HOUSE PRICE PREDICTION MODEL USING REGRESSION TECHNIQUE

FINAL PROJECT REPORT

for

DATA MINING TECHNIQUES (ITE2006)

in

B.Tech (IT)

by

RITWIK RISHABH - 19BIT0203

Fall Semester, 2021-22

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

DECEMBER , 2021

TABLE OF CONTENTS

Data Mining Functionality and Platform used :-

Functionality used :-

PREDICTION : Supervised Learning method of Regression is applied to build a Predictive Data Mining model.

Platform used :-

KAGGLE : for dataset
JUPYTER NOTEBOOK : environment used for execution of code
PYTHON : language used for execution on Jupyter notebook

Dataset and its Description :-

kc_house_data :-

- The dataset consists of house prices from King County, an area in the US State of Washington.
- The dataset consisted of 21 features (including the class label price) and 21613 records.
- Kaggle link to the dataset :-
  https://www.kaggle.com/shivachandel/kc-house-data

Data Mining algorithms explored :-

- Linear Regression : Prediction was highly inaccurate on prediction based on single variables, so scrapped this technique.
- Multiple Linear Regression : Proved to be much accurate on prediction based on 20 features as compared to Linear Regression.

CODE SNAPSHOTS

Importing libraries and reading dataset :-

localhost:8888/notebooks/Desktop/DMT-J/House%20Price%20Prediction%20using%20Regression%20Techniques%20-%2019BIT0203.ip...

jupyter House Price Prediction using Regression Techniques - 19BIT0203 Last Checkpoint: 02/11/2021 (unsaved changes)

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

In [2]:
```python
#READING DATASET
data = pd.read_csv("kc_house_data.csv")
#READ BY 19BIT0203
```

In [3]:
```python
#DESCRIBING DATASET
data.describe()
```

Description and shape of dataset :-

In [3]:
```python
#DESCRIPTION OF DATASET BY (RITWIK RISHABH 19BIT0203)
data.describe()
```

Out[3]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	gr
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

In [4]:
```python
#NO. OF ROWS AND COLUMNS IN THE DATASET
data.shape
```

Out[4]: (21613, 21)

First 5 rows of dataset :-

In [5]:
```python
data.head()
```

Out[5]:

	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	0	1955
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170.0	400	1951
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770.0	0	1933
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050.0	910	1965
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680.0	0	1987

5 rows × 21 columns

Changing date to meaningful format and rounding no. of floors and bathrooms (some values are in decimal) :-

In [6]:
```
#CORRECTIONS IN DATE, BATHROOMS AND FLOORS ATTRIBUTES (19BIT0203)
data['date']=pd.to_datetime(data["date"])
data['bathrooms'] = np.round(data['bathrooms'])
data['floors'] = np.round(data['floors'])
data.head()
```

Out[6]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovate
0	7129300520	2014-10-13	221900.0	3	1.0	1180	5650	1.0	0	0	...	7	1180.0	0	1955	
1	6414100192	2014-12-09	538000.0	3	2.0	2570	7242	2.0	0	0	...	7	2170.0	400	1951	199
2	5631500400	2015-02-25	180000.0	2	1.0	770	10000	1.0	0	0	...	6	770.0	0	1933	
3	2487200875	2014-12-09	604000.0	4	3.0	1960	5000	1.0	0	0	...	7	1050.0	910	1965	
4	1954400510	2015-02-18	510000.0	3	2.0	1680	8080	1.0	0	0	...	8	1680.0	0	1987	

5 rows × 21 columns

Dataset information showing data types and no. of non-null values :-

In [7]:
```
#INFORMATION OF DATASET
data.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  datetime64[ns]
 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     21611 non-null  float64
 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: datetime64[ns](1), float64(6), int64(14)
memory usage: 3.5 MB
```

Finding no. of null values for each attribute and displaying the records with null values (NaN) :-

```
In [8]: #FINDING COUNT OF NULL VALUES OF EACH ATTRIBUTE
        data.isnull().sum()
```

```
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       2
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

```
In [9]: #DISPLAYING ROWS WITH NULL PRICE VALUE
        data[data['sqft_above'].isnull()]
```

Out[9]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovat
10	1736800520	2015-04-03	662500.0	3	2.0	3560	9796	1.0	0	0	...	8	NaN	1700	1965	
17	6865200140	2014-05-29	485000.0	4	1.0	1600	4300	2.0	0	0	...	7	NaN	0	1916	

Replacing null value with the mean of attribute, checking for duplicate records :-

```
In [10]: #REPLACING THE NULL VALUES WITH MEAN OF PRICE
         data['sqft_above'] = data['sqft_above'].fillna(data['sqft_above'].mean())
         data.iloc[[10,17]]
```

Out[10]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renova
10	1736800520	2015-04-03	662500.0	3	2.0	3560	9796	1.0	0	0	...	8	1788.396095	1700	1965	
17	6865200140	2014-05-29	485000.0	4	1.0	1600	4300	2.0	0	0	...	7	1788.396095	0	1916	

2 rows × 21 columns

```
In [11]: #CHECKING FOR ANY DUPLICATE RECORDS
         data[data.duplicated()]
```

Out[11]:

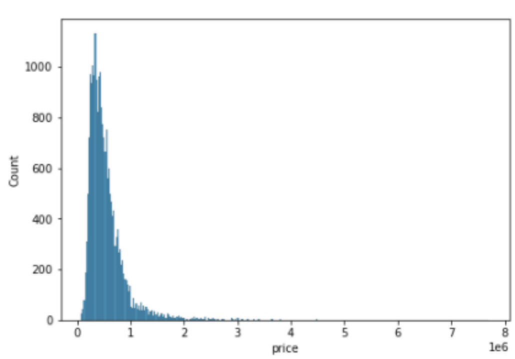
id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat

0 rows × 21 columns

Visualising histogram for price attribute, left skewness depicts presence of outliers :-

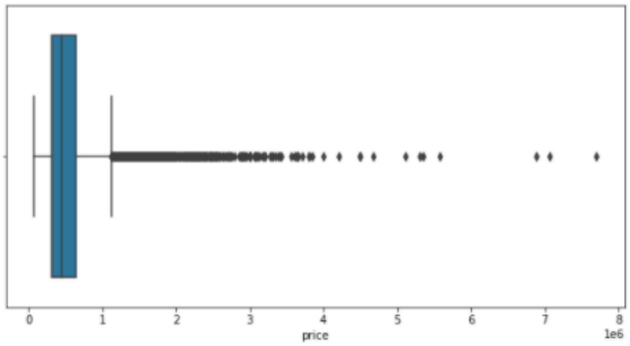
```
In [12]: #VISUALISING HISTOGRAM FOR CHECKING SKEWNESS (LEFT SKEWED)
         plt.figure(figsize=(16,5))
         plt.subplot(1,2,1)
         sns.histplot(data['price'])
```

```
Out[12]: <AxesSubplot:xlabel='price', ylabel='Count'>
```



Boxplot for price attribute (outliers visible), Box plot analysis - calculation of 1st quartile, 2nd quartile, InterQuartile range, minimum and maximum :-

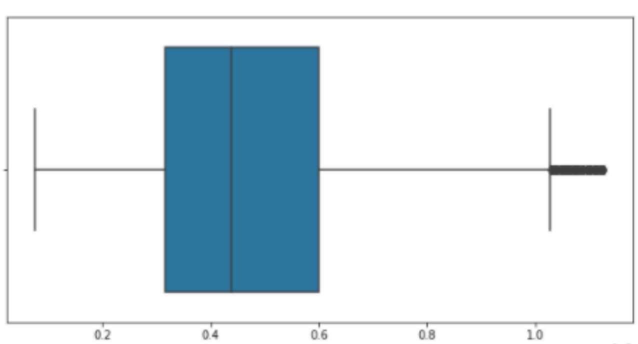
```
In [13]: #VISULATISING WHISKER PLOT/BOX PLOT TO CHECK FOR OUTLIERS
         fig=plt.gcf()
         fig.set_size_inches(10,5)
         sns.boxplot(x=data['price'])
Out[13]: <AxesSubplot:xlabel='price'>
```




```
In [14]: #BOX PLOT ANALYSIS
         Q1 = data['price'].quantile(0.25)
         Q3 = data['price'].quantile(0.75)
         IQR = Q3-Q1
         MIN = Q1-1.5*IQR
         MAX = Q3+1.5*IQR
         filter= (data['price']< MIN) | (data['price']>MAX)
         print('1st Quartile = ',Q1,'\n3rd Quatiile = ',Q3,'\nInter Quartile Range = ',IQR,'\nMinimum = ',MIN,'\nMaximum = ',MAX

         1st Quartile =  321950.0
         3rd Quatiile =  645000.0
         Inter Quartile Range =  323050.0
         Minimum =  -162625.0
         Maximum =  1129575.0
```

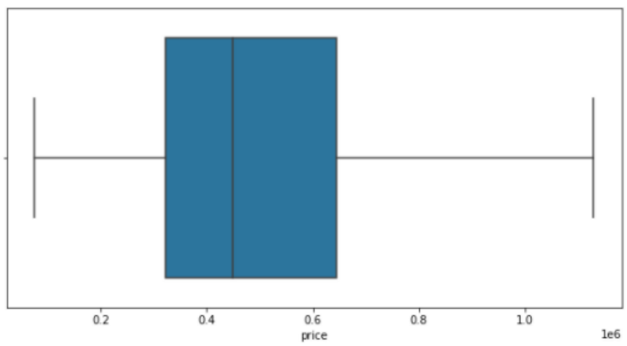
Displaying filtered 1146 outliers and Box plot after removal of outliers :-

```
In [15]: #DISPLAYING THE OUTLIERS
         data['price'][filter]
Out[15]: 5        1225000.0
         21       2000000.0
         49       1350000.0
         69       1325000.0
         125      1450000.0
                     ...
         21568    1700000.0
         21576    3567000.0
         21590    1222500.0
         21597    1575000.0
         21600    1537000.0
         Name: price, Length: 1146, dtype: float64

In [16]: #FILTERING OUT THE 1146 OUTLIERS IN THE DATA (19bit0203)
         data_clean = data[(data['price']<=MAX) & (data['price']>=MIN)]
         fig=plt.gcf()
         fig.set_size_inches(10,5)
         #DISPLAYING THE BOX PLOT AFTER REMOVAL OF OUTLIERS
         sns.boxplot(x=data_clean['price'])
Out[16]: <AxesSubplot:xlabel='price'>
```



Box plot after capping the remaining outliers and shape of data after removal of outliers (No. of rows reduced 20467 from 21613 after removing 1164 outliers) :-

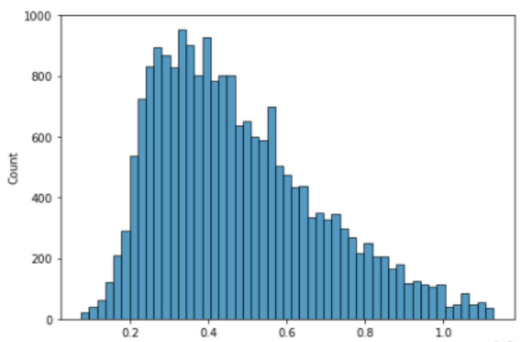
```
In [17]: #CAPPING THE OUTLIERS
         data_clean_cap = data.copy()
         data_clean_cap['price'] = np.where(data['price'] > MAX,MAX, np.where(data['price'] < MIN, MIN, data['price']))
         fig=plt.gcf()
         fig.set_size_inches(10,5)
         sns.boxplot(x=data_clean_cap['price'])
Out[17]: <AxesSubplot:xlabel='price'>
```




```
In [18]: #NUMBER OF ROWS REDUCED FROM 21613 TO 20467 AFTER OUTLIER REMOVAL
         data_clean.shape
Out[18]: (20467, 21)
```

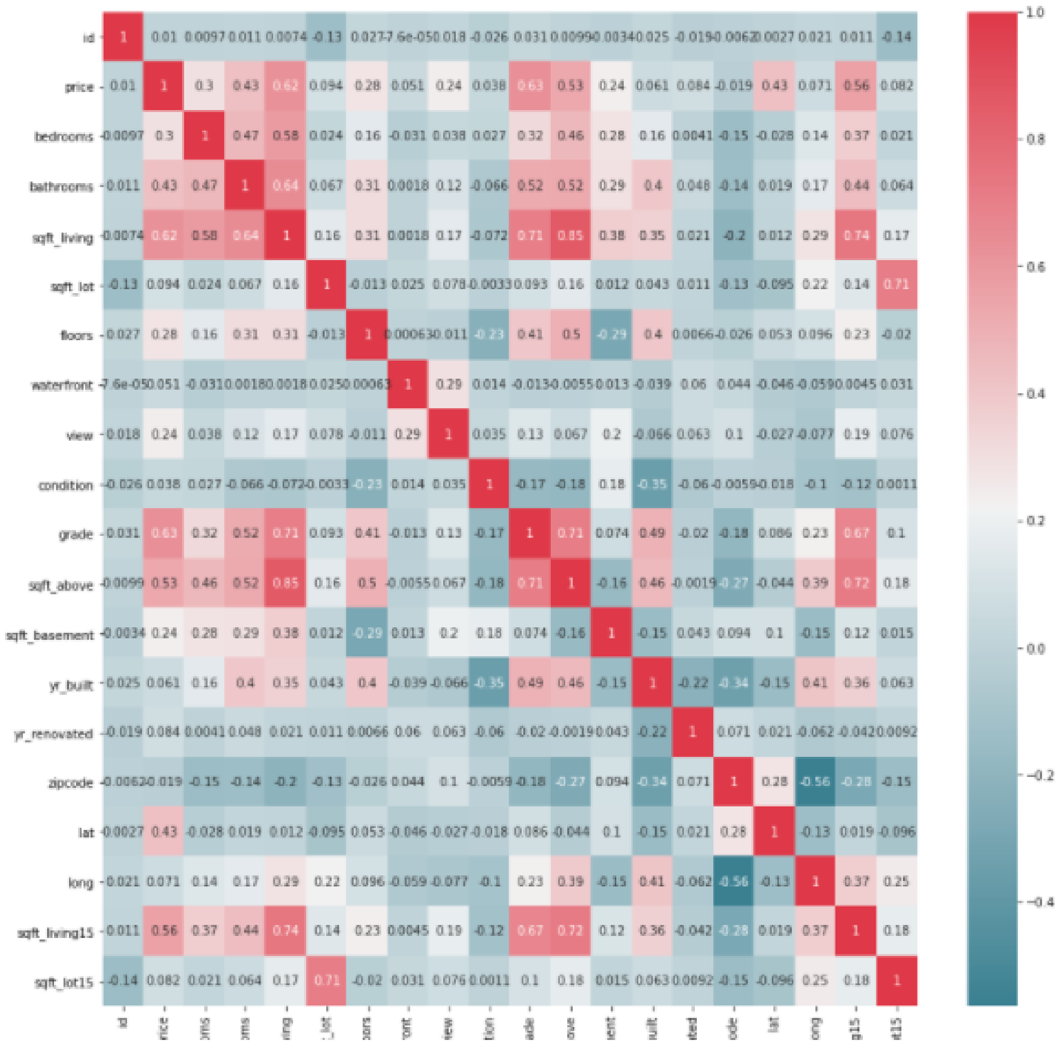
Histogram for price attribute after removal of outliers (Now somewhat centrally skewed from earlier histogram, depicting removal of outliers) :-

```
In [19]: plt.figure(figsize=(16,5))
         plt.subplot(1,2,1)
         sns.histplot(data_clean['price'])
Out[19]: <AxesSubplot:xlabel='price', ylabel='Count'>
```



Plotting of heatmap - depicts pearson's correlation coefficient of two attributes :-

```
In [21]: #CORRELATION ANALYSIS
         fig, ax = plt.subplots(figsize=(15,15))
         sns.heatmap(data_clean.corr(), annot=True, cmap=sns.diverging_palette(220, 10, as_cmap=True))

Out[21]: <AxesSubplot:>
```



Removal of one among two highly correlated features depicted by dark red and dark blue, Dropping "sqft_living" and "sqft_lot15" as they are highly correlated with similar attributes "sqft_living15" and "sqft_lot" respectively. "Id","zipcode" and "date" are also dropped as they are irrelevant for price prediction :-

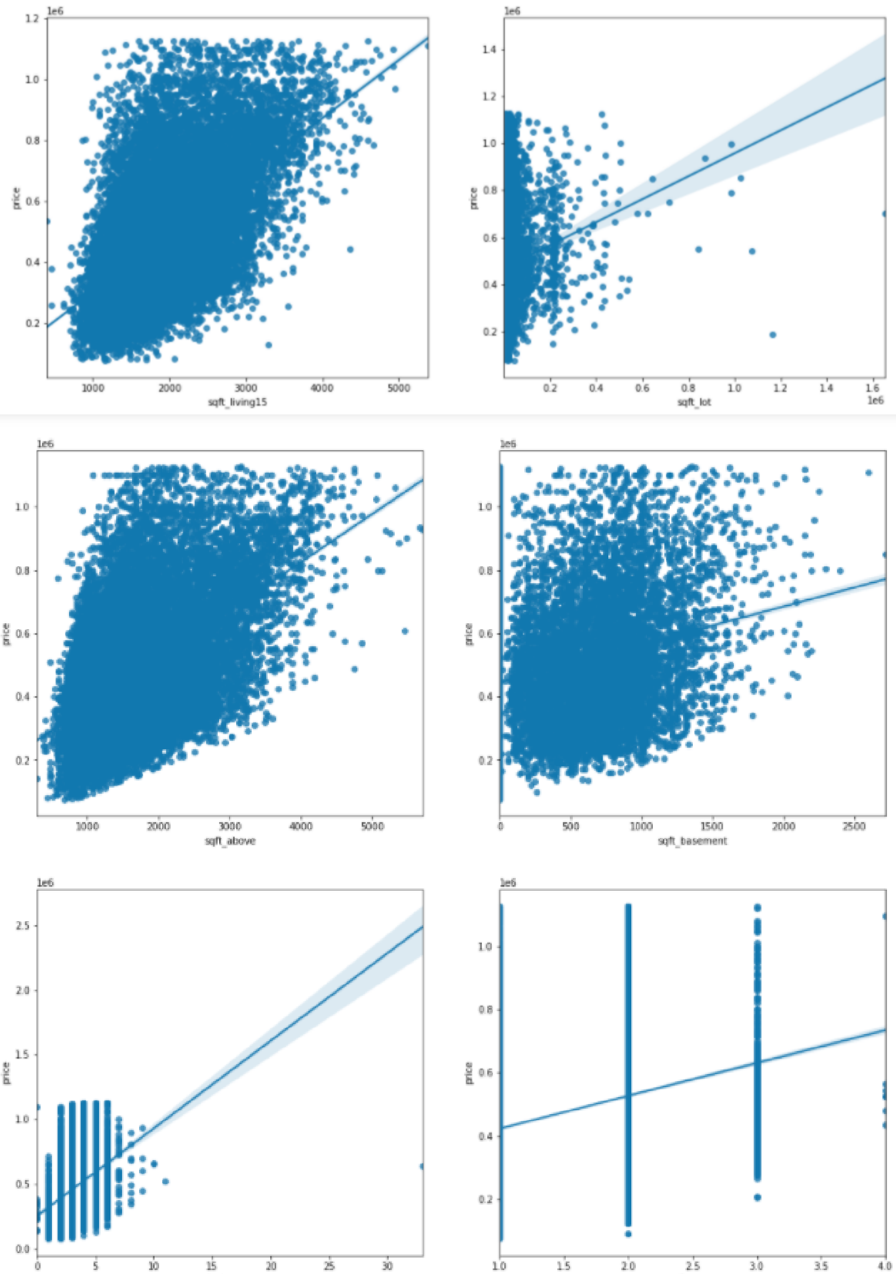
```
In [22]: #DROPPING 'sqft_living', 'sqft_lot15'
         data_p = data_clean_cap.drop(['sqft_living','sqft_lot15'], axis = 1)
         #'id' AND 'zipcode' ARE NOT REQUIRED FOR PREDICTION HENCE WE DROP THEM TOO
         p_data = data_p.drop(['id','zipcode','date'],axis=1)
         p_data.head(5)
```

Out[22]:

	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	lat	long	sq
0	221900.0	3	1.0	5650	1.0	0	0	3	7	1180.0	0	1955	0	47.5112	-122.257	
1	538000.0	3	2.0	7242	2.0	0	0	3	7	2170.0	400	1951	1991	47.7210	-122.319	
2	180000.0	2	1.0	10000	1.0	0	0	3	6	770.0	0	1933	0	47.7379	-122.233	
3	604000.0	4	3.0	5000	1.0	0	0	5	7	1050.0	910	1965	0	47.5208	-122.393	
4	510000.0	3	2.0	8080	1.0	0	0	3	8	1680.0	0	1987	0	47.6168	-122.045	

Scatter plots showing variation of price w.r.t “sqft_living15”, “sqft_loy”, “sqft_above”, “sqft_basement”, “bedrooms” and “floor” attributes :-

```
In [24]: fig = plt.figure(figsize=(16,24))

         plt.subplot(3,2,1)
         sns.regplot(data=data_clean, x='sqft_living15', y='price')

         plt.subplot(3,2,2)
         sns.regplot(data=data_clean, x='sqft_lot', y='price')

         plt.subplot(3,2,3)
         sns.regplot(data=data_clean, x='sqft_above', y='price')

         plt.subplot(3,2,4)
         sns.regplot(data=data_clean, x='sqft_basement', y='price')

         plt.subplot(3,2,5)
         sns.regplot(data=data_clean, x='bedrooms', y='price')

         plt.subplot(3,2,6)
         sns.regplot(data=data_clean, x='floors', y='price')

         plt.show()
```



Separation of class label “price” (dependent variable) from other independent variables followed by splitting of dataset in training and testing set (90% randomly chosen data considered for training and the rest 10% considered for testing). Shape of train and test sets are also displayed :-

```
In [25]: #SEPARATING DEPENDENT VARIABLE (CLASS LABEL) FROM INDEPENDENT VARIABLES :-
pp_data = p_data.copy()
y = pp_data['price'] #CLASS LABEL
x = pp_data.drop(['price'],axis=1) #MULTIPLE INDEPENDENT ATTRIBUTES
#SPLITTING THE DATASET INTO TRAINING AND TESTING DATA :-
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1,random_state = 2)
print('Rows and Columns :- \nx_train : ',x_train.shape,'\nx_test : ',x_test.shape,'\ny_train : ',y_train.shape,'\ny_tes
```

```
Rows and Columns :-
x_train :  (19451, 15)
x_test :  (2162, 15)
y_train :  (19451,)
y_test :  (2162,)
```

Training and Testing of Multilinear Regression Model. A trait score of 74.3% and test score (accuracy of prediction) of 76.3% was obtained :-

```
In [26]: #MUTLTI LINEAR REGRESSION
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(x_train,y_train)
print('TRAIN SCORE : ',reg.score(x_train,y_train))
print('TEST SCORE : ',reg.score(x_test,y_test))
```

```
TRAIN SCORE :  0.7430736520889798
TEST SCORE :  0.7632880163361065
```

Displaying actual values and predicted values of the test set by the model :-

```
In [27]: y_pred = reg.predict(x_test)
df = pd.DataFrame({'Actual price': y_test, 'Predicted price': y_pred})
df
```

Out[27]:

	Actual price	Predicted price
6638	735000.0	6.316176e+05
7366	1129575.0	9.159633e+05
3158	350500.0	3.993298e+05
9117	860000.0	1.001346e+06
3392	122000.0	4.143575e+04
...	...	...
3823	294950.0	4.115622e+05
3268	732000.0	6.422467e+05
19051	299000.0	2.628060e+05
1486	229950.0	2.587728e+05
10955	571000.0	4.653907e+05

2162 rows × 2 columns

Variance, Standard Deviation and R-squared of the predicted values :-

```
In [28]: from sklearn.metrics import r2_score,mean_squared_error
mse = mean_squared_error(y_test,y_pred).round(2) #VARIANCE
rmse = np.sqrt(mse).round(2) #STANDARD DEVIATION
rSq = r2_score(y_test, y_pred).round(2)
print('MEAN SQUARE ERROR : ',mse,'\nROOT MEAN SQUARED ERROR : ',rmse,'\nR SQUARED : ',rSq)
```

```
MEAN SQUARE ERROR :  15230005872.66
ROOT MEAN SQUARED ERROR :  123409.91
R SQUARED :  0.76
```

Pros and Cons of preprocessing w.r.t the dataset :-

Pros :-

- Not many null values were present
- No duplicate records were present

Cons :-

- Presence of too many outliers.
- Had to apply capping even after removal of outliers.
- Accuracy decreased on removal of some correlated features.
-

Exposure gained through this project :-

- Learned to determine the usage of right type of model for right type of output (prediction discrete class label/ continuous quantity)
- Learned how to manage :-
  - Missing values - either by deleting records with missing data or inserting the mean of its corresponding attribute.
  - Duplicate records - must be deleted.
  - Outliers - detection and filtering of outliers by Box Plot Analysis followed by deleting them from the data
  - Correlated features - determining highly correlated features (either negatively or positively correlated) removing one of two highly correlated features to increase the accuracy of the prediction.
- Visualised the data using scatter plots to see the variation of price w.r.t to different attributes. Plotted box plots and histograms to detect the presence of outliers (outliers depicted by dotted points in box plot and skewness of histogram)
- Prediction of numerical values using a single feature as in case of Linear Regression is highly inaccurate, so multiple linear regression should be used which considers multiple features for prediction of class label.