

26 MasVnrArea 1161 non-null float64

27 ExterQual 1168 non-null object

28 ExterCond 1168 non-null object

29 Foundation 1168 non-null object

30 BsmtQual 1138 non-null object

31 BsmtCond 1138 non-null object

32 BsmtExposure 1137 non-null object

33 BsmtFinType1 1138 non-null object

34 BsmtFinSF1 1168 non-null int64

35 BsmtFinType2 1137 non-null object

36 BsmtFinSF2 1168 non-null int64

37 BsmtUnfSF 1168 non-null int64

38 TotalBsmtSF 1168 non-null int64

39 Heating 1168 non-null object

40 HeatingQC 1168 non-null object

41 CentralAir 1168 non-null object

42 Electrical 1168 non-null object

43 1stFlrSF 1168 non-null int64

44 2ndFlrSF 1168 non-null int64

45 LowQualFinSF 1168 non-null int64

46 GrLivArea 1168 non-null int64

47 BsmtFullBath 1168 non-null int64

48 BsmtHalfBath 1168 non-null int64

49 FullBath 1168 non-null int64

50 HalfBath 1168 non-null int64

51 BedroomAbvGr 1168 non-null int64

52 KitchenAbvGr 1168 non-null int64

53 KitchenQual 1168 non-null object

54 TotRmsAbvGrd 1168 non-null int64

55 Functional 1168 non-null object

56 Fireplaces 1168 non-null int64

57 FireplaceQu 617 non-null object

58 GarageType 1104 non-null object

59 GarageYrBlt 1104 non-null float64

60 GarageFinish 1104 non-null object

61 GarageCars 1168 non-null int64

62 GarageArea 1168 non-null int64

63 GarageQual 1104 non-null object

64 GarageCond 1104 non-null object

65 PavedDrive 1168 non-null object

66 WoodDeckSF 1168 non-null int64

67 OpenPorchSF 1168 non-null int64

68 EnclosedPorch 1168 non-null int64

69 3SsnPorch 1168 non-null int64

70 ScreenPorch 1168 non-null int64

71 PoolArea 1168 non-null int64

72 PoolQC 7 non-null object

73 Fence 237 non-null object

74 MiscFeature 44 non-null object

75 MiscVal 1168 non-null int64

76 MoSold 1168 non-null int64

77 YrSold 1168 non-null int64

78 SaleType 1168 non-null object

79 SaleCondition 1168 non-null object

80 SalePrice 1168 non-null int64

dtypes: float64(3), int64(35), object(43)

memory usage: 739.2+ KB

We can see that there are three(3) columns of float64 datatype, thirty five(35) columns of int64 datatype and forty three(43) columns of object datatype.

There are some null values in the dataset. Now, I am checking for missing values using the below code.

# checking the null values

df.isnull().sum()

This shows the sum of missing values of the data.

We can see that there are missing values in LotFrontage, Alley, MasVnrType, MasVnrArea, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, FireplaceQu, GarageType, GarageYrBlt, GarageFinish, GarageQual, GarageCond, PoolQC, Fence and MiscFeature.

**Handling Missing Values**

**Let us remove these missing values as they affect the model prediction. For this, let us check the percentage of missing values in each column.**

**Code:**

# Getting the number of missing values in each column

num\_missing = df.isna().sum()

# Excluding columns that contains 0 missing values

num\_missing = num\_missing[num\_missing > 0]

# Getting the percentages of missing values

percent\_missing = num\_missing \* 100 / df.shape[0]

# Concatenating the number and perecentage of missing values

# into one dataframe and sorting it

pd.concat([num\_missing, percent\_missing], axis=1,

keys=['Missing Values', 'Percentage']).\

sort\_values(by="Missing Values", ascending=False)

**Output**:

MissingValues Percentage

PoolQC 1161 99.400685

MiscFeature 1124 96.232877

Alley 1091 93.407534

Fence 931 79.708904

FireplaceQu 551 47.174658

LotFrontage 214 18.321918

GarageType 64 5.479452

GarageYrBlt 64 5.479452

GarageFinish 64 5.479452

GarageQual 64 5.479452

GarageCond 64 5.479452

BsmtExposure 31 2.654110

BsmtFinType2 31 2.654110

BsmtCond 30 2.568493

BsmtFinType1 30 2.568493

BsmtQual 30 2.568493

MasVnrArea 7 0.599315

MasVnrType 7 0.599315

The percentage of missing values in PoolQC column is 99.40% which is very high. A missing value in this column might denote that the corresponding house doesn't have a pool. To verify this, let's take a look at the values of Pool Area column.

**PoolArea**:

We can see that there are 1161 entries in PoolArea column that have a value of 0. This verfies our hypothesis that each house without a pool has a missing value in Pool QC column and a value of 0 in Pool Area column. So let's fill the missing values in Pool QC column with "No Pool"

df["PoolArea"].fillna("No Pool", inplace=True)

**PoolQC**:

The percentage of missing values in Pool QC column is 96.23% which is very high also. Let's take a look at the values of Misc Val column.

df["PoolQC"].fillna("No Pool", inplace=True)

**MiscFeature**

We can see that MiscVal column has 1126 entries with a value of 0. Then, as with Pool QC, we can say that each house without a "miscellaneous feature" has a missing value in Misc Feature column and a value of 0 in Misc Val column. So let's fill the missing values in Misc Feature column with "No Feature":

df['MiscFeature'].fillna('No feature', inplace=True)

According to the dataset documentation, NA in Alley, Fence, and FireplaceQu columns denotes that the house doesn't have an alley, fence, or fireplace. So we fill in the missing values in these columns with "No Alley", "No Fence", and "No Fireplace" accordingly:

df['Alley'].fillna('No Alley', inplace=True)

df['Fence'].fillna('No Fence', inplace=True)

df['FireplaceQu'].fillna('No Fireplace', inplace=True)

As we saw previously, Lot Frontage represents the linear feet of street connected to the house. So we assume that the missing values in this column indicates that the house is not connected to any street, and we fill in the missing values with 0

df['LotFrontage'].fillna(0, inplace=True)

According to the dataset documentation, NA in GarageCond, GarageQual, GarageFinish, and GarageType indicates that there is no garage in the house. So we fill in the missing values in these columns with "No Garage". We notice that GarageCond, GarageQual, GarageFinish, GarageYrBlt columns have 64 missing values each.

**Code**:

df['GarageCars'].fillna(0, inplace=True)

df['GarageArea'].fillna(0, inplace=True)

df.loc[~pd.isna(df['GarageType']) &

pd.isna(df['GarageQual']), "GarageType"] = "No Garage"

for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']:

df[col].fillna('No Garage', inplace=True)

df['GarageYrBlt'].fillna(0, inplace=True)

Similarly, checking remaining columns that has missing values and replacing them.

for col in ["BsmtHalfBath", "BsmtFullBath", "TotalBsmtSF",

"BsmtUnfSF", "BsmtFinSF2", "BsmtFinSF1"]:

df[col].fillna(0, inplace=True)

df.loc[~pd.isna(df['BsmtCond']) &

pd.isna(df['BsmtExposure']), "BsmtExposure"] = "No"

df.loc[~pd.isna(df['BsmtCond']) &

pd.isna(df['BsmtFinType2']), "BsmtFinType2"] = "Unf"

for col in ["BsmtExposure", "BsmtFinType2",

"BsmtFinType1", "BsmtQual", "BsmtCond"]:

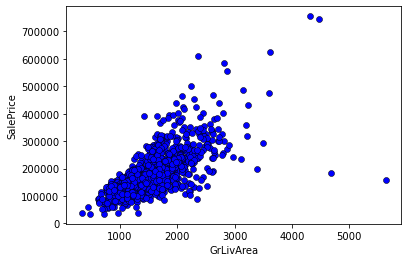
df[col].fillna("No Basement", inplace=True)

df['MasVnrArea'].fillna(0, inplace=True)

df['MasVnrType'].fillna("None", inplace=True)

We have removed all the missing values now.

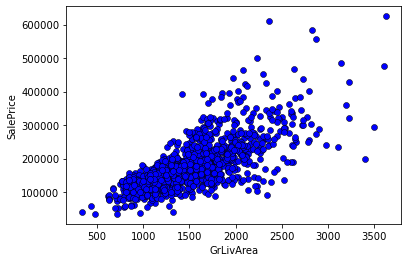
**HANDLING OUTLIERS**



We can see that there are few outliers. Now, we will remove them from our dataset. We can do so by keeping data points that have GrLivArea less than 4,000. But first we take a look at the dataset rows that correspond to these unusual values

# removing outliers from GrLivArea column

df = df[df["GrLivArea"] < 4000]



#To avoid problems in modeling later, we will reset our dataset index after removing the outlier rows, so no gaps remain in our dataset inde

df.reset\_index(drop=True, inplace=True)

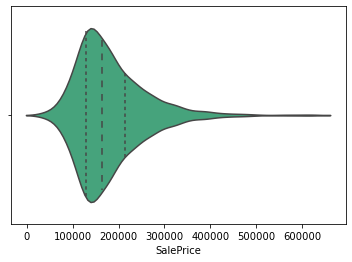
**Deleting unnecessary columns:**

There are some columns in the dataset which are not useful in predicting the model. We can remove such columns.

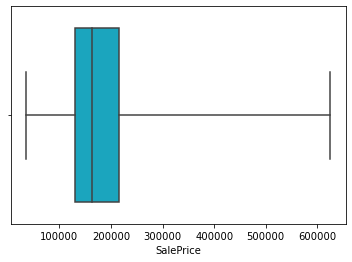
 The column to be deleted is Id

df.drop(['Id'], axis=1, inplace=True)

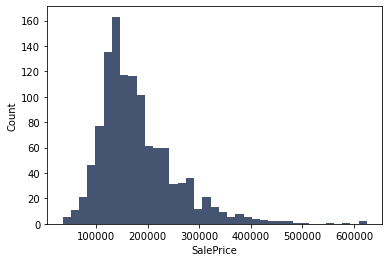
Let us understand the target variable(SalePrice) distribution now.



We can see from the plot that most house prices fall between 100,000 and 250,000. The dashed lines represent the locations of the three quartiles Q1, Q2 (the median), and Q3.



This shows us the minimum and maximum values of SalePrice. It shows us also the three quartiles represented by the box and the vertical line inside of it.



Let us check correlation now.

**CORRELATION**

**Code**:

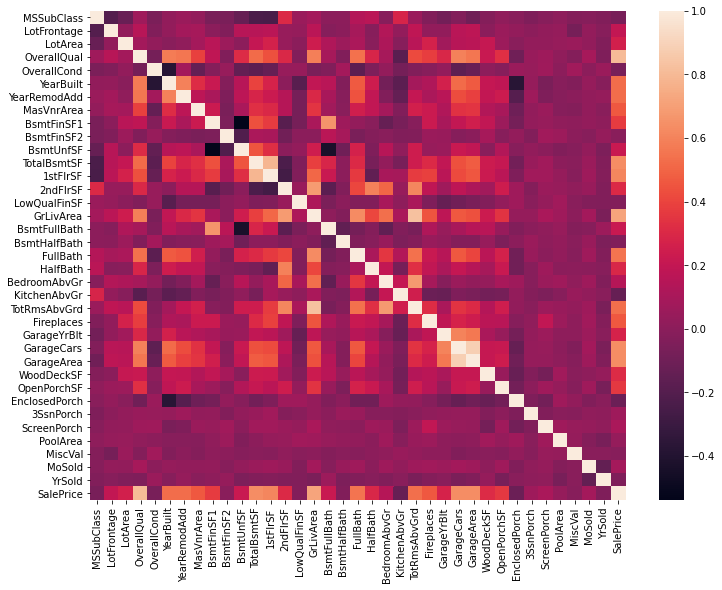
fig, ax = plt.subplots(figsize=(12,9))

sns.heatmap(df.corr(), ax=ax);

We want to see how the dataset variables are correlated with each other and how predictor variables are correlated with the target variable. For example, we would like to see how Lot Area and SalePrice are correlated: Do they increase and decrease together (positive correlation)? Does one of them increase when the other decrease or vice versa (negative correlation)? Or are they not correlated?

Correlation is represented as a value between -1 and +1 where +1 denotes the highest positive correlation, -1 denotes the highest negative correlation, and 0 denotes that there is no correlation.

We will show correlation between our dataset variables (numerical and boolean variables only) using a heatmap graph:

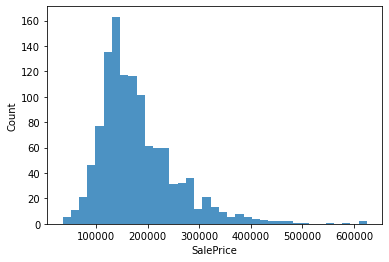


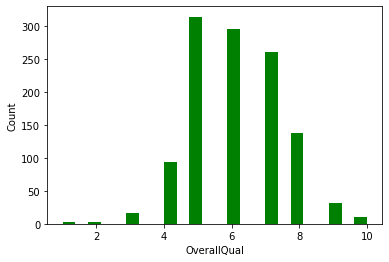
Observations:

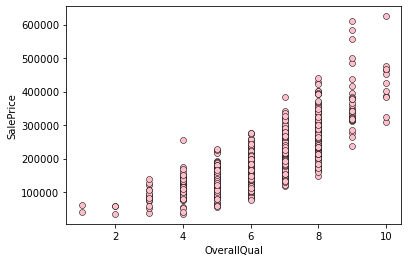
* We notice that GarageCars and GarageArea have high positive correlation which is reasonable because when the garage area increases, its car capacity increases too.
* We see also that GrLivArea and TotRms AbvGrd are highly positively correlated which also makes sense because when living area above ground increases, it is expected for the rooms above ground to increase too.
* We can see that BsmtUnfSF is negatively correlated with BsmtFinSF1
* We note also that BsmtUnfSF is negatively correlated with BsmtFullBath
* We see that the target variable(SalePrice) is highly positively correlated with OverallQual and GrLivArea. We see also that the target variable is positively correlated with YearBuilt, Year Remod/Add, MasVnrArea, TotalBsmtSF, 1stFlrSF, FullBath, GarageCars, and GarageArea.

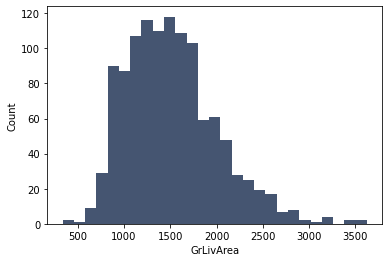
**Let us check the relation between target variable and other variables**

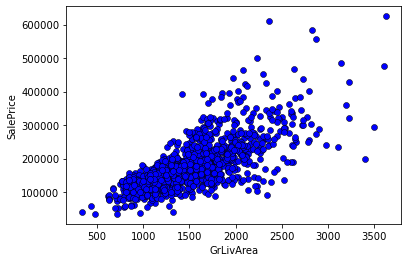
----> High Positive Correlation



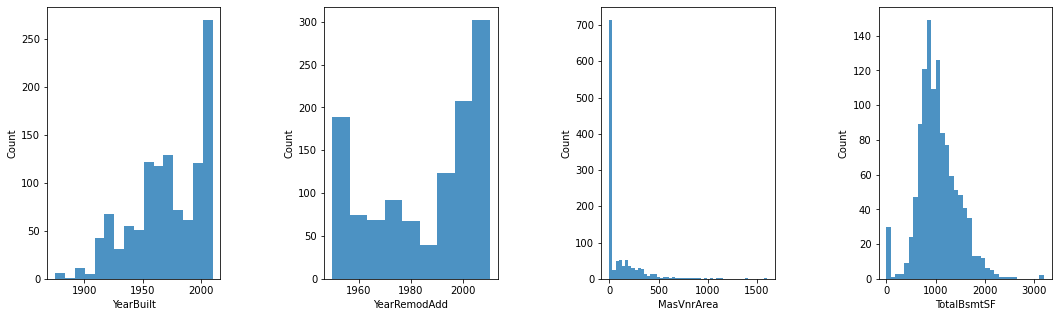




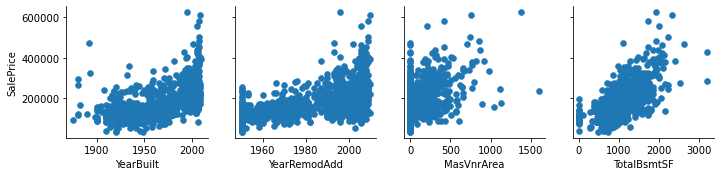


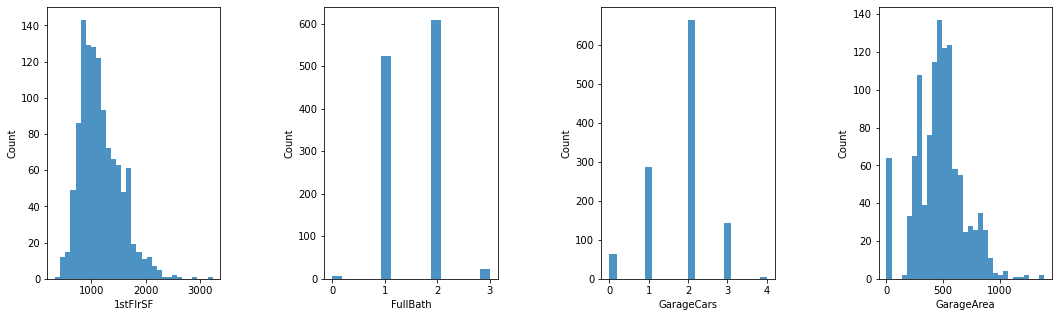


**----> Moderate Positive Correlation**

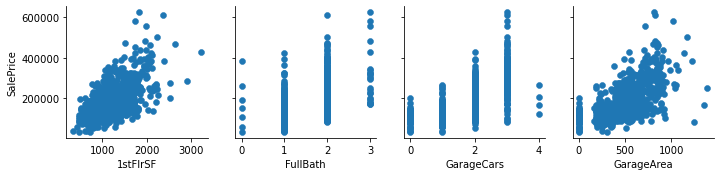


Now let us see their relationships with the target variable using scatter plots:

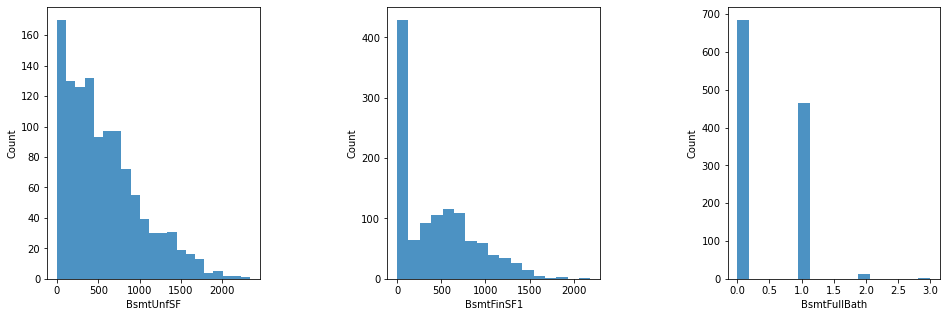


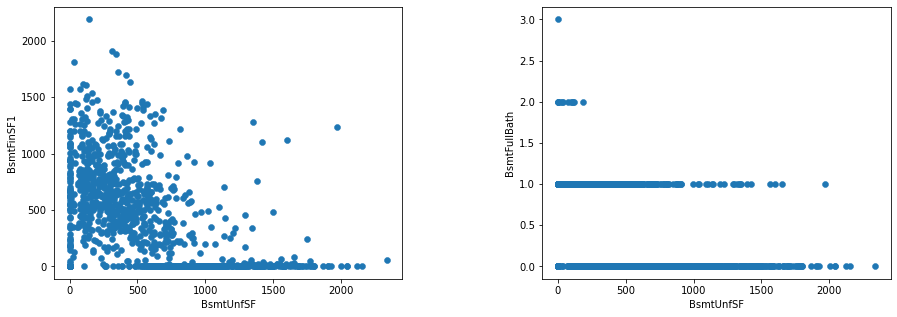


Relationship with target variable



**----> Negative relationship**





**DATA FORMATTING**

Let us encode the data i.e, converting the non numerical data into numerical data.

**---> Categorical Features**

Here I am using One hot encoding

# One hot encoding

from sklearn.preprocessing import OneHotEncoder

from sklearn import preprocessing

enc = OneHotEncoder(handle\_unknown='ignore')

labelencoder=preprocessing.LabelEncoder()

df['PavedDrive'] = labelencoder.fit\_transform(df["PavedDrive"])

df['Exterior1st'] = labelencoder.fit\_transform(df["Exterior1st"])

df['Exterior2nd'] = labelencoder.fit\_transform(df["Exterior2nd"])

df['MasVnrType'] = labelencoder.fit\_transform(df["MasVnrType"])

df['Foundation'] = labelencoder.fit\_transform(df["Foundation"])

df['Heating'] = labelencoder.fit\_transform(df["Heating"])

df['GarageType'] = labelencoder.fit\_transform(df["GarageType"])

df['Electrical'] = labelencoder.fit\_transform(df["Electrical"])

df['MiscFeature'] = labelencoder.fit\_transform(df["MiscFeature"])

df['SaleType'] = labelencoder.fit\_transform(df["SaleType"])

df['SaleCondition'] = labelencoder.fit\_transform(df["SaleCondition"])

**---> Ordinal variables**

Here, I am assigning numerical values to all the non numerical values of Ordinal variables. Example:

'HouseStyle' has ‘1Story’, ‘2Story’, ‘1.5Fin’, '1.5Fin':2, 'SFoyer', '1.5Unf', 'SLvl', '2.5Fin', '2.5Unf' as unique values.

I am assigning 0,1,2,3,4,5,6,7 to these unique values respectively.

**Code**:

df['HouseStyle'] = df['HouseStyle'].map({'1Story':0, '2Story':1, '1.5Fin':2, 'SFoyer':3, '1.5Unf':4, 'SLvl':5, '2.5Fin':6, '2.5Unf':7})

Similarly, converting the other ordinal variables too.

We have cleaned our data. We can now move to data modelling.

**PREDICTIVE MODELLING**

X=df.drop(['SalePrice'], axis=1)

y=df['SalePrice']

I am splitting the data for validation. 70% for training and 30% for testing.

X\_train, X\_test,y\_train,y\_test = train\_test\_split(X,y, test\_size = 0.3, random\_state=3)

**Checking Mean Absolute Error of the Models.**

This tells us that the average difference between the actual data value and the value predicted by the model.

The lower the MAE for a given model, the more closely the model is able to predict the actual values.

The following are the Mean Absolute error values I have got.

|  |  |  |
| --- | --- | --- |
|  | Model | Mean Absolute Error |
| 1 | Linear regression Model | 19454.107746670077 |
| 2 | Random Forest Model | 27044.84857142857 |
| 3 | KNN Model | 53894.05714285714 |
| 4 | SVM Model | 34533.86571428571 |
| 5 | Naïve Bayes Model | 37224.83714285714 |
| 6 | Decision Tree Model | 39591.422857142854 |

**HYPER PARAMETER TUNING**

**---> Decision Tree Model**

GridSearchCV(cv=4, estimator=DecisionTreeClassifier(),

param\_grid={'max\_depth': [7, 15], 'max\_features': [30, 45],

'min\_samples\_split': [5, 10]},

scoring='neg\_mean\_absolute\_error')

**Best parameters:**

{'criterion': 'mae', 'max\_features': 79, 'min\_samples\_leaf': 7, 'min\_samples\_split': 5}

Decision Tree MAE after Hyper parameter tuning= 27254.204285714284

**---> KNN Model**

**Best parameters:**

{'algorithm': 'ball\_tree', 'leaf\_size': 1, 'n\_neighbors': 9, 'weights': 'distance'}

K-Nearest Neighbors MAE after Hyper parameter tuning = 31979.6794615044

**---> Random Forest Model**

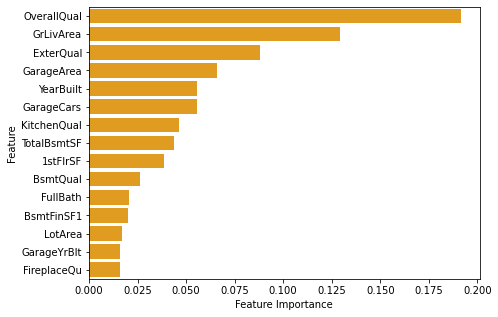
**Best parameters:**

{'n\_estimators': 600, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_features': 19, 'max\_depth': 254, 'criterion': 'mse', 'bootstrap': False}

Random Forest MAE after Hyper parameter tuning = 17517.426319047616

We can see that the mean absolute error (MAE) is less for Random Forest Model when

compared to other models.



**PREDICIONS**

|  |  |  |
| --- | --- | --- |
|  | Actual Data | Predicted Data |
| 355 | 256000 | 201305.475000 |
| 831 | 319000 | 296902.101667 |
| 984 | 195400 | 191507.910000 |
| 144 | 274000 | 266208.441667 |
| 411 | 131500 | 140648.771667 |

**SAVING THE MODEL**

I am saving the model as “housing\_price\_prediction.pkl” using joblib.

**Code**:

import joblib

joblib.dump(rf\_model, 'housing\_price\_prediction.pkl')

**CONCLUSION**

We built serveral regression models to predict the price of some house given some of the

house features. We evaluated and compared each model to determine the one with highest

performance. We also looked at how some models rank the features according to their

importance.

The model can be used also with datasets that cover different cities and areas provided that they contain the same features. We also suggest that people take into consideration the

features that were deemed as most important as seen in the previous section; this might

help them estimate the house price better.