House Price Prediction in India

The purpose of the project is to predict the sale prices of houses in different cities of India. Given many features from these cities.

Import libraries

In [1]:

```
#import the generic libraries required
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import style
from matplotlib.gridspec import GridSpec
import seaborn as sns
from scipy import stats
from fast_ml.feature_selection import get_constant_features

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import sklearn.metrics as metrics

from pandas import read_csv, set_option
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split, cross_val_score

sns.set()
style.use('fivethirtyeight')
pd.options.mode.chained_assignment = None
```

Read the both train and test dataset

In [2]:

```
#get the input dataset
train_data = pd.read_csv("C:/Users/KIIT/Documents/3rd Year/6th SEM/T&T Lab/Project/train.cs
train_data
```

Out[2]:

	ID	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUAR
0	2200001	Owner	0	0	2	BHK	1300.2
1	2200002	Dealer	0	0	2	BHK	1275.0
2	2200003	Owner	0	0	2	BHK	933.1
3	2200004	Owner	0	1	2	BHK	929.9
4	2200005	Dealer	1	0	2	BHK	999.0
...	...	...	...	...	...	...	
29445	2229446	Owner	0	0	2	BHK	1062.1
29446	2229447	Owner	0	0	3	BHK	2500.0
29447	2229448	Owner	0	0	2	BHK	769.2
29448	2229449	Dealer	0	0	2	BHK	1022.6
29449	2229450	Owner	0	0	2	BHK	927.0

29450 rows × 13 columns

In [3]:

```
#read the test data
test_data = pd.read_csv("C:/Users/KIIT/Documents/3rd Year/6th SEM/T&T Lab/Project/test.csv"
test_data
```

Out[3]:

	ID	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUAI
0	2229451	Owner	0	0	1	BHK	545.1
1	2229452	Dealer	1	1	2	BHK	800.0
2	2229453	Dealer	0	0	2	BHK	1257.0
3	2229454	Dealer	0	0	3	BHK	1400.3
4	2229455	Owner	0	0	1	BHK	430.4
...	...	...	...	...	...	...	
68715	2298166	Dealer	0	1	2	BHK	856.5
68716	2298167	Dealer	0	1	3	BHK	2304.1
68717	2298168	Dealer	1	1	1	BHK	33362.7
68718	2298169	Dealer	0	0	2	BHK	1173.7
68719	2298170	Dealer	0	0	3	BHK	2439.5

68720 rows × 12 columns

Storing the testdata into a different dataset for BACKUP

In [4]:

```
test_data1 = test_data.copy()
```

Display the 5 records from traindata and testdata dataframes

In [5]:

```
train_data.head()
```

Out[5]:

	ID	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUARE_FT
0	2200001	Owner	0	0	2	BHK	1300.23640
1	2200002	Dealer	0	0	2	BHK	1275.00000
2	2200003	Owner	0	0	2	BHK	933.15972
3	2200004	Owner	0	1	2	BHK	929.92114
4	2200005	Dealer	1	0	2	BHK	999.00924

In [6]:

```
test_data.head()
```

Out[6]:

	ID	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUARE_FT
0	2229451	Owner	0	0	1	BHK	545.17134
1	2229452	Dealer	1	1	2	BHK	800.00000
2	2229453	Dealer	0	0	2	BHK	1257.09651
3	2229454	Dealer	0	0	3	BHK	1400.32948
4	2229455	Owner	0	0	1	BHK	430.47783

Check the Detail information of the both dataframe

In [7]:

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29450 entries, 0 to 29449
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   ID                     29450 non-null  int64  
 1   POSTED_BY              29450 non-null  object 
 2   UNDER_CONSTRUCTION     29450 non-null  int64  
 3   RERA                   29450 non-null  int64  
 4   BHK_NO.                29450 non-null  int64  
 5   BHK_OR_RK              29450 non-null  object 
 6   SQUARE_FT              29450 non-null  float64
 7   READY_TO_MOVE          29450 non-null  int64  
 8   RESALE                 29450 non-null  int64  
 9   ADDRESS                29450 non-null  object 
 10  LONGITUDE              29450 non-null  float64
 11  LATITUDE               29450 non-null  float64
 12  TARGET(PRICE_IN_LACS)  29450 non-null  float64
dtypes: float64(4), int64(6), object(3)
memory usage: 2.9+ MB
```

In [8]:

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 68720 entries, 0 to 68719
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  68720 non-null  int64  
 1   POSTED_BY           68720 non-null  object 
 2   UNDER_CONSTRUCTION  68720 non-null  int64  
 3   RERA                68720 non-null  int64  
 4   BHK_NO.             68720 non-null  int64  
 5   BHK_OR_RK           68720 non-null  object 
 6   SQUARE_FT           68720 non-null  float64
 7   READY_TO_MOVE       68720 non-null  int64  
 8   RESALE              68720 non-null  int64  
 9   ADDRESS             68720 non-null  object 
 10  LONGITUDE           68720 non-null  float64
 11  LATITUDE            68720 non-null  float64
dtypes: float64(3), int64(6), object(3)
memory usage: 6.3+ MB
```

Check the number of Rows and Columns for both the dataframes

In [9]:

```
print("Number of rows and columns of train dataset: ", train_data.shape)
print("\nNumber of rows and columns of test dataset: ", test_data.shape)
```

```
Number of rows and columns of train dataset:  (29450, 13)

Number of rows and columns of test dataset:  (68720, 12)
```

Display All the column names for both the dataframes

In [10]:

```
print("Column names are:\n", train_data.columns)
print("\nColumn names are:\n", test_data.columns)
```

```
Column names are:
 Index(['ID', 'POSTED_BY', 'UNDER_CONSTRUCTION', 'RERA', 'BHK_NO.', 'BHK_OR_
RK',
       'SQUARE_FT', 'READY_TO_MOVE', 'RESALE', 'ADDRESS', 'LONGITUDE',
       'LATITUDE', 'TARGET(PRICE_IN_LACS)'],
      dtype='object')

Column names are:
 Index(['ID', 'POSTED_BY', 'UNDER_CONSTRUCTION', 'RERA', 'BHK_NO.', 'BHK_OR_
RK',
       'SQUARE_FT', 'READY_TO_MOVE', 'RESALE', 'ADDRESS', 'LONGITUDE',
       'LATITUDE'],
      dtype='object')
```

Describe the train dataset

In [11]:

```
train_data.describe
```

Out[11]:

```
<bound method NDFrame.describe of             ID POSTED_BY  UNDER_CONSTRUCTI
ON  RERA  BHK_NO. BHK_OR_RK  \
0      2200001     Owner                  0     0        2       BHK
1      2200002    Dealer                  0     0        2       BHK
2      2200003     Owner                  0     0        2       BHK
3      2200004     Owner                  0     1        2       BHK
4      2200005    Dealer                  1     0        2       BHK
...        ...       ...                ...   ...      ...       ...
29445  2229446     Owner                  0     0        2       BHK
29446  2229447     Owner                  0     0        3       BHK
29447  2229448     Owner                  0     0        2       BHK
29448  2229449    Dealer                  0     0        2       BHK
29449  2229450     Owner                  0     0        2       BHK

         SQUARE_FT  READY_TO_MOVE  RESALE                           ADDRESS
\
0      1300.236407              1       1             Ksfc Layout,Bangalore
1      1275.000000              1       1          Vishweshwara Nagar,Mysore
2       933.159722              1       1                  Jigani,Bangalore
3       929.921143              1       1        Sector-1 Vaishali,Ghaziabad
4       999.009247              0       1                  New Town,Kolkata
...            ...            ...     ...                               ...
29445  1062.134891              1       1                 Tilakwadi,Belgaum
29446  2500.000000              1       1              Shamshabad Road,Agra
29447   769.230769              1       1  E3-108, Lake View Recidency,,Vapi
29448  1022.641509              1       1                 Ajmer Road,Jaipur
29449   927.079009              1       1            Sholinganallur,Chennai

       LONGITUDE   LATITUDE  TARGET(PRICE_IN_LACS)
0      12.969910  77.597960                   55.0
1      12.274538  76.644605                   51.0
2      12.778033  77.632191                   43.0
3      28.642300  77.344500                   62.5
4      22.592200  88.484911                   60.5
...          ...        ...                    ...
29445  15.866670  74.500000                   40.0
29446  27.140626  78.043277                   45.0
29447  39.945409 -86.150721                   16.0
29448  26.928785  75.828002                   27.1
29449  12.900150  80.227910                   67.0

[29450 rows x 13 columns]>
```

In [12]:

```
#to infer the details by just lookig at this
train_data.describe()
```

Out[12]:

	ID	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	REAI
count	2.945000e+04	29450.000000	29450.000000	29450.000000	2.945000e+04	
mean	2.214726e+06	0.179762	0.317895	2.392292	1.980281e+04	
std	8.501627e+03	0.383996	0.465666	0.879103	1.901367e+06	
min	2.200001e+06	0.000000	0.000000	1.000000	3.000000e+00	
25%	2.207363e+06	0.000000	0.000000	2.000000	9.000219e+02	
50%	2.214726e+06	0.000000	0.000000	2.000000	1.175058e+03	
75%	2.222088e+06	0.000000	1.000000	3.000000	1.550738e+03	
max	2.229450e+06	1.000000	1.000000	20.000000	2.545455e+08	

Display each column wise total number of record for traindata

In [13]:

```
train_data["ID"].value_counts()
```

Out[13]:

```
2200001    1
2219643    1
2219641    1
2219640    1
2219639    1
          ..
2209814    1
2209813    1
2209812    1
2209811    1
2229450    1
Name: ID, Length: 29450, dtype: int64
```

In [14]:

```
train_data["POSTED_BY"].value_counts()
```

Out[14]:

```
Dealer     18290
Owner      10538
Builder      622
Name: POSTED_BY, dtype: int64
```

In [15]:

```
train_data["UNDER_CONSTRUCTION"].value_counts()
```

Out[15]:

```
0    24156
1     5294
Name: UNDER_CONSTRUCTION, dtype: int64
```

In [16]:

```
train_data["RERA"].value_counts()
```

Out[16]:

```
0    20088
1     9362
Name: RERA, dtype: int64
```

In [17]:

```
train_data["BHK_NO."].value_counts()
```

Out[17]:

```
2     13323
3     10546
1      3574
4      1723
5       190
6        52
7        11
8        10
20        4
10        4
15        4
12        3
9         3
13        1
17        1
11        1
Name: BHK_NO., dtype: int64
```

In [18]:

```
train_data["BHK_OR_RK"].value_counts()
```

Out[18]:

```
BHK    29426
RK        24
Name: BHK_OR_RK, dtype: int64
```

In [19]:

```
train_data["SQUARE_FT"].value_counts()
```

Out[19]:

```
1000.000000    479
1250.000000    294
800.000000     202
1200.000000    179
1600.000000    125
              ...
1248.285322      1
1685.097420      1
1085.176085      1
1300.142248      1
927.079009       1
Name: SQUARE_FT, Length: 19560, dtype: int64
```

In [20]:

```
train_data["READY_TO_MOVE"].value_counts()
```

Out[20]:

```
1    24156
0     5294
Name: READY_TO_MOVE, dtype: int64
```

In [21]:

```
train_data["RESALE"].value_counts()
```

Out[21]:

```
1    27376
0     2074
Name: RESALE, dtype: int64
```

In [22]:

```
train_data["ADDRESS"].value_counts()
```

Out[22]:

```
Zirakpur,Chandigarh                    509
Whitefield,Bangalore                   230
Raj Nagar Extension,Ghaziabad          215
Sector-137 Noida,Noida                 139
New Town,Kolkata                       131
                                      ...
Ambika Township,Jivarajpark,Rajkot       1
Cheranalloor,Kochi                       1
Baba Nagar,Bangalore                     1
Barra Devi Naubasta road,Kanpur          1
E3-108, Lake View Recidency,,Vapi        1
Name: ADDRESS, Length: 6899, dtype: int64
```

In [23]:

```
train_data["LONGITUDE"].value_counts()
```

Out[23]:

```
24.690280    1009
12.969910     671
30.662283     509
22.541110     479
19.058710     242
             ...
27.292573       1
23.389986       1
17.827647       1
27.882520       1
39.945409       1
Name: LONGITUDE, Length: 4087, dtype: int64
```

In [24]:

```
train_data["LATITUDE"].value_counts()
```

Out[24]:

```
 78.418890    1009
 77.597960     671
 76.822397     509
 88.337780     479
 72.899690     242
              ...
 85.289842       1
 82.231207       1
 79.915810       1
 83.400211       1
-86.150721       1
Name: LATITUDE, Length: 4078, dtype: int64
```

In [25]:

```
train_data["TARGET(PRICE_IN_LACS)"].value_counts()
```

Out[25]:

```
110.0      795
100.0      770
120.0      652
130.0      598
45.0       583
          ...
86.1         1
1550.0       1
28000.0      1
9910.0       1
18.3         1
Name: TARGET(PRICE_IN_LACS), Length: 1172, dtype: int64
```

Data Cleaning

Display the Null values percentage against every columns (compare to the total number of records) for the traindata

In [26]:

```
#to check if there are any null values present or not, if there are no null values in the d
total = train_data.isnull().sum().sort_values(ascending=False)
percent = (train_data.isnull().mean()*100).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data
```

Out[26]:

	Total	Percent
ID	0	0.0
POSTED_BY	0	0.0
UNDER_CONSTRUCTION	0	0.0
RERA	0	0.0
BHK_NO.	0	0.0
BHK_OR_RK	0	0.0
SQUARE_FT	0	0.0
READY_TO_MOVE	0	0.0
RESALE	0	0.0
ADDRESS	0	0.0
LONGITUDE	0	0.0
LATITUDE	0	0.0
TARGET(PRICE_IN_LACS)	0	0.0

Display the number of unique elements present in each column for the traindata

In [27]:

```
count_uniques = train_data.nunique(axis=0)
count_uniques
```

Out[27]:

```
ID                        29450
POSTED_BY                     3
UNDER_CONSTRUCTION            2
RERA                          2
BHK_NO.                      16
BHK_OR_RK                     2
SQUARE_FT                 19560
READY_TO_MOVE                 2
RESALE                        2
ADDRESS                    6899
LONGITUDE                  4087
LATITUDE                   4078
TARGET(PRICE_IN_LACS)      1172
dtype: int64
```

Check the number of records present in target column and display the shape for traindata

In [28]:

```
train_data["TARGET(PRICE_IN_LACS)"].count()
```

Out[28]:

```
29450
```

In [29]:

```
train_data.shape
```

Out[29]:

```
(29450, 13)
```

Display the Number of Duplicate Rows

In [30]:

```
train_data.duplicated().sum()
```

Out[30]:

```
0
```

In [31]:

```
test_data.duplicated().sum()
```

Out[31]:

```
0
```

Drop all the Duplicate Rows

In [32]:

```
train_data.drop_duplicates(inplace=True)
train_data.shape
```

Out[32]:

```
(29450, 13)
```

In [33]:

```
test_data.drop_duplicates(inplace=True)
test_data.shape
```

Out[33]:

```
(68720, 12)
```

Split the column and extract only the city name

In [34]:

```
train_data["ADDRESS"] = train_data["ADDRESS"].apply(lambda x: x.split(',')[-1])
train_data
```

Out[34]:

	ID	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUAR
0	2200001	Owner	0	0	2	BHK	1300.2
1	2200002	Dealer	0	0	2	BHK	1275.0
2	2200003	Owner	0	0	2	BHK	933.1
3	2200004	Owner	0	1	2	BHK	929.9
4	2200005	Dealer	1	0	2	BHK	999.0
...	...	...	...	...	...	...	
29445	2229446	Owner	0	0	2	BHK	1062.1
29446	2229447	Owner	0	0	3	BHK	2500.0
29447	2229448	Owner	0	0	2	BHK	769.2
29448	2229449	Dealer	0	0	2	BHK	1022.6
29449	2229450	Owner	0	0	2	BHK	927.0

29450 rows × 13 columns

In [35]:

```
train_data["ADDRESS"].value_counts()
```

Out[35]:

```
Bangalore      4340
Lalitpur       2993
Mumbai         2023
Pune           1991
Noida          1767
               ...
Bhadrak           1
Kurukshetra       1
Dibrugarh         1
Sagar             1
Washim            1
Name: ADDRESS, Length: 256, dtype: int64
```

In [36]:

```
test_data["ADDRESS"] = test_data["ADDRESS"].apply(lambda x: x.split(',')[-1])
test_data
```

Out[36]:

	ID	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUAI
0	2229451	Owner	0	0	1	BHK	545.1
1	2229452	Dealer	1	1	2	BHK	800.0
2	2229453	Dealer	0	0	2	BHK	1257.0
3	2229454	Dealer	0	0	3	BHK	1400.3
4	2229455	Owner	0	0	1	BHK	430.4
...	...	...	...	...	...	...	
68715	2298166	Dealer	0	1	2	BHK	856.5
68716	2298167	Dealer	0	1	3	BHK	2304.1
68717	2298168	Dealer	1	1	1	BHK	33362.7
68718	2298169	Dealer	0	0	2	BHK	1173.7
68719	2298170	Dealer	0	0	3	BHK	2439.5

68720 rows × 12 columns

In [37]:

```
test_data["ADDRESS"].value_counts()
```

Out[37]:

```
Bangalore    10001
Lalitpur      7070
Pune          4600
Mumbai        4516
Kolkata       4141
             ...
Kaithal          1
Harda            1
Theni            1
Morena           1
Shimoga          1
Name: ADDRESS, Length: 293, dtype: int64
```

check the columns with constant and quasi-constant variable in the both dataframes

In [38]:

```
get_constant_features(train_data)
```

Out[38]:

	Desc	Var	Value	Perc
0	Quasi Constant	BHK_OR_RK	BHK	99.918506

In [39]:

```
get_constant_features(test_data)
```

Out[39]:

	Desc	Var	Value	Perc
0	Quasi Constant	BHK_OR_RK	BHK	99.9156

Drop the columns with constant and quasi-constant variable from the both dataframes

In [40]:

```
train_data.drop(columns="BHK_OR_RK", inplace=True)
```

In [41]:

```
test_data.drop(columns="BHK_OR_RK", inplace=True)
```

Check for the number of Rows and Columns in your dataset in the both dataframes

In [42]:

```
print(train_data.shape)
print(test_data.shape)
```

```
(29450, 12)
(68720, 11)
```

Splitting of Train and the Test Data

Look for all columns containing null value

In [43]:

```
train_data.isnull().sum() > 0
```

Out[43]:

```
ID                        False
POSTED_BY                 False
UNDER_CONSTRUCTION        False
RERA                      False
BHK_NO.                   False
SQUARE_FT                 False
READY_TO_MOVE             False
RESALE                    False
ADDRESS                   False
LONGITUDE                 False
LATITUDE                  False
TARGET(PRICE_IN_LACS)     False
dtype: bool
```

In [44]:

```
test_data.isnull().sum() > 0
```

Out[44]:

```
ID                    False
POSTED_BY             False
UNDER_CONSTRUCTION    False
RERA                  False
BHK_NO.               False
SQUARE_FT             False
READY_TO_MOVE         False
RESALE                False
ADDRESS               False
LONGITUDE             False
LATITUDE              False
dtype: bool
```

Split the "traindata" columns into X and y dataframe

In [45]:

```
y = train_data[['TARGET(PRICE_IN_LACS)']].copy()
y.shape
```

Out[45]:

```
(29450, 1)
```

In [46]:

```
X = train_data.copy()
X.drop(columns = 'TARGET(PRICE_IN_LACS)', inplace=True)
X.shape
```

Out[46]:

```
(29450, 11)
```

Split both the dataframes into train and test format in 60:40 ratio

In [47]:

```
X_train, X_loc_test, y_train, y_loc_test = train_test_split(X, y, test_size=.40, shuffle=Fa
```

Check for the number of rows and columns of all the new dataframes (all 4)

In [48]:

```
print(X_train.shape)
print(X_loc_test.shape)
print(y_train.shape)
print(y_loc_test.shape)
```

```
(17670, 11)
(11780, 11)
(17670, 1)
(11780, 1)
```

Split the "X_loc_test" and "y_loc_test" dataset into "Test" and "Validation" dataframe with 50:50 format

In [49]:

```
X_val, X_test, y_val, y_test = train_test_split(X_loc_test, y_loc_test, test_size=.50, shuf
```

Check for the number of rows and columns of all the 4 dataframes

In [50]:

```
print(X_val.shape)
print(X_test.shape)
print(y_val.shape)
print(y_test.shape)
```

```
(5890, 11)
(5890, 11)
(5890, 1)
(5890, 1)
```

Exploratory Data Analysis(EDA)

Histogram Plot (use the dataframe which contains the target field)

In [51]:

```
#plotting the histogram
train_data.hist(figsize = (25, 25))
```

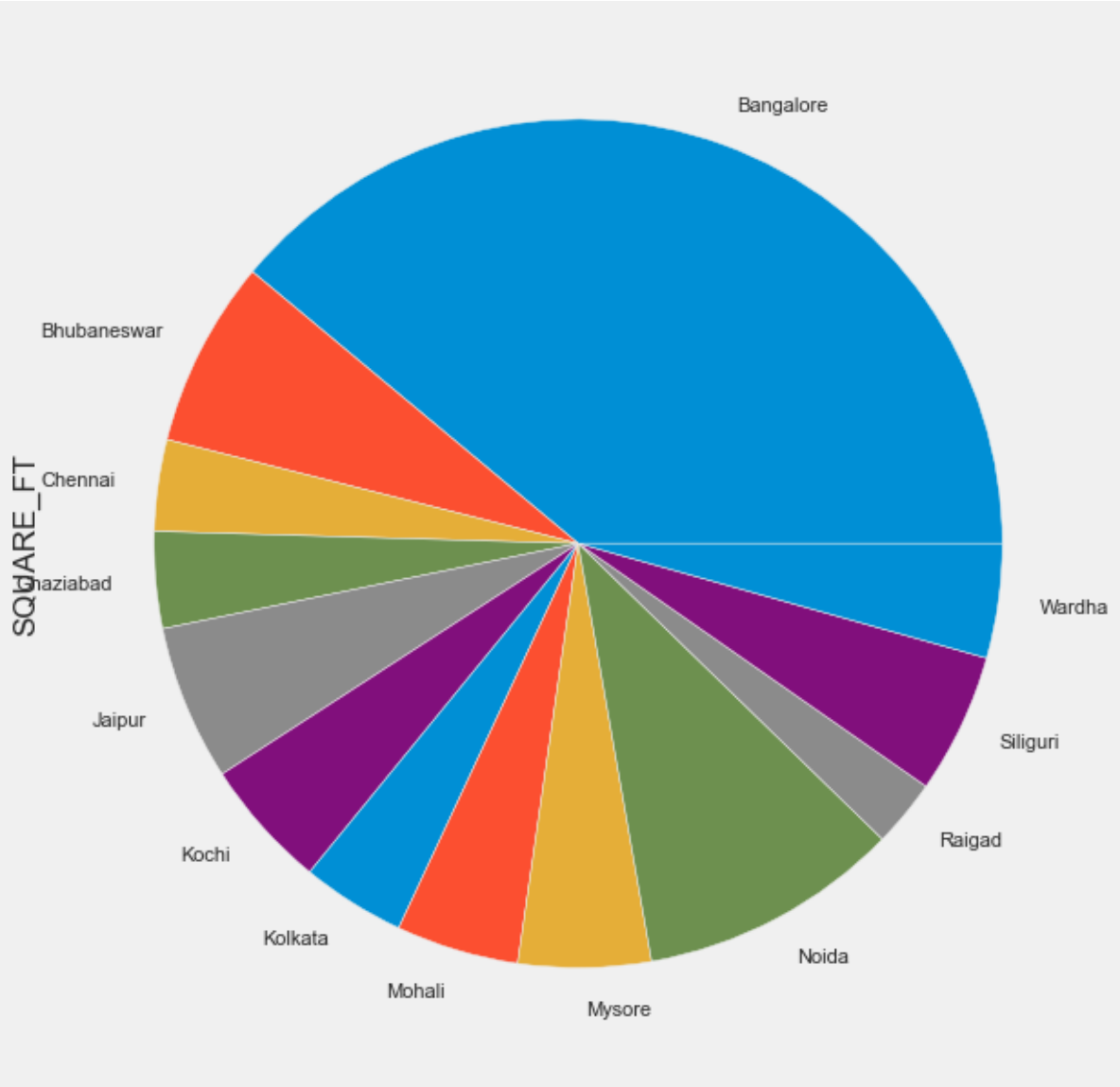
Out[51]:

```
array([[<AxesSubplot:title={'center':'ID'}>,
        <AxesSubplot:title={'center':'UNDER_CONSTRUCTION'}>,
        <AxesSubplot:title={'center':'RERA'}>],
       [<AxesSubplot:title={'center':'BHK_NO.'}>,
        <AxesSubplot:title={'center':'SQUARE_FT'}>,
        <AxesSubplot:title={'center':'READY_TO_MOVE'}>],
       [<AxesSubplot:title={'center':'RESALE'}>,
        <AxesSubplot:title={'center':'LONGITUDE'}>,
        <AxesSubplot:title={'center':'LATITUDE'}>],
       [<AxesSubplot:title={'center':'TARGET(PRICE_IN_LACS)'}>,
        <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```

Pie Graph of the target variable (use the dataframe which contains the target field)

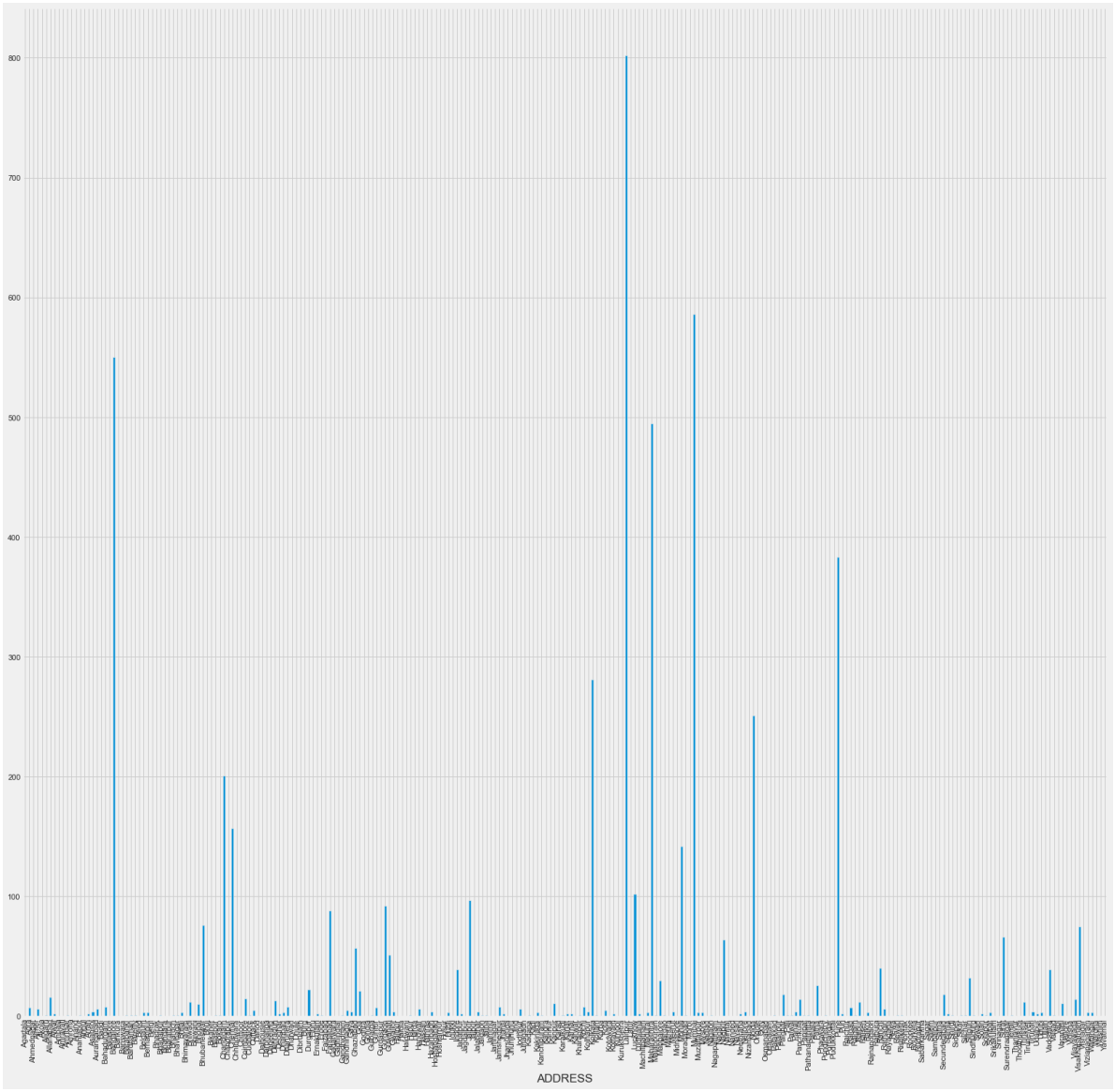
In [52]:

```
#plotting pie graph for the cities based on square_ft of houses
plt.figure(figsize = (10, 10))
train_data.head(20).groupby(["ADDRESS"]).sum().SQUARE_FT.plot(kind = "pie")
plt.show()
```



In [53]:

```
#plotting bar diagram for the cities based on square_ft of houses
plt.figure(figsize = (25, 25))
train_data.groupby(["ADDRESS"]).sum().UNDER_CONSTRUCTION.plot(kind = "bar")
plt.show()
```



In [54]:

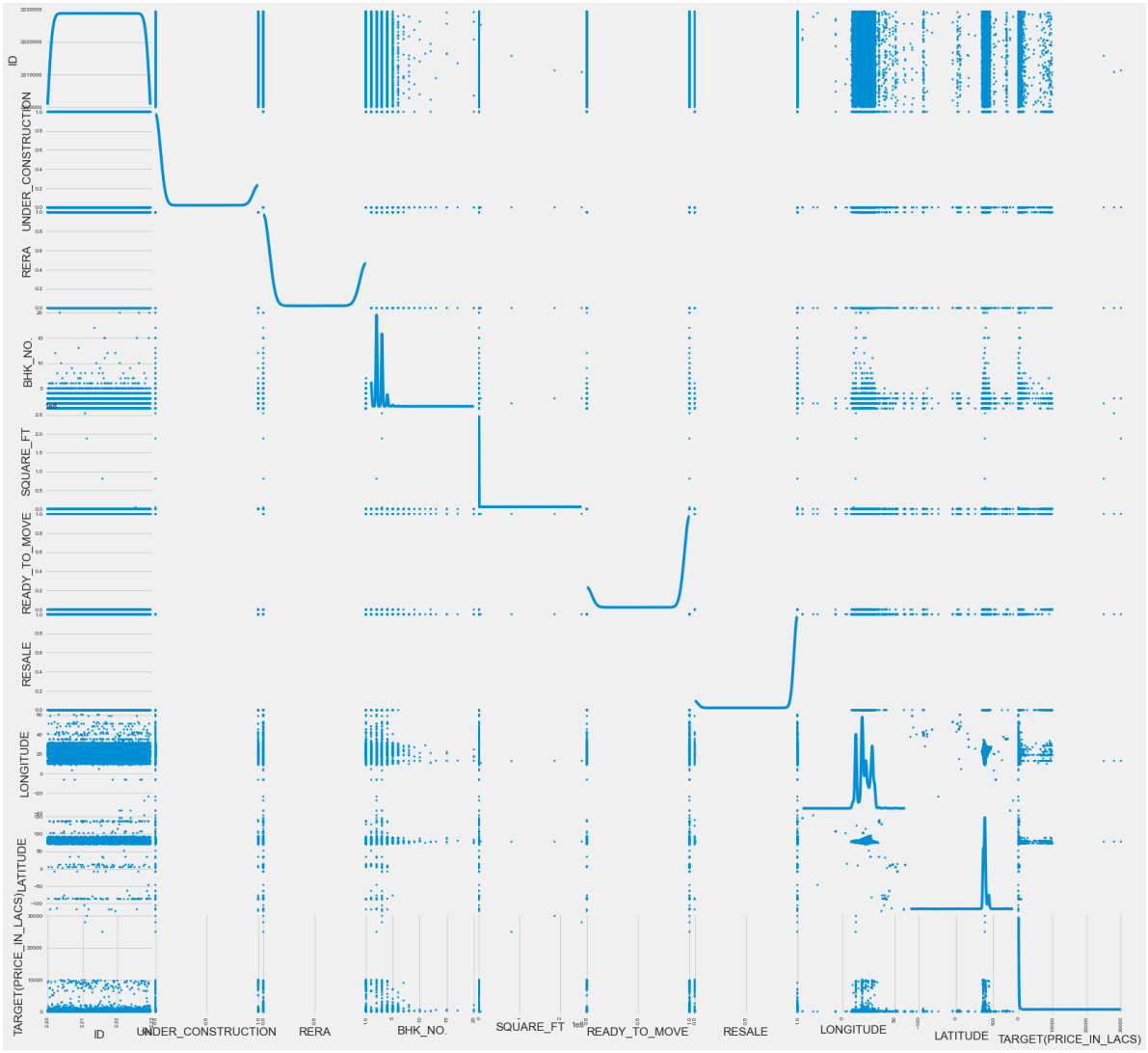
```
#display scatter matrix for all variables
train_fig = plt.figure()
scatter_matrix(train_data, figsize = (25, 25), alpha = 1.0, diagonal = "kde", marker = ".")
```

Out[54]:

```
array([[<AxesSubplot:xlabel='ID', ylabel='ID'>,
        <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='ID'>,
        <AxesSubplot:xlabel='RERA', ylabel='ID'>,
        <AxesSubplot:xlabel='BHK_NO.', ylabel='ID'>,
        <AxesSubplot:xlabel='SQUARE_FT', ylabel='ID'>,
        <AxesSubplot:xlabel='READY_TO_MOVE', ylabel='ID'>,
        <AxesSubplot:xlabel='RESALE', ylabel='ID'>,
        <AxesSubplot:xlabel='LONGITUDE', ylabel='ID'>,
        <AxesSubplot:xlabel='LATITUDE', ylabel='ID'>,
        <AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='ID'>],
       [<AxesSubplot:xlabel='ID', ylabel='UNDER_CONSTRUCTION'>,
        <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='UNDER_CONSTRUCTI
ON'>,
        <AxesSubplot:xlabel='RERA', ylabel='UNDER_CONSTRUCTION'>,
        <AxesSubplot:xlabel='BHK_NO.', ylabel='UNDER_CONSTRUCTION'>,
        <AxesSubplot:xlabel='SQUARE_FT', ylabel='UNDER_CONSTRUCTION'>,
        <AxesSubplot:xlabel='READY_TO_MOVE', ylabel='UNDER_CONSTRUCTION'>,
        <AxesSubplot:xlabel='RESALE', ylabel='UNDER_CONSTRUCTION'>,
        <AxesSubplot:xlabel='LONGITUDE', ylabel='UNDER_CONSTRUCTION'>,
        <AxesSubplot:xlabel='LATITUDE', ylabel='UNDER_CONSTRUCTION'>,
        <AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='UNDER_CONSTRU
CTION'>],
       [<AxesSubplot:xlabel='ID', ylabel='RERA'>,
        <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='RERA'>,
        <AxesSubplot:xlabel='RERA', ylabel='RERA'>,
        <AxesSubplot:xlabel='BHK_NO.', ylabel='RERA'>,
        <AxesSubplot:xlabel='SQUARE_FT', ylabel='RERA'>,
        <AxesSubplot:xlabel='READY_TO_MOVE', ylabel='RERA'>,
        <AxesSubplot:xlabel='RESALE', ylabel='RERA'>,
        <AxesSubplot:xlabel='LONGITUDE', ylabel='RERA'>,
        <AxesSubplot:xlabel='LATITUDE', ylabel='RERA'>,
        <AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='RERA'>],
       [<AxesSubplot:xlabel='ID', ylabel='BHK_NO.'>,
        <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='BHK_NO.'>,
        <AxesSubplot:xlabel='RERA', ylabel='BHK_NO.'>,
        <AxesSubplot:xlabel='BHK_NO.', ylabel='BHK_NO.'>,
        <AxesSubplot:xlabel='SQUARE_FT', ylabel='BHK_NO.'>,
        <AxesSubplot:xlabel='READY_TO_MOVE', ylabel='BHK_NO.'>,
        <AxesSubplot:xlabel='RESALE', ylabel='BHK_NO.'>,
        <AxesSubplot:xlabel='LONGITUDE', ylabel='BHK_NO.'>,
        <AxesSubplot:xlabel='LATITUDE', ylabel='BHK_NO.'>,
        <AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='BHK_NO.'>],
       [<AxesSubplot:xlabel='ID', ylabel='SQUARE_FT'>,
        <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='SQUARE_FT'>,
        <AxesSubplot:xlabel='RERA', ylabel='SQUARE_FT'>,
        <AxesSubplot:xlabel='BHK_NO.', ylabel='SQUARE_FT'>,
        <AxesSubplot:xlabel='SQUARE_FT', ylabel='SQUARE_FT'>,
        <AxesSubplot:xlabel='READY_TO_MOVE', ylabel='SQUARE_FT'>,
        <AxesSubplot:xlabel='RESALE', ylabel='SQUARE_FT'>,
        <AxesSubplot:xlabel='LONGITUDE', ylabel='SQUARE_FT'>,
        <AxesSubplot:xlabel='LATITUDE', ylabel='SQUARE_FT'>,
        <AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='SQUARE_FT'>],
       [<AxesSubplot:xlabel='ID', ylabel='READY_TO_MOVE'>,
```

```
        <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='READY_TO_MOVE'>,
        <AxesSubplot:xlabel='RERA', ylabel='READY_TO_MOVE'>,
        <AxesSubplot:xlabel='BHK_NO.', ylabel='READY_TO_MOVE'>,
        <AxesSubplot:xlabel='SQUARE_FT', ylabel='READY_TO_MOVE'>,
        <AxesSubplot:xlabel='READY_TO_MOVE', ylabel='READY_TO_MOVE'>,
        <AxesSubplot:xlabel='RESALE', ylabel='READY_TO_MOVE'>,
        <AxesSubplot:xlabel='LONGITUDE', ylabel='READY_TO_MOVE'>,
        <AxesSubplot:xlabel='LATITUDE', ylabel='READY_TO_MOVE'>,
        <AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='READY_TO_MOV
E'>],
       [<AxesSubplot:xlabel='ID', ylabel='RESALE'>,
        <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='RESALE'>,
        <AxesSubplot:xlabel='RERA', ylabel='RESALE'>,
        <AxesSubplot:xlabel='BHK_NO.', ylabel='RESALE'>,
        <AxesSubplot:xlabel='SQUARE_FT', ylabel='RESALE'>,
        <AxesSubplot:xlabel='READY_TO_MOVE', ylabel='RESALE'>,
        <AxesSubplot:xlabel='RESALE', ylabel='RESALE'>,
        <AxesSubplot:xlabel='LONGITUDE', ylabel='RESALE'>,
        <AxesSubplot:xlabel='LATITUDE', ylabel='RESALE'>,
        <AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='RESALE'>],
       [<AxesSubplot:xlabel='ID', ylabel='LONGITUDE'>,
        <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='LONGITUDE'>,
        <AxesSubplot:xlabel='RERA', ylabel='LONGITUDE'>,
        <AxesSubplot:xlabel='BHK_NO.', ylabel='LONGITUDE'>,
        <AxesSubplot:xlabel='SQUARE_FT', ylabel='LONGITUDE'>,
        <AxesSubplot:xlabel='READY_TO_MOVE', ylabel='LONGITUDE'>,
        <AxesSubplot:xlabel='RESALE', ylabel='LONGITUDE'>,
        <AxesSubplot:xlabel='LONGITUDE', ylabel='LONGITUDE'>,
        <AxesSubplot:xlabel='LATITUDE', ylabel='LONGITUDE'>,
        <AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='LONGITUDE'>],
       [<AxesSubplot:xlabel='ID', ylabel='LATITUDE'>,
        <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='LATITUDE'>,
        <AxesSubplot:xlabel='RERA', ylabel='LATITUDE'>,
        <AxesSubplot:xlabel='BHK_NO.', ylabel='LATITUDE'>,
        <AxesSubplot:xlabel='SQUARE_FT', ylabel='LATITUDE'>,
        <AxesSubplot:xlabel='READY_TO_MOVE', ylabel='LATITUDE'>,
        <AxesSubplot:xlabel='RESALE', ylabel='LATITUDE'>,
        <AxesSubplot:xlabel='LONGITUDE', ylabel='LATITUDE'>,
        <AxesSubplot:xlabel='LATITUDE', ylabel='LATITUDE'>,
        <AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='LATITUDE'>],
       [<AxesSubplot:xlabel='ID', ylabel='TARGET(PRICE_IN_LACS)'>,
        <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='TARGET(PRICE_IN_
LACS)'>,
        <AxesSubplot:xlabel='RERA', ylabel='TARGET(PRICE_IN_LACS)'>,
        <AxesSubplot:xlabel='BHK_NO.', ylabel='TARGET(PRICE_IN_LACS)'>,
        <AxesSubplot:xlabel='SQUARE_FT', ylabel='TARGET(PRICE_IN_LACS)'>,
        <AxesSubplot:xlabel='READY_TO_MOVE', ylabel='TARGET(PRICE_IN_LAC
S)'>,
        <AxesSubplot:xlabel='RESALE', ylabel='TARGET(PRICE_IN_LACS)'>,
        <AxesSubplot:xlabel='LONGITUDE', ylabel='TARGET(PRICE_IN_LACS)'>,
        <AxesSubplot:xlabel='LATITUDE', ylabel='TARGET(PRICE_IN_LACS)'>,
        <AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='TARGET(PRICE_
IN_LACS)'>]],
      dtype=object)
```

```
<Figure size 432x288 with 0 Axes>
```

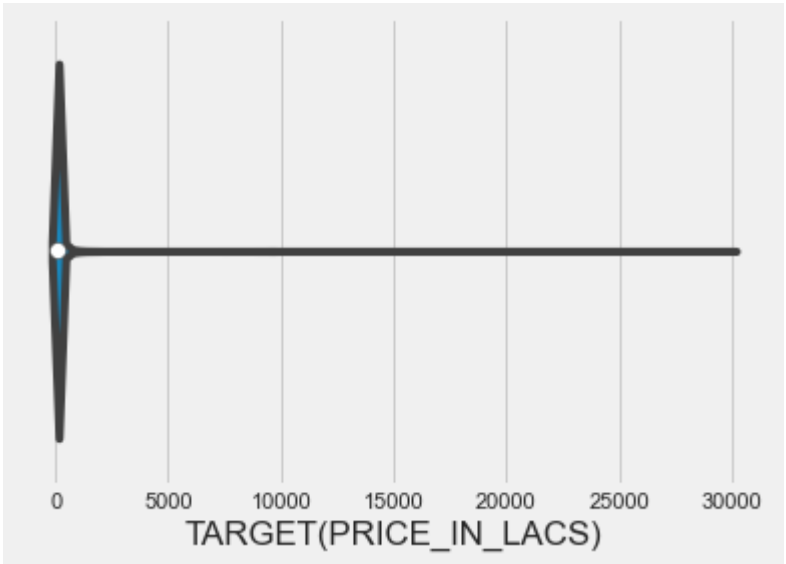


In [55]:

```
#plotting violin plot for the target column
sns.violinplot(data=train_data,x='TARGET(PRICE_IN_LACS)')
```

Out[55]:

```
<AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)'>
```

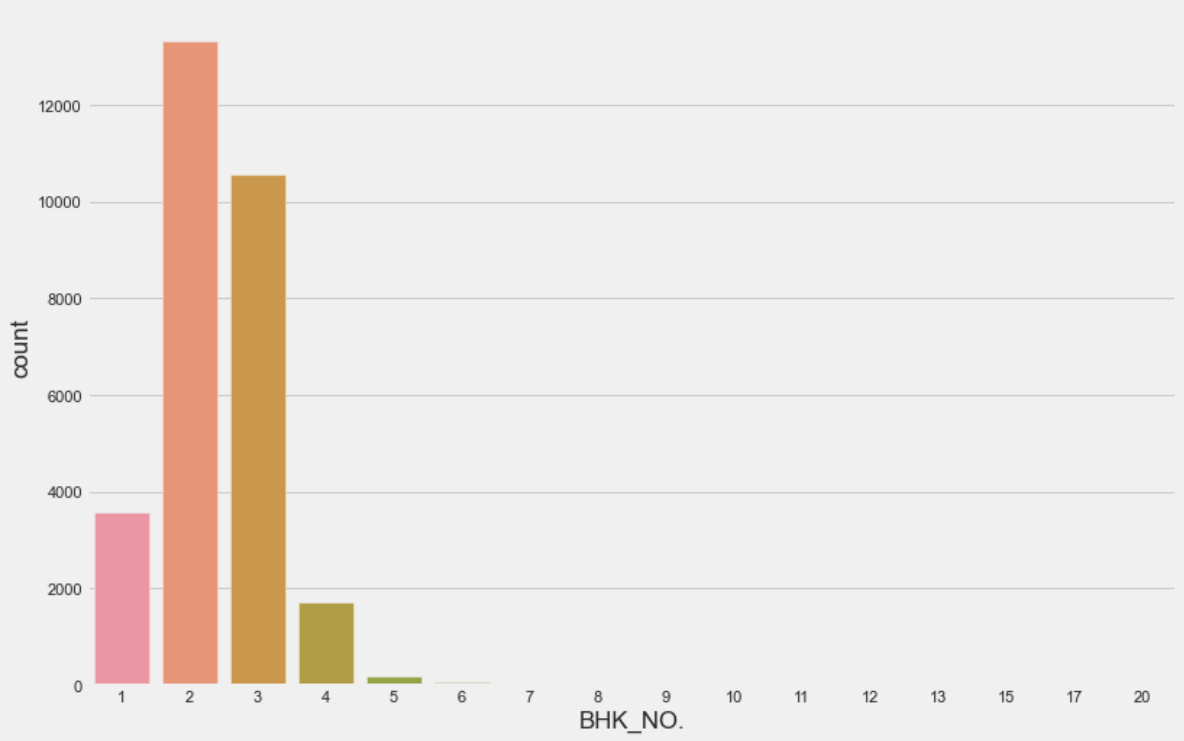


In [56]:

```
#plotting count plot for a specific column
fig = plt.figure(figsize=(12, 8))
sns.countplot(x=train_data['BHK_NO.'])
```

Out[56]:

```
<AxesSubplot:xlabel='BHK_NO.', ylabel='count'>
```



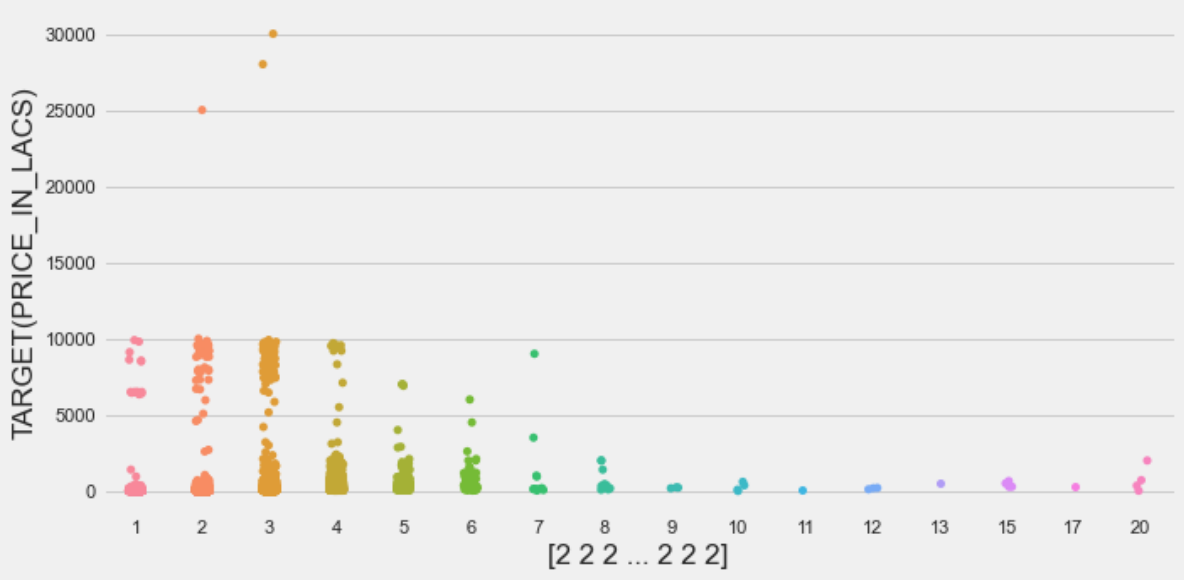
In [57]:

```
#plotting categorical plot for a specific column
sns.catplot(x=train_data["BHK_NO."].values, y='TARGET(PRICE_IN_LACS)', data=train_data, hei
```

```
C:\Users\KIIT\anaconda3\lib\site-packages\matplotlib\text.py:1215: FutureWar
ning: elementwise comparison failed; returning scalar instead, but in the fu
ture will perform elementwise comparison
  if s != self._text:
```

Out[57]:

```
<seaborn.axisgrid.FacetGrid at 0x13af7afaa30>
```



In [58]:

```
#plotting histogram for the target column
fig = plt.figure(figsize=(12, 8))
grid = GridSpec(ncols=1, nrows=2, figure=fig)

ax1 = fig.add_subplot(grid[0, :])
sns.histplot(train_data['TARGET(PRICE_IN_LACS)'], ax=ax1, kde=True)
```

Out[58]:

```
<AxesSubplot:xlabel='TARGET(PRICE_IN_LACS)', ylabel='Count'>
```

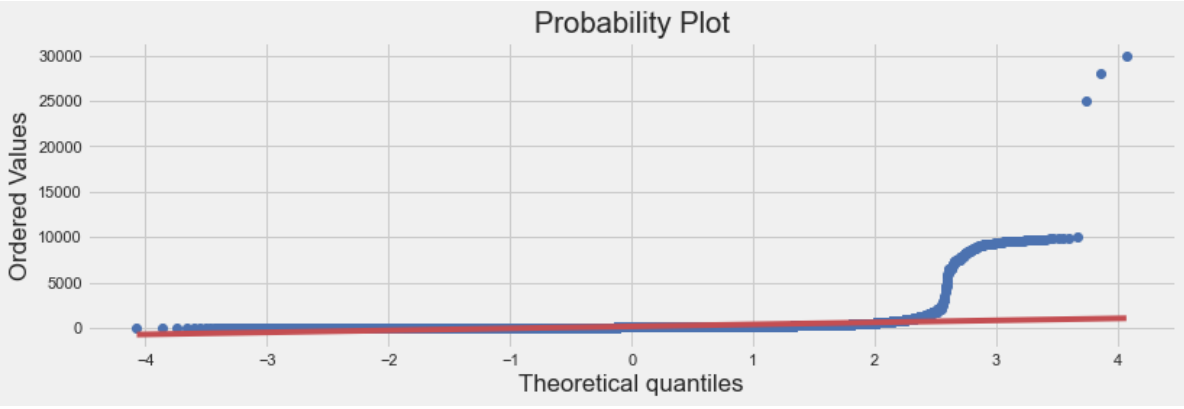
In [59]:

```
#plotting probablity plot for the target column
fig = plt.figure(figsize=(12, 8))
grid = GridSpec(ncols=1, nrows=2, figure=fig)

ax2 = fig.add_subplot(grid[1, :])
stats.probplot(train_data['TARGET(PRICE_IN_LACS)'], plot=ax2)
```

Out[59]:

```
((array([-4.0697072 , -3.85812536, -3.74253808, ...,  3.74253808,
          3.85812536,  4.0697072 ]),
  array([2.5e-01, 2.9e-01, 4.8e-01, ..., 2.5e+04, 2.8e+04, 3.0e+04])),
 (220.63309636584572, 142.90265398981333, 0.3358350641887042))
```



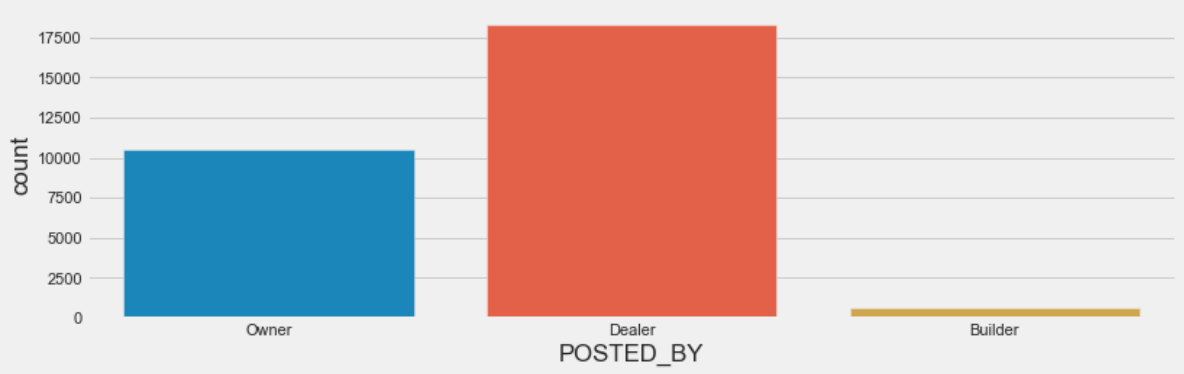
In [60]:

```
#plotting count plot for a specific column
fig = plt.figure(figsize=(12, 8))
grid = GridSpec(ncols=1, nrows=2, figure=fig)

ax3 = fig.add_subplot(grid[0, :])
sns.countplot(x=train_data["POSTED_BY"], ax=ax3)
```

Out[60]:

```
<AxesSubplot:xlabel='POSTED_BY', ylabel='count'>
```



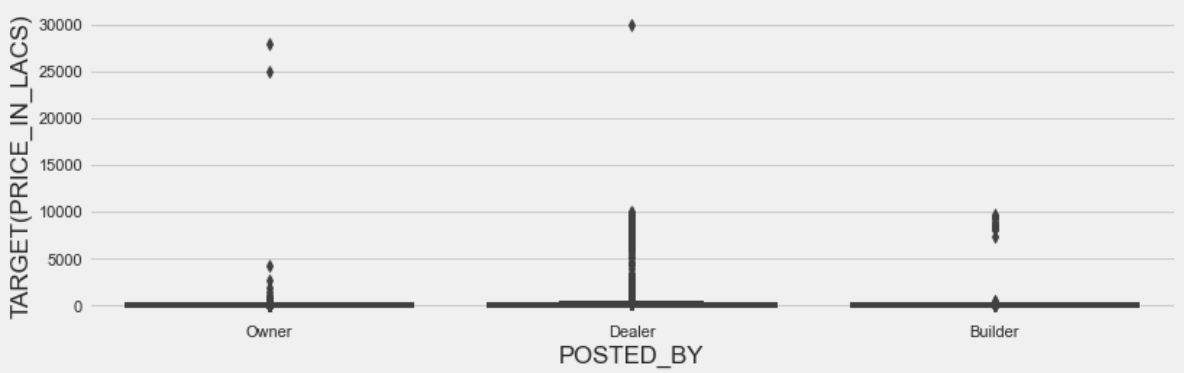
In [61]:

```
#plotting box plot for the target column based on by whom a house is posted
fig = plt.figure(figsize=(12, 8))
grid = GridSpec(ncols=1, nrows=2, figure=fig)

ax4 = fig.add_subplot(grid[1, :])
sns.boxplot(x=train_data["POSTED_BY"], y='TARGET(PRICE_IN_LACS)', data=train_data, ax=ax4)
```

Out[61]:

```
<AxesSubplot:xlabel='POSTED_BY', ylabel='TARGET(PRICE_IN_LACS)'>
```

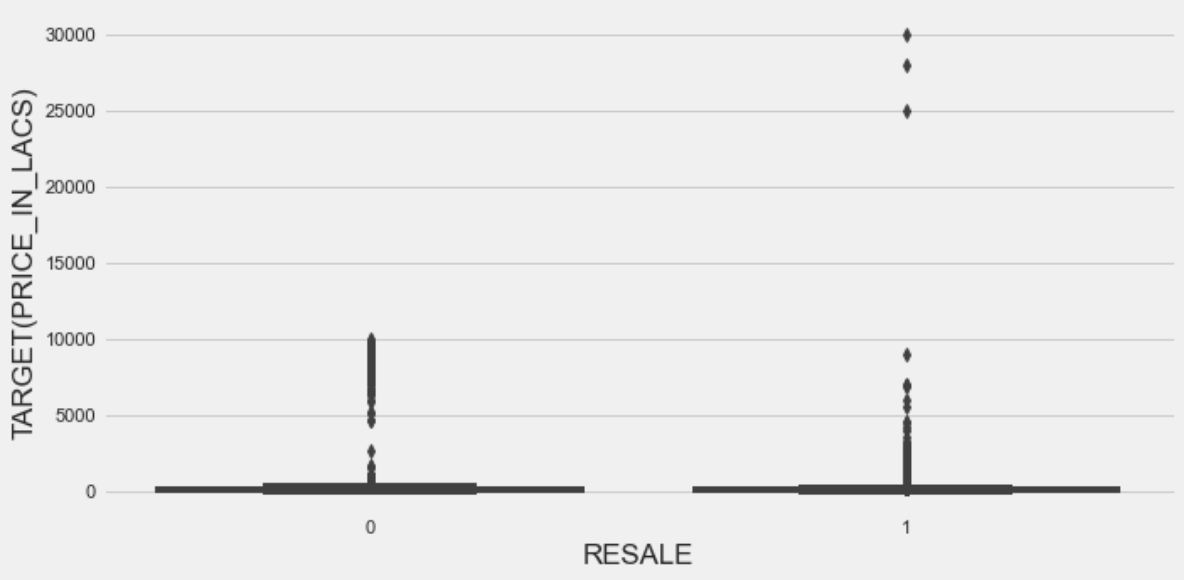


In [62]:

```
#plotting categorical-box plot for the target column based on resale of a house
sns.catplot(x='RESALE', y='TARGET(PRICE_IN_LACS)', data=train_data, kind='box', height=5, a
```

Out[62]:

```
<seaborn.axisgrid.FacetGrid at 0x13a85e4a8b0>
```

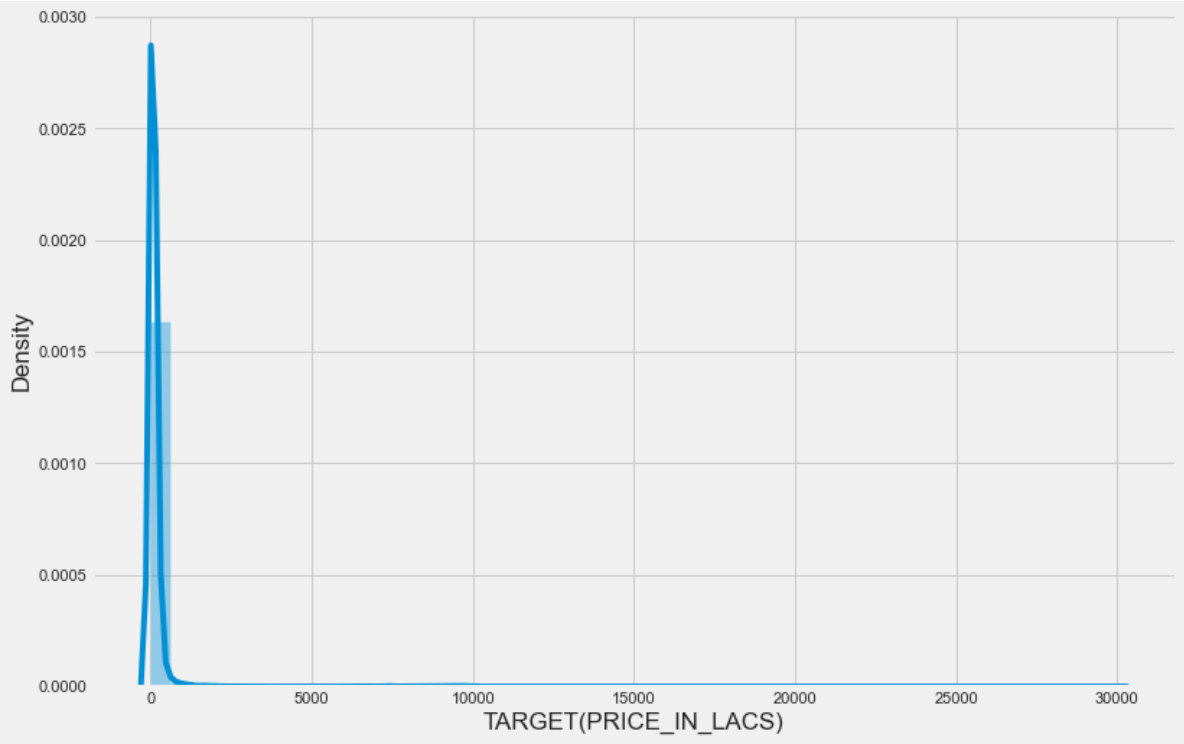


In [63]:

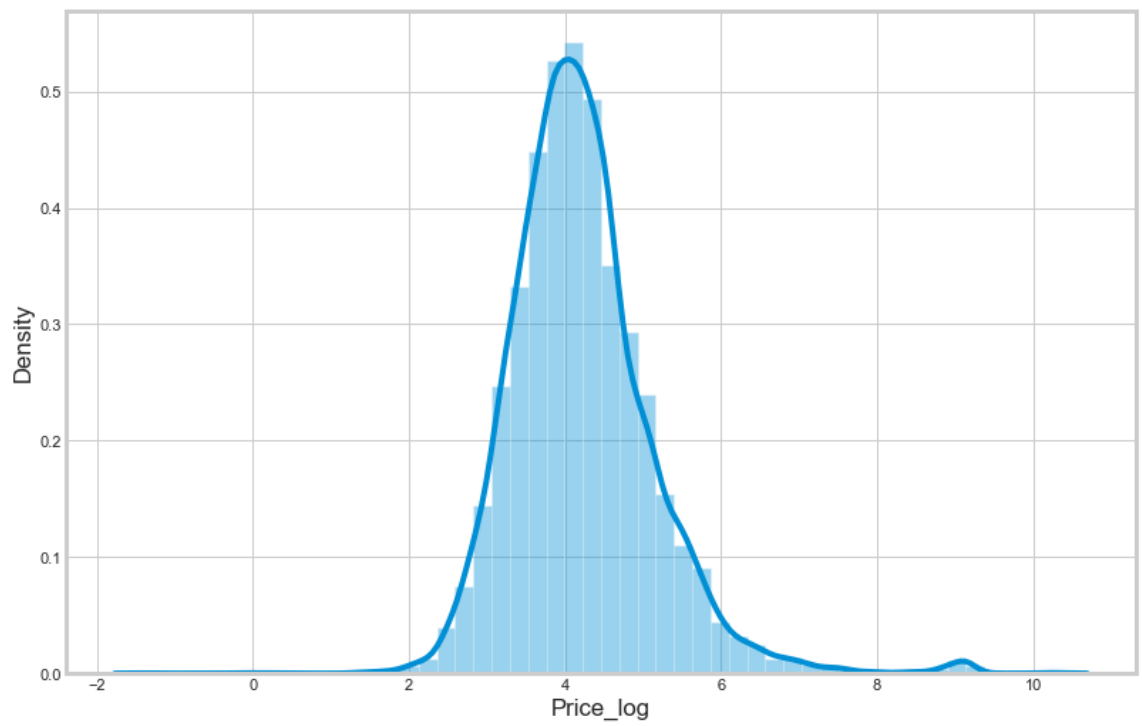
```
#plotting distribution plot for the target column from y_train and also for a new column th
plt.subplots(figsize=(12,8))
x = y_train["TARGET(PRICE_IN_LACS)"]
sns.set_style("whitegrid")
sns.distplot(x)
plt.show()

plt.subplots(figsize=(12,8))
y_train["Price_log"] = np.log(y_train["TARGET(PRICE_IN_LACS)"])
x = y_train.Price_log
sns.distplot(x)
plt.show()
```

```
C:\Users\KIIT\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a fut
ure version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\KIIT\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a fut
ure version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



In [64]:

```
y_train.drop(columns="Price_log", inplace=True)
```

Data Analysis using sweetviz library

In [65]:

```
import sweetviz as sv
report = sv.analyze(train_data)
report.show_html("./group10_report.html")
```

Done! Use 'show' commands to display/save. [100%] 00:00 -> (00:00 left)

```
Report ./group10_report.html was generated! NOTEBOOK/COLAB USERS: the web br
owser MAY not pop up, regardless, the report IS saved in your notebook/colab
files.
```

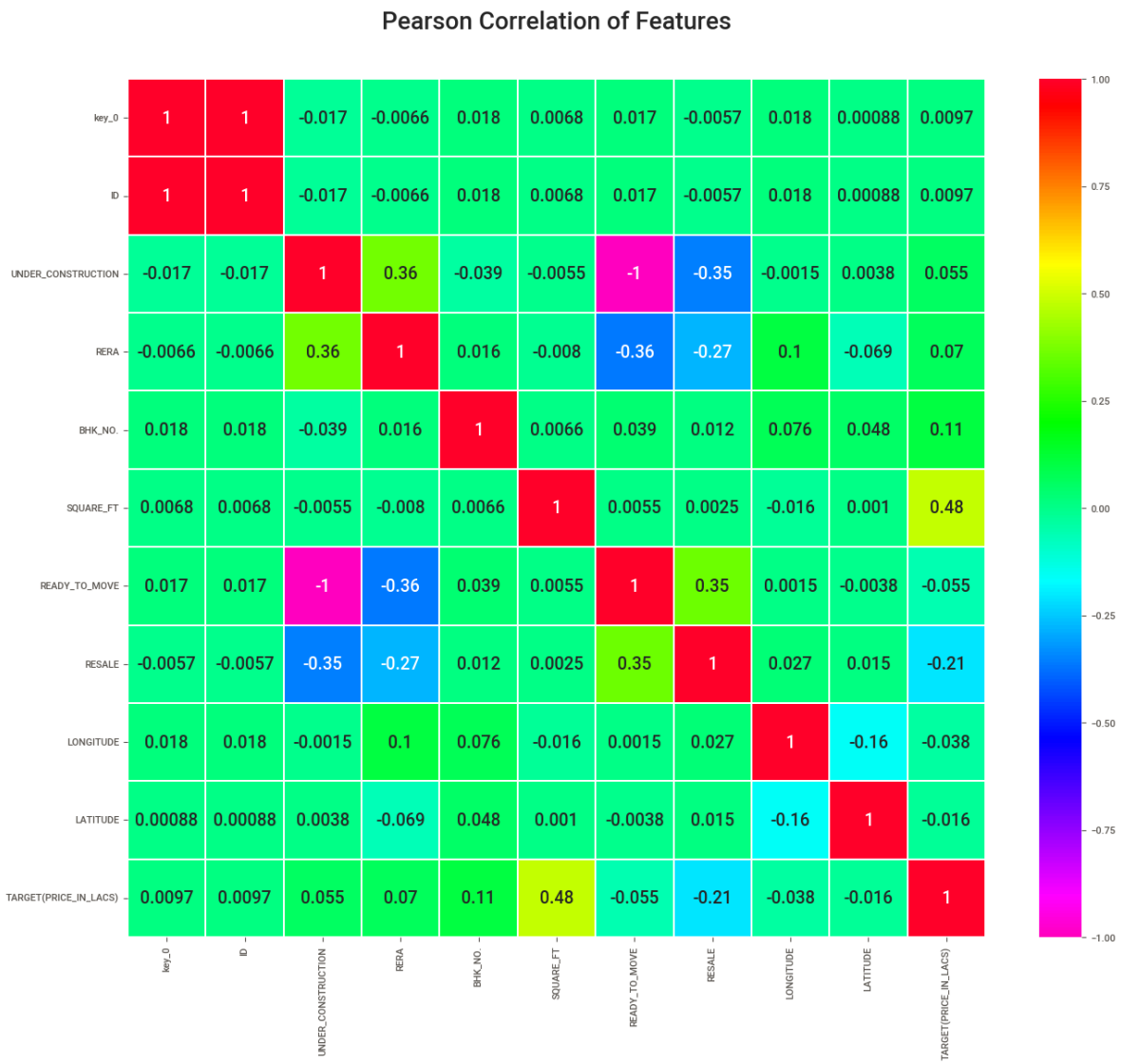
Plotting Heatmap for X_train based on correlation

In [66]:

```
colormap = plt.cm.RdBu
plt.figure(figsize=(14,12))
plt.title('Pearson Correlation of Features', y=1.05, size=20)
sns.heatmap(X_train.merge(y_train , on = X_train.index ).corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap='gist_rainbow_r', linecolor='white', annot=True)
```

Out[66]:

```
<AxesSubplot:title={'center':'Pearson Correlation of Features'}>
```



Feature Engineering

Display and describe the X_train dataframe

In [67]:

```
X_train.head()
```

Out[67]:

	ID	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	READY_TO_
0	2200001	Owner	0	0	2	1300.236407	
1	2200002	Dealer	0	0	2	1275.000000	
2	2200003	Owner	0	0	2	933.159722	
3	2200004	Owner	0	1	2	929.921143	
4	2200005	Dealer	1	0	2	999.009247	

In [68]:

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17670 entries, 0 to 17669
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  17670 non-null  int64  
 1   POSTED_BY           17670 non-null  object 
 2   UNDER_CONSTRUCTION  17670 non-null  int64  
 3   RERA                17670 non-null  int64  
 4   BHK_NO.             17670 non-null  int64  
 5   SQUARE_FT           17670 non-null  float64
 6   READY_TO_MOVE       17670 non-null  int64  
 7   RESALE              17670 non-null  int64  
 8   ADDRESS             17670 non-null  object 
 9   LONGITUDE           17670 non-null  float64
 10  LATITUDE            17670 non-null  float64
dtypes: float64(3), int64(6), object(2)
memory usage: 1.6+ MB
```

The "ADDRESS" column inside X_train, is a categorical column, so it is needed to perform Labelencoder on that particular column

In [69]:

```
from sklearn.preprocessing import LabelEncoder
address_coder = LabelEncoder()
address_coder.fit(X_train['ADDRESS'])
```

Out[69]:

```
LabelEncoder()
```

Store the value into a new column i.e. "ADDRESS_enc"

In [70]:

```
X_train['ADDRESS_enc'] = address_coder.fit_transform(X_train['ADDRESS'])
```

In [71]:

```
X_val['ADDRESS_enc'] = address_coder.fit_transform(X_val['ADDRESS'])
X_test['ADDRESS_enc'] = address_coder.fit_transform(X_test['ADDRESS'])
```

Display "ADDRESS" and "ADDRESS_enc" together from X_train dataframe

In [72]:

```
X_train[['ADDRESS', 'ADDRESS_enc']]
```

Out[72]:

	ADDRESS	ADDRESS_enc
0	Bangalore	19
1	Mysore	144
2	Bangalore	19
3	Ghaziabad	67
4	Kolkata	119
...	...	...
17665	Noida	156
17666	Faridabad	62
17667	Kolkata	119
17668	Bangalore	19
17669	Jaipur	91

17670 rows × 2 columns

Create a function called "drop_col" for dropping the columns 'ADDRESS' from train, test and validation dataframe

In [73]:

```
def drop_col(col, traindf = X_train, valdf = X_val, testdf = X_test):
    traindf.drop(col, axis =1,inplace=True)
    valdf.drop(col,axis=1 , inplace=True)
    testdf.drop(col,axis=1 , inplace=True)

    return traindf, valdf, testdf
```

Call the function by passing the column name which needed to be dropped from train, test and validation dataframes. Return updated dataframes to be stored in X_train , X_val, X_test

In [74]:

```
X_train, X_val, X_test = drop_col(['ADDRESS'])
```

It differs from LabelEncoder by handling new classes and providing a value for it [Unknown]. Unknown will be added in fit and transform will take care of new item. It gives unknown class id.

This will fit the encoder for all the unique values and introduce unknown value

In [75]:

```
#For encoding unseen labels
class EncoderExt(object):
    def __init__(self):
        self.label_encoder = LabelEncoder()
    def fit(self, data_list):
        self.label_encoder = self.label_encoder.fit(list(data_list) + ['Unknown'])
        self.classes_ = self.label_encoder.classes_
        return self
    def transform(self, data_list):
        new_data_list = list(data_list)
        for unique_item in np.unique(data_list):
            if unique_item not in self.label_encoder.classes_:
                new_data_list = ['Unknown' if x==unique_item else x for x in new_data_list]
        return self.label_encoder.transform(new_data_list)
```

Use the user define Label Encoder function called "EncoderExt" for the "POSTED_BY" column

In [76]:

```
label_encoder = EncoderExt()
label_encoder.fit(X_train['POSTED_BY'])
X_train['POSTED_BY_enc']=label_encoder.transform(X_train['POSTED_BY'])
X_val['POSTED_BY_enc']=label_encoder.transform(X_val['POSTED_BY'])
X_test['POSTED_BY_enc']=label_encoder.transform(X_test['POSTED_BY'])
```

Display "POSTED_BY" and "POSTED_BY_enc" together from X_train

dataframe

In [77]:

```
X_train[["POSTED_BY","POSTED_BY_enc"]]
```

Out[77]:

	POSTED_BY	POSTED_BY_enc
0	Owner	2
1	Dealer	1
2	Owner	2
3	Owner	2
4	Dealer	1
...	...	...
17665	Owner	2
17666	Dealer	1
17667	Owner	2
17668	Owner	2
17669	Owner	2

17670 rows × 2 columns

As we have created the a new column "POSTED_BY_enc", so now drop "POSTED_BY" column from all three dataframes

In [78]:

```
X_train, X_val, X_test = drop_col(['POSTED_BY'])
```

Check the datatype of all the columns of Train, Test and Validation dataframes realted to X

In [79]:

```
X_train.dtypes
```

Out[79]:

```
ID                      int64
UNDER_CONSTRUCTION      int64
RERA                    int64
BHK_NO.                 int64
SQUARE_FT             float64
READY_TO_MOVE           int64
RESALE                  int64
LONGITUDE             float64
LATITUDE              float64
ADDRESS_enc             int32
POSTED_BY_enc           int32
dtype: object
```

In [80]:

```
X_test.dtypes
```

Out[80]:

```
ID                      int64
UNDER_CONSTRUCTION      int64
RERA                    int64
BHK_NO.                 int64
SQUARE_FT             float64
READY_TO_MOVE           int64
RESALE                  int64
LONGITUDE             float64
LATITUDE              float64
ADDRESS_enc             int32
POSTED_BY_enc           int32
dtype: object
```

In [81]:

```
X_val.dtypes
```

Out[81]:

```
ID                      int64
UNDER_CONSTRUCTION      int64
RERA                    int64
BHK_NO.                 int64
SQUARE_FT             float64
READY_TO_MOVE           int64
RESALE                  int64
LONGITUDE             float64
LATITUDE              float64
ADDRESS_enc             int32
POSTED_BY_enc           int32
dtype: object
```

Feature Selection

Filter Method

In [82]:

```
from sklearn.feature_selection import VarianceThreshold
constant_filter = VarianceThreshold(threshold=0)
constant_filter.fit(X_train)
len(X_train.columns[constant_filter.get_support()])
```

Out[82]:

```
11
```

In [83]:

```
constant_columns = [column for column in X_train.columns
                    if column not in X_train.columns[constant_filter.get_support()]]
print(len(constant_columns))
```

```
0
```

Transpose the feature matrice

Print the number of duplicated features

select the duplicated features columns names

In [84]:

```
x_train_T = X_train.T
print(x_train_T.duplicated().sum())
duplicated_columns = x_train_T[x_train_T.duplicated()].index.values
```

```
0
```

Create a function called "handling correlation" which is going to return fields based on the correlation matrix value on which filtering depended with a threshold of 0.8

In [85]:

```
def handling_correlation(X_train,threshold=0.8):
    corr_features = set()
    corr_matrix = X_train.corr()
    for i in range(len(corr_matrix .columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) >threshold:
                colname = corr_matrix.columns[i]
                corr_features.add(colname)
    return list(corr_features)
```

Call handling_correlation() function with a threshold value of 0.85

In [86]:

```
train=X_train.copy()
handling_correlation(train.copy(),threshold=0.85)
```

Out[86]:

```
['READY_TO_MOVE']
```

Calling variance threshold for threshold value = 0.8

In [87]:

```
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(0.8)
sel.fit(X_train)
```

Out[87]:

```
VarianceThreshold(threshold=0.8)
```

In [88]:

```
sel.variances_
```

Out[88]:

```
array([2.60190749e+07, 1.47324546e-01, 2.16253873e-01, 7.61354263e-01,
       6.02364833e+12, 1.47324546e-01, 6.43253363e-02, 3.80712594e+01,
       1.10015050e+02, 3.52130389e+03, 2.65399266e-01])
```

Modelling

Make different blank list for different evaluation matrix

In [89]:

```
MSE_Score = []
R2_Score = []
Algorithm = []
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

start with the baseline model Linear Regression

In [90]:

```
from sklearn.linear_model import LinearRegression
Algorithm.append('Linear Regression')
regressor = LinearRegression()
regressor.fit(X_train, y_train)
predicted= regressor.predict(X_test)
```

Check for the Mean Square Error & R Square Score

In [91]:

```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [92]:

```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

Out[92]:

```
526.1233574108559
```

Display The Comparison Lists

In [93]:

```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

```
['Linear Regression'],[317234.6021880041],[0.061042956928039516],
```

Next model would be Support Vector Regression

In [94]:

```
from sklearn.svm import SVR
Algorithm.append('Support Vector Regression')
regressor = SVR()
regressor.fit(X_train, y_train)
predicted= regressor.predict(X_test)
```

```
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: Da
taConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel
().
  return f(*args, **kwargs)
```

Check for the Mean Square Error & R Square Score

In [95]:

```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [96]:

```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

Out[96]:

```
547.4051371561852
```

Display The Comparison Lists

In [97]:

```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

```
['Linear Regression', 'Support Vector Regression'],[317234.6021880041, 34352
6.8284507773],[0.061042956928039516, -0.01677727723683864],
```

Next model would be Decision Tree Regression

In [98]:

```
from sklearn.tree import DecisionTreeRegressor
Algorithm.append('Decision Tree Regression')
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
predicted = regressor.predict(X_test)
```

Check for the Mean Square Error & R Square Score

In [99]:

```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [100]:

```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

Out[100]:

```
194.5356054742052
```

Display The Comparison Lists

In [101]:

```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

```
['Linear Regression', 'Support Vector Regression', 'Decision Tree Regressio
n'],[317234.6021880041, 343526.8284507773, 36448.76334901528],[0.06104295692
8039516, -0.01677727723683864, 0.8921182531105518],
```

Next model would be Random Forest Regression

In [102]:

```
from sklearn.ensemble import RandomForestRegressor
Algorithm.append('Random Forest Regression')
regressor = RandomForestRegressor()
regressor.fit(X_train,y_train)
predicted=regressor.predict(X_test)
```

```
C:\Users\KIIT\AppData\Local\Temp/ipykernel_7116/1525345632.py:4: DataConvers
ionWarning: A column-vector y was passed when a 1d array was expected. Pleas
e change the shape of y to (n_samples,), for example using ravel().
  regressor.fit(X_train,y_train)
```

Check for the Mean Square Error & R Square Score

In [103]:

```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [104]:

```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

Out[104]:

```
137.4015818328652
```

Display The Comparison Lists

In [105]:

```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

```
['Linear Regression', 'Support Vector Regression', 'Decision Tree Regressio
n', 'Random Forest Regression'],[317234.6021880041, 343526.8284507773, 3644
8.76334901528, 24241.026548794554],[0.061042956928039516, -0.016777277236838
64, 0.8921182531105518, 0.9282509459803634],
```

last but not the least model would be XGBoost or Extreme Gradient Boost Regression

In [106]:

```
import xgboost as xgb
Algorithm.append('Extreme Gradient Boost Regression')
regressor = xgb.XGBRegressor()
regressor.fit(X_train, y_train)
predicted = regressor.predict(X_test)
```

Check for the Mean Square Error & R Square Score

In [107]:

```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [108]:

```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

Out[108]:

```
169.5773144604059
```

Display The Comparison Lists

In [109]:

```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

```
['Linear Regression', 'Support Vector Regression', 'Decision Tree Regressio
n', 'Random Forest Regression', 'Extreme Gradient Boost Regression'],[31723
4.6021880041, 343526.8284507773, 36448.76334901528, 24241.026548794554, 3686
4.11949807391],[0.061042956928039516, -0.01677727723683864, 0.89211825311055
18, 0.9282509459803634, 0.8908888740363526],
```

Make the comparison list into a comparison dataframe

In [110]:

```
Comparison = pd.DataFrame(list(zip(Algorithm, MSE_Score, R2_Score)), columns = ['Algorithm'
Comparison
```

Out[110]:

	Algorithm	MSE_Score	R2_Score
0	Linear Regression	317234.602188	0.061043
1	Support Vector Regression	343526.828451	-0.016777
2	Decision Tree Regression	36448.763349	0.892118
3	Random Forest Regression	24241.026549	0.928251
4	Extreme Gradient Boost Regression	36864.119498	0.890889

Now from the Comparison table, it is needed to choose the best fit model

Fit X_train and y_train inside the model

Predict the X_test dataset

Predict the X_val dataset

In [111]:

```
regressorfinal = RandomForestRegressor()
regressorfinal.fit(X_train, y_train)
predictedfinal = regressorfinal.predict(X_test)
predict_testfinal = regressorfinal.predict(X_val)
```

```
C:\Users\KIIT\AppData\Local\Temp/ipykernel_7116/362160554.py:2: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
  regressorfinal.fit(X_train, y_train)
```

Calculate the Mean Square Error for test dataset

In [112]:

```
mean_squared_error(y_test,predictedfinal,squared=False)
```

Out[112]:

```
160.91936138351528
```

Calculate the mean Square Error for validation dataset

In [113]:

```
mean_squared_error(y_val,predict_testfinal,squared=False)
```

Out[113]:

```
142.00857500866303
```

Calculate the R2 score for test

In [114]:

```
r2_score(y_test,predictedfinal)
```

Out[114]:

```
0.9233553627630162
```

Calculate the R2 score for Validation

In [115]:

```
r2_score(y_val,predict_testfinal)
```

Out[115]:

```
0.9315093575829952
```

Calculate the Accuracy for train Dataset

In [116]:

```
regressorfinal.score(X_train, y_train)
```

Out[116]:

```
0.9787479666493318
```

Calculate the accuracy for validation

In [117]:

```
regressorfinal.score(X_val, y_val)
```

Out[117]:

```
0.9315093575829952
```

Calculate the accuracy for test

In [118]:

```
regressorfinal.score(X_test, y_test)
```

Out[118]:

```
0.9233553627630162
```

Passing the "testdata" dataframe into this machine learning model

Display the testdata

In [119]:

```
test_data
```

Out[119]:

	ID	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	READY
0	2229451	Owner	0	0	1	545.171340	
1	2229452	Dealer	1	1	2	800.000000	
2	2229453	Dealer	0	0	2	1257.096513	
3	2229454	Dealer	0	0	3	1400.329489	
4	2229455	Owner	0	0	1	430.477830	
...	...	...	...	...	...	...	
68715	2298166	Dealer	0	1	2	856.555505	
68716	2298167	Dealer	0	1	3	2304.147465	
68717	2298168	Dealer	1	1	1	33362.792750	
68718	2298169	Dealer	0	0	2	1173.708920	
68719	2298170	Dealer	0	0	3	2439.532944	

68720 rows × 11 columns

Check for the number of rows and columns in the testdata

In [120]:

```
test_data.shape
```

Out[120]:

```
(68720, 11)
```

Check the Description and Information of the testdata

In [121]:

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 68720 entries, 0 to 68719
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  68720 non-null  int64  
 1   POSTED_BY           68720 non-null  object 
 2   UNDER_CONSTRUCTION  68720 non-null  int64  
 3   RERA                68720 non-null  int64  
 4   BHK_NO.             68720 non-null  int64  
 5   SQUARE_FT           68720 non-null  float64
 6   READY_TO_MOVE       68720 non-null  int64  
 7   RESALE              68720 non-null  int64  
 8   ADDRESS             68720 non-null  object 
 9   LONGITUDE           68720 non-null  float64
 10  LATITUDE            68720 non-null  float64
dtypes: float64(3), int64(6), object(2)
memory usage: 6.3+ MB
```

In [122]:

```
test_data.describe()
```

Out[122]:

	ID	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	READ
count	6.872000e+04	68720.000000	68720.000000	68720.000000	6.872000e+04	
mean	2.263810e+06	0.176557	0.316531	2.388198	2.762419e+03	
std	1.983790e+04	0.381296	0.465126	0.864577	1.640991e+05	
min	2.229451e+06	0.000000	0.000000	1.000000	1.000000e+00	
25%	2.246631e+06	0.000000	0.000000	2.000000	9.000310e+02	
50%	2.263810e+06	0.000000	0.000000	2.000000	1.174982e+03	
75%	2.280990e+06	0.000000	1.000000	3.000000	1.550265e+03	
max	2.298170e+06	1.000000	1.000000	31.000000	4.016393e+07	

Call the Label Encoder for testdata

In [123]:

```
from sklearn.preprocessing import LabelEncoder
ADDRESSrn = LabelEncoder()
ADDRESSrn.fit(test_data['ADDRESS'])
test_data['ADDRESS_enc'] = ADDRESSrn.transform(test_data['ADDRESS'])
```

Use Label Encoder of all the following columns

In [124]:

```
test_data['ADDRESS_enc']=label_encoder.transform(test_data['ADDRESS'])
test_data['POSTED_BY_enc']=label_encoder.transform(test_data['POSTED_BY'])
```

Check for the datatypes of all the columns of testdata

In [125]:

```
test_data.dtypes
```

Out[125]:

```
ID                      int64
POSTED_BY              object
UNDER_CONSTRUCTION      int64
RERA                    int64
BHK_NO.                 int64
SQUARE_FT             float64
READY_TO_MOVE           int64
RESALE                  int64
ADDRESS                object
LONGITUDE             float64
LATITUDE              float64
ADDRESS_enc             int32
POSTED_BY_enc           int32
dtype: object
```

Now it is needed to drop all the unnecessary columns

In [126]:

```
test_data.drop(columns=['ADDRESS', 'POSTED_BY'], inplace=True)
```

Check the information of the "testdata" dataframe

In [127]:

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 68720 entries, 0 to 68719
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  68720 non-null  int64  
 1   UNDER_CONSTRUCTION  68720 non-null  int64  
 2   RERA                68720 non-null  int64  
 3   BHK_NO.             68720 non-null  int64  
 4   SQUARE_FT           68720 non-null  float64
 5   READY_TO_MOVE       68720 non-null  int64  
 6   RESALE              68720 non-null  int64  
 7   LONGITUDE           68720 non-null  float64
 8   LATITUDE            68720 non-null  float64
 9   ADDRESS_enc         68720 non-null  int32  
 10  POSTED_BY_enc       68720 non-null  int32  
dtypes: float64(3), int32(2), int64(6)
memory usage: 5.8 MB
```

Compare "testdata" with the "X_test" dataframe

In [128]:

```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5890 entries, 23560 to 29449
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5890 non-null   int64  
 1   UNDER_CONSTRUCTION  5890 non-null   int64  
 2   RERA                5890 non-null   int64  
 3   BHK_NO.             5890 non-null   int64  
 4   SQUARE_FT           5890 non-null   float64
 5   READY_TO_MOVE       5890 non-null   int64  
 6   RESALE              5890 non-null   int64  
 7   LONGITUDE           5890 non-null   float64
 8   LATITUDE            5890 non-null   float64
 9   ADDRESS_enc         5890 non-null   int32  
 10  POSTED_BY_enc       5890 non-null   int32  
dtypes: float64(3), int32(2), int64(6)
memory usage: 506.2 KB
```

In [129]:

```
test_data.columns == X_test.columns
```

Out[129]:

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True])
```

Display the Final Dataset

In [130]:

```
test_data
```

Out[130]:

	ID	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	READY_TO_MOVE	F
0	2229451	0	0	1	545.171340	1	
1	2229452	1	1	2	800.000000	0	
2	2229453	0	0	2	1257.096513	1	
3	2229454	0	0	3	1400.329489	1	
4	2229455	0	0	1	430.477830	1	
...	...	...	...	...	...	...	
68715	2298166	0	1	2	856.555505	1	
68716	2298167	0	1	3	2304.147465	1	
68717	2298168	1	1	1	33362.792750	0	
68718	2298169	0	0	2	1173.708920	1	
68719	2298170	0	0	3	2439.532944	1	

68720 rows × 11 columns

Now passing this dataset into the final model and store it into "final_result"

In [131]:

```
final_result = regressorfinal.predict(test_data)
```

Making the final_result as dataframe, with a column name "TARGET(PRICE_IN_LACS)"

In [132]:

```
final_result = pd.Series(final_result, name='TARGET(PRICE_IN_LACS)')
```

Display the "TARGET(PRICE_IN_LACS)" column

In [133]:

```
final_result
```

Out[133]:

```
0          47.115
1          42.371
2          75.909
3          44.122
4          57.981
           ...
68715     120.512
68716     111.690
68717    6137.868
68718      75.343
68719     798.700
Name: TARGET(PRICE_IN_LACS), Length: 68720, dtype: float64
```

Now merge this final_result dataframe with the BACKUP of "test_data" Dataframe which we have created in earlier steps

In [134]:

```
test_data1.reset_index(drop=True,inplace=True)
Final = test_data1.merge(final_result , on = test_data.index )
```

Display the "Final" dataframe

In [135]:

```
Final
```

Out[135]:

	key_0	ID	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_F
0	0	2229451	Owner	0	0	1	BH
1	1	2229452	Dealer	1	1	2	BH
2	2	2229453	Dealer	0	0	2	BH
3	3	2229454	Dealer	0	0	3	BH
4	4	2229455	Owner	0	0	1	BH
...	...	...	...	...	...	...	
68715	68715	2298166	Dealer	0	1	2	BH
68716	68716	2298167	Dealer	0	1	3	BH
68717	68717	2298168	Dealer	1	1	1	BH
68718	68718	2298169	Dealer	0	0	2	BH
68719	68719	2298170	Dealer	0	0	3	BH

68720 rows × 14 columns

Drop the extra index column with name "key_0" from the "Final" dataframe

In [136]:

```
Final.drop(columns="key_0", inplace=True)
```

Null values present in 'TARGET(PRICE_IN_LACS)' column

In [137]:

```
Final['TARGET(PRICE_IN_LACS)'].isnull().sum()
```

Out[137]:

```
0
```

Check for the Number of Rows and Columns in your "Final" dataframe

In [138]:

```
Final.shape
```

Out[138]:

```
(68720, 13)
```

Now displaying our final dataset

In [139]:

```
Final
```

)	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUARE_FT	READY_TO_MOVE	RE
1	Owner	0	0	1	BHK	545.171340	1	
2	Dealer	1	1	2	BHK	800.000000	0	
3	Dealer	0	0	2	BHK	1257.096513	1	
4	Dealer	0	0	3	BHK	1400.329489	1	
5	Owner	0	0	1	BHK	430.477830	1	
.	...	...	...	...	...	...	...	
3	Dealer	0	1	2	BHK	856.555505	1	
7	Dealer	0	1	3	BHK	2304.147465	1	

Display the location of the houses from Final dataset on the map of India

In [140]:

```
import folium
from folium import Choropleth, Circle, Marker
from folium.plugins import HeatMap, MarkerCluster
```

In [141]:

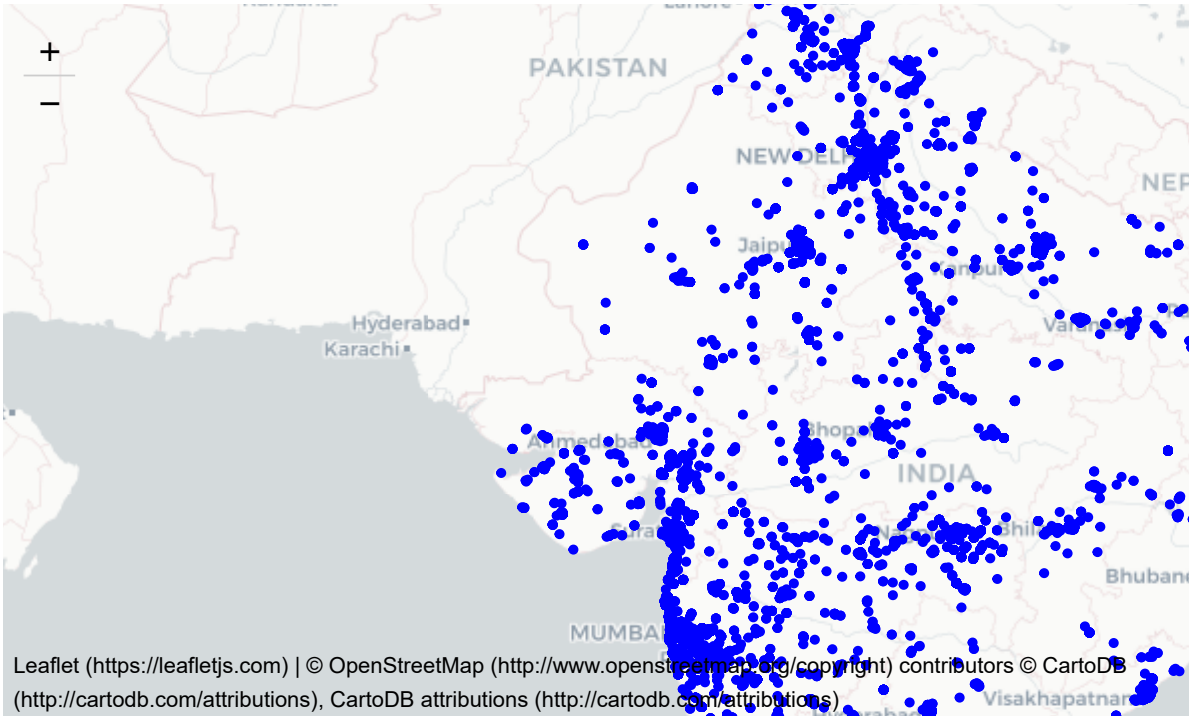
```
map = folium.Map(location=[22.00,78.00], tiles='cartodbpositron', zoom_start=6)


for i in range(0,len(Final)):
    Circle(
        location=[Final.iloc[i]['LONGITUDE'], Final.iloc[i]['LATITUDE']],
        radius=100,
        color='blue').add_to(map)
```

In [142]:

```
#displaying the map
map
```

Out[142]:



Store this dataframe into the .csv format

In [143]:

```
Final.to_csv("C:/Users/KIIT/Documents/3rd Year/6th SEM/T&T Lab/Project/Group10_T&T.csv")
```

END OF THE PROJECT

- Group-10
- Kushagra Kumar Rai (1906484)
- Shivam Kumar (1906504)
- Souhardya Pal (1906511)
- Tapabrata Roy (1906522)