HOUSE PRICE PREDICTION

Context: If you are like me, you might get overwhelmed when having to make big decisions such as buying a house. In such cases, I always like to go for a data driven approach, that will help me find an optimum solution. This involves two steps. First, we need to gather as much data as we can. Second, we need to define a metric for success.

Gathering housing prices requires some effort. A caveat is that the asking prices are not the prices to which the houses were actually sold. Defining a metric for success is somewhat subjective. I consider a house to be a good option if the house price is cheap compared to other listings in the area.

Content: The housing prices have been obtained from Pararius.nl as a snapshot in August 2021. The original data provided features such as price, floor area and the number of rooms. The data has been further enhanced by utilising the Mapbox API to obtain the coordinates of each listing.

Project Description: Building a simple machine learning model to predict house prices based on various features such as square footage, number of bedrooms, neighborhood, and more. This project will introduce you to regression analysis, which is a fundamental concept in data science

Importing the libraries

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

Load the dataset

```
df=pd.read_csv('/content/HousingPrices-Amsterdam-August-2021.csv')
df
```

	Unnamed: 0	Address	Zip	Price	Area	Room	Lon	Lat
0	1	Blasiusstraat 8 2, Amsterdam	1091 CR	685000.0	64	3	4.907736	52.356157
1	2	Kromme Leimuidenstraat 13 H, Amsterdam	1059 EL	475000.0	60	3	4.850476	52.348586
2	3	Zaaiersweg 11 A, Amsterdam	1097 SM	850000.0	109	4	4.944774	52.343782
3	4	Tenerifestraat 40, Amsterdam	1060 TH	580000.0	128	6	4.789928	52.343712

Data Exploration

```
df.head()
```

	Unnamed: 0	Address	Zip	Price	Area	Room	Lon	Lat
0	1	Blasiusstraat 8 2, Amsterdam	1091 CR	685000.0	64	3	4.907736	52.356157
1	2	Kromme Leimuidenstraat 13 H, Amsterdam	1059 EL	475000.0	60	3	4.850476	52.348586
2	3	Zaaiersweg 11 A, Amsterdam	1097 SM	850000.0	109	4	4.944774	52.343782

```
df.tail()
```

	Unnamed: 0	Address	Zip	Price	Area	Room	Lon	Lat
919	920	Ringdijk, Amsterdam	1097 AE	750000.0	117	1	4.927757	52.354173
920	921	Kleine Beerstraat 31, Amsterdam	1033 CP	350000.0	72	3	4.890612	52.414587
921	922	Stuyvesantstraat 33 II, Amsterdam	1058 AK	350000.0	51	3	4.856935	52.363256

```
df.dropna()
```

	Unnamed: 0	Address	Zip	Price	Area	Room	Lon	Lat
0	1	Blasiusstraat 8 2, Amsterdam	1091 CR	685000.0	64	3	4.907736	52.356157
1	2	Kromme Leimuidenstraat 13 H, Amsterdam	1059 EL	475000.0	60	3	4.850476	52.348586
2	3	Zaaiersweg 11 A, Amsterdam	1097 SM	850000.0	109	4	4.944774	52.343782
3	4	Tenerifestraat 40, Amsterdam	1060 TH	580000.0	128	6	4.789928	52.343712
4	5	Winterjanpad 21, Amsterdam	1036 KN	720000.0	138	5	4.902503	52.410538
...	...	...	...	...	...	...	...	...
919	920	Ringdijk, Amsterdam	1097 AE	750000.0	117	1	4.927757	52.354173
920	921	Kleine Beerstraat	1033	350000.0	72	3	4.890612	52.414587

```
df.shape
```

```
(924, 8)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 924 entries, 0 to 923
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   924 non-null    int64
```

```
 1   Address     924 non-null    object
 2   Zip         924 non-null    object
 3   Price       920 non-null    float64
 4   Area        924 non-null    int64
 5   Room        924 non-null    int64
 6   Lon         924 non-null    float64
 7   Lat         924 non-null    float64
dtypes: float64(3), int64(3), object(2)
memory usage: 57.9+ KB
```

```
df.describe()
```

	Unnamed: 0	Price	Area	Room	Lon	Lat
count	924.000000	9.200000e+02	924.000000	924.000000	924.000000	924.000000
mean	462.500000	6.220654e+05	95.952381	3.571429	4.888605	52.363326
std	266.880123	5.389942e+05	57.447436	1.592332	0.053140	0.024028
min	1.000000	1.750000e+05	21.000000	1.000000	4.644819	52.291519
25%	231.750000	3.500000e+05	60.750000	3.000000	4.855834	52.352077
50%	462.500000	4.670000e+05	83.000000	3.000000	4.886818	52.364631
75%	693.250000	7.000000e+05	113.000000	4.000000	4.922337	52.377598
max	924.000000	5.950000e+06	623.000000	14.000000	5.029122	52.423805

```
df.dtypes
```

```
Unnamed: 0      int64
Address        object
Zip            object
Price         float64
Area            int64
Room            int64
Lon           float64
Lat           float64
dtype: object
```

```
df.isnull().sum()
```

```
Unnamed: 0    0
Address       0
Zip           0
Price         4
Area          0
Room          0
Lon           0
Lat           0
dtype: int64
```

Data Visualization

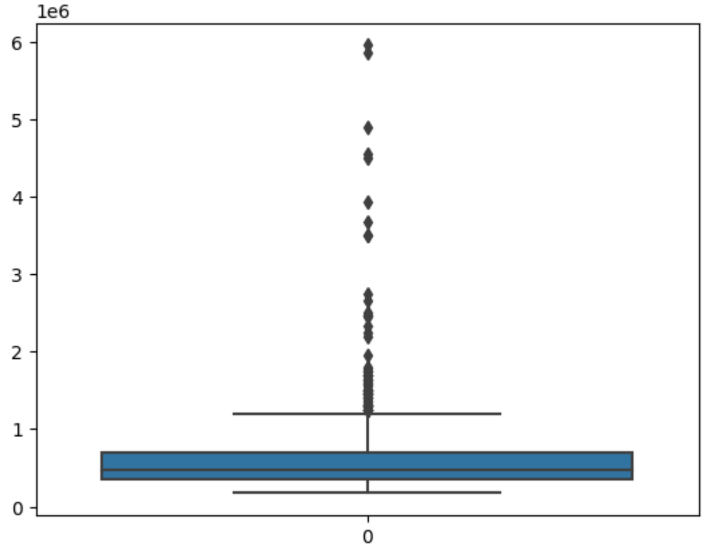
```
df.drop(['Unnamed: 0','Zip','Address'],axis=1,inplace=True)
df
```

	Price	Area	Room	Lon	Lat
0	685000.0	64	3	4.907736	52.356157
1	475000.0	60	3	4.850476	52.348586
2	850000.0	109	4	4.944774	52.343782
3	580000.0	128	6	4.789928	52.343712
4	720000.0	138	5	4.902503	52.410538
...	...	...	...	...	...
919	750000.0	117	1	4.927757	52.354173
920	350000.0	72	3	4.890612	52.414587
921	350000.0	51	3	4.856935	52.363256
922	599000.0	113	4	4.965731	52.375268
923	300000.0	79	4	4.810678	52.355493

924 rows × 5 columns

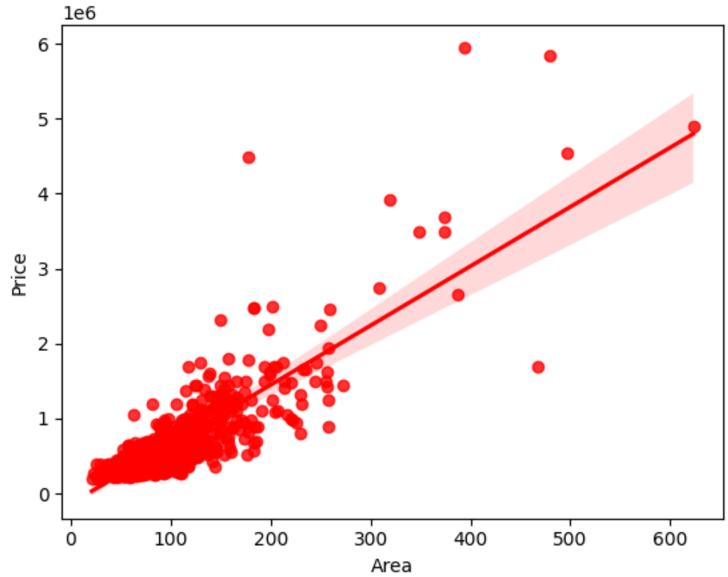
```
sns.boxplot(df['Price'])
```

```
<Axes: >
```



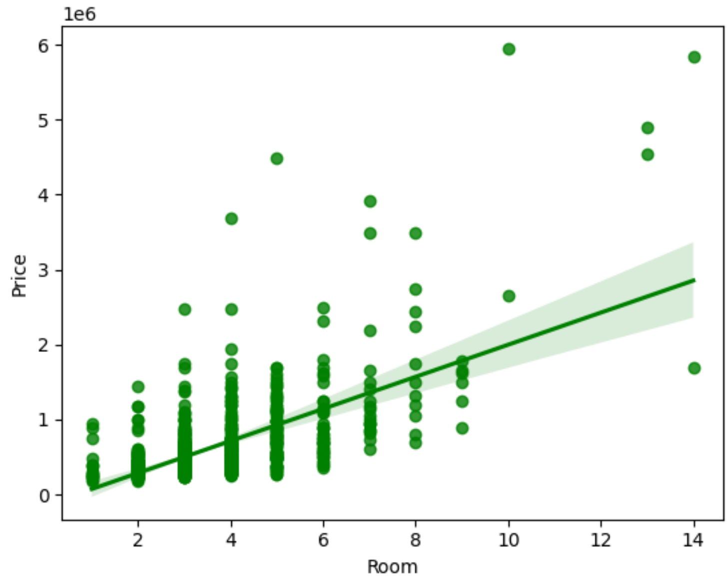

```
sns.regplot(x=df['Area'],y=df['Price'],color='r')
```

```
<Axes: xlabel='Area', ylabel='Price'>
```



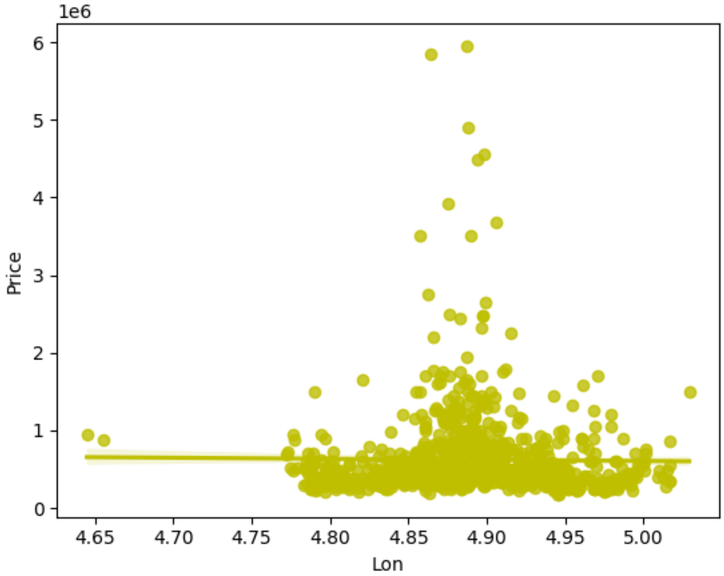

```
sns.regplot(x=df['Room'],y=df['Price'],color='g')
```

```
<Axes: xlabel='Room', ylabel='Price'>
```



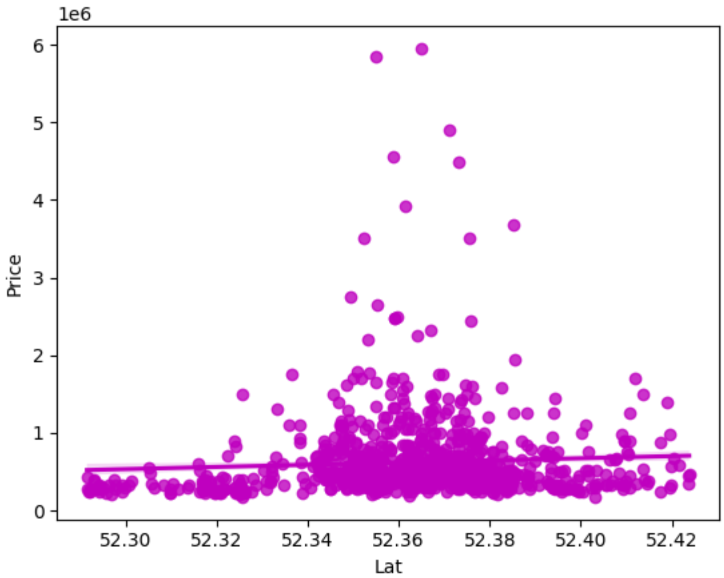
```
sns.regplot(x=df['Lon'],y=df['Price'],color='y')
```

```
<Axes: xlabel='Lon', ylabel='Price'>
```




```
sns.regplot(x=df['Lat'],y=df['Price'],color='m')
```

```
<Axes: xlabel='Lat', ylabel='Price'>
```

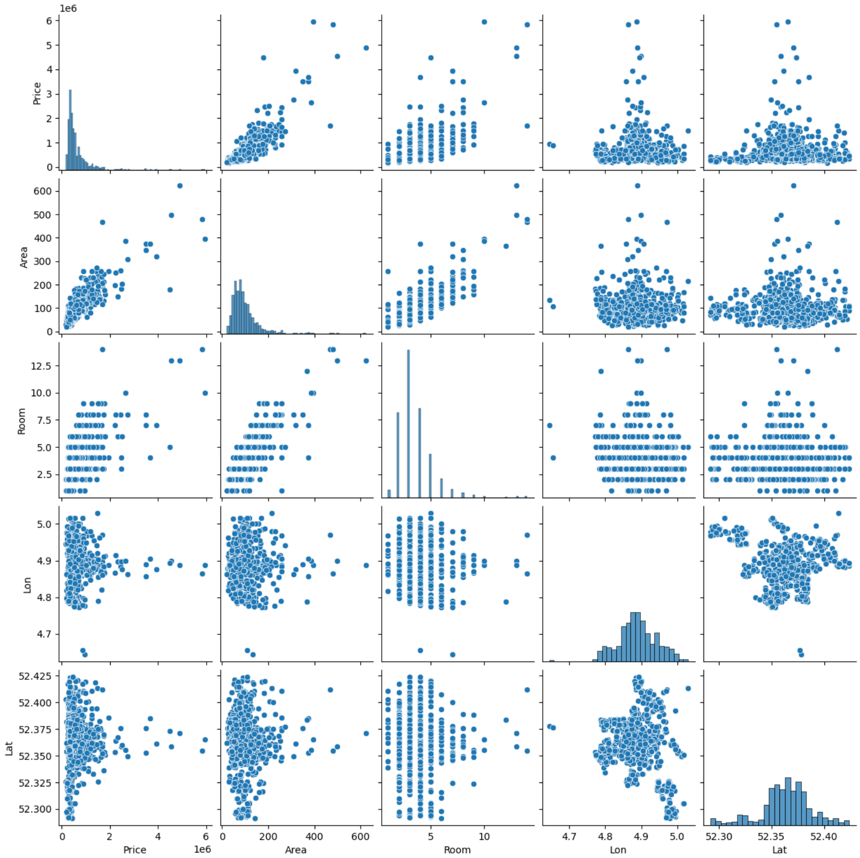



```
sns.countplot(x='Area',hue='Room',data=df)
```

```
<Axes: xlabel='Area', ylabel='count'>
```

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7da293948c10>
```

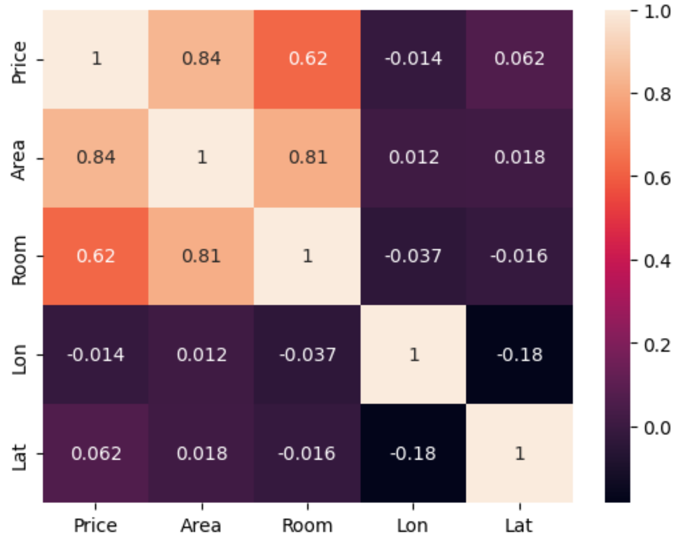



```
sns.distplot(df['Price'])
```

```
<ipython-input-19-87e11caeb2c4>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Price'])
<Axes: xlabel='Price', ylabel='Density'>
```


```
sns.heatmap(df.corr(),annot=True)
```

```
<Axes: >
```



Data Preprocessing

```
plt.hist(df['Price'])
```

```
(array([729., 136.,  36.,   7.,   3.,   2.,   2.,   2.,   1.,   2.]),
 array([ 175000.,  752500., 1330000., 1907500., 2485000., 3062500.,
        3640000.,  4217500.,  4795000.,  5372500.,  5950000. ]),
```

```
df['Price'].fillna(df['Price'].median(),inplace=True)
```

```
df.isnull().sum()
```

```
Price    0
Area     0
Room     0
Lon      0
Lat      0
dtype: int64
```

```
x=df.drop(['Price'],axis=1)
y=df['Price']
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x_train)
x_train=scaler.transform(x_train)
x_test=scaler.transform(x_test)
```

Model Creation using Linear Regression

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
y_pred
```

```
 418999.83901778,  433729.3365759 ,   92322.49716398,
1056791.91177767,  217662.22451742, 1037749.9423757 ,
 336701.87791902, 1092701.91916406, 1024349.20605896,
 531266.70180507,  768437.14618923,  367343.29826624,
1135639.04512495, 1655542.54657242,  806076.79601244,
 720155.0044908 ,  721185.39793928,  309740.94210572,
 556985.56570351, 1085021.24924702,  365863.32206416,
 252239.02913629,  393586.05706971,  335750.22341682,
1272135.06630752,  462288.09139105,  400332.20466805,
 417554.22636245,  433983.49033683,  526761.67350884,
 856435.85149899,  992480.3826116 ,  550944.35849641,
 352183.39472115,  441318.4194798 ,  856323.83701208,
 459035.51953363,  429387.52592299,  330693.13213096,
1410008.41337104,  338538.15036367,  507597.32048498,
1146863.00773898,  608446.99050516,  534005.58117469,
 567106.33615369,  631259.4310401 ,  620620.47083643,
 508122.71812145, 1147592.6619758 ,  765653.71490697,
 696557.41950745,  352965.05721431, 1478845.566259  ,
 348481.69325476,  525626.80935878,  460201.77294935,
1771756.87158417,  258989.45084816,  633474.99086347,
 786842.96490485,  210616.44330539,  261247.05863206,
 588578.38671764,  636611.06439448, 1533201.78847212,
 710704.95852672,  156141.91742531,  157951.67503939,
 924451.93799586,  549225.76237149,  962685.47623512,
 914361.11431845,  665638.79878457,  304801.92003789,
 894704.89176944,  329852.58093301,  445855.59475204,
 510900.15400457,  303998.45073847,  614857.5235804 ,
 270588.81307479,  507701.2302193 ,  148516.07098119,
 316807.27903403,  588525.55234003,  744393.63937814,
 687740.97327401,  570572.4820381 ,  428156.1027189 ,
 951513.67053915,  656053.8177622 ,  454169.41810156,
1229580.11465593,  403802.04260728, 1108387.92495231,
 602892.48101901,  269685.59971123,  925842.60203085,
 305035.57228267,  489291.94725163,  405799.52082079,
 725658.38042041,  270406.65788875,  301464.68817434,
 413003.41115469,  531831.97274425,  784540.0367422 ,
 889286.32604338,  457095.70589811,  386321.45386654,
 287027.87636036,  748001.39681837, 4519751.19257104,
 870631.60205292,  440284.28610974, 1218709.15011781,
1488746.47601056,  771288.13771025,  605324.4027997 ,
 609165.08570592, 1371749.56943239,  287730.12492489,
 588972.19485336, 1119983.68887885, 1745603.50643764,
 386450.09637594,  769014.0577339 ,  282746.40286645,
 594625.83322407,  373817.8961457 ,  615466.91496409,
```

```
       874532.22310162,  444407.63562659,  286428.47579987,
       244896.43938609,  667048.06641301, 2300472.98268151,
      1969572.06603795,  327963.12478113,  593352.70645819,
       568400.62979847,  258097.77219236,  382836.94823732,
       511591.52341752,  750543.2005787 ,  444148.1499956 ,
       355810.78069627,  378146.72078774, 1018275.97544878,
       411314.65039329,  808335.64879981,  219126.41707932,
       878224.46635309,  511059.7760042 ,  359544.38209601,
       484231.46417276,  194404.93236226,  348009.80395552,
       409611.07685271,  397569.68160476])
```

Model Evaluation

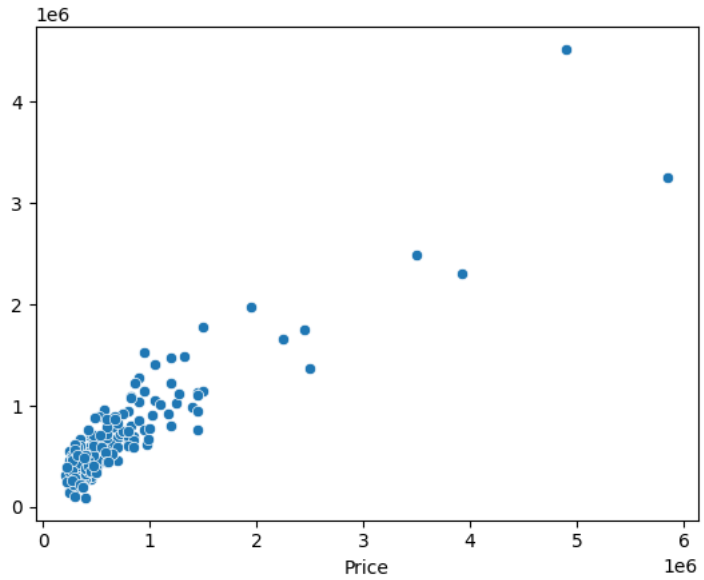
```
df1=pd.DataFrame({'actualvalue':y_test,'predicted_value':y_pred,'error':y_test-y_pred})
df1
```

	actualvalue	predicted_value	error
323	450000.0	6.950501e+05	-2.450501e+05
861	475000.0	3.752984e+05	9.970160e+04
30	800000.0	6.537566e+05	1.462434e+05
837	5850000.0	3.255480e+06	2.594520e+06
294	700000.0	4.659076e+05	2.340924e+05
...	...	...	...
54	385000.0	4.842315e+05	-9.923146e+04
827	375000.0	1.944049e+05	1.805951e+05
490	500000.0	3.480098e+05	1.519902e+05
753	475000.0	4.096111e+05	6.538892e+04
843	225000.0	3.975697e+05	-1.725697e+05

185 rows × 3 columns

```
sns.scatterplot(x=y_test,y=y_pred)
```

```
<Axes: xlabel='Price'>
```




```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
print("Mean Absolute Error is",mean_absolute_error(y_test,y_pred))
```

```
Mean Absolute Error is 188764.28934550055
```

```
mse=mean_squared_error(y_test,y_pred)
print("Mean Squared Error is",mse)
```

```
Mean Squared Error is 103467772655.20413
```

```
rmse=np.sqrt(mse)
print("Root Mean Squared Error",rmse)
```

```
Root Mean Squared Error 321664.068020045
```

```
print("r2_score is",r2_score(y_test,y_pred))
```

```
r2_score is 0.7941224508887771
```