ASSIGNMENT REPORT [CA-1]

(***Project : House Price Prediction and EDA***)

Submitted by

(Mogalraj Kushal Dath)

Registration Number : 12100559

Course Name : Python for Machine Learning

Course Code : INT522

Section Number : K21ML

Roll Number : RK21MLA01

Date of Submission : 20-11-2021

Submitted to

(Dr. Dhanpratap Singh Sir)

School of Computer Science and Engineering

House Price Prediction and Exploratory Data Analysis

GitHub Link : **https://github.com/Kushal11608202/PRJ_CA1**

Abstract :

Nowadays house prices increment consistently, so there is a requirement to design a framework to understand and predict the house prices. House price prediction can help the developer to predict the selling cost of a house and can assist the client in organizing the ideal opportunity to buy a house. The common and main affecting factors on house price are current condition, area, time/year and location of the house.

Housing price patterns are not only the concern of purchasers and venders, however it likewise shows the current financial circumstance. Thus, it is important to predict housing prices without bias to help both the purchasers and venders settle on their decisions. This project utilizes an open source dataset. (Dataset Link)

ACKNOWLEDGEMENT :

I would like to express my special thanks of gratitude to my mentor Dr. Dhanpratap Singh Sir as well as other higher authorities along with Lovely Professional University who gave me this golden opportunity to do a wonderful project on the topic 'House Price Prediction', which also helped me in doing a lot of research and I came to know about so many new things. So, I am thankful to them and extremely privileged to have got all this to complete my project duly.

Introduction :

1.1 Description of the project :

Investment is a business activity that most people are interested in this globalization era. There are several objects that are often used for investment, for example, gold, stocks and property. Property investment has increased significantly since 2011, both on demand and property selling. At younger generation will need a house or buy a house in the future. Based on preliminary research conducted, there are two standards of house price which are valid in buying and selling transaction of a house that is house price based on the developer (market selling price) and price based on value of selling tax object.

The fundamental problem for a developer is to determine the selling price of a house. In determining the price of a house, the developer must calculate carefully and determine the appropriate method because property prices always increase continuously and almost never fall in the long term or short.

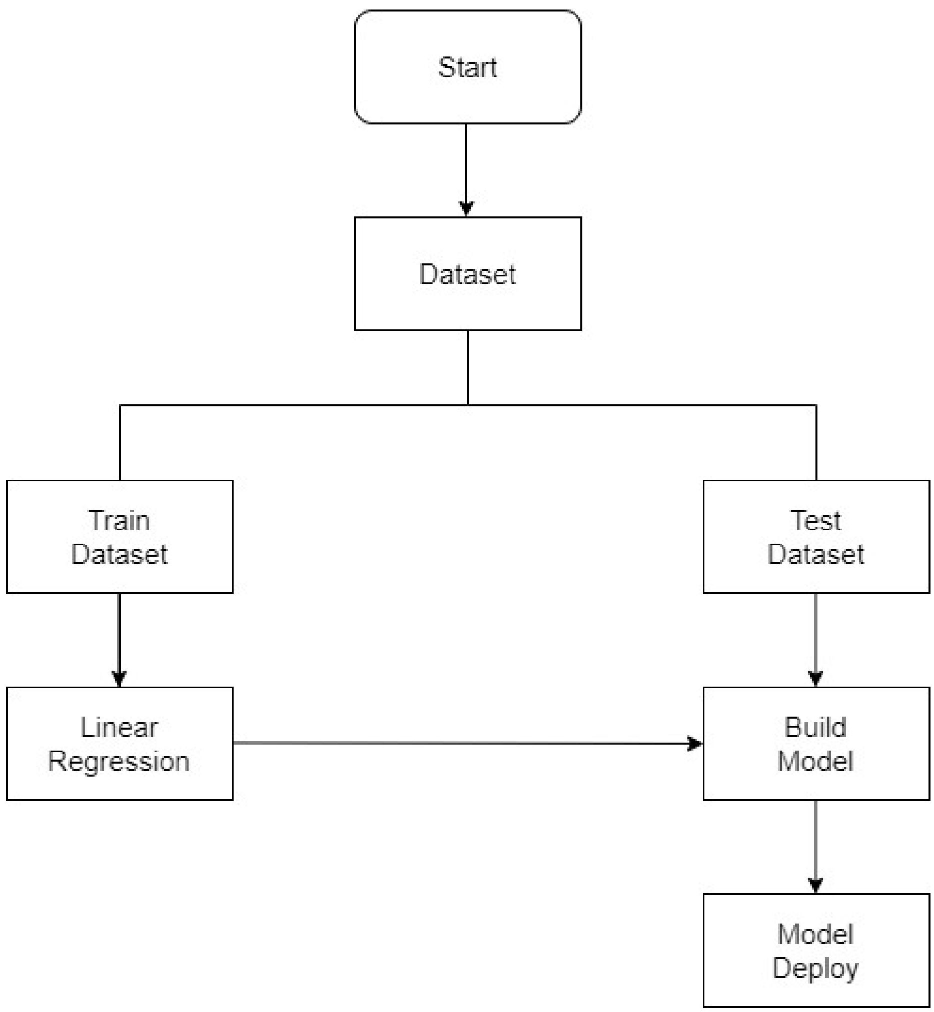
There are several approaches that can be used to do Exploratory data analysis in KC house dataset , one of it is matplotlib and the other is seaborn which are used for the data visualization or data representation in graphical form. And for Prediction of house prices , there is a basic approach called linear Regression will be utilized. To improve or boost the performance we are going to use gradient boosting regression in this project.

1.2 Limitations :

There is no guarantee that the data will be contains the exact list of features which affect the prediction of house price. Thus, there might be a risk if the project will be accomplished based only on the public dataset.

Moreover, this project will not cover all regression algorithms; instead, it is focused on the EDA and chosen algorithm, starting from the basic regression techniques (Linear Regression) to the advanced ones (Gradient Boosting Regression).

1.3 System Design :



Libraries :

1. **Numpy :**
   NumPy (Numerical Python) is a linear algebra library in Python. It is very useful for performing mathematical and logical operations on Arrays. It provides an abundance of useful features for operations on n-arrays and matrices in Python. It is the fundamental package for scientific computing with Python. As the whole project is based on whole complex stats ,we will use these fast calculations and provide results.

2. **Pandas :**
   Pandas is the most popular python library that is used for data analysis. We will provide highly optimized performance with back-end source code with the use of Pandas.

3. **Matplotlib :**
   Matplotlib tries to make easy things easy and hard things possible. We will generate plots, histograms, scatterplots, etc… to make our project more appealing and easier to understand.

4. **Seaborn :**
   We will use it for statistical data visualization as Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

5. **Scikit-learn :**
   It is a Python library is associated with NumPy and SciPy. It is considered as one of the best libraries for working with complex data. There are a lot of changes being made in this library. We will use it for cross validation feature, providing the ability to use more than one metric. Lots of training methods like logistics regression will be used to provide some little improvements.

Screenshots and some content based on project :

Screenshot 1 :

House Price Prediction

November 20, 2021

Kushal Dath

The main aim of this project is to predict the house prices in King Country, Seattle (USA). We have the KC Sales Dataset available here

About Dataset:

This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015.

The screenshot-1 shows about the aim of the project and dataset.

Screenshot 2 :

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing the dataset

In [2]:
```
df = pd.read_csv('kc_house_data.csv')
```

We'll take a look at first 5 rows of our dataset by using the head function.

In [3]:
```
df.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1965
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987

5 rows × 21 columns

Screenshot-2 shows about the importing the basic required libraries and importing the .csv format dataset to the object df as data frame.

It also presents the first five rows of our dataset using head() function.

Screenshot 3 :

About some attributes related to Area :

1. sqft_living -> the total house square footage of the house
2. sqft_basement -> size of the basement
3. sqft_above = sqft_living - sqft_basement
4. sqft_lot -> lot size of the house
5. sqft_living15 -> the average house square footage of the 15 closest houses
6. sqft_lot15 -> the average lot square footage of the 15 closest houses
7. Total Area = sqft_living + sqft_basement

Shape of Our dataset.

In [4]:
```
df.shape
```
Out[4]:
```
(21613, 21)
```

Size of our dataset.

In [5]:
```
df.size
```
Out[5]:
```
453873
```

Screenshot-3 shows about shape and size of our KC house dataset and some attributes information.

Screenshot 4 :

A statistical description of our dataset.

In [6]:
```
df.describe()
```
Out[6]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.00
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.65
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.17
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.00
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.00
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.00
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.00
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.00

Screenshot-4 shows about the statistical description of the dataset.

Screenshot 5 :

Info about attributes of the dataset.

```
In [7]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
```

Screenshot-5 shows about the information of the data attributes and luckily we don’t have any null values.

Screenshot 6 :

In this Dataset :

1. It has 21613 house information.
2. It has 21 feature.
3. Five features(price, bathrooms, floors,lat and long) are float64 type.
4. 15 features (id, bedrooms, sqft_living, sqft_lot, waterfront, view, condition, grade, sqft_above, sqft_basement, yr_built, 5. yr_renovated, zipcode, sqft_living15, sqft_lot15) are int64 type.
5. One feature (object) is object type.
6. There isn't null all feature.

Checking sum of null-values(if any) in each column

```
In [8]: df.isnull().sum()
```

```
Out[8]: id               0
        date             0
        price            0
        bedrooms         0
        bathrooms        0
        sqft_living      0
        sqft_lot         0
        floors           0
        waterfront       0
        view             0
        condition        0
        grade            0
        sqft_above       0
        sqft_basement    0
        yr_built         0
        yr_renovated     0
```

Screenshot-6 shows about the sum of null values of the dataset attributes in which this case we don't have any null values so the sum obviously going to be zero.

Screenshot 7 :

Cleaning the data

Dropping all the variables that are not necessary for the model

In [9]:
```
df.drop(['view','grade','yr_renovated','sqft_living15','sqft_lot15'],inplace=True, axis=1)
df.head()
```

Out[9]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	sqft_above	sqft_basement	yr_built	zipco
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	3	1180	0	1955	981
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	3	2170	400	1951	981
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	3	770	0	1933	980
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	5	1050	910	1965	981
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	3	1680	0	1987	980

Screenshot-7 shows that we are dropping the unnecessary attributes of the data i.e., removing the attributes which does not affect much on house price.

Screenshot 8 :

Exploratory Data Analysis (EDA) on the dataset and

Common affecting factors on the price of the houses

In [10]:
```
def des():
    sns.set_style('darkgrid')
    plt.figure(figsize=(10,6))

def info(t=None, x_lab=None, y_lab=None):    # info -> (title , x_label , y_label)
    plt.title(t)
    plt.xlabel(x_lab)
    plt.ylabel(y_lab)
    plt.legend()
```

Screenshot-8 shows that the starting point of Exploratory data analysis and some user-defined functions used in our project.

Screenshot 9 :

```
In [11]: des()
         plt.scatter(df.price,df.long)
         info("Price vs Longitude","Price","Longitude")
```

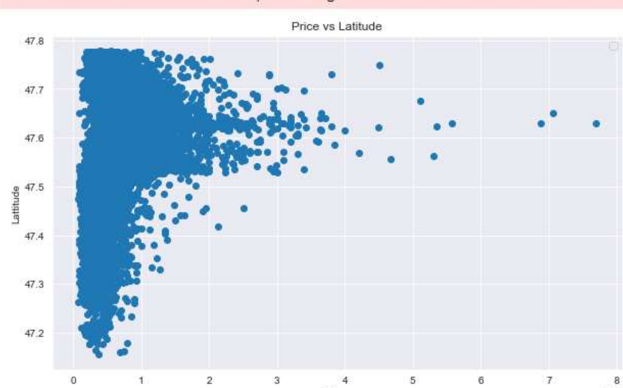
```
No handles with labels found to put in legend.
```

The plot that we used above is called scatter plot , scatter plot helps us to see how our data points are scattered and are usually used for two variables. The figure tells us about the location of the houses in terms of longitude and it gives us quite an interesting observation that -122.2 to -122.4 sells houses at much higher amount.

Screenshot 10 :

```
In [12]: des()
         plt.scatter(df.price,df.lat)
         info("Price vs Latitude","Price","Lattitude")
```

```
No handles with labels found to put in legend.
```

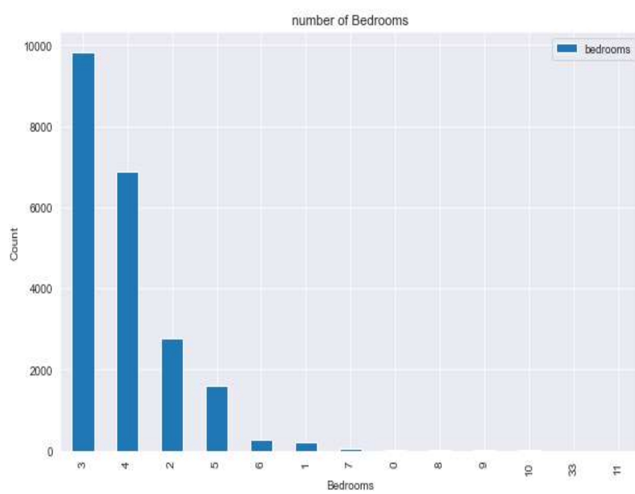


In Scr-10, The figure tells us about the location of the houses in terms of latitude and it gives us quite an interesting observation that 47.6 to 47.7 sells houses at much higher amount.

Let's see which is most common bedroom number. You may wonder why is it important ? Let's look at this problem from a builder's perspective, sometimes it's important for a builder to see which is the highest selling house type which enables the builder to make house based on that. Here in India , for a good locality a builder opts to make houses which are more than 3 bedrooms which attracts the higher middle class and upper-class section of the society.

Screenshot 11 :

```
In [13]: des()
         # sns.countplot(x='bedrooms', data=df, palette='viridis')
         df['bedrooms'].value_counts().plot(kind='bar')
         info('number of Bedrooms','Bedrooms','Count')
```

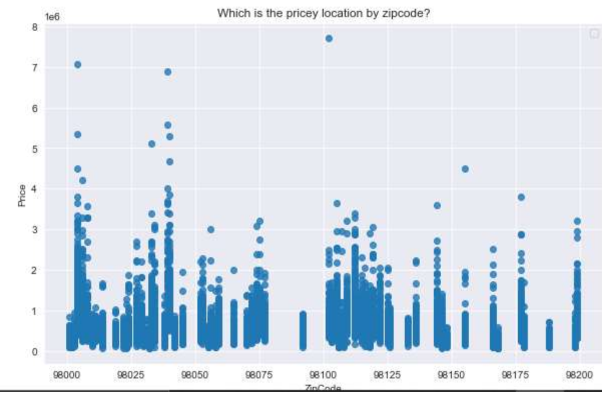


As we can see from the visualization 3 bedroom houses are most commonly sold followed by 4 bedroom. So how is it useful ? For a builder having this data , He can make a new building with more 3 and 4 bedroom's to attract more buyers.

Screenshot 12 :

```
In [14]: des()
         sns.regplot(df['zipcode'], df['price'], fit_reg = False)
         info('Which is the pricey location by zipcode?','ZipCode','Price')
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword ar
gs: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
No handles with labels found to put in legend.
```



Even location influencing the prices of the house. As we can see many houses are sold in between the zip code of 98100 and 96125.

Screenshot 13 :

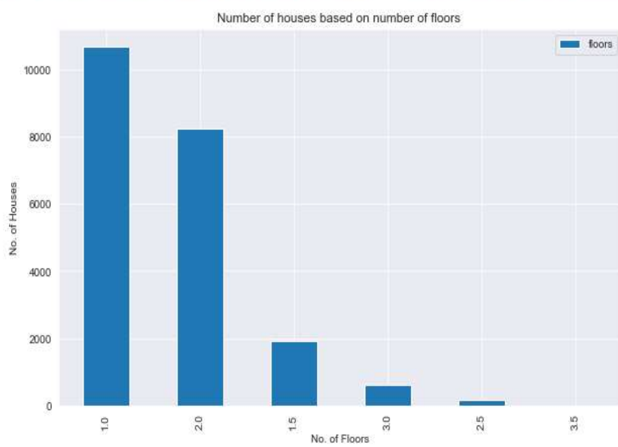
```
In [15]: des()
         fig = sns.FacetGrid(df, hue='waterfront', aspect=4)
         fig.map(sns.kdeplot, 'price', shade=True)   # kde - kernel density estimate
         x_max = df['price'].max()
         fig.set(xlim=(0,x_max))
         info('Distribution of Price Grouped by waterfront','price','density')
```

```
<Figure size 720x432 with 0 Axes>
```

The above graph is a regression plot which gives us graph based on probability density function which is bounded in a contiguous curve.

Screenshot 14 :

```
In [16]: des()
         df.floors.value_counts().plot(kind='bar')
         info("Number of houses based on number of floors ","No. of Floors","No. of Houses")
```



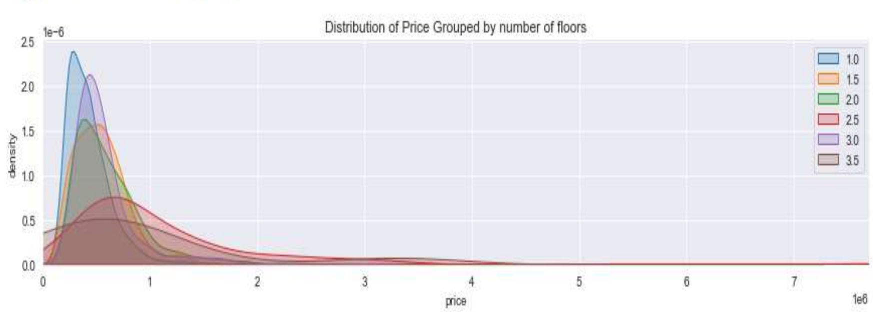
The above graph shows a count plot based on number of floors . As we can observe many I floor houses are sold compare to other.

We can see more factors affecting the price :

Screenshot 15 :

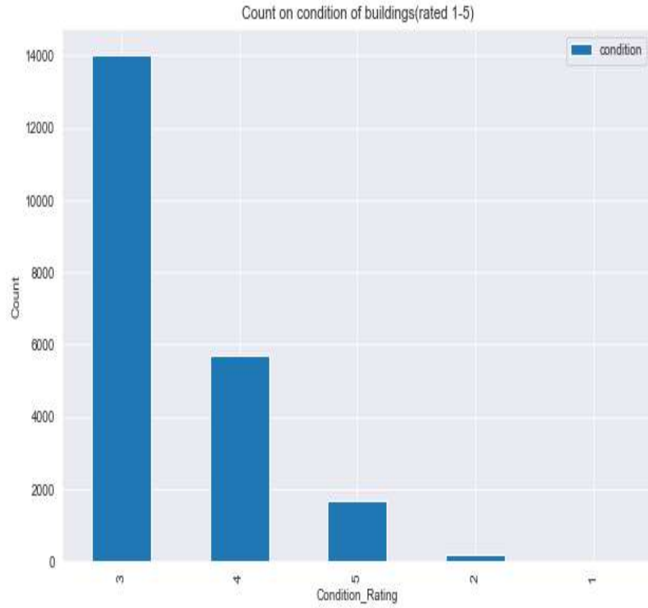
```
In [17]: des()
         fig = sns.FacetGrid(df, hue='floors', aspect=4)
         fig.map(sns.kdeplot, 'price', shade=True)
         x_max = df['price'].max()
         fig.set(xlim=(0,x_max))
         info('Distribution of Price Grouped by number of floors','price','density')
```

```
<Figure size 720x432 with 0 Axes>
```



Screenshot 16 :

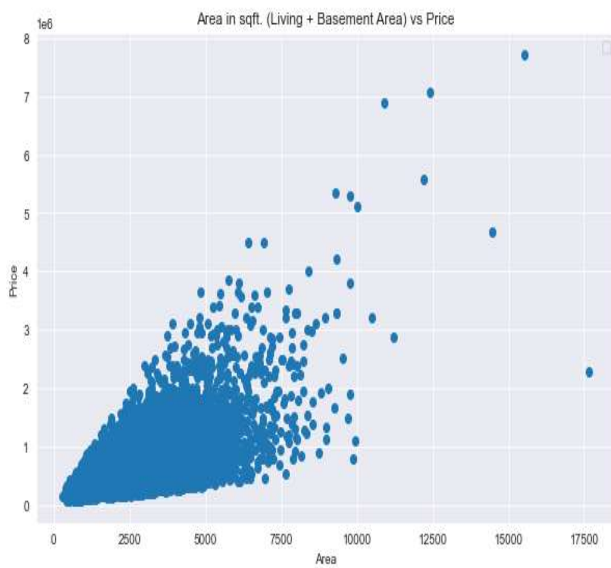
In [18]:
```
des()
df.condition.value_counts().plot(kind='bar')
info('Count on condition of buildings(rated 1-5)','Condition_Rating','Count')
```



Screenshot 17 :

In [19]:
```
des()
plt.scatter((df['sqft_living'] + df['sqft_basement']), df['price'])
info('Area in sqft. (Living + Basement Area) vs Price','Area','Price')
```

```
No handles with labels found to put in legend.
```



From the above figure we can see that more the Area , more the price though data is concentrated towards a particular price zone , but from the figure we can see that the data points seem to be in linear direction. Thanks to scatter plot we can also see some irregularities that the house with the highest square feet was sold for very less , maybe there is another factor or probably the data must be wrong.

Screenshot 18 :

More Affecting Factors on Price with their Pari-wise Correlation Coefficient in heatmap:

In [20]:
```
df1 = pd.DataFrame(df, columns = ['id','date','sqft_living','sqft_basement','condition','zipcode','price','lat','long'])
des()
sns.heatmap(df1.corr(),annot=True, fmt=".2f", linewidths=0.5)
plt.show()
```

The above figure shows about the correlation matrix os the essential factors on house price using heatmap.

Screenshot 19 :

Training a Linear Regression Model

Let's now begin to train out regression model! We will need to first split up our data into an X array that contains the features to train on, and a y array with the target variable, in this case the Price column. We will toss out the Address column because it only has text info that the linear regression model can't use.

X and y arrays

In [21]:
```
sum_columns = df['sqft_living'] + df['sqft_basement']
df['Area'] = sum_columns

New_dates = [1 if values==2014 else 0 for values in df.date]
df['date'] = New_dates

X = df.drop(['id','date','Area','condition','zipcode','price'],axis=1)
y = df['price']
```

Taking Training and Testing data for train test split.

Screenshot 20 :

Train Test Split

Now let's split the data into a training set and a testing set. We will train out model on the training set and then use the test set to evaluate the model.

In [22]:
```
from sklearn.model_selection import train_test_split
```

In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("X Train Shape", X_train.shape)
print("Y Train Shape", y_train.shape)
print("X Test Shape", X_test.shape)
print("Y Test Shape", y_test.shape)
```

```
X Train Shape (17290, 11)
Y Train Shape (17290,)
X Test Shape (4323, 11)
Y Test Shape (4323,)
```

Linear Regression :-

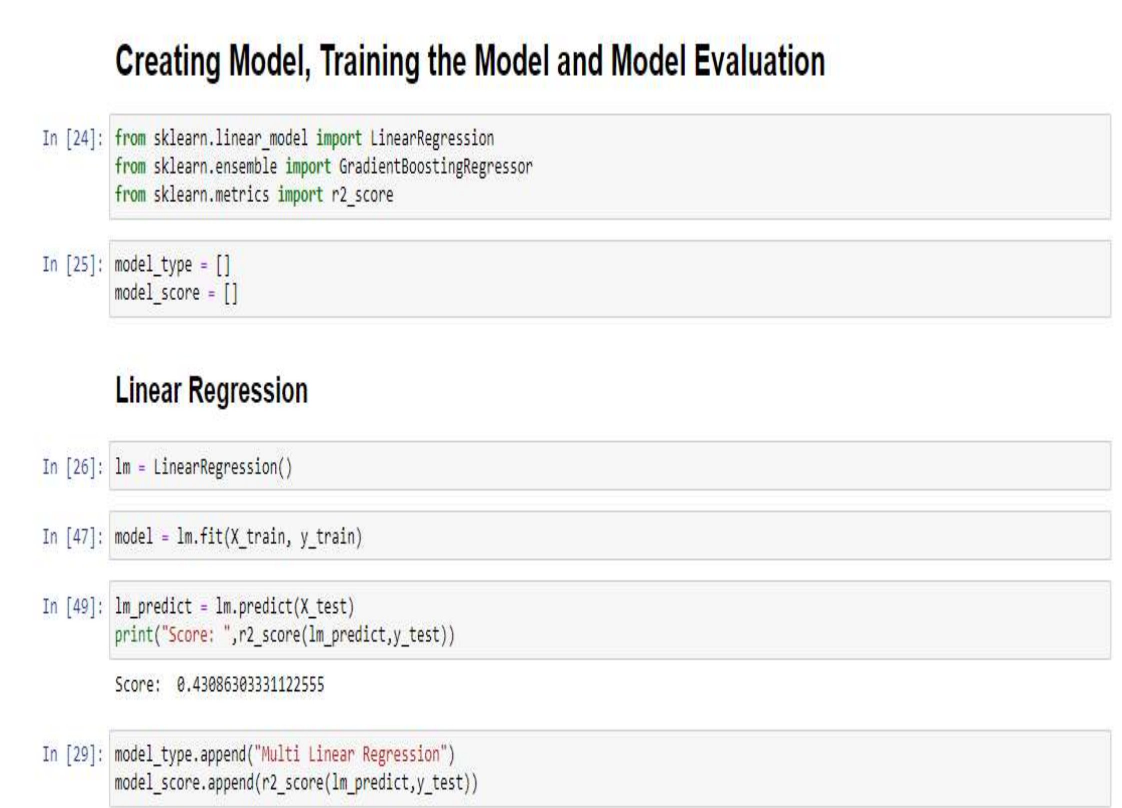
In easy words a model in statistics which helps us predicts the future based upon past relationship of variables. So when you see your scatter plot being having data points placed linearly you know regression can help you!

This Regression works on the line equation , y=mx+c , trend line is set through the data points to predict the outcome.

The variable we are predicting is called the criterion variable and is referred to as Y. The variable we are basing our predictions on is called the predictor variable and is referred to as X. When there is only one predictor variable, the prediction method is called Simple Regression. and if multiple predictor variable are present then multiple regression.

Let's look at the code ,

Screenshot 21 :



Creating Model, Training the Model and Model Evaluation

```
In [24]: from sklearn.linear_model import LinearRegression
         from sklearn.ensemble import GradientBoostingRegressor
         from sklearn.metrics import r2_score
```

```
In [25]: model_type = []
         model_score = []
```

Linear Regression

```
In [26]: lm = LinearRegression()
```

```
In [47]: model = lm.fit(X_train, y_train)
```

```
In [49]: lm_predict = lm.predict(X_test)
         print("Score: ",r2_score(lm_predict,y_test))

         Score:  0.43086303331122555
```

```
In [29]: model_type.append("Multi Linear Regression")
         model_score.append(r2_score(lm_predict,y_test))
```

So what did we do ? Let's go step by step :

1. We import our dependencies , for linear regression we use sklearn (built in python library) and import linear regression from it.
2. We then initialize Linear Regression to a variable reg. Now we know that prices are to be predicted , hence we set labels (output) as price columns and we also convert dates to 1's and 0's so that it doesn't influence our data much . We use 0 for houses which are new that is built after 2014.
3. We again import another dependency to split our data into train and test. I've made my train data as 80% and 20% of the data to be my test data , and randomized the splitting of data by using random_state.
4. So now , we have train data , test data and labels for both let us fit our train and test data into linear regression model.
5. After fitting our data to the model we can check the score of our data ie , prediction. in this case the prediction is 43%

Screenshot 22 :

Gradient Boosting Regressor

```
In [30]: gbr = GradientBoostingRegressor(n_estimators = 500, max_depth = 5, min_samples_split = 2,learning_rate = 0.1, loss = 'ls')

In [46]: gbr.fit(X_train,y_train)

Out[46]: GradientBoostingRegressor(max_depth=5, n_estimators=500)

In [48]: gbr_predict = gbr.predict(X_test)
         print("Score: ",r2_score(gbr_predict,y_test))

         Score:  0.8361913616518081

In [33]: model_type.append("Gradient Boosting Regression")
         model_score.append(r2_score(gbr_predict,y_test))
```

Linear Regression after Boosting

```
In [34]: print("Prediction Score of Linear Model after Boosting : ",lm.score(X_test, y_test))

         Prediction Score of Linear Model after Boosting :  0.643891588453672

In [35]: model_type.append("Multi Linear Regression After Boost")
         model_score.append(lm.score(X_test, y_test))
```

Gradient Boosting Regression :

For building a prediction model , many experts use gradient boosting regression , so what is gradient boosting? It is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

1. We first import the library from sklearn

2. We create a variable where we define our gradient boosting regressor and set parameters to it , here

   n_estimator → The number of boosting stages to perform. We should not set it too high which would overfit our model.

   max_depth → The depth of the tree node

   learning_rate → Rate of learning the data.

   loss → loss function to be optimized. 'ls' refers to least squares regression

   minimum sample split → Number of sample to be split for learning the data

3. We then fit our training data into the gradient boosting model and check for accuracy

4. We got an accuracy of 83.85% which is amazing!!!

Screenshot 23 :

Let's evaluate the model by checking out it's coefficients and how we can interpret them.

In [36]:
```
print(lm.intercept_)
```

```
-45633199.29638978
```

In [37]:
```
coeff_df = pd.DataFrame(lm.coef_, X.columns, columns=['Coefficient'])
coeff_df
```

Out[37]:

	Coefficient
bedrooms	-51065.194604
bathrooms	64682.508363
sqft_living	178.987474
sqft_lot	-0.071592
floors	8866.771570
waterfront	741638.793624
sqft_above	117.734120
sqft_basement	61.253354
yr_built	-2269.072229
lat	623942.839625
long	-166815.702072

Interpreting the coefficients:

OUTPUT of the Project :

Screenshot 24 :

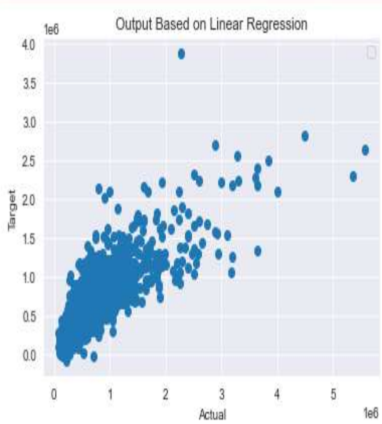
Predictions from our Model

Let's grab predictions off our test set and see how well it did!

In [38]:
```
predictions1 = lm.predict(X_test)
```

In [39]:
```
plt.scatter(y_test, predictions1)
info('Output Based on Linear Regression','Actual','Target')
```

```
No handles with labels found to put in legend.
```

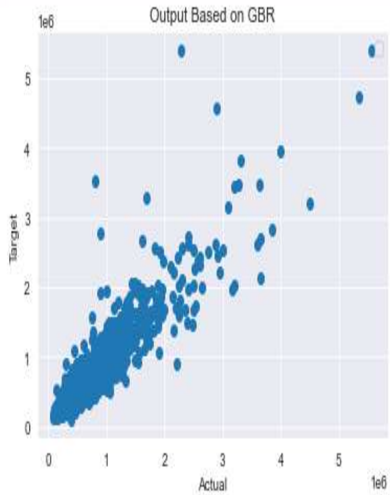


Screenshot 25 :

```
In [40]: predictions2 = gbr.predict(X_test)
```

```
In [41]: plt.scatter(y_test, predictions2)
         info('Output Based on GBR','Actual','Target')
```

```
No handles with labels found to put in legend.
```

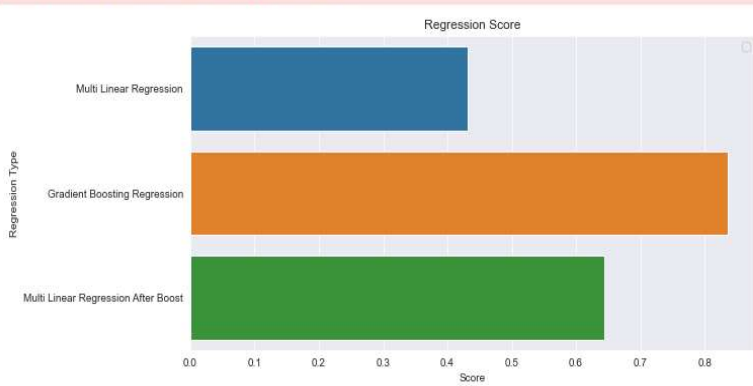


Comparing the Score of Linear Regression and GBR :

Screenshot 26 :

```
In [42]: plt.subplots(figsize=(10,5))
         sns.barplot(x = model_score , y = model_type)
         info('Regression Score','Score','Regression Type')
         plt.show()
```

```
No handles with labels found to put in legend.
```



Screenshot 27 :

Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

Screenshot 28 :

```
In [46]: from sklearn import metrics
```

```
In [47]: print('MAE:', metrics.mean_absolute_error(y_test, predictions2))
         print('MSE:', metrics.mean_squared_error(y_test, predictions2))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions2)))
```

```
MAE: 77650.58744892168
MSE: 22813055783.506916
RMSE: 151039.91453753845
```

```
In [48]: ids = df['id']
         predict = gbr.predict(df.drop(['id','date','Area','condition','zipcode','price'] , axis=1))

         #set the output as a dataframe and convert to csv file named submission.csv
         output = pd.DataFrame({ 'HOUSE_ID' : ids, 'PRICE': predict})
         output.to_csv('Output.csv', index=False)
```

Some Error metrics and Storing the predicted output to Output.csv file.

References :

https://ieeexplore.ieee.org/abstract/document/8473231

http://www.mecs-press.net/ijieeb/ijieeb-v12-n2/IJIEEB-V12-N2-3.pdf

https://www.tandfonline.com/doi/abs/10.1080/09599916.2020.1832558

https://ieeexplore.ieee.org/abstract/document/8882834