Housing Price Prediction Project

Submitted By:
SYED MUGERA BILAL
syedbilalarmaan@gmail.com

ACKNOWLEDGMENT

Working on this project ***"Housing Price Prediction Project."*** Was a source of immense knowledge to me. We would like to express my sincere gratitude to FlipRobo Technologies for guidance and valuable support throughout this project work.

We acknowledge with a deep sense of gratitude, the encouragement and inspiration received from our SME/Mentor. We would also like to thank our parents for their love and support.

–SYED MUGERA BILAL

CONTENTS

Introduction:

Business Problem

- 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?

Business Goal

We are required to model the price of houses with the available independent variables. 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 and their format

The data is given by US-based housing company named Surprise Housing and they gave it to us in a CSV file, with data description file in pdf and txt format. They also had provided the problem statement by explaining what they need from us and also the required criteria to be satisfied.

Hardware, Software and Tools:

- For doing this project, we require laptop with high configuration and specification with a stable Internet connection.

- Microsoft office, Anaconda distribution as software.

- Python 3.x as programming language.

- Jupyter Notebook as Editor which is in Anaconda navigator.

- Some tools or libraries required like Numpy – used to numerical calculations. Pandas – used to data manipulation. Matplotlib and Seaborn – used to data visualization.

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')
```

```
In [3]: # Loading dataset
        data_train = pd.read_csv("train.csv")
```

Data Analysis:

```
In [5]: # Dimensions of dataset
        data_train.shape

Out[5]: (1168, 81)
```

The train dataset has 1168 rows and 81 columns.

```
In [12]: # Check missing values
         data_train.isnull().sum().sort_values(ascending=False).head(50)

Out[12]: PoolQC           1161
         MiscFeature      1124
         Alley            1091
         Fence             931
         FireplaceQu       551
         LotFrontage       214
         GarageYrBlt        64
         GarageFinish       64
         GarageType         64
         GarageQual         64
         GarageCond         64
         BsmtExposure       31
         BsmtFinType2       31
         BsmtQual           30
         BsmtCond           30
         BsmtFinType1       30
         MasVnrType          7
         MasVnrArea          7
         Id                  0
         Functional          0
         Fireplaces          0
         KitchenQual         0
         KitchenAbvGr        0
         BedroomAbvGr        0
         HalfBath            0
         FullBath            0
         BsmtHalfBath        0
         BsmtFullBath        0
         TotRmsAbvGrd        0
         GarageCars          0
         LowQualFinSF        0
         GarageArea          0
         PavedDrive          0
         WoodDeckSF          0
         OpenPorchSF         0
         EnclosedPorch       0
         3SsnPorch           0
         ScreenPorch         0
         PoolArea            0
         MiscVal             0
         MoSold              0
         YrSold              0
         SaleType            0
         SaleCondition       0
         GrLivArea           0
         HeatingQC           0
         2ndFlrSF            0
         LandSlope           0
         OverallQual         0
         HouseStyle          0
         dtype: int64
```

There are 1161 missing values in the column PoolQC, 1124 in MiscFeature, 1091 in Alley, 931 in Fence, 551 in FireplaceQu, 214 in LotFrontage, 64 each in GarageYrBlt, GarageFinish, GarageTyp, GarageQual, GarageCond, 31 in BsmtExposure and BsmtFinType2, 30 in BsmtQual, BsmtCond and BsmtFinType1, 7 in MasVnrType, MasVnrArea...present in dataset. We need to handle these null values.

Checking percentage of missing data

```
In [18]: # Checking the percentage of missing data
def missing_values_table(data_train):
    mis_val = data_train.isnull().sum()
    mis_val_percent = 100 * data_train.isnull().sum() / len(data_train)
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
    mis_val_table_ren_columns = mis_val_table.rename(
    columns = {0 : 'Missing Values', 1 : '% of Total Values'})
    mis_val_table_ren_columns = mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
    '% of Total Values', ascending=False).round(1)
    print ("Your selected dataframe has " + str(data_train.shape[1]) + " columns.\n"
        "There are " + str(mis_val_table_ren_columns.shape[0]) +
            " columns that have missing values.")
    return mis_val_table_ren_columns
missing_values_table(data_train)
```

```
Your selected dataframe has 77 columns.
There are 17 columns that have missing values.
```

Out[18]:

	Missing Values	% of Total Values
MiscFeature	1124	96.2
Alley	1091	93.4
Fence	931	79.7
FireplaceQu	551	47.2
LotFrontage	214	18.3
GarageType	64	5.5
GarageYrBlt	64	5.5
GarageFinish	64	5.5
GarageQual	64	5.5
GarageCond	64	5.5
BsmtExposure	31	2.7
BsmtFinType2	31	2.7
BsmtCond	30	2.6
BsmtFinType1	30	2.6
BsmtQual	30	2.6
MasVnrArea	7	0.6
MasVnrType	7	0.6

Data Cleaning

- ❖ Removing unnecessary features.

```
In [14]: # Dropping Utilities column
data_train.drop(['Utilities'], axis =1, inplace=True)
```

```
In [16]: # Dropping other unnecessary columns
data_train.drop('PoolQC', axis =1, inplace=True) # Contains same value in every row
data_train.drop('PoolArea', axis =1, inplace=True) # Contains same value in every row
data_train.drop('Id', axis =1, inplace=True) # id column not necessary for prediction
```

❖ Imputing the missing values.

```
In [20]: basement = ['BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2']
         #NA means No_Basement for all i in basement. Let's replace NAs with 'No_Basement'
         for i in basement:
             data_train[i].fillna('No_Basement',inplace=True)
             print(data_train[i].value_counts())

         #As per given definition, NA means None. Let's replace NAs with 'None'
         data_train['MiscFeature'].fillna('None',inplace=True)
         print(data_train['MiscFeature'].value_counts())

         #As per given definition, NA means No_alley_access. Let's replace missing data with 'No_alley_access'
         data_train['Alley'].fillna('No_alley_access',inplace=True)
         print(data_train['Alley'].value_counts())

         #As per given definition, NA means No_Fence. Let's replace missing data with 'No_Fence'
         data_train['Fence'].fillna('No_Fence',inplace=True)
         print(data_train['Fence'].value_counts())

         #NA means No_Fireplace. Let's replace missing data with 'No_Fireplace'
         data_train['FireplaceQu'].fillna('No_Fireplace',inplace=True)
         print(data_train['FireplaceQu'].value_counts())

         #Let's Impute the missing values and replace it with the median
         data_train['LotFrontage'].fillna(data_train['LotFrontage'].median(),inplace=True)

         garage=['GarageType','GarageFinish','GarageQual','GarageCond']
         #NA means No_Garage for all i in garage
         for i in garage:
             data_train[i].fillna('No_Garage',inplace=True)
             print(data_train[i].value_counts())

         #As per dataframe "df" we can say that most of the rows of GarageYrBlt has same value as YearBuilt so we replace with that
         data_train["GarageYrBlt"] = data_train["GarageYrBlt"].fillna(data_train["YearBuilt"])
         print(data_train['GarageYrBlt'].value_counts())

         #As per given values of MasVnrArea,  Let's replace missing data with 0's
         data_train['MasVnrArea'].fillna(0,inplace=True)
         print(data_train['MasVnrArea'].value_counts())

         #Let's fill the missing values in MasVnrType with None
         data_train['MasVnrType'] = data_train['MasVnrType'].fillna('None')
```

Exploratory Data Analysis (EDA):

Observations from univariate analysis:

- By looking at countplot of MSZoning ,which Identifies the general zoning classification of the sale,we find that 79 % of houses were sold in Low density resedential Areas.

- For street ,which states:Type of road access to property,we observe that almost 100% of house which were sold had access to paved roads so we can consider that no houses were purchased which had gravel road access.

- For Alley,93% of the purchased house do not have access to alley.Only 4% have gravel & 3% have paved alley.

- LotShape : 63% of the sold property was of Regular shape followed by slightly irregular type (33%).It means Australian gives priority to regular shaped houses.

- LandContour :90% of sold houses were neary flat level.

- LotConfig : 72% of purchased houses had Inside lot of the property.

- LandSlope :Around 95% of the sold property had gentle slope.

- Neighborhood: Physical locations within Ames city limits-:highest 16% of purhcased houses has neighbourhood of NWAmes(Northwest Ames) followed by CollgCr(College Creek) and least houses were purchased in neighbour hood of Bluestem.

- Condition1: Proximity to various conditions-:86% of purchased houses had normal proximity to various conditions1 and least 0.00 had RRne,RRNn proximity.

- Condition2: Proximity to various conditions (if more than one is present)-:99% of purchased houses had normal proximity to various conditions2.

- BldgType: Type of dwelling-:84% purchased houses were single family detached,followed by 8% 2FmCon(Two-family Conversion).

- HouseStyle: Style of dwelling-:49% houses had 1story followed by 2story style (31%)

- RoofStyle: Type of roof-:78% of houses have Gable roof style and 19% have Hip roof style.

- RoofMatl: Roof material-:98% houses have CompShg(Standard (Composite) Shingle) roof material.

- Exterior1st: Exterior covering on house-:34% houses have Vinylsiding covering on exteriors 15% have hard board and metal siding.

- Exterior2nd: Exterior covering on house (if more than one material)-:33% houses have VinylSd(Vinyl Siding) 15% have hard board and metal siding.

- MasVnrType: Masonry veneer type-:60% of houses have no masonry veneer type followed by BrkFace(Brick Face) (30%)

- ExterQual: Evaluates the quality of the material on the exterior-:61% of the sold hoUse have TA(Average/Typical) quality material on exterior followed by Gd(Good) 34%

- ExterCond: Evaluates the present condition of the material on the exterior-:88% houses are currently in TA(average) condition of exterior material.

- Foundation: Type of foundation-:44% houses have foundation CBlock(Cinder Block) & 44% have PConc(Poured Contrete)

- BsmtQual: Evaluates the height of the basement-:44% of houses have TA(typical) (80-89 inches) basement height followed by Gd(Good) (90-99 inches)

- BsmtCond: Evaluates the general condition of the basement-:89% of houses have TA(Typical - slight dampness allowed) basement.

- BsmtExposure: Refers to walkout or garden level walls-:64% of houses have No(No Exposure) followed by Av(Average Exposure) 15%

- BsmtFinType1: Rating of basement finished area-:(30%) have Unf(Unfinshed) basement area and 28% comes under GLQ(good living quarters)

- Heating: Type of heating-:98% houses have GasA(Gas forced warm air furnace) heating type.

- HeatingQC: Heating quality and condition-:30% houses have average quality heating.

- CentralAir: Central air conditioning-:93% houses are central air.

- Electrical: Electrical system-:92% houses have SbrKr(Standard Circuit Breakers & Romex) type of electrical systems.

- KitchenQual: Kitchen quality-:49% houses have average (TA) kitchen quality.

- Functional: Home functionality (Assume typical unless deductions are warranted)-:92% houses have typical (TA) home functionality.

- FireplaceQu: Fireplace quality-:47% of the houses donot have fireplace,25% houses have Gd(Good - Masonry Fireplace in main level) FireplaceQuality.

- GarageType: Garage location-:57% houses have Attached garage type,while 29% have Detchd(Detached from home).

- GarageFinish: Interior finish of the garage:42% of houses have unfinished garage while 29% have RFn(Rough Finished).

- GarageQual: Garage quality-:90% of houses have average garage quality.

- GarageCond: Garage condition-:91% of houses have TA(average garage condition).

- PavedDrive: Paved driveway-:92% of houses have Y(paved drive) way.

- Fence: Fence quality-:89% houses have NA(no fence).

- MiscFeature: Miscellaneous feature-:96% houses have no miscellaneous features.

- SaleType: Type of sale-:85% houses have sale type WD(warranty deed -conventional).

- SaleCondition:81% of houses are in normal sale condition

Observations from bivariate analysis:

- MSZoning:The avg sale price of the house is maximum in FV(Floating Village Residential) foloowed by RL(Residential Low Density) zone.

- Street:The property that have access to paved road have much higher average sale price as compared to that with gravel street.

- Alley:houses that do not have access to alley have higher sale price as compared to those with paved or gravel alley.

- LotShape:sale price is not much affected by lotshape,however IR2(Moderately Irregular) have a bit higher price compared to other while Reg(Regular) have lowest avg sale price.

- LandContour:Flatness of the property-:HLS(Hillside - Significant slope from side to side) have maximum average sale price & Bnk(Banked - Quick and significant rise from street grade to building) have miimum average sale price.

- LandSlope: It doesn't affect the average sale price of house.

- Neighborhood:The houses that has a neighbourhood of NoRidge(Northridge) has the maximum sale price followed by that with a neighbourhood of NridgHt(Northridge Heights)

- Condition1:house that is RRAn(Adjacent to North-South Railroad) has hightest avg sale price followed by PosA(Adjacent to postive off-site feature) while houses that is Artery(Adjacent to arterial street) has a minimum average sale price.

- BldgType: Type of dwelling-:TwnhsE(Townhouse End Unit) & 1Fam(Single-family Detached) type house have hightset selling price.

- HouseStyle: Style of dwelling-:The average sale price of 2.5Fin(Two and one-half story) is maximum followed by 2Story(Two story). 1.5Unf(One and one-half story: 2nd level unfinished) have lowest avg selling price.

- RoofMatl: Roof material-:House with roof material WdShngl(Wood Shingles) have a very high average selling price,followed by that with roof of WdShake(Wood Shakes),while house with roof material Roll(Roll) have lowest sale price.

- Exterior1st: Exterior covering on house-:House with exterior covering of ImStucc(Imitation Stucco) have maximum selling price while that with exterior coverng of BrkComm(Brick Common) have minimum average selling price.

- ExterQual: Evaluates the quality of the material on the exterior-:Houses with exterior material of excellent quality have highest saelling price followed by that of gd(good) quality.

- KitchenQual: Kitchen quality-:Houses with Ex(Excellent) kitchen quality have higher sale price while that with Fa(Fair) kitchen quality of lower selling price.

Observations from multivariate analysis:

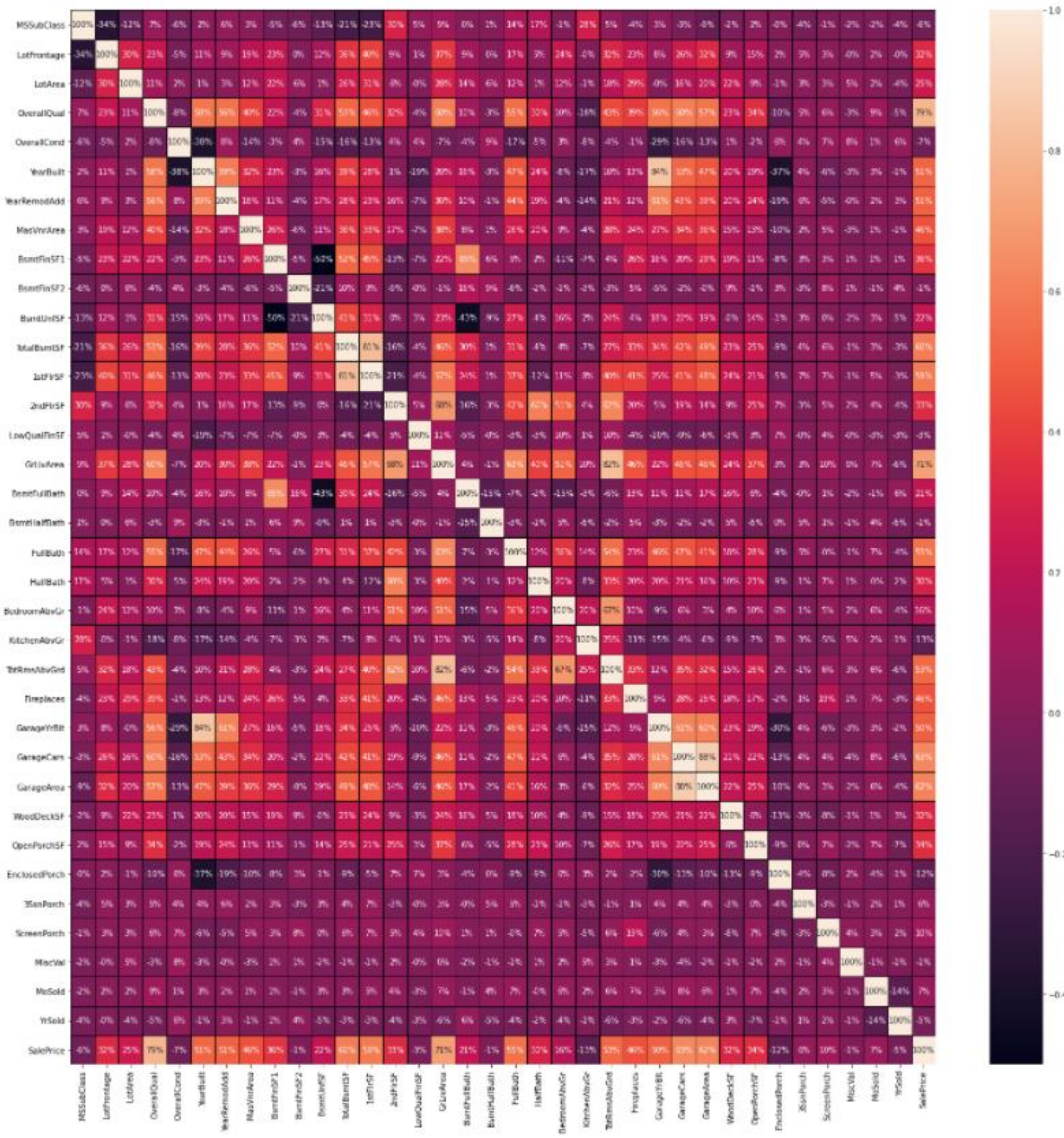
- Maximum standard deviation of 8957.44 is observed in LotArea column.

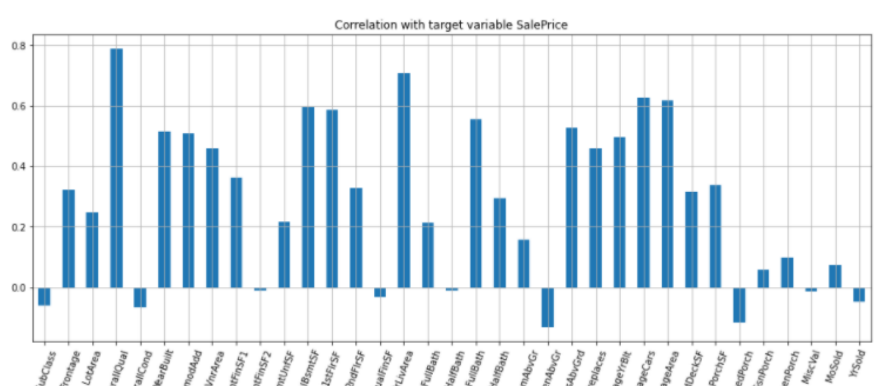
- Maximum SalePrice of a house observed is 755000 and minimum is 34900.

- In the columns MSSubclass, LotArea, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfsF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtFullBath, HalfBath, TotRmsAbvGrd, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, Miscval, salePrice mean is considerably greater than median so the columns are positively skewed.

- In the columns FullBath, BedroomAbvGr, Fireplaces, Garagecars, GarageArea, YrSold Median is greater than mean so the columns are negatively skewed.

Heatmap




Correlation with target variable SalePrice
0.8
0.6
0.4
0.2
0.0
MSSubClass
LotFrontage
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
MiscVal
MoSold
YrSold

Checking for outliers:

- Outliers are the data points that differ significantly from other observations. Any data points greater than +3 and -3 standard deviations are called as outliers.

- Z-score is the automated method used for handling outliers and it’s important to remove outliers as it impacts on the Accuracy of the Model.

- During the outlier’s analysis, we found that we are losing nearly 6.25% of data and it’s not a big loss.

```
In [54]: # Removing outliers
         from scipy.stats import zscore
         z_score = abs(zscore(data_train))
         print(data_train.shape)

         df_train = data_train.loc[(z_score<6).all(axis=1)]
         print(df_train.shape)

         (1168, 69)
         (1095, 69)

In [55]: # Data Loss
         dataloss = ((data_train.shape[0] - df_train.shape[0])/data_train.shape[0])*100
         dataloss

Out[55]: 6.25
```

Checking for skewness:

- Skewed data are not normally distributed; either they are positive skewed or negative skewed. If the data is skewed, it impacts on the accuracy of the model. So, it’s very important to remove the skewness for right and left skewed data by using transform methods like square and cube root, boxcox and logarithm transformation.

- For visualization, we use distplot to check the distribution of data points and the shape of the curve. Any value greater than 0.55 or less than -0.55 is considered to be skewed data.

- In our case, most of the data are skewed and hence we have to remove the skewness during Scaling because if we remove the skewness by log or boxcox method it will induce nan values. Sometimes while using root transforms, it can cause to form nan values. It is better to remove those values before scaling because it will show ValueError while running the code.

Standardization / Scaling:

As the values in the dataset have high ranges, it becomes complex for a ML model to understand and read the data, hence data training becomes difficult which is not a proper way to deal with data to achieve good accuracy and get accurate predictions. Therefore, it is very important to normalize/standardize data which means getting data within certain range to have proper understanding of data. In this project we have used StandardScaler() techniques to normalize the data which brings data between the range of 0 to 1.

In [62]:
```
# Scaling the dataset using StandardScaler
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(df_x)
x = pd.DataFrame(x, columns = df_x.columns)
x
```

Out[62]:

	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	...	GarageCond	PavedDrive	WoodDe
0	0.045740	0.074811	-1.267021	0.0	0.014247	-1.435050	0.275363	0.560583	-0.204609	0.147523	...	0.268178	0.270750	-0.9
1	0.045740	1.102980	0.160834	0.0	0.014247	-1.435050	0.275363	-0.726254	-0.204609	0.481624	...	0.268178	0.270750	0.9
2	0.045740	1.652949	0.588365	0.0	0.014247	-1.435050	0.275363	0.560583	-0.204609	0.314574	...	0.268178	0.270750	-0.9
3	0.045740	0.074811	1.587624	0.0	0.014247	-1.435050	0.275363	-0.193229	-0.204609	0.314574	...	0.268178	0.270750	1.2
4	0.045740	-0.555343	1.081204	0.0	0.014247	-1.435050	0.275363	0.560583	-0.204609	-0.687730	...	0.268178	0.270750	0.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1090	0.045740	0.074811	0.136146	0.0	0.014247	-1.435050	0.275363	0.560583	-0.204609	1.149826	...	0.268178	0.270750	-0.9
1091	0.045740	-0.077307	-0.126372	0.0	0.014247	0.718496	0.275363	0.560583	-0.204609	-0.854780	...	0.268178	-3.955457	-0.9
1092	0.045740	-2.835490	-2.357377	0.0	0.014247	0.718496	0.275363	-0.193229	-0.204609	0.147523	...	0.268178	0.270750	0.3
1093	-7.498944	-1.012502	-0.198747	0.0	3.914429	0.718496	0.275363	0.560583	-0.204609	-0.520679	...	-5.104820	-3.955457	-0.9
1094	0.045740	0.074811	-0.370343	0.0	0.014247	-1.435050	0.275363	0.560583	-0.204609	-0.687730	...	0.268178	0.270750	0.4

1095 rows × 68 columns

Principle Component Analysis (PCA):

An important machine learning method for dimensionality reduction is called Principal Component Analysis. It is a method that uses simple matrix operations from linear algebra and statistics to calculate a projection of the original data into the same number or fewer dimensions.

In [64]:
```
# PCA is required for the analysis to reduce curse of Dimensionality & at the same time minimizing information Loss
from sklearn.decomposition import PCA
for i in range (20, 50):
    pca = PCA(n_components=i)
    x_pca = pca.fit_transform(x)
    print(i, "Variance :{}".format(np.sum(pca.explained_variance_ratio_)))
```

```
20 Variance :0.6451956375959751
21 Variance :0.6601642735394087
22 Variance :0.6756229697477567
23 Variance :0.6894698944375471
24 Variance :0.7029831702253675
25 Variance :0.7170144377631933
26 Variance :0.7292746838959736
27 Variance :0.7431929682231497
28 Variance :0.7553461282317497
29 Variance :0.7673583789496558
30 Variance :0.7790131188332371
31 Variance :0.7906736281354999
32 Variance :0.8021612120454631
33 Variance :0.8126359079465703
34 Variance :0.8225721666724171
35 Variance :0.8326576138943128
36 Variance :0.8424875411102838
37 Variance :0.8520302890623318
38 Variance :0.8613929335879896
39 Variance :0.8703186638940696
40 Variance :0.8791143350115205
41 Variance :0.8876196987609931
42 Variance :0.8961581642917524
43 Variance :0.9038166189452442
44 Variance :0.9110275708233209
45 Variance :0.9179444891926768
46 Variance :0.9244108456868674
47 Variance :0.9308375709352044
48 Variance :0.936860782811821
49 Variance :0.9427454120585999
```

In [65]:
```
# As 49 has the highest value, we will select it.
pca = PCA(n_components=49)
x_pca = pca.fit_transform(x)
```

Splitting the data:

Using scikit learn library of python we have imported train test split to divide data into train data and test data. We have use test size is 0.2 (20%), rest of 0.8 (80%) data as train data. Random state provides us a seed to the random number generator by using following codes.

from sklearn.model_selection import train_train_split
x_train, x_test, y_train, y_test = train_train_split (x, y, test_size=0.2, random_state=85)

```
In [69]: # Creating train_test_split using best random_state
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=85)
```

Building Machine Learning Algorithms:

The following classifier algorithms we used are:

- Linear Regression, Lasso, Ridge, ElasticNet
- Decision Tree Regressor
- KNeighbors Regressor
- Support Vector Machine
- Random Forest Regressor, AdaBoost Regressor, Gradient Boosting Regressor

```
In [70]: # Importing the ML Algorithms
from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
```

```
In [71]: LR=LinearRegression()
l=Lasso()
en=ElasticNet()
rd=Ridge()
svr=SVR()
dtr=DecisionTreeRegressor()
knr=KNeighborsRegressor()
```

Evaluation of ML Models:

- As you can see , I had called the algorithms, then I called the empty list with the name models [ ], and calling all the model one by one and storing the result in that.

- We can observe that I imported the metrics to find the r2_score, mean_absolute_error (MAE), mean_squared_error (MSE), root_mean_squared_error (RMSE) in order to interpret the model's output. Then I also selected the model to find the cross_validation_score value.

```
In [73]: models= []
models.append(('Linear Regression',LR))
models.append(('Lasso Regression',l))
models.append(('Elastic Net Regression',en))
models.append(('Ridge Regression',rd))
models.append(('Support Vector Regressor',svr))
models.append(('Decision Tree Regressor',dtr))
models.append(('KNeighbors Regressor',knr))
```

As you can observe, I made a for loop and called all the algorithms one by one and appending their result to models.

Let me show the output so that we can glance the result in more appropriate way.

```
In [75]: # Finding the required metrices for all models together using a for loop
Model=[]
score=[]
cvs=[]
sd=[]
mae=[]
mse=[]
rmse=[]
for name,model in models:
    print('******************************',name,'***************************')
    print('\n')
    Model.append(name)
    model.fit(x_train,y_train)
    print(model)
    pre=model.predict(x_test)
    print('\n')
    AS=r2_score(y_test,pre)
    print('r2_score: ',AS)
    score.append(AS*100)
    print('\n')
    sc=cross_val_score(model,x,y,cv=5,scoring='r2').mean()
    print('cross_val_score: ',sc)
    cvs.append(sc*100)
    print('\n')
    std=cross_val_score(model,x,y,cv=5,scoring='r2').std()
    print('Standard Deviation: ',std)
    sd.append(std)
    print('\n')
    MAE=mean_absolute_error(y_test,pre)
    print('Mean Absolute Error: ',MAE)
    mae.append(MAE)
    print('\n')
    MSE=mean_squared_error(y_test,pre)
    print('Mean Squared Error: ',MSE)
    mse.append(MSE)
    print('\n')
    RMSE=np.sqrt(mean_squared_error(y_test,pre))
    print('Root Mean Squared Error: ',RMSE)
    rmse.append(RMSE)
    print('\n\n')
```

Result of ML Models:

- We saved result of ML Models in DataFrame.

```
In [76]: # Result stored in DataFrame
result=pd.DataFrame({'Model':Model, 'r2_score': score, 'Cross_val_score':cvs, 'Standard_deviation':sd,
                     'Mean_absolute_error':mae, 'Mean_squared_error':mse, 'Root_Mean_Squared_error':rmse})
result
```

Out[76]:

	Model	r2_score	Cross_val_score	Standard_deviation	Mean_absolute_error	Mean_squared_error	Root_Mean_Squared_error
0	Linear Regression	92.261355	-1.453855e+20	2.907710e+18	15293.505579	4.296199e+08	20727.273596
1	Lasso Regression	92.274787	8.743602e+01	1.747070e-02	15292.296023	4.288742e+08	20709.278653
2	Elastic Net Regression	91.387575	8.717424e+01	2.169517e-02	15333.467620	4.781288e+08	21866.156891
3	Ridge Regression	92.276365	8.745026e+01	1.750199e-02	15284.415405	4.287866e+08	20707.163478
4	Support Vector Regressor	-7.448199	-6.231802e+00	5.889473e-02	53841.366348	5.965112e+09	77234.136948
5	Decision Tree Regressor	78.172307	7.589358e+01	1.297640e-02	23784.917808	1.211790e+09	34810.771478
6	KNeighbors Regressor	87.761170	8.091807e+01	2.316573e-02	19072.355251	6.794529e+08	26066.317834

- We can see that Ridge and Lasso Regression algorithms are performing well, as compared to other algorithms. Now we will try Hyperparameter Tuning to find out the best parameters and try to increase their scores.

Hyperparameter Tuning:

Lasso

```
In [77]: from sklearn.model_selection import GridSearchCV
parameters={'alpha' :[0.001, 0.01, 0.1, 1], 'random_state':range(42, 100), 'selection':['cyclic','random']}
```

```
In [78]: # Using GridSearchCV to run the parameters and checking final r2_score
l=Lasso()
grid=GridSearchCV(l,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_)
print(grid.best_score_)

{'alpha': 1, 'random_state': 78, 'selection': 'random'}
0.8699734337258509
```

```
In [80]: # Using the best parameters obtained
l=Lasso(alpha=1, random_state=78, selection='random')
l.fit(x_train,y_train)
pred=l.predict(x_test)
print('Final r2_score after tuning is: ',r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(l,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(l,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))

Final r2_score after tuning is:  92.27461938500518
Cross validation score:  87.43613655181512
Standard deviation:  0.017473986682220097


Mean absolute error:  15291.75868717318
Mean squared error:  428883503.6394191
Root Mean squared error:  20709.50273761828
```

Ridge

In [81]:
```
# Creating parameter list to pass in GridSearchCV
parameters={'alpha' :[0.001, 0.01, 0.1, 1], 'random_state':range(42, 100), 'solver':['auto','lsqr','svd']}
```

In [82]:
```
# Using GridSearchCV to run the parameters and checking final r2_score
rd=Ridge()
grid=GridSearchCV(rd,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_)
print(grid.best_score_)
```

```
{'alpha': 1, 'random_state': 42, 'solver': 'auto'}
0.8701594055154434
```

In [83]:
```
# Using the best parameters obtained
rd=Ridge(alpha=1, random_state=42, solver='auto')
rd.fit(x_train,y_train)
pred=rd.predict(x_test)
print('Final r2_score after tuning is: ',r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(rd,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(rd,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
Final r2_score after tuning is:  92.27636454029935
Cross validation score:  87.45025791873428
Standard deviation:  0.017501991778444744


Mean absolute error:  15284.415405253565
Mean squared error:  428786619.3104439
Root Mean squared error:  20707.163478140697
```

After Tuning the best algorithms, we can see that Lasso and Ridge Regression has been decreased slightly. Now, we will try Ensemble techniques like Random Forest Regressor, AdaBoost Regressor and GradientBoosting Regressor to boost up our scores.

Random Forest Regressor

In [84]:
```
from sklearn.ensemble import RandomForestRegressor
rfr=RandomForestRegressor(random_state=85)
parameters={'n_estimators':[10,50,100,500]}
grid=GridSearchCV(rfr,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_)
print(grid.best_score_)
```

```
{'n_estimators': 500}
0.8635894913596076
```

In [85]:
```
# Using the best parameters obtained
RF=RandomForestRegressor(random_state=85, n_estimators=500)
RF.fit(x_train,y_train)
pred=RF.predict(x_test)
print('r2_score: ',r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(RF,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(RF,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
r2_score:  91.8578268362492
Cross validation score:  87.63658607667804
Standard deviation:  0.018058961265762747


Mean absolute error:  14716.83336986301
Mean squared error:  452022227.47838545
Root Mean squared error:  21260.814365362054
```

AdaBoost Regressor

In [86]:
```
from sklearn.ensemble import AdaBoostRegressor
adr=AdaBoostRegressor(random_state=85)
parameters={'n_estimators':[10,50,100,500,1000],'learning_rate':[0.001,0.01,0.1,1],'loss':['linear','square']}
grid=GridSearchCV(adr,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_)
print(grid.best_score_)
```

```
{'learning_rate': 0.1, 'loss': 'linear', 'n_estimators': 500}
0.8222094327821395
```

In [87]:
```
# Using the best parameters obtained
adr=AdaBoostRegressor(random_state=85, n_estimators=500, learning_rate=0.1, loss='linear')
adr.fit(x_train,y_train)
pred=adr.predict(x_test)
print("r2_score: ",r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(adr,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(adr,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
r2_score:  87.97323866559798
Cross validation score:  82.5300828529447
Standard deviation:  0.028301353110579495


Mean absolute error:  18680.316147195226
Mean squared error:  667679664.678489
Root Mean squared error:  25839.498150670206
```

Gradient Boosting Regressor

```
In [88]: from sklearn.ensemble import GradientBoostingRegressor
gbr=GradientBoostingRegressor(random_state=85)
parameters={'n_estimators':[10,50,100,500,1000]}
grid=GridSearchCV(gbr,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_)
print(grid.best_score_)
```

```
{'n_estimators': 1000}
0.894872662879014
```

```
In [89]: # Using the best parameters obtained
gbr=GradientBoostingRegressor(random_state=85, n_estimators=1000)
gbr.fit(x_train,y_train)
pred=gbr.predict(x_test)
print("r2_score: ",r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(gbr,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(gbr,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
r2_score:  91.86923130369165
Cross validation score:  89.42774619443226
Standard deviation:  0.01191632240611046


Mean absolute error:  14845.693132790902
Mean squared error:  451389095.18398887
Root Mean squared error:  21245.919494905105
```

Key Metrics for success in solving problem under consideration

- r2_score
- Mean Absolute Error (MAE)
- Mean Square Error (MSE)
- Root Mean Square Error (RMSE)
- Cross-validation
- Hyperparameter Tuning using GridSearchCV

After applying Ensemble Techniques, we can see that GradientBoostingRegressor is the best performing algorithm among all other algorithms as it is giving a r2_score of **91.86** and cross validation score of **89.42**. It has also the less amount of error values obtained. Lesser the RMSE score, the better the model.

Saving the model

In order to dump the model which we have developed so that we can use it to make predictions in future, we have saved or dumped the best model.

Finalizing the model

```
In [90]: gbr_prediction=gbr.predict(x)
         print('Predictions of GradientBoosting Regressor: ',gbr_prediction)

Predictions of GradientBoosting Regressor:  [129774.0123627  229319.55616798 191039.6212634  ... 149066.80511538
  41106.40312179 185120.24999403]
```

Saving the model

```
In [91]: import pickle
         filename='Housing_Price_Prediction.pkl'
         pickle.dump(gbr,open(filename,'wb'))
```

Conclusion:

- After getting an insight of this dataset, we were able to understand that the Housing prices are done on basis of different features.

- First, we loaded the train dataset and did the EDA process and other pre-processing techniques like skewness check and removal, handling the outliers present, filling the missing data, visualizing the distribution of data, etc.

- Then we did the model training, building the model and finding out the best model on the basis of different metrices scores we got like Mean Absolute Error, Mean squared Error, Root Mean Squared Error, etc.

- We got Lasso Regressor as the best algorithm among all as it gave more r2_score and cross_val_score. Then for finding out the best parameter and improving the scores, we performed Hyperparameter Tuning.

- As the scores were not increased, we also tried using Ensemble Techniques like RandomForestRegressor, AdaBoostRegressor and GradientBoostingRegressor algorithms for boosting up our scores. Finally, we concluded that GradientBoostingRegressor was the best performing algorithm, although there were more errors in it and it had less RMSE compared to other algorithms. It gave an r2_score of **91.86** and cross_val_score of **89.42** which is the highest scores among all.

- We saved the model in a pickle with a filename in order to use whenever we require.

- We predicted the values obtained and saved it separately in a csv file.

- Then we used the test dataset and performed all the pre-processing pipeline methods to it.

- After treating skewness, we loaded the saved model that we obtained and did the predictions over the test data and then saving the predictions separately in a csv file.

- From this project, we learnt that how to handle train and test data separately and how to predict the values from them. This will be useful while we are working in a real-time case study as we can get any new data from the client we work on and we can proceed our analysis by loading the best model we obtained and start working on the analysis of the new data we have.

- The final result will be the predictions we get from the new data and saving it separately.

- Overall, we can say that this dataset is good for predicting the Housing prices using regression analysis and GradientBoostingRegressor is the best working algorithm model we obtained.

- We can improve the data by adding more features that are positively correlated with the target variable, having less outliers, normally distributed values, etc.