Real Estate Price Prediction - Bengaluru House Price Dataset

E.D.A. (Exploratory Data Analysis) and trainig on the Bengaluru House Price Dataset using sklearn libraries and matplotlib.

1. Importing necessary Libraries

> Add blockquote

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib
matplotlib.rcParams['figure.figsize'] = (20,10)
import seaborn as sns
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
import pickle
import json
```

2. Reading Data

We will use pandas `read_csv` function to read the data in csv file.

```
data_pred = pd.read_csv("https://raw.githubusercontent.com/Bhagyakhajuria/Bengaluru-House-price-prediction/main/Bengaluru_House_Data%20(1
```

```
data_pred
```

	area_type	availability	location	size	society	total_sqft	bath
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0
...	...	...	...	...	...	...	...
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0

Next steps: Generate code with `data_pred` View recommended plots

```
data_pred.head()
```

	area_type	availability	location	size	society	total_sqft	bath	bal
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	
2	Built-up	Ready To	Uttarahalli	3 BHK	NaN	1440	2.0	

Next steps: Generate code with data_pred View recommended plots

#

3. Data Preprcessing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format.

```
data_pred.shape
```

```
(13320, 9)
```

```
data_pred.groupby('area_type')['area_type'].agg('count')
```

```
area_type
Built-up  Area          2418
Carpet  Area              87
Plot  Area              2025
Super built-up  Area    8790
Name: area_type, dtype: int64
```

groupby() function is used to split the data into groups based on some criteria and agg() function abbreviation of aggregate is used to define what we want to do with the grouped data.

```
df1 =data_pred.drop(['availability','area_type','society','balcony'],axis =(1))
df1.head()
```

	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

Next steps: Generate code with df1 View recommended plots

```
df1.isnull().sum()          # finding total empty values in each column
```

```
location       1
size          16
total_sqft     0
bath          73
price          0
dtype: int64
```

```
df2 = df1.dropna()            # dropping 'NA' values
df2.isnull().sum()
```

```
location      0
size          0
total_sqft    0
bath          0
price         0
dtype: int64
```

```
df2.shape
```

```
(13246, 5)
```

```
df2['size'].unique()        # checking unique values in size column
```

```
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)
```

```
df2['bhk'] = df2['size'].apply(lambda x:int(x.split(' ')[0]))
# getting the number of bedrooms from size column
```

```
<ipython-input-15-e55918dc8493>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df2['bhk'] = df2['size'].apply(lambda x:int(x.split(' ')[0]))
```

```
df2.head()
```

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

Next steps: Generate code with df2 View recommended plots

```
df2.drop(('size'),axis = (1))
```

	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
2	Uttarahalli	1440	2.0	62.00	3
3	Lingadheeranahalli	1521	3.0	95.00	3
4	Kothanur	1200	2.0	51.00	2
...	...	...	...	...	...
13315	Whitefield	3453	4.0	231.00	5
13316	Richards Town	3600	5.0	400.00	4
13317	Raja Rajeshwari Nagar	1141	2.0	60.00	2
13318	Padmanabhanagar	4689	4.0	488.00	4
13319	Doddathoguru	550	1.0	17.00	1

13246 rows × 5 columns

```
df2['bhk'].unique()           # checking unique values in size column
```

```
array([ 2,  4,  3,  6,  1,  8,  7,  5, 11,  9, 27, 10, 19, 16, 43, 14, 12,
       13, 18])
```

```
df2[df2.bhk>20]
```

	location	size	total_sqft	bath	price	bhk
1718	2Electronic City Phase II	27 BHK	8000	27.0	230.0	27
4684	Munnekollal	43 Bedroom	2400	40.0	660.0	43

```
df2.total_sqft.unique()
```

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

Finding rows with extraordinary values! 😅 (Outliers)

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

```
df2[~df2['total_sqft'].apply(is_float)].head(10)         # finding values those not got converted
```

	location	size	total_sqft	bath	price	bhk
30	Yelahanka	4 BHK	2100 - 2850	4.0	186.000	4
122	Hebbal	4 BHK	3067 - 8156	4.0	477.000	4
137	8th Phase JP Nagar	2 BHK	1042 - 1105	2.0	54.005	2
165	Sarjapur	2 BHK	1145 - 1340	2.0	43.490	2
188	KR Puram	2 BHK	1015 - 1540	2.0	56.800	2
410	Kengeri	1 BHK	34.46Sq. Meter	1.0	18.500	1
549	Hennur Road	2 BHK	1195 - 1440	2.0	63.770	2
648	Arekere	9 Bedroom	4125Perch	9.0	265.000	9
661	Yelahanka	2 BHK	1120 - 1145	2.0	48.130	2
672	Bettahalsoor	4 Bedroom	3090 - 5002	4.0	445.000	4

```
def convert_sqft_to_num(x):
    '''
    Function to convert those unusual format of data
    '''
    token = x.split('-')
    if len(token) == 2:
        return (float(token[0]) + float(token[1])) / 2
    try:
        return float(x)
    except:
        return None

    # if x : return float(x)
    # else : return None
```

```
convert_sqft_to_num('2100-2850')           # usage of the function
```

```
2475.0
```

Double-click (or enter) to edit

```
df3 = df2.copy()
df3['total_sqft'] = df3['total_sqft'].apply(convert_sqft_to_num)
df3.head(3)
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3

Next steps: Generate code with df3 View recommended plots

```
df3.loc[30]                    # loc function is used to see data row-wise
```

```
location      Yelahanka
size              4 BHK
total_sqft       2475.0
bath                4.0
price             186.0
bhk                   4
Name: 30, dtype: object
```

```
df4 = df3.copy()               # copy function is used to copy the whole dataframe
df4.head()
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3
4	Kothanur	2 BHK	1200.0	2.0	51.00	2

Next steps: Generate code with df4 View recommended plots

```
df4['price_per_sqft'] = df4['price']*100000/df4['total_sqft']
# making a new column in the dataframe named `price_per_sqft` and see the logic to create it
df4.head()
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699.810606
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	4615.384615
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305.555556
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245.890861

Next steps: Generate code with df4 View recommended plots

```
len(df4.location.unique())
```

```
1304
```

```
df4.location = df4.location.apply(lambda x : x.strip())
# strip is used to remove the white spaces around the data points
```

```
location_stats = df4.groupby('location')['location'].agg('count').sort_values(ascending=False)    # sorting the location column in desce
location_stats
```

```
location
Whitefield                 535
Sarjapur  Road             392
Electronic City            304
Kanakpura Road             266
Thanisandra                236
                          ...
1 Giri Nagar                 1
Kanakapura Road,             1
Kanakapura main  Road        1
Karnataka Shabarimala        1
whitefiled                   1
Name: location, Length: 1293, dtype: int64
```

```
len(location_stats[location_stats<=10])         # totaling the minor locations
```

```
1052
```

```
location_stats_less_than_10 = location_stats[location_stats <= 10]
location_stats_less_than_10
```

```
location
Basapura                 10
1st Block Koramangala    10
Gunjur Palya             10
Kalkere                  10
Sector 1 HSR Layout      10
                         ..
1 Giri Nagar              1
Kanakapura Road,          1
Kanakapura main  Road     1
Karnataka Shabarimala     1
whitefiled                1
Name: location, Length: 1052, dtype: int64
```

```
df4.location = df4.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)    # changing minor locations into `othe
```

```
len(df4.location.unique())
```

```
242
```

Removal of Outliers

```
df4.head(10)
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699.810606
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	4615.384615
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305.555556
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245.890861
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250.000000
5	Whitefield	2 BHK	1170.0	2.0	38.00	2	3247.863248
6	Old Airport Road	4 BHK	2732.0	4.0	204.00	4	7467.057101
7	Rajaji Nagar	4 BHK	3300.0	4.0	600.00	4	18181.818182
8	Marathahalli	3 BHK	1310.0	3.0	63.25	3	4828.244275
9	other	6 Bedroom	1020.0	6.0	370.00	6	36274.509804

Next steps: Generate code with df4 View recommended plots

```
df4[df4.total_sqft / df4.bhk < 300].head()
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
9	other	6 Bedroom	1020.0	6.0	370.0	6	36274.509804
45	HSR Layout	8 Bedroom	600.0	9.0	200.0	8	33333.333333
58	Murugeshpalya	6 Bedroom	1407.0	4.0	150.0	6	10660.980810
68	Devarachikkanahalli	8 Bedroom	1350.0	7.0	85.0	8	6296.296296
70	other	3 Bedroom	500.0	3.0	100.0	3	20000.000000

```
df4.shape
```

```
(13246, 7)
```

```
df5= df4[~(df4.total_sqft/df4.bhk<300)]
df5.shape
```

```
(12502, 7)
```

```
df5.price_per_sqft.describe()
```

```
count     12456.000000
mean       6308.502826
std        4168.127339
min         267.829813
25%        4210.526316
50%        5294.117647
75%        6916.666667
max      176470.588235
Name: price_per_sqft, dtype: float64
```

4. Data Cleaning

find outliers and remove them.

```
def remove_pps_outliers(df):

    df_out = pd.DataFrame()
    for key, subdf in df.groupby('location'):
        m = np.mean(subdf.price_per_sqft)
        st = np.std(subdf.price_per_sqft)
        reduced_df = subdf[(subdf.price_per_sqft>(m-st)) & (subdf.price_per_sqft<=(m+st))]
        df_out = pd.concat([df_out,reduced_df],ignore_index=True)
    return df_out
```

```
df6 =  remove_pps_outliers(df5)
df6.shape
```

```
(10241, 7)
```

```
df6.head()
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	1st Block Jayanagar	4 BHK	2850.0	4.0	428.0	4	15017.543860
1	1st Block Jayanagar	3 BHK	1630.0	3.0	194.0	3	11901.840491
2	1st Block Jayanagar	3 BHK	1875.0	2.0	235.0	3	12533.333333
3	1st Block Jayanagar	3 BHK	1200.0	2.0	130.0	3	10833.333333
4	1st Block Jayanagar	2 BHK	1235.0	2.0	148.0	2	11983.805668

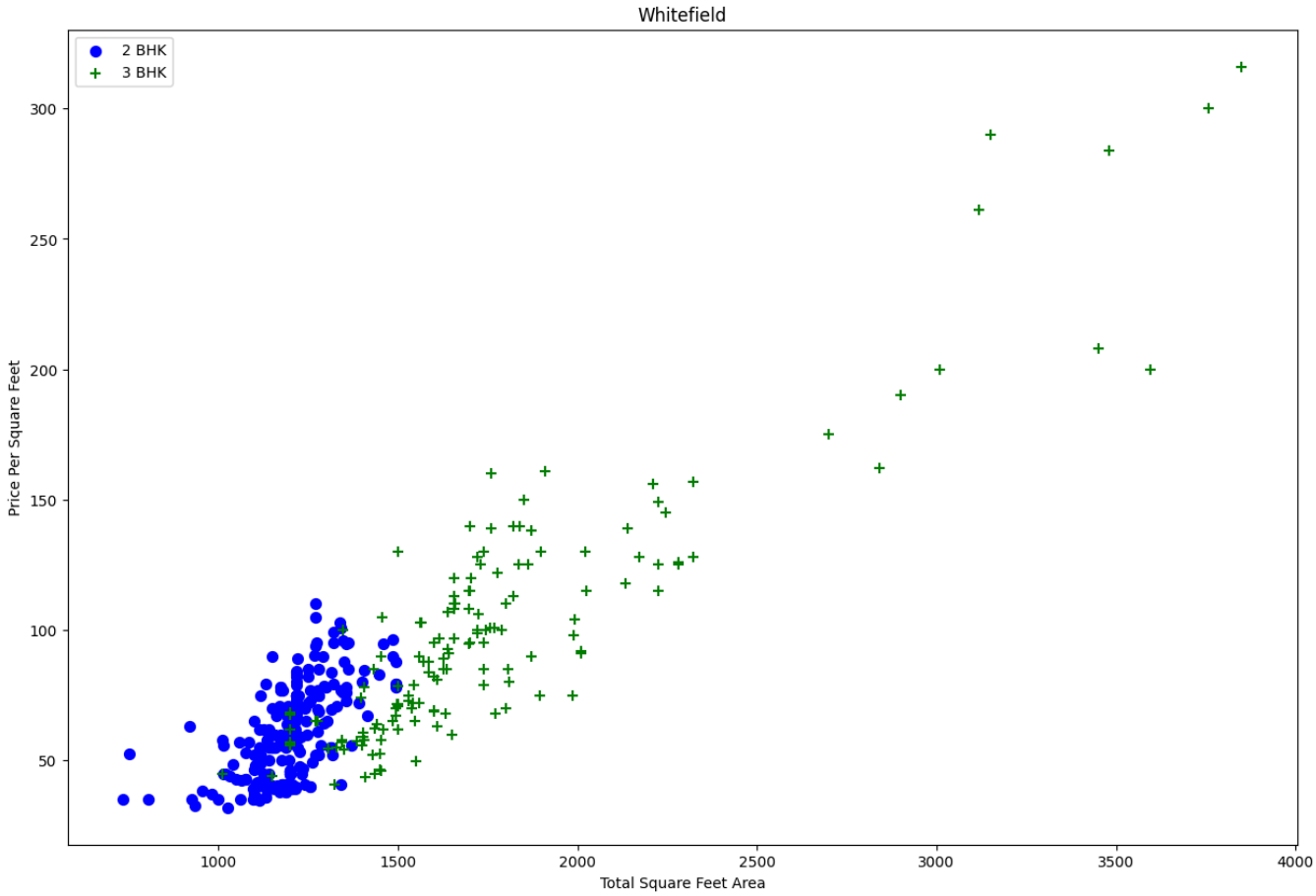
Next steps: Generate code with df6 View recommended plots

5. Data Visualization

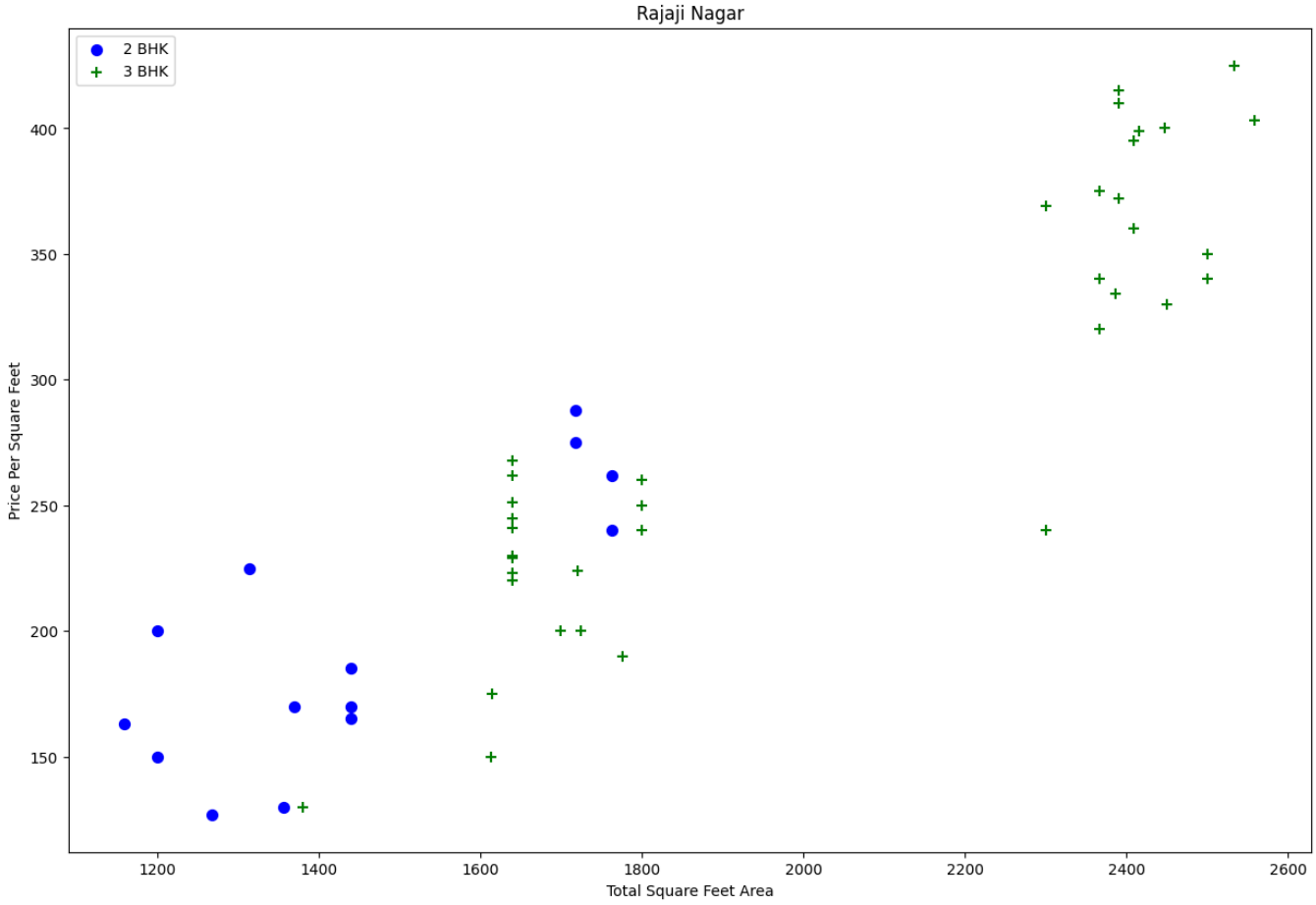
Time to visualize our data

```
def plot_scatter_chart(df,location):
    '''
    Function that will help us to visualize the data of the different locations
    '''
    bhk2 = df[(df.location==location) & (df.bhk==2)]
    bhk3 = df[(df.location==location) & (df.bhk==3)]
    matplotlib.rcParams['figure.figsize'] = (15, 10)
    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 Per Square Feet')
    plt.title(location)
    plt.legend()
```

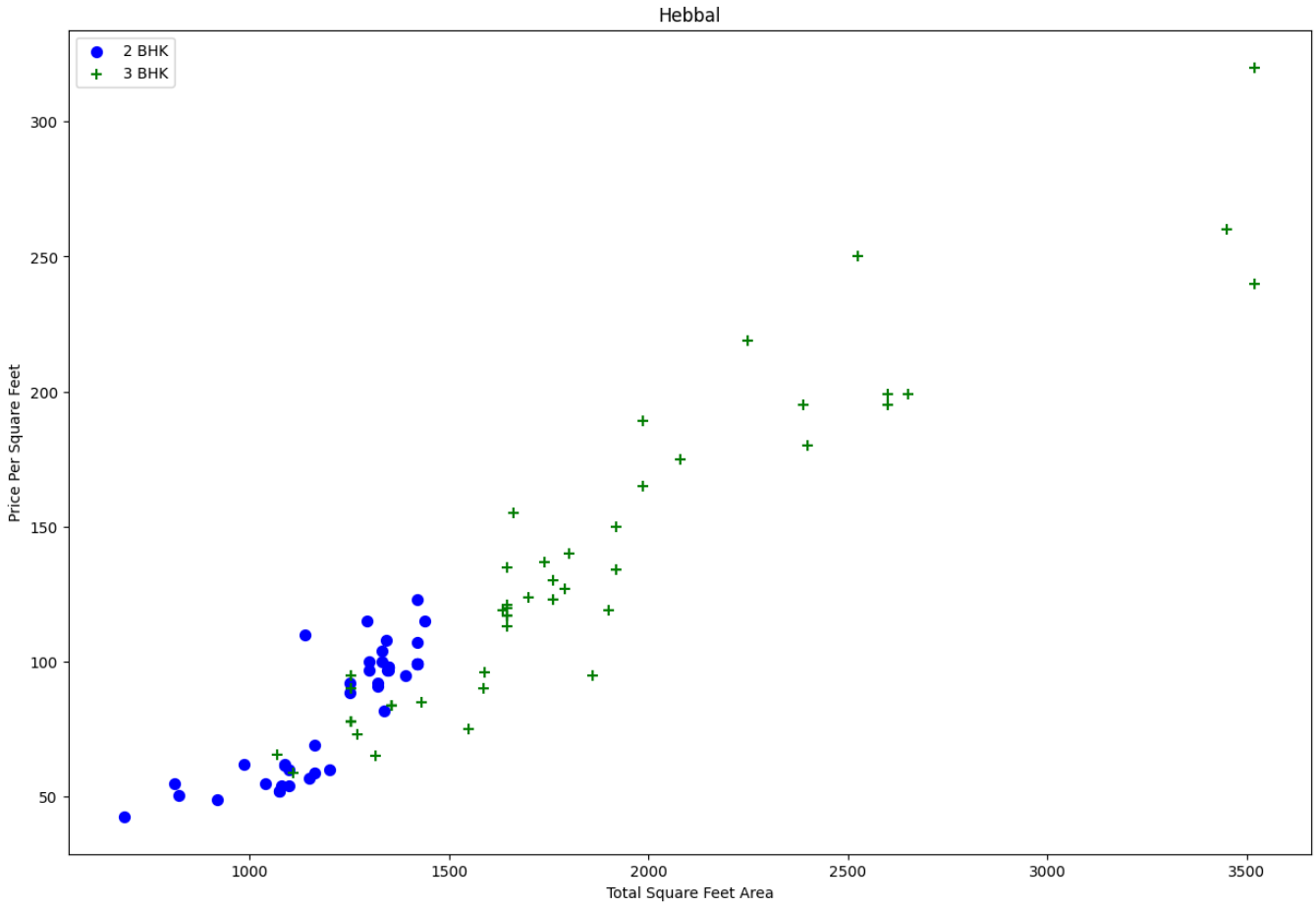
```
plot_scatter_chart(df6, 'Whitefield')
```




```
plot_scatter_chart(df6, 'Rajaji Nagar')
```




```
plot_scatter_chart(df6,"Hebbal")
```

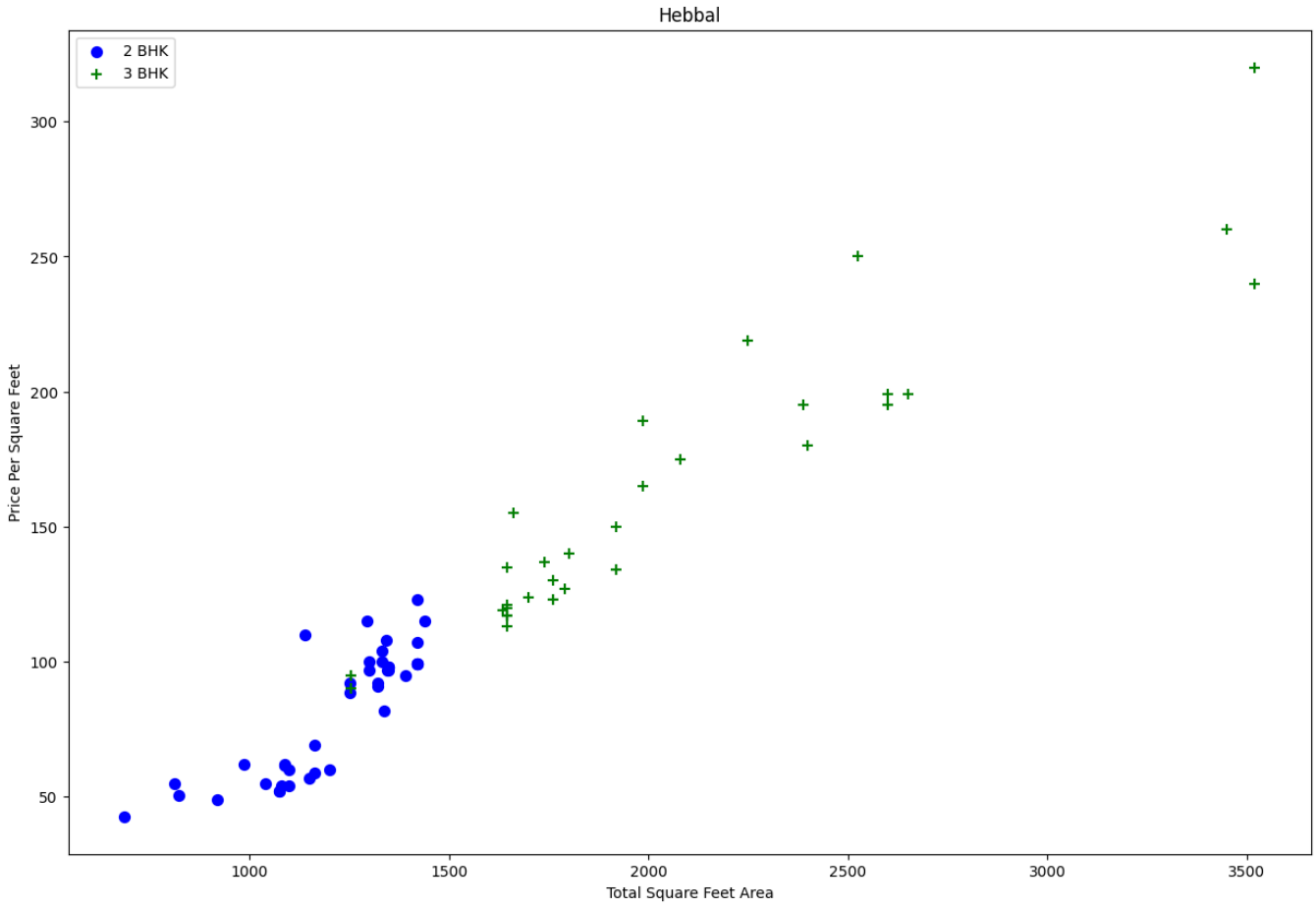



```
def remove_bhk_outliers(df):
    '''
    Function to clear stuff (outliers) in the bhk column so that we don't live in a hypothetical dataset. 😄
    '''
    exclude_indices = np.array([])
    for location, location_df in df.groupby('location'):
        bhk_stats = {}
        for bhk, bhk_df in location_df.groupby('bhk'):
            bhk_stats[bhk] = {
                'mean': np.mean(bhk_df.price_per_sqft),
                'std': np.std(bhk_df.price_per_sqft),
                'count': bhk_df.shape[0]
            }
        for bhk, bhk_df in location_df.groupby('bhk'):
            stats = bhk_stats.get(bhk-1)
            if stats and stats['count']>5:
                exclude_indices = np.append(exclude_indices, bhk_df[bhk_df.price_per_sqft<(stats['mean'])].index.values)
    return df.drop(exclude_indices,axis='index')
```

```
df7= remove_bhk_outliers(df6)
df7.shape
```

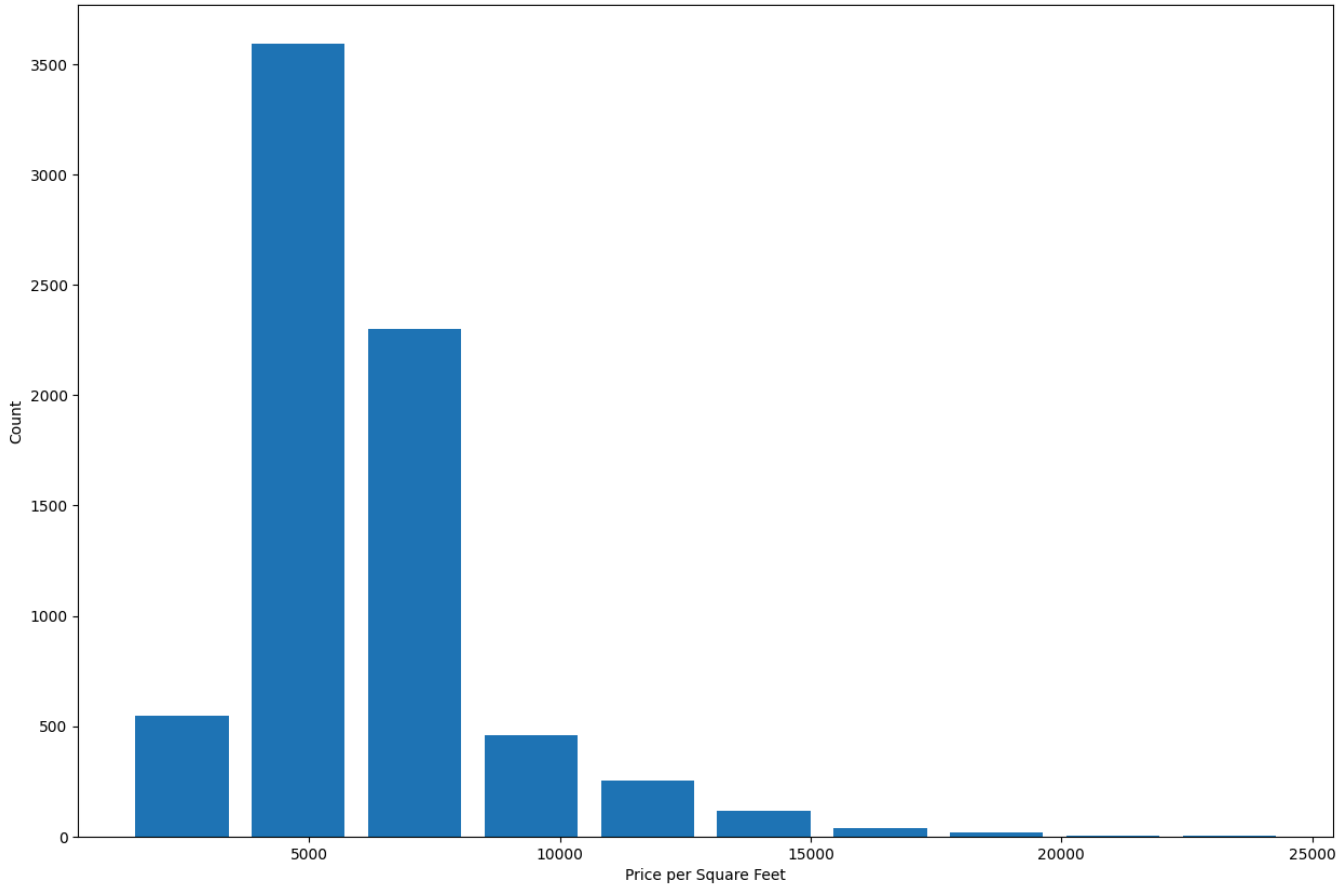
```
(7329, 7)
```

```
plot_scatter_chart(df7,"Hebbal")
```




```
plt.hist(df7.price_per_sqft, rwidth=0.8)    # visualization the price_per_sqft column
plt.xlabel('Price per Square Feet')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```




```
df7.bath.unique()
```

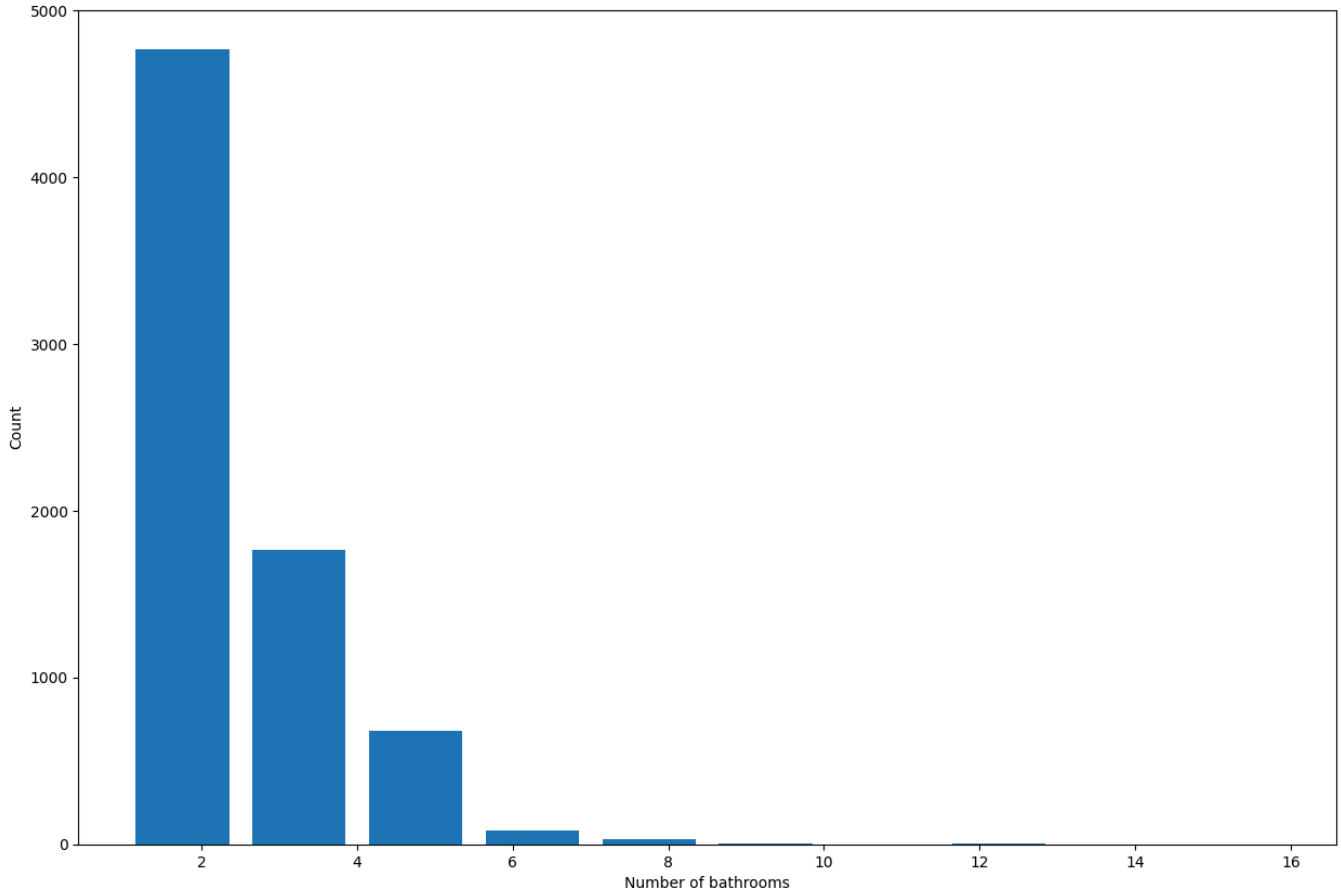
```
array([ 4.,  3.,  2.,  5.,  8.,  1.,  6.,  7.,  9., 12., 16., 13.])
```

```
df7[df7.bath>10]
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
5277	Neeladri Nagar	10 BHK	4000.0	12.0	160.0	10	4000.000000
8486	other	10 BHK	12000.0	12.0	525.0	10	4375.000000
8575	other	16 BHK	10000.0	16.0	550.0	16	5500.000000
9308	other	11 BHK	6000.0	12.0	150.0	11	2500.000000
9639	other	13 BHK	5425.0	13.0	275.0	13	5069.124424

```
plt.hist(df7.bath, rwidth=0.8)
plt.xlabel('Number of bathrooms')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```




```
df7[df7.bath > df7.bhk + 2]
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
1626	Chikkabanavar	4 Bedroom	2460.0	7.0	80.0	4	3252.032520
5238	Nagasandra	4 Bedroom	7000.0	8.0	450.0	4	6428.571429
6711	Thanisandra	3 BHK	1806.0	6.0	116.0	3	6423.034330
8411	other	6 BHK	11338.0	9.0	1000.0	6	8819.897689

```
df8 = df7[df7.bath < df7.bhk + 2]
df8.shape
```

```
(7251, 7)
```

```
df9 = df8.drop(["size","price_per_sqft"],axis='columns')    # removing or dropping 'size' and 'prize_per_sqft' as we don't require them
df9.head()
```

	location	total_sqft	bath	price	bhk
0	1st Block Jayanagar	2850.0	4.0	428.0	4
1	1st Block Jayanagar	1630.0	3.0	194.0	3
2	1st Block Jayanagar	1875.0	2.0	235.0	3
3	1st Block Jayanagar	1200.0	2.0	130.0	3
4	1st Block Jayanagar	1235.0	2.0	148.0	2

Next steps: Generate code with df9 View recommended plots

6. Creating Dummies

We will use pandas' get_dummies() to create dummies variables. It is used for data manipulation. It converts categorical data into dummy or indicator variables.

```
dummies = pd.get_dummies(df9.location,dtype=int)
dummies.head()
```

	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar	9th Phase JP Nagar	...	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	W
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 242 columns

```
df10= pd.concat([df9, dummies.drop('other', axis='columns')], axis='columns')    # joining the dummy values again with the dataset excep
df10.head()
```

	location	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	...	Vijayanagar	Vishveshwarya Layout	Vishwapri Layo
0	1st Block Jayanagar	2850.0	4.0	428.0	4	1	0	0	0	0	...	0	0	
1	1st Block Jayanagar	1630.0	3.0	194.0	3	1	0	0	0	0	...	0	0	
2	1st Block Jayanagar	1875.0	2.0	235.0	3	1	0	0	0	0	...	0	0	
3	1st Block Jayanagar	1200.0	2.0	130.0	3	1	0	0	0	0	...	0	0	
4	1st Block Jayanagar	1235.0	2.0	148.0	2	1	0	0	0	0	...	0	0	

5 rows × 246 columns

```
df11 = df10.drop('location', axis='columns')    # dropping original location as now we have dummmies in its place.
df11.head()
```

	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout
0	2850.0	4.0	428.0	4	1	0	0	0	0	0	...	0	0	0
1	1630.0	3.0	194.0	3	1	0	0	0	0	0	...	0	0	0
2	1875.0	2.0	235.0	3	1	0	0	0	0	0	...	0	0	0
3	1200.0	2.0	130.0	3	1	0	0	0	0	0	...	0	0	0
4	1235.0	2.0	148.0	2	1	0	0	0	0	0	...	0	0	0

5 rows × 245 columns

Start coding or generate with AI.

```
df11.shape
```

```
(7251, 245)
```

6a Storing the clean data in csv file

Double-click (or enter) to edit

```
df11.to_csv("Bengaluru_House_Data_clean.csv")
```

7. Splitting Data for Training and Testing

Before Training the model, it is required to split the data into train and test data. For this we will use, sklearn's train_test_split

```
X = df11.drop('price', axis='columns', )  # dropping price column as we don't want it in our train dataset
X.head()
```

	total_sqft	bath	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout
0	2850.0	4.0	4	1	0	0	0	0	0	0	...	0	0	0
1	1630.0	3.0	3	1	0	0	0	0	0	0	...	0	0	0
2	1875.0	2.0	3	1	0	0	0	0	0	0	...	0	0	0
3	1200.0	2.0	3	1	0	0	0	0	0	0	...	0	0	0
4	1235.0	2.0	2	1	0	0	0	0	0	0	...	0	0	0

5 rows × 244 columns

```
X.shape
```

```
(7251, 244)
```

```
y = df11.price           # taking the price column as our target to predict
```

```
y.shape
```

```
(7251,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)
```

```
X_train.shape
```

```
(5800, 244)
```

8. Training the Models

Preciesly, we will be trying more than one model, therefore, Training the Models

```
lr_clf = LinearRegression()     # first trying training with LinearRegression
lr_clf.fit(X_train, y_train)
lr_clf.score(X_test, y_test)
```

```
0.8452277697874376
```

```
cv = ShuffleSplit(n_splits = 5, test_size = 0.2, random_state = 0)     # ShuffleSplit is just a another type of splitting data
cross_val_score(LinearRegression(), X, y, cv=cv)
```

```
array([0.82430186, 0.77166234, 0.85089567, 0.80837764, 0.83653286])
```

Cross Validation is mainly used for the comparison of different models. For each model, you may get the average generalization error on the k validation sets. Then you will be able to choose the model with the lowest average generation error as your optimal model.

```
def find_best_model_using_gridsearchcv(X,y):
    '''
    Function to try different models at once of the data with different parameters to find the best ones.
    '''
    algos = {
        'linear_regression':{
            'model': LinearRegression(),
            'params':{
                'normalize':[True,False]
            }
        },
        'lasso':{
            'model': Lasso(),
            'params':{
                'alpha' : [1,2],
                'selection':['random','cyclic']
            }
        },
        'decision_tree':{
            'model': DecisionTreeRegressor(),
            'params':{
                'criterion':['mse','friedman_mse'],
                'splitter':['best','random']
            }
        }
    }
    scores = []
    cv = ShuffleSplit(n_splits=5,test_size=0.2,random_state=0)
    for algo_name, config in algos.items():
        gs = GridSearchCV(config['model'],config['params'], cv=cv, return_train_score=False)     # GridSearchCV is the main focus as it
        gs.fit(X,y)
        scores.append({
            'model': algo_name,
            'best_score':gs.best_score_,
            'best_params':gs.best_params_
        })
    return pd.DataFrame(scores,columns=['model','best_score','best_params'])     # At last binding the results of the models with best p
```

9. Prediction Time

Predicting the prices using LinearRegression in Lakhs.

```
def predict_price(location,sqft,bath,bhk):
    '''
    Function which helps to actually predict the prices.
    '''
    loc_index = np.where(X.columns==location)[0][0]     # np.where() function returns the indices of elements in an input array where th

    x = np.zeros(len(X.columns))    # np.zeros() function returns a new array of given shape and type, with zeros.
    x[0] = sqft
    x[1] = bath
    x[2] = bhk
    if loc_index >= 0:
        x[loc_index] = 1

    return lr_clf.predict([x])[0]
```

```
print(predict_price('1st Phase JP Nagar', 1000, 2, 3).round(3),'Lakhs')
```

```
81.726 Lakhs
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LinearRegression
  warnings.warn(
```

```
# def predict_price(location,sqft,bath,bhk)
predict_price('Kothanur', 1200, 2, 2).round(3)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LinearRegression
  warnings.warn(
53.096
```

```
# def predict_price(location,sqft,bath,bhk)
predict_price('Kothanur', 1400, 2, 3).round(3)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LinearRegression
  warnings.warn(
67.206
```

```
# def predict_price(location,sqft,bath,bhk)
predict_price('Kothanur', 1400, 2, 3).round(3)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LinearRegression
  warnings.warn(
67.206
```

```
# def predict_price(location,sqft,bath,bhk)
predict_price('Kothanur', 1400, 2, 10).round(3)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LinearRegression
  warnings.warn(
54.796
```

```
# def predict_price(location,sqft,bath,bhk)
predict_price('Kothanur', 1400, 2, 10).round(3)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LinearRegression
  warnings.warn(
54.796
```

```
# def predict_price(location,sqft,bath,bhk)
predict_price('Murugeshpalya', 1500, 2, 6).round(3)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LinearRegression
  warnings.warn(
68.965
```

10. Saving Model

For saving the model, we will be using pickle module and json module for saving the locations' names.

```
with open('BHP_model.pickle','wb') as f:
    pickle.dump(lr_clf,f)


columns = {
    'data_columns' : [col.lower() for col in X.columns]
}
with open('columns.json','w') as f:
    f.write(json.dumps(columns))
```

11. Loading Model

Loading the model, do prediction

```
# load the saved model file and use for prediction
# load_model = pickle.load('/content/BHP_model.pickle')
file_path = '/content/BHP_model.pickle'

with open(file_path , 'rb') as f:
    loadmodel = pickle.load(f)


def predict_price(location,sqft,bath,bhk):
    '''
    Function which helps to actually predict the prices.
    '''
    loc_index = np.where(X.columns==location)[0][0]     # np.where() function returns the indices of elements in an input array where th
    print (loc_index)
    x = np.zeros(len(X.columns))    # np.zeros() function returns a new array of given shape and type, with zeros.
    x[0] = sqft
    x[1] = bath
    x[2] = bhk
    if loc_index >= 0:
        x[loc_index] = 1

    return loadmodel.predict([x])[0]
```

```
def pprice( sqft,bath,bhk):
    '''
    Function which helps to actually predict the prices.
    '''
```