House Price Prediction

Import Library

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

Load Dataset

In [2]:
```
df=pd.read_csv("MagicBricks.csv")
```

Top 3 data

In [3]:
```
df.head(3)
```

Out[3]:

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Status	Transaction	Type	Per_Sqft	Price
0	800.0	3	2.0	Semi-Furnished	Rohini Sector 25	1.0	Ready_to_move	New_Property	Builder_Floor	NaN	6500000
1	750.0	2	2.0	Semi-Furnished	J R Designers Floors, Rohini Sector 24	1.0	Ready_to_move	New_Property	Apartment	6667.0	5000000
2	950.0	2	2.0	Furnished	Citizen Apartment, Rohini Sector 13	1.0	Ready_to_move	Resale	Apartment	6667.0	15500000

Describe Data

Area: The total built-up area of the property (usually in square feet).

BHK: Number of bedrooms, halls, and kitchens in the property.

Bathroom: Total number of bathrooms in the property.

Furnishing: Indicates whether the property is furnished, semi-furnished, or unfurnished.

Locality: The specific area or neighborhood where the property is located.

Parking: Availability of parking space(s) for vehicles.

Status: Current condition of the property (e.g., ready to move, under construction).

Transaction: Whether the property is a new booking or a resale.

Type: Type of property (e.g., apartment, villa, independent house).

Per_Sqft: Price per square foot of the property.

Price: Total selling price of the property.

Size of Data

In [4]:
```
df.shape
```

Out[4]:
```
(1259, 11)
```

Check the data type

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Area         1259 non-null   float64
 1   BHK          1259 non-null   int64  
 2   Bathroom     1257 non-null   float64
 3   Furnishing   1254 non-null   object 
 4   Locality     1259 non-null   object 
 5   Parking      1226 non-null   float64
 6   Status       1259 non-null   object 
 7   Transaction  1259 non-null   object 
 8   Type         1254 non-null   object 
 9   Per_Sqft     1018 non-null   float64
 10  Price        1259 non-null   int64  
dtypes: float64(4), int64(2), object(5)
memory usage: 108.3+ KB
```

Describe the Numerical Data

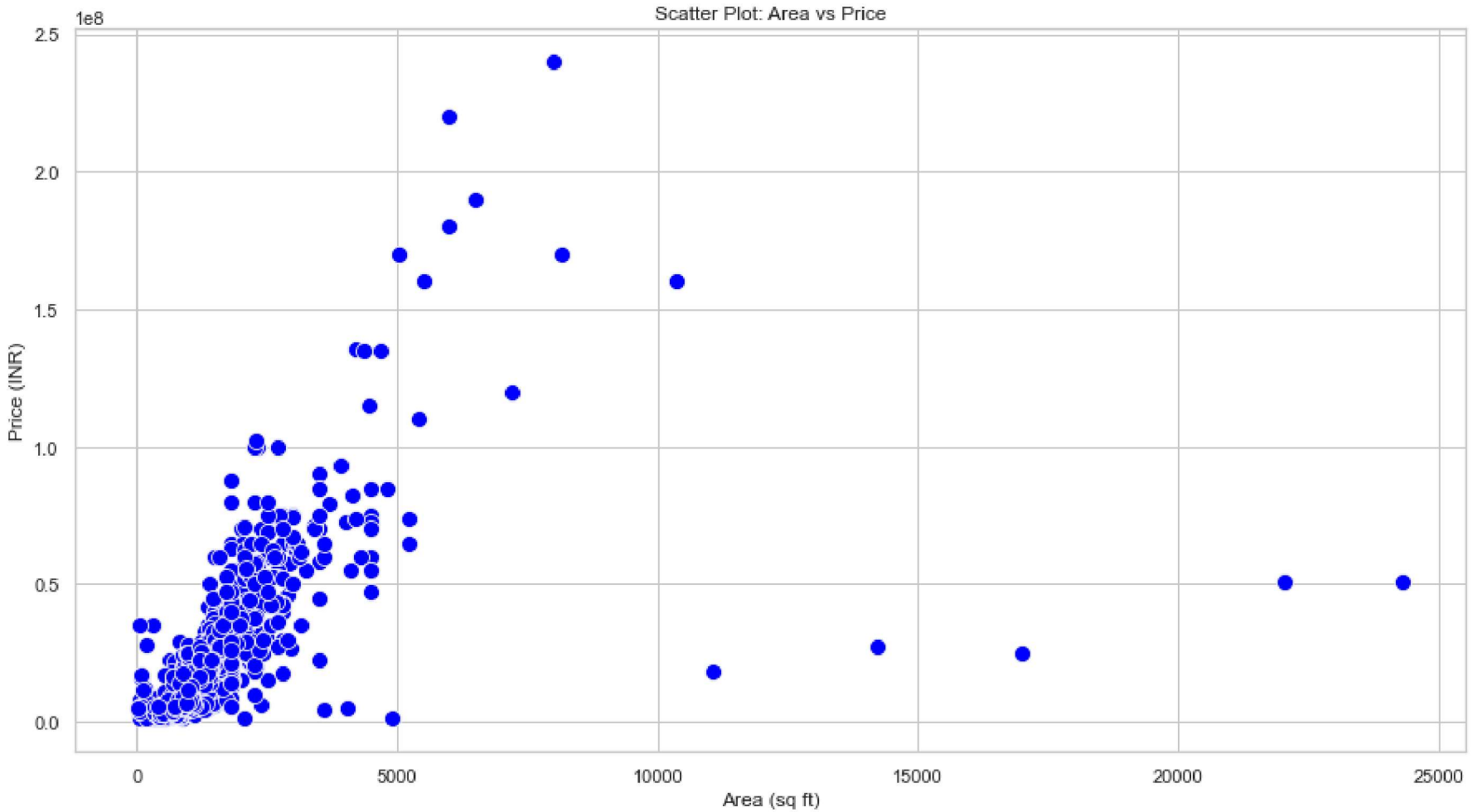
In [7]:
```
df.describe()
```

Out[7]:

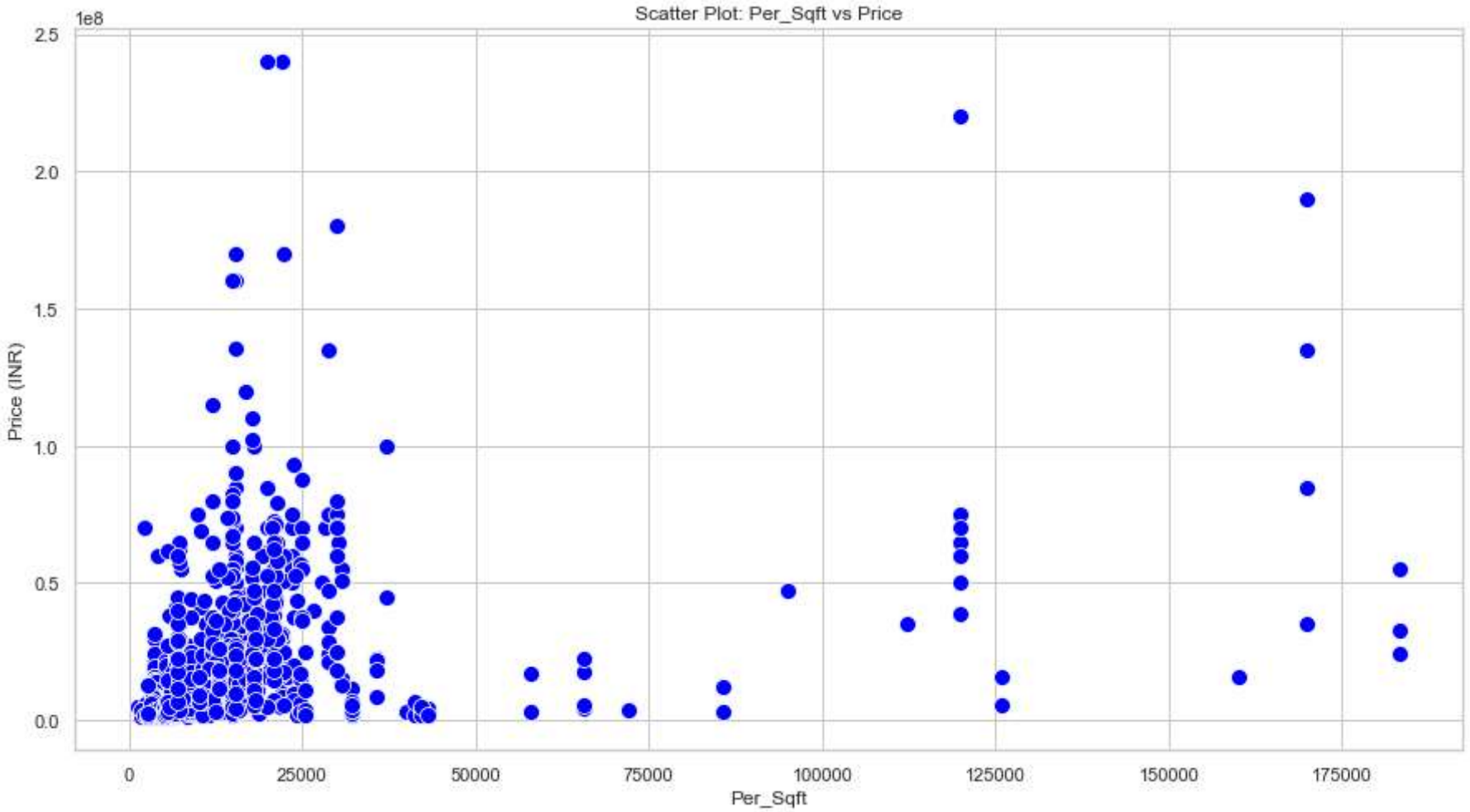
	Area	BHK	Bathroom	Parking	Per_Sqft	Price
count	1259.000000	1259.000000	1257.000000	1226.000000	1018.000000	1.259000e+03
mean	1466.452724	2.796664	2.556086	1.935563	15690.136542	2.130670e+07
std	1568.055040	0.954425	1.042220	6.279212	21134.738568	2.560115e+07
min	28.000000	1.000000	1.000000	1.000000	1259.000000	1.000000e+06
25%	800.000000	2.000000	2.000000	1.000000	6364.000000	5.700000e+06
50%	1200.000000	3.000000	2.000000	1.000000	11291.500000	1.420000e+07
75%	1700.000000	3.000000	3.000000	2.000000	18000.000000	2.550000e+07
max	24300.000000	10.000000	7.000000	114.000000	183333.000000	2.400000e+08

In [8]:

```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.scatterplot(x='Area', y='Price', data=df, color='blue', s=100)

plt.title('Scatter Plot: Area vs Price')
plt.xlabel('Area (sq ft)')
plt.ylabel('Price (INR)')
plt.show()
```



In [9]:
```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.scatterplot(x='Per_Sqft', y='Price', data=df, color='blue', s=100)

plt.title('Scatter Plot: Per_Sqft vs Price')
plt.xlabel('Per_Sqft')
plt.ylabel('Price (INR)')
plt.show()
```



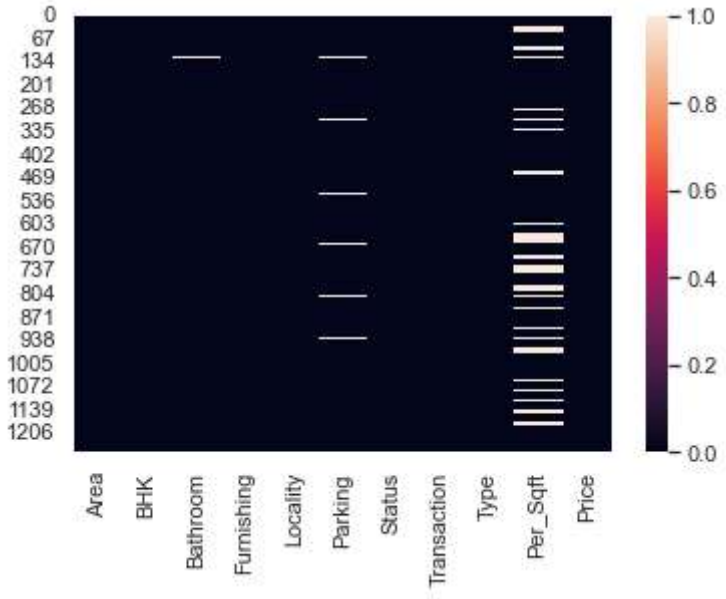
Data Cleaning

Check the Null Value

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
Area             0
BHK              0
Bathroom         2
Furnishing       5
Locality         0
Parking         33
Status           0
Transaction      0
Type             5
Per_Sqft       241
Price            0
dtype: int64
```

In [12]:
```
sns.heatmap(df.isnull())
plt.show()
```



Per_sqft has almost maximum of null value. We will fill them

In [13]:
```
a=pd.DataFrame(df["Price"]//df["Area"],columns=["x"])
```

In [14]:
```
a.head()
```

Out[14]:

	x
0	8125.0
1	6666.0
2	16315.0
3	7000.0
4	9538.0

In [15]:
```
df["Per_Sqft"]=a["x"]
```

In [16]:
```
df.head()
```

Out[16]:

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Status	Transaction	Type	Per_Sqft	Price
0	800.0	3	2.0	Semi-Furnished	Rohini Sector 25	1.0	Ready_to_move	New_Property	Builder_Floor	8125.0	6500000
1	750.0	2	2.0	Semi-Furnished	J R Designers Floors, Rohini Sector 24	1.0	Ready_to_move	New_Property	Apartment	6666.0	5000000
2	950.0	2	2.0	Furnished	Citizen Apartment, Rohini Sector 13	1.0	Ready_to_move	Resale	Apartment	16315.0	15500000
3	600.0	2	2.0	Semi-Furnished	Rohini Sector 24	1.0	Ready_to_move	Resale	Builder_Floor	7000.0	4200000
4	650.0	2	2.0	Semi-Furnished	Rohini Sector 24 carpet area 650 sqft status R...	1.0	Ready_to_move	New_Property	Builder_Floor	9538.0	6200000

In [18]:
```
df.isnull().sum()
```

Out[18]:
```
Area            0
BHK             0
Bathroom        2
Furnishing      5
Locality        0
Parking        33
Status          0
Transaction     0
Type            5
Per_Sqft        0
Price           0
dtype: int64
```

Drop other null Value

In [19]:
```
df.dropna(inplace = True)
```

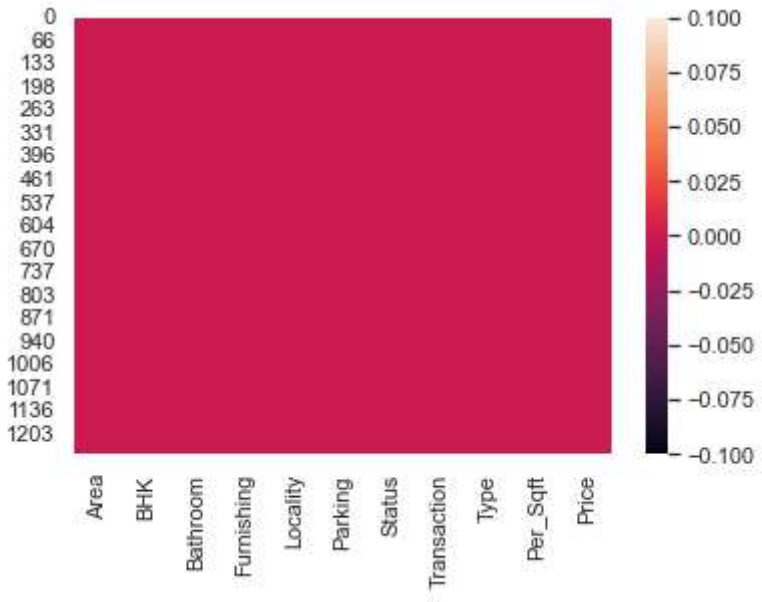
In [20]:
```
df.isnull().sum()
```

Out[20]:
```
Area            0
BHK             0
Bathroom        0
Furnishing      0
Locality        0
Parking         0
Status          0
Transaction     0
Type            0
Per_Sqft        0
Price           0
dtype: int64
```

In [21]:
```
df.shape
```

Out[21]:
```
(1225, 11)
```

In [22]:
```
sns.heatmap(df.isnull())
plt.show()
```



Finding Duplicate

In [23]:
```
df.duplicated().sum()
```

Out[23]:
```
103
```

Drop Duplicate Value

In [24]:
```
df.drop_duplicates(inplace = True)
```

In [25]:
```
df.shape
```

Out[25]:
```
(1122, 11)
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1122 entries, 0 to 1258
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Area         1122 non-null   float64
 1   BHK          1122 non-null   int64  
 2   Bathroom     1122 non-null   float64
 3   Furnishing   1122 non-null   object 
 4   Locality     1122 non-null   object 
 5   Parking      1122 non-null   float64
 6   Status       1122 non-null   object 
 7   Transaction  1122 non-null   object 
 8   Type         1122 non-null   object 
 9   Per_Sqft     1122 non-null   float64
 10  Price        1122 non-null   int64  
dtypes: float64(4), int64(2), object(5)
memory usage: 105.2+ KB
```

Exploratory Data Analysis

Convert float to int

In [27]:
```
df[['Parking','Bathroom','Per_Sqft','Area']]=df[['Parking','Bathroom','Per_Sqft','Area']].astype('int64')
```

In [28]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1122 entries, 0 to 1258
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Area         1122 non-null   int64
 1   BHK          1122 non-null   int64
 2   Bathroom     1122 non-null   int64
 3   Furnishing   1122 non-null   object
 4   Locality     1122 non-null   object
 5   Parking      1122 non-null   int64
 6   Status       1122 non-null   object
 7   Transaction  1122 non-null   object
 8   Type         1122 non-null   object
 9   Per_Sqft     1122 non-null   int64
 10  Price        1122 non-null   int64
dtypes: int64(6), object(5)
memory usage: 105.2+ KB
```

In [29]:
```
df.head()
```

Out[29]:

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Status	Transaction	Type	Per_Sqft	Price
0	800	3	2	Semi-Furnished	Rohini Sector 25	1	Ready_to_move	New_Property	Builder_Floor	8125	6500000
1	750	2	2	Semi-Furnished	J R Designers Floors, Rohini Sector 24	1	Ready_to_move	New_Property	Apartment	6666	5000000
2	950	2	2	Furnished	Citizen Apartment, Rohini Sector 13	1	Ready_to_move	Resale	Apartment	16315	15500000
3	600	2	2	Semi-Furnished	Rohini Sector 24	1	Ready_to_move	Resale	Builder_Floor	7000	4200000
4	650	2	2	Semi-Furnished	Rohini Sector 24 carpet area 650 sqft status R...	1	Ready_to_move	New_Property	Builder_Floor	9538	6200000

Correlation

In [30]:
```
round(df.corr(), 3)
```

Out[30]:

	Area	BHK	Bathroom	Parking	Per_Sqft	Price
Area	1.000	0.462	0.551	-0.011	0.019	0.592
BHK	0.462	1.000	0.779	-0.072	0.211	0.579
Bathroom	0.551	0.779	1.000	-0.035	0.265	0.740
Parking	-0.011	-0.072	-0.035	1.000	0.002	-0.001
Per_Sqft	0.019	0.211	0.265	0.002	1.000	0.338
Price	0.592	0.579	0.740	-0.001	0.338	1.000

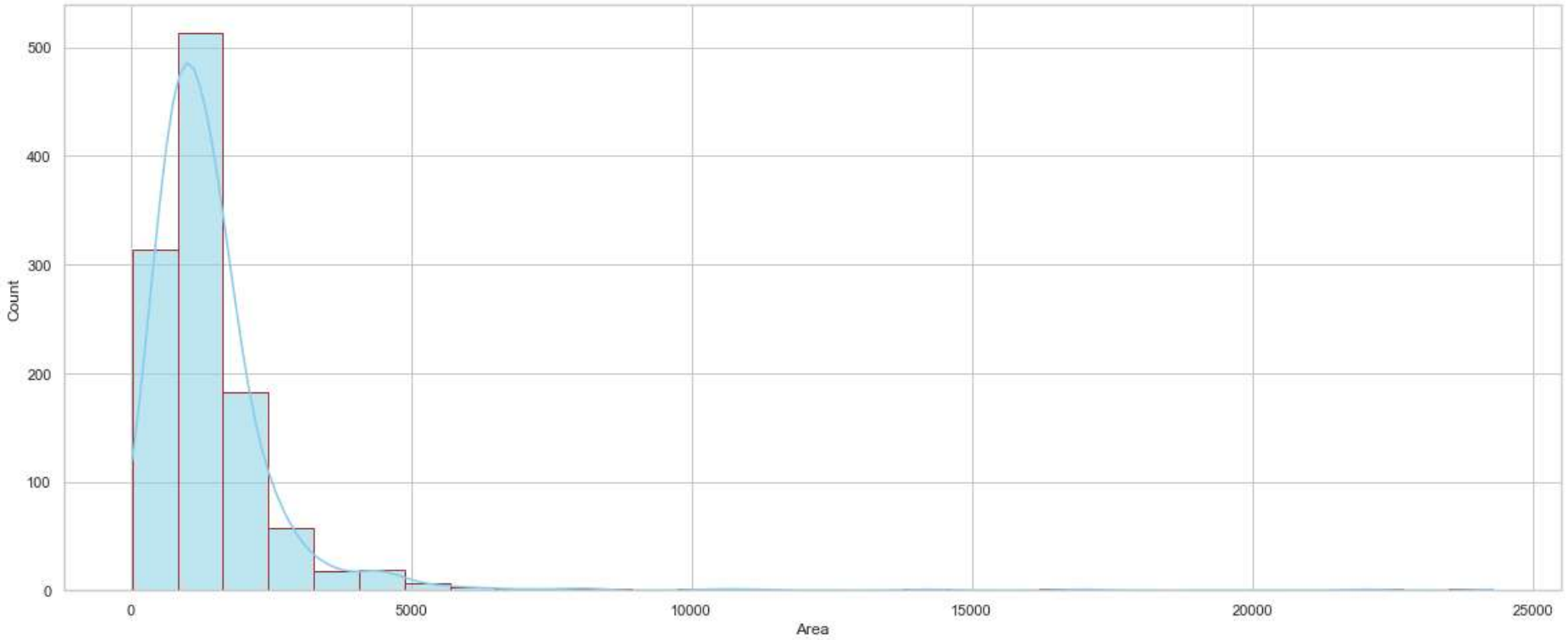
In [31]:
```
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(),annot=True)
plt.show()
```

In [32]:
```
df.nunique()
```

Out[32]:
```
Area           310
BHK              8
Bathroom         7
Furnishing       3
Locality       353
Parking          9
Status           2
Transaction      2
Type             2
Per_Sqft       781
Price          279
dtype: int64
```

Data Visualization

In [33]:
```
plt.figure(figsize=(20,8))
sns.histplot(df["Area"], bins=30, kde=True, color='skyblue', edgecolor='Brown')
plt.show()
```



In [34]:
```
df["BHK"].value_counts()
```

Out[34]:
```
3     480
2     327
4     197
1      90
5      20
6       6
7       1
10      1
Name: BHK, dtype: int64
```

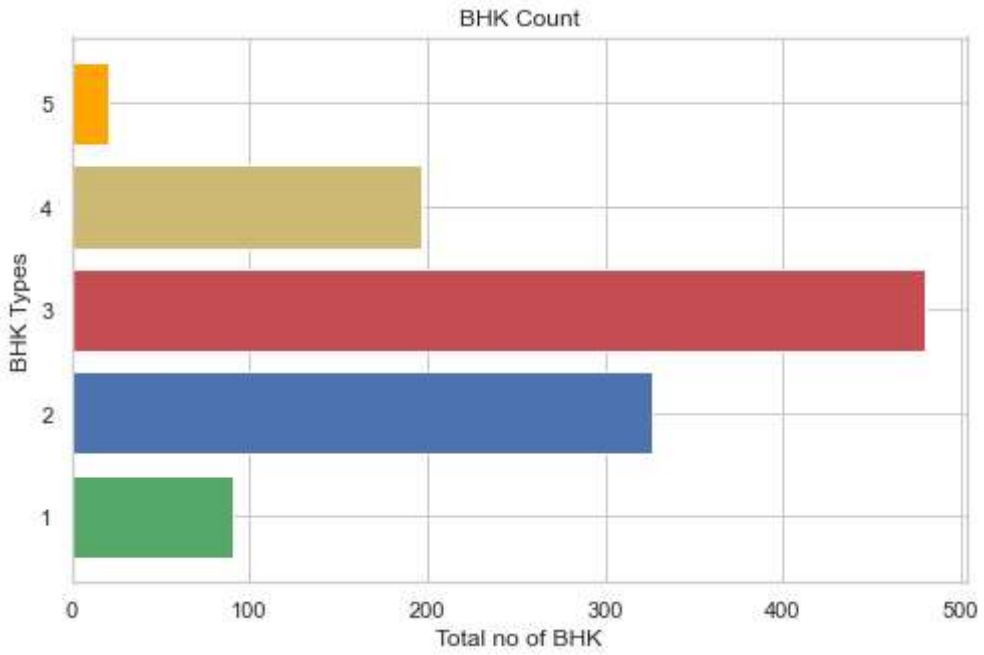
In [35]:
```
df["BHK"].value_counts()[0:5]
```

Out[35]:
```
3    480
2    327
4    197
1     90
5     20
Name: BHK, dtype: int64
```

In [36]:
```
list(df["BHK"].value_counts()[0:5].keys())
```

Out[36]:
```
[3, 2, 4, 1, 5]
```

In [37]:
```
plt.figure(figsize=(8,5))
plt.title("BHK Count")
plt.xlabel("Total no of BHK")
plt.ylabel("BHK Types")
plt.barh(list(df["BHK"].value_counts()[0:5].keys()),list(df["BHK"].value_counts()[0:5]),color=['r','b','y','g','orange'])
plt.show()
```



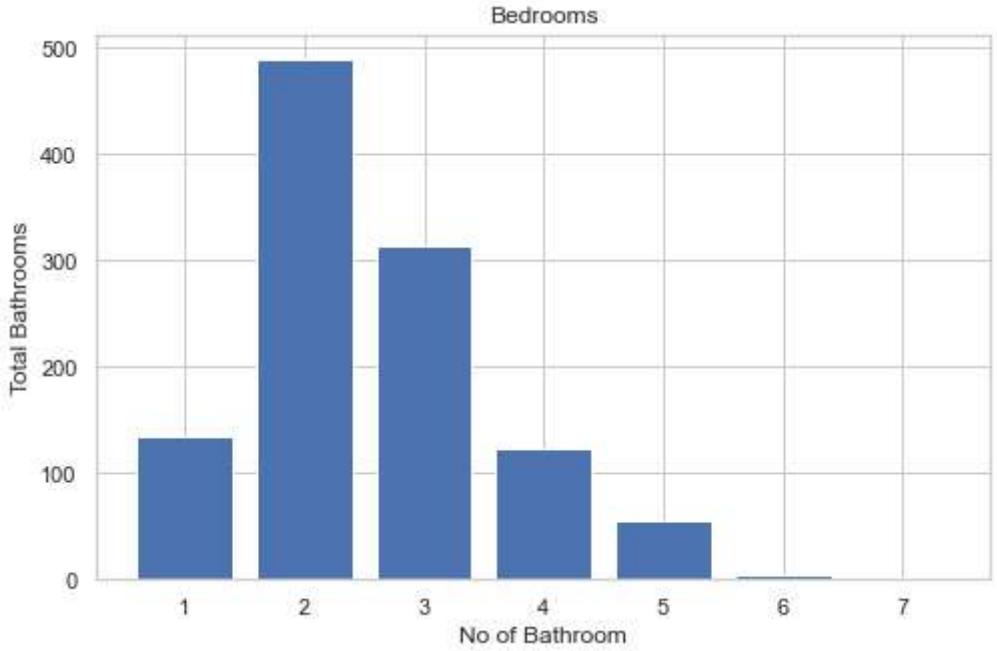
In [38]:
```
df["Bathroom"].value_counts()
```

Out[38]:
```
2    488
3    313
1    134
4    124
5     55
6      5
7      3
Name: Bathroom, dtype: int64
```

In [39]:
```
list(df["Bathroom"].value_counts().keys())
```

Out[39]:
```
[2, 3, 1, 4, 5, 6, 7]
```

In [40]:
```
plt.figure(figsize=(8,5))
plt.title("Bedrooms")
plt.xlabel("No of Bathroom")
plt.ylabel("Total Bathrooms")
plt.bar(list(df["Bathroom"].value_counts().keys()),list(df["Bathroom"].value_counts()))
plt.show()
```



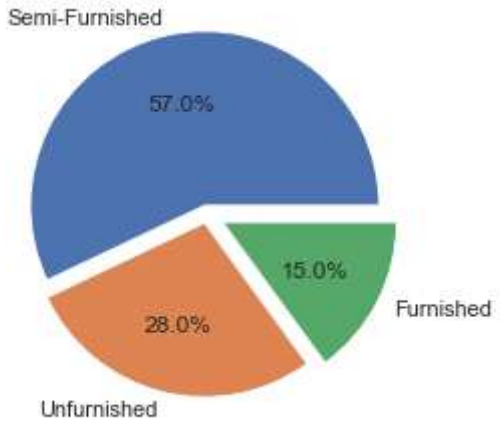
In [41]:
```
df["Furnishing"].value_counts()
```

Out[41]:
```
Semi-Furnished    640
Unfurnished       314
Furnished         168
Name: Furnishing, dtype: int64
```

In [42]:
```
list(df["Furnishing"].value_counts().keys())
```

Out[42]:
```
['Semi-Furnished', 'Unfurnished', 'Furnished']
```

In [43]:
```
plt.pie(df["Furnishing"].value_counts(),labels=list(df["Furnishing"].value_counts().keys()),autopct="%0.1f%%",explode=[0.05,0.05,
plt.show()
```



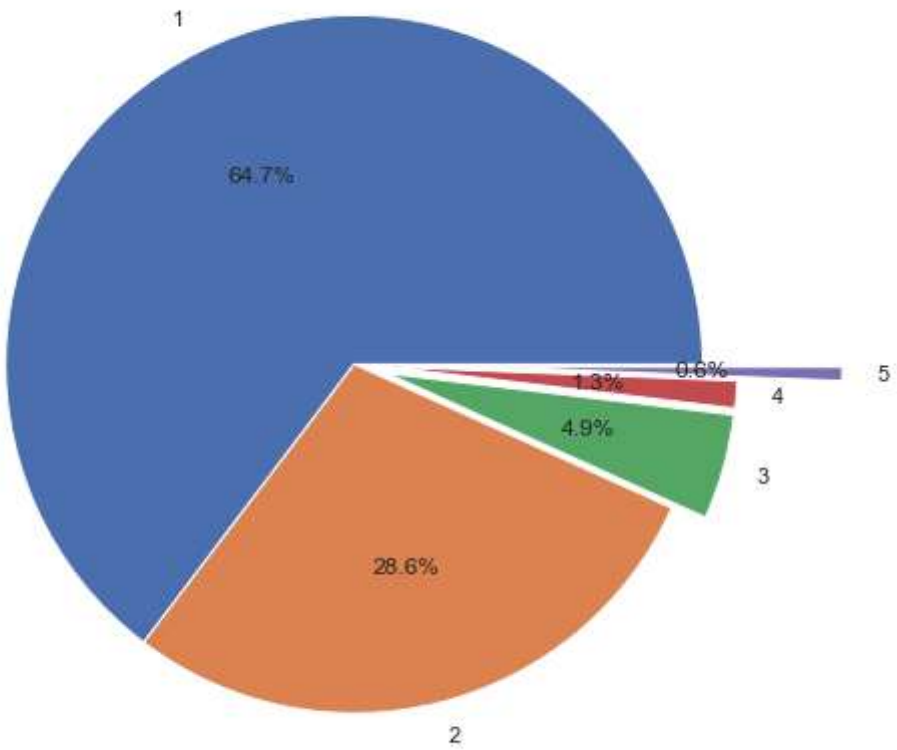
In [44]:
```
df["Parking"].value_counts()
```

Out[44]:
```
1      718
2      317
3       54
4       14
5        7
39       7
114      3
9        1
10       1
Name: Parking, dtype: int64
```

In [45]:
```
list(df["Parking"].value_counts()[0:5].keys())
```

Out[45]:
```
[1, 2, 3, 4, 5]
```

In [46]:
```
plt.figure(figsize=(10,8))
plt.pie(df["Parking"].value_counts()[0:5],labels=list(df["Parking"].value_counts()[0:5].keys()),autopct="%0.1f%%",explode=[0,0,0.
plt.show()
```



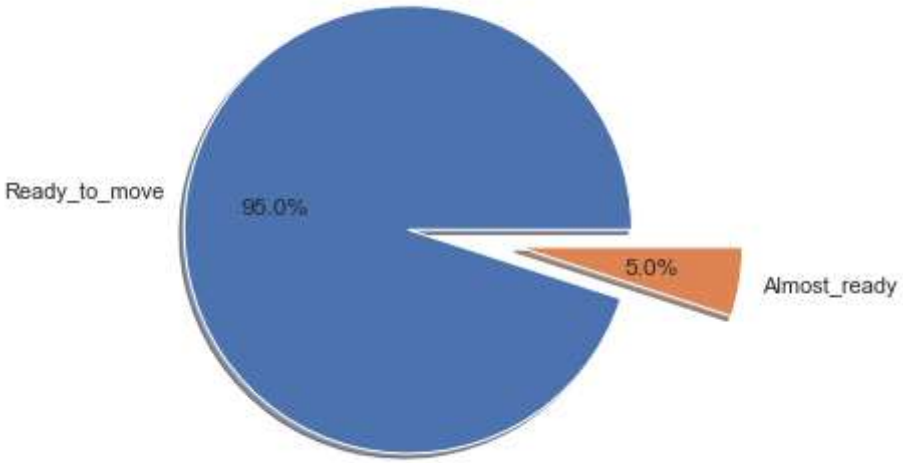
In [47]:
```
df["Status"].value_counts()
```

Out[47]:
```
Ready_to_move    1066
Almost_ready       56
Name: Status, dtype: int64
```

In [48]:
```
list(df["Status"].value_counts().keys())
```

Out[48]:
```
['Ready_to_move', 'Almost_ready']
```

In [49]:
```
plt.figure(figsize=(5,6))
plt.pie(df["Status"].value_counts(),labels=list(df["Status"].value_counts().keys()),autopct="%0.1f%%",explode=[0,0.5],shadow=True
plt.show()
```



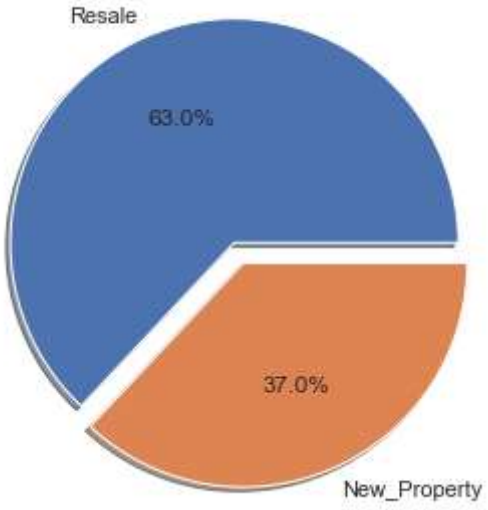
In [50]:
```
df["Transaction"].value_counts()
```

Out[50]:
```
Resale          707
New_Property    415
Name: Transaction, dtype: int64
```

In [51]:
```
list(df["Transaction"].value_counts().keys())
```

Out[51]:
```
['Resale', 'New_Property']
```

In [52]:
```
plt.figure(figsize=(5,6))
plt.pie(df["Transaction"].value_counts(),labels=list(df["Transaction"].value_counts().keys()),autopct="%0.1f%%",explode=[0,0.1],s
plt.show()
```



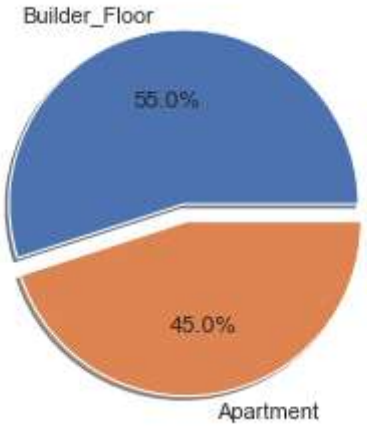
In [53]:
```
df["Type"].value_counts()
```

Out[53]:
```
Builder_Floor    617
Apartment        505
Name: Type, dtype: int64
```

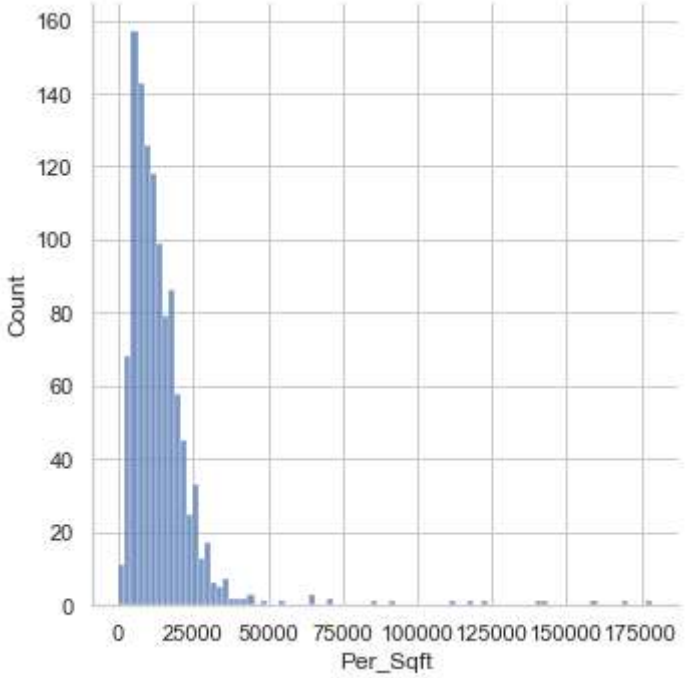
In [54]:
```
list(df["Type"].value_counts().keys())
```

Out[54]:
```
['Builder_Floor', 'Apartment']
```

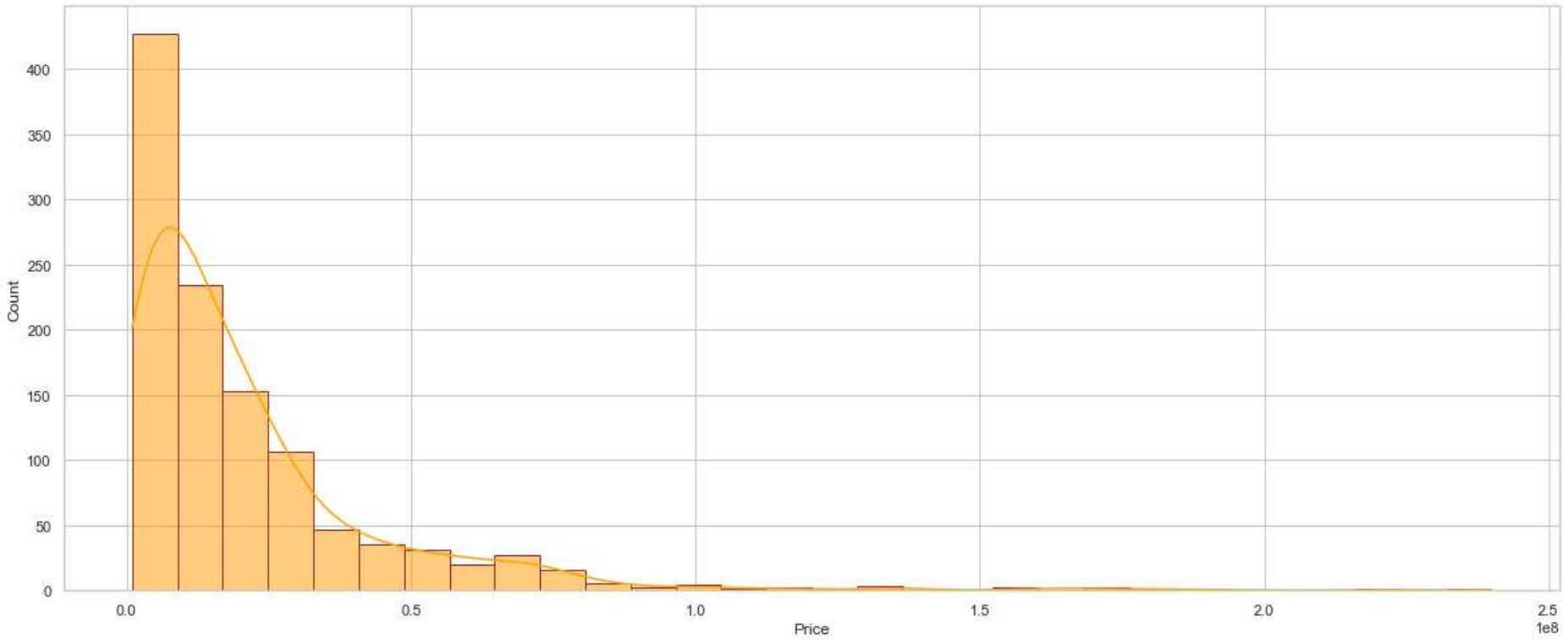
In [55]:
```
plt.pie(df["Type"].value_counts(),labels=list(df["Type"].value_counts().keys()),autopct="%0.1f%%",shadow=True,explode=[0,0.1])
plt.show()
```



In [56]:
```
sns.displot(df['Per_Sqft'])
plt.show()
```



In [57]:
```
plt.figure(figsize=(20,8))
sns.histplot(df["Price"], bins=30, kde=True, color='orange', edgecolor='Brown')
plt.show()
```



In [58]:
```
df["Price"].max(),df["Price"].min()
```

Out[58]:
```
(240000000, 1000000)
```

In [59]:
```
df["Per_Sqft"].max(),df["Per_Sqft"].min()
```

Out[59]:
```
(178571, 204)
```

Convert categorical data to numerical

In [60]:
```
df.columns
```

Out[60]:
```
Index(['Area', 'BHK', 'Bathroom', 'Furnishing', 'Locality', 'Parking',
       'Status', 'Transaction', 'Type', 'Per_Sqft', 'Price'],
      dtype='object')
```

In [61]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1122 entries, 0 to 1258
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Area         1122 non-null   int64 
 1   BHK          1122 non-null   int64 
 2   Bathroom     1122 non-null   int64 
 3   Furnishing   1122 non-null   object
 4   Locality     1122 non-null   object
 5   Parking      1122 non-null   int64 
 6   Status       1122 non-null   object
 7   Transaction  1122 non-null   object
 8   Type         1122 non-null   object
 9   Per_Sqft     1122 non-null   int64 
 10  Price        1122 non-null   int64 
dtypes: int64(6), object(5)
memory usage: 105.2+ KB
```

In [62]:
```
from sklearn.preprocessing import LabelEncoder
```

In [63]:
```
le=LabelEncoder()
```

In [64]:
```
df["Furnishing"]=le.fit_transform(df["Furnishing"])
```

In [65]:
```
df["Locality"]=le.fit_transform(df["Locality"])
df["Status"]=le.fit_transform(df["Status"])
df["Transaction"]=le.fit_transform(df["Transaction"])
df["Type"]=le.fit_transform(df["Type"])
```

In [67]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1122 entries, 0 to 1258
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Area         1122 non-null   int64
 1   BHK          1122 non-null   int64
 2   Bathroom     1122 non-null   int64
 3   Furnishing   1122 non-null   int32
 4   Locality     1122 non-null   int32
 5   Parking      1122 non-null   int64
 6   Status       1122 non-null   int32
 7   Transaction  1122 non-null   int32
 8   Type         1122 non-null   int32
 9   Per_Sqft     1122 non-null   int64
 10  Price        1122 non-null   int64
dtypes: int32(5), int64(6)
memory usage: 83.3 KB
```

In [68]:
```
df[['Furnishing','Locality','Status','Transaction','Type']]=df[['Furnishing','Locality','Status','Transaction','Type']].astype('i
```

In [69]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1122 entries, 0 to 1258
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Area         1122 non-null   int64
 1   BHK          1122 non-null   int64
 2   Bathroom     1122 non-null   int64
 3   Furnishing   1122 non-null   int64
 4   Locality     1122 non-null   int64
 5   Parking      1122 non-null   int64
 6   Status       1122 non-null   int64
 7   Transaction  1122 non-null   int64
 8   Type         1122 non-null   int64
 9   Per_Sqft     1122 non-null   int64
 10  Price        1122 non-null   int64
dtypes: int64(11)
memory usage: 105.2 KB
```

In [70]:
```
df.head(3)
```

Out[70]:

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Status	Transaction	Type	Per_Sqft	Price
0	800	3	2	1	274	1	1	0	1	8125	6500000
1	750	2	2	1	135	1	1	0	0	6666	5000000
2	950	2	2	0	49	1	1	1	0	16315	15500000

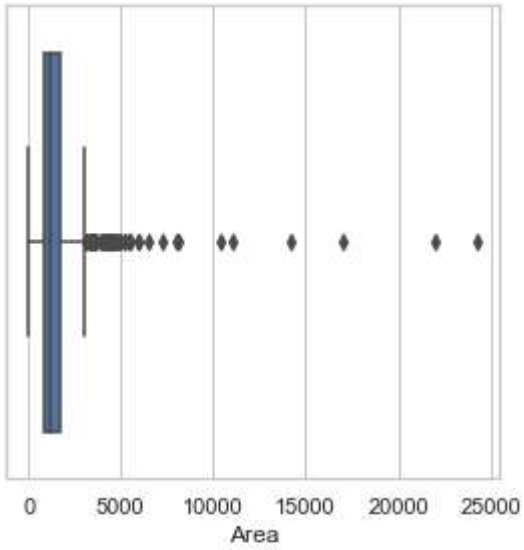
Check the Outliers

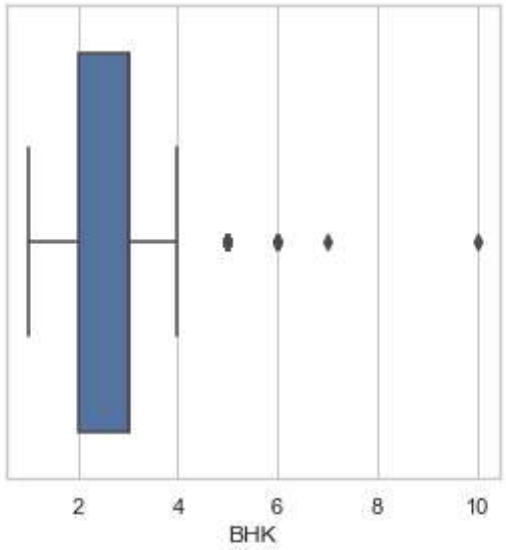
In [71]:
```
df.columns
```

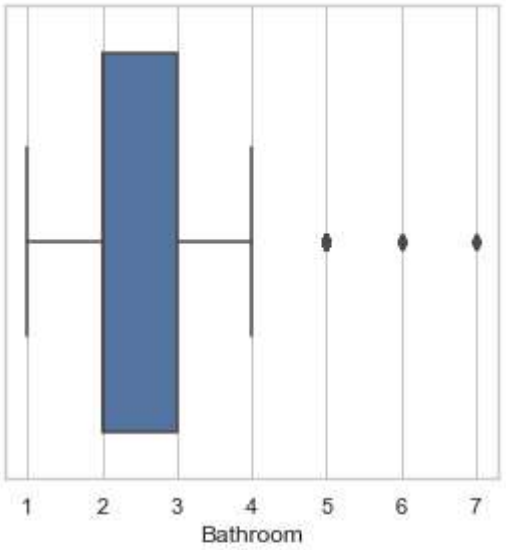
Out[71]:
```
Index(['Area', 'BHK', 'Bathroom', 'Furnishing', 'Locality', 'Parking',
       'Status', 'Transaction', 'Type', 'Per_Sqft', 'Price'],
      dtype='object')
```

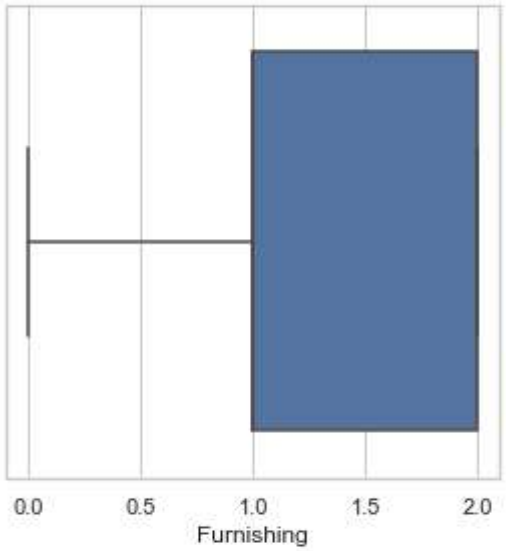
In [72]:
```
plt.figure(figsize=(15,20))
plt.subplot(4,3,1)
sns.boxplot(x=df["Area"])

plt.subplot(4,3,2)
sns.boxplot(x=df["BHK"])

plt.subplot(4,3,3)
sns.boxplot(x=df["Bathroom"])

plt.subplot(4,3,4)
sns.boxplot(x=df["Furnishing"])

plt.subplot(4,3,5)
sns.boxplot(x=df["Locality"])

plt.subplot(4,3,6)
sns.boxplot(x=df["Parking"])

plt.subplot(4,3,7)
sns.boxplot(x=df["Status"])

plt.subplot(4,3,8)
sns.boxplot(x=df["Transaction"])
```

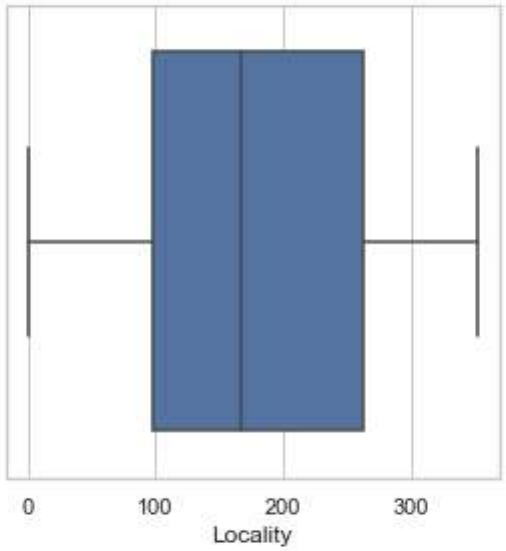
```
plt.subplot(4,3,9)
sns.boxplot(x=df["Type"])

plt.subplot(4,3,10)
sns.boxplot(x=df["Per_Sqft"])

plt.subplot(4,3,11)
sns.boxplot(x=df["Price"])

plt.show()
```

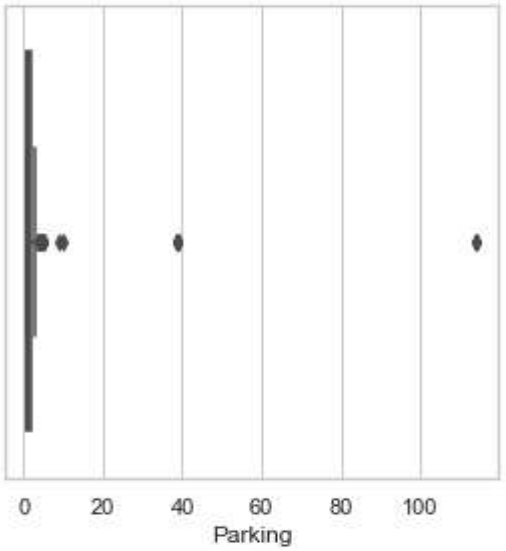

0
5000
10000
15000
20000
25000
Area

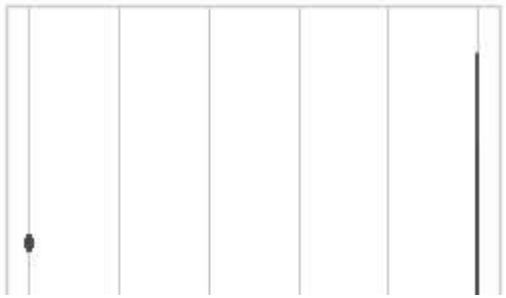

2
4
6
8
10
BHK

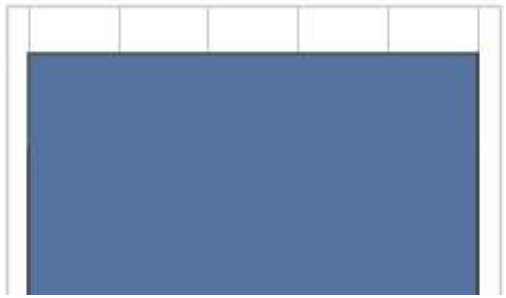

1
2
3
4
5
6
7
Bathroom

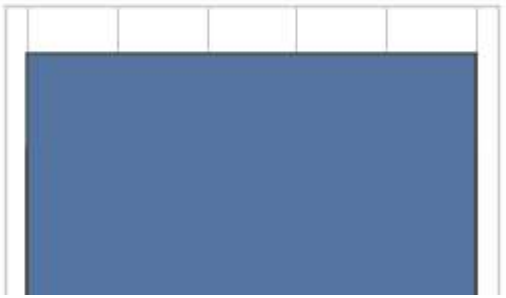

0.0
0.5
1.0
1.5
2.0
Furnishing

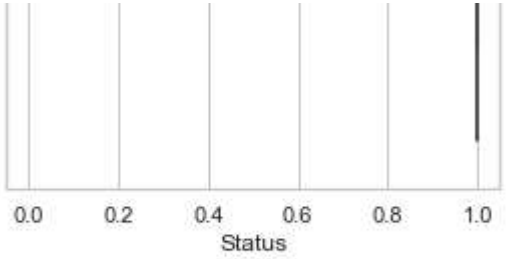

0
100
200
300
Locality

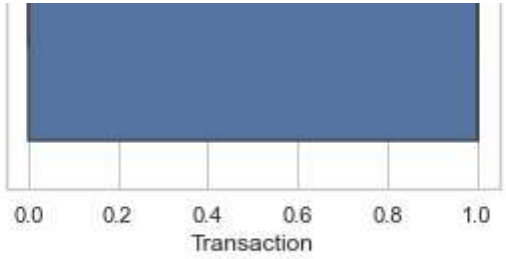

0
20
40
60
80
100
Parking

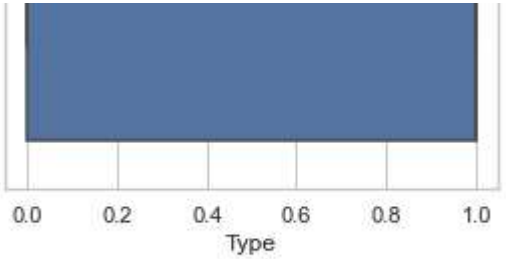


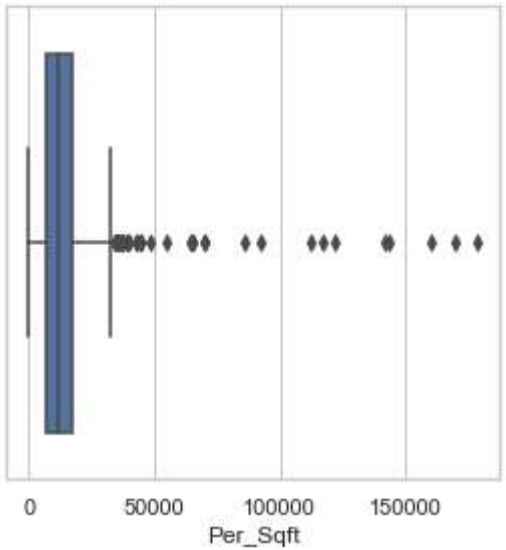


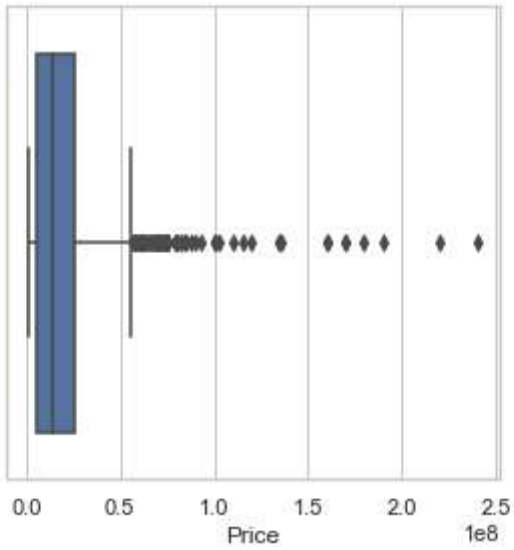








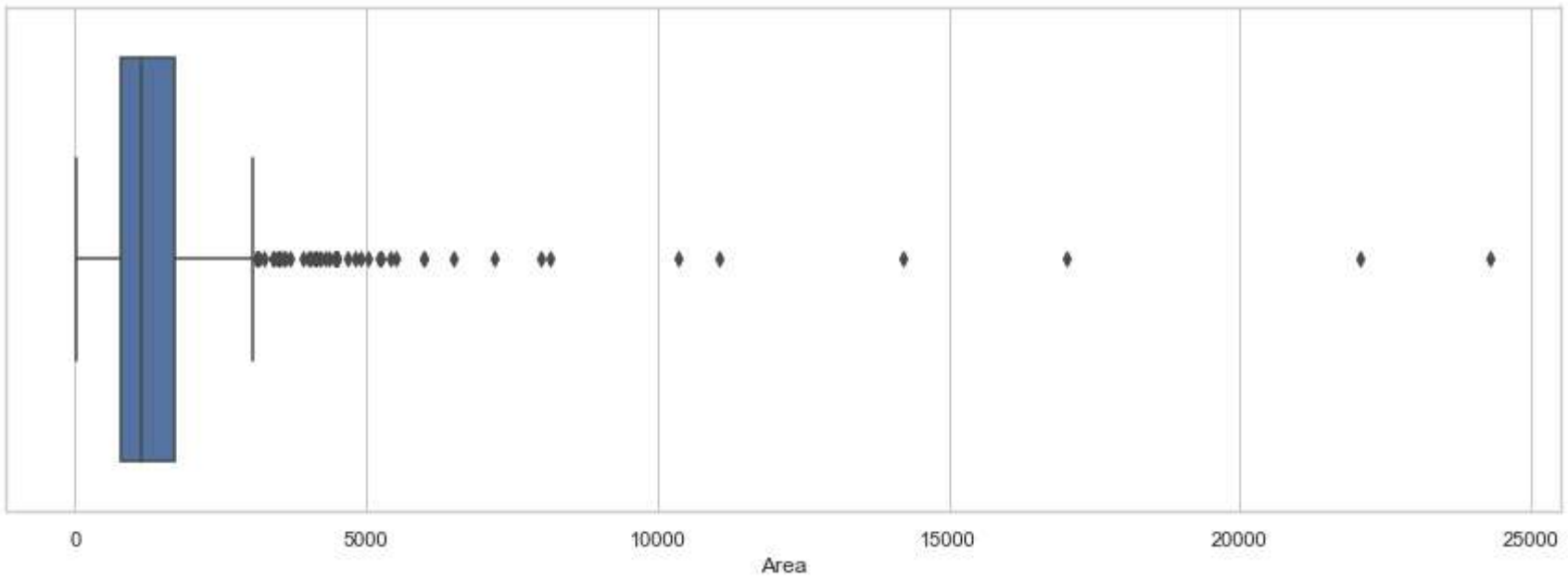




In [73]:
```
df.shape
```

Out[73]:
```
(1122, 11)
```

In [74]:
```
plt.figure(figsize=(15,5))
sns.boxplot(x="Area",data=df)
plt.show()
```



In [75]:
```
q1=df["Area"].quantile(0.25)
q3=df["Area"].quantile(0.75)
iqr=q3-q1
min_range=q1-(1.5*iqr)
max_range=q3+(1.5*iqr)

q1,q3,iqr,min_range,max_range
```

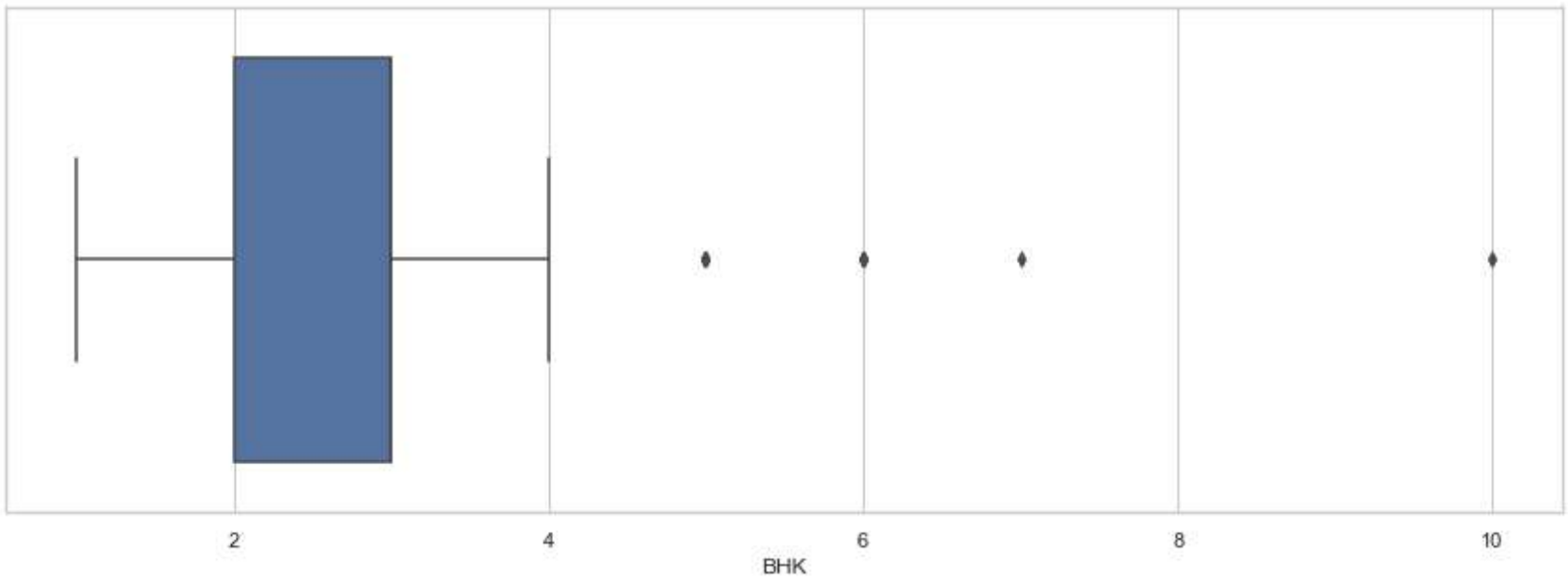
Out[75]:
```
(800.0, 1700.0, 900.0, -550.0, 3050.0)
```

In [76]:
```
df1=df[df["Area"]<=max_range]
```

In [77]:
```
df1.shape
```

Out[77]:
```
(1062, 11)
```

In [78]:
```
plt.figure(figsize=(15,5))
sns.boxplot(x="BHK",data=df1)
plt.show()
```



In [79]:
```
q1=df1["BHK"].quantile(0.25)
q3=df1["BHK"].quantile(0.75)
iqr=q3-q1
min_range=q1-(1.5*iqr)
max_range=q3+(1.5*iqr)

q1,q3,iqr,min_range,max_range
```

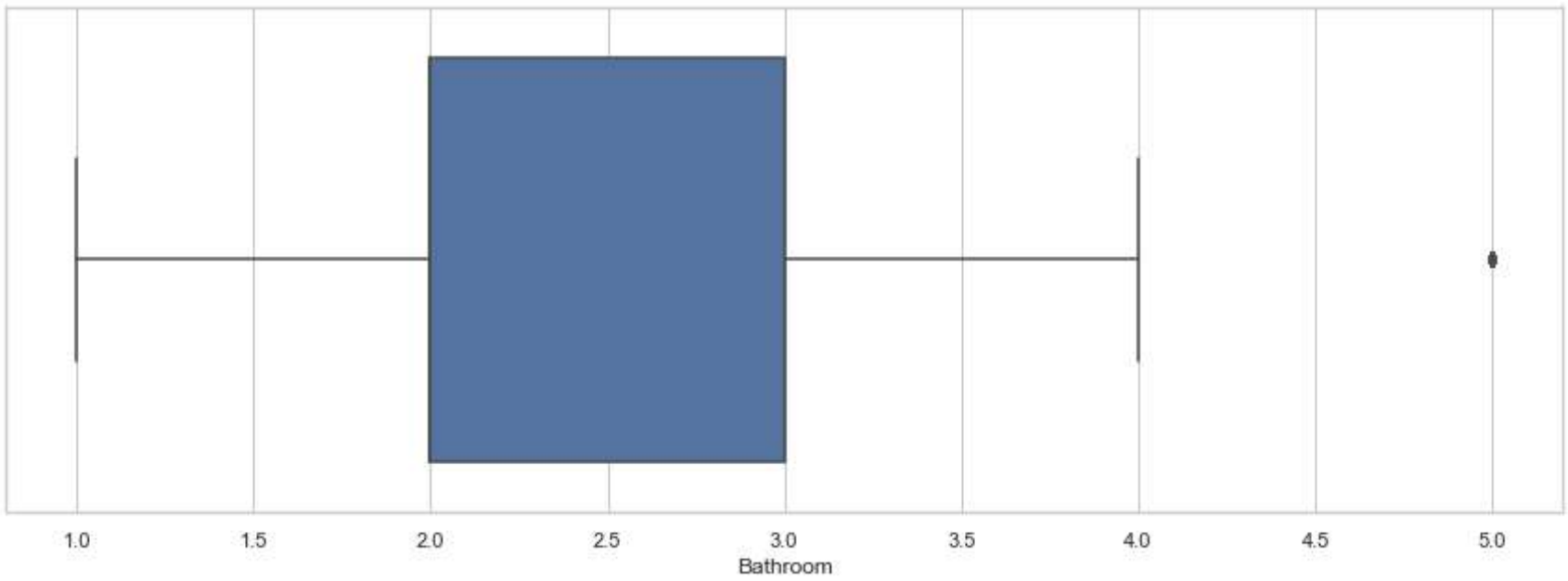
Out[79]:
```
(2.0, 3.0, 1.0, 0.5, 4.5)
```

In [80]:
```
df1=df1[df1["BHK"]<=max_range]
```

In [81]:
```
df1.shape
```

Out[81]:
```
(1050, 11)
```

In [82]:
```
plt.figure(figsize=(15,5))
sns.boxplot(x="Bathroom",data=df1)
plt.show()
```



In [83]:
```
q1=df1["Bathroom"].quantile(0.25)
q3=df1["Bathroom"].quantile(0.75)
iqr=q3-q1
min_range=q1-(1.5*iqr)
max_range=q3+(1.5*iqr)

q1,q3,iqr,min_range,max_range
```

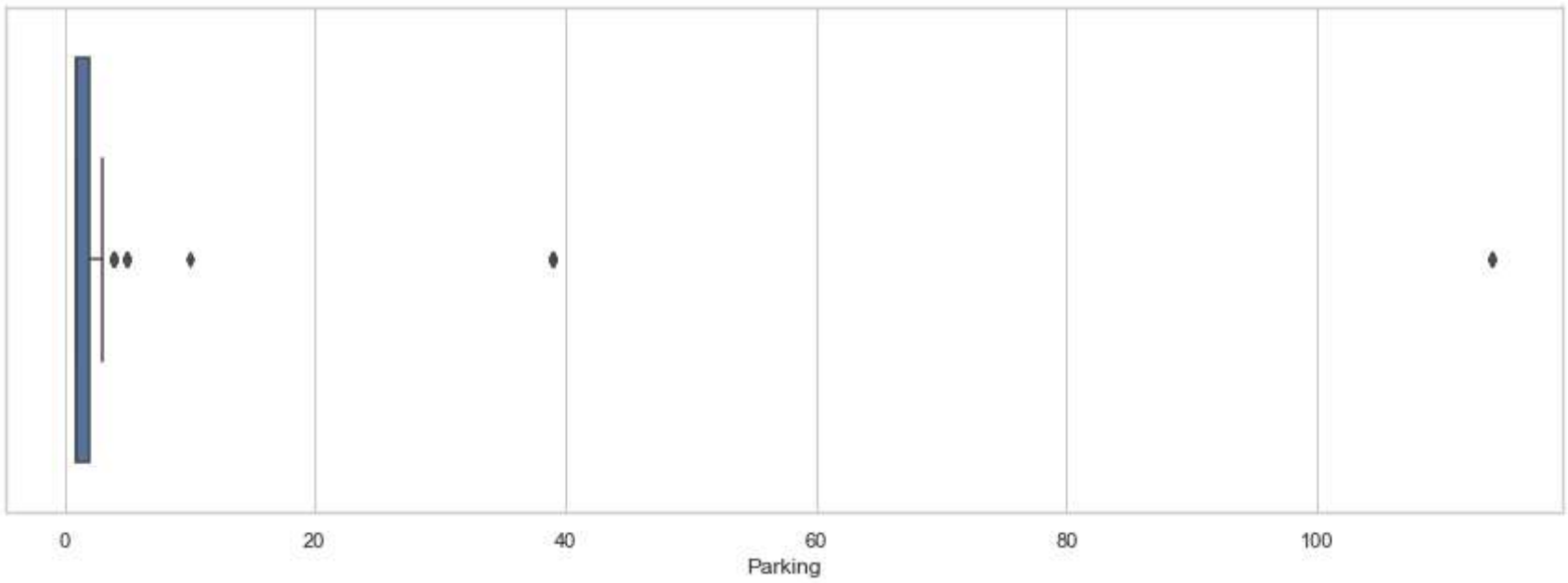
Out[83]:
```
(2.0, 3.0, 1.0, 0.5, 4.5)
```

In [84]:
```
df1=df1[df1["Bathroom"]<=max_range]
```

In [85]:
```
df1.shape
```

Out[85]:
```
(1018, 11)
```

In [86]:
```
plt.figure(figsize=(15,5))
sns.boxplot(x="Parking",data=df1)
plt.show()
```



In [87]:
```
q1=df1["Parking"].quantile(0.25)
q3=df1["Parking"].quantile(0.75)
iqr=q3-q1
min_range=q1-(1.5*iqr)
max_range=q3+(1.5*iqr)

q1,q3,iqr,min_range,max_range
```

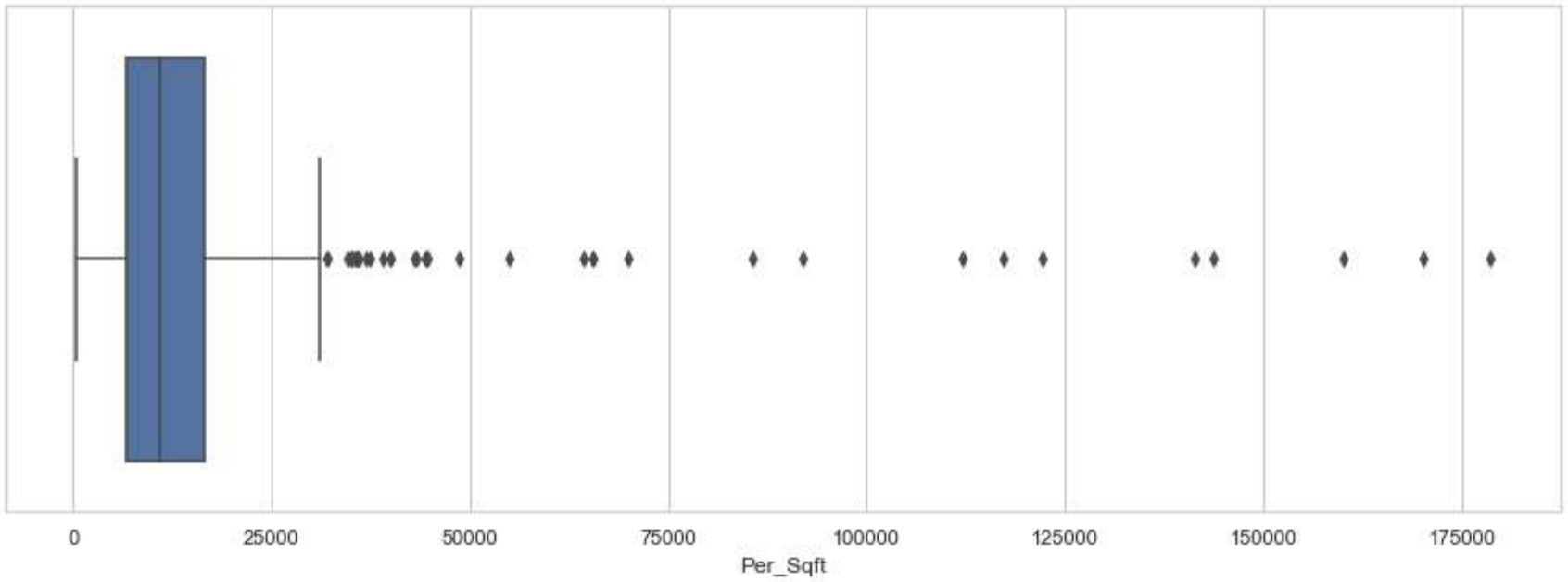
Out[87]:
```
(1.0, 2.0, 1.0, -0.5, 3.5)
```

In [88]:
```
df1=df1[df1["Parking"]<=max_range]
```

In [89]:
```
df1.shape
```

Out[89]:
```
(992, 11)
```

In [90]:
```
plt.figure(figsize=(15,5))
sns.boxplot(x="Per_Sqft",data=df1)
plt.show()
```



In [91]:
```
q1=df1["Per_Sqft"].quantile(0.25)
q3=df1["Per_Sqft"].quantile(0.75)
iqr=q3-q1
min_range=q1-(1.5*iqr)
max_range=q3+(1.5*iqr)

q1,q3,iqr,min_range,max_range
```

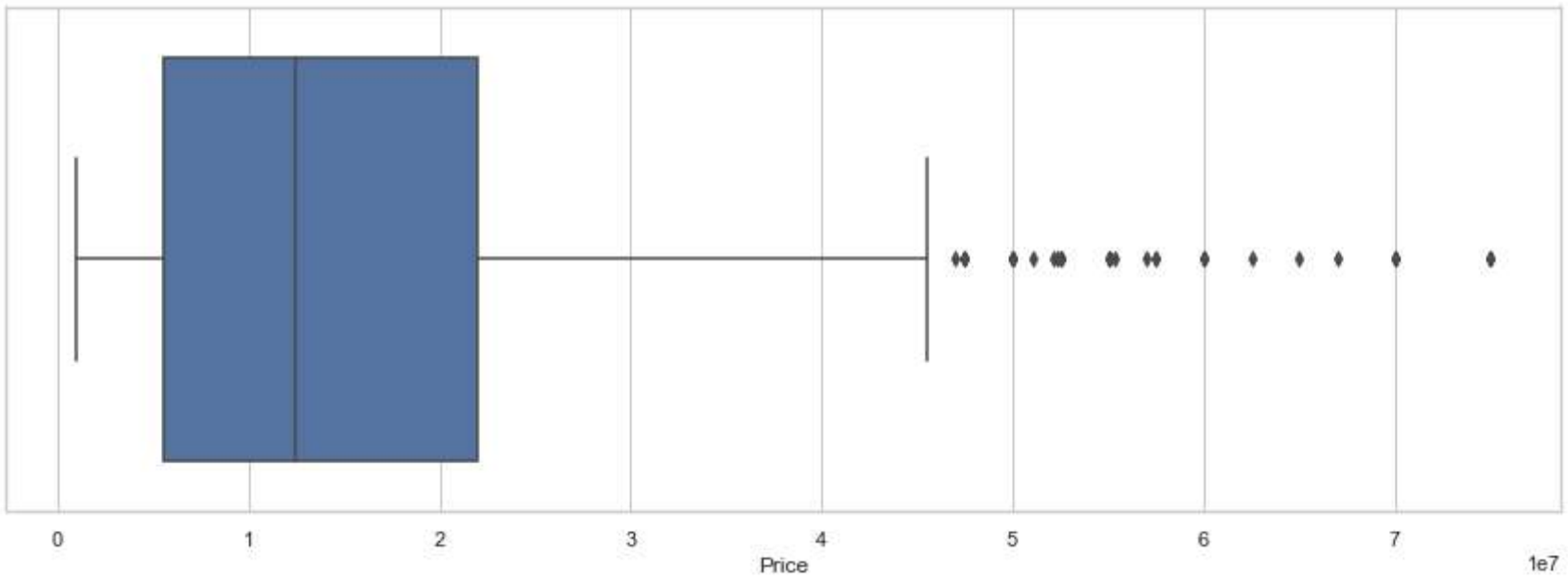
Out[91]:
```
(6666.0, 16666.0, 10000.0, -8334.0, 31666.0)
```

In [92]:
```
df1=df1[df1["Per_Sqft"]<=max_range]
```

In [93]:
```
df1.shape
```

Out[93]:
```
(956, 11)
```

In [94]:
```
plt.figure(figsize=(15,5))
sns.boxplot(x="Price",data=df1)
plt.show()
```



In [95]:
```
q1=df1["Price"].quantile(0.25)
q3=df1["Price"].quantile(0.75)
iqr=q3-q1
min_range=q1-(1.5*iqr)
max_range=q3+(1.5*iqr)

q1,q3,iqr,min_range,max_range
```

Out[95]:
```
(5500000.0, 22000000.0, 16500000.0, -19250000.0, 46750000.0)
```

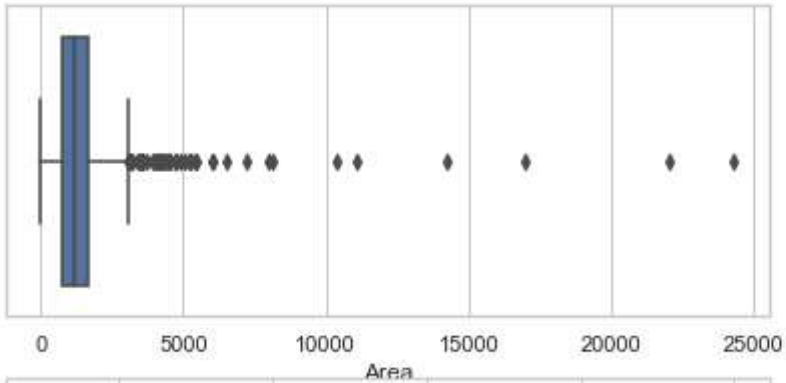
In [96]:
```
df1=df1[df1["Price"]<=max_range]
```

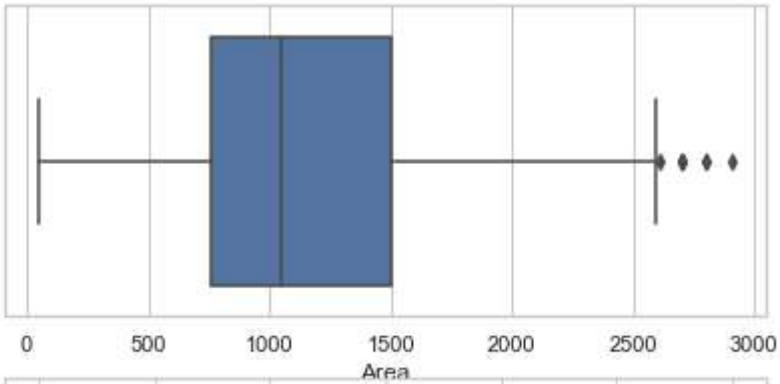
In [97]:
```
df1.shape
```

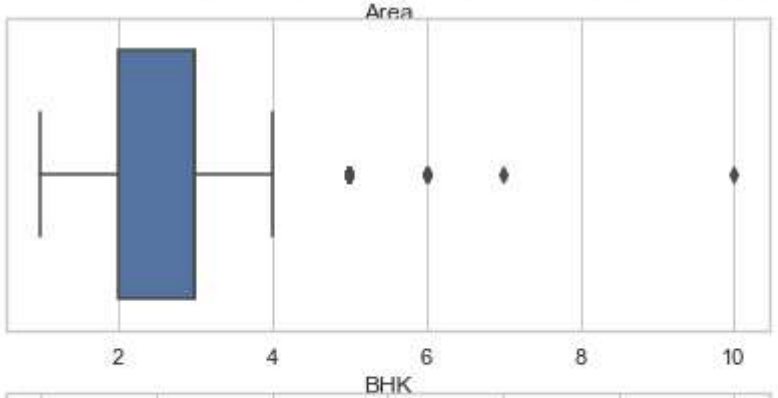
Out[97]:
```
(922, 11)
```

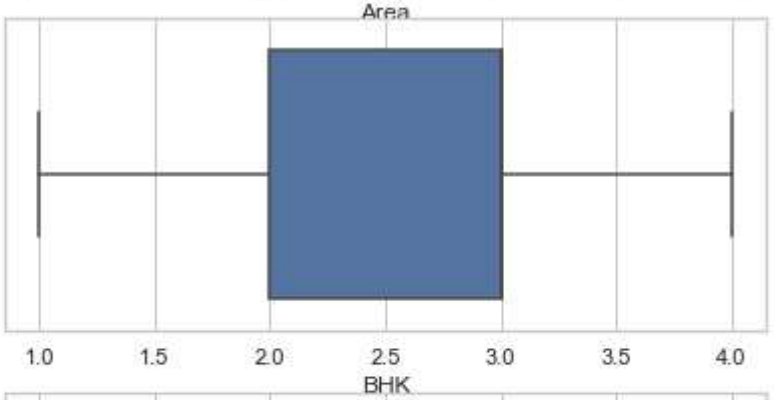
In [98]:
```
plt.figure(figsize=(15,20))
plt.subplot(6,2,1)
sns.boxplot(x=df["Area"])
plt.subplot(6,2,2)
sns.boxplot(x=df1["Area"])
```

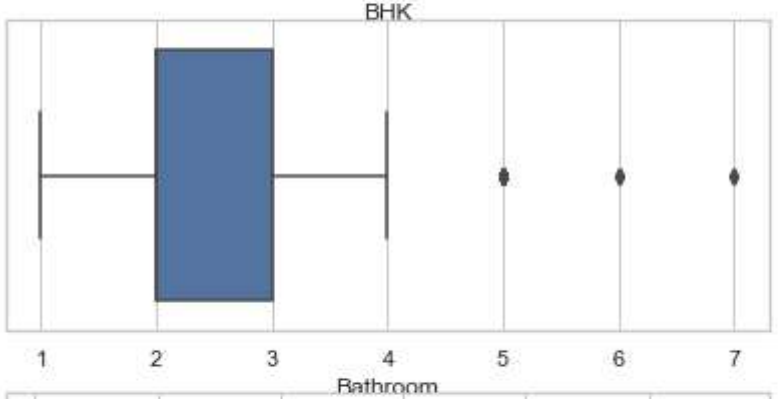
```
plt.subplot(6,2,3)
sns.boxplot(x=df["BHK"])
plt.subplot(6,2,4)
sns.boxplot(x=df1["BHK"])

plt.subplot(6,2,5)
sns.boxplot(x=df["Bathroom"])
plt.subplot(6,2,6)
sns.boxplot(x=df1["Bathroom"])

plt.subplot(6,2,7)
sns.boxplot(x=df["Parking"])
plt.subplot(6,2,8)
sns.boxplot(x=df1["Parking"])

plt.subplot(6,2,9)
sns.boxplot(x=df["Per_Sqft"])
plt.subplot(6,2,10)
sns.boxplot(x=df1["Per_Sqft"])

plt.subplot(6,2,11)
sns.boxplot(x=df["Price"])
plt.subplot(6,2,12)
sns.boxplot(x=df1["Price"])

plt.show()
```

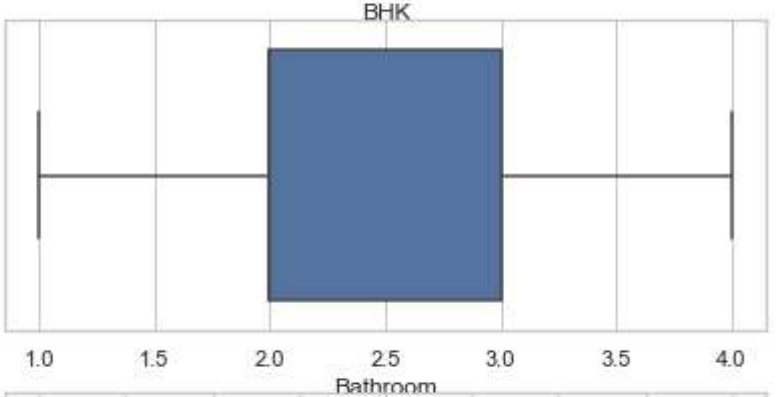

0
5000
10000
15000
20000
25000
Area

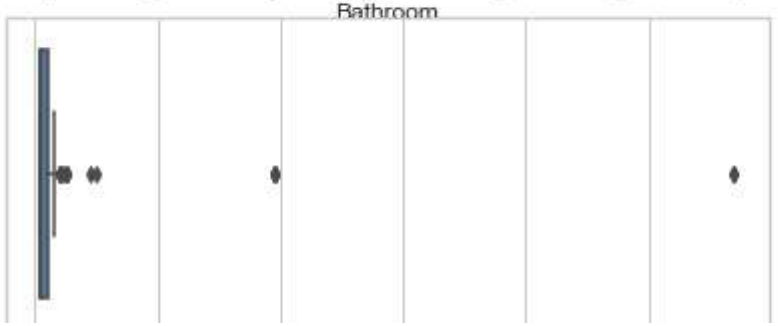

0
500
1000
1500
2000
2500
3000
Area

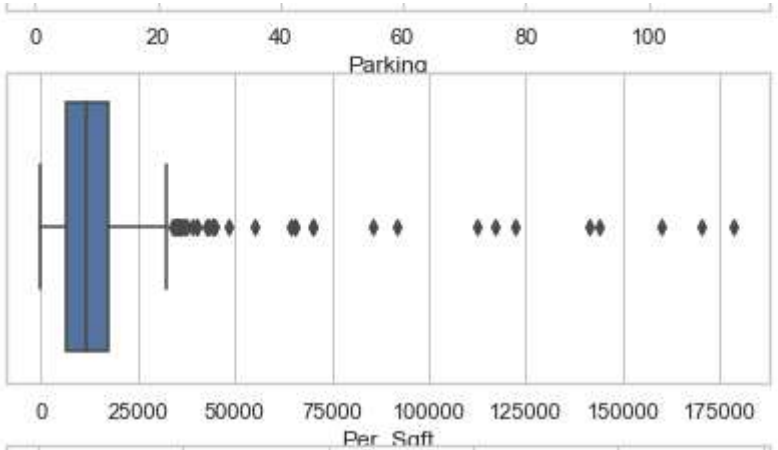

2
4
6
8
10
BHK

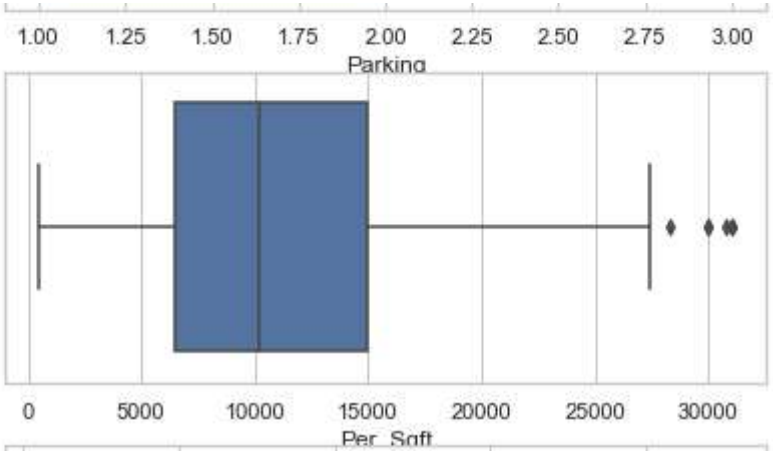

1.0
1.5
2.0
2.5
3.0
3.5
4.0
BHK

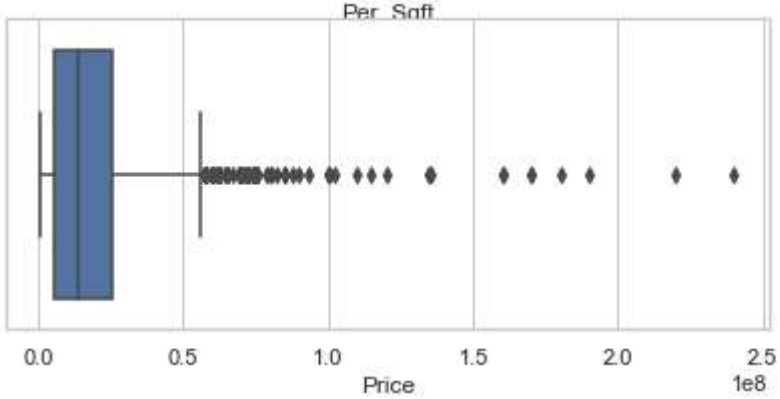

1
2
3
4
5
6
7
Bathroom


1.0
1.5
2.0
2.5
3.0
3.5
4.0
Bathroom









In [99]:
```
plt.figure(figsize=(15,15))
sns.heatmap(data=df1.corr(),annot=True)
plt.show()
```


Area 1 0.73 0.76 -0.066 0.039 0.24 -0.037 0.04 -0.088 0.4 0.79
BHK 0.73 1 0.71 -0.12 0.067 0.13 -0.043 -0.051 0.031 0.3 0.55
Bathroom 0.76 0.71 1 -0.02 0.048 0.2 -0.049 -0.063 0.081 0.44 0.69
Furnishing -0.066 -0.12 -0.02 1 0.019 -0.057 -0.022 -0.023 0.059 -0.0039 -0.011
Locality 0.039 0.067 0.048 0.019 1 -0.045 -0.056 -0.046 0.087 0.051 0.054
Parking 0.24 0.13 0.2 -0.057 -0.045 1 -0.00075 0.076 -0.039 0.29 0.32
Status -0.037 -0.043 -0.049 -0.022 -0.056 -0.00075 1 0.22 0.021 -0.099 -0.085
Transaction 0.04 -0.051 -0.063 -0.023 -0.046 0.076 0.22 1 -0.23 -0.032 -0.017
Type -0.088 0.031 0.081 0.059 0.087 -0.039 0.021 -0.23 1 0.018 -0.014
_Sqft 0.4 0.3 0.44 -0.0039 0.051 0.29 -0.099 -0.032 0.018 1 0.83
1.0
0.8
0.6
0.4
0.2
0.0

In [100…

```
df1.head(3)
```

Out[100]:

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Status	Transaction	Type	Per_Sqft	Price
0	800	3	2	1	274	1	1	0	1	8125	6500000
1	750	2	2	1	135	1	1	0	0	6666	5000000
2	950	2	2	0	49	1	1	1	0	16315	15500000

Exploratory Data Analysis

- **BHK & Bathroom**: 40–50% of apartments are 3BHK with 2 bathrooms.
- **Furnishing Status**: 57% are semi-furnished, 15% fully furnished, and 28% unfurnished.
- **Parking**: Around 65% of apartments have only single parking.
- **Property Status**: Nearly 95% of properties are ready to move.
- **Transaction Type**: Only 37% are new properties; others are resales.
- **Property Type**: 55% are full-building floors, the rest are standard apartments.
- **Price Extremes**:
  Highest price: ₹24 crores
  Lowest price: ₹10 lakhs
- **Price per Sqft Extremes**:
  Highest rate: ₹1.7 lakh/sqft
  Lowest rate: ₹204/sqft
- **Data Cleaning Performed**:
  ~20% null values handled (filled or removed).
  Removed 103 duplicate rows.
  Outliers (~20%) identified and removed.

Train The Data

In [101…
```
x=df1.iloc[:,:-1]
y=df1["Price"]
```

In [102…
```
x.head(3)
```

Out[102]:

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Status	Transaction	Type	Per_Sqft
0	800	3	2	1	274	1	1	0	1	8125
1	750	2	2	1	135	1	1	0	0	6666
2	950	2	2	0	49	1	1	1	0	16315

In [103…
```
x.tail(3)
```

Out[103]:

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Status	Transaction	Type	Per_Sqft
1255	1050	3	2	1	44	3	1	1	1	11904
1256	875	3	3	1	44	3	1	0	1	20000
1257	990	2	2	2	45	1	1	1	1	11616

In [104…
```
y.head(3)
```

Out[104]:
```
0     6500000
1     5000000
2    15500000
Name: Price, dtype: int64
```

In [105…
```
y.tail(3)
```

Out[105]:
```
1255    12500000
1256    17500000
1257    11500000
Name: Price, dtype: int64
```

In [106…
```
from sklearn.model_selection import train_test_split
```

In [107…
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

Apply Machine Learning Regression Models

In [108…
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
```

Linear Regression

In [109…
```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[109]:
```
LinearRegression()
```

In [112…
```
lr.score(x_test,y_test)*100
```

Out[112]:
```
93.23856692159778
```

In [113…
```
lr.score(x_train,y_train)*100
```

Out[113]:
```
94.60702131281151
```

Decision Tree

In [114…
```
dt=DecisionTreeRegressor(max_depth=11,random_state=42)
```

In [115…
```
dt.fit(x_train,y_train)
```

Out[115]:
```
DecisionTreeRegressor(max_depth=11, random_state=42)
```

In [118…
```
dt.score(x_test,y_test)*100
```

Out[118]:
```
97.03762456019948
```

In [119…
```
dt.score(x_train,y_train)*100
```

Out[119]:
```
99.9962278577032
```

K-Neighbors

In [120…
```
kn=KNeighborsRegressor(n_neighbors=3)
```

In [121…
```
kn.fit(x_train,y_train)
```

Out[121]:
```
KNeighborsRegressor(n_neighbors=3)
```

In [122…
```
kn.score(x_train,y_train)*100
```

Out[122]:
```
97.91226797561005
```

In [124…
```
kn.score(x_test,y_test)*100
```

Out[124]:
```
95.92295693514855
```

Random Forest

In [125…
```
rf=RandomForestRegressor(n_estimators=17,random_state=42)
```

In [126…
```
rf.fit(x_train,y_train)
```

Out[126]:
```
RandomForestRegressor(n_estimators=17, random_state=42)
```

In [129…
```
rf.score(x_test,y_test)*100
```

Out[129]:
```
99.44730303368287
```

In [130…
```
rf.score(x_train,y_train)*100
```

Out[130]:
```
99.82664817131528
```

Check the models Accuracy using example

In [131…
```
#800    3       2       1       274     1       1       0       1       8125            #6500000
```

In [132…
```
lr.predict([[800,3,2,1,274,1,1,0,1,8125]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegr
ession was fitted with feature names
  warnings.warn(
```

Out[132]:
```
array([6286786.6217701])
```

In [133…
```
dt.predict([[800,3,2,1,274,1,1,0,1,8125]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTr
eeRegressor was fitted with feature names
  warnings.warn(
```

Out[133]:
```
array([6500000.])
```

In [134…
```
kn.predict([[800,3,2,1,274,1,1,0,1,8125]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but KNeighbors
Regressor was fitted with feature names
  warnings.warn(
```

Out[134]:
```
array([7033333.33333333])
```

In [136…
```
rf.predict([[800,3,2,1,274,1,1,0,1,8125]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

Out[136]:
```
array([6411764.70588235])
```

In [137…
```
# 750   2       2       1       135     1       1       0       0       6666                #5000000
```

In [138…
```
lr.predict([[750,2,2,1,135,1,1,0,0,6666]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

Out[138]:
```
array([4570894.96701807])
```

In [139…
```
dt.predict([[750,2,2,1,135,1,1,0,0,6666]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeRegressor was fitted with feature names
  warnings.warn(
```

Out[139]:
```
array([5000000.])
```

In [140…
```
kn.predict([[750,2,2,1,135,1,1,0,0,6666]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but KNeighborsRegressor was fitted with feature names
  warnings.warn(
```

Out[140]:
```
array([5033333.33333333])
```

In [141…
```
rf.predict([[750,2,2,1,135,1,1,0,0,6666]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

Out[141]:
```
array([5076470.58823529])
```

In [142…
```
#1050   3       2       1       44      3       1       1       1       11904             12500000
```

In [143…
```
lr.predict([[1050,3,2,1,44,3,1,1,1,11904]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegr
ession was fitted with feature names
  warnings.warn(
```

Out[143]:
```
array([14136524.05779947])
```

In [144…
```
dt.predict([[1050,3,2,1,44,3,1,1,1,11904]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTr
eeRegressor was fitted with feature names
  warnings.warn(
```

Out[144]:
```
array([12700000.])
```

In [145…
```
kn.predict([[1050,3,2,1,44,3,1,1,1,11904]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but KNeighbors
Regressor was fitted with feature names
  warnings.warn(
```

Out[145]:
```
array([13133333.33333333])
```

In [146…
```
rf.predict([[1050,3,2,1,44,3,1,1,1,11904]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomFore
stRegressor was fitted with feature names
  warnings.warn(
```

Out[146]:
```
array([12405882.35294118])
```

In [147…
```
# 875   3       3       1       44      3       1       0       1       20000       #17500000
```

In [148…
```
lr.predict([[875,3,3,1,44,3,1,0,1,20000]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegr
ession was fitted with feature names
  warnings.warn(
```

Out[148]:
```
array([21643616.84064677])
```

In [149…
```
dt.predict([[875,3,3,1,44,3,1,0,1,20000]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTr
eeRegressor was fitted with feature names
  warnings.warn(
```

Out[149]:
```
array([17500000.])
```

In [150…
```
kn.predict([[875,3,3,1,44,3,1,0,1,20000]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but KNeighbors
Regressor was fitted with feature names
  warnings.warn(
```

Out[150]:
```
array([17233333.33333333])
```

In [151…
```
rf.predict([[875,3,3,1,44,3,1,0,1,20000]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomFore
stRegressor was fitted with feature names
  warnings.warn(
```

Out[151]:
```
array([16958823.52941176])
```

In [152…
```
# 990   2       2       2       45      1       1       1       1       11616        11500000
```

In [157…
```
lr.predict([[990,2,2,2,45,1,1,1,1,11616]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegr
ession was fitted with feature names
  warnings.warn(
```

Out[157]:
```
array([13295512.75695247])
```

In [154…
```
dt.predict([[990,2,2,2,45,1,1,1,1,11616]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTr
eeRegressor was fitted with feature names
  warnings.warn(
```

Out[154]:
```
array([11500000.])
```

In [155…
```
kn.predict([[990,2,2,2,45,1,1,1,1,11616]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but KNeighbors
Regressor was fitted with feature names
  warnings.warn(
```

Out[155]:
```
array([10833333.33333333])
```

In [156…
```
rf.predict([[990,2,2,2,45,1,1,1,1,11616]])
```

```
C:\Users\shiva\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomFore
stRegressor was fitted with feature names
  warnings.warn(
```

Out[156]:
```
array([11452941.17647059])
```

Check The Error

In [158…
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [159…
```
mean_squared_error(y_test,lr.predict(x_test)),mean_absolute_error(y_test,lr.predict(x_test))
```

Out[159]:
```
(5989775141346.587, 1852894.6412756655)
```

In [160…
```
mean_squared_error(y_test,dt.predict(x_test)),mean_absolute_error(y_test,dt.predict(x_test))
```

Out[160]:
```
(2624290230030.0303, 884938.7387387387)
```

In [161…
```
mean_squared_error(y_test,rf.predict(x_test)),mean_absolute_error(y_test,rf.predict(x_test))
```

Out[161]:
```
(489619657720.0039, 418779.0143084262)
```

In [162…
```
mean_squared_error(y_test,kn.predict(x_test)),mean_absolute_error(y_test,kn.predict(x_test))
```

Out[162]:
```
(3611744864864.8647, 948324.3243243245)
```

ML Regression Models

- **1. Linear Regression**
  Accuracy: **93.2%**
  Baseline model with linear assumption
  Moderate performance, struggles with non-linearity
- **2. Decision Tree Regressor**
  Accuracy: **97%**
  Good for non-linear & categorical data
  Fast, interpretable, may overfit
- **3. KNN Regressor**
  Accuracy: **95.9%**
  Predicts based on nearest neighbors
  Simple but slower and sensitive to scaling
- **4. Random Forest Regressor**
  Accuracy: **99.45%**
  Best performance, low error
  Robust, avoids overfitting, handles complex data

Final Conclusion

Both **Decision Tree** and **Random Forest** performed well on different examples.

After comparing **MSE and MAE**, **Random Forest** had the **lowest error**, indicating **more stable and reliable predictions**.

✓ **Random Forest** is concluded as the **best regression model** for this **Housing Price Prediction** problem.

In [ ]: