Data Science Regression Project: Predicting Home Prices in Bangalore

The Dataset was downloaded from https://www.kaggle.com/datasets/amitabhajoy/bengaluru-house-price-data

Setting Up the Data Analysis Environment

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib
matplotlib.rcParams['figure.figsize'] = (20,10)
```

Data Load: Load banglore home prices into a dataframe

In [2]:
```
df = pd.read_csv('Bengaluru_House_Data.csv')
df.head()
```

Out[2]:

	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

In [3]:
```
df.shape
```

Out[3]:
```
(13320, 9)
```

In [4]:
```
df.groupby('area_type')['area_type'].agg('count')
```

Out[4]:
```
area_type
Built-up  Area          2418
Carpet  Area              87
Plot  Area              2025
Super built-up  Area    8790
Name: area_type, dtype: int64
```

Drop features that are not required to build our model

In [5]:
```
df1 = df.drop(['area_type','society','balcony','availability'],axis='columns')
df1.head()
```

Out[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

Data Cleaning: Handle NA values

In [6]:
```
df1.isnull().sum()
```

Out[6]:
```
location       1
size          16
total_sqft     0
bath          73
price          0
dtype: int64
```

In [7]:
```
df2 = df1.dropna()
df2.isnull().sum()
```

Out[7]:
```
location      0
size          0
total_sqft    0
bath          0
price         0
dtype: int64
```

In [8]:
```
df2.shape
```

Out[8]:
```
(13246, 5)
```

In [9]:
```
df2['size'].unique()
```

Out[9]:
```
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)
```

Feature Engineering

Add new feature(integer) for bhk (Bedrooms Hall Kitchen)

In [10]:
```
df2['bhk'] = df2['size'].apply(lambda x: int(x.split(' ')[0]))
```

```
C:\Users\47455\AppData\Local\Temp\ipykernel_6512\1142257054.py:1: SettingWithCopyWarnin
g:
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-a-copy
  df2['bhk'] = df2['size'].apply(lambda x: int(x.split(' ')[0]))
```

In [11]:
```
df2.head()
```

Out[11]:

	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

In [12]:
```
df2['bhk'].unique()
```

Out[12]:
```
array([ 2,  4,  3,  6,  1,  8,  7,  5, 11,  9, 27, 10, 19, 16, 43, 14, 12,
       13, 18], dtype=int64)
```

In [13]:
```
df2[df2.bhk>20]
```

Out[13]:

	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

In [14]:
```
df2.total_sqft.unique()
```

Out[14]:
```
array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)
```

Explore total_sqft feature

In [15]:
```
def is_float(x):
    try:
        float(x)
    except:
        return False
    return True
```

In [16]:
```
df2[~df2['total_sqft'].apply(is_float)].head()
```

Out[16]:

	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

In [17]:
```
def convert_sqft_to_num(x):
    tokens = x.split('-')
    if len(tokens) == 2:
        return (float(tokens[0])+float(tokens[1]))/2
    try:
        return float(x)
    except:
        return None
```

In [18]:
```
convert_sqft_to_num('2166')
```
Out[18]:
```
2166.0
```
In [19]:
```
convert_sqft_to_num('2100 - 2850')
```
Out[19]:
```
2475.0
```
In [20]:
```
df3 = df2.copy()
df3['total_sqft'] = df3['total_sqft'].apply(convert_sqft_to_num)
df3.head(3)
```
Out[20]:

	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

Feature Engineering

Add new feature called price per square feet

In [21]:
```
df4 = df3.copy()
df4['price_per_sqft'] = df4['price']*100000/df4['total_sqft']
df4.head()
```
Out[21]:

	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

In [22]:
```
df4.location.unique()
```
Out[22]:
```
array(['Electronic City Phase II', 'Chikka Tirupathi', 'Uttarahalli', ...,
       '12th cross srinivas nagar banshankari 3rd stage',
       'Havanur extension', 'Abshot Layout'], dtype=object)
```
In [23]:
```
len(df4.location.unique())
```
Out[23]:
```
1304
```

Examine locations which is a categorical variable. We need to apply dimensionality reduction technique here to reduce number of locations

In [24]:
```
df4.location = df4.location.apply(lambda x: x.strip())
location_stats = df4.groupby('location')['location'].agg('count').sort_values(ascending=
location_stats
```
Out[24]:
```
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
```

In [25]:
```
len(location_stats[location_stats<=10])
```

Out[25]:
```
1052
```

Dimensionality Reduction

Any location having less than 10 data points should be tagged as "other" location. This way number of categories can be reduced by huge amount. Later on when we do one hot encoding, it will help us with having fewer dummy columns

In [26]:
```
location_stats_less_than_10 = location_stats[location_stats<=10]
location_stats_less_than_10
```

Out[26]:
```
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
```

In [27]:
```
len(df4.location.unique())
```

Out[27]:
```
1293
```

In [28]:
```
df4.location = df4.location.apply(lambda x: 'other' if x in location_stats_less_than_10
len(df4.location.unique())
```

Out[28]:
```
242
```

In [29]:
```
df4.head(5)
```

Out[29]:

	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

Outlier Removal Using Business Logic

In [30]:
```
df4[df4.total_sqft/df4.bhk<300].head()
```

Out[30]:

	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

Check above data points. We have 6 bhk apartment with 1020 sqft. Another one is 8 bhk and total sqft is 600. These are clear data errors that can be removed safely

In [31]:
```
df4.shape
```

Out[31]:
```
(13246, 7)
```

In [32]:
```
df5 = df4[~(df4.total_sqft/df4.bhk<300)]
df5.shape
```

Out[32]:
```
(12502, 7)
```

Outlier Removal Using Standard Deviation and Mean

In [33]:
```
df5.price_per_sqft.describe()
```

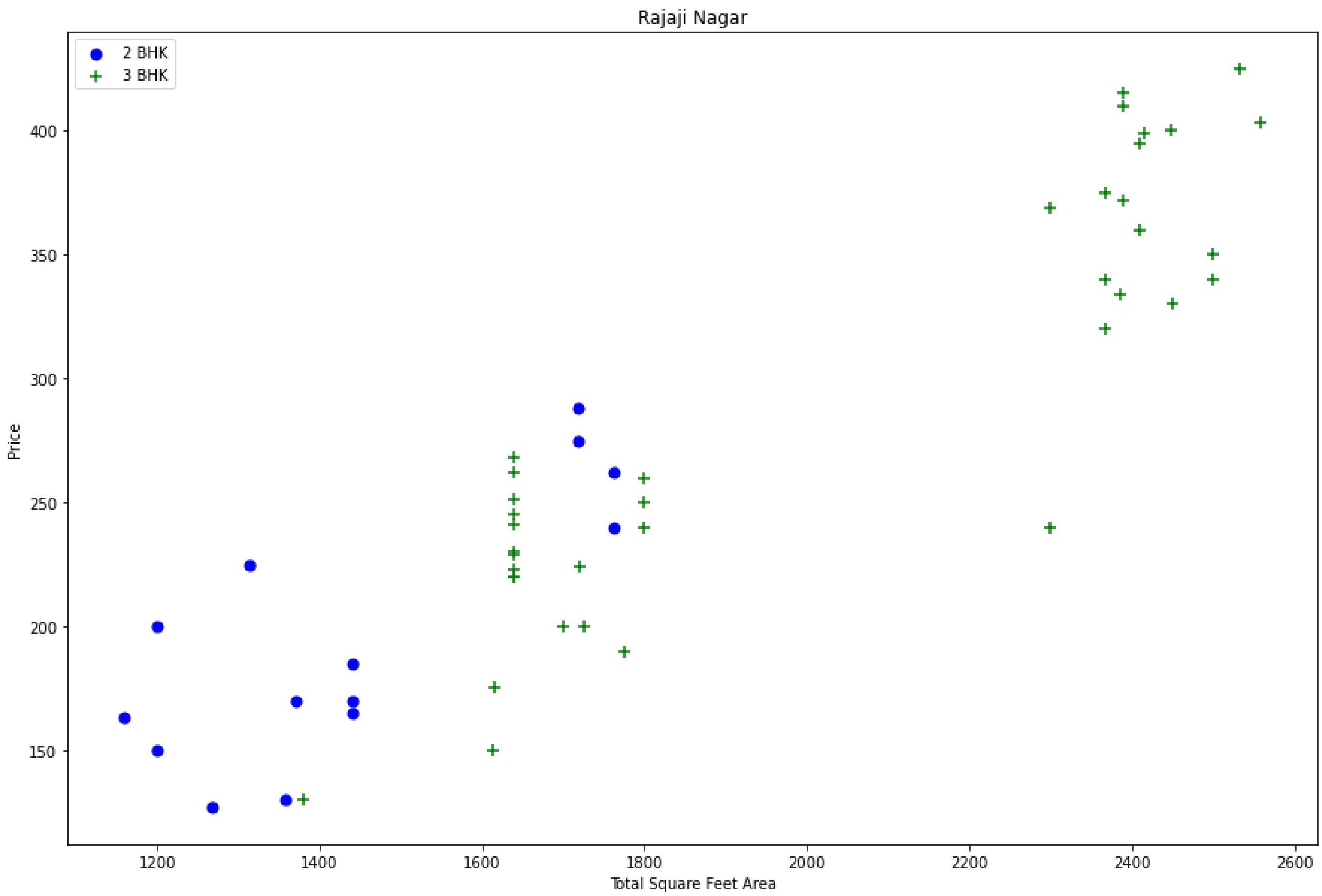
Out[33]:
```
count     12456.000000
mean       6308.502826
std        4168.127339
min         267.829813
25%        4210.526316
50%        5294.117647
75%        6916.666667
max      176470.588235
Name: price_per_sqft, dtype: float64
```

In [34]:
```
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
```

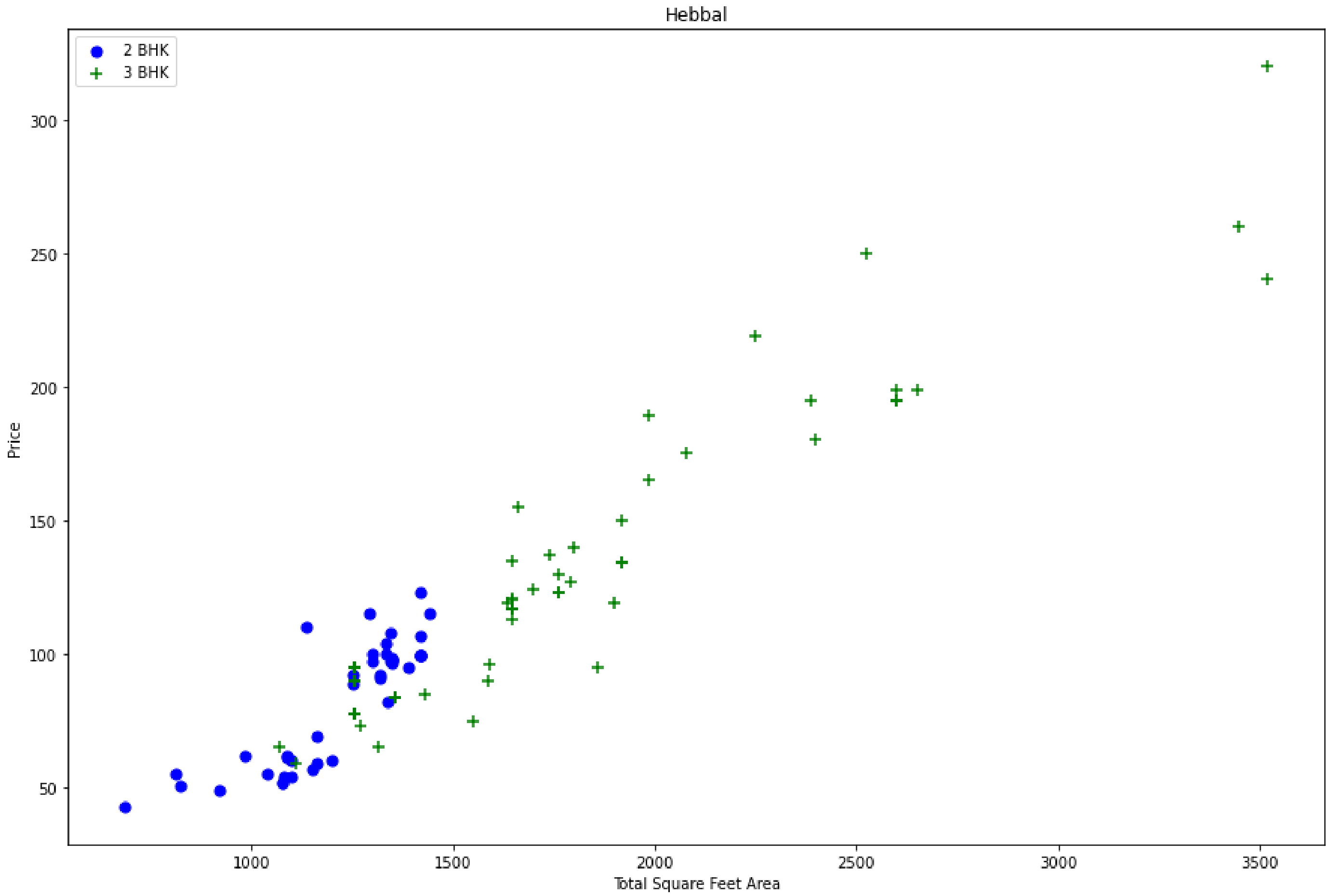
Out[34]:
```
(10241, 7)
```

Let's check if for a given location how does the 2 BHK and 3 BHK property prices look like

In [35]:
```
def plot_scatter_chart(df,location):
    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')
    plt.title(location)
    plt.legend()
plot_scatter_chart(df6, 'Rajaji Nagar')
```



In [36]:
```
plot_scatter_chart(df6, 'Hebbal')
```



We should also remove properties where for same location, the price of (for example) 3 bedroom apartment is less than 2 bedroom apartment (with same square ft area). What we will do is for a given location, we will build a dictionary of stats per bhk, i.e.

{ '1' : { 'mean': 4000, 'std: 2000, 'count': 34 }, '2' : { 'mean': 4300, 'std: 2300, 'count': 22 }, }

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

In [37]:

```
def remove_bhk_outliers(df):
    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_sqf
    return df.drop(exclude_indices,axis='index')
df7 = remove_bhk_outliers(df6)
df7.shape
```

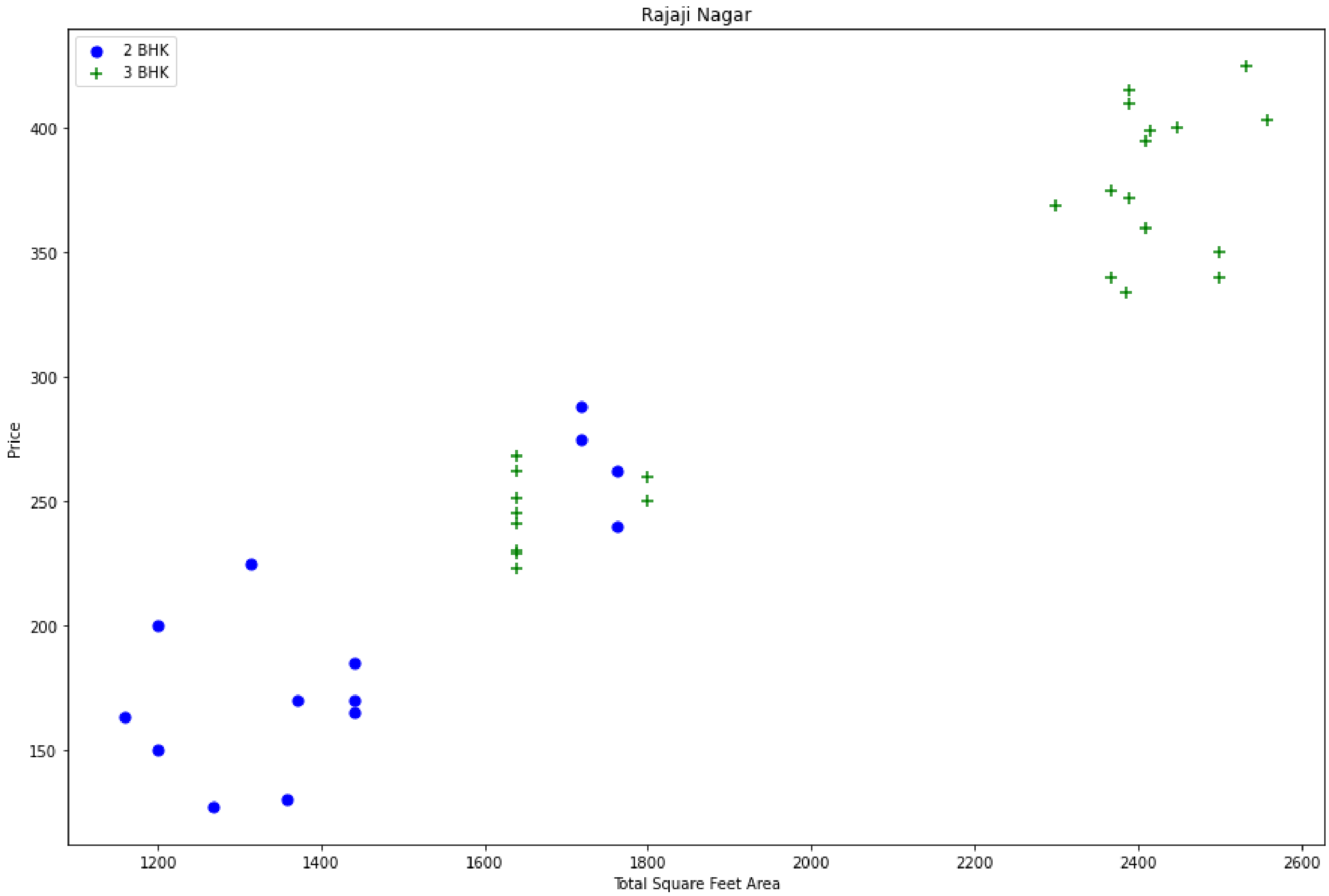
Out[37]:

```
(7329, 7)
```

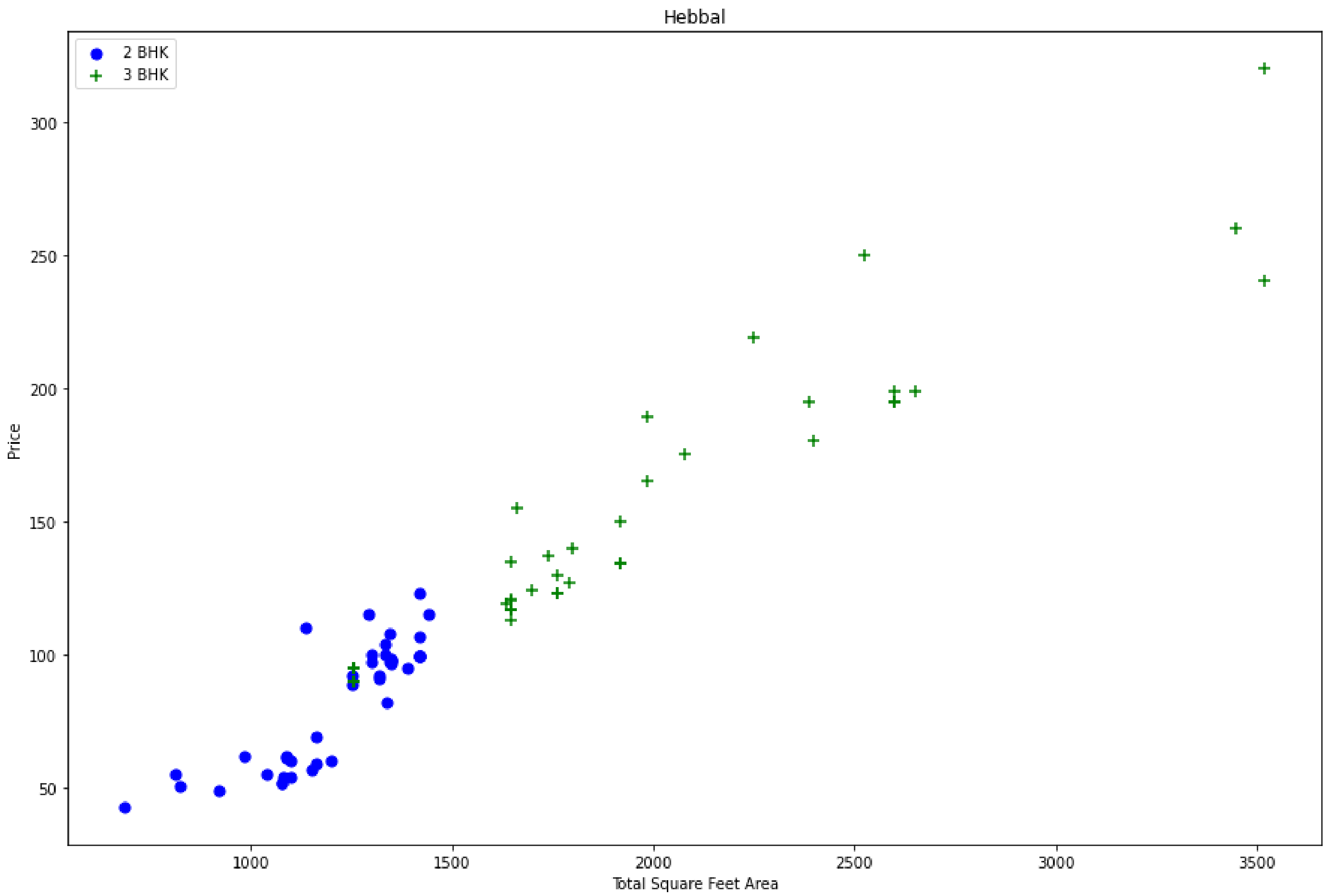
Plot some scatter chart again to visualize price_per_sqft for 2 BHK and 3 BHK properties

In [38]:

```
plot_scatter_chart(df7,"Rajaji Nagar")
```



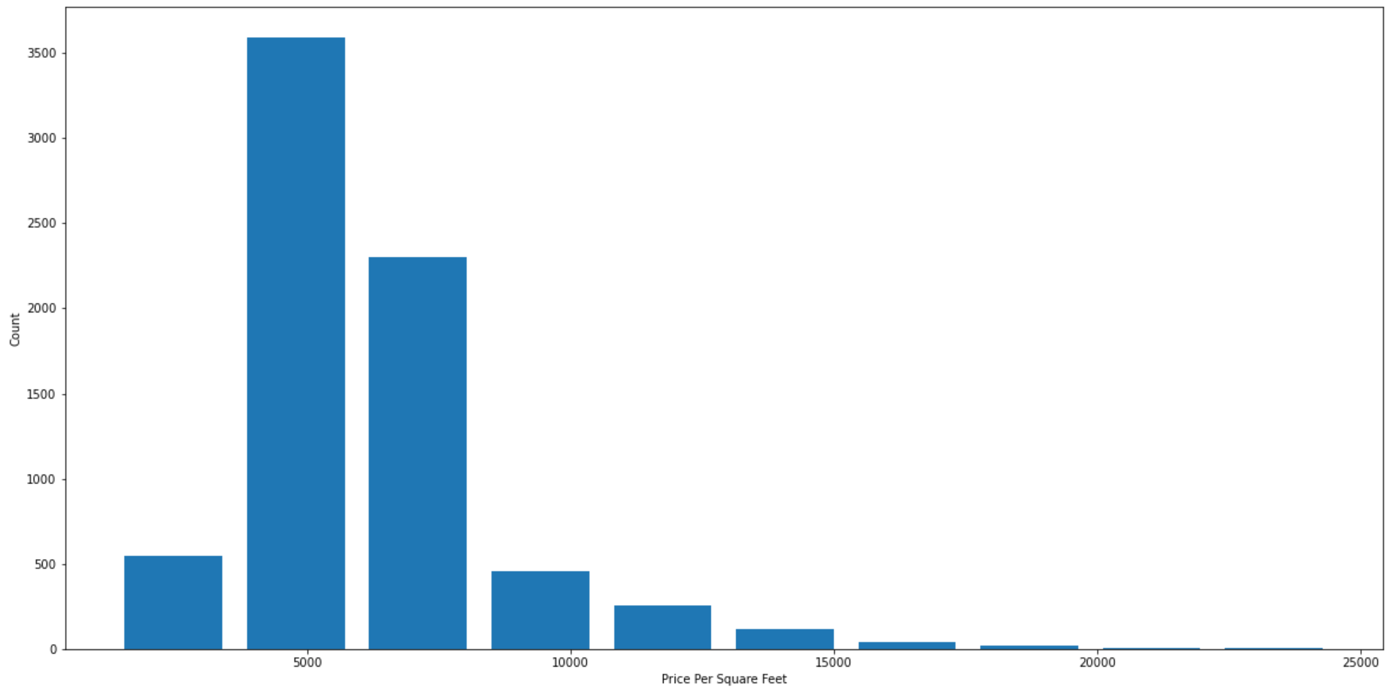
In [39]:
```
plot_scatter_chart(df7,'Hebbal')
```



In [40]:
```
plt.figure(figsize=(20,10))
plt.hist(df7.price_per_sqft,rwidth = 0.8)
```

```
plt.xlabel('Price Per Square Feet')
plt.ylabel('Count')
```

Out[40]: Text(0, 0.5, 'Count')



Outlier Removal Using Bathrooms Feature

In [41]:
```
df7.bath.unique()
```

Out[41]: array([4., 3., 2., 5., 8., 1., 6., 7., 9., 12., 16., 13.])

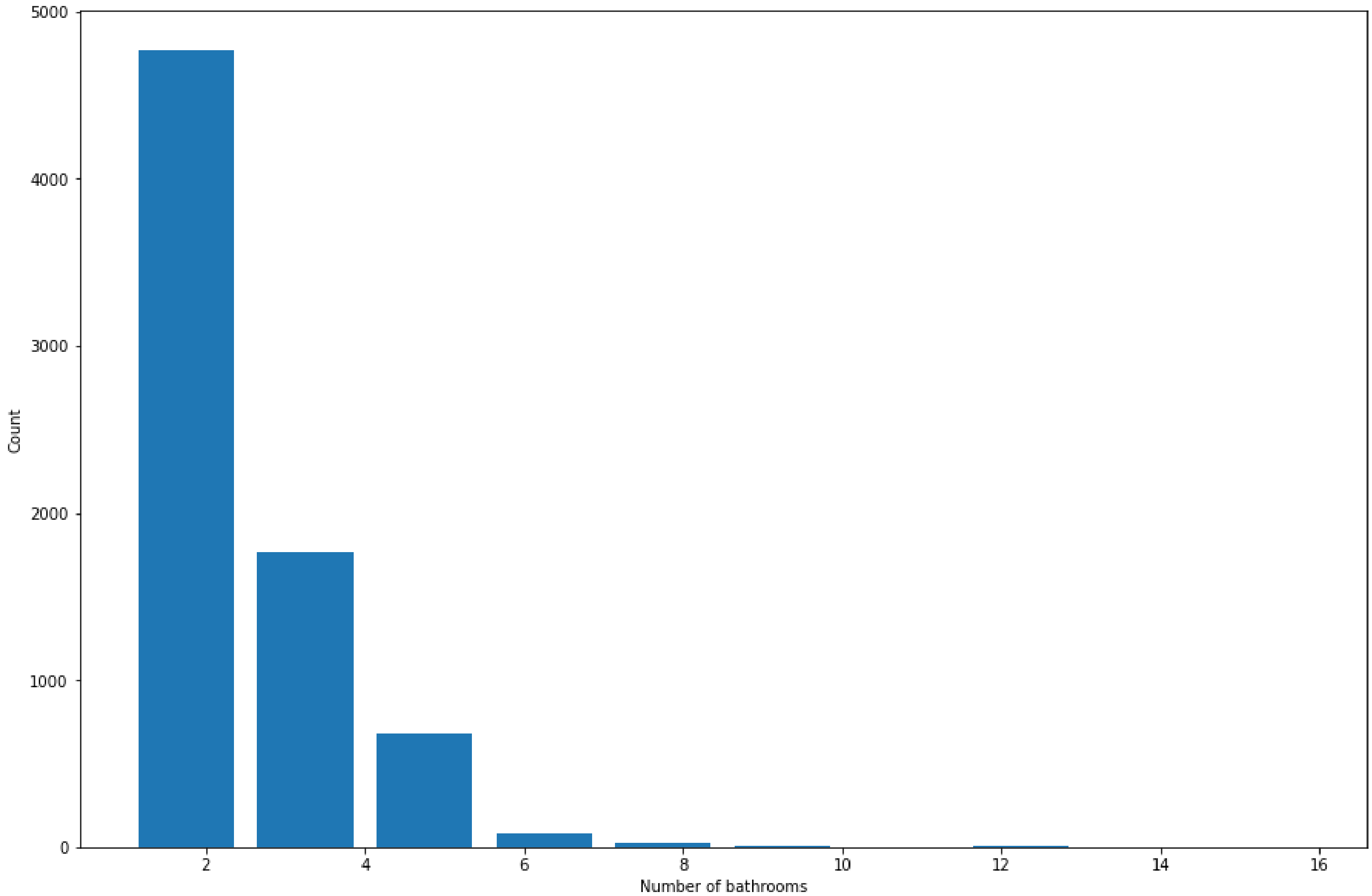
In [42]:
```
df7[df7.bath>10]
```

Out[42]:

	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

In [43]:
```
plt.hist(df7.bath,rwidth=0.8)
plt.xlabel('Number of bathrooms')
plt.ylabel('Count')
```

Out[43]: Text(0, 0.5, 'Count')



In [44]:
```
df7[df7.bath>10]
```

Out[44]:

	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

It is unusual to have 2 more bathrooms than number of bedrooms in a home

In [45]:
```
df7[df7.bath>df7.bhk+2]
```

Out[45]:

	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

In [46]:
```
df8=df7[df7.bath<df7.bhk+2]
df8.shape
```

Out[46]:
```
(7251, 7)
```

In [47]:
```
df9 = df8.drop(['size','price_per_sqft'],axis='columns')
```

```
df9.head(5)
```

Out[47]:

	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

Use one Hot Encoding For Location

In [48]:

```
dummies=pd.get_dummies(df9.location)
dummies.head()
```

Out[48]:

	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	Vis
0	1	0	0	0	0	0	0	0	0	0	...	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	
3	1	0	0	0	0	0	0	0	0	0	...	0	
4	1	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 242 columns

In [49]:

```
df10 = pd.concat([df9,dummies.drop('other',axis='columns')],axis='columns')
df10.head(3)
```

Out[49]:

	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	Vish
0	1st Block Jayanagar	2850.0	4.0	428.0	4	1	0	0	0	0	...	0	
1	1st Block Jayanagar	1630.0	3.0	194.0	3	1	0	0	0	0	...	0	
2	1st Block Jayanagar	1875.0	2.0	235.0	3	1	0	0	0	0	...	0	

3 rows × 246 columns

In [50]:

```
df11 = df10.drop('location',axis='columns')
df11.head(2)
```

Out[50]:

	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	Vishves

											...	
0	2850.0	4.0	428.0	4	1	0	0	0	0	0	...	0
1	1630.0	3.0	194.0	3	1	0	0	0	0	0	...	0

2 rows × 245 columns

Build a Model Now...

In [51]:
```
df11.shape
```

Out[51]:
```
(7251, 245)
```

In [52]:
```
X = df11.drop('price',axis='columns')
X.head()
```

Out[52]:

	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	Vishve
0	2850.0	4.0	4	1	0	0	0	0	0	0	...	0	
1	1630.0	3.0	3	1	0	0	0	0	0	0	...	0	
2	1875.0	2.0	3	1	0	0	0	0	0	0	...	0	
3	1200.0	2.0	3	1	0	0	0	0	0	0	...	0	
4	1235.0	2.0	2	1	0	0	0	0	0	0	...	0	

5 rows × 244 columns

In [53]:
```
y = df11.price
y.head()
```

Out[53]:
```
0    428.0
1    194.0
2    235.0
3    130.0
4    148.0
Name: price, dtype: float64
```

In [54]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=10)
```

In [55]:
```
from sklearn.linear_model import LinearRegression
lr_clf = LinearRegression()
lr_clf.fit(x_train,y_train)
lr_clf.score(x_test,y_test)
```

Out[55]:
```
0.845227769787429
```

Use K Fold cross validation to measure accuracy of our LinearRegression model

In [56]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
```

```
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
cross_val_score(LinearRegression(),X,y,cv=cv)
```

Out[56]:

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

We can see that in 5 iterations we get a score above 80% all the time. This is pretty good but we want to test few other algorithms for regression to see if we can get even better score. We will use GridSearchCV for this purpose

Find best model using GridSearchCV

In [57]:

```
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor

def find_best_model_using_gridsearchcv(X,y):
    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=
        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'])

find_best_model_using_gridsearchcv(X,y)
```

```
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing sta
ge. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline
```

```
model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to
each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing sta
ge. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to
each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing sta
ge. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to
each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing sta
ge. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to
each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing sta
ge. To reproduce the previous behavior:
```

```
from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to
each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:148: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2. Please leave
the normalize parameter to its default value to silence this warning. The default behavi
or of this estimator is to not do any normalization. If normalization is needed please u
se sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:148: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2. Please leave
the normalize parameter to its default value to silence this warning. The default behavi
or of this estimator is to not do any normalization. If normalization is needed please u
se sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:148: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2. Please leave
the normalize parameter to its default value to silence this warning. The default behavi
or of this estimator is to not do any normalization. If normalization is needed please u
se sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:148: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2. Please leave
the normalize parameter to its default value to silence this warning. The default behavi
or of this estimator is to not do any normalization. If normalization is needed please u
se sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:148: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2. Please leave
the normalize parameter to its default value to silence this warning. The default behavi
or of this estimator is to not do any normalization. If normalization is needed please u
se sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarn
ing: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing sta
ge. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to
each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning:
Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterio
n='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning:
Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterio
```

```
n='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning:
Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterio
n='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning:
Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterio
n='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning:
Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterio
n='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning:
Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterio
n='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning:
Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterio
n='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning:
Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterio
n='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning:
Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterio
n='squared_error'` which is equivalent.
  warnings.warn(
C:\Users\47455\anaconda3\lib\site-packages\sklearn\tree\_classes.py:359: FutureWarning:
Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterio
n='squared_error'` which is equivalent.
  warnings.warn(
```

Out[57]:

	model	best_score	best_params
0	linear_regression	0.818354	{'normalize': True}
1	lasso	0.687478	{'alpha': 2, 'selection': 'random'}
2	decision_tree	0.721802	{'criterion': 'friedman_mse', 'splitter': 'ran...

Based on above results we can say that LinearRegression gives the best score. Hence we will use that.

Test the model for few properties

In [58]:

```python
def predict_price(location,sqft,bath,bhk):
    loc_index = np.where(X.columns==location)[0][0]

    x = np.zeros(len(X.columns))
    x[0] = sqft
    x[1] = bath
    x[2] = bhk
    if loc_index >= 0:
        x[loc_index] = 1
```

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

In [59]:
```
predict_price('1st Phase JP Nagar',1000, 2, 2)
```

```
C:\Users\47455\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[59]:
```
83.49904677172415
```

In [60]:
```
predict_price('1st Phase JP Nagar',1000, 3, 3)
```

```
C:\Users\47455\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[60]:
```
86.80519395199
```

In [61]:
```
predict_price('Indira Nagar',1000, 2, 2)
```

```
C:\Users\47455\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[61]:
```
181.27815484006965
```

In [62]:
```
predict_price('Indira Nagar',1000, 3, 3)
```

```
C:\Users\47455\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[62]:
```
184.58430202033549
```