Sales Price Prediction

Linear regression – Report

(Note- the Jupyter notebook for this project is attached at the end of this report as pdf.)

Introduction –

The dataset used for this project was sales price data of houses in the state of Washington DC, USA. The data set contained information of 21,613 houses and data of 21 different features (21 columns) was present in the dataset.

Following are the various columns present in the datset:

'ID', 'Date House was Sold', 'Sale Price', 'No of Bedrooms',
'No of Bathrooms', 'Flat Area (in Sqft)', 'Lot Area (in Sqft)',
'No of Floors', 'Waterfront View', 'No of Times Visited',
'Condition of the House', 'Overall Grade',
'Area of the House from Basement (in Sqft)', 'Basement Area (in Sqft)',
'Age of House (in Years)', 'Renovated Year', 'Zipcode', 'Latitude',
'Longitude', 'Living Area after Renovation (in Sqft)',
'Lot Area after Renovation (in Sqft)

Objective:

- To predict the sales price of the Houses.
- To use various regression model.
- To compare the performance of these models and choose the model whi ch performs the best in prediction.

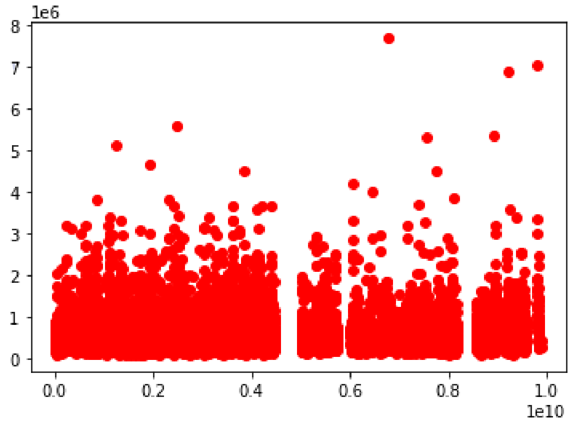
Exploratory Data Analysis:

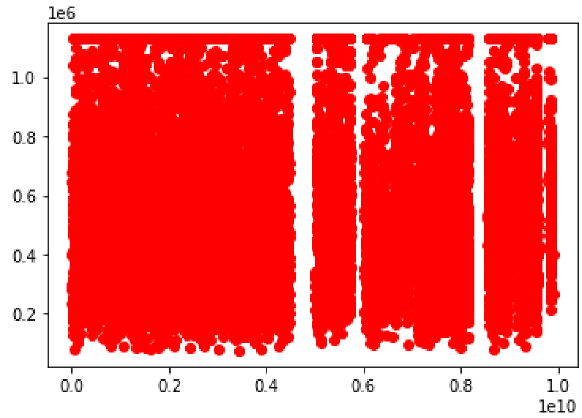
<u>Data Cleaning and Treating Outliers</u>:

On initial visualisation, it was found that the 'Sale Prices' contain some outliers.
This outliers were removed through imputation as,
Upper_limit = q3 + 1.5*iqr
Lower_limit = q1 + 1.5*iqr

Where,
q3 = 75th quartile, q1 = 25th quartile and iqr = q3 – q1 (Inter quartile range)

Doing so, the outliers were removed from sales price.





Before Imputation After Imputation

It was also found that some of the independent variables (columns other than 'Sale Prices') contains missing values. These missing values were then replaced with median in case of continuous variables and with mode in case of object variables.

Also, row which contained missing values in the 'Sales Prices' were removed from the dataset since imputing them will cause the model to learn from bias data.

After this, the dataset was left with 21609 entries of housing data.

Variable Transformation:

The categorical variables were transformed as follows –

From datatype which contained unique values ['None', 'Thrice', 'Four', 'Twice', ' Once'] were replaced by ['0', '3', '4', '2', '1'].

Feature Engineering :

New features were created to provide better information about the dataset fro m the existing features.

‘Ever Renovated’ was created using the features ‘Renovated Year’.
And
‘Year Since Renovation’ was created using ‘Purchase Year’ and ‘Renovated Year’. These new features provided better information than the existing features. And hence, the unwanted features were dropped from the data set.

Dummy variables were created for the features 'Ever Renovated','Waterfront View’, 'Condition of the House'.
Dummy variables were created for the features Zipcode, as this features contained several unique values the features was binned into group of 10.

Scaling the dataset:
The dataset was scaled using StandardScaler.

Removing Multicollinearity –

Variables which can be perfectly defined by other variables were checked In the dataset. And variables with high multicollinearity were removed from the dataset.

After, the above steps the dataset is ready for model building.

The dataset was split into train and test set, with 30% of the data for testing.

Model Building-

4 models were created namely basic linear regression and lasso , ridge and elastic net regression with transformation of independent variable with polynomial features with 2 degrees.

Rmse (root mean squared error) value was calculated for each model.

	RMSE
Linear	98639.493148
Ridge	88196.307098
Lasso	535861.437113
ElasticNet	88282.148070

Results –

Upon comparing the rmse values, we can clearly infer that **Lasso regression gives the best fit with minimum error.**

The problem of multicollinearity was removed from the dataset before building the model. This is the reason that upon lasso regression all the available features were non-zero.

In [1]:

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

import warnings
warnings.simplefilter('ignore')

from sklearn.metrics import mean_squared_error as mse
```

In [2]:

```python
data = pd.read_csv('Raw_Housing_Prices.csv')
data.head()
```

Out[2]:

	ID	Date House was Sold	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	Waterfront View	No of Times Visited	...
0	7129300520	14 October 2017	221900.0	3	1.00	1180.0	5650.0	1.0	No	None	...
1	6414100192	14 December 2017	538000.0	3	2.25	2570.0	7242.0	2.0	No	None	...
2	5631500400	15 February 2016	180000.0	2	1.00	770.0	10000.0	1.0	No	None	...
3	2487200875	14 December 2017	604000.0	4	3.00	1960.0	5000.0	1.0	No	None	...
4	1954400510	15 February 2016	510000.0	3	2.00	1680.0	8080.0	1.0	No	None	...

5 rows × 21 columns

In [3]:

```python
data.columns
```

Out[3]:

```
Index(['ID', 'Date House was Sold', 'Sale Price', 'No of Bedrooms',
       'No of Bathrooms', 'Flat Area (in Sqft)', 'Lot Area (in Sqft)',
       'No of Floors', 'Waterfront View', 'No of Times Visited',
       'Condition of the House', 'Overall Grade',
       'Area of the House from Basement (in Sqft)', 'Basement Area (in Sqft)',
       'Age of House (in Years)', 'Renovated Year', 'Zipcode', 'Latitude',
       'Longitude', 'Living Area after Renovation (in Sqft)',
       'Lot Area after Renovation (in Sqft)'],
      dtype='object')
```

In [4]:

```python
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 House was Sold                        21613 non-null  object 
 2   Sale Price                                 21609 non-null  float64
```

```
 3   No of Bedrooms                             21613 non-null  int64
 4   No of Bathrooms                            21609 non-null  float64
 5   Flat Area (in Sqft)                        21604 non-null  float64
 6   Lot Area (in Sqft)                         21604 non-null  float64
 7   No of Floors                               21613 non-null  float64
 8   Waterfront View                            21613 non-null  object
 9   No of Times Visited                        21613 non-null  object
 10  Condition of the House                     21613 non-null  object
 11  Overall Grade                              21613 non-null  int64
 12  Area of the House from Basement (in Sqft)  21610 non-null  float64
 13  Basement Area (in Sqft)                    21613 non-null  int64
 14  Age of House (in Years)                    21613 non-null  int64
 15  Renovated Year                             21613 non-null  int64
 16  Zipcode                                    21612 non-null  float64
 17  Latitude                                   21612 non-null  float64
 18  Longitude                                  21612 non-null  float64
 19  Living Area after Renovation (in Sqft)     21612 non-null  float64
 20  Lot Area after Renovation (in Sqft)        21613 non-null  int64
dtypes: float64(10), int64(7), object(4)
memory usage: 3.5+ MB
```

In [5]:

```
data['Sale Price'].head(10)
```

Out[5]:

```
0     221900.0
1     538000.0
2     180000.0
3     604000.0
4     510000.0
5    1230000.0
6     257500.0
7     291850.0
8     229500.0
9     323000.0
Name: Sale Price, dtype: float64
```

In [6]:

```
data['Sale Price'].describe()
```

Out[6]:

```
count    2.160900e+04
mean     5.401984e+05
std      3.673890e+05
min      7.500000e+04
25%      3.219500e+05
50%      4.500000e+05
75%      6.450000e+05
max      7.700000e+06
Name: Sale Price, dtype: float64
```

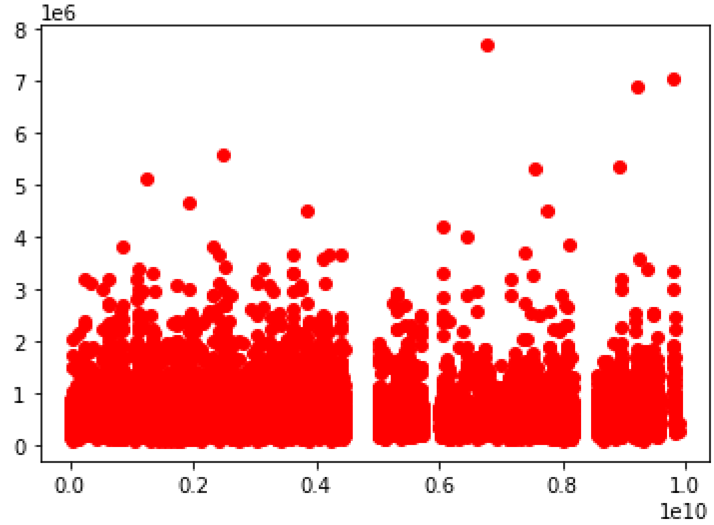
Scater plot for sale price, Finding OUtliers

In [7]:

```
plt.scatter(x = data['ID'], y = data['Sale Price'], color = 'red')
```

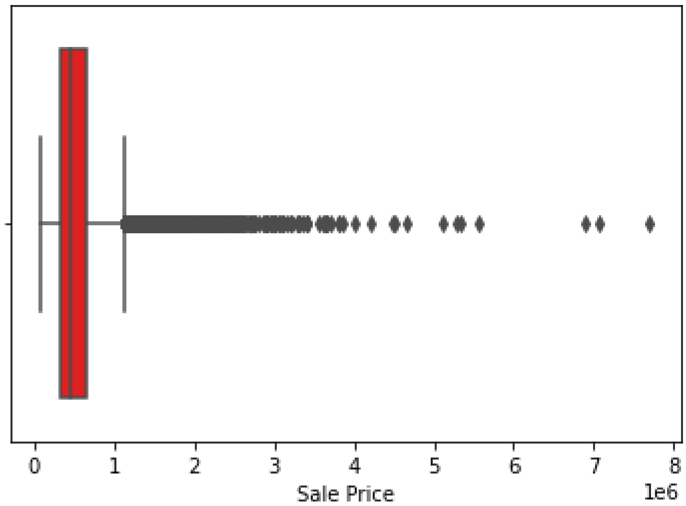
Out[7]:

```
<matplotlib.collections.PathCollection at 0x25adadfa1f0>
```



In [8]:
```
sns.boxplot(x = data['Sale Price'],color = 'red')
```

Out[8]: <AxesSubplot:xlabel='Sale Price'>



Treating Outliers in Sale Price by Imputing

In [9]:
```
q1 = data['Sale Price'].quantile(0.25)
q3 = data['Sale Price'].quantile(0.75)
```

In [10]:
```
iqr = q3 - q1
iqr
```

Out[10]: 323050.0

In [11]:
```
upper_limit = q3 + 1.5*iqr
lower_limit = q1 - 1.5*iqr
upper_limit, lower_limit
```

Out[11]: (1129575.0, -162625.0)

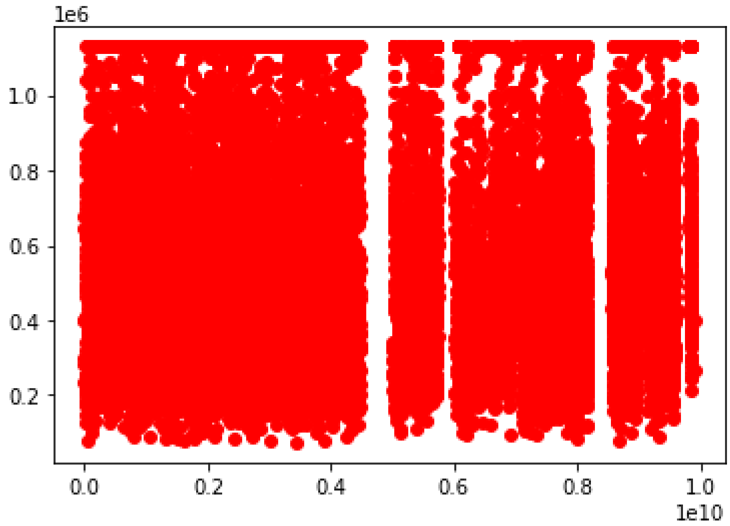
In [12]:
```
def limit_imputer(value):
    if value > upper_limit:
        return upper_limit
```

```
    if value < lower_limit:
        return lower_limit
    else:
        return value
```

In [13]:
```
data['Sale Price'] = data['Sale Price'].apply(limit_imputer)
```

In [14]:
```
plt.scatter(x = data['ID'], y = data['Sale Price'], color = 'red')
```

Out[14]: <matplotlib.collections.PathCollection at 0x25adb2b4820>



In [15]:
```
data['Sale Price'].describe()
```

Out[15]:
```
count    2.160900e+04
mean     5.116186e+05
std      2.500620e+05
min      7.500000e+04
25%      3.219500e+05
50%      4.500000e+05
75%      6.450000e+05
max      1.129575e+06
Name: Sale Price, dtype: float64
```

FInding and Treating missing values

deletion is preffered for treating missing values in target variable

In [16]:
```
data.dropna(inplace = True, axis = 0, subset = ['Sale Price'])
```

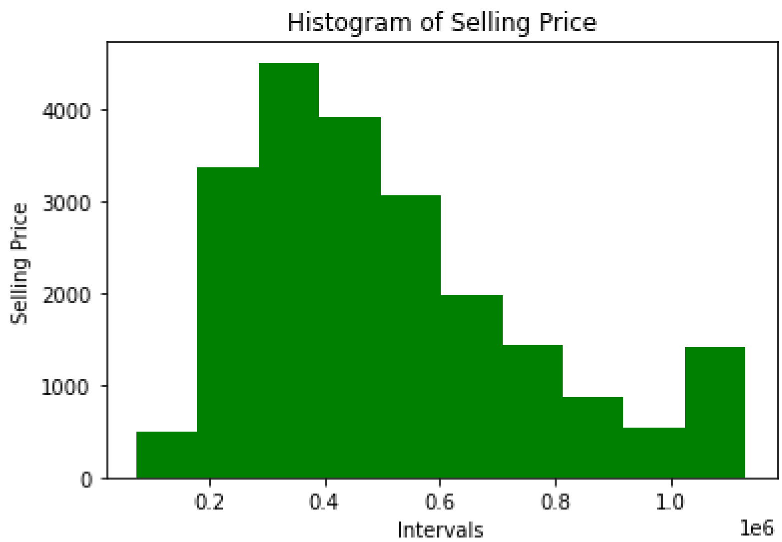
In [17]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21609 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column                                     Non-Null Count  Dtype
---  ------                                     --------------  -----
 0   ID                                         21609 non-null  int64
 1   Date House was Sold                        21609 non-null  object
 2   Sale Price                                 21609 non-null  float64
 3   No of Bedrooms                             21609 non-null  int64
 4   No of Bathrooms                            21605 non-null  float64
 5   Flat Area (in Sqft)                        21600 non-null  float64
 6   Lot Area (in Sqft)                         21600 non-null  float64
 7   No of Floors                               21609 non-null  float64
```

```
 8   Waterfront View                              21609 non-null  object
 9   No of Times Visited                          21609 non-null  object
 10  Condition of the House                       21609 non-null  object
 11  Overall Grade                                21609 non-null  int64
 12  Area of the House from Basement (in Sqft)    21606 non-null  float64
 13  Basement Area (in Sqft)                      21609 non-null  int64
 14  Age of House (in Years)                      21609 non-null  int64
 15  Renovated Year                               21609 non-null  int64
 16  Zipcode                                      21608 non-null  float64
 17  Latitude                                     21608 non-null  float64
 18  Longitude                                    21608 non-null  float64
 19  Living Area after Renovation (in Sqft)       21608 non-null  float64
 20  Lot Area after Renovation (in Sqft)          21609 non-null  int64
dtypes: float64(10), int64(7), object(4)
memory usage: 3.6+ MB
```

checking spread of data over the range

In [18]:
```
plt.hist(data['Sale Price'], bins = 10, color = 'green')
plt.xlabel('Intervals')
plt.ylabel('Selling Price')
plt.title('Histogram of Selling Price')
plt.show()
```



Finding and treating missing values in independent variables

missing values in independent variables are treated by imputation mean or median for continous variable mode for object variable

In [19]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21609 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   ID                                           21609 non-null  int64
 1   Date House was Sold                          21609 non-null  object
 2   Sale Price                                   21609 non-null  float64
 3   No of Bedrooms                               21609 non-null  int64
 4   No of Bathrooms                              21605 non-null  float64
 5   Flat Area (in Sqft)                          21600 non-null  float64
 6   Lot Area (in Sqft)                           21600 non-null  float64
 7   No of Floors                                 21609 non-null  float64
 8   Waterfront View                              21609 non-null  object
 9   No of Times Visited                          21609 non-null  object
```

```
 10  Condition of the House                      21609 non-null  object 
 11  Overall Grade                               21609 non-null  int64  
 12  Area of the House from Basement (in Sqft)   21606 non-null  float64
 13  Basement Area (in Sqft)                     21609 non-null  int64  
 14  Age of House (in Years)                     21609 non-null  int64  
 15  Renovated Year                              21609 non-null  int64  
 16  Zipcode                                     21608 non-null  float64
 17  Latitude                                    21608 non-null  float64
 18  Longitude                                   21608 non-null  float64
 19  Living Area after Renovation (in Sqft)      21608 non-null  float64
 20  Lot Area after Renovation (in Sqft)         21609 non-null  int64  
dtypes: float64(10), int64(7), object(4)
memory usage: 3.6+ MB
```

In [20]:

```
numerical_columns = ['No of Bathrooms', 'Flat Area (in Sqft)', 'Lot Area (in Sqft)',
                       'Area of the House from Basement (in Sqft)','Latitude','Longitude',
                       'Living Area after Renovation (in Sqft)']
```

In [21]:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'median')
data[numerical_columns] = imputer.fit_transform(data[numerical_columns])
```

In [22]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21609 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column                                     Non-Null Count  Dtype  
---  ------                                     --------------  -----  
 0   ID                                          21609 non-null  int64  
 1   Date House was Sold                         21609 non-null  object 
 2   Sale Price                                  21609 non-null  float64
 3   No of Bedrooms                              21609 non-null  int64  
 4   No of Bathrooms                             21609 non-null  float64
 5   Flat Area (in Sqft)                         21609 non-null  float64
 6   Lot Area (in Sqft)                          21609 non-null  float64
 7   No of Floors                                21609 non-null  float64
 8   Waterfront View                             21609 non-null  object 
 9   No of Times Visited                         21609 non-null  object 
 10  Condition of the House                      21609 non-null  object 
 11  Overall Grade                               21609 non-null  int64  
 12  Area of the House from Basement (in Sqft)   21609 non-null  float64
 13  Basement Area (in Sqft)                     21609 non-null  int64  
 14  Age of House (in Years)                     21609 non-null  int64  
 15  Renovated Year                              21609 non-null  int64  
 16  Zipcode                                     21608 non-null  float64
 17  Latitude                                    21609 non-null  float64
 18  Longitude                                   21609 non-null  float64
 19  Living Area after Renovation (in Sqft)      21609 non-null  float64
 20  Lot Area after Renovation (in Sqft)         21609 non-null  int64  
dtypes: float64(10), int64(7), object(4)
memory usage: 3.6+ MB
```

In [23]:

```
data['Zipcode'].shape
```

Out[23]: (21609,)

In [24]:

```
column = data['Zipcode'].values.reshape(-1,1)
column.shape
```

Out[24]: (21609, 1)

In [25]:
```
column = data['Zipcode'].values.reshape(-1,1)
imputer = SimpleImputer(missing_values = np.nan, strategy =
'most_frequent')
data['Zipcode'] = imputer.fit_transform(column)
```

In [26]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21609 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column                                     Non-Null Count  Dtype
---  ------                                     --------------  -----
 0   ID                                         21609 non-null  int64
 1   Date House was Sold                        21609 non-null  object
 2   Sale Price                                 21609 non-null  float64
 3   No of Bedrooms                             21609 non-null  int64
 4   No of Bathrooms                            21609 non-null  float64
 5   Flat Area (in Sqft)                        21609 non-null  float64
 6   Lot Area (in Sqft)                         21609 non-null  float64
 7   No of Floors                               21609 non-null  float64
 8   Waterfront View                            21609 non-null  object
 9   No of Times Visited                        21609 non-null  object
 10  Condition of the House                     21609 non-null  object
 11  Overall Grade                              21609 non-null  int64
 12  Area of the House from Basement (in Sqft)  21609 non-null  float64
 13  Basement Area (in Sqft)                    21609 non-null  int64
 14  Age of House (in Years)                    21609 non-null  int64
 15  Renovated Year                             21609 non-null  int64
 16  Zipcode                                    21609 non-null  float64
 17  Latitude                                   21609 non-null  float64
 18  Longitude                                  21609 non-null  float64
 19  Living Area after Renovation (in Sqft)     21609 non-null  float64
 20  Lot Area after Renovation (in Sqft)        21609 non-null  int64
dtypes: float64(10), int64(7), object(4)
memory usage: 3.6+ MB
```

Variable Transformation

In [27]:
```
data['Zipcode'] = data['Zipcode'].astype(object)
data.dtypes
```

Out[27]:
```
ID                                             int64
Date House was Sold                           object
Sale Price                                   float64
No of Bedrooms                                 int64
No of Bathrooms                              float64
Flat Area (in Sqft)                          float64
Lot Area (in Sqft)                           float64
No of Floors                                 float64
Waterfront View                               object
No of Times Visited                           object
Condition of the House                        object
Overall Grade                                  int64
Area of the House from Basement (in Sqft)    float64
Basement Area (in Sqft)                        int64
Age of House (in Years)                        int64
Renovated Year                                 int64
Zipcode                                       object
Latitude                                     float64
Longitude                                    float64
Living Area after Renovation (in Sqft)       float64
Lot Area after Renovation (in Sqft)            int64
dtype: object
```

In [28]:
```
data['No of Times Visited'].unique()
```

Out[28]:
```
array(['None', 'Thrice', 'Four', 'Twice', 'Once'], dtype=object)
```

In [29]:
```
mapping = {'None':'0',
           'Once':'1',
           'Twice':'2',
           'Thrice':'3',
           'Four': '4'}
data['No of Times Visited'] = data['No of Times Visited'].map(mapping)
```

In [30]:
```
data['No of Times Visited'].unique()
```

Out[30]:
```
array(['0', '3', '4', '2', '1'], dtype=object)
```

In [31]:
```
data['Ever Renovated'] = np.where(data['Renovated Year'] ==
0,'No','Yes')
```

In [32]:
```
data.head()
```

Out[32]:

	ID	Date House was Sold	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	Waterfront View	No of Times Visited	...
0	7129300520	14 October 2017	221900.0	3	1.00	1180.0	5650.0	1.0	No	0	...
1	6414100192	14 December 2017	538000.0	3	2.25	2570.0	7242.0	2.0	No	0	...
2	5631500400	15 February 2016	180000.0	2	1.00	770.0	10000.0	1.0	No	0	...
3	2487200875	14 December 2017	604000.0	4	3.00	1960.0	5000.0	1.0	No	0	...
4	1954400510	15 February 2016	510000.0	3	2.00	1680.0	8080.0	1.0	No	0	...

5 rows × 22 columns

In [33]:
```
data['Purchase Year'] = pd.DatetimeIndex(data['Date House was
Sold']).year
```

In [34]:
```
data['Year Since Renovation'] = np.where(data['Ever Renovated'] ==
'Yes',
                                         abs(data['Purchase Year'] -
data['Renovated Year']),0)
```

In [35]:
```
data.head()
```

Out[35]:

	ID	Date House was Sold	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	Waterfront View	No of Times Visited	...
0	7129300520	14 October 2017	221900.0	3	1.00	1180.0	5650.0	1.0	No	0	...
1	6414100192	14 December 2017	538000.0	3	2.25	2570.0	7242.0	2.0	No	0	...
2	5631500400	15 February 2016	180000.0	2	1.00	770.0	10000.0	1.0	No	0	...
3	2487200875	14 December 2017	604000.0	4	3.00	1960.0	5000.0	1.0	No	0	...
4	1954400510	15 February 2016	510000.0	3	2.00	1680.0	8080.0	1.0	No	0	...

5 rows × 24 columns

In [36]:

```
data.drop(columns = ['Purchase Year','Date House was Sold', 'Renovated Year'],inplace = True)
```

In [37]:

```
data.head()
```

Out[37]:

	ID	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	Waterfront View	No of Times Visited	Condition of the House	...
0	7129300520	221900.0	3	1.00	1180.0	5650.0	1.0	No	0	Fair	...
1	6414100192	538000.0	3	2.25	2570.0	7242.0	2.0	No	0	Fair	...
2	5631500400	180000.0	2	1.00	770.0	10000.0	1.0	No	0	Fair	...
3	2487200875	604000.0	4	3.00	1960.0	5000.0	1.0	No	0	Excellent	...
4	1954400510	510000.0	3	2.00	1680.0	8080.0	1.0	No	0	Fair	...

5 rows × 21 columns

In [38]:

```
data.drop(columns = ['ID'],inplace = True)
```

In [39]:

```
data.head()
```

Out[39]:

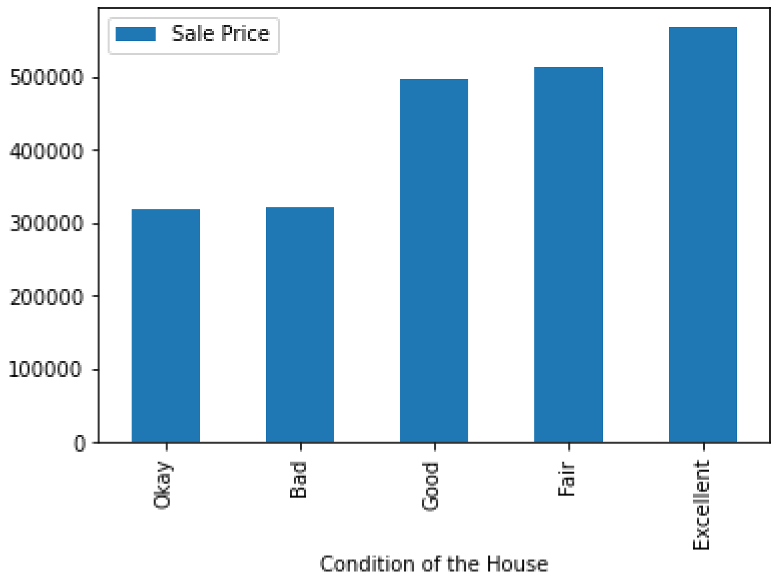
	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	Waterfront View	No of Times Visited	Condition of the House	Overall Grade	Are the Ho f Basen (in S
0	221900.0	3	1.00	1180.0	5650.0	1.0	No	0	Fair	7	11
1	538000.0	3	2.25	2570.0	7242.0	2.0	No	0	Fair	7	21
2	180000.0	2	1.00	770.0	10000.0	1.0	No	0	Fair	6	7
3	604000.0	4	3.00	1960.0	5000.0	1.0	No	0	Excellent	7	10
4	510000.0	3	2.00	1680.0	8080.0	1.0	No	0	Fair	8	16

In [40]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21609 entries, 0 to 21612
Data columns (total 20 columns):
 #   Column                                     Non-Null Count  Dtype  
---  ------                                     --------------  -----  
 0   Sale Price                                 21609 non-null  float64
 1   No of Bedrooms                             21609 non-null  int64  
 2   No of Bathrooms                            21609 non-null  float64
 3   Flat Area (in Sqft)                        21609 non-null  float64
 4   Lot Area (in Sqft)                         21609 non-null  float64
 5   No of Floors                               21609 non-null  float64
 6   Waterfront View                            21609 non-null  object 
 7   No of Times Visited                        21609 non-null  object 
 8   Condition of the House                     21609 non-null  object 
 9   Overall Grade                              21609 non-null  int64  
 10  Area of the House from Basement (in Sqft)  21609 non-null  float64
 11  Basement Area (in Sqft)                    21609 non-null  int64  
 12  Age of House (in Years)                    21609 non-null  int64  
 13  Zipcode                                    21609 non-null  object 
 14  Latitude                                   21609 non-null  float64
 15  Longitude                                  21609 non-null  float64
 16  Living Area after Renovation (in Sqft)     21609 non-null  float64
 17  Lot Area after Renovation (in Sqft)        21609 non-null  int64  
 18  Ever Renovated                             21609 non-null  object 
 19  Year Since Renovation                      21609 non-null  int64  
dtypes: float64(9), int64(6), object(5)
memory usage: 3.5+ MB
```

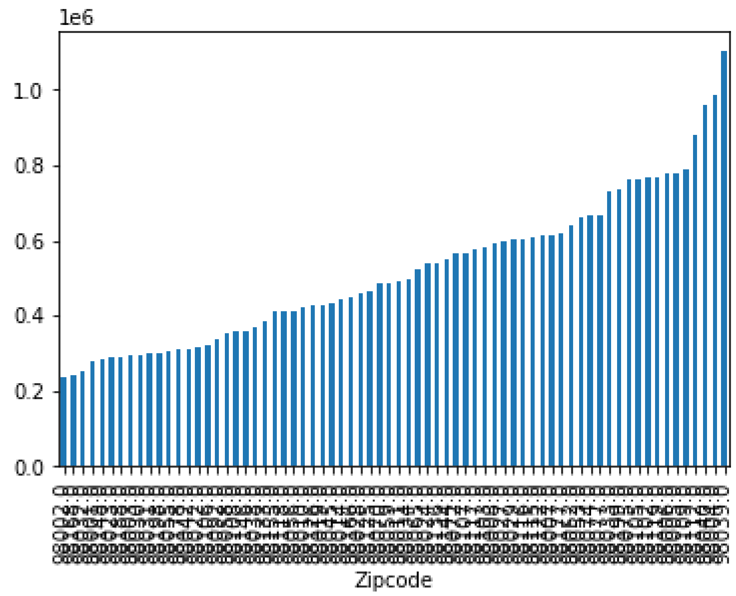
In [41]:
```
data.groupby('Condition of the House'
            )['Sale Price'].mean().sort_values().plot(kind = 'bar')
plt.legend()
```

Out[41]: <matplotlib.legend.Legend at 0x25adbd919a0>



In [42]:
```
data.groupby('Zipcode'
            )['Sale Price'].mean().sort_values().plot(kind = 'bar')
```

Out[42]: <AxesSubplot:xlabel='Zipcode'>



binning and creation of dummy variable

In [43]:
```
data = pd.get_dummies(data, columns = ['Ever Renovated','Waterfront View'], drop_first = True)
```

In [44]:
```
data = pd.get_dummies(data, columns = ['Condition of the House'], drop_first = True)
```

In [45]:
```
data.head()
```

Out[45]:

	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	No of Times Visited	Overall Grade	Area of the House from Basement (in Sqft)	Basement Area (in Sqft)	...	Lo
0	221900.0	3	1.00	1180.0	5650.0	1.0	0	7	1180.0	0	...	-
1	538000.0	3	2.25	2570.0	7242.0	2.0	0	7	2170.0	400	...	-
2	180000.0	2	1.00	770.0	10000.0	1.0	0	6	770.0	0	...	-
3	604000.0	4	3.00	1960.0	5000.0	1.0	0	7	1050.0	910	...	-
4	510000.0	3	2.00	1680.0	8080.0	1.0	0	8	1680.0	0	...	-

5 rows × 23 columns

In [46]:
```
Zip_table = data.groupby('Zipcode').agg({'Sale Price':'mean'}).sort_values('Sale Price', ascending = True)
```

In [47]:
```
Zip_table.head()
```

Out[47]:

Zipcode	Sale Price
98002.0	234284.035176
98168.0	240328.371747
98032.0	251296.240000
98001.0	280804.690608

98148.0	284908.596491

In [48]:
```
Zip_table['Zipcode_Group'] = pd.cut(Zip_table['Sale Price'],bins = 10,
                                    labels = ['Zipcode_0',
                                              'Zipcode_1',
                                              'Zipcode_2',
                                              'Zipcode_3',
                                              'Zipcode_4',
                                              'Zipcode_5',
                                              'Zipcode_6',
                                              'Zipcode_7',
                                              'Zipcode_8',
                                              'Zipcode_9'],
                                    include_lowest = True)
```

In [49]:
```
Zip_table = Zip_table.drop(columns = 'Sale Price')
```

In [50]:
```
data = pd.merge(data,Zip_table,
                left_on = 'Zipcode',
                how= 'left',
                right_index = True)
```

In [51]:
```
data = data.drop(columns = 'Zipcode')
```

In [52]:
```
data.head()
```

Out[52]:

	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	No of Times Visited	Overall Grade	Area of the House from Basement (in Sqft)	Basement Area (in Sqft)	...	Li Re
0	221900.0	3	1.00	1180.0	5650.0	1.0	0	7	1180.0	0	...	
1	538000.0	3	2.25	2570.0	7242.0	2.0	0	7	2170.0	400	...	
2	180000.0	2	1.00	770.0	10000.0	1.0	0	6	770.0	0	...	
3	604000.0	4	3.00	1960.0	5000.0	1.0	0	7	1050.0	910	...	
4	510000.0	3	2.00	1680.0	8080.0	1.0	0	8	1680.0	0	...	

5 rows × 23 columns

In [53]:
```
data = pd.get_dummies(data, columns = ['Zipcode_Group'], drop_first =
True)
```

In [54]:
```
data.head()
```

Out[54]:

	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	No of Times Visited	Overall Grade	Area of the House from Basement (in Sqft)	Basement Area (in Sqft)	...	C Hc

0	221900.0	3	1.00	1180.0	5650.0	1.0	0	7	1180.0	0 ...
1	538000.0	3	2.25	2570.0	7242.0	2.0	0	7	2170.0	400 ...
2	180000.0	2	1.00	770.0	10000.0	1.0	0	6	770.0	0 ...
3	604000.0	4	3.00	1960.0	5000.0	1.0	0	7	1050.0	910 ...
4	510000.0	3	2.00	1680.0	8080.0	1.0	0	8	1680.0	0 ...

5 rows × 31 columns

In [55]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
Y = data['Sale Price']
X = scaler.fit_transform(data.drop(columns = ['Sale Price']))
X = pd.DataFrame(data = X, columns = data.drop(columns = ['Sale
Price']).columns)
```

Corelation between Variables

In [56]:

```
corr_matrix = X.corr()
corr_matrix
```

Out[56]:

	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	No of Times Visited	Overall Grade	the Ba (
No of Bedrooms	1.000000	0.515813	0.576628	0.031692	0.175536	0.079575	0.349223	0
No of Bathrooms	0.515813	1.000000	0.754568	0.087732	0.500776	0.187791	0.635638	0
Flat Area (in Sqft)	0.576628	0.754568	1.000000	0.172721	0.354142	0.284678	0.705725	0
Lot Area (in Sqft)	0.031692	0.087732	0.172721	1.000000	-0.005162	0.074668	0.102314	0
No of Floors	0.175536	0.500776	0.354142	-0.005162	1.000000	0.029504	0.461368	0
No of Times Visited	0.079575	0.187791	0.284678	0.074668	0.029504	1.000000	0.223661	0
Overall Grade	0.349223	0.635638	0.705725	0.102314	0.461368	0.223661	1.000000	0
Area of the House from Basement (in Sqft)	0.477549	0.685088	0.876226	0.183492	0.524031	0.167812	0.705153	1
Basement Area (in Sqft)	0.303294	0.283798	0.435142	0.015252	-0.245572	0.276974	0.145232	-0
Age of House (in Years)	-0.154113	-0.505954	-0.318146	-0.053119	-0.489244	0.053395	-0.456711	-0
Latitude	-0.008708	0.024570	0.052538	-0.085719	0.049692	0.006162	0.111226	-0
Longitude	0.129569	0.223171	0.240091	0.229449	0.125620	-0.078453	0.201736	0
Living Area after Renovation (in Sqft)	0.391771	0.568568	0.756185	0.144507	0.280106	0.280452	0.681362	0
Lot Area after Renovation (in Sqft)	0.029264	0.087226	0.183223	0.718527	-0.011204	0.072561	0.107581	0
Year Since Renovation	-0.007198	0.003551	0.023503	0.013835	-0.000901	0.093546	-0.024388	0
Ever Renovated_Yes	0.018573	0.050282	0.055111	0.007736	0.006297	0.104051	0.010010	0
Waterfront View_Yes	-0.006578	0.063761	0.103841	0.021605	0.023719	0.401856	0.070332	0
Condition of the House_Excellent	0.028148	-0.034281	-0.018182	-0.014503	-0.120524	0.034392	-0.082628	-0
Condition of the	0.004778	0.190440	0.102627	-0.011334	0.317934	-0.037127	0.197510	0

House_Fair								
Condition of the House_Good	-0.008847	-0.166037	-0.083995	0.013033	-0.257680	0.022690	-0.140113	-0
Condition of the House_Okay	-0.051957	-0.077419	-0.065334	0.037619	-0.055951	-0.018557	-0.090561	-0
Zipcode_Group_Zipcode_1	-0.010603	-0.032810	-0.058817	0.023684	-0.003385	-0.065000	-0.075495	-0
Zipcode_Group_Zipcode_2	-0.039342	-0.081460	-0.063005	0.052103	-0.067904	0.004754	-0.121379	-0
Zipcode_Group_Zipcode_3	-0.074129	-0.034459	-0.078761	-0.041112	0.079211	0.005905	-0.047869	-0
Zipcode_Group_Zipcode_4	0.024433	0.084054	0.086139	-0.012050	0.071786	0.003509	0.151245	0
Zipcode_Group_Zipcode_5	0.019420	0.052804	0.075978	0.015320	0.009203	0.024801	0.095613	0
Zipcode_Group_Zipcode_6	0.090177	0.123256	0.160045	-0.023270	0.069857	0.068144	0.200548	0
Zipcode_Group_Zipcode_7	0.016725	0.037746	0.051211	-0.027419	0.064981	-0.012548	0.077126	0
Zipcode_Group_Zipcode_8	0.102736	0.110012	0.169576	-0.007025	-0.008633	0.065335	0.156952	0
Zipcode_Group_Zipcode_9	0.035694	0.067871	0.090253	0.002671	0.005868	0.012923	0.048638	0

30 rows × 30 columns

In [57]:
```
k = X.corr()
z = [[str(i),str(j)] for i in k.columns for j in k.columns if
(k.loc[i,j] > abs(0.5)) & (i!=j)]
z,len(z)
```

Out[57]:
```
([['No of Bedrooms', 'No of Bathrooms'],
  ['No of Bedrooms', 'Flat Area (in Sqft)'],
  ['No of Bathrooms', 'No of Bedrooms'],
  ['No of Bathrooms', 'Flat Area (in Sqft)'],
  ['No of Bathrooms', 'No of Floors'],
  ['No of Bathrooms', 'Overall Grade'],
  ['No of Bathrooms', 'Area of the House from Basement (in Sqft)'],
  ['No of Bathrooms', 'Living Area after Renovation (in Sqft)'],
  ['Flat Area (in Sqft)', 'No of Bedrooms'],
  ['Flat Area (in Sqft)', 'No of Bathrooms'],
  ['Flat Area (in Sqft)', 'Overall Grade'],
  ['Flat Area (in Sqft)', 'Area of the House from Basement (in Sqft)'],
  ['Flat Area (in Sqft)', 'Living Area after Renovation (in Sqft)'],
  ['Lot Area (in Sqft)', 'Lot Area after Renovation (in Sqft)'],
  ['No of Floors', 'No of Bathrooms'],
  ['No of Floors', 'Area of the House from Basement (in Sqft)'],
  ['Overall Grade', 'No of Bathrooms'],
  ['Overall Grade', 'Flat Area (in Sqft)'],
  ['Overall Grade', 'Area of the House from Basement (in Sqft)'],
  ['Overall Grade', 'Living Area after Renovation (in Sqft)'],
  ['Area of the House from Basement (in Sqft)', 'No of Bathrooms'],
  ['Area of the House from Basement (in Sqft)', 'Flat Area (in Sqft)'],
  ['Area of the House from Basement (in Sqft)', 'No of Floors'],
  ['Area of the House from Basement (in Sqft)', 'Overall Grade'],
  ['Area of the House from Basement (in Sqft)',
   'Living Area after Renovation (in Sqft)'],
  ['Living Area after Renovation (in Sqft)', 'No of Bathrooms'],
  ['Living Area after Renovation (in Sqft)', 'Flat Area (in Sqft)'],
  ['Living Area after Renovation (in Sqft)', 'Overall Grade'],
  ['Living Area after Renovation (in Sqft)',
   'Area of the House from Basement (in Sqft)'],
  ['Lot Area after Renovation (in Sqft)', 'Lot Area (in Sqft)'],
  ['Year Since Renovation', 'Ever Renovated_Yes'],
  ['Ever Renovated_Yes', 'Year Since Renovation']],
 32)
```

In [58]:
```
#importing variance inflation factor function from the statsmodels
from statsmodels.stats.outliers_influence import
```

```
variance_inflation_factor

vif_data = X

## calculating VIF for every column
VIF = pd.Series([variance_inflation_factor(vif_data.values, i) for i in
range(vif_data.shape[1])],index = vif_data.columns)
```

In [59]:
```
VIF[VIF == VIF.max()].index[0]
```

Out[59]: 'Flat Area (in Sqft)'

In [60]:
```
def MC_remover(data):
    vif = pd.Series([variance_inflation_factor(data.values,i) for i in
range(data.shape[1])],index = data.columns)
    if vif.max()> 5:
        print(vif[vif == vif.max()].index[0],'has been removed')
        data = data.drop(columns = [vif[vif == vif.max()].index[0]])
        return data
    else:
        print('No multicollinearity present anymore')
        return data
```

In [61]:
```
for i in range(7):
    vif_data = MC_remover(vif_data)

vif_data.head()
```

```
Flat Area (in Sqft) has been removed
Condition of the House_Fair has been removed
No multicollinearity present anymore
No multicollinearity present anymore
No multicollinearity present anymore
No multicollinearity present anymore
No multicollinearity present anymore
```

Out[61]:

	No of Bedrooms	No of Bathrooms	Lot Area (in Sqft)	No of Floors	No of Times Visited	Overall Grade	Area of the House from Basement (in Sqft)	Basement Area (in Sqft)	Age of House (in Years)	Lati
0	-0.398724	-1.447526	-0.228291	-0.915389	-0.30579	-0.563993	-0.734722	-0.658697	0.544734	-0.35
1	-0.398724	0.175684	-0.189858	0.936817	-0.30579	-0.563993	0.460990	0.245134	0.680915	1.16
2	-1.474115	-1.447526	-0.123276	-0.915389	-0.30579	-1.468566	-1.229916	-0.658697	1.293731	1.28
3	0.676667	1.149611	-0.243983	-0.915389	-0.30579	-0.563993	-0.891735	1.397518	0.204281	-0.28
4	-0.398724	-0.148958	-0.169628	-0.915389	-0.30579	0.340581	-0.130827	-0.658697	-0.544715	0.40

5 rows × 28 columns

In [62]:
```
VIF = pd.Series([variance_inflation_factor(vif_data.values, i) for i in
range(vif_data.shape[1])],index = vif_data.columns)
VIF,len(vif_data.columns)
```

```
Out[62]: (No of Bedrooms                             1.638990
 No of Bathrooms                            3.373805
 Lot Area (in Sqft)                         2.107495
 No of Floors                               2.127703
 No of Times Visited                        1.432363
 Overall Grade                              2.956967
 Area of the House from Basement (in Sqft)  4.580042
 Basement Area (in Sqft)                    1.974981
 Age of House (in Years)                    2.626504
 Latitude                                   2.471343
 Longitude                                  1.672667
 Living Area after Renovation (in Sqft)     3.063886
 Lot Area after Renovation (in Sqft)        2.144068
 Year Since Renovation                      2.788064
 Ever Renovated_Yes                         2.955539
 Waterfront View_Yes                        1.208288
 Condition of the House_Excellent           1.206487
 Condition of the House_Good                1.251488
 Condition of the House_Okay                1.025386
 Zipcode_Group_Zipcode_1                    1.538211
 Zipcode_Group_Zipcode_2                    2.570583
 Zipcode_Group_Zipcode_3                    2.818509
 Zipcode_Group_Zipcode_4                    3.192429
 Zipcode_Group_Zipcode_5                    1.728047
 Zipcode_Group_Zipcode_6                    2.014775
 Zipcode_Group_Zipcode_7                    1.233626
 Zipcode_Group_Zipcode_8                    1.389355
 Zipcode_Group_Zipcode_9                    1.048571
 dtype: float64,
 28)
```

Train/Test Set

```
In [63]: X = vif_data
         y = data['Sale Price']
```

```
In [64]: from sklearn.model_selection import train_test_split
         x_train,x_test,y_train, y_test = train_test_split(X,Y, test_size =
         0.3,random_state = 101)
         x_train.shape, x_test.shape , y_train.shape, y_test.shape

Out[64]: ((15126, 28), (6483, 28), (15126,), (6483,))
```

Linear Regression

```
In [65]: from sklearn.linear_model import LinearRegression
         lr = LinearRegression(normalize = True)
         lr.fit(x_train, y_train)

Out[65]: LinearRegression(normalize=True)
```

```
In [66]: lr.coef_

Out[66]: array([ -3928.66247639,  12028.44560689,  14967.00497585,   2697.55278605,
          27220.31313417,  59965.44665815,  80697.80906997,  27729.56715434,
          27873.90231343,  21397.40341959, -23854.32640243,  17943.26729788,
          -2896.98542901, -10179.085198  ,  14594.33847962,  10761.77007875,
          14239.3533334 ,   5095.97603572,  -2296.64888137,  12165.83372082,
```

```
       33842.29544383,   63269.82875283,   81086.08553213,   50718.63947886,
       73274.09568028,   40153.03595158,   67405.70271285,   22113.74944051])
```

In [67]:
```
Y_predicted = lr.predict(x_test)
lr.score(x_test,y_test)
```

Out[67]:
```
0.8461987715586199
```

In [68]:
```
from sklearn.metrics import r2_score
R = r2_score(y_test,Y_predicted)
n = len(y)
m = len(X.columns)
adj_R = 1 - ((1-R)*(n-1))/(n-m-1)
adj_R
```

Out[68]:
```
0.8459992148210685
```

In [69]:
```
from sklearn.metrics import mean_squared_error

rmse_linear = np.sqrt(mean_squared_error(y_test, Y_predicted))
rmse_linear
```

Out[69]:
```
98639.49314807726
```

Polynomial Regression

In [70]:
```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 2)
poly_x = poly.fit_transform(X)
```

In [71]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train, y_test = train_test_split(poly_x,Y, test_size =
0.3,random_state = 101)
x_train.shape, x_test.shape , y_train.shape, y_test.shape
```

Out[71]:
```
((15126, 435), (6483, 435), (15126,), (6483,))
```

In [72]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression(normalize = True)
lr.fit(x_train, y_train)
y_predicted2 = lr.predict(x_test)
```

In [73]:
```
from sklearn.metrics import mean_squared_error

rmse_poly = np.sqrt(mean_squared_error(y_test, y_predicted2))
rmse_poly
```

Out[73]:
```
3.1312585518948364e+16
```

Ridge Regression

In [74]:
```
from sklearn.linear_model import RidgeCV

alphas = [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 80]

ridgeCV = RidgeCV( alphas=alphas, cv=4)

ridgeCV.fit(x_train,y_train)
```

Out[74]:
```
RidgeCV(alphas=array([5.0e-03, 5.0e-02, 1.0e-01, 3.0e-01, 1.0e+00, 3.0e+00, 5.0e+00,
       1.0e+01, 1.5e+01, 3.0e+01, 8.0e+01]),
        cv=4)
```

In [75]:
```
Y_predicted3 = ridgeCV.predict(x_test)

rmse_ridgeCV = np.sqrt(mean_squared_error(y_test, Y_predicted3))
rmse_ridgeCV
```

Out[75]:
```
88196.30709770852
```

Lasso Regression

In [76]:
```
from sklearn.linear_model import LassoCV

alphas2 = np.array([1e-5, 5e-5, 0.0001, 0.0005])

lassoCV = LassoCV(alphas=alphas2,max_iter=5e4,cv=3)

lassoCV.fit(x_train,y_train)

Y_predicted4 = lassoCV.predict(x_test)
```

In [77]:
```
rmse_lassoCV = np.sqrt(mean_squared_error(y_test, Y_predicted4))
rmse_lassoCV
```

Out[77]:
```
535861.4371132243
```

In [78]:
```
print('Of {} coefficients, {} are non-zero with
Lasso.'.format(len(lassoCV.coef_),

len(lassoCV.coef_.nonzero()[0])))
```

```
Of 435 coefficients, 434 are non-zero with Lasso.
```

ElasticNetCV

In [79]:
```
from sklearn.linear_model import ElasticNetCV
```

```
l1_ratios = np.linspace(0.1, 0.9, 9)

elasticNetCV = 
ElasticNetCV(alphas=alphas2,l1_ratio=l1_ratios,max_iter=1e4)

elasticNetCV.fit(x_train,y_train)

y_predicted5 = elasticNetCV.predict(x_test)
```

In [80]:

```
rmse_elasticNetCV = np.sqrt(mean_squared_error(y_test, y_predicted5))
rmse_elasticNetCV
```

Out[80]: 88282.14806969585

In [81]:

```
rmse_vals = [rmse_linear, rmse_ridgeCV, rmse_lassoCV, rmse_elasticNetCV
]

labels = ['Linear', 'Ridge', 'Lasso', 'ElasticNet']

rmse_df = pd.Series(rmse_vals, index=labels).to_frame()
rmse_df.rename(columns={0: 'RMSE'}, inplace=1)
rmse_df
```

Out[81]:

	RMSE
Linear	98639.493148
Ridge	88196.307098
Lasso	535861.437113
ElasticNet	88282.148070

After comparing the root_mean_squared_error of all this regression model we can clearly infer that the losso regression fits the dataset best.