ML Project Report

Group Name - G_078_098_125_128

Report-

Task 1: Exploratory Data Analysis (EDA)

Objective: Initial understanding of the real estate data.

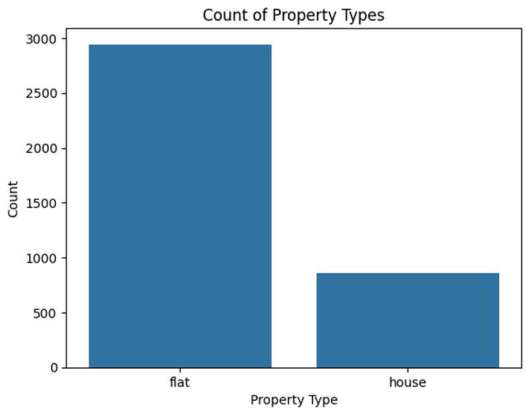
Methods:

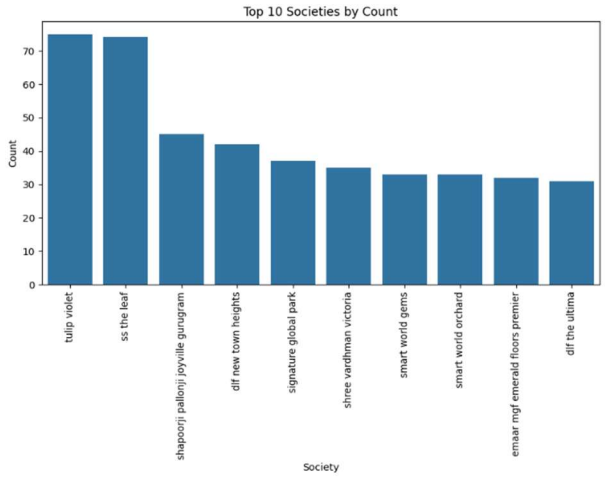
- Analyze data types and identify missing values for all columns.
- Focus on specific features like:
  - **Property characteristics**: Property Type, Area, Area with Type, Super Built-Up Area, Built-Up Area, Carpet Area
  - **Pricing**: Price, Price per sqft.
  - **Bedroom and Bathroom count:** Bedroom, Bathroom
- Employ descriptive statistics and visualizations (histograms, boxplots) to explore the distribution of these features.

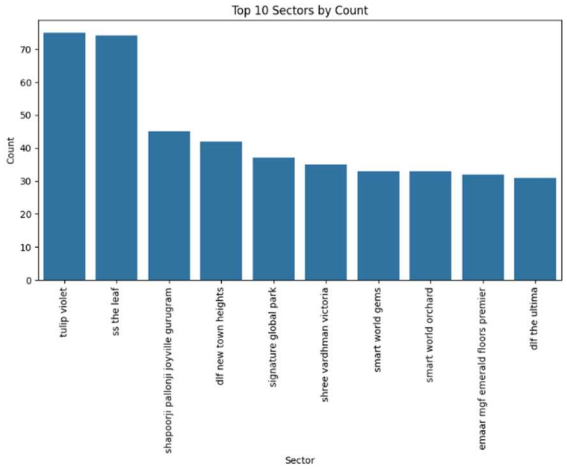
Outcomes:

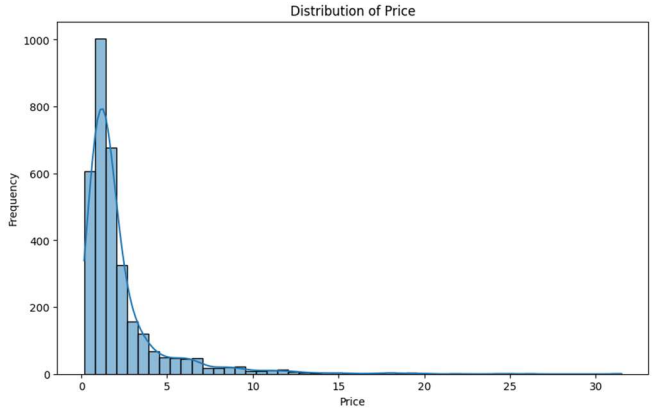
- Identify potential outliers, skewness, and initial trends.
- Prepare the data for further cleaning and analysis.

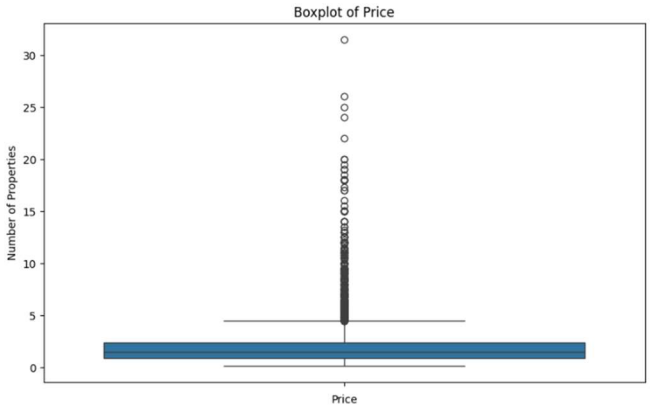
Following are the graphs which visualize the data: -

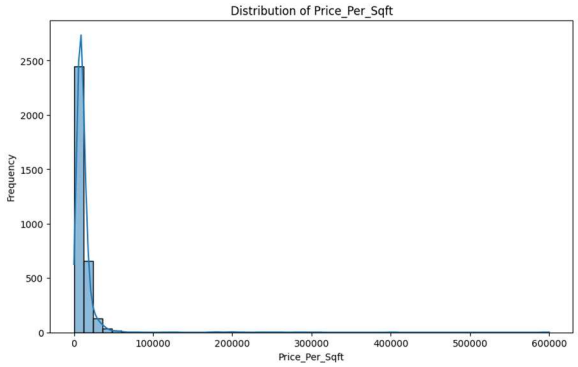


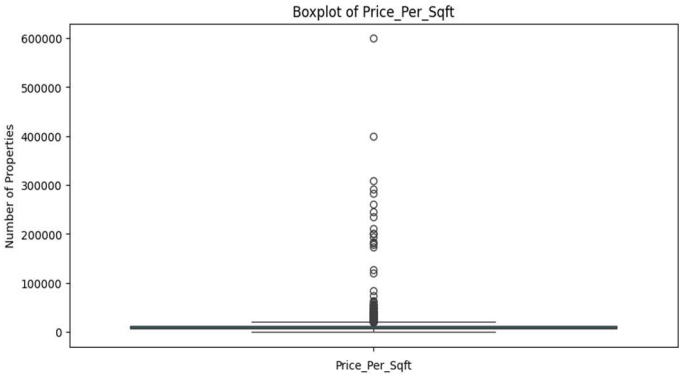


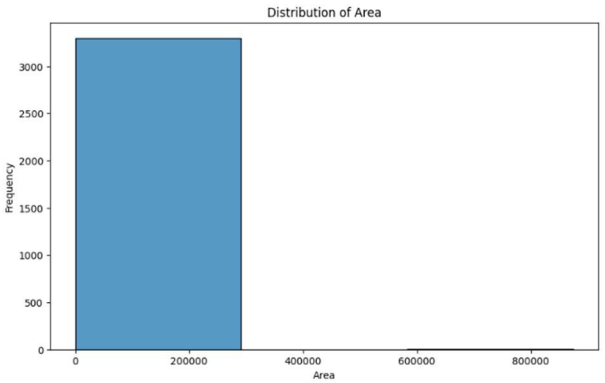

Top 10 Sectors by Count
Count
Sector
tulip violet
ss the leaf
shapoorji pallonji joyville gurugram
dlf new town heights
signature global park
shree vardhman victoria
smart world gems
smart world orchard
emaar mgf emerald floors premier
dlf the ultima

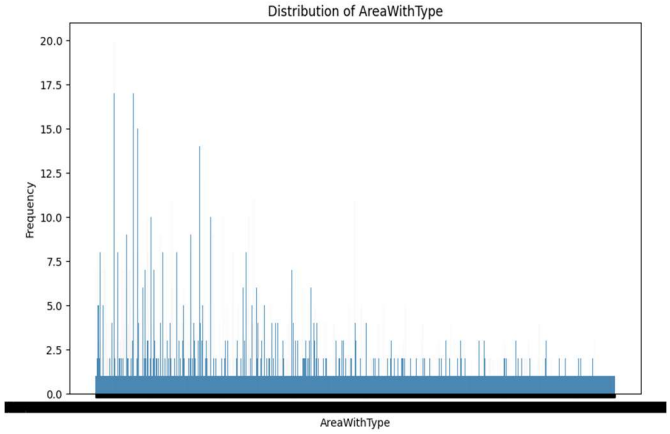

Distribution of Price
Frequency
Price

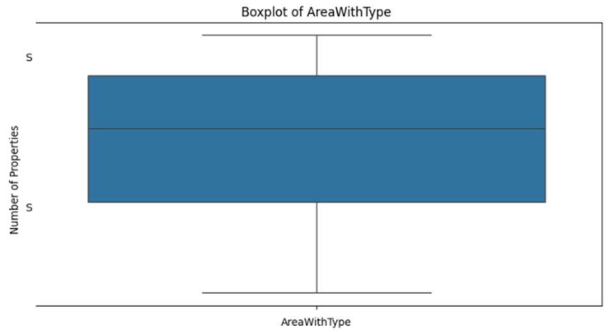

Boxplot of Price
Number of Properties
Price

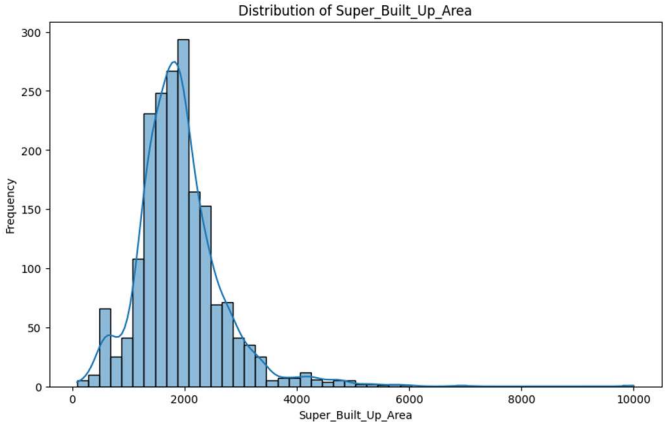

Distribution of Price_Per_Sqft
Frequency
Price_Per_Sqft

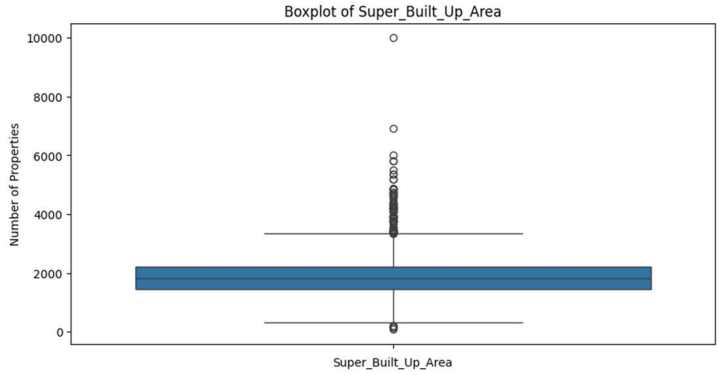

Boxplot of Price_Per_Sqft
Number of Properties
Price_Per_Sqft

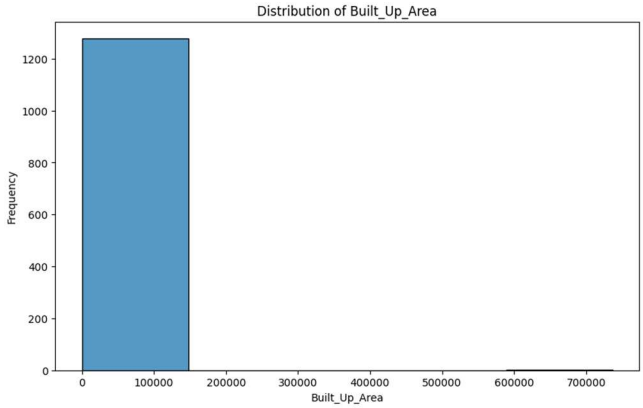

Distribution of Area
Frequency
Area

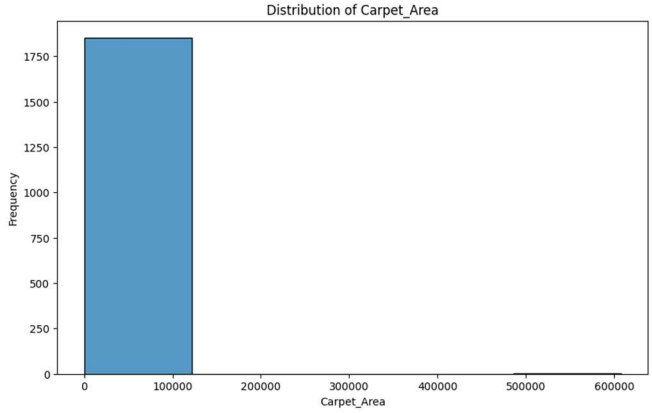

Distribution of AreaWithType
Frequency
AreaWithType

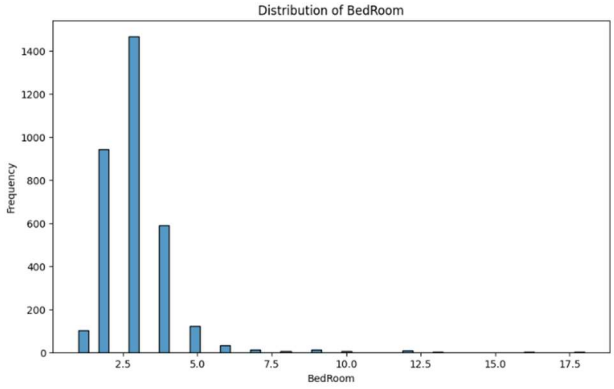

Boxplot of AreaWithType
Number of Properties
AreaWithType

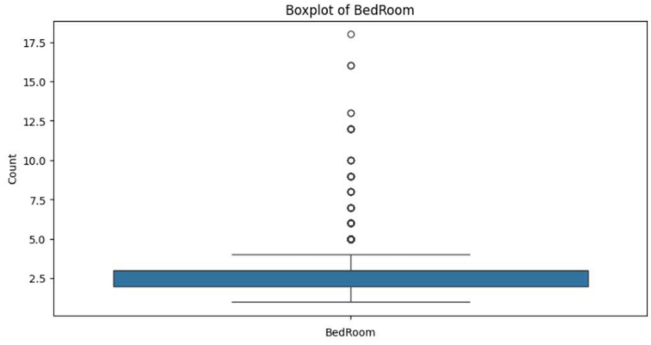

Distribution of Super_Built_Up_Area
Frequency
Super_Built_Up_Area

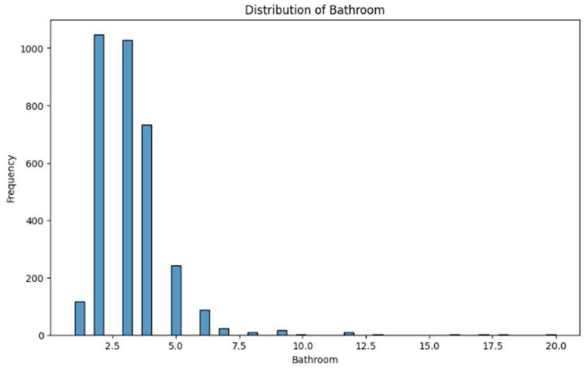

Boxplot of Super_Built_Up_Area
Number of Properties
Super_Built_Up_Area

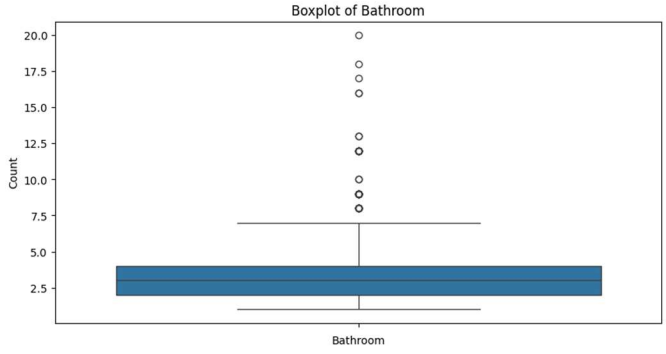

Distribution of Built_Up_Area
Frequency
Built_Up_Area

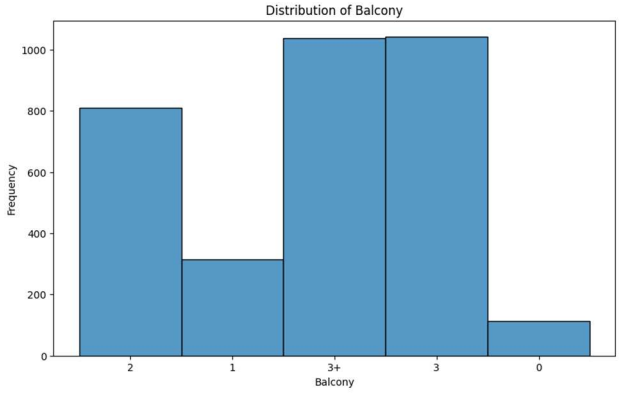

Distribution of Carpet_Area
Frequency
Carpet_Area

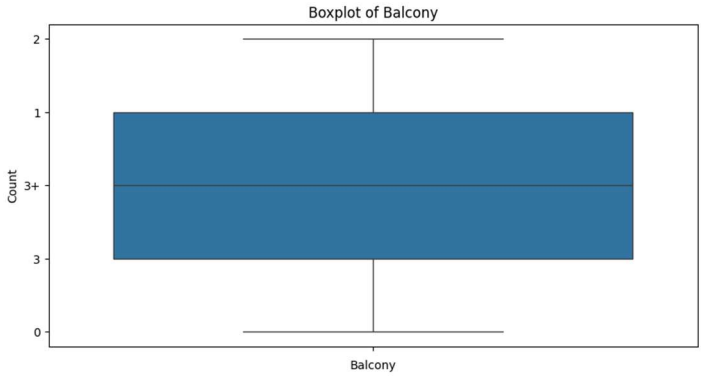

Distribution of BedRoom
Frequency
BedRoom

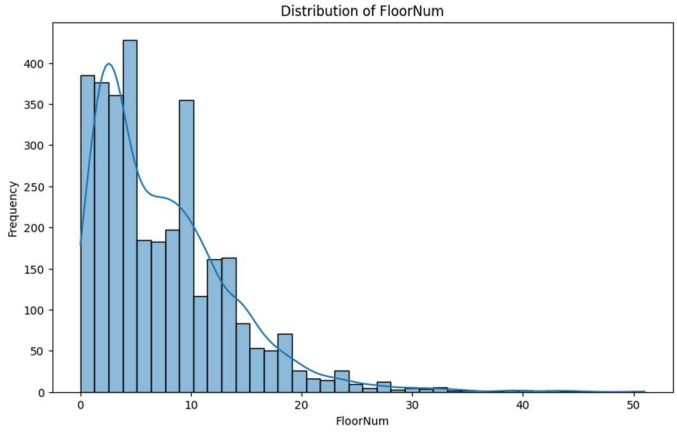

Boxplot of BedRoom
Count
BedRoom

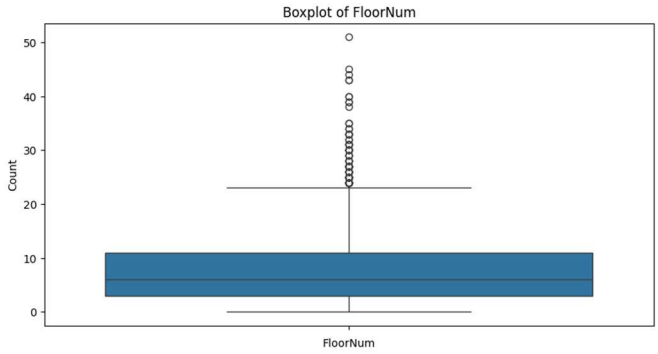

Distribution of Bathroom
Frequency
Bathroom

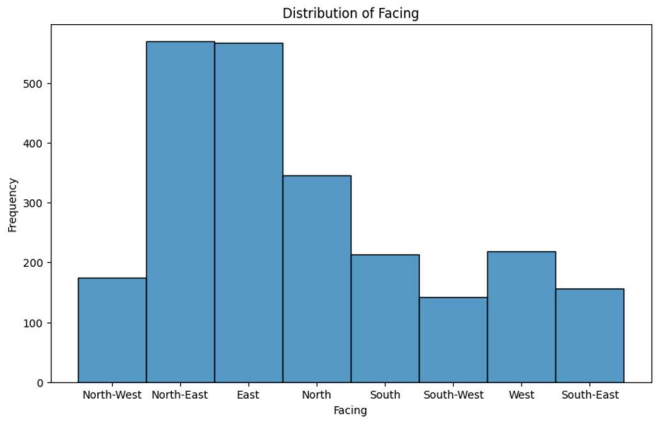

Boxplot of Bathroom
Count
Bathroom

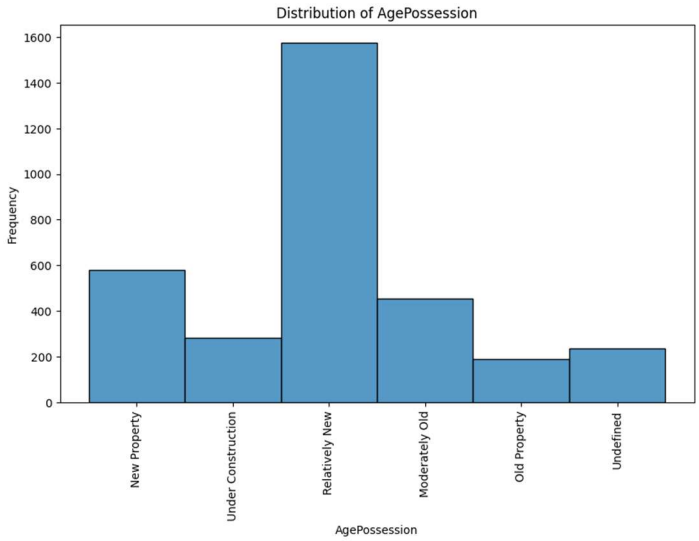

Distribution of Balcony
Frequency
Balcony
2
1
3+
3
0

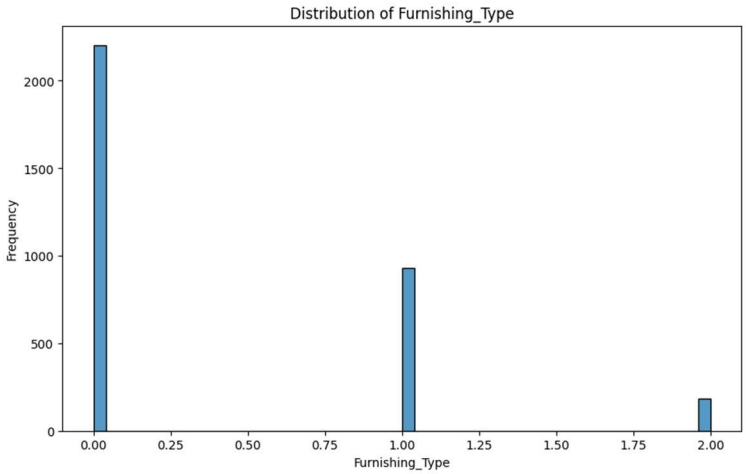

Boxplot of Balcony
Count
Balcony

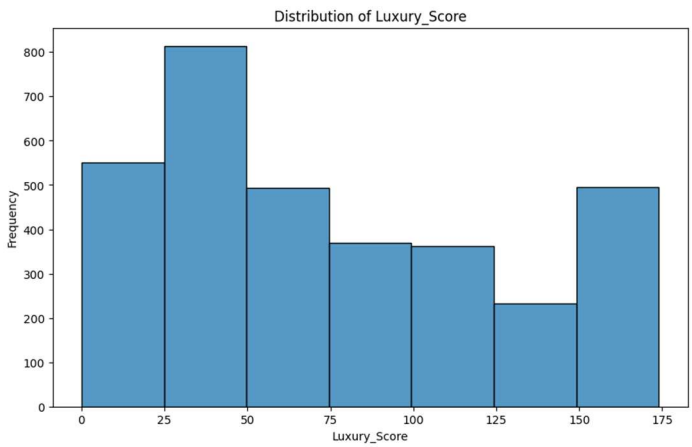

Distribution of FloorNum
Frequency
FloorNum

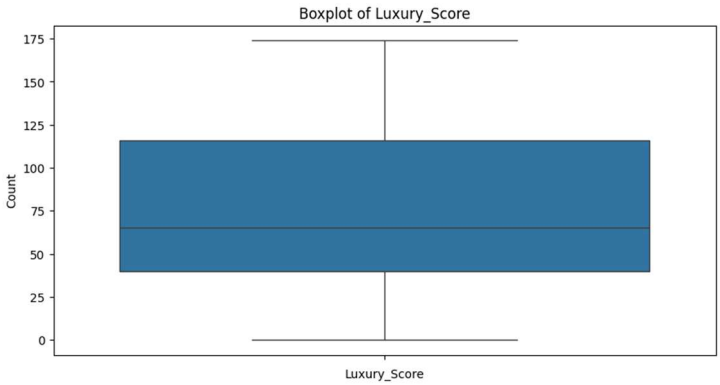

Boxplot of FloorNum
Count
FloorNum

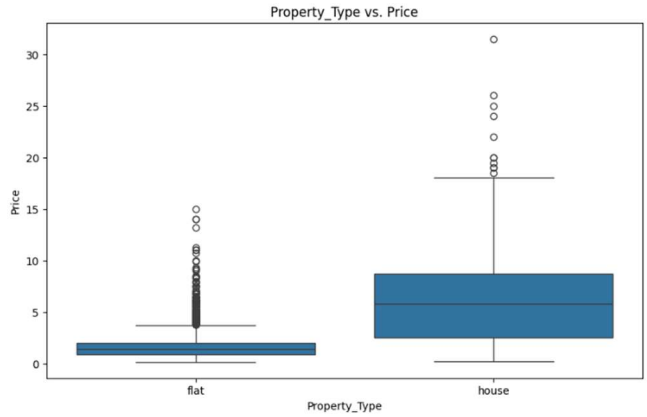

Distribution of Facing
Frequency
North-West
North-East
East
North
South
South-West
West
South-East
Facing

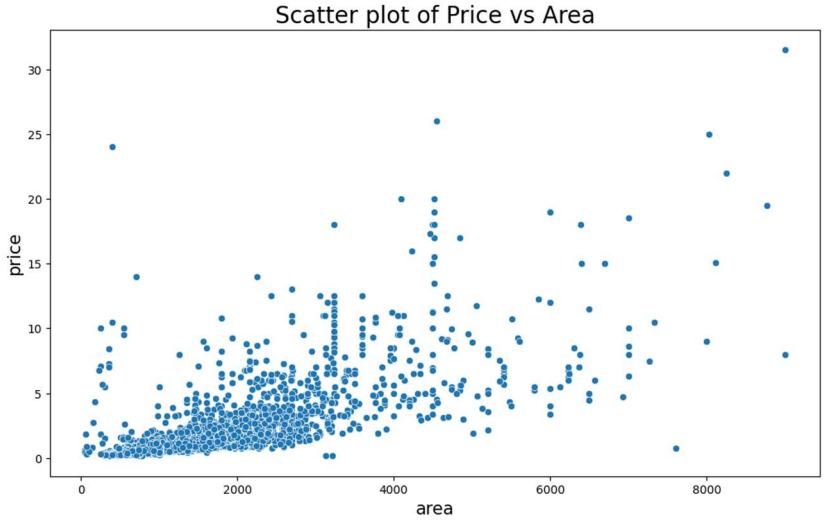

Distribution of AgePossession
Frequency
New Property
Under Construction
Relatively New
Moderately Old
Old Property
Undefined
AgePossession











Code snippet:

Following is the code snippet used to generate the graphs:

```
#Nature of the column
df['floorNum'].describe()
```

we obtain the graphs for the rest of the columns similarly.

Moreover, we used binning based on frequency and implemented it on dataset.

Code snippet used for binning:

```
#Defining bins based on frequency
bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200]

#Using pd.cut() to bin the frequency counts
frequency_bins = pd.cut(sector_count, bins=bins)

print(frequency_bins)
```

similarly, binning was used on other parameters like price, etc.

Task-2 : Handling missing values:

Code snippets for Task-2 and their purpose:

```
#Checking for missing values
missing = df.isnull().sum()
print(missing)
```

The above code is used for finding if any missing values exist in the dataset.

The output of the above code:

```
property_type              0
society                    1
sector                     0
price                     18
price_per_sqft            18
area                      18
areaWithType               0
bedRoom                    0
bathroom                   0
balcony                    0
floorNum                  19
facing                  1105
agePossession              0
super_built_up_area     1888
built_up_area           2070
carpet_area             1859
study room                 0
servant room               0
```

```
store room                  0
pooja room                  0
others                      0
furnishing_type             0
luxury_score                0
dtype: int64
```

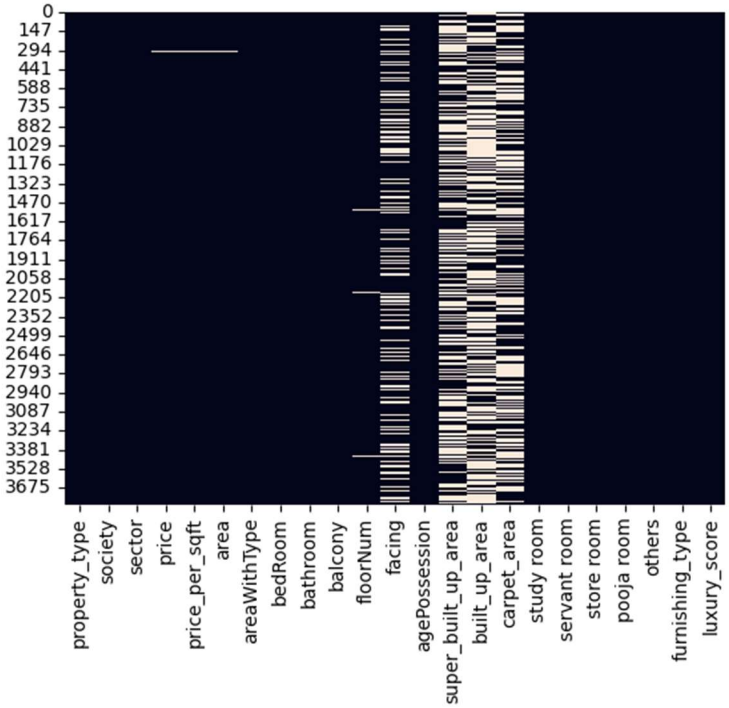
Here in the output, we are finding the number of missing values for each column(parameter)

```
sns.heatmap(df.isnull(), cbar=False)
```

In the output of above code, we can see that the columns price, price_per_sqft, and area have correlation in how their missing data occur.

One observation is that missing value in society column, doesn't make a huge difference.

Output for the above code:



Column: Society

```
df['society'].isnull().sum()
```

This code finds the number of missing values in the “society” column(parameter).

Output:

```
1
```

As only one missing value exist, it is better to delete that row containing missing value in society.

The code snippet to delete the “society” row from the dataset.

```
df.dropna(subset=['society'], inplace=True)
df['society'].isnull().sum()
```

Column: Area

the following code snippet is used to find number of missing values in the column “Area”.

```
df['area'].isnull().sum()
```

output:

```
18
```

The following code snippet is used to analyse the nature of the column:

```
df['area'].describe()
```

output:

```
count     3784.000000
mean      2846.322146
std      22786.351485
min         50.000000
25%       1220.000000
50%       1725.000000
75%       2295.000000
max     875000.000000
Name: area, dtype: float64
```

Now, Assuming data is your DataFrame and the columns are named as specified, we check for missing values in the 'area' column using the below code.

```
missing_area = df['area'].isnull()
```

Then, we replace missing values in 'area' with corresponding values from 'built_up_area' if available otherwise replace with 'carpet_area'.

The following code is used to implement it

```
df.loc[missing_area, 'area'] = df.loc[missing_area,
'super_built_up_area'].combine_first(df.loc[missing_area,
'built_up_area']).combine_first(df.loc[missing_area, 'carpet_area'])
```

Now, we find out the number of missing values, with this code:

```
df['area'].isnull().sum()
```

Column: Price_per_sqft

The following code snippet is to find the number of missing values in the column “ Price_per_sqft”.

```
df['price_per_sqft'].isnull().sum()
```

Output:

```
18
```

To understand the nature of the column:

```
df['price_per_sqft'].describe()
```

Output:

```
count     3784.000000
mean     13802.839059
std      23054.466260
min          4.000000
25%       6810.250000
50%       9000.000000
75%      13766.000000
max     600000.000000
Name: price_per_sqft, dtype: float64
```

We can confidently replace missing values in the 'price_per_sqft' column with the mean 'price_per_sqft' corresponding to each unique combination of 'property_type', 'society', and 'sector'. Since 'price_per_sqft' tends to remain constant within the same society, this approach ensures reasonable estimates for missing values.

First, we group the DataFrame by 'property_type', 'society', and 'sector', calculating the mean 'price_per_sqft' for each group.

Next, we iterate over rows with missing 'price_per_sqft' values. For each row, we extract its 'property_type', 'society', and 'sector'. If there exists a mean 'price_per_sqft' value for the corresponding combination, we replace the missing value with this mean.

This iterative process ensures that missing 'price_per_sqft' values are replaced with meaningful estimates based on similar property types within the same society and sector.

Code snippet for the above:

```
# Group by 'property_type', 'society', and 'sector' and calculate the mean
of 'price_per_sqft'
mean_price_per_sqft = df.groupby(['property_type', 'society',
'sector'])['price_per_sqft'].mean()

# Iterate over the rows with missing 'price_per_sqft' values
for index, row in df[df['price_per_sqft'].isnull()].iterrows():
    # Extract the values of 'property_type', 'society', and 'sector' for
the current row
    property_type = row['property_type']
    society = row['society']
    sector = row['sector']

    # Check if there is a corresponding mean value for the combination
    if (property_type, society, sector) in mean_price_per_sqft.index:
        # Replace the missing value with the mean value for the
corresponding combination
        df.at[index, 'price_per_sqft'] =
mean_price_per_sqft.loc[(property_type, society, sector)]

df['price_per_sqft'].isnull().sum()
```

Output:

```
3
```

Now, we delete the remaining three rows.

```
df.dropna(subset=['price_per_sqft'], inplace=True)
df['price_per_sqft'].isnull().sum()
```

Output:

```
0
```

Column: price

Now, for the price column, we first find the number of missing values.

```
df['price'].isnull().sum()
```

Output:

```
15
```

To find out the nature of this column, we use the following code:

```
df['price'].describe()
```

Output:

```
count   3784.000000
mean       2.506308
std        2.950348
min        0.070000
25%        0.940000
50%        1.500000
75%        2.700000
max       31.500000
Name: price, dtype: float64
```

Now, we first find the product of “price_per_sqft” and “area” and scale the product as the price value is too high, in crores. Then we replace the missing values in “price” column with the scaled product.

Code for the above is as follows:

```
product = df['price_per_sqft'] * df['area']
scaled_product = product / 10**7
df['price'].fillna(scaled_product, inplace=True)
```

Now the number of missing values will be zero as shown by the output for the following code:

```
df['price'].isnull().sum()
```

Output:

```
0
```

After this, we do not require the area column

```
df.drop(columns=['area'], inplace=True)
```

Column: Built_Up_Area

The following code snippet is to find the number of missing values in the column “Built_Up_Area”.

```
#Number of missing values
df['built_up_area'].isnull().sum()
```

output:

```
2069
```

Now, to find out the nature of this column, we use the following code:

```
#Nature of column
df['built_up_area'].describe()
```

output:

```
count     1730.000000
mean      2360.366109
std      17734.773676
min          2.000000
25%       1100.000000
50%       1650.000000
75%       2398.750000
max     737147.000000
Name: built_up_area, dtype: float64
```

Then, we check if all 3 values(built up, super built up and carpet areas) are missing, with the following code:

```
((df['super_built_up_area'].isnull()) & (df['built_up_area'].isnull()) &
(df['carpet_area'].isnull())).sum()
```

Output:

```
0
```

Storing all 3 values(built up, super built up and carpet areas) present in all_present_df

```
all_present_df = df[~((df['super_built_up_area'].isnull()) |
(df['built_up_area'].isnull()) | (df['carpet_area'].isnull()))]
```

now, calculating the super TO builtup ratio and carpet TO builtup ratio, with the following code:

```
super_to_built_up_ratio =
(all_present_df['super_built_up_area']/all_present_df['built_up_area']).me
dian()
carpet_to_built_up_ratio =
(all_present_df['carpet_area']/all_present_df['built_up_area']).median()

print(super_to_built_up_ratio)
print(carpet_to_built_up_ratio)
```

output:

```
1.1048701298701298
0.900140056022409
```

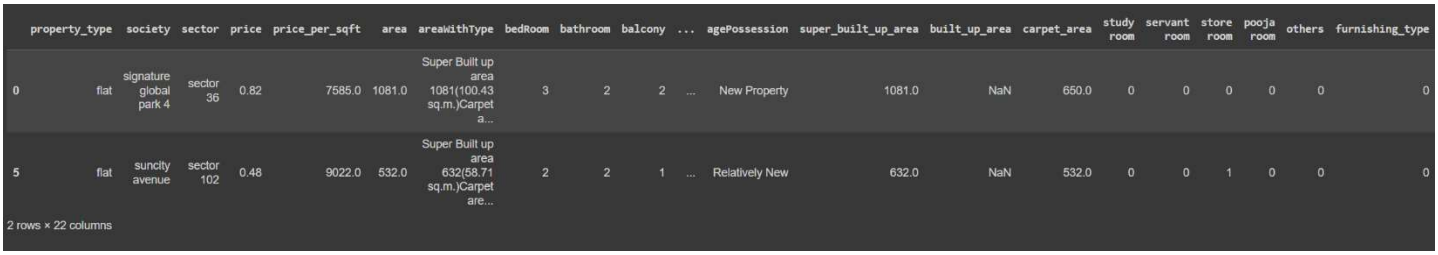
Now, we follow the following sequence of steps:

1) Storing super_built_up_area & carpet_area is present , built_up_area is null to sbc_df

```
sbc_df = df[~(df['super_built_up_area'].isnull()) &
(df['built_up_area'].isnull()) & ~(df['carpet_area'].isnull())]
```

```
sbc_df.head(2)
```

We get the following output:



	property_type	society	sector	price	price_per_sqft	area	areaWithType	bedRoom	bathroom	balcony	...	agePossession	super_built_up_area	built_up_area	carpet_area	study room	servant room	store room	pooja room	others	furnishing_type
0	flat	signature global park 4	sector 36	0.82	7585.0	1081.0	Super Built up area 1081(100.43 sq.m.)Carpet a...	3	2	2	...	New Property	1081.0	NaN	650.0	0	0	0	0	0	0
5	flat	suncity avenue	sector 102	0.48	9022.0	532.0	Super Built up area 632(58.71 sq.m.)Carpet are...	2	2	1	...	Relatively New	632.0	NaN	532.0	0	0	1	0	0	0

2 rows × 22 columns

Now, filling these values by using the ratio values of super_built_up_area & carpet_area, with the following code.

```
sbc_df['built_up_area'].fillna(round(((sbc_df['super_built_up_area']/1.105
) + (sbc_df['carpet_area']/0.9))/2),inplace=True)
```

and, updating it to the dataframe, with this code

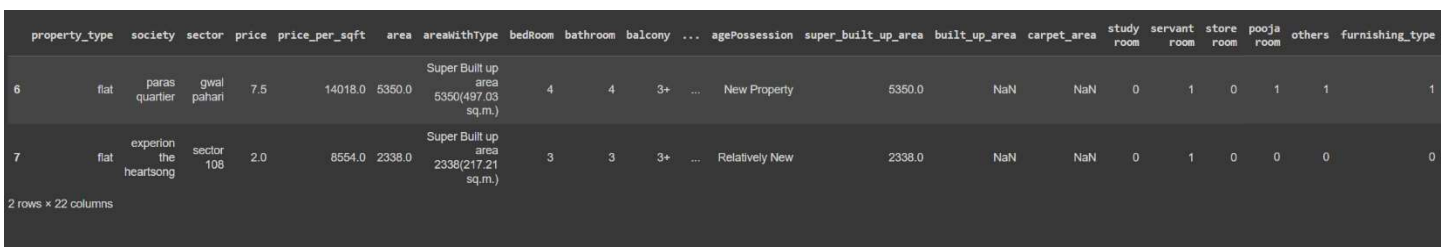
```
df.update(sbc_df)
```

2) Storing super_built_up_area is present , but built_up_area & carpet_area is null in sb_df:

```
sb_df = df[~(df['super_built_up_area'].isnull()) &
(df['built_up_area'].isnull()) & (df['carpet_area'].isnull())]
```

```
sb_df.head(2)
```

we get the following output:



	property_type	society	sector	price	price_per_sqft	area	areaWithType	bedRoom	bathroom	balcony	...	agePossession	super_built_up_area	built_up_area	carpet_area	study room	servant room	store room	pooja room	others	furnishing_type
6	flat	paras quartier	gwal pahari	7.5	14018.0	5350.0	Super Built up area 5350(497.03 sq.m.)	4	4	3+	...	New Property	5350.0	NaN	NaN	0	1	0	1	1	1
7	flat	experion the heartsong	sector 108	2.0	8554.0	2338.0	Super Built up area 2338(217.21 sq.m.)	3	3	3+	...	Relatively New	2338.0	NaN	NaN	0	1	0	0	0	0

2 rows × 22 columns

Now,Filling the values by using the ratio value of super_built_up_area, with the following code:

```
sb_df['built_up_area'].fillna(round(sb_df['super_built_up_area']/1.105),inplace=True)
```

and updating to dataframe, with the following code:

```
df.update(sb_df)
```

3) Storing carpet_area is present , but built_up_area & super_built_up_area is null in c_df:

```
c_df = df[(df['super_built_up_area'].isnull()) &
(df['built_up_area'].isnull()) & ~(df['carpet_area'].isnull())]

c_df.head(2)
```

we get the following output for this code:

	property_type	society	sector	price	price_per_sqft	area	areaWithType	bedRoom	bathroom	balcony	...	agePossession	super_built_up_area	built_up_area	carpet_area	study room	servant room	store room	pooja room	others	furnishing_type
1	flat	smart world gems	sector 89	0.95	8600.0	1105.0	Carpet area: 1103 (102.47 sq.m.)	2	2	2	...	New Property	NaN	NaN	1103.0	1	1	0	0	0	0
2	flat	pyramid elite	sector 86	0.46	79.0	58228.0	Carpet area: 58141 (5401.48 sq.m.)	2	2	1	...	Under Construction	NaN	NaN	58141.0	0	0	0	0	0	0

2 rows × 22 columns

Filling the values by using the ratio value of carpet_area, with this code snippet

```
c_df['built_up_area'].fillna(round(c_df['carpet_area']/0.9),inplace=True)
```

and updating to dataframe, with this code snippet:

```
df.update(c_df)
```

Next, we fill the values by using the ratio value of carpet_area, as shown in the below code:

```
c_df['built_up_area'].fillna(round(c_df['carpet_area']/0.9),inplace=True)
```

and update the dataframe.

```
df.update(c_df)
```

Finally, we create a dataframe for the anamoly records and replacing the area column values into the built_up_area column for anamolies. After that we update the main dataframe and delete the unwanted columns.

The code snippet for this is as follows:

```
anamoly_df = df[(df['built_up_area'] < 2000) & (df['price'] >
2.5)][['price','area','built_up_area']]

anamoly_df['built_up_area'] = anamoly_df['area']

df.update(anamoly_df)

df.drop(columns=['area','areaWithType','super_built_up_area','carpet_area'
],inplace=True)
```

Column: Facing

Now for the column “facing”, Since, it doesn't make much of interest for the given dataset, we will just remove this column.

```
df.drop(columns=['facing'], inplace=True)
```

Column: Floor_num

For the column “Floor_num”, we will as before find the number of missing values.

```
df['floorNum'].isnull().sum()
```

Output:

```
18
```

Then we will try to describe the nature of the column with the following code:

```
df['floorNum'].describe()
```

Output:

```
count   3781.000000
mean       6.810897
std        6.029070
min        0.000000
25%        2.000000
50%        5.000000
75%       10.000000
max       51.000000
Name: floorNum, dtype: float64
```

Now, We compute the mean 'floorNum' for each unique combination of 'property_type', 'society', and 'sector' by grouping the data accordingly. This allows us to capture the typical floor number within similar property types in the same society and sector.

Then, we iterate through rows with missing 'floorNum' values. For each row, we identify its 'property_type', 'society', and 'sector'. If there exists a mean 'floorNum' value for the corresponding combination, we replace the missing value with this mean.

This process ensures that missing 'floorNum' values are filled with sensible estimates based on comparable property types within the same society and sector.

The code for the above is as follows:

```
# Group by 'property_type', 'society', and 'sector' and calculate the mean
of 'floorNum'
mean_price_per_sqft = df.groupby(['property_type', 'society',
'sector'])['floorNum'].mean()

# Iterate over the rows with missing 'built_up_area' values
for index, row in df[df['floorNum'].isnull()].iterrows():
    # Extract the values of 'property_type', 'society', and 'sector' for
the current row
    property_type = row['property_type']
    society = row['society']
    sector = row['sector']

    # Check if there is a corresponding mean value for the combination
    if (property_type, society, sector) in mean_price_per_sqft.index:
        # Replace the missing value with the mean value for the
corresponding combination
        df.at[index, 'floorNum'] = mean_price_per_sqft.loc[(property_type,
society, sector)]

#Number of missing values
df['floorNum'].isnull().sum()
```

Output:

```
3
```

Now, we delete those 3 rows with the following code:

```
df.dropna(subset=['floorNum'], inplace=True)
df['floorNum'].isnull().sum()
```

Output:

```
0
```

Now, finally we find the number of missing values for all column to cross verify if we have dealt with all the missing values.

The following code will help us figure it out:

```
#Checking for missing values
missing = df.isnull().sum()
print(missing)
```

Output:

```
property_type       0
society             0
sector              0
price               0
price_per_sqft      0
areaWithType        0
bedRoom             0
bathroom            0
balcony             0
floorNum            0
agePossession       0
study room          0
servant room        0
store room          0
pooja room          0
others              0
furnishing_type     0
luxury_score        0
dtype: int64
```

From this output, we can see that none of the columns have any missing values, therefore we have successfully handled the missing values.

Task-3: Outlier Detection

Here, We start by selecting numerical columns from the DataFrame.

Then, we define a function to calculate the Z-scores for each numerical column, which helps in identifying outliers. Additionally, we create a function to count outliers using the Z-score method, considering a threshold of 3 standard deviations from the mean.

Another function calculates the Interquartile Range (IQR) for each numerical column, which is another method to detect outliers.

We apply both Z-score and IQR calculations to the numerical columns in the DataFrame.

Finally, we count the number of outliers using the Z-score method for each numerical column.

This process provides insights into the distribution of numerical data and identifies potential outliers based on statistical methods.

The following code snippet is used to implement this,

```
# Select numerical columns
numerical_cols = df.select_dtypes(include=[np.number]).columns

def calc_zscore(col):
    z_scores = (col - col.mean()) / col.std()
    return z_scores

# Function to count outliers using z-score method
def count_outliers_zscore(col, threshold=3):
    z_scores = calc_zscore(col)
    return (z_scores > threshold).sum()

def calc_iqr(col):
    Q1 = np.nanpercentile(col, 25)
    Q3 = np.nanpercentile(col, 75)
    IQR = Q3 - Q1
    return IQR

# Calculate z-score scores
zscore_scores = df[numerical_cols].apply(calc_zscore)

iqr_scores = df[numerical_cols].apply(calc_iqr)

# Count outliers using z-score method
outliers_count_zscore = zscore_scores.apply(count_outliers_zscore)

print("Z-score Scores:")
print(zscore_scores)
print()

print("\nIQR Scores:")
print(iqr_scores)
print()
```

```
print("Number of Outliers using Z-score Method:")
print(outliers_count_zscore)
print()
```

The output for the code:

Z-score Scores:

	price	price_per_sqft	bedRoom	bathroom	floorNum	built_up_area	\
0	-0.549166	-0.214519	-0.035491	-0.810526	-0.878196	NaN	
1	-0.498156	-0.164952	-0.803990	-0.810526	-0.554639	-0.062221	
2	-0.690426	-0.581074	-0.803990	-0.810526	-1.201752	NaN	
3	-0.745360	-0.317805	-0.803990	-0.810526	1.548476	-0.080594	
4	-0.243104	-0.193276	-0.035491	0.555330	0.416029	-0.037664	
...	...	...	...	...	...	...	
3797	-0.576633	0.178114	0.733009	0.555330	-1.039974	-0.116893	
3798	-0.725741	-0.275026	-0.803990	-0.810526	-0.392861	-0.110959	
3799	1.483399	-0.114456	1.501509	1.238258	-0.878196	0.284347	
3800	-0.635491	-0.186293	-1.572490	-1.493454	-0.392861	NaN	
3802	-0.172475	-0.139167	-0.035491	-0.127598	3.166257	-0.021609	

	study room	servant room	store room	pooja room	others	\
0	-0.478318	-0.754964	-0.289951	-0.428957	-0.349103	
1	2.090027	1.324167	-0.289951	-0.428957	-0.349103	
2	-0.478318	-0.754964	-0.289951	-0.428957	-0.349103	
3	-0.478318	-0.754964	-0.289951	-0.428957	-0.349103	
4	-0.478318	1.324167	-0.289951	-0.428957	2.863617	
...	...	...	...	...	...	
3797	-0.478318	-0.754964	-0.289951	-0.428957	-0.349103	

3798 -0.478318 -0.754964 -0.289951 -0.428957 -0.349103

3799 2.090027 1.324167 3.447820 2.330530 -0.349103

3800 -0.478318 -0.754964 -0.289951 -0.428957 -0.349103

3802 -0.478318 -0.754964 -0.289951 -0.428957 -0.349103

```
furnishing_type
0           -0.662097
1           -0.662097
2           -0.662097
3           -0.662097
4            1.033055
...               ...
3797        -0.662097
3798        -0.662097
3799        -0.662097
3800         1.033055
3802         1.033055

[3313 rows x 12 columns]
```

```
IQR Scores:
price                 1.430000
price_per_sqft     5556.000000
bedRoom               1.000000
bathroom              2.000000
floorNum              8.000000
built_up_area       665.176471
study room            0.000000
servant room          1.000000
store room            0.000000
pooja room            0.000000
others                0.000000
furnishing_type       1.000000
dtype: float64
```

```
Number of Outliers using Z-score Method:
price               77
price_per_sqft      21
bedRoom             55
bathroom            42
floorNum            49
built_up_area        6
study room           0
servant room         0
store room         257
pooja room           0
others               0
```

```
furnishing_type       0
dtype: int64
```

Now, We plot histograms for the numerical columns in the DataFrame, utilizing a figure size of 16x10 inches and 30 bins for better visualization. The histograms offer insights into the distribution of numerical data.

After plotting, we ensure tight layout to avoid overlap and then display the plot.

Additionally, we create a new figure with a size of 8x6 inches for further visualization.

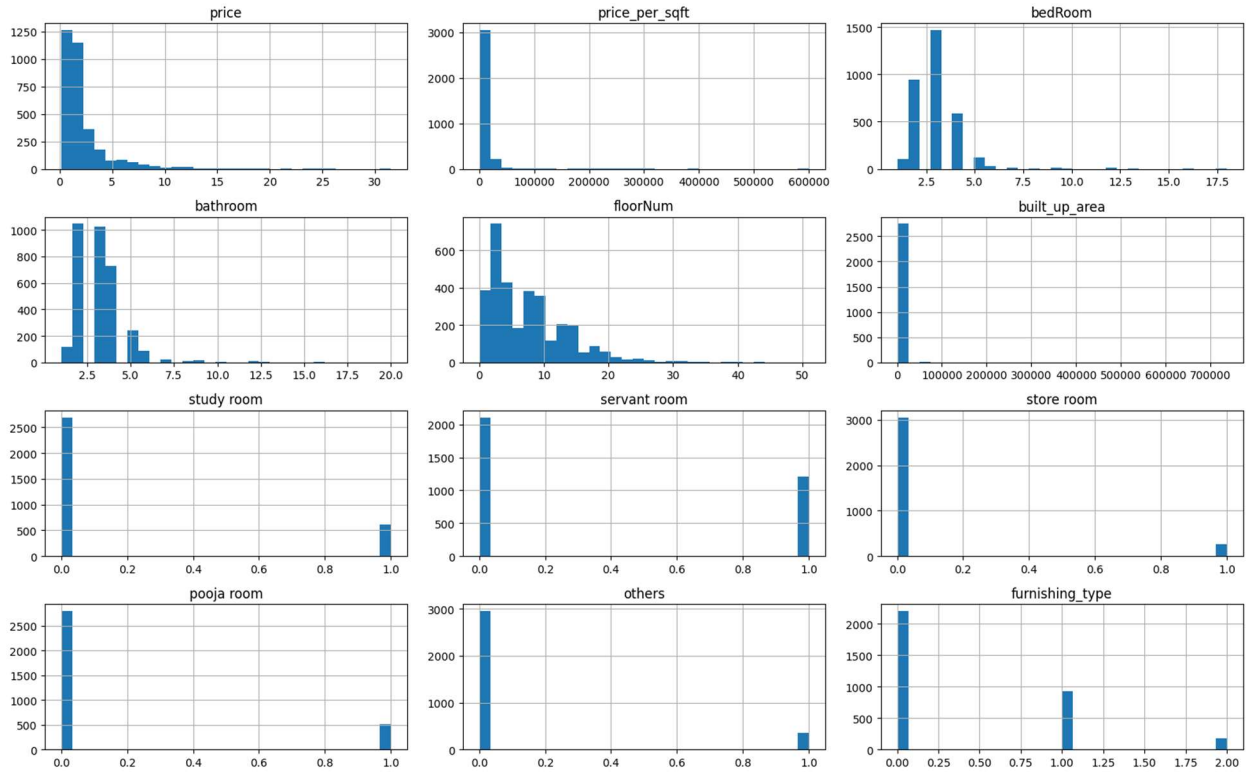
Following the plots, we print descriptive statistics for the numerical columns, providing key metrics like mean, standard deviation, minimum, maximum, and quartile values, aiding in understanding the central tendency and spread of the data.

This process combines visual exploration with statistical summaries to gain a comprehensive understanding of the numerical data distribution.

Here's the code to implement this:

```
#Plotting
df[numerical_cols].hist(figsize=(16, 10), bins=30)
plt.tight_layout()
plt.show()
plt.figure(figsize=(8, 6))
plt.show()
print(df[numerical_cols].describe())
```

The graphs that are generated in the output are as follows:



<Figure size 800x600 with 0 Axes>

	price	price_per_sqft	bedRoom	bathroom	floorNum
count	3313.000000	3313.000000	3313.000000	3313.000000	3313.000000
mean	2.219552	11977.739511	3.046182	3.186840	7.428396
std	2.548503	20477.135498	1.301237	1.464284	6.181306
min	0.160000	4.000000	1.000000	1.000000	0.000000
25%	0.920000	6666.000000	2.000000	2.000000	3.000000
50%	1.450000	8615.000000	3.000000	3.000000	6.000000
75%	2.350000	12222.000000	3.000000	4.000000	11.000000
max	31.500000	600000.000000	18.000000	20.000000	51.000000

	built_up_area	study room	servant room	store room	pooja room
count	2754.000000	3313.000000	3313.000000	3313.000000	3313.000000
mean	2154.557865	0.186236	0.363115	0.077573	0.155448
std	14325.608872	0.389356	0.480970	0.267539	0.362386
min	30.000000	0.000000	0.000000	0.000000	0.000000
25%	1300.000000	0.000000	0.000000	0.000000	0.000000
50%	1600.000000	0.000000	0.000000	0.000000	0.000000
75%	1965.176471	0.000000	1.000000	0.000000	0.000000
max	737147.000000	1.000000	1.000000	1.000000	1.000000

	others	furnishing_type

count	3313.000000	3313.000000
mean	0.108663	0.390583
std	0.311263	0.589918
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	1.000000
max	1.000000	2.000000

Now, for each numerical column in the DataFrame, we create box plots to visualize the distribution of data points and identify any potential outliers.

Each box plot is displayed in a figure with a size of 8x6 inches.

The seaborn library's boxplot function is used for generating the plots, providing a clear depiction of the median, quartiles, and the spread of the data.

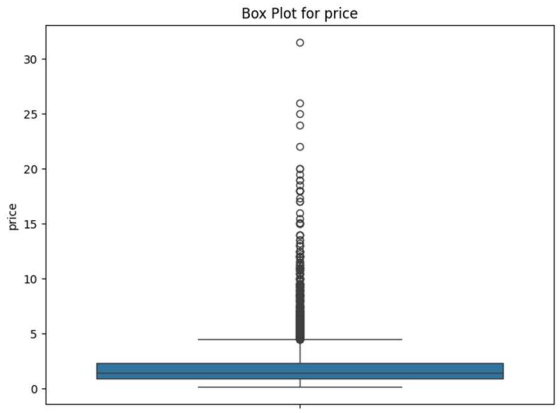
Additionally, each plot is titled according to the respective column name for easy identification.

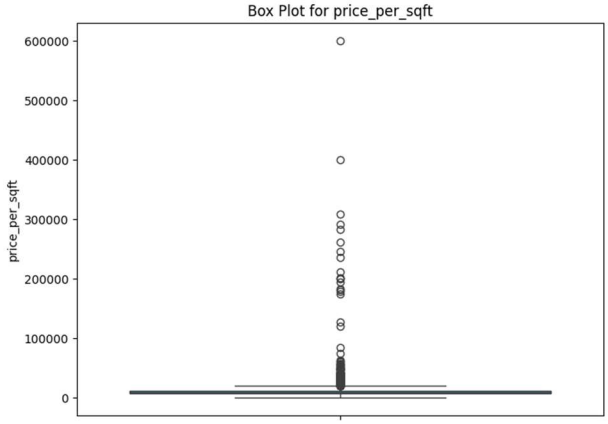
This approach facilitates the visual examination of each numerical feature's distribution and helps in detecting potential outliers or understanding the variability within the data.

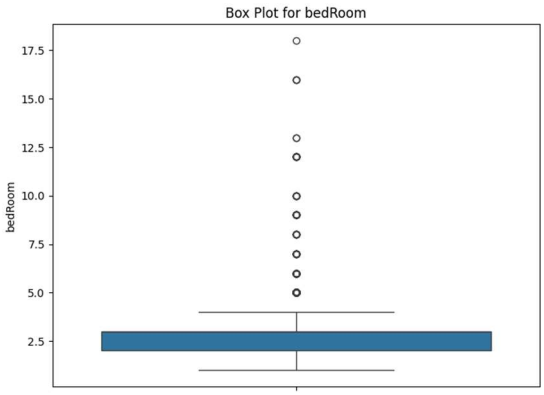
The code to implement the following is as follows:

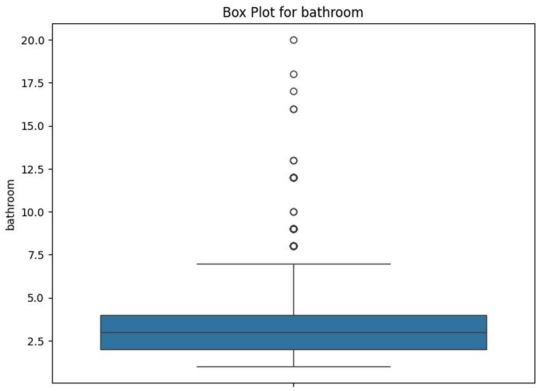
```
#Plotting
for col in numerical_cols:
    plt.figure(figsize=(8, 6))
    sns.boxplot(data=df[col])
    plt.title(f'Box Plot for {col}')
    plt.show()
```

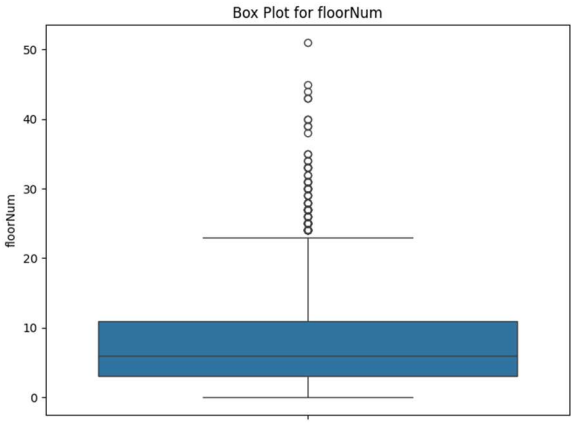
The graphs obtained in the output for this code is as follows:

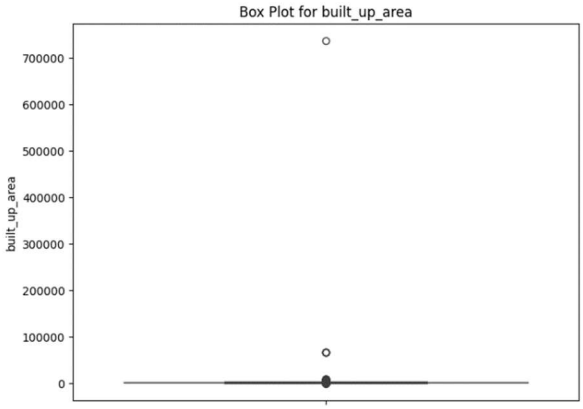


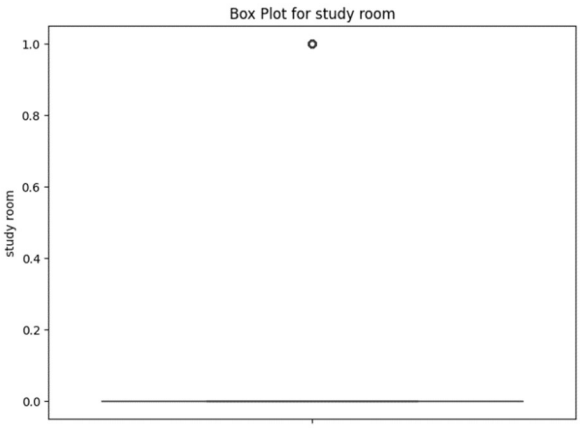


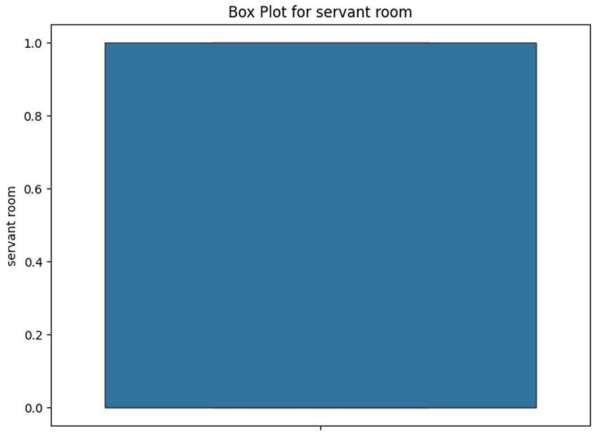

Box Plot for bedRoom
bedRoom
17.5
15.0
12.5
10.0
7.5
5.0
2.5

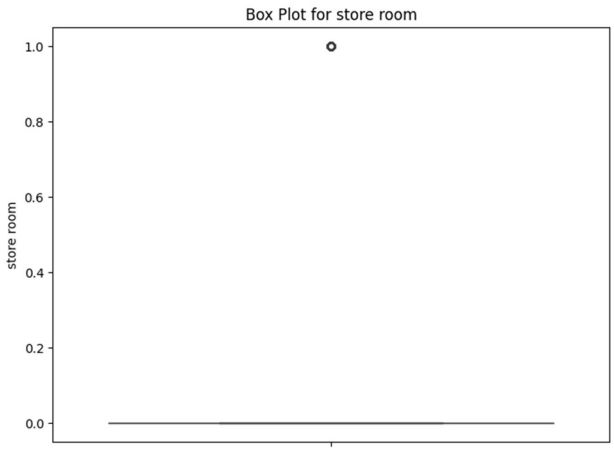

Box Plot for bathroom
bathroom
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5

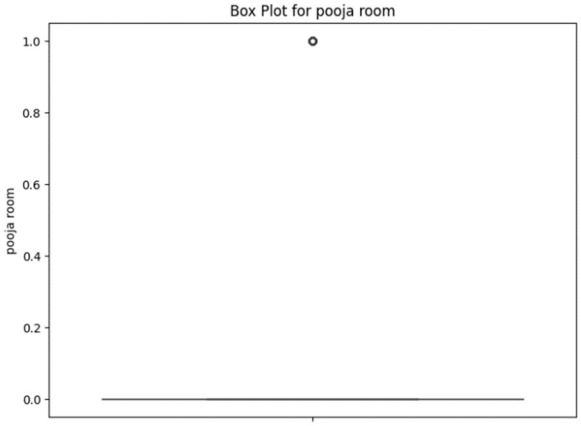

Box Plot for floorNum
floorNum
50
40
30
20
10
0

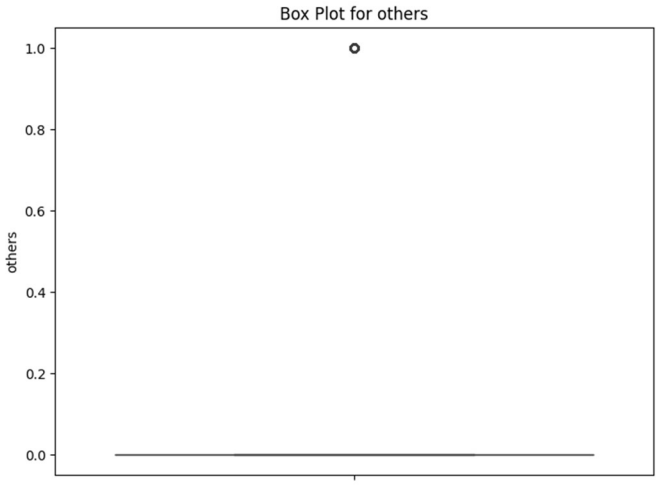

Box Plot for built_up_area
built_up_area
700000
600000
500000
400000
300000
200000
100000
0

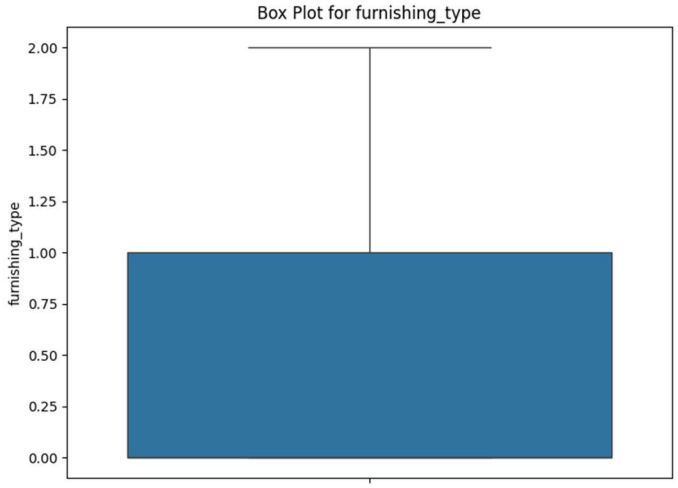

Box Plot for study room
study room
1.0
0.8
0.6
0.4
0.2
0.0


Box Plot for servant room
servant room
1.0
0.8
0.6
0.4
0.2
0.0


Box Plot for store room
store room
1.0
0.8
0.6
0.4
0.2
0.0


Box Plot for pooja room
pooja room
1.0
0.8
0.6
0.4
0.2
0.0





Task-4: Outliers Handling

Column: price

We have 89 outliers in this column

Here we will not use any outlier handling techniques because they are not potential outliers as the are all independent houses with many bedrooms, bathrooms and other rooms and many of them are furnished also.

Column: Price_Per_Sqft

We identify outliers in the 'price_per_sqft' column using the boxplot visualization. Based on the boxplot, we determine that there are 39 outliers in this column, which are likely data points significantly higher than the typical range.

To further investigate these outliers, we filter the DataFrame to include only rows where 'price_per_sqft' is greater than 50,000.

This filtering step allows us to examine the specific properties that have unusually high 'price_per_sqft' values, providing insights into anomalies or potential errors in the dataset.

the following code snipper was used to do this:

```
#We have 39 outliers in this column
#Based on the boxplot
df[df['price_per_sqft']>50000]
```

The output obtained for this code is as follows:

	property_type	society	sector	price	price_per_sqft	area	areaWithType	bedRoom	bathroom	balcony	floorNum	agePossession	study room	servant room	store room	pooja room	others	furnishing_type
229	house	rk excelo	sector 12	0.60	120000.0	50.0	Plot area 50(4.65 sq.m.)Built Up area: 30 sq.f...	5	3	2	3.0	Moderately Old	0	0	0	0	1	0
342	house	unitech nirvana birch court	sector 50	7.10	283333.0	251.0	Plot area 240(22.3 sq.m.)	3	3	3	1.0	Moderately Old	0	1	0	0	0	0
386	house	cloudnine cottages	sohna road	5.50	55000.0	1000.0	Plot area 1000(92.9 sq.m.)	3	3	0	1.0	Moderately Old	0	0	0	0	0	0
671	house	emaar mgf marbella	sector 66	18.00	55556.0	3240.0	Plot area 360(301.01 sq.m.)	4	4	3+	3.0	Moderately Old	1	1	0	1	0	2
867	house	ardee city	sector 52	5.50	183333.0	300.0	Plot area 300(27.87 sq.m.)	9	9	3+	3.0	Moderately Old	0	1	0	1	1	1
975	house	uppal southend	sector 49	6.75	290948.0	232.0	Plot area 232(21.55 sq.m.)	12	12	3+	3.0	Moderately Old	1	1	0	1	1	1
1027	house	dlf the grove	sector 54	5.70	211111.0	270.0	Built Up area: 270 (25.08 sq.m.)	4	4	0	1.0	Undefined	0	0	0	0	0	0
1259	house	vipul tatvam villa	sector 48	7.25	201388.0	360.0	Plot area 360(33.45 sq.m.)	4	4	3	2.0	Relatively New	1	1	1	1	0	1
1277	house	not applicable	sector 4	0.80	54794.0	146.0	Plot area 146(13.56 sq.m.)	5	4	3	3.0	Under Construction	0	0	0	0	0	0
1332	house	ansal api esencia	sector 67	1.85	74000.0	250.0	Carpet area: 250 (23.23 sq.m.)	3	3	3	1.0	Undefined	0	0	0	0	0	0
1349	flat	unitech vistas	sector 70	9.00	57507.0	1565.0	Built Up area: 1565 (145.39 sq.m.)	3	3	0	7.0	Undefined	0	0	0	0	0	0
1385	house	unitech unIworld resorts	sector 33	9.50	173992.0	546.0	Plot area 546(50.73 sq.m.)	6	6	3+	3.0	Moderately Old	0	0	0	0	0	1
1425	house	project housing board colony	sector 31	8.00	63492.0	1260.0	Built Up area: 140 (117.06 sq.m.)	2	1	0	1.0	Undefined	0	0	0	0	0	0
1628	house	eros rosewood villas	sector 49	2.75	178571.0	154.0	Plot area 154(14.31 sq.m.)Carpet	3	3	1	2.0	Old Property	0	0	0	0	1	1
1640	house	emaar the palm springs	sector 54	14.00	62222.0	2250.0	Plot area 250(209.03 sq.m.)	4	5	3+	2.0	Old Property	1	1	0	0	0	1
1696	house	ss omnia	sector 86	0.42	84000.0	50.0	Plot area 50(4.65 sq.m.)	5	3	2	3.0	Relatively New	0	0	0	0	0	0
1839	house	unitech escape	sector 50	10.80	60000.0	1800.0	Plot area 290(242.48 sq.m.)Built Up area: 250 ...	4	4	3	2.0	Relatively New	1	1	1	1	0	1
2047	house	ansal	sector 43	1.85	308333.0	60.0	Plot area 60(5.57 sq.m.)	8	8	3+	5.0	Relatively New	0	0	0	0	0	0
2161	house	dlf city plots phase 2	sector 25	10.50	261194.0	402.0	Plot area 402(37.35 sq.m.)	4	4	2	2.0	Old Property	0	1	0	1	0	1
2290	house	malibu towne	sector 47	8.50	53125.0	1600.0	Built Up area: 1600 (148.64 sq.m.)	12	12	3+	4.0	New Property	0	0	0	0	0	1
2418	flat	ambience caitriona	sector 24	14.00	200000.0	700.0	Built Up area: 700 (65.03 sq.m.)	4	5	3	3.0	Undefined	0	0	0	0	0	0
2488	house	dlf city plots	sector 26	26.00	57206.0	4545.0	Plot area 505(422.24 sq.m.)	6	7	3+	2.0	New Property	1	1	0	1	1	1
2738	house	vatika india next	sector 82	7.00	194444.0	360.0	Plot area 360(33.45 sq.m.)Built Up area: 3900 ...	4	4	3+	3.0	Relatively New	0	1	0	0	0	2
2850	house	dlf city plots phase 2	sector 25	10.00	400000.0	250.0	Plot area 250(23.23 sq.m.)	12	12	3+	4.0	Relatively New	1	1	0	0	0	1
3015	house	unitech deerwood chase	sector 50	8.45	235376.0	359.0	Plot area 359(33.35 sq.m.)	3	3	2	2.0	Old Property	1	1	0	0	0	1
3046	house	emaar the palm springs	sector 54	24.00	600000.0	400.0	Plot area 400(37.16 sq.m.)	5	5	2	1.0	Old Property	1	1	0	1	0	1
3090	flat	ambience caitriona	sector 24	14.00	200000.0	700.0	Built Up area: 700 (65.03 sq.m.)	4	5	3	3.0	Undefined	0	0	0	0	0	0
3148	house	bhim nagar, sector 6	sector 6	0.85	126865.0	67.0	Plot area 67(6.22 sq.m.)	5	2	2	1.0	Old Property	0	0	0	1	0	0
3279	house	unitech unIworld resorts	sector 33	10.00	181818.0	550.0	Plot area 550(51.1 sq.m.)	5	6	3	4.0	New Property	1	1	0	0	0	0
3285	house	nul	sector 28	12.50	51440.0	2430.0	Plot area 270(225.75 sq.m.)	16	17	3+	4.0	Relatively New	1	1	0	1	1	2
3519	house	huda plot sector 38	sector 38	4.30	245398.0	175.0	Plot area 163(15.14 sq.m.)Built Up area: 145 s...	13	13	3+	5.0	Relatively New	0	0	0	0	0	2

To handle the outliers identified in the 'price_per_sqft' column, we remove them from the DataFrame. We filter the DataFrame to include only the rows where 'price_per_sqft' is less than or equal to 50,000.

By doing this, we ensure that the dataset retains only the data points within the typical range, improving the accuracy of any subsequent analysis or modeling by eliminating the influence of extreme values.

Code snippet for this is as follows:

```
#Removing outliers otherwise
df = df[df['price_per_sqft']<=50000]
```

Column: Built_Up_Area

We identify outliers in the 'built_up_area' column using the boxplot visualization.

Based on the boxplot, we determine that there are 7 outliers in this column, which are data points significantly higher than the typical range.

To investigate these outliers further, we filter the DataFrame to include only rows where 'built_up_area' is greater than 200,000.

This step allows us to examine the specific properties that have unusually large 'built_up_area' values, providing insights into anomalies or potential errors in the dataset.

Code snippet for this is as follows:

```
#We have 7 outliers in this column
#Based on the boxplot
df[df['built_up_area']>200000]
```

Output for this code:

	property_type	society	sector	price	price_per_sqft	bedRoom	bathroom	balcony	floorNum	agePossession	built_up_area	study room	servant room	store room	pooja room	others	furnishing_type
1638	flat	signature global solera 2	sector 107	0.51	9.0	2	2	1	3.0	New Property	571551.0	0	0	0	0	0	0
1648	flat	hcbs sports ville	sohna road	0.35	4.0	2	2	2	8.0	Relatively New	737147.0	0	0	0	0	0	2
2111	flat	signature the rosella	sector 95	0.45	7.0	2	2	2	2.0	New Property	632492.0	0	0	0	0	0	0
2651	flat	ramsons kshitij	sector 95	0.31	5.0	2	2	1	1.0	Relatively New	675484.0	1	0	0	0	1	0
3021	house	independent	sector 50	5.00	232.0	6	5	3+	2.0	New Property	215517.0	1	1	0	1	1	1

Now, To handle the outliers identified in the 'built_up_area' column, we remove them from the DataFrame.

We filter the DataFrame to include only the rows where 'built_up_area' is less than or equal to 200,000.

By doing this, we ensure that the dataset retains only the data points within the typical range, enhancing the accuracy of any subsequent analysis or modeling by eliminating the influence of extreme values.

Code snippet for this:

```
#Removing outliers otherwise
df = df[df['built_up_area']<=200000]
```

Column: BedRoom, Bathroom, FloorNum, and StoreRoom

We have 112 outliers in Bedroom Column, 60 outliers in Bathroom Column, 58 outliers in FloorNum Column and 344 outliers in StoreRoom Column.

There is no need of removing the outliers in this column, Because these are not potential outliers.

According to the DataFrame, bigger houses will have more of these rooms

Bonus:

Code snippet for predicting:

```
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, r2_score

imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train_imputed, y_train)

y_train_pred = rf_model.predict(X_train_imputed)

y_test_pred = rf_model.predict(X_test_imputed)
```

```
train_rmse = mean_squared_error(y_train, y_train_pred, squared=False)
test_rmse = mean_squared_error(y_test, y_test_pred, squared=False)
train_r2_score = r2_score(y_train, y_train_pred)
test_r2_score = r2_score(y_test, y_test_pred)

print("Train RMSE:", train_rmse)
print("Test RMSE:", test_rmse)
print("Train R-squared score:", train_r2_score)
print("Test R-squared score:", test_r2_score)
```

Output for the above code:

```
Train RMSE: 0.15774741153387517
Test RMSE: 0.28121704716872825
Train R-squared score: 0.9958150022804465
Test R-squared score: 0.9857617064605074
```

--------THE END--------