House Price Prediction Using Python

> *In this project, we will develop and evaluate the performance and the predictive power of a model trained and tested on data collected from various sources.*

Problem Statement:

> 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 uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the CSV file below. The company is looking at prospective properties to buy houses to enter the market. You are required 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. For this company wants to know: • Which variables are important to predict the price of variable? • How do these variables describe the price of the house?

Importing necessary libraries :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns


import warnings
warnings.filterwarnings('ignore')
```

Importing dataset:

```
df_train =
pd.read_csv(f"C:\\Users\\lokes\\OneDrive\\Desktop\\Project-
Housing_splitted\\train.csv")


df_test =
pd.read_csv(f"C:\\Users\\lokes\\OneDrive\\Desktop\\Project-
Housing_splitted\\test.csv")
```

Exploratory Data Analysis (EDA):

```
print("shape of df_train",df_train.shape)
print("shape of df_test",df_test.shape)


#checking the shape of data


df_train.head()


#checking the 5 rows of data set


df_train.info()


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1168 entries, 0 to 1167
Data columns (total 81 columns):
 #   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
```

The data-set contains 1168 **rows and** 80 **features + the target variable (**SalePrice**)**. as observed there are many columns which contains the categorical data. Below I have listed the features with a short description:

MSSubClass: Identifies the type of dwelling involved in the sale.
MSZoning: Identifies the general zoning classification of the sale.
LotFrontage: Linear feet of street connected to property
LotArea: Lot size in square feet

Street: Type of road access to property
Alley: Type of alley access to property
LotShape: General shape of property
LandContour: Flatness of the property
Utilities: Type of utilities available
LotConfig: Lot configuration
LandSlope: Slope of property
Neighborhood: Physical locations within Ames city limits
Condition1: Proximity to various conditions
Condition2: Proximity to various conditions (if more than one is present)
BldgType: Type of dwelling
HouseStyle: Style of dwelling
OverallQual: Rates the overall material and finish of the house
OverallCond: Rates the overall condition of the house
YearBuilt: Original construction date
YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)
RoofStyle: Type of roof
RoofMatl: Roof material
Exterior1st: Exterior covering on house
Exterior2nd: Exterior covering on house (if more than one material)
MasVnrType: Masonry veneer type
MasVnrArea: Masonry veneer area in square feet
ExterQual: Evaluates the quality of the material on the exterior
ExterCond: Evaluates the present condition of the material on the exterior
Foundation: Type of foundation
BsmtQual: Evaluates the height of the basement
BsmtCond: Evaluates the general condition of the basement
BsmtExposure: Refers to walkout or garden level walls
BsmtFinType1: Rating of basement finished area
BsmtFinSF1: Type 1 finished square feet
BsmtFinType2: Rating of basement finished area (if multiple types)
BsmtFinSF2: Type 2 finished square feet
BsmtUnfSF: Unfinished square feet of basement area
TotalBsmtSF: Total square feet of basement area
Heating: Type of heating
HeatingQC: Heating quality and condition
CentralAir: Central air conditioning
Electrical: Electrical system

1stFlrSF: First Floor square feet
2ndFlrSF: Second floor square feet
LowQualFinSF: Low quality finished square feet (all floors)
GrLivArea: Above grade (ground) living area square feet
BsmtFullBath: Basement full bathrooms
BsmtHalfBath: Basement half bathrooms
FullBath: Full bathrooms above grade
HalfBath: Half baths above grade
Bedroom: Bedrooms above grade (does NOT include basement bedrooms)
Kitchen: Kitchens above grade
KitchenQual: Kitchen quality
TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
Functional: Home functionality (Assume typical unless deductions are warranted)
Fireplaces: Number of fireplaces
FireplaceQu: Fireplace quality
GarageType: Garage location
GarageYrBlt: Year garage was built
GarageFinish: Interior finish of the garage
GarageCars: Size of garage in car capacity
GarageArea: Size of garage in square feet
GarageQual: Garage quality
GarageCond: Garage condition
PavedDrive: Paved driveway
WoodDeckSF: Wood deck area in square feet
OpenPorchSF: Open porch area in square feet
EnclosedPorch: Enclosed porch area in square feet
3SsnPorch: Three season porch area in square feet
ScreenPorch: Screen porch area in square feet
PoolArea: Pool area in square feet
PoolQC: Pool quality
Fence: Fence quality
MiscFeature: Miscellaneous feature not covered in other categories
MiscVal: $Value of miscellaneous feature
MoSold: Month Sold (MM)
YrSold: Year Sold (YYYY)
SaleType: Type of sale
SaleCondition: Condition of sale

```
df_train.describe()

#checking the statistical records
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2
count	1168.000000	1168.000000	954.00000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1161.000000	1168.000000	1168.000000
mean	724.136130	56.767979	70.98847	10484.749144	6.104452	5.595890	1970.930651	1984.758562	102.310078	444.726027	46.647260
std	416.159877	41.940650	24.82875	8957.442311	1.390153	1.124343	30.145255	20.785185	182.595606	462.664785	163.520016
min	1.000000	20.000000	21.00000	1300.000000	1.000000	1.000000	1875.000000	1950.000000	0.000000	0.000000	0.000000
25%	360.500000	20.000000	60.00000	7621.500000	5.000000	5.000000	1954.000000	1966.000000	0.000000	0.000000	0.000000
50%	714.500000	50.000000	70.00000	9522.500000	6.000000	5.000000	1972.000000	1993.000000	0.000000	385.500000	0.000000
75%	1079.500000	70.000000	80.00000	11515.500000	7.000000	6.000000	2000.000000	2004.000000	160.000000	714.500000	0.000000
max	1460.000000	190.000000	313.00000	164660.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000

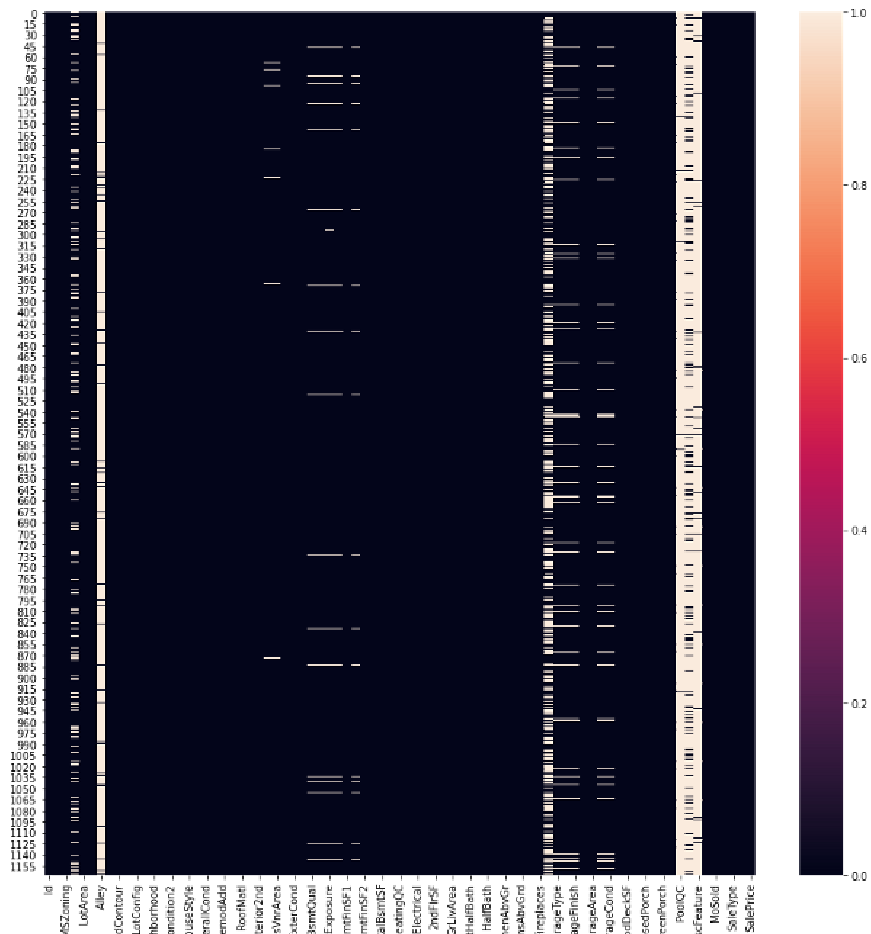
BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr
1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000
569.721747	1061.095034	1169.860445	348.826199	6.380137	1525.066781	0.425514	0.055651	1.562500	0.388699	2.884418
449.375525	442.272249	391.161983	439.696370	50.892844	528.042957	0.521615	0.236699	0.551882	0.504929	0.817229
0.000000	0.000000	334.000000	0.000000	0.000000	334.000000	0.000000	0.000000	0.000000	0.000000	0.000000
216.000000	799.000000	892.000000	0.000000	0.000000	1143.250000	0.000000	0.000000	1.000000	0.000000	2.000000
474.000000	1005.500000	1096.500000	0.000000	0.000000	1468.500000	0.000000	0.000000	2.000000	0.000000	3.000000
816.000000	1291.500000	1392.000000	729.000000	0.000000	1795.000000	1.000000	0.000000	2.000000	1.000000	3.000000
2336.000000	6110.000000	4692.000000	2065.000000	572.000000	5642.000000	3.000000	2.000000	3.000000	2.000000	8.000000

KitchenAbvGr	TotRmsAbvGrd	Fireplaces	GarageYrBlt	GarageCars	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch
1168.000000	1168.000000	1168.000000	1104.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000
1.045377	6.542808	0.617295	1978.193841	1.776541	476.860445	96.206336	46.559932	23.015411	3.639555	15.051370
0.216292	1.598484	0.650575	24.890704	0.745554	214.466769	126.158988	66.381023	63.191089	29.088867	55.080816
0.000000	2.000000	0.000000	1900.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	5.000000	0.000000	1961.000000	1.000000	338.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	6.000000	1.000000	1980.000000	2.000000	480.000000	0.000000	24.000000	0.000000	0.000000	0.000000
1.000000	7.000000	1.000000	2002.000000	2.000000	576.000000	171.000000	70.000000	0.000000	0.000000	0.000000
3.000000	14.000000	3.000000	2010.000000	4.000000	1418.000000	857.000000	547.000000	552.000000	508.000000	480.000000

As per above sort description we have seen in ‘count’ there are many null values in the dataset. And, we can also see ‘mean’ it shows variation among the features and values are on different scales so we have to scale the features in similar scale.

As we have checked statistical descriptions which shows only numeric data and dataset contains categorical values as well as. Therefore we have to check null values.

```
plt.figure(figsize=(15,15))
sns.heatmap(df_train.isnull())
```



> *Here, we are using heatmap and we can see the white lines show missing values in the data set.in addition there are few columns which are not contributing so we can drop them.*

Fill Missing Values:

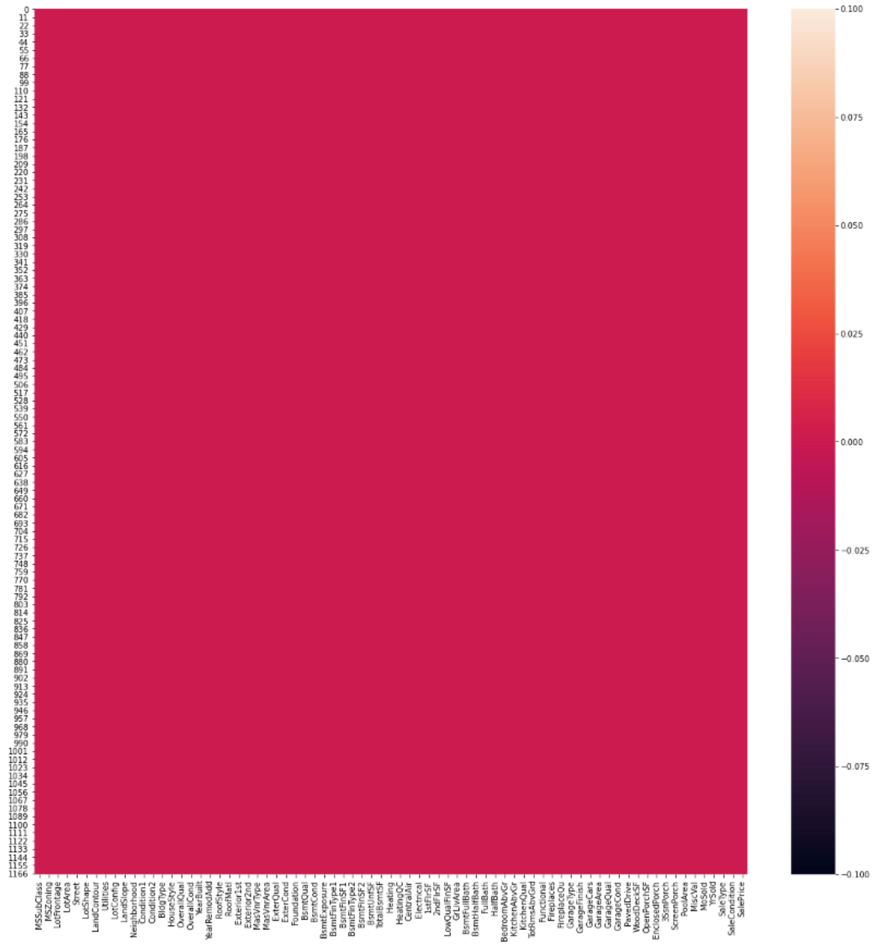
```
df_train['LotFrontage']=df_train['LotFrontage'].fillna(df_train['LotFrontage'].mean())
df_test['LotFrontage']=df_test['LotFrontage'].fillna(df_test['LotFrontage'].mean())


df_train['MasVnrArea']=df_train['MasVnrArea'].fillna(df_train['MasVnrArea'].mean())
df_test['MasVnrArea']=df_test['MasVnrArea'].fillna(df_test['MasVnrArea'].mean())




df_train['BsmtCond']=df_train['BsmtCond'].fillna(df_train['BsmtCond'].mode()[0])
df_test['BsmtQual']=df_test['BsmtQual'].fillna(df_test['BsmtQual'].mode()[0])




df_train['FireplaceQu']=df_train['FireplaceQu'].fillna(df_train['FireplaceQu'].mode()[0])
df_test['GarageType']=df_test['GarageType'].fillna(df_test['GarageType'].mode()[0])


df_train['GarageFinish']=df_train['GarageFinish'].fillna(df_train['GarageFinish'].mode()[0])
df_train['GarageQual']=df_train['GarageQual'].fillna(df_train['GarageQual'].mode()[0])
df_train['GarageCond']=df_train['GarageCond'].fillna(df_train['GarageCond'].mode()[0])
df_train['GarageType']=df_train['GarageType'].fillna(df_train['GarageType'].mode()[0])
df_train['BsmtQual']=df_train['BsmtQual'].fillna(df_train['BsmtQual'].mode()[0])
df_train['GarageType']=df_train['GarageType'].fillna(df_train['GarageType'].mode()[0])
df_train['BsmtFinType1']=df_train['BsmtFinType1'].fillna(df_train['BsmtFinType1'].mode()[0])


df_test['GarageFinish']=df_test['GarageFinish'].fillna(df_test['GarageFinish'].mode()[0])
df_test['GarageQual']=df_test['GarageQual'].fillna(df_test['GarageQual'].mode()[0])
df_test['GarageType']=df_test['GarageType'].fillna(df_test['GarageType'].mode()[0])
df_test['BsmtQual']=df_test['BsmtQual'].fillna(df_test['BsmtQual'].mode()[0])
df_test['GarageType']=df_test['GarageType'].fillna(df_test['GarageType'].mode()[0])
df_test['BsmtFinType1']=df_test['BsmtFinType1'].fillna(df_te
```

```
st['BsmtFinType1'].mode()[0])
df_test['GarageCond']=df_test['GarageCond'].fillna(df_test['
GarageCond'].mode()[0])
df_test['MasVnrType']=df_test['MasVnrType'].fillna(df_test['
MasVnrType'].mode()[0])
df_test['BsmtCond']=df_test['BsmtCond'].fillna(df_test['Bsmt
Cond'].mode()[0])
df_test['Electrical']=df_test['Electrical'].fillna(df_test['
Electrical'].mode()[0])
df_test['FireplaceQu']=df_test['FireplaceQu'].fillna(df_test
['FireplaceQu'].mode()[0])




df_train['MasVnrType']=df_train['MasVnrType'].fillna(df_trai
n['MasVnrType'].mode()[0])
df_test['MasVnrArea']=df_test['MasVnrArea'].fillna(df_test['
MasVnrArea'].mode()[0])


df_train['BsmtExposure']=df_train['BsmtExposure'].fillna(df_
train['BsmtExposure'].mode()[0])
df_test['BsmtExposure']=df_test['BsmtExposure'].fillna(df_te
st['BsmtExposure'].mode()[0])


df_train['BsmtFinType2']=df_train['BsmtFinType2'].fillna(df_
train['BsmtFinType2'].mode()[0])
df_test['BsmtFinType2']=df_test['BsmtFinType2'].fillna(df_te
st['BsmtFinType2'].mode()[0])
```

Droping the unrelavent columns:

```
df_train.drop(['Alley'],axis=1,inplace=True)
df_test.drop(['Alley'],axis=1,inplace=True)


df_train.drop(['GarageYrBlt'],axis=1,inplace=True)
df_test.drop(['GarageYrBlt'],axis=1,inplace=True)


df_train.drop(['PoolQC','Fence','MiscFeature'],axis=1,inplac
e=True)
df_test.drop(['PoolQC','Fence','MiscFeature'],axis=1,inplace
=True)


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



# checking the null values remove or not
```

```
plt.figure(figsize=(20,20))
sns.heatmap(df_train.isnull())
```



Here we can see null values has been removed.

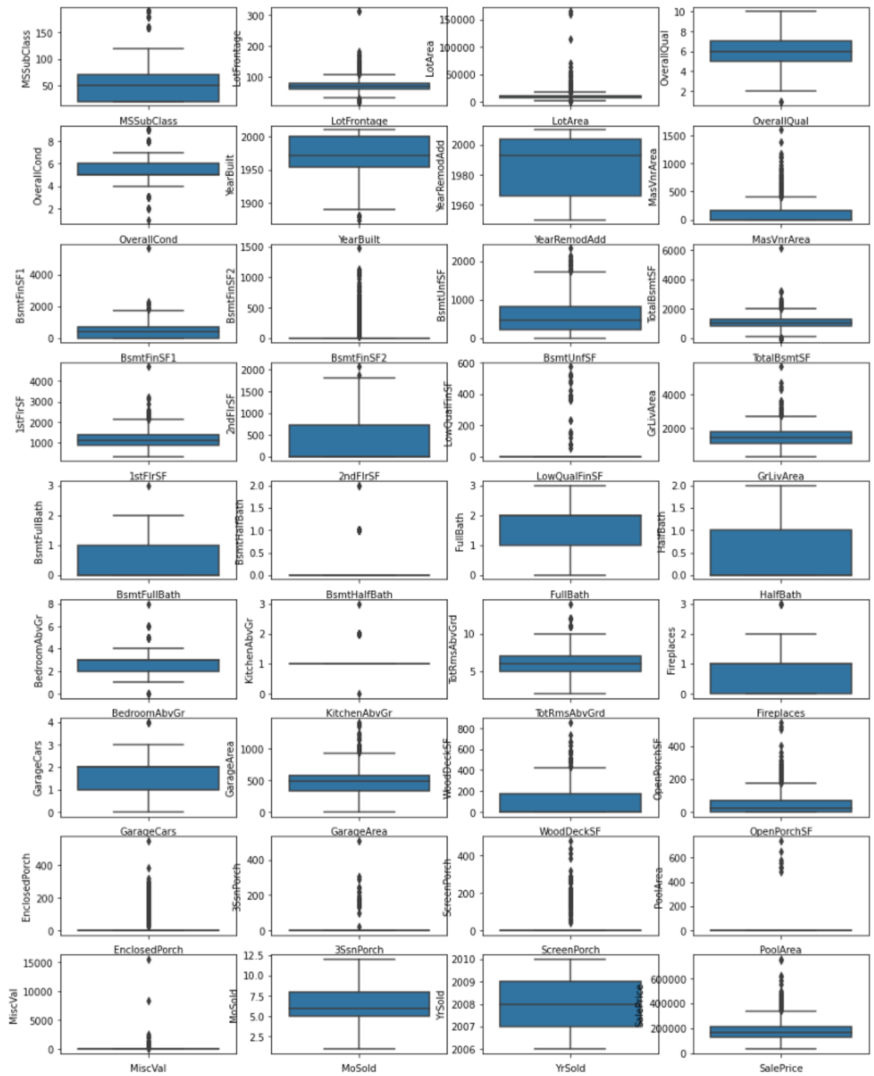
filltering the numeric values :

```
numeric_data = df_train.select_dtypes(include=
['int64','float64'])
numeric_data.columns


Index(['MSSubClass', 'LotFrontage', 'LotArea',
'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea',
'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
'FullBath', 'HalfBath',
```

```
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
'Fireplaces',
       'GarageCars', 'GarageArea', 'WoodDeckSF',
'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch',
'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

Visualization for outliers :

```
plt.figure(figsize=(15,20))
plotnumber = 1
for column in numeric_data:
    if plotnumber <=36:
        plt.subplot(9,4,plotnumber)
        sns.boxplot(numeric_data[column],orient='v')
        plt.xlabel(column,fontsize=10)
        plotnumber+=1
```



As per above we can see there are many columns which has outliers. hence we are using power transformation technique.

```
from sklearn.preprocessing import PowerTransformer
power = PowerTransformer(method='yeo-johnson')


df_train['MSSubClass'] =
power.fit_transform(df_train['MSSubClass'].values.reshape(-1
,1))
df_train['LotFrontage'] =
power.fit_transform(df_train['LotFrontage'].values.reshape(-
1,1))
df_train['LotArea'] =
power.fit_transform(df_train['LotArea'].values.reshape(-1,1)
)
df_train['OverallCond'] =
power.fit_transform(df_train['OverallCond'].values.reshape(-
1,1))
```

```
df_train['YearBuilt'] =
power.fit_transform(df_train['YearBuilt'].values.reshape(-1,
1))
df_train['MasVnrArea'] =
power.fit_transform(df_train['MasVnrArea'].values.reshape(-1
,1))
df_train['BsmtFinSF1'] =
power.fit_transform(df_train['BsmtFinSF1'].values.reshape(-1
,1))
df_train['BsmtFinSF2'] =
power.fit_transform(df_train['BsmtFinSF2'].values.reshape(-1
,1))
df_train['BsmtUnfSF'] =
power.fit_transform(df_train['BsmtUnfSF'].values.reshape(-1,
1))
df_train['TotalBsmtSF'] =
power.fit_transform(df_train['TotalBsmtSF'].values.reshape(-
1,1))
df_train['1stFlrSF'] =
power.fit_transform(df_train['1stFlrSF'].values.reshape(-1,1
))
df_train['LowQualFinSF'] =
power.fit_transform(df_train['LowQualFinSF'].values.reshape(
-1,1))
df_train['GrLivArea'] =
power.fit_transform(df_train['GrLivArea'].values.reshape(-1,
1))
df_train['BsmtHalfBath'] =
power.fit_transform(df_train['BsmtHalfBath'].values.reshape(
-1,1))
df_train['BedroomAbvGr'] =
power.fit_transform(df_train['BedroomAbvGr'].values.reshape(
-1,1))
df_train['KitchenAbvGr'] =
power.fit_transform(df_train['KitchenAbvGr'].values.reshape(
-1,1))
df_train['TotRmsAbvGrd'] =
power.fit_transform(df_train['TotRmsAbvGrd'].values.reshape(
-1,1))
df_train['GarageCars'] =
power.fit_transform(df_train['GarageCars'].values.reshape(-1
,1))
df_train['GarageArea'] =
power.fit_transform(df_train['GarageArea'].values.reshape(-1
,1))
df_train['WoodDeckSF'] =
power.fit_transform(df_train['WoodDeckSF'].values.reshape(-1
,1))
df_train['OpenPorchSF'] =
power.fit_transform(df_train['OpenPorchSF'].values.reshape(-
1,1))
df_train['EnclosedPorch'] =
power.fit_transform(df_train['EnclosedPorch'].values.reshape
(-1,1))
df_train['3SsnPorch'] =
power.fit_transform(df_train['3SsnPorch'].values.reshape(-1,
1))
df_train['ScreenPorch'] =
power.fit_transform(df_train['ScreenPorch'].values.reshape(-
1,1))
```

```
df_train['PoolArea'] =
power.fit_transform(df_train['PoolArea'].values.reshape(-1,1
))
df_train['MiscVal'] =
power.fit_transform(df_train['MiscVal'].values.reshape(-1,1)
)


df_test['LotFrontage'] =
power.fit_transform(df_test['LotFrontage'].values.reshape(-1
,1))
df_test['LotArea'] =
power.fit_transform(df_test['LotArea'].values.reshape(-1,1))
df_test['OverallCond'] =
power.fit_transform(df_test['OverallCond'].values.reshape(-1
,1))
df_test['YearBuilt'] =
power.fit_transform(df_test['YearBuilt'].values.reshape(-1,1
))
df_test['MasVnrArea'] =
power.fit_transform(df_test['MasVnrArea'].values.reshape(-1,
1))
df_test['BsmtFinSF1'] =
power.fit_transform(df_test['BsmtFinSF1'].values.reshape(-1,
1))
df_test['BsmtFinSF2'] =
power.fit_transform(df_test['BsmtFinSF2'].values.reshape(-1,
1))
df_test['BsmtUnfSF'] =
power.fit_transform(df_test['BsmtUnfSF'].values.reshape(-1,1
))
df_test['TotalBsmtSF'] =
power.fit_transform(df_test['TotalBsmtSF'].values.reshape(-1
,1))
df_test['1stFlrSF'] =
power.fit_transform(df_test['1stFlrSF'].values.reshape(-1,1)
)
df_test['LowQualFinSF'] =
power.fit_transform(df_test['LowQualFinSF'].values.reshape(-
1,1))
df_test['GrLivArea'] =
power.fit_transform(df_test['GrLivArea'].values.reshape(-1,1
))
df_test['BsmtHalfBath'] =
power.fit_transform(df_test['BsmtHalfBath'].values.reshape(-
1,1))
df_test['BedroomAbvGr'] =
power.fit_transform(df_test['BedroomAbvGr'].values.reshape(-
1,1))
df_test['KitchenAbvGr'] =
power.fit_transform(df_test['KitchenAbvGr'].values.reshape(-
1,1))
df_test['TotRmsAbvGrd'] =
power.fit_transform(df_test['TotRmsAbvGrd'].values.reshape(-
1,1))
df_test['GarageCars'] =
power.fit_transform(df_test['GarageCars'].values.reshape(-1,
1))
df_test['GarageArea'] =
power.fit_transform(df_test['GarageArea'].values.reshape(-1,
1))
```

```
df_test['WoodDeckSF'] =
power.fit_transform(df_test['WoodDeckSF'].values.reshape(-1,
1))
df_test['OpenPorchSF'] =
power.fit_transform(df_test['OpenPorchSF'].values.reshape(-1
,1))
df_test['EnclosedPorch'] =
power.fit_transform(df_test['EnclosedPorch'].values.reshape(
-1,1))
df_test['3SsnPorch'] =
power.fit_transform(df_test['3SsnPorch'].values.reshape(-1,1
))
df_test['ScreenPorch'] =
power.fit_transform(df_test['ScreenPorch'].values.reshape(-1
,1))
df_test['PoolArea'] =
power.fit_transform(df_test['PoolArea'].values.reshape(-1,1)
)
df_test['MiscVal'] =
power.fit_transform(df_test['MiscVal'].values.reshape(-1,1))
```

Dealing with categorical data:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df_train = df_train.apply(LabelEncoder().fit_transform)
df_train.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2
0	11	3	41	80	1	0	3	0	4	0	13	2	2
1	0	3	66	808	1	0	3	0	4	1	12	2	2
2	5	3	63	449	1	0	3	0	1	0	15	2	2
3	0	3	76	632	1	0	3	0	4	0	14	2	2
4	0	3	41	821	1	0	3	0	2	0	14	2	2

BldgType	HouseStyle	OverallQual	OverallCond	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Exterior1st	Exterior2nd	MasVnrType	MasVnrArea
4	2	5	4	75	26	1	1	8	9	2	0
0	2	7	5	69	20	0	5	12	13	2	0
0	5	6	4	95	47	1	1	7	7	2	0
0	2	5	5	76	27	3	1	8	9	1	237
0	2	5	6	76	50	1	1	4	4	3	74

ExterQual	ExterCond	Foundation	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinSF1	BsmtFinType2	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF
3	4	1	2	3	3	0	25	5	0	500	286
2	2	2	3	1	1	0	112	4	107	527	624
2	4	2	2	3	0	2	382	5	0	135	312
3	4	1	2	3	3	1	312	5	0	558	590
2	4	1	2	3	3	0	489	5	0	192	537

Model Building And Saving.

> Now we will train several Machine Learning models and compare their results. Note that because the dataset does not provide labels for their testing-set, we need to use the sklearn train test split and divide into test and train. Before that we have to divide into dependent and independent variables.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


X = df_train.drop(columns=['SalePrice'],axis=1)
y = df_train['SalePrice']


print("Shape of X", X.shape)
print("Shape of Y", y.shape)


Shape of X (1168, 74)
Shape of Y (1168,)


# Scaling the features using standardscaler
scaler = StandardScaler()


# Scaling the features using standardscaler
scaler = StandardScaler()


x_scale =  scaler.fit_transform(X)


X_train, X_test, y_train, y_test  =
train_test_split(x_scale, y, test_size=0.30,random_state=42)
```

Our data set divided into train and test. Now we will continue with model building.

```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
```

```
lr=LinearRegression()
knn=KNeighborsRegressor()
dt=DecisionTreeRegressor()
rf=RandomForestRegressor()
adb=AdaBoostRegressor()

print("Model is created")




lr.fit(X_train, y_train)
knn.fit(X_train,y_train)
dt.fit(X_train,y_train)
rf.fit(X_train,y_train)
adb.fit(X_train,y_train)

print("Model is trained")


print("lr_score",lr.score(X_train,y_train))
print("knn_score",knn.score(X_train,y_train))
print("dt_score",dt.score(X_train,y_train))
print("rf_score",rf.score(X_train,y_train))
print("adb_score",adb.score(X_train,y_train))
```

Model Evaluation:

```
from sklearn.metrics import
mean_absolute_error,mean_squared_error




lr_pred_y = lr.predict(X_test)
knn_pred_y = knn.predict(X_test)
dt_pred_y = dt.predict(X_test)
rf_pred_y = rf.predict(X_test)
adb_pred_y = adb.predict(X_test)


print("lr_score",mean_squared_error(y_test,lr_pred_y))
print("knn_score",mean_squared_error(y_test,knn_pred_y))
print("dt_score",mean_squared_error(y_test,dt_pred_y))
print("rf_score",mean_squared_error(y_test,rf_pred_y))
print("adb_score",mean_squared_error(y_test,adb_pred_y))


lr_score 2539.4564283154805
knn_score 4420.931737891738
dt_score 5497.119658119658
rf_score 2979.4789096866098
adb_score 3668.6127538802184
```

Cross validation:

```
from sklearn.model_selection import KFold,cross_val_score

k_f = KFold(n_splits=3,shuffle=True)
k_f

KFold(n_splits=3, random_state=None, shuffle=True)

print("Cross validation score for lr model","=>",cross_val_score(lr,X,y,cv=5))
print("Cross validation score for knn model","=>",cross_val_score(knn,X,y,cv=5))
print("Cross validation score for dt model","=>",cross_val_score(dt,X,y,cv=5))
print("Cross validation score for rf model","=>",cross_val_score(rf,X,y,cv=5))
print("Cross validation score for adb model","=>",cross_val_score(adb,X,y,cv=5))

Cross validation score for lr model => [0.9192724  0.87008538 0.84820237 0.90708106 0.89187413]
Cross validation score for knn model => [0.81521299 0.75622831 0.7686182  0.7725378  0.71334831]
Cross validation score for dt model => [0.76553021 0.70606012 0.68827123 0.78442901 0.72986909]
Cross validation score for rf model => [0.88510284 0.89335863 0.86435327 0.89511177 0.88456767]
Cross validation score for adb model => [0.85462362 0.85714421 0.83411719 0.84986722 0.82702801]

print("Cross validation score for lr model","=>",cross_val_score(lr,X,y,cv=5).mean())
print("Cross validation score for knn model","=>",cross_val_score(knn,X,y,cv=5).mean())
print("Cross validation score for dt model","=>",cross_val_score(dt,X,y,cv=5).mean())
print("Cross validation score for rf model","=>",cross_val_score(rf,X,y,cv=5).mean())
print("Cross validation score for adb model","=>",cross_val_score(adb,X,y,cv=5).mean())

Cross validation score for lr model => 0.887303067459977
Cross validation score for knn model => 0.7651891192339664
Cross validation score for dt model => 0.7353322831756455
Cross validation score for rf model => 0.8841972611635459
Cross validation score for adb model => 0.8441032220350648
```

As per cross validation i found decesion tree model will be good predictor for our problem.hence we will be trying to increase the chances of the accuracy using hyperparameter.

Hyperparameter Technique:

```
from sklearn.model_selection import GridSearchCV

dt.get_params().keys()
dict_keys(['ccp_alpha', 'criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'min_impurity_decrease', 'min_impurity_split', 'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'random_state', 'splitter'])
```

```
parm_grid = {'criterion':['mse','mae'],'splitter':['best'],
             'max_depth':[3,4,5,6],'min_samples_split':[2,5,6],
             'min_samples_leaf':[1,2,3,4,5,],'max_features':['auto','sqrt']}
parm_grid

{'criterion': ['mse', 'mae'],
 'splitter': ['best'],
 'max_depth': [3, 4, 5, 6],
 'min_samples_split': [2, 5, 6],
 'min_samples_leaf': [1, 2, 3, 4, 5],
 'max_features': ['auto', 'sqrt']}
```

```
gridsearch = GridSearchCV(dt, param_grid = parm_grid , cv=3 , verbose = 2 ,n_jobs =4)

gridsearch.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 240 candidates, totalling 720
fits


GridSearchCV(cv=3, estimator=DecisionTreeRegressor(),
n_jobs=4,
             param_grid={'criterion': ['mse', 'mae'],
'max_depth': [3, 4, 5, 6],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 3, 4,
5],
                         'min_samples_split': [2, 5, 6],
'splitter': ['best']},
             verbose=2)




gridsearch.best_params_


{'criterion': 'mse',
 'max_depth': 6,
 'max_features': 'auto',
 'min_samples_leaf': 5,
 'min_samples_split': 2,
 'splitter': 'best'}


dt1=DecisionTreeRegressor(criterion='mse',max_depth=5,max_fe
atures='auto',min_samples_leaf=1,min_samples_split=2,splitte
r='best')


dt1.fit(X_train, y_train)


DecisionTreeRegressor(max_depth=5, max_features='auto')


dt1.score(X_train, y_train)


dt_pred_y = dt1.predict(X_test)


print("dt_score",mean_squared_error(y_test,dt_pred_y))


dt_score 4940.020428351136
```

Model saving:

```
import pickle
```

```
HOUSEPRICE = 'HOUSEPRICE_model.pickle'


pickle.dump(dt1,open(HOUSEPRICE,'wb'))
```

Predicting test data using saved model

```
loaded_model = pickle.load(open(HOUSEPRICE, 'rb'))
loaded_model.predict(df_test)
```

```
array([236.        , 236.        , 236.        , 236.
,
       236.        , 470.65384615, 236.        , 236.
,
       236.        , 236.        , 393.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       247.66666667, 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        ,
387.67391304,
       236.        , 236.        , 236.        , 236.
,
       236.        , 276.10416667, 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       470.65384615, 236.        , 236.        ,
470.65384615,
       236.        , 236.        , 236.        , 236.
,
       470.65384615, 236.        , 236.        , 236.
,
       236.        , 470.65384615, 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
```

```
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 266.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 470.65384615, 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       470.65384615, 236.        , 236.        ,
247.66666667,
       236.        , 236.        , 236.        , 236.
,
       276.10416667, 266.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 276.10416667, 236.
,
       236.        , 236.        , 236.        ,
470.65384615,
       236.        , 236.        , 236.        , 236.
,
       236.        , 470.65384615, 247.66666667, 236.
,
       236.        , 247.66666667, 236.        , 236.
,
       236.        , 236.        , 236.        ,
470.65384615,
       247.66666667, 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        ,
247.66666667,
       236.        , 236.        , 236.        , 236.
```

```
,
       236.        , 470.65384615, 236.        ,
276.10416667,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       266.        , 236.        , 247.66666667, 236.
,
       236.        , 236.        , 247.66666667,
247.66666667,
       236.        , 266.        , 470.65384615, 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        ,
470.65384615,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        ,
470.65384615,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       247.66666667, 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 470.65384615, 236.
,
       236.        , 236.        , 387.67391304, 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
,
       236.        , 470.65384615, 236.        , 236.
,
       236.        , 236.        , 236.        , 236.
])
```

Concluding Remarks:

We started the our project to import various libraries and imported the dataset from GitHub. Observing the many important points like problem type and how many columns contains int ,float and object values. As per statistic observations we found huge variations among the features and we have used standard scaler to scale the variables. Besides this, we have identified there are many null values and few unwanted columns deleted such elements. During this process we used seaborn and matplotlib to do the visualizations and converted categorical features into numeric using label encoder pandas function. Afterwards we started training different different machine learning models, picked one of them (decision tree model) and applied cross validation on it and we tried to tune model using hyperparameter tuning.

To conclude, There are many other ways also to improve the model accuracy like doing a more extensive feature engineering, by comparing and plotting the features against each other and identifying and removing the noisy features, Along with resampling the data in case of imbalance or more extensive hyperparameter tuning on several machine learning models

> *Thanks*