Bengaluru House Price Prediction

Author: Sami Ullah

Import Libraries

```
!pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.7-cp311-cp311-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.11/dist-packages (from catboost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from catboost) (1.13.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (3.2.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly->catboost) (9.0.0)
Downloading catboost-1.2.7-cp311-cp311-manylinux2014_x86_64.whl (98.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 7.2 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.7
```

```
import re

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

%matplotlib inline
import missingno as msng
sns.set()


from sklearn.preprocessing import OneHotEncoder , OrdinalEncoder, StandardScaler
from sklearn.model_selection import train_test_split, ShuffleSplit, cross_val_score, GridSearchCV, RandomizedSearchCV

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error, root_mean_squared_error

from sklearn.linear_model import LinearRegression, Lasso , Ridge
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor



import warnings
warnings.filterwarnings('ignore')

import scipy.stats as stats
import pylab
```

Load dataset

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable

```
data_set = pd.read_csv("Bengaluru_House_Data.csv")
data_set. head()
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00

About Data

📌 **Area_type** - Description of the area

📌 **Availability** - When it can be possessed or when it is ready

📌 **Location** - Where it is located in Bengaluru

📌 **Size** - BHK or Bedrooms

📌 **Society** - To which society it belongs

📌 **Total_sqft** - Size of the property in sq.ft

📌 **Bath** - No. of Bathrooms

📌 **Balcony** - No. of the Balcony

📌 **Price** - Value of the property in lakhs (Indian Rupee - ₹)

Identify and prioritize significant features

Show first 5 Rows

```
data = data_set[["location", "size" , "total_sqft", "bath", "price"]]
data.head(5)
```

	location	size	total_sqft	bath	price
0	Electronic City Phase II	2 BHK	1056	2.0	39.07
1	Chikka Tirupathi	4 Bedroom	2600	5.0	120.00
2	Uttarahalli	3 BHK	1440	2.0	62.00
3	Lingadheeranahalli	3 BHK	1521	3.0	95.00
4	Kothanur	2 BHK	1200	2.0	51.00

Size of Dataset

```
data.shape
```

```
(13320, 5)
```

Show 5 Samples

```
data.sample(5)
```

	location	size	total_sqft	bath	price
2000	Bellandur	4 BHK	2025	4.0	109.00
5850	Haralur Road	2 BHK	1027	2.0	44.00
6744	Rayasandra	3 BHK	1458	3.0	60.00
11372	Brindavan Layout	2 BHK	1100	2.0	38.68
2041	Ramamurthy Nagar	1 BHK	360	1.0	26.00

Retrieve detailed information about the dataset's features using data.info().

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   location    13319 non-null  object 
 1   size        13304 non-null  object 
 2   total_sqft  13320 non-null  object 
 3   bath        13247 non-null  float64
 4   price       13320 non-null  float64
dtypes: float64(2), object(3)
memory usage: 520.4+ KB
```

Check for missing values and identify duplicate entries in the dataset.

```
print("Total Missing Rows --->>>  " , data.isnull().sum().sum())
```

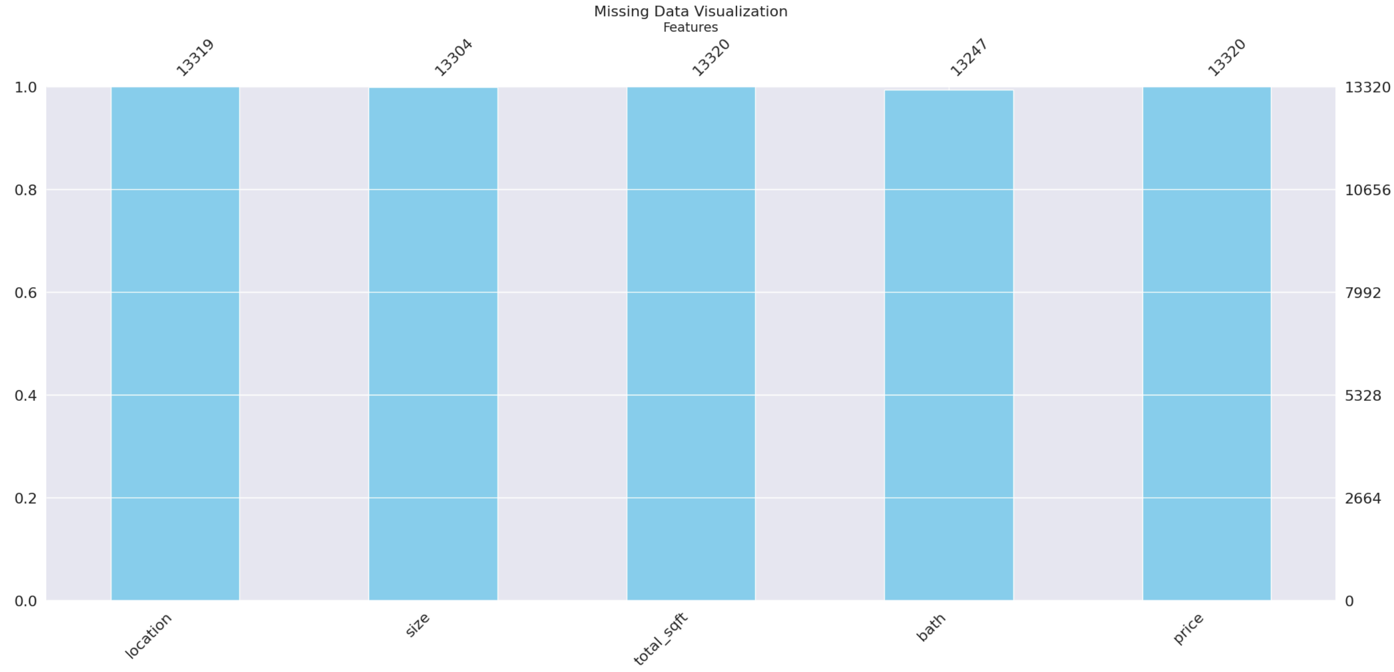
```
Total Missing Rows --->>>   90
```

```
data.isnull().sum()
```

	0
location	1
size	16
total_sqft	0
bath	73
price	0

dtype: int64

```
# Create the figure with the desired size
plt.figure(figsize=(12, 5))

# Plot the bar chart using 'msng' (assumed to be a variable holding the data)
msng.bar(data, color='skyblue')

# Set the title of the plot
plt.title('Missing Data Visualization', fontsize=16)

# Label the x-axis and y-axis
plt.xlabel('Features', fontsize=14)
plt.ylabel('Number of Missing Values', fontsize=14)

# Rotate the x-axis labels for better readability if needed
plt.xticks(rotation=45)

# Show the plot
plt.show()
```



Remove rows with missing values as the dataset contains only a small number of such entries.

```
data.dropna(inplace=True)
```

```
data.duplicated().sum()
```

```
881
```

```
# Drop Duplicated Values
data.drop_duplicates(inplace=True)
```

Display the shape of the dataset after removing missing & duplicate entries.

```
data.shape
```

```
(12365, 5)
```

Perform a statistical analysis of the dataset using data.describe() to summarize key metrics

```
data.describe()
```

	bath	price
count	12365.000000	12365.000000
mean	2.719693	115.229230
std	1.369955	153.201909
min	1.000000	8.000000
25%	2.000000	50.000000
50%	2.000000	73.870000
75%	3.000000	123.000000
max	40.000000	3600.000000

Feature Egineering

- Perform feature engineering to create, transform, or optimize features for improving the model's performance

Size

```
data["size"].nunique()
```

```
31
```

```
data["size"].unique()
```

```
array(['2 BHK', '4 Bedroom', '3 BHK', '4 BHK', '6 Bedroom', '3 Bedroom',
       '1 BHK', '1 RK', '1 Bedroom', '8 Bedroom', '2 Bedroom',
       '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',
       '9 BHK', '9 Bedroom', '27 BHK', '10 Bedroom', '11 Bedroom',
       '10 BHK', '19 BHK', '16 BHK', '43 Bedroom', '14 BHK', '8 BHK',
       '12 Bedroom', '13 BHK', '18 Bedroom'], dtype=object)
```

Extracts the number of bedrooms (BHK) from the size column

Drop the size column

```
data["BHK"] = data["size"].str.split(" ").str[0].astype('int')
# Drop Size Column
data.drop("size" , axis=1 ,inplace=True)
```

```
data.head(2)
```

	location	total_sqft	bath	price	BHK
0	Electronic City Phase II	1056	2.0	39.07	2
1	Chikka Tirupathi	2600	5.0	120.00	4

bath

```
data["bath"].nunique()
```

```
19
```

```
data["bath"].unique()
```

```
array([ 2.,  5.,  3.,  4.,  6.,  1.,  9.,  8.,  7., 11., 10., 14., 27.,
       12., 16., 40., 15., 13., 18.])
```

```
data["bath"].value_counts()
```

bath	count
2.0	6322
3.0	3092
4.0	1183
1.0	741
5.0	516
6.0	270
7.0	102
8.0	64
9.0	42
10.0	13
12.0	7
13.0	3
11.0	3
16.0	2
27.0	1
40.0	1
15.0	1
14.0	1
18.0	1

dtype: int64

Convert Into int

```
data["bath"] = data["bath"].astype('int')
```

total_sqft

```
data["total_sqft"].nunique()
```

```
2067
```

```
data["total_sqft"].unique()
```

```
array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)
```

```
def to_float(x):
  try:
    float(x)
  except:
    return False
  return True
```

```
data[~data["total_sqft"].apply(to_float)]
```

	location	total_sqft	bath	price	BHK
30	Yelahanka	2100 - 2850	4	186.000	4
122	Hebbal	3067 - 8156	4	477.000	4
137	8th Phase JP Nagar	1042 - 1105	2	54.005	2
165	Sarjapur	1145 - 1340	2	43.490	2
188	KR Puram	1015 - 1540	2	56.800	2
...	...	...	...	...	...
12955	Thanisandra	1437 - 1629	3	75.885	3
12975	Whitefield	850 - 1060	2	38.190	2
12990	Talaghattapura	1804 - 2273	3	122.000	3
13059	Harlur	1200 - 1470	2	72.760	2
13265	Hoodi	1133 - 1384	2	59.135	2

189 rows × 5 columns

Convert Units for Total_sqft

- 1437 - 1629 = (1439+1629)/2
- Sq. Meter: 1 Sq. Meter = 10.7639 Sq.ft
- sq yd: 1 sq yd = 9 Sq. ft
- Perch: Perch = 272.25 Sq. ft
- Grounds: 1 Ground = 2400 Sq. Ft.
- Acres: 1 Acre = 43,560 Sq. Ft.
- Cents: 1 Cent = 435.6 Sq. Ft.
- Guntha: 1 Guntha = 1,089 Sq. Ft.

```
def to_sqft(x):
    # Handle ranges (e.g., "4000 - 5249")
    if '-' in x:
        token = x.split('-')
        if len(token) == 2:
            return (float(token[0].strip()) + float(token[1].strip())) / 2

    # Handle Sq. Yards to Sq. Feet
    if "Sq. Yards" in x:
        return float(x.split("Sq. Yards")[0].strip()) * 9

    # Handle Sq. Meter to Sq. Feet
    if "Sq. Meter" in x:
        return float(x.split("Sq. Meter")[0].strip()) * 10.7639

    # Handle Perch to Sq. Feet
    if "Perch" in x:
        return float(x.split("Perch")[0].strip()) * 272.25

    # Handle Grounds to Sq. Feet
    if "Grounds" in x:
        return float(x.split("Grounds")[0].strip()) * 2400

    # Handle Acres to Sq. Feet
    if "Acres" in x:
        return float(x.split("Acres")[0].strip()) * 43560

    # Handle Cents to Sq. Feet
    if "Cents" in x:
        return float(x.split("Cents")[0].strip()) * 435.6

    # Handle Guntha to Sq. Feet
    if "Guntha" in x:
        return float(x.split("Guntha")[0].strip()) * 1089

    # Handle standalone numeric values
    try:
        return float(x)
    except ValueError:
```

```
    return None  # For invalid or unknown formats
```

```
to_sqft('12 - 6')
```

```
9.0
```

```
data["total_sqft"] = data["total_sqft"].apply(to_sqft)
```

location

```
data["location"].nunique()
```

```
1304
```

```
data["location"].unique()
```

```
array(['Electronic City Phase II', 'Chikka Tirupathi', 'Uttarahalli', ...,
       '12th cross srinivas nagar banshankari 3rd stage',
       'Havanur extension', 'Abshot Layout'], dtype=object)
```

```
data["location"] = data["location"].apply(lambda x:x.lower())
data["location"].nunique()
```

```
1294
```

- First, clean up any extra spaces in the location column.
- Then, count the occurrences of each location.

To perform the dimensionality reduction by tagging locations with fewer than 10 data points as "other":

- Filter the locations that have 10 or fewer data points.
- Replace the locations with fewer than 10 data points with the label "other".

```
data["location"] = data["location"].apply(lambda x:x.strip())
loaction_states = data.groupby('location')['location'].agg('count').sort_values(ascending=False)
loaction_states
```

	location
location	
whitefield	502
sarjapur road	357
electronic city	275
thanisandra	225
kanakpura road	217
...	...
kamanahalli main road	1
kamdhenu nagar	1
1 giri nagar	1
kanakadasa layout	1
zuzuvadi	1

1282 rows × 1 columns

dtype: int64

```
loaction_states_less_then_10 = loaction_states[loaction_states<=10]
loaction_states_less_then_10
```

location	location
pattandur agrahara	10
gunjur palya	10
naganathapura	10
ganga nagar	10
dodsworth layout	10
...	...
kamanahalli main road	1
kamdhenu nagar	1
1 giri nagar	1
kanakadasa layout	1
zuzuvadi	1

1051 rows × 1 columns

dtype: int64

```
data["location"] = data["location"].apply(lambda x: 'other' if x in loaction_states_less_then_10 else x)
```

```
data["location"].nunique()
```

```
232
```

```
data["location"].unique()
```

```
array(['electronic city phase ii', 'chikka tirupathi', 'uttarahalli',
       'lingadheeranahalli', 'kothanur', 'whitefield', 'old airport road',
       'rajaji nagar', 'marathahalli', 'other', '7th phase jp nagar',
       'gottigere', 'sarjapur', 'mysore road', 'bisuvanahalli',
       'raja rajeshwari nagar', 'kengeri', 'binny pete', 'thanisandra',
       'bellandur', 'electronic city', 'ramagondanahalli', 'yelahanka',
       'hebbal', 'kasturi nagar', 'kanakpura road',
       'electronics city phase 1', 'kundalahalli', 'chikkalasandra',
       'murugeshpalya', 'sarjapur  road', 'hsr layout', 'doddathoguru',
       'kr puram', 'bhoganhalli', 'lakshminarayana pura', 'begur road',
       'varthur', 'bommanahalli', 'gunjur', 'devarachikkanahalli',
       'hegde nagar', 'haralur road', 'hennur road', 'kothannur',
       'kalena agrahara', 'kaval byrasandra', 'isro layout',
       'garudachar palya', 'epip zone', 'dasanapura', 'kasavanhalli',
       'sanjay nagar', 'domlur', 'sarjapura - attibele road',
       'yeshwanthpur', 'chandapura', 'nagarbhavi', 'devanahalli',
       'ramamurthy nagar', 'malleshwaram', 'akshaya nagar', 'shampura',
       'kadugodi', 'lb shastri nagar', 'hormavu', 'vishwapriya layout',
       'kudlu gate', '8th phase jp nagar', 'bommasandra industrial area',
       'anandapura', 'vishveshwarya layout', 'kengeri satellite town',
       'kannamangala', 'hulimavu', 'mahalakshmi layout', 'hosa road',
       'attibele', 'cv raman nagar', 'kumaraswami layout', 'nagavara',
       'hebbal kempapura', 'vijayanagar', 'nagasandra', 'kogilu',
       'panathur', 'padmanabhanagar', '1st block jayanagar',
       'kammasandra', 'dasarahalli', 'magadi road', 'koramangala',
       'dommasandra', 'budigere', 'kalyan nagar', 'ombr layout',
       'horamavu agara', 'ambedkar nagar', 'talaghattapura', 'balagere',
       'jigani', 'gollarapalya hosahalli', 'old madras road',
       'kaggadasapura', '9th phase jp nagar', 'jakkur', 'tc palaya',
       'giri nagar', 'singasandra', 'aecs layout', 'mallasandra', 'begur',
       'jp nagar', 'malleshpalya', 'munnekollal', 'kaggalipura',
       '6th phase jp nagar', 'ulsoor', 'thigalarapalya',
       'somasundara palya', 'basaveshwara nagar', 'bommasandra',
       'ardendale', 'harlur', 'kodihalli', 'bannerghatta road', 'hennur',
       '5th phase jp nagar', 'kodigehaali', 'billekahalli', 'jalahalli',
       'mahadevpura', 'anekal', 'sompura', 'dodda nekkundi', 'hosur road',
       'battarahalli', 'sultan palaya', 'ambalipura', 'hoodi',
       'brookefield', 'yelenahalli', 'vittasandra',
       '2nd stage nagarbhavi', 'vidyaranyapura', 'amruthahalli',
       'kodigehalli', 'subramanyapura', 'basavangudi', 'kenchenahalli',
       'banjara layout', 'kereguddadahalli', 'kambipura',
       'banashankari stage iii', 'sector 7 hsr layout', 'rajiv nagar',
       'arekere', 'mico layout', 'kammanahalli', 'banashankari',
       'chikkabanavar', 'hrbr layout', 'nehru nagar', 'kanakapura',
       'konanakunte', 'margondanahalli', 'r.t. nagar', 'tumkur road',
```

```
'gm palaya', 'jalahalli east', 'hosakerehalli', 'indira nagar',
'kodichikkanahalli', 'varthur road', 'anjanapura', 'abbigere',
'tindlu', 'gubbalala', 'cunningham road', 'kudlu',
'banashankari stage vi', 'cox town', 'kathriguppe', 'hbr layout',
'yelahanka new town', 'sahakara nagar', 'rachenahalli',
'yelachenahalli', 'green glen layout', 'thubarahalli',
'horamavu banaswadi', '1st phase jp nagar', 'ngr layout',
'seegehalli', 'nri layout', 'babusapalaya', 'iblur village',
'ananth nagar', 'channasandra', 'choodasandra', 'kaikondrahalli',
'neeladri nagar', 'frazer town', 'cooke town', 'doddakallasandra',
'chamrajpet', 'rayasandra', '5th block hbr layout', 'pai layout',
'banashankari stage v', 'sonnenahalli', 'benson town',
'judicial layout', 'banashankari stage ii', 'karuna nagar',
```

Creating a New Feature: price_per_sqft for Outlier Removal

To help with the identification and removal of outliers, we have introduced a new feature, price_per_sqft, which is calculated by multiplying the price by 100,000 and dividing it by the total square footage (total_sqft).

```
data["price_per_sqft"] = (data["price"]*100000)/data["total_sqft"]
data.head(2)
```

	location	total_sqft	bath	price	BHK	price_per_sqft
0	electronic city phase ii	1056.0	2	39.07	2	3699.810606
1	chikka tirupathi	2600.0	5	120.00	4	4615.384615

Setting Color Palette for Visualizations

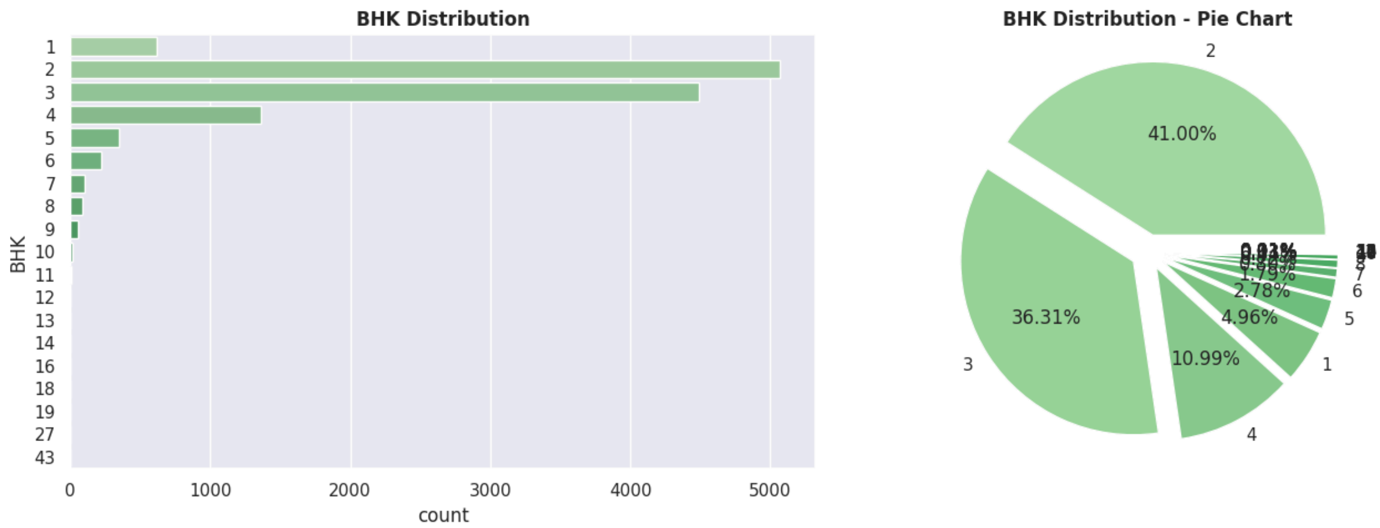
```
colors = ['#A3D2A3','#E6B3B3','#C7E1A6','#B3E0E0','#A0D7D7','#C2C7E1','#D9E1C3' ,"#24C06A" ,'#B3E5BB','#A2D6A6','#A3C1AD','#ff9999','#66b3ff
```

Price Distribution and Normality Check

```
# Set the figure size
plt.figure(figsize=(14, 6))

# Define color palette for consistency
colors = sns.color_palette("Set2")[0]  # Select a single color from the palette

# Plot 1: Price Distribution
plt.subplot(1, 2, 1)
sns.histplot(data["price"], kde=True, bins=120, color=colors, stat='density')
sns.kdeplot(data["price"], color='red')  # Adding the KDE in red separately
plt.title("Price Distribution", fontweight='bold')
plt.xlabel('Price', fontsize=12)
plt.ylabel('Density', fontsize=12)

# Plot 2: Normality Check using Q-Q Plot
plt.subplot(1, 2, 2)
stats.probplot(np.log(data["price"]), dist="norm", plot=pylab)
plt.title("Q-Q Plot for Price (Log Transformed)", fontweight='bold')
plt.xlabel('Theoretical Quantiles', fontsize=12)
plt.ylabel('Sample Quantiles', fontsize=12)

# Adjust layout for a cleaner look
plt.tight_layout()

# Display the plots
plt.show()
```

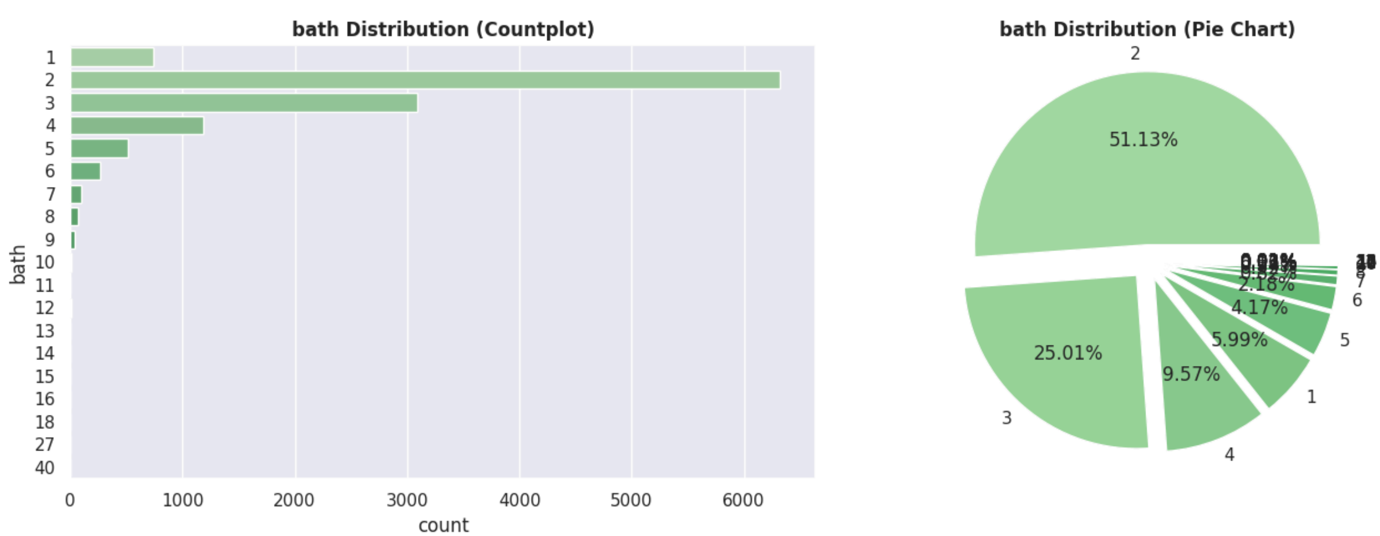
BHK

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the figure size
plt.figure(figsize=(14, 5))

# Define color palette for consistency
colors = sns.color_palette("Greens_d", n_colors=len(data["BHK"].value_counts()))

# Plot 1: Countplot for BHK Distribution
plt.subplot(1, 2, 1)
sns.countplot(y=data["BHK"], palette='Greens_d')
plt.title("BHK Distribution", fontweight='bold')

# Plot 2: Pie Chart for BHK Distribution
plt.subplot(1, 2, 2)
plt.pie(data["BHK"].value_counts(), labels=data["BHK"].value_counts().index,
        autopct="%0.2f%%", colors=colors,
        explode=[0.1] * len(data["BHK"].value_counts()))
plt.title("BHK Distribution - Pie Chart", fontweight='bold')

# Adjust layout
plt.tight_layout()
plt.show()
```



bath

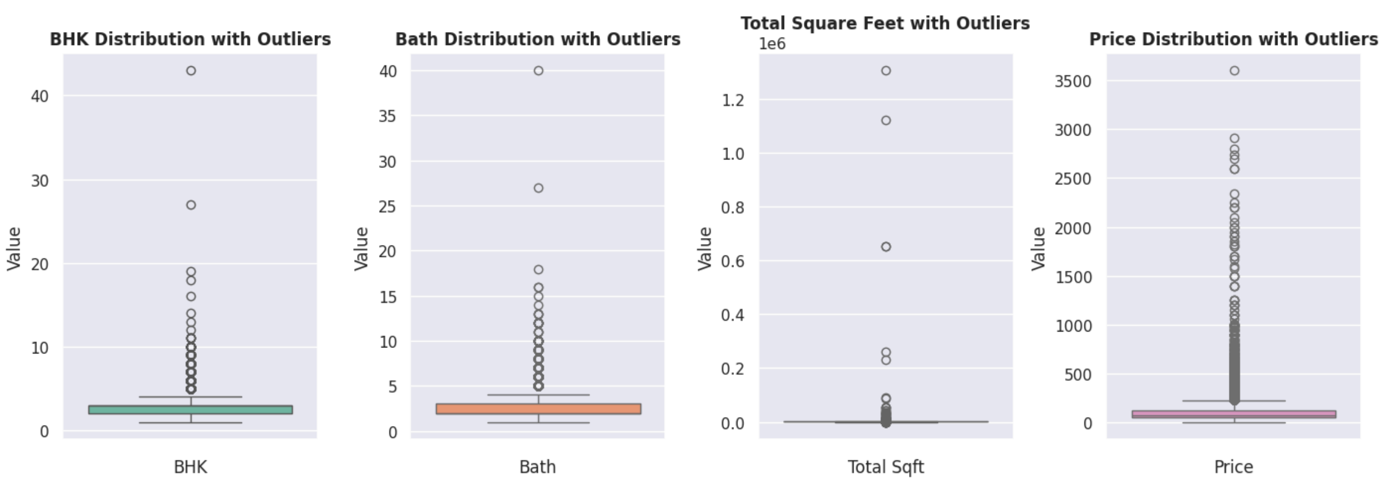
```
plt.figure(figsize=(14,5))

# BHK Distribution - Countplot (Horizontal bar plot)
plt.subplot(1, 2, 1)
sns.countplot(y=data["bath"], palette='Greens_d')
plt.title("bath Distribution (Countplot)", fontweight='bold')

# BHK Distribution - Pie Chart (Percentage of each BHK category)
plt.subplot(1, 2, 2)
plt.pie(data["bath"].value_counts(), labels=data["BHK"].value_counts().index,
        autopct="%0.2f%%", colors=colors,
        explode=[0.1] * len(data["bath"].value_counts()))  # Exploding all sections for visual emphasis
plt.title("bath Distribution (Pie Chart)", fontweight='bold')

# Adjust layout for better spacing between plots
plt.tight_layout()

# Show the plots
plt.show()
```



Outliers

BHK

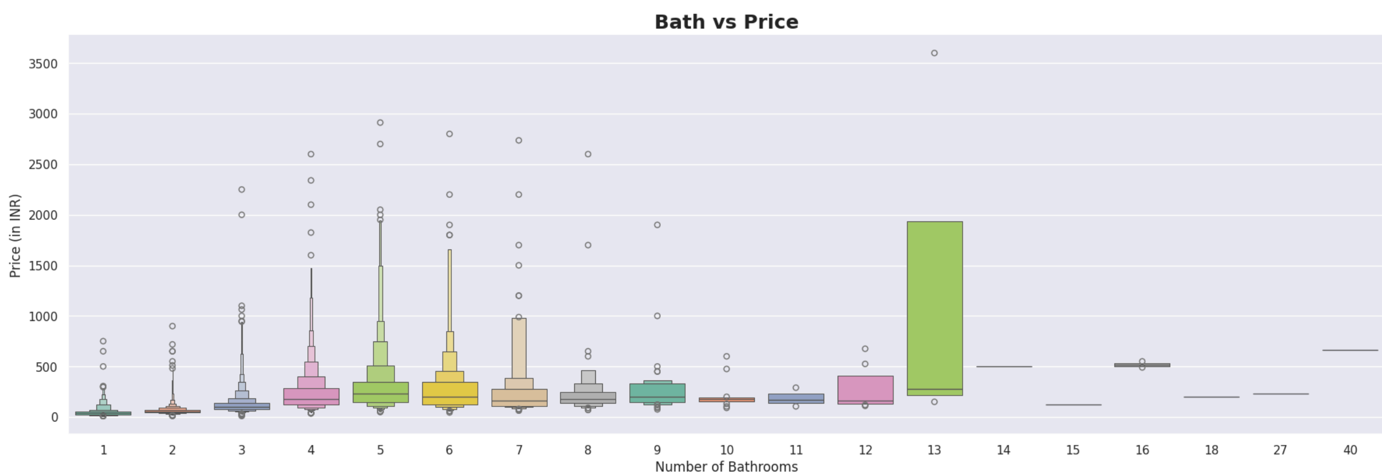
```
# Set the figure size
fig, ax = plt.subplots(1, 4, figsize=(14, 5))

# Define color palette for consistency
colors = sns.color_palette("Set2")

# BOX Plot for BHK
sns.boxplot(data["BHK"], ax=ax[0], color=colors[0])  # Use a valid color index
ax[0].set_title("BHK Distribution with Outliers", fontweight='bold', fontsize=12)
ax[0].set_xlabel('BHK', fontsize=12)
ax[0].set_ylabel('Value', fontsize=12)
ax[0].tick_params(axis='x', labelsize=10)

# BOX Plot for Bath
sns.boxplot(data["bath"], ax=ax[1], color=colors[1])  # Use a valid color index
ax[1].set_title("Bath Distribution with Outliers", fontweight='bold', fontsize=12)
ax[1].set_xlabel('Bath', fontsize=12)
ax[1].set_ylabel('Value', fontsize=12)
ax[1].tick_params(axis='x', labelsize=10)

# BOX Plot for Total Square Feet
sns.boxplot(data["total_sqft"], ax=ax[2], color=colors[2])  # Use a valid color index
ax[2].set_title("Total Square Feet with Outliers", fontweight='bold', fontsize=12)
ax[2].set_xlabel('Total Sqft', fontsize=12)
ax[2].set_ylabel('Value', fontsize=12)
ax[2].tick_params(axis='x', labelsize=10)

# BOX Plot for Price
sns.boxplot(data["price"], ax=ax[3], color=colors[3])  # Use a valid color index
ax[3].set_title("Price Distribution with Outliers", fontweight='bold', fontsize=12)
ax[3].set_xlabel('Price', fontsize=12)
ax[3].set_ylabel('Value', fontsize=12)
ax[3].tick_params(axis='x', labelsize=10)

# Adjust layout for cleaner appearance
plt.tight_layout()

# Display the plots
plt.show()
```

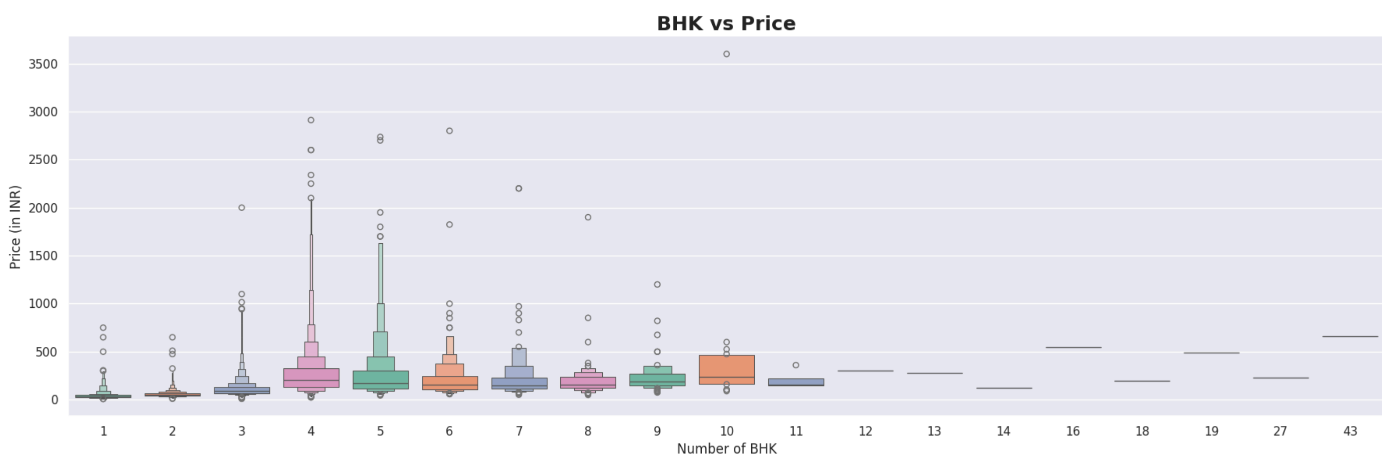


bath vs Price & BHK vs Price Using Catplot

```
# Define color palette for consistency
colors = sns.color_palette("Set2")
```

```
# Bath vs Price
sns.catplot(x="bath", y="price", data=data.sort_values("price", ascending=False),
            kind="boxen", palette=colors, height=6, aspect=3)
plt.title("Bath vs Price", fontsize=18, fontweight='bold')
plt.xlabel("Number of Bathrooms", fontsize=12)
plt.ylabel("Price (in INR)", fontsize=12)

# BHK vs Price (use valid color range)
sns.catplot(x="BHK", y="price", data=data.sort_values("price", ascending=False),
            kind='boxen', palette=colors[:4], height=6, aspect=3)  # Use the first 4 colors
plt.title("BHK vs Price", fontsize=18, fontweight='bold')
plt.xlabel("Number of BHK", fontsize=12)
plt.ylabel("Price (in INR)", fontsize=12)

# Adjust layout and display plots
plt.tight_layout()
plt.show()
```






```
data[data["BHK"]>20]
```

	location	total_sqft	bath	price	BHK	price_per_sqft
1718	other	8000.0	27	230.0	27	2875.0
4684	munnekollal	2400.0	40	660.0	43	27500.0

Identifying Outliers and Data Quality Issues in BHK and Total_Sqft

In the dataset, properties with unusually high values of BHK relative to their total_sqft (such as 43 BHK in 2400 Sqft) present potential data quality issues. These values are unlikely to represent realistic or accurate data and may indicate errors or anomalies that require further investigation or correction.

Problem Identified:

- A 43 BHK property in 2400 Sqft is unusual and likely erroneous based on typical real estate patterns.
- Such discrepancies can significantly affect model performance and predictive accuracy.

```
data["total_sqft"].min()
```

```
1.0
```

A total_sqft value of 1.0 is indeed highly unrealistic for a house, and it is likely an error or outlier.

Such data points may have resulted from:

- Data entry mistakes, where values may have been misrecorded.
- Issues during the data collection or conversion process.

1. **Outliers with BHK and Total_Sqft Below 300 sqft per BHK:**
   - We set a threshold that 1 BHK should not be less than 300 sqft.
   - Rows where the total sqft per BHK is below this threshold are considered as errors or outliers.
2. **Steps to Remove These Errors:**
   - First, we'll filter the rows where the total sqft per BHK is less than 300.
   - Then, remove these rows from the dataset.

```
data[(data["total_sqft"]/data["BHK"])<300]
```

	location	total_sqft	bath	price	BHK	price_per_sqft
9	other	1020.0	6	370.0	6	36274.509804
45	hsr layout	600.0	9	200.0	8	33333.333333
58	murugeshpalya	1407.0	4	150.0	6	10660.980810
68	devarachikkanahalli	1350.0	7	85.0	8	6296.296296
70	other	500.0	3	100.0	3	20000.000000
...	...	...	...	...	...	...
13221	other	1178.0	9	75.0	9	6366.723260
13277	other	1400.0	7	218.0	7	15571.428571
13279	other	1200.0	5	130.0	6	10833.333333
13281	margondanahalli	1375.0	5	125.0	5	9090.909091
13303	vidyaranyapura	774.0	5	70.0	5	9043.927649

733 rows × 6 columns

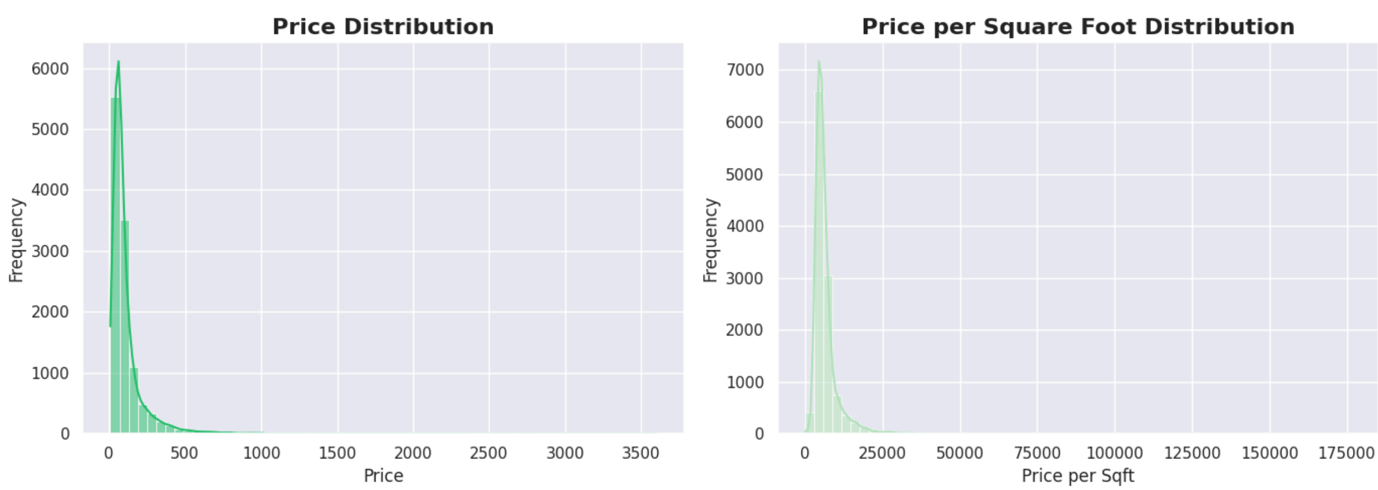
```
data2 = data[~((data["total_sqft"]/data["BHK"])<300)]
data2.shape
```

```
(11632, 6)
```

```
data2[["price_per_sqft" , "price"]].describe()
```

	price_per_sqft	price
count	11632.000000	11632.000000
mean	6391.626781	114.278035
std	4274.127799	156.580092
min	2.257423	9.000000
25%	4250.150361	50.000000
50%	5341.563316	70.387500
75%	7000.000000	120.000000
max	176470.588235	3600.000000

```
plt.figure(figsize=(14,5))

# Histogram for Price Distribution
plt.subplot(1,2,1)
sns.histplot(data2["price"], kde=True, color=colors[7], bins=60)
plt.title("Price Distribution", fontweight="bold", fontsize=16)
plt.xlabel("Price", fontsize=12)
plt.ylabel("Frequency", fontsize=12)

# Histogram for Price per Square Foot Distribution
plt.subplot(1,2,2)
sns.histplot(data2["price_per_sqft"], kde=True, color=colors[8], bins=60)
plt.title("Price per Square Foot Distribution", fontweight="bold", fontsize=16)
plt.xlabel("Price per Sqft", fontsize=12)
plt.ylabel("Frequency", fontsize=12)

# Adjust layout to avoid overlap
plt.tight_layout()

# Show the plots
plt.show()
```



Outlier Removal Using the Empirical Rule for Price per Square Foot

In this step, we apply the Empirical Rule (68-95-99.7 Rule) to remove outliers in the price_per_sqft feature. The rule states that for normally distributed data:

- 68% of the data lies within one standard deviation from the mean.
- Data points outside this range are considered outliers and are removed.

By grouping the data by location, we ensure that the outlier detection is applied within each individual location. This allows for more accurate filtering, as different locations may have different price ranges.

```
def remove_outliers_sqft(data):
    # Create an empty dataframe to store the cleaned data
    data_output = pd.DataFrame()

    # Group data by location
    for key, sub_data in data.groupby('location'):

        # Calculate mean and standard deviation for 'price_per_sqft' in each location
        mean = np.mean(sub_data['price_per_sqft'])
        std = np.std(sub_data['price_per_sqft'])

        # Filter out outliers: Keep data within one standard deviation from the mean
        data_cleaned = sub_data[(sub_data['price_per_sqft'] > (mean - std)) & (sub_data['price_per_sqft'] <= (mean + std))]

        # Concatenate the cleaned data back into the output dataframe
        data_output = pd.concat([data_output, data_cleaned], ignore_index=True)

    return data_output

# Apply the function to remove outliers
data3 = remove_outliers_sqft(data2)

# Print the shapes of the original and cleaned datasets
print("Original data shape:", data2.shape)
print("Cleaned data shape:", data3.shape)
```

```
Original data shape: (11632, 6)
Cleaned data shape: (9567, 6)
```

```
# Statistical Summary for 'price_per_sqft' and 'price' features
data3[["price_per_sqft", "price"]].describe()
```

	price_per_sqft	price
count	9567.000000	9567.000000
mean	5732.821678	93.441949
std	2300.129097	88.814935
min	33.210897	10.000000
25%	4285.714286	50.000000
50%	5217.391304	68.000000
75%	6513.774437	102.000000
max	24509.803922	2200.000000

```
# Visualizing the Distribution of 'Price' and 'Price per Sqft' after Outlier Removal
plt.figure(figsize=(14,5))

plt.subplot(1,2,1)
sns.histplot(data3["price"], kde=True)
plt.title("Histogram Distribution for Price")

plt.subplot(1,2,2)
sns.histplot(data3["price_per_sqft"], kde=True)
plt.title("Histogram Distribution for Price per Sqft")

plt.show()
```

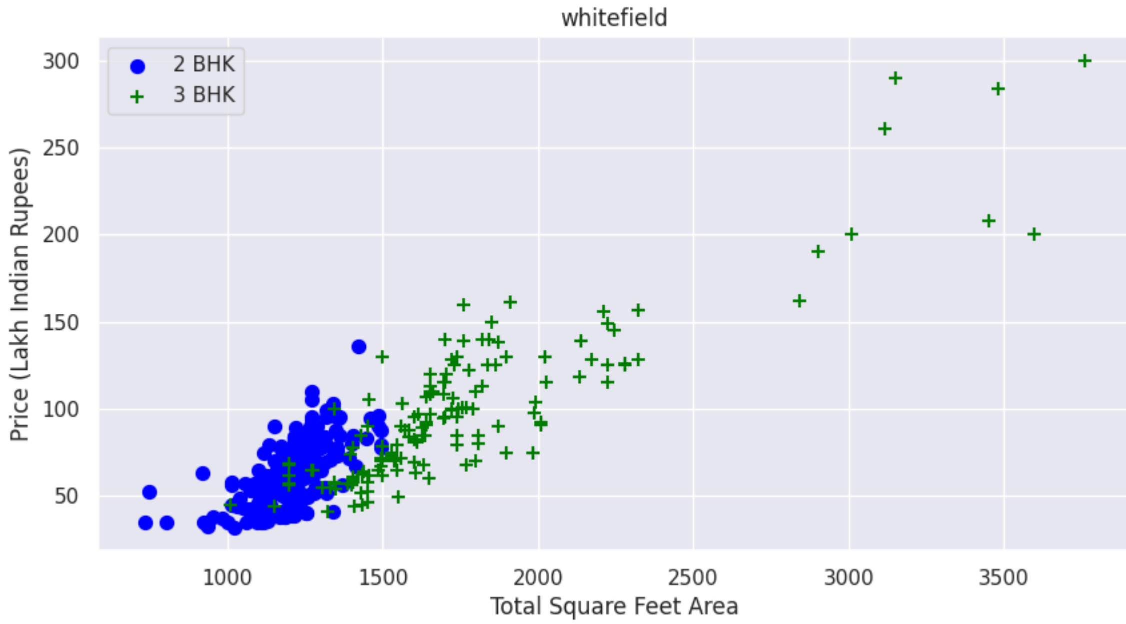
Remove Outliers where Number of Bathrooms is Greater than BHK by More Than 2

```
data4 = data3[(data3["bath"]) < (data3["BHK"] + 2)]
print(data3.shape)
print(data4.shape)
```

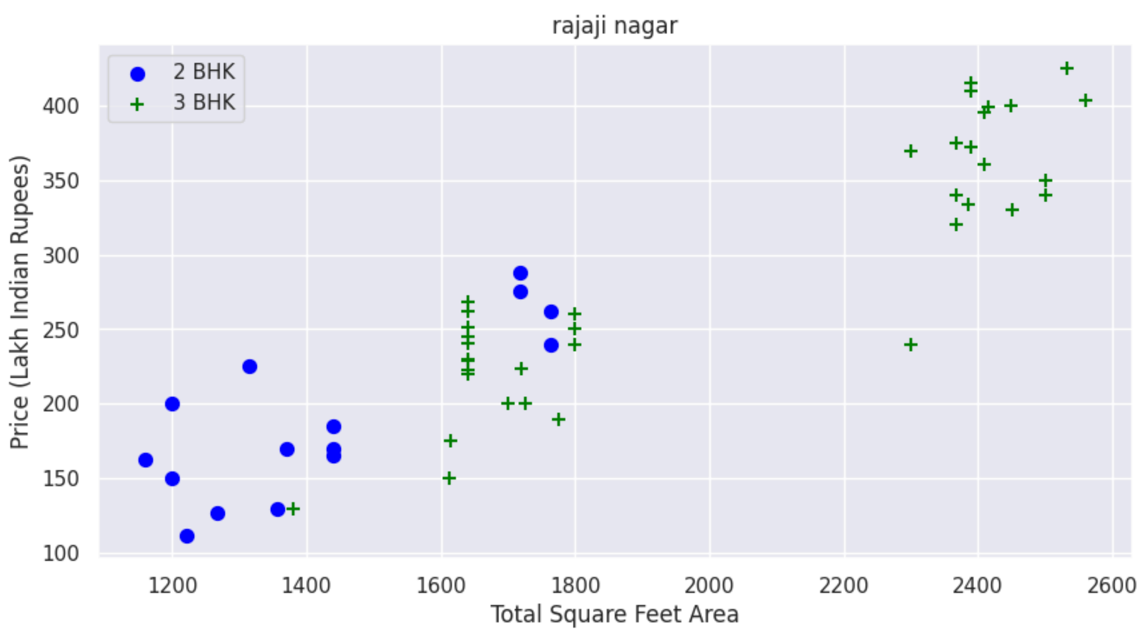
```
(9567, 6)
(9468, 6)
```

Scatter Plot for 2 BHK vs 3 BHK Property Prices in a Given Location

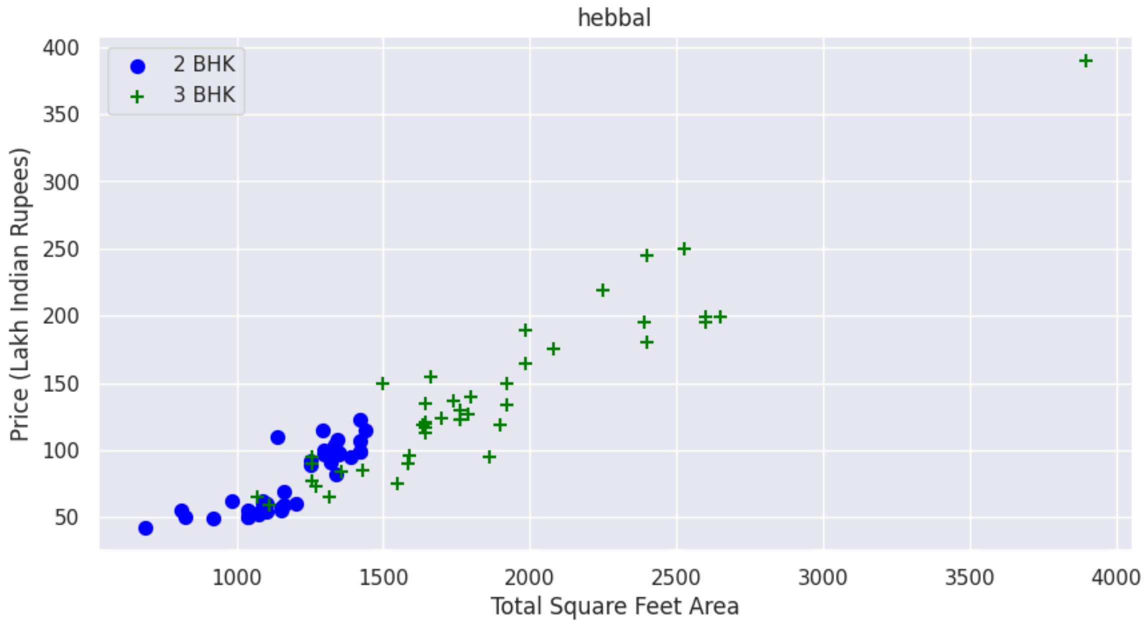
```
def plot_scatter_chart(data,location):
    bhk2 = data[(data["location"]==location) & (data["BHK"]==2)]
    bhk3 = data[(data["location"]==location) & (data["BHK"]==3)]
    plt.rcParams['figure.figsize'] = (10,5)
    plt.scatter(bhk2["total_sqft"],bhk2["price"],color='blue',label='2 BHK', s=50)
    plt.scatter(bhk3["total_sqft"],bhk3["price"],marker='+', color='green',label='3 BHK', s=50)
    plt.xlabel("Total Square Feet Area")
    plt.ylabel("Price (Lakh Indian Rupees)")
    plt.title(location)
    plt.legend()

plot_scatter_chart(data4,"whitefield")
```




```
plot_scatter_chart(data4,"rajaji nagar")
```




```
plot_scatter_chart(data4,"hebbal")
```



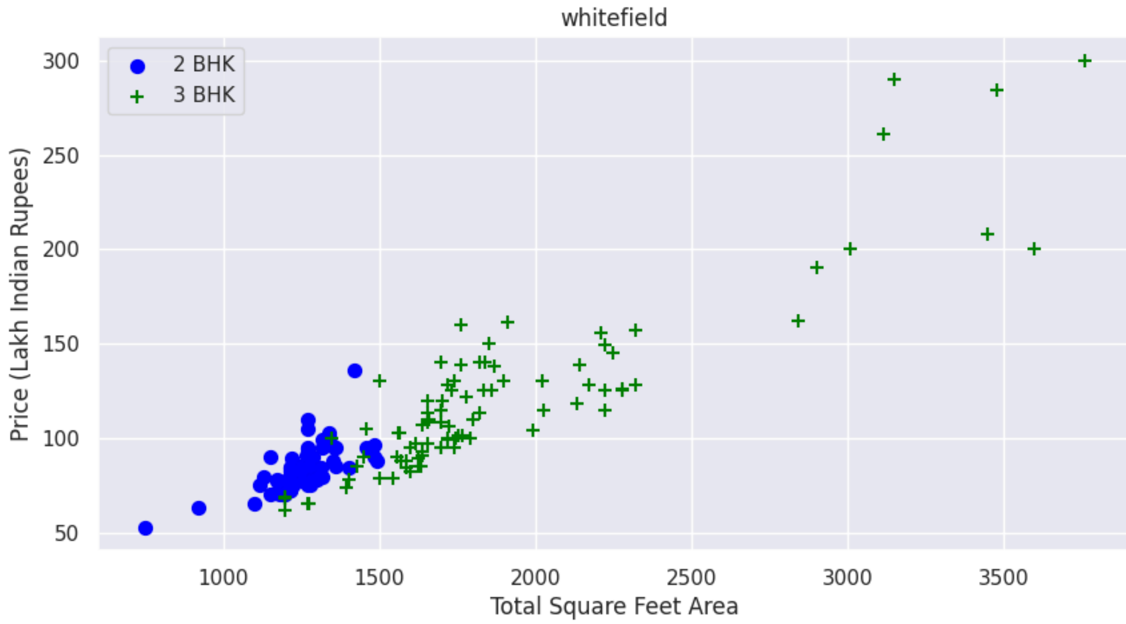
Now we can remove those 2 BHK apartments whose price_per_sqft is less than mean price_per_sqft of 1 BHK apartment

```
def remove_bhk_outliers(data):
    outliers = np.array([])
    for location, location_data in data.groupby('location'):
        bhk_stats = {}
        for bhk, bhk_data in location_data.groupby('BHK'):
            bhk_stats[bhk] = {
                'mean': np.mean(bhk_data['price_per_sqft']),
                'std': np.std(bhk_data['price_per_sqft']),
                'count': bhk_data.shape[0]
            }
        for bhk, bhk_data in location_data.groupby('BHK'):
            stats = bhk_stats.get(bhk-1)
            if stats and stats['count']>5:
                outliers = np.append(outliers, bhk_data[bhk_data['price_per_sqft']<(stats['mean'])].index.values)
    return data.drop(outliers,axis='index')

data5 = remove_bhk_outliers(data4)
print(data4.shape)
print(data5.shape)
```

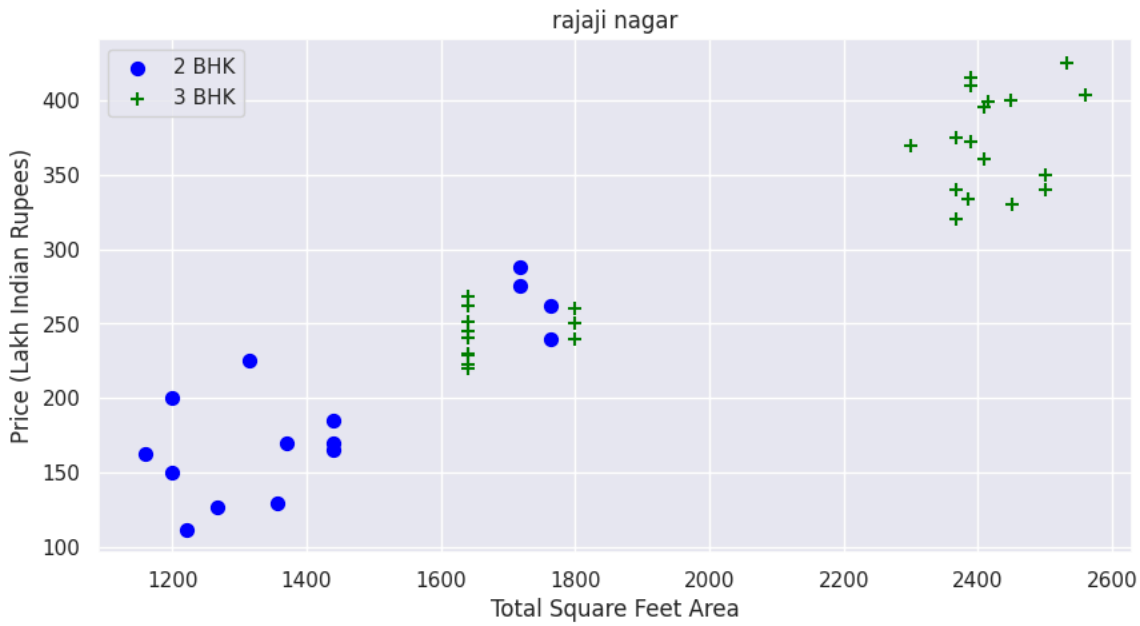
```
(9468, 6)
(6751, 6)
```

After remove these outliers data look like this

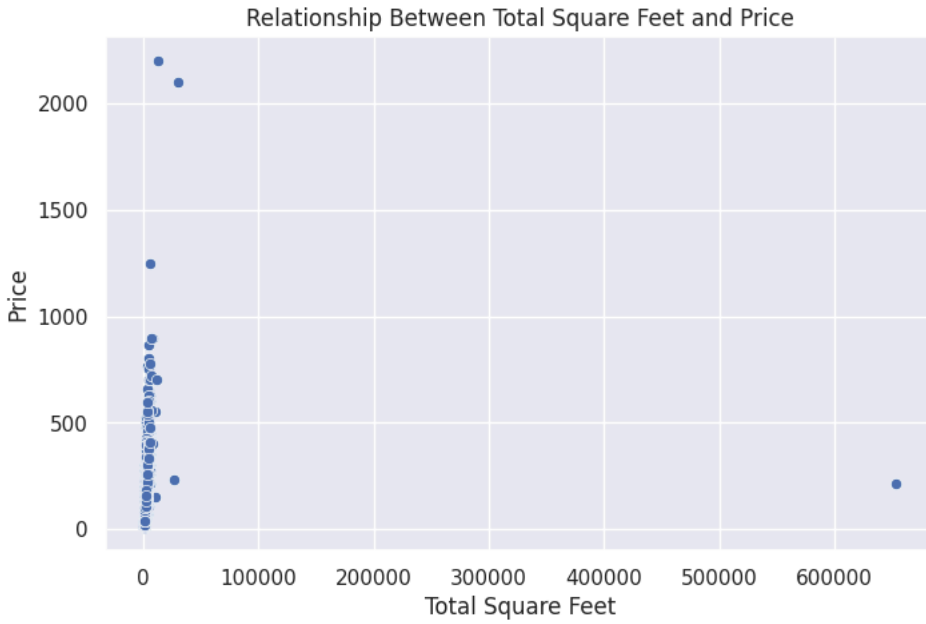
```
plot_scatter_chart(data5,"whitefield")
```




```
plot_scatter_chart(data5,"rajaji nagar")
```



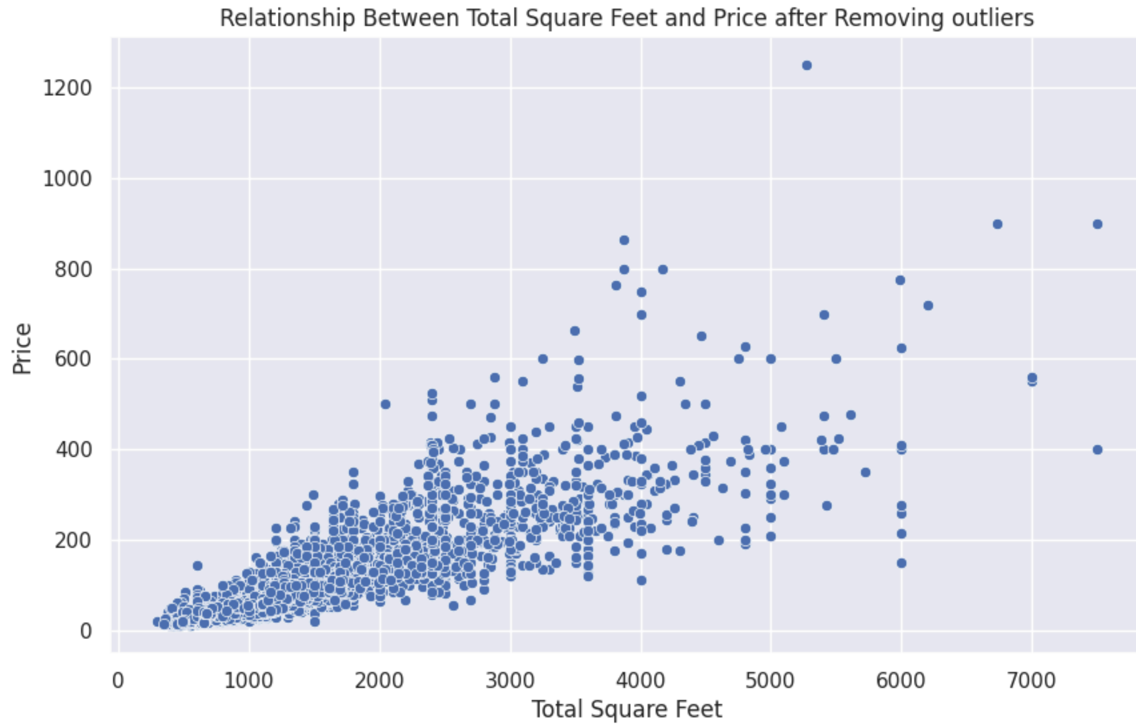

```
plt.figure(figsize=(8, 5))

# Scatter plot
sns.scatterplot(x='total_sqft', y='price', data=data5)

# Add labels and a title for better readability
plt.xlabel('Total Square Feet')
plt.ylabel('Price')
plt.title('Relationship Between Total Square Feet and Price')

# Display the plot
plt.show()
```



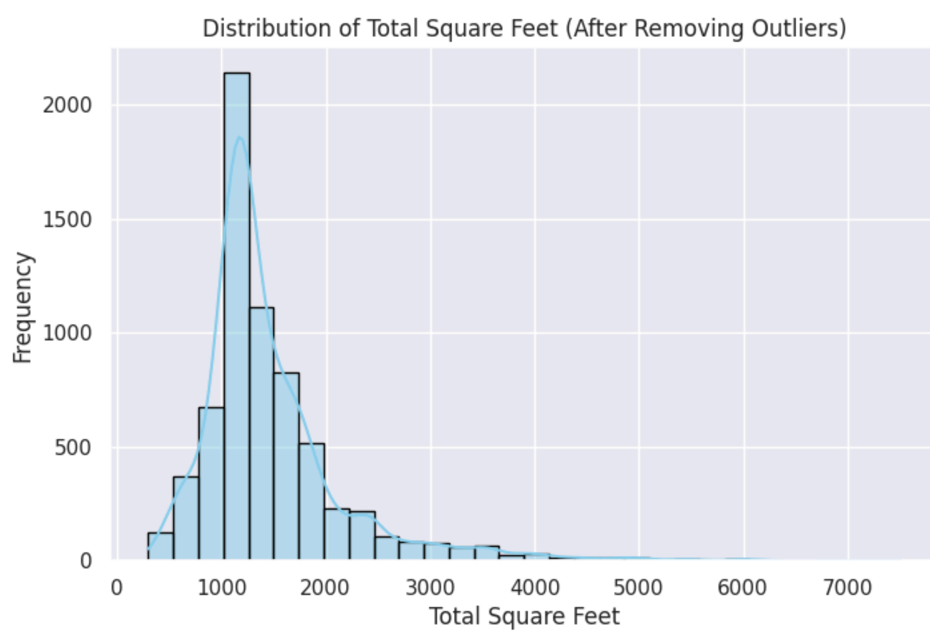

```
# Calculate mean and standard deviation
mean_total_sqft = data5["total_sqft"].mean()
std_total_sqft = data5["total_sqft"].std()

# Filter the dataset to remove outliers based on 1 standard deviation
data6 = data5[(data5["total_sqft"] > (mean_total_sqft - std_total_sqft)) &
              (data5["total_sqft"] < (mean_total_sqft + std_total_sqft))]

# Print dataset shapes before and after removing outliers
print(f"Before removing outliers: {data5.shape}")
print(f"After removing outliers: {data6.shape}")
```

```
Before removing outliers: (6751, 6)
After removing outliers: (6743, 6)
```

```
plt.figure(figsize=(10, 6))

# Scatter plot
sns.scatterplot(x='total_sqft', y='price', data=data6)

# Add labels and a title for better readability
plt.xlabel('Total Square Feet')
plt.ylabel('Price')
plt.title('Relationship Between Total Square Feet and Price after Removing outliers')

# Display the plot
plt.show()
```




```
plt.figure(figsize=(8,5))
# Plotting the histogram
sns.histplot(data6["total_sqft"], bins=30, kde=True, color='skyblue', edgecolor='black')

# Adding labels and title
plt.xlabel('Total Square Feet')
plt.ylabel('Frequency')
plt.title('Distribution of Total Square Feet (After Removing Outliers)')

# Display the plot
plt.show()
```



The correlation matrix represents the relationship between pairs of features in the dataset.

The values range from -1 to 1:

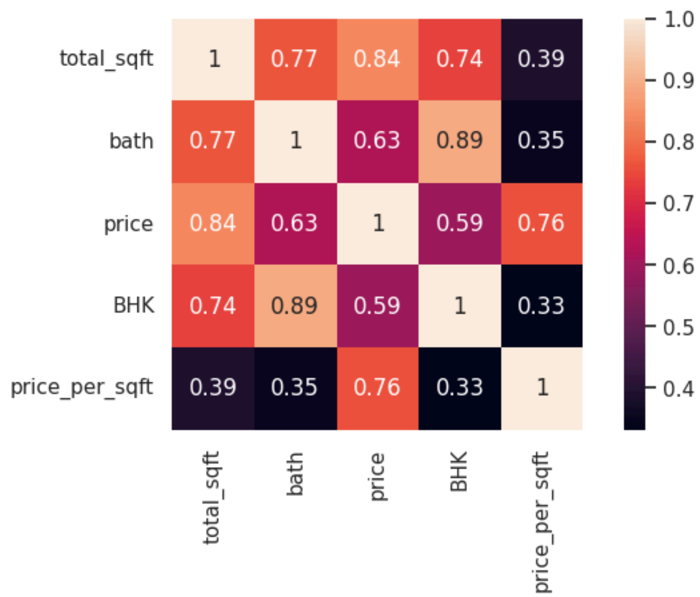
- 1: Perfect positive correlation
- -1: Perfect negative correlation

- 0: No correlation

```
num_featrures = data6.select_dtypes(include='number')
corr = num_featrures.corr()
corr
```

	total_sqft	bath	price	BHK	price_per_sqft
total_sqft	1.000000	0.766407	0.839229	0.738826	0.389864
bath	0.766407	1.000000	0.626663	0.890446	0.348455
price	0.839229	0.626663	1.000000	0.592138	0.759986
BHK	0.738826	0.890446	0.592138	1.000000	0.332259
price_per_sqft	0.389864	0.348455	0.759986	0.332259	1.000000

```
plt.figure(figsize=(8,4))
sns.heatmap(corr , annot=True, square=True)
plt.show()
```



Observations from the Correlation Matrix:

1 bath vs. price: Moderate correlation (0.597). Properties with more bathrooms tend to have higher prices.

2 BHK vs. price: Moderate correlation (0.566). Larger properties (with more bedrooms, hall, and kitchen) are generally priced higher.

3 total_sqft vs. price: Weak correlation (0.105). Total square footage has a limited impact on the property price compared to other features.

```
data6.to_csv("clean_Data.csv",index=False)
```

Model Trainning

```
X = data6.drop(["price","price_per_sqft"],axis=1)
Y = data6["price"]
X.shape , Y.shape
```

```
((6743, 4), (6743,))
```

```
x_train,x_test,y_train,y_test = train_test_split(X,Y, test_size=0.2, random_state=42)
print("x_train size -- >> ",x_train.shape)
print("x_test size -- >> ",x_test.shape)
print("y_train size -- >> ",y_train.shape)
print("y_test size -- >> ",y_test.shape)
```

```
x_train size -- >>  (5394, 4)
x_test size -- >>  (1349, 4)
y_train size -- >>  (5394,)
y_test size -- >>  (1349,)
```

Apply OneHotEncoding

```
ohe = OneHotEncoder(sparse_output=False, dtype=np.int32)

x_train_ohe = ohe.fit_transform(x_train[["location"]])
x_test_ohe = ohe.transform(x_test[["location"]])

ohe_columns = ohe.get_feature_names_out(['location'])


x_train_ohe = pd.DataFrame(x_train_ohe, columns=ohe_columns, index=x_train.index)
x_train_ohe.drop('location_other',axis=1,inplace=True)
x_test_ohe = pd.DataFrame(x_test_ohe, columns=ohe_columns, index=x_test.index)
x_test_ohe.drop('location_other',axis=1,inplace=True)
```

```
x_train_ohe.head(2)
```

	location_1st block jayanagar	location_1st phase jp nagar	location_2nd stage nagarbhavi	location_5th block hbr layout	location_5th phase jp nagar	location_6th phase jp nagar	location_7th phase jp nagar	location_8th phase jp nagar	location_9th phase jp nagar	locat
8222	0	0	0	0	0	0	0	0	0	
5019	0	0	0	0	0	0	0	0	0	

2 rows × 231 columns

```
x_train = pd.concat([x_train.drop('location',axis=1), x_train_ohe] ,axis=1)
x_test = pd.concat([x_test.drop('location',axis=1), x_test_ohe] ,axis=1)
```

```
x_train.shape
```

```
(5394, 234)
```

Apply Scaling

```
scaling = StandardScaler()
x_train[["total_sqft","bath","BHK"]] = scaling.fit_transform(x_train[["total_sqft","bath","BHK"]])
x_test[["total_sqft","bath","BHK"]] = scaling.transform(x_test[["total_sqft","bath","BHK"]])
```

```
x_train.head(2)
```

	total_sqft	bath	BHK	location_1st block jayanagar	location_1st phase jp nagar	location_2nd stage nagarbhavi	location_5th block hbr layout	location_5th phase jp nagar	location_6th phase jp nagar	location_7th phase jp nagar
8222	-0.229514	-0.450767	-0.543940	0	0	0	0	0	0	0
5019	-0.888192	-1.502880	-1.631013	0	0	0	0	0	0	0

2 rows × 234 columns

```
def train_model(model):
  model.fit(x_train,y_train)

  y_train_pred = model.predict(x_train)
  y_test_pred = model.predict(x_test)
  print()
  print("=="*20)
  print('model -- >> ', model)
  print()
  print("Train Data Peformance ------>>>>  ::: ")
  print()

  train_score = r2_score(y_train_pred,y_train)
```

```
    train_MSE = mean_squared_error(y_train_pred,y_train)
    train_MAE = mean_absolute_error(y_train_pred,y_train)
    train_RMSE = root_mean_squared_error(y_train_pred,y_train)

    print("r2_Score of Train Data is -->>> " , train_score)
    print("mean_squared_error of Train Data is -->>> " , train_MSE)
    print("mean_absolute_error of Train Data is -->>> " , train_MAE)
    print("root_mean_squared_error of Train Data is -->>> " , train_RMSE)

    print()
    print("--"*20)
    print()


    print("Test Data Peformance ------>>>>  ::: ")
    print()

    test_score = r2_score(y_test_pred,y_test)
    test_MSE = mean_squared_error(y_test_pred,y_test)
    test_MAE = mean_absolute_error(y_test_pred,y_test)
    test_RMSE = root_mean_squared_error(y_test_pred,y_test)

    print("r2_Score of Test Data is -->>> " , test_score)
    print("mean_squared_error of Test Data is -->>> " , test_MSE)
    print("mean_absolute_error of Test Data is -->>> " , test_MAE)
    print("root_mean_squared_error of Test Data is -->>> " , test_RMSE)
```

```
# Initialize Regression Models for Comparison
# The following models are initialized for training and evaluation on the dataset:

model_LinearRegression = LinearRegression()              # Linear Regression
model_Lasso = Lasso()                       # Lasso Regression
model_ridge = Ridge()                       # Ridge Regression
model_svm = SVR()                           # Support Vector Machine for Regression
model_DecisionTree = DecisionTreeRegressor()          # Decision Tree Regressor
model_RandomForest = RandomForestRegressor()          # Random Forest Regressor
model_AdaBoost = AdaBoostRegressor()             # AdaBoost Regressor
model_GradientBoosting = GradientBoostingRegressor()      # Gradient Boosting Regressor
model_XGBoost = XGBRegressor()                 # XGBoost Regressor
model_CatBoost = CatBoostRegressor(verbose=False) # CatBoost Regressor
```

Linear Regression

```
train_model(model_LinearRegression)
train_model(model_Lasso)
train_model(model_ridge)
train_model(model_svm)
train_model(model_DecisionTree)
train_model(model_RandomForest)
train_model(model_AdaBoost)
train_model(model_GradientBoosting)
train_model(model_XGBoost)
train_model(model_CatBoost)
```

```
========================================
model -- >>  LinearRegression()

Train Data Peformance ------>>>>  :::

r2_Score of Train Data is -->>>  0.8536502850138281
mean_squared_error of Train Data is -->>>  909.7955093482018
mean_absolute_error of Train Data is -->>>  17.572478924276503
root_mean_squared_error of Train Data is -->>>  30.162816668013644

----------------------------------------

Test Data Peformance ------>>>>  :::

r2_Score of Test Data is -->>>  0.8230464196243785
mean_squared_error of Test Data is -->>>  909.1407805814669
mean_absolute_error of Test Data is -->>>  18.37500940223718
```

```
root_mean_squared_error of Test Data is -->>>  30.151961471543885

=======================================
model -- >>  Lasso()

Train Data Peformance ------>>>>  :::

r2_Score of Train Data is -->>>  0.584388654743644
mean_squared_error of Train Data is -->>>  2033.2919074546267
mean_absolute_error of Train Data is -->>>  25.231096571755433
root_mean_squared_error of Train Data is -->>>  45.092038182528704

---------------------------------------

Test Data Peformance ------>>>>  :::

r2_Score of Test Data is -->>>  0.6667081829486267
mean_squared_error of Test Data is -->>>  1527.4156313881842
mean_absolute_error of Test Data is -->>>  24.134643120306354
root_mean_squared_error of Test Data is -->>>  39.082165131785935

=======================================
model -- >>  Ridge()

Train Data Peformance ------>>>>  :::

r2_Score of Train Data is -->>>  0.8459543335361022
mean_squared_error of Train Data is -->>>  925.5030844859187
mean_absolute_error of Train Data is -->>>  17.553688389364865
root_mean_squared_error of Train Data is -->>>  30.422082185246932

---------------------------------------

Test Data Peformance ------>>>>  :::

r2_Score of Test Data is -->>>  0.8216196959238409
mean_squared_error of Test Data is -->>>  897.738823918265
mean_absolute_error of Test Data is -->>>  18.118590430878037
root_mean_squared_error of Test Data is -->>>  29.962290031275398
```

Hyper Parameter Tunning on Ridge

- Hyperparameter tuning improved the model's training accuracy, but there was a small trade-off on generalization to new data (the test set).

```
# Define the parameter grid for Ridge
params_ridge = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'fit_intercept': [True, False],
    'positive': [True, False],
    'copy_X': [True, False],
    'max_iter': [100,200,500,1000, 1500],
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'saga','lbfgs']

}

# Initialize the Ridge model
ridge_tun = Ridge(random_state=42)

# Set up the cross-validation strategy
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)

# Perform GridSearchCV
gs_ridge = GridSearchCV(ridge_tun, params_ridge, cv=cv)

# Fit the model on the training data
gs_ridge.fit(x_train, y_train)

# Print the best parameters and score
print("Best Parameters:", gs_ridge.best_params_)
print("Best Cross-validation Score:", gs_ridge.best_score_)

# Evaluate the model on the training and testing datasets
y_train_pred = gs_ridge.predict(x_train)
y_test_pred = gs_ridge.predict(x_test)

# Print performance metrics for training data
print("Train Data Performance ------>>>>  ::: ")
print("R2 Score of Train Data:", r2_score(y_train, y_train_pred))
```

```
print("Mean Squared Error of Train Data:", mean_squared_error(y_train, y_train_pred)

# Print performance metrics for test data
print("Test Data Performance ------>>>>  ::: ")
print("R2 Score of Test Data:", r2_score(y_test, y_test_pred))
print("Mean Squared Error of Test Data:", mean_squared_error(y_test, y_test_pred))
```

```
Best Parameters: {'alpha': 0.1, 'copy_X': True, 'fit_intercept': True, 'max_iter': 500, 'positive': False, 'solver': 'saga'}
Best Cross-validation Score: 0.848337591056028
Train Data Performance ------>>>>  :::
R2 Score of Train Data: 0.8721647813284064
Mean Squared Error of Train Data: 911.0024687588641
Test Data Performance ------>>>>  :::
R2 Score of Test Data: 0.8280119886757524
Mean Squared Error of Test Data: 904.139790136041
```

```
ridge = Ridge(
    alpha=0.5,
    copy_X=True,
    fit_intercept=True,
    max_iter=500,
    positive=False,
    solver='saga'
)

# Fit the model on the training data
ridge.fit(x_train, y_train)

# Evaluate the model on the training and testing datasets
y_train_pred = ridge.predict(x_train)
y_test_pred = ridge.predict(x_test)

# Print performance metrics for training data
print("Train Data Performance ------>>>>  ::: ")
print("R2 Score of Train Data:", r2_score(y_train, y_train_pred))
print("Mean Squared Error of Train Data:", mean_squared_error(y_train, y_train_pred))

# Print performance metrics for test data
print("Test Data Performance ------>>>>  ::: ")
print("R2 Score of Test Data:", r2_score(y_test, y_test_pred))
print("Mean Squared Error of Test Data:", mean_squared_error(y_test, y_test_pred))
```

```
Train Data Performance ------>>>>  :::
R2 Score of Train Data: 0.8714459103611754
Mean Squared Error of Train Data: 916.1254171346866
Test Data Performance ------>>>>  :::
R2 Score of Test Data: 0.8289500067727282
Mean Squared Error of Test Data: 899.2086354653555
```

Hyper Parameter Tunning on XGBoost

```
# # Define the parameter grid for XGBoost
# params_xgb = {
#     'learning_rate': [0.01, 0.03, 0.05,0.0],  # Fine-tuned for performance
#     'min_child_weight': [1, 3, 5],  # Tuning the minimum leaf node weight
#     'max_depth': [4, 6, 8],  # Tree depth tuning
#     'subsample': [0.7, 1.0],  # Sampling fractions for preventing overfitting
#     'colsample_bytree': [0.7, 1.0],  # Column subsampling, also for overfitting control
```