

PROJECT DOCUMENTATION

REAL ESTATE PRICE PREDICTION WEB APPLICATION

Submitted by:

Herbert Ifeanyichukwu Ukaegbu (Coordinator), Milene Noumbissi Yimdjo

Client: Emmanuel Ajaero

Supervisor: Prof. Martin Knahl

Winter Semester 2022/2023

AIM

Creating a Machine Learning model to predict the home prices in Bangalore, India and create a single page website which will provide the front end to access our model for predictions.

We use the dataset from Kaggle.com.

Below are data science concepts used in this project:

Data loading and cleaning

Outlier detection and removal

Feature engineering

Dimensionality reduction

Gridsearchcv for hyperparameter tuning

K fold cross validation

Technology and tools used in this project:

Python

NumPy and Pandas for data cleaning

Matplotlib for data visualization

Sklearn for model building

Python flask for http server

HTML/CSS/JavaScript for UI

STEPS

We first build a model using sklearn and linear regression using banglore home prices dataset from kaggle.com.

Second step was to write a python flask server that uses the saved model to serve http requests.

Third component is the website built in html, CSS and JavaScript that allows user to enter home square ft area, bedrooms etc. and it will call python flask server to retrieve the predicted price.

Step#1: Import the required libraries

Step#2: Load the data

Step#3: Understand the data

-drop unnecessary columns

Step#4: Data Cleaning

- Check for NA values
- Verify unique values of each column
- Make sure values are correct (eg. 23 BHK home with 2000 Sqrft size is wrong)
- Feature Engineering
- Dimensionality Reduction
- Outlier removal using domain knowledge (2bhk price < 3bhk price, size per bhk >= 300 sqft)
- Outlier removal using standard deviation and mean
- One Hot encoding

Step#5: Build Machine Learning Model

Step#6: Testing The model

Step#7: Export the model

Step#8: Export any other important info

DATA

Data provided by client

	A	B	C	D	E	F	G	H	I
1	area_type	availability	location	size	society	total_sqft	bath	balcony	price
2	Super built-up Area	19-Dec	Electronic City Phase 2	2 BHK	Coomee	1056	2	1	39.07
3	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5	3	120
4	Built-up Area	Ready To Move	Uttarahalli	3 BHK		1440	2	3	62
5	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3	1	95
6	Super built-up Area	Ready To Move	Kothanur	2 BHK		1200	2	1	51
7	Super built-up Area	Ready To Move	Whitefield	2 BHK	DuenaTa	1170	2	1	38
8	Super built-up Area	18-May	Old Airport Road	4 BHK	Jaades	2732	4		204
9	Super built-up Area	Ready To Move	Rajaji Nagar	4 BHK	Brway G	3300	4		600
10	Super built-up Area	Ready To Move	Marathahalli	3 BHK		1310	3	1	63.25
11	Plot Area	Ready To Move	Gandhi Bazar	6 Bedroom		1020	6		370
12	Super built-up Area	18. Feb	Whitefield	3 BHK		1800	2	2	70
13	Plot Area	Ready To Move	Whitefield	4 Bedroom	Prrry M	2785	5	3	295
14	Super built-up Area	Ready To Move	7th Phase JP Nagar	2 BHK	Shncyas	1000	2	1	38
15	Built-up Area	Ready To Move	Gottigere	2 BHK		1100	2	2	40
16	Plot Area	Ready To Move	Sarjapur	3 Bedroom	Skityer	2250	3	2	148
17	Super built-up Area	Ready To Move	Mysore Road	2 BHK	PrntaEn	1175	2	2	73.5
18	Super built-up Area	Ready To Move	Bisuvanahalli	3 BHK	Prityel	1180	3	2	48
19	Super built-up Area	Ready To Move	Raja Rajeshwari Nagar	3 BHK	GrrvaGr	1540	3	3	60
20	Super built-up Area	Ready To Move	Ramakrishnappa Layout	3 BHK	PeBayle	2770	4	2	290
21	Super built-up Area	Ready To Move	Manayata Tech Park	2 BHK		1100	2	2	48
22	Built-up Area	Ready To Move	Kengeri	1 BHK		600	1	1	15
23	Super built-up Area	19-Dec	Binny Pete	3 BHK	She 2rk	1755	3	1	122
24	Plot Area	Ready To Move	Thanisandra	4 Bedroom	Soitya	2800	5	2	380
25	Super built-up Area	Ready To Move	Bellandur	3 BHK		1767	3	1	103
26	Super built-up Area	18. Nov	Thanisandra	1 RK	Bhe 2ko	510	1	0	25.25
27	Super built-up Area	18-May	Mangammanapalya	3 BHK		1250	3	2	56

Step#1: Importing the required libraries

```
In [4]: import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import matplotlib
matplotlib.rcParams["figure.figsize"] = (20, 10)
pd.options.mode.chained_assignment = None
```

Step#2: Load the data

Loading the Bangalore home prices into the data frame

```
In [5]: df1 = pd.read_csv("Bengaluru_House_Data.csv")
df1.head()
```

```
Out[5]:
```

	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

Step#3: Understanding the data

We try to understand the data to finalize the columns to work with and drop the rest of them

- I. Getting to know the number of column and row: we have 13320 columns and 9 rows

```
In [6]: df1.shape
```

```
Out[6]: (13320, 9)
```

- II. Getting to know all columns names

For each column we have area type, availability, location size, society, total square feet (sqft), bath, balcony, and price

```
In [7]: df1.columns
```

```
Out[7]: Index(['area_type', 'availability', 'location', 'size', 'society',  
              'total_sqft', 'bath', 'balcony', 'price'],  
              dtype='object')
```

III. Checking the unique values “area_type” column

```
In [8]: df1['area_type'].unique()
```

```
Out[8]: array(['Super built-up Area', 'Plot Area', 'Built-up Area',  
              'Carpet Area'], dtype=object)
```

IV. Counting the training example of for each area type

```
In [9]: df1['area_type'].value_counts()
```

```
Out[9]: Super built-up Area    8790  
        Built-up Area         2418  
        Plot Area             2025  
        Carpet Area           87  
        Name: area_type, dtype: int64
```

V. Dropping columns

To build our model we drop certain columns which are not required to avoid an overfitting problem. Some unnecessary columns are dropped because an over-fitted prediction model will make the prediction look good only on the current data. However, the result will not look good if using other data sources. These features are area type, availability, society, and balcony. When removing these “NOT REQUIRED” columns, we are left with 13320 rows and 5 columns.

To be noted: Every time we make change in dataset, we store it in new data frame (that is df1 to df2)

```
In [16]: df2 = df1.drop(['area_type', 'society', 'balcony', 'availability'], axis='columns')  
         print('Rows and columns are = ', df2.shape)  
         df2.head()
```

```
Rows and columns are = (13320, 5)
```

```
Out[16]:
```

	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

Step#4: Data Cleaning

- We Check for NA (Not Applicable) values
- Verify unique values of each column
- We make sure values are correct (e.g., 23 BHK home with 2000 Sqft size seems wrong)

Handling Null values

We get the sum of all NA values form dataset

```
In [11]: df2.isnull().sum()
```

```
Out[11]: location      1  
size      16  
total_sqft    0  
bath      73  
price      0  
dtype: int64
```

Since null values as compared to total training examples (13320) which are few, we can safely drop those examples and data stored into a new dataset (that is from df2 to df3)

```
In [12]: df2.shape
```

```
Out[12]: (13320, 5)
```

```
In [13]: df3 = df2.dropna()  
df3.isnull().sum()
```

```
Out[13]: location      0  
size      0  
total_sqft    0  
bath      0  
price      0  
dtype: int64
```

Since all training examples containing null values are dropped, we check the shape of the dataset we are left with 13246 columns and 5 rows.

```
In [14]: df3.shape
```

```
Out[14]: (13246, 5)
```

Feature Engineering

- o The “size” column contains the size of house in terms of BHK (Bedroom Hall Kitchen)
- o To simplify it we create a new column by the name “BHK” and add only numeric value of how many BHK’s

Reading the size column

```
In [17]: df3['size'].unique()
```

```
Out[17]: 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)
```

We copy the data into a new data frame (that is df3 to df4) and use the lambda function to get the BHK numeric values

```
In [18]: df4 = df3.copy()  
df4['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0]))  
df4.bhk.unique()
```

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

From the above data we can see that there is home with up to 43 BHK's in Bangalore.

Getting to know the training example with data more than 20 BHK's

```
In [19]: df4[df4.bhk >20]
```

```
Out[19]:
```

	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

To be noted: above 43 BHK are only 2400 sqft only, we remove this data error later. First, we will clean the "total_sqft" column.

The next step is to check unique value in "total_sqft" column

```
In [20]: df4.total_sqft.unique()
```

```
Out[20]: array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],  
               dtype=object)
```

From above, there are few records with range of area like "1133 - 1384". We write a function to identify such values

```
In [21]: def is_float(x):
        try:
            float(x)
        except:
            return False

        return True
```

Testing the function

```
In [22]: print('is this (123) float value = %s' % (is_float(123)))
        print('is this (1133 - 1384) float value = %s' % (is_float('1133 - 1384')))
```

is this (123) float value = True
is this (1133 - 1384) float value = False

We then apply this function to “total_sqft” column. The function will show training examples where “total_sqft” is not a float.

```
In [23]: df4[~df4['total_sqft'].apply(is_float)].head(10)
```

Out[23]:

	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

Above shows that total_sqft can be a range (example 2100 - 2850), so we write a function to get the value from a range. There are few values like “34.46Sq. Meter” and “4125Perch” which can be converted to square ft using unit conversion are going to be ignored to keep things simple.


```
In [24]: def convert_range_to_sqft(x):
        try:
            tokens = x.split('-')

            if len(tokens) == 2:
                return (float(tokens[0]) + float(tokens[1]))/2
            else:
                return float(x)
        except:
            return None
```

Testing the function (convert_range_to_sqft())

```
In [26]: print('Return value for i/p 12345 = %s' % (convert_range_to_sqft('12345')))
        print('Return value for i/p 1133 - 1384 = %s' % (convert_range_to_sqft('1133 - 1384')))
        print('Return value for i/p 34.46Sq. Meter = %s' % (convert_range_to_sqft('34.46Sq. Meter')))
```

```
Return value for i/p 12345 = 12345.0
Return value for i/p 1133 - 1384 = 1258.5
Return value for i/p 34.46Sq. Meter = None
```

We then copy the df4 dataset into the new df5 dataset and then we apply the function for the total_sqft function.

```
In [27]: df5 = df4.copy()

        df5.total_sqft = df4.total_sqft.apply(convert_range_to_sqft)
        df5.head()
```

Out[27]:

	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

Since the function will return null values for values like 34.26Sq. Meter, we check for any null values in it

```
In [28]: df5.total_sqft.isnull().sum()
```

```
Out[28]: 46
```

After the check in we then drop the null training set from total_sqft, and store it in a new dataset df6

```
In [29]: df6 = df5.dropna()  
df6.total_sqft.isnull().sum()
```

```
Out[29]: 0
```

We then cross check the values of “total_sqft”. From previous training set df4 we had a range (example 2100 – 2850), from new training set df6 we do not have a range no more (example 2475.0)

```
In [30]: print('total_sqft value for 30th training set in df4 = %s' % (df4.total_sqft[30]))  
print('total_sqft value for 30th training set in df6 = %s' % (df6.total_sqft[30]))  
  
total_sqft value for 30th training set in df4 = 2100 - 2850  
total_sqft value for 30th training set in df6 = 2475.0
```

Feature Engineering – Price Column

- o The 'price' column contains the price of house in lacka (1 lakh = 100000)
- o Price per square fit is important parameter in house prices.
- o So, we create a new column by the name 'price_per_sqft' and add price per sqft in it.
*formula = (price * 100000)/total_sqft*

The training set is stored into a new dataset df7

```
In [31]: df7 = df6.copy()  
  
df7['price_per_sqft'] = (df6['price'] * 100000)/df6['total_sqft']  
df7.head()
```

```
Out[31]:
```

	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

Statistics of the price_per_sqft

```
In [32]: df7_stats = df7['price_per_sqft'].describe()
df7_stats
```

```
Out[32]: count    1.320000e+04
mean      7.920759e+03
std       1.067272e+05
min       2.678298e+02
25%      4.267701e+03
50%      5.438331e+03
75%      7.317073e+03
max      1.200000e+07
Name: price_per_sqft, dtype: float64
```

Dimensionality Reduction

- o Dimensionality reduction is the process of reducing the dimension (or number of random variables) of our feature set.
- o In our dataset 'location' is categorical variable with 1287 categories.
- o Before using One Hot Encoding to create dummy variables, we must reduce the number of categories by using dimensionality reduction so that we will get a smaller number of dummy variables.
- o Our criteria for dimensionality reduction for 'location' is to use 'other' location for any location having less than 10 data points.

First, we trim the location values

```
In [33]: df7.location = df7.location.apply(lambda x: x.strip())
df7.head()
```

```
Out[33]:
```

	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

We then get the count of each location

```
In [34]: location_stats = df7.location.value_counts(ascending=False)
location_stats
```

```
Out[34]: Whitefield          533
Sarjapur Road             392
Electronic City           304
Kanakpura Road            264
Thanisandra               235
...
Rajanna Layout            1
Subramanyanagar           1
Lakshmipura Vidyaanyapura 1
Malur Hosur Road          1
Abshot Layout             1
Name: location, Length: 1287, dtype: int64
```

We later read the total number of unique location categories

```
In [35]: len(location_stats)
```

```
Out[35]: 1287
```

Next, we are going assign a category 'other' for every location where total datapoints are less than 10

```
In [36]: print('Total no of locations where data points are more than 10 = %s' % (len(location_stats[location_stats > 10])))
print('Total no of locations where data points are less than 10 = %s' % (len(location_stats[location_stats <= 10])))
```

```
Total no of locations where data points are more than 10 = 240
Total no of locations where data points are less than 10 = 1047
```

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, when we do one hot encoding, it will help us with having fewer dummy column.

Location with less than equal to 10

```
In [37]: location_stats_less_than_10 = location_stats[location_stats <= 10]
location_stats_less_than_10
```

```
Out[37]: BTM 1st Stage          10
Gunjur Palya                   10
Nagappa Reddy Layout           10
Sector 1 HSR Layout            10
Thyagaraja Nagar               10
..
Rajanna Layout                 1
Subramanyanagar                1
Lakshmipura Vidyaanyapura       1
Malur Hosur Road               1
Abshot Layout                  1
Name: location, Length: 1047, dtype: int64
```

Using lambda function assign the “other” type to every element in “location_stats_less_than_10” and the data is copied into the new dataset df8

```
In [38]: df8 = df7.copy()

df8.location = df7.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)
len(df8.location.unique())

Out[38]: 241
```

Since 1047 location with less than 10 data points are converted to one category 'other' Total no of unique location categories are = 240 +1 = 241

```
In [39]: df8.head(10)
```

```
Out[39]:
```

	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

Outlier Removal

- o An outlier is an observation that is unlike the other observations. It is rare, or distinct, or does not fit in some way.
- o Outliers are the data points that represent the extreme variation of dataset
- o Outliers can be valid data points but since our model is generalization of the data, outliers can affect the performance of the model. We are going to remove the outliers, but please note it's not always a good practice to remove the outliers.
- o To remove the outliers, we use domain **knowledge and standard deviation**

a) Outlier removal - Using Domain Knowledge

- Normally square fit per bedroom is 300 (i.e., 2 bhk apartment is minimum 600 sqft)
- If we have for example 400 sqft apartment with 2 bhk than that seems suspicious and can be removed as an outlier.

- We will remove such outliers by keeping our minimum threshold per bhk to be 300 sqft

We first visualize the data where square fit per bedroom is less than 300

```
In [40]: df8[(df8.total_sqft / df8.bhk) < 300]
```

Out[40]:

	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
...
13277	other	7 Bedroom	1400.0	7.0	218.0	7	15571.428571
13279	other	6 Bedroom	1200.0	5.0	130.0	6	10833.333333
13281	Margondanahalli	5 Bedroom	1375.0	5.0	125.0	5	9090.909091
13303	Vidyaranyapura	5 Bedroom	774.0	5.0	70.0	5	9043.927649
13311	Ramamurthy Nagar	7 Bedroom	1500.0	9.0	250.0	7	16666.666667

744 rows × 7 columns

From above training examples, we have 744 training examples where square fit per bedroom is less than 300. These are outliers, so we can remove them

We first check current dataset shape before removing outliers. We have 13200 rows and 7 columns

```
In [41]: df8.shape
```

Out[41]: (13200, 7)

Next, we remove the outliers, store it into a new dataset df9 and check the dataset shape after removal of the outliers. We are now left with 12456 rows and 7 columns

```
In [42]: df9 = df8[~((df8.total_sqft / df8.bhk) < 300)]
df9.shape
```

Out[42]: (12456, 7)

b) Outlier Removal - Using Standard Deviation and Mean

Standard Deviation

- o Standard deviation is measure of spread that is to know how much the data varies from the average.

- o A low standard deviation tells us that the data is closely clustered around the mean (or average), while a high standard deviation indicates that the data is dispersed over a wider range of values.
- o It is used when the distribution of data is approximately normal, resembling a bell curve.
- One standard deviation (1 Sigma) of the mean will cover 68% of the data. i.e., Data between (mean - standard deviation) & (mean + standard deviation) is 1 Sigma and which is equal to 68%. Here we are going to consider 1 Sigma as our threshold and ***any data outside 1 Sigma will be considered as outlier.***

First, we get the basic statistics of the column "price_per_sqft"

```
In [43]: df9.price_per_sqft.describe()
```

```
Out[43]: 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
```

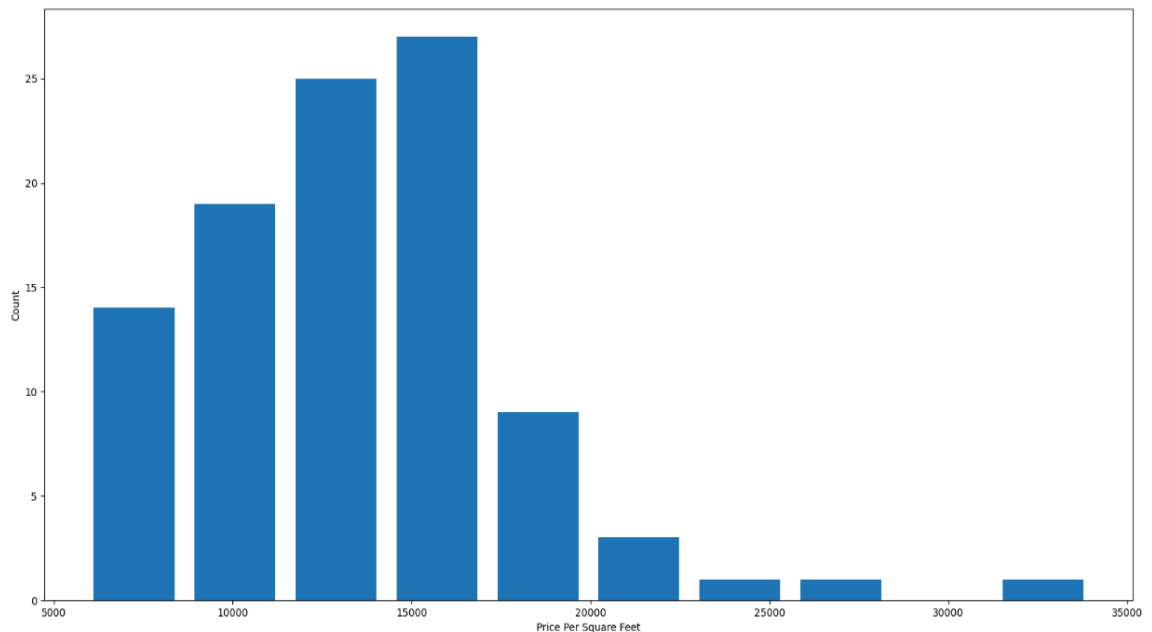
Point to note: it is important to understand that price of every house is specific for every location. We are going to remove outlier's using "price_per_sqft" for each location.

Data visualization for 'price_per_sqft' for **location 'Rajaji Nagar'** before outlier removal.

To be noted: here it is a normal of data so outlier removal using standard deviation and mean works perfectly.

```
In [44]: plt.hist(df9[df9.location == "Rajaji Nagar"].price_per_sqft,rwidth=0.8)
plt.xlabel("Price Per Square Feet")
plt.ylabel("Count")
```

```
Out[44]: Text(0, 0.5, 'Count')
```



After the visualization of the price per sqft of the location “Rajaji Nagar”, we check current dataset shape before outliers are being removed. Currently we have 12456 columns and 7 rows.

```
In [45]: df9.shape
```

```
Out[45]: (12456, 7)
```

Removing outliers using price per sqft. After removal, we have 10242 column and 7 rows.

```
In [46]: def remove_pps_outliers(df):
df_out = pd.DataFrame()
for key, subdf in df.groupby('location'):
    mean = np.mean(subdf.price_per_sqft)
    std = np.std(subdf.price_per_sqft)
    reduced_df = subdf[(subdf.price_per_sqft>(mean-std)) & (subdf.price_per_sqft<=(mean + std))]
    # 1 Sigma value i.e 68% of data
    df_out = pd.concat([df_out,reduced_df],ignore_index=True) # Storing data in 'df_out' dataframe
return df_out

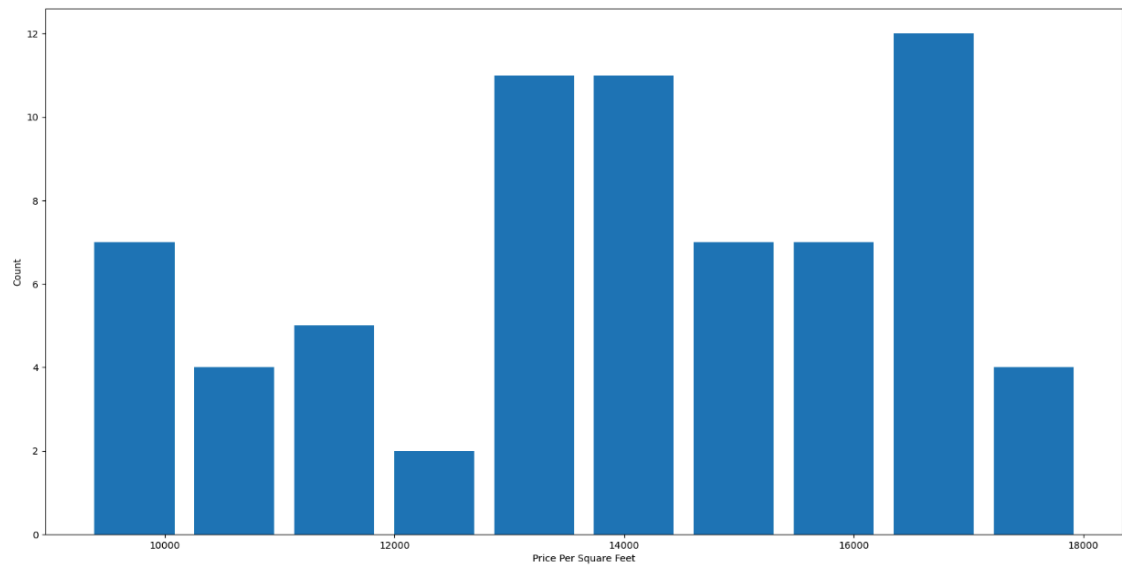
df10 = remove_pps_outliers(df9)
df10.shape
```

```
Out[46]: (10242, 7)
```

Data visualization for 'price_per_sqft' for location 'Rajaji Nagar' after outlier removal


```
In [47]: plt.hist(df10[df10.location == "Rajaji Nagar"].price_per_sqft,rwidth=0.8)
plt.xlabel("Price Per Square Feet")
plt.ylabel("Count")
```

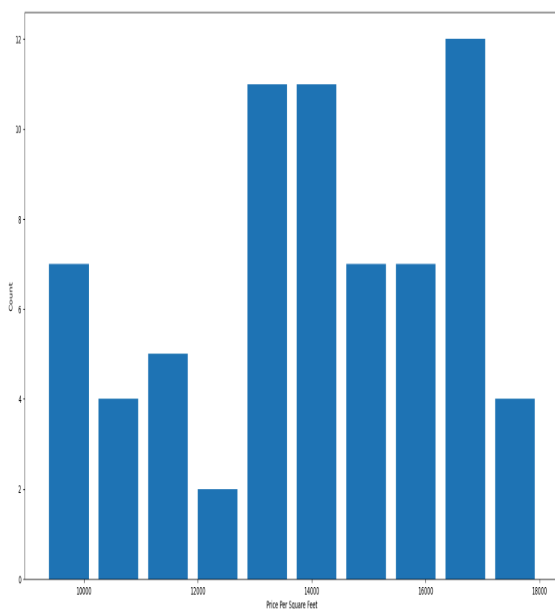
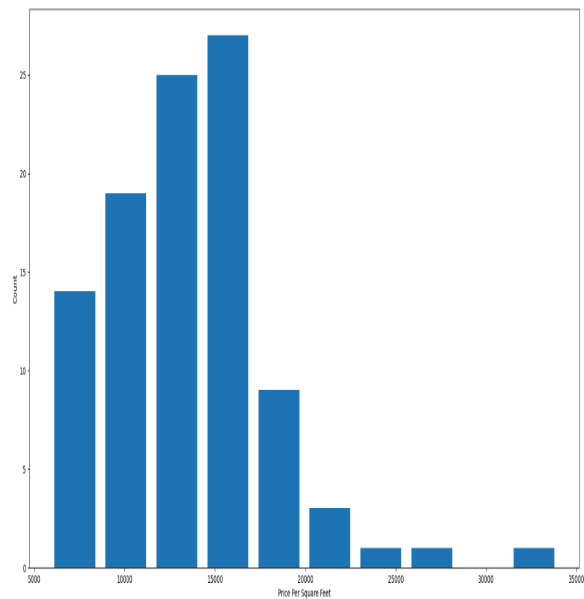
Out[47]: Text(0, 0.5, 'Count')



Summary of Data visualization for 'price_per_sqft' for location Rajaji Nagar Before and after outlier removal.

Before

After



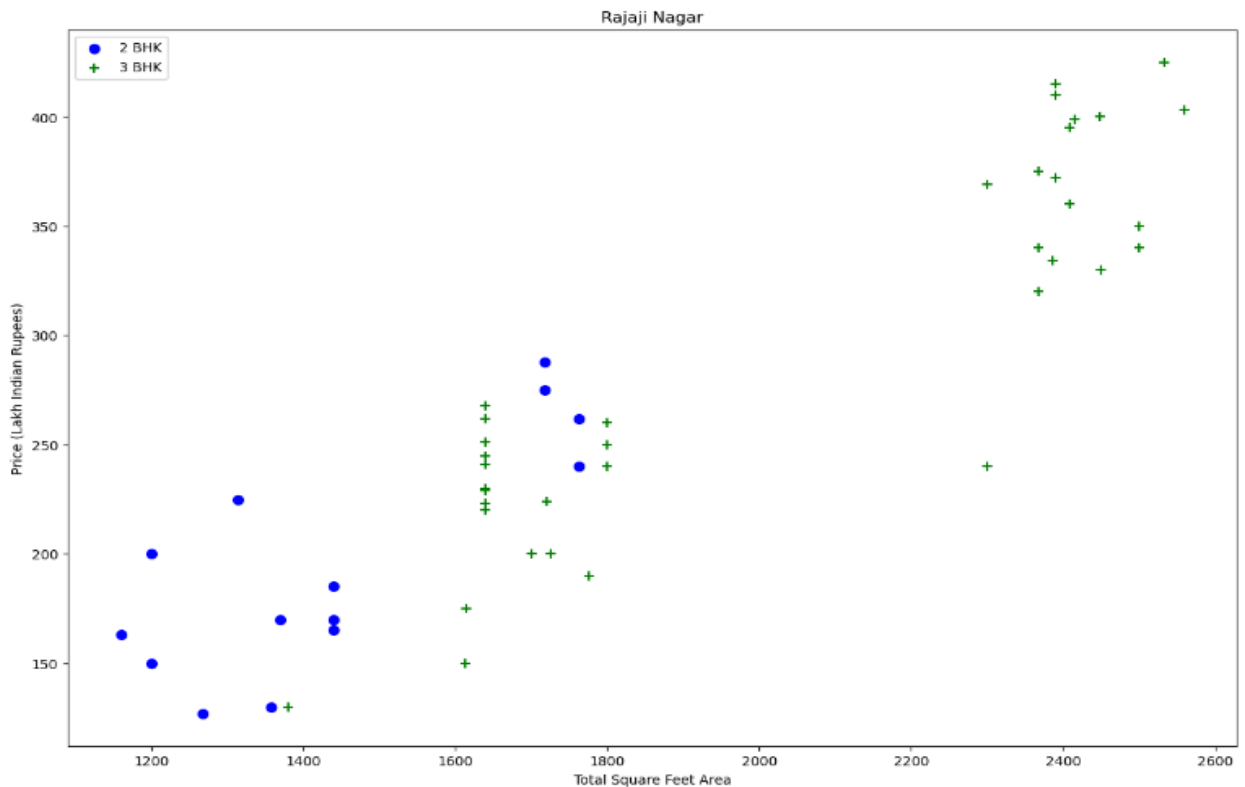
Using domain knowledge for outlier removal; If location and square foot area is also same then price of 3BHK should be more than 2 BHK, there are other factors that also affect the price but for this training set we are treating such values as outlier and remove them.

We check if for a given location how does the 2 BHK and 3 BHK property prices look like. In this example we shall use the location “Rajaji Nagar” and “Hebbal”

I. “Rajaji Nagar”

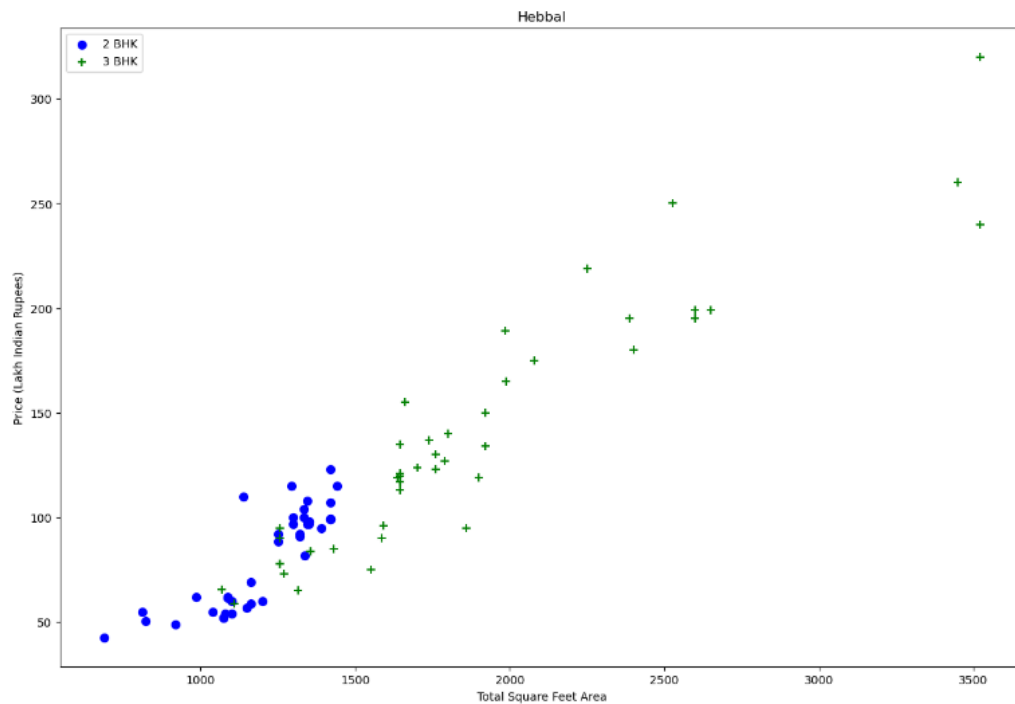
```
In [48]: def plot_scatter_chart(df,location):
bhk2 = df[(df.location==location) & (df.bhk==2)]
bhk3 = df[(df.location==location) & (df.bhk==3)]
matplotlib.rcParams['figure.figsize'] = (15,10)
plt.scatter(bhk2.total_sqft,bhk2.price,color='blue',label='2 BHK', s=50)
plt.scatter(bhk3.total_sqft,bhk3.price,marker='+', color='green',label='3 BHK', s=50)
plt.xlabel("Total Square Feet Area")
plt.ylabel("Price (Lakh Indian Rupees)")
plt.title(location)
plt.legend()

plot_scatter_chart(df10,"Rajaji Nagar")
```



II. Hebbal

```
In [49]: plot_scatter_chart(df10,"Hebbal")
```



We 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 by name 'bhk_stats' with below values of 'price_per_sqft'

```
{
    '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 and store it in a new data set df11. We now have 7317 rows and 7 columns.

```

In [50]: 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_sqft > stats['std'] * 3].index)
          return df.drop(exclude_indices,axis='index')

df11 = remove_bhk_outliers(df10)
df11.shape

```

Out[50]: (7317, 7)

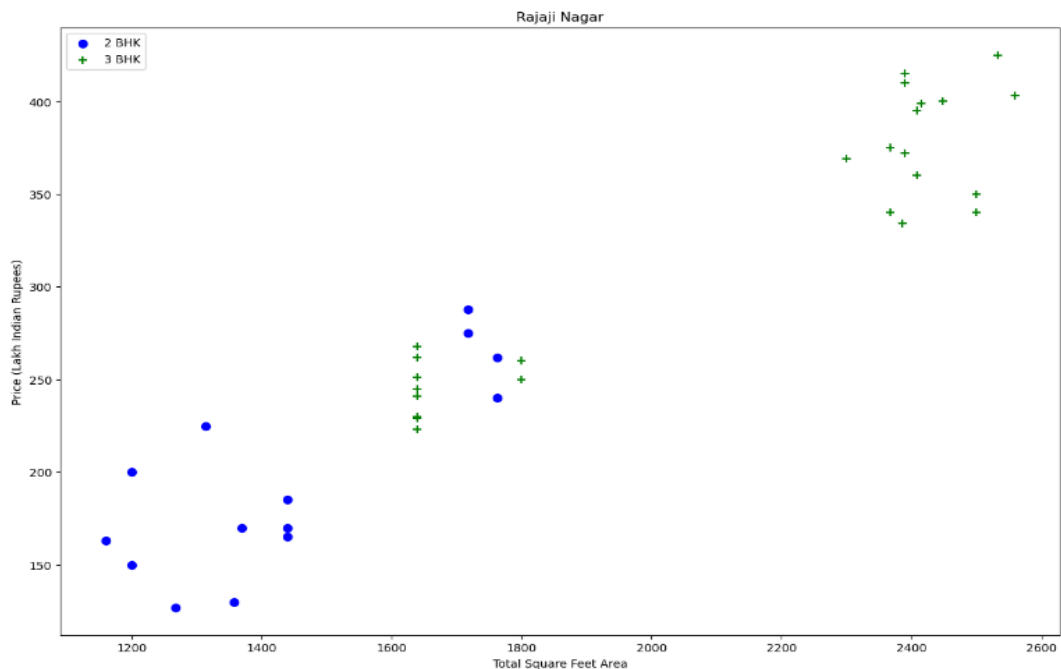
We Plot same scatter chart again to visualize price_per_sqft for 2 BHK and 3 BHK properties for the location “Rajaji Nagar” and “Hebbal”

I. “Rajaji Nagar”

```

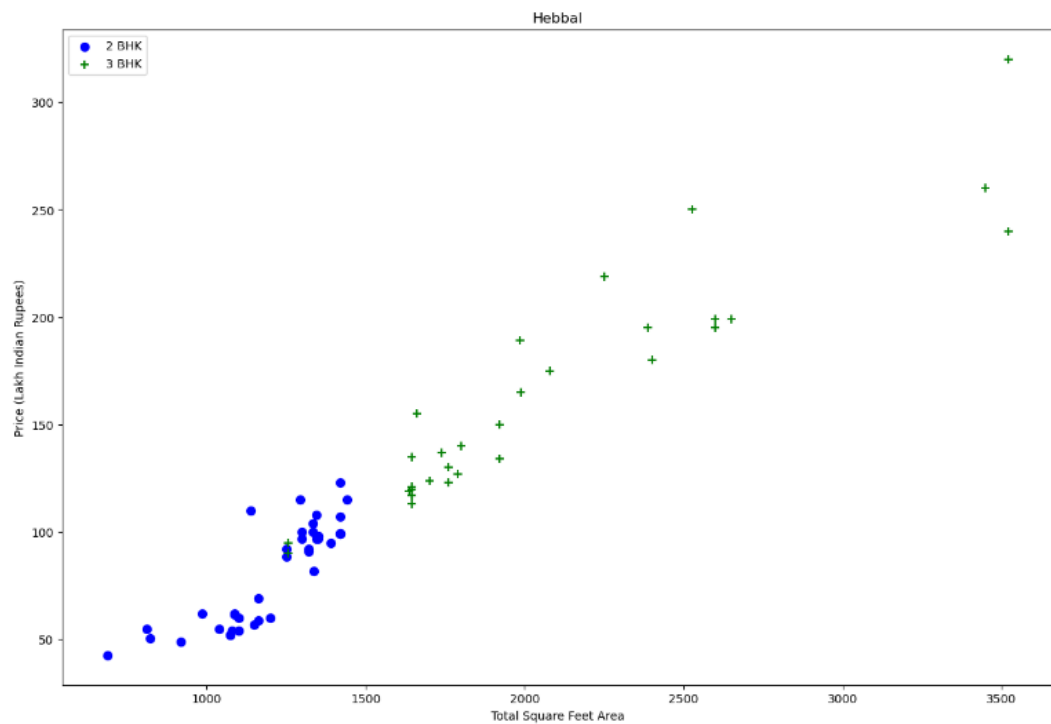
In [51]: plot_scatter_chart(df11,"Rajaji Nagar")

```



II. Hebbal

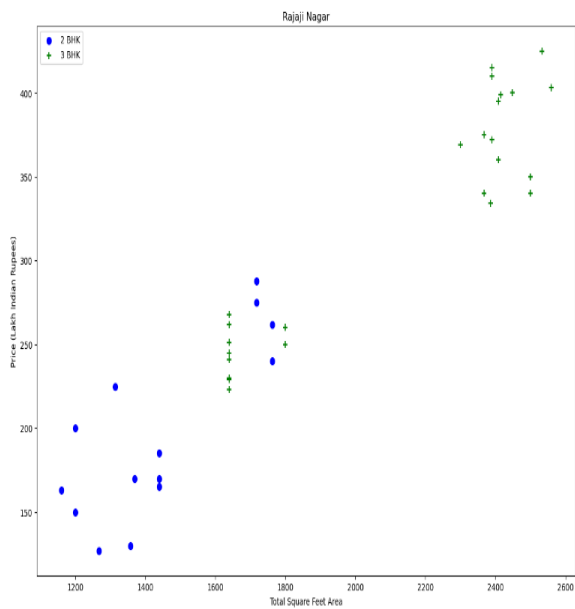
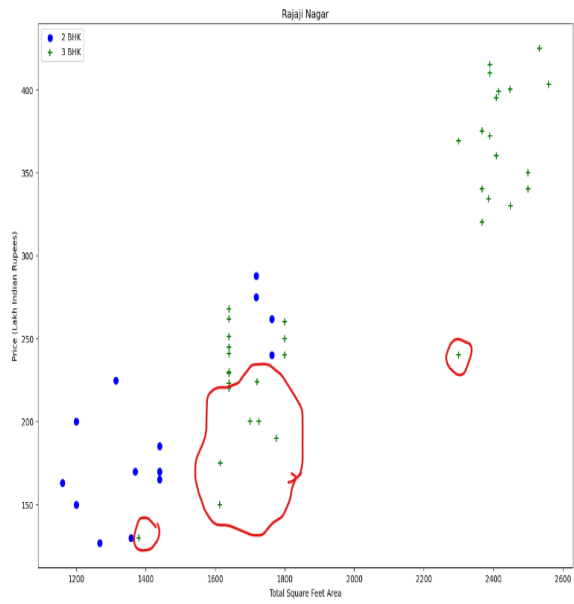
```
In [52]: plot_scatter_chart(df11,"Hebbal")
```



Summary for “price_per_sqft”/ “total square feet area” for location Rajaji Nagar Before and after outlier removal.

Before

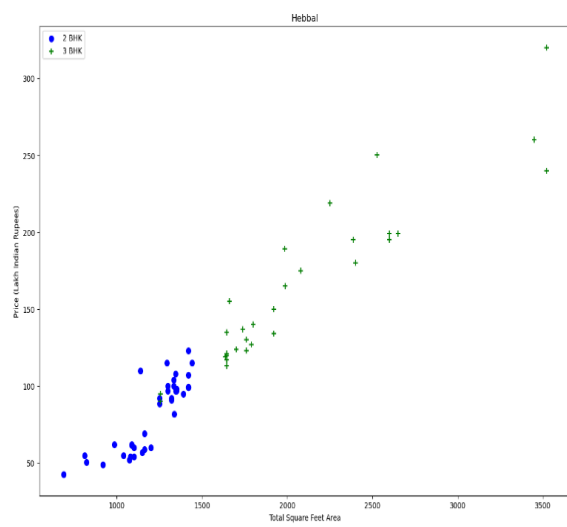
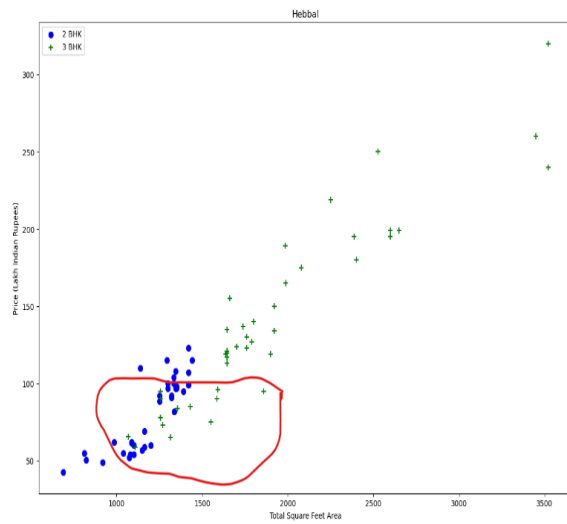
After



Summary for “price_per_sqft”/ “total square feet area” for location Hebbal Before and after outlier removal.

Before

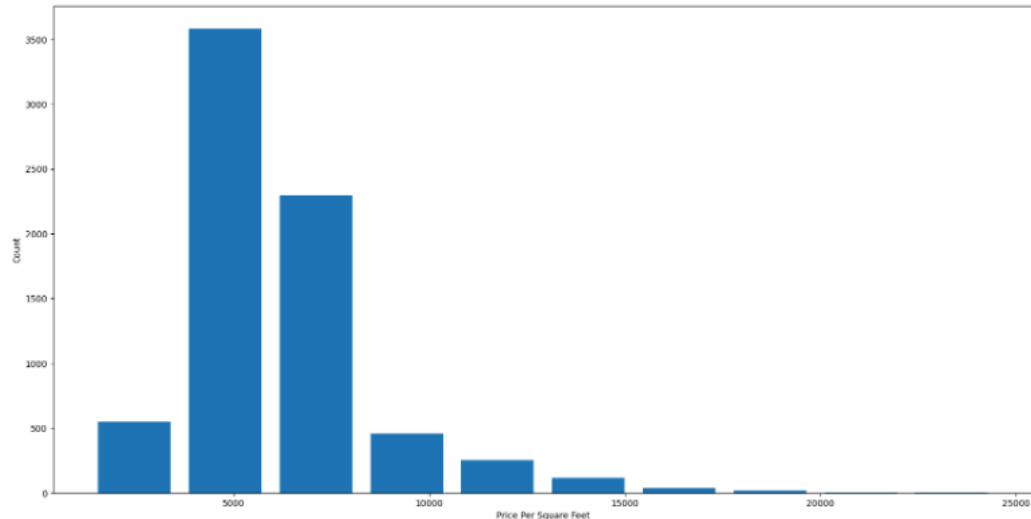
After



Now, we plot a Histogram to visualize the price_per_sqft data after outlier removal


```
In [53]: matplotlib.rcParams["figure.figsize"] = (20,10)
plt.hist(df11.price_per_sqft,rwidth=0.8)
plt.xlabel("Price Per Square Feet")
plt.ylabel("Count")
```

Out[53]: Text(0, 0.5, 'Count')



c) Outlier removal using Domain knowledge - for Bathroom

- Generally, number of bathrooms per BHK (Bathroom Hall Kitchen) are **(number of BHK) + 2**.
- So, using above understanding we identify the outliers and remove them

We get the unique bath from the dataset

```
In [54]: df11.bath.unique()
```

Out[54]: array([4., 3., 2., 5., 8., 1., 6., 7., 9., 12., 16., 13.])

Then we get the training example where the number of baths is more than (number of BHK +2)

```
In [55]: df11[df11.bath > df11.bhk + 2]
```

Out[55]:

	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
8408	other	6 BHK	11338.0	9.0	1000.0	6	8819.897689

We now remove the outliers from the dataset

We first check the current dataset shape before removing the outliers

```
In [56]: df11.shape
```

Out[56]: (7317, 7)

We then remove the outliers with more than (number of BHK +2) bathrooms and store it in a new dataset df12.

```
In [57]: df12 = df11[df11.bath < (df11.bhk + 2)]  
df12.shape
```

Out[57]: (7239, 7)

Visualizing the dataset headers

```
In [58]: df12.head(3)
```

Out[58]:

	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

This concludes our **data cleaning**; we then drop unnecessary columns.

- o Since we have BHK we drop "Size"

- o We have created “price_per_sqft” for **outlier detection and removal purpose**, so we also drop it
- o We store the data into a new dataset df13

```
In [59]: df13 = df12.drop(['size', 'price_per_sqft'], axis='columns')
df13.head()
```

Out[59]:

	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

One Hot encoding

- o Since we have 'location' as categorical feature we use One Hot Encoding to create separate column for each location category and assign binary value 1 or 0.

```
In [60]: dummies = pd.get_dummies(df13.location)
dummies.head()
```

Out[60]:

	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	...	Vishv
0	1	0	0	0	0	0	0	0	0	0	...	
1	1	0	0	0	0	0	0	0	0	0	...	
2	1	0	0	0	0	0	0	0	0	0	...	
3	1	0	0	0	0	0	0	0	0	0	...	
4	1	0	0	0	0	0	0	0	0	0	...	

5 rows × 241 columns

- o To avoid dummy variable (*dummy variables are numerical variables used to represent subgroups of a sample data, it takes the values 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome*) trap problem, we delete the one of the dummy variable columns

```
In [62]: dummies = dummies.drop(['other'],axis='columns')
dummies.head()
```

Out[62]:

	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	...	Vijaya
0	1	0	0	0	0	0	0	0	0	0	...	
1	1	0	0	0	0	0	0	0	0	0	...	
2	1	0	0	0	0	0	0	0	0	0	...	
3	1	0	0	0	0	0	0	0	0	0	...	
4	1	0	0	0	0	0	0	0	0	0	...	

5 rows × 240 columns

- o Then we add dummies data frame to original data frame

```
In [63]: df14 = pd.concat([df13,dummies],axis='columns')
df14.head()
```

Out[63]:

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

5 rows × 245 columns

- o To end our One Hot Encoding, we drop the location feature

```
In [64]: df15 = df14.drop(['location'],axis='columns')
df15.head()
```

Out[64]:

	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	...	Vijayan
0	2850.0	4.0	428.0	4	1	0	0	0	0	0	...	
1	1630.0	3.0	194.0	3	1	0	0	0	0	0	...	
2	1875.0	2.0	235.0	3	1	0	0	0	0	0	...	
3	1200.0	2.0	130.0	3	1	0	0	0	0	0	...	
4	1235.0	2.0	148.0	2	1	0	0	0	0	0	...	

5 rows × 244 columns

Step#5: Build Machine Learning Model

Final shape of our dataset

```
In [65]: df15.shape
```

Out[65]: (7239, 244)

Now we create X (independent variable/ features) and y (dependent variables/target)

- o X (independent variable/ features)

```
In [66]: X = df15.drop(['price'],axis='columns')
X.head()
```

Out[66]:

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

5 rows × 243 columns

x-shape

```
In [67]: X.shape
```

```
Out[67]: (7239, 243)
```

o y (dependent variables/target)

```
In [68]: y = df15.price  
y.head()
```

```
Out[68]: 0    428.0  
1    194.0  
2    235.0  
3    130.0  
4    148.0  
Name: price, dtype: float64
```

y-length

```
In [69]: len(y)
```

```
Out[69]: 7239
```

Splitting the dataset to training and test dataset

```
In [71]: 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)  
print('X_train shape = ',X_train.shape)  
print('X_test shape = ',X_test.shape)  
print('y_train shape = ',y_train.shape)  
print('y_test shape = ',y_test.shape)  
  
X_train shape = (5791, 243)  
X_test shape = (1448, 243)  
y_train shape = (5791,)  
y_test shape = (1448,)
```

Linear Regression

We test the score with the Linear Regression Model (*linear regression model describes the relationship between a dependent variable, y, and one or more independent variables, X.*)

```
In [72]: from sklearn.linear_model import LinearRegression

lr_clf = LinearRegression()
lr_clf.fit(X_train, y_train)
lr_clf.score(X_test, y_test)
```

```
Out[72]: 0.8629132245229444
```

Use K Fold cross validation to measure accuracy of our Linear Regression model

- Using Sklearn cross_val_score function
- ShuffleSplit is used to randomize each fold

To be noted Sklearn's cross_val_score uses **Stratified K Fold** by default

```
In [73]: from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(LinearRegression(), X, y, cv = cv)
```

```
Out[73]: array([0.82702546, 0.86027005, 0.85322178, 0.8436466 , 0.85481502])
```

From above, 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 [76]: 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=False)
        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)
```

Out[76]:

	model	best_score	best_params
0	linear_regression	0.847796	{'normalize': False}
1	lasso	0.726745	{'alpha': 2, 'selection': 'random'}
2	decision_tree	0.716191	{'criterion': 'mse', 'splitter': 'best'}

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

Step#6: Testing The model

- o Since all our locations are now columns in form of dummy variables, all other dummy variables value should be 0 except the one (dummy variable column for our location) we are predicting for.
- o This `(np.where(X.columns==location)[0][0])` code will give us index of dummy column for our location.
- o Now we assign value '1' to this index and keep all other dummy variable columns as '0'


```
In [87]: 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 [88]: predict_price('1st Phase JP Nagar',1000, 2, 2)
```

```
Out[88]: 83.8657025831206
```

```
In [89]: predict_price('1st Phase JP Nagar',1000, 3, 3)
```

```
Out[89]: 86.0806228498683
```

```
In [92]: predict_price('Indira Nagar',1000, 2, 2)
```

```
Out[92]: 193.31197733179843
```

```
In [93]: predict_price('Indira Nagar',1000, 3, 3)
```

```
Out[93]: 195.52689759854616
```

Step#7: Export the tested model to Pickle File

```
In [94]: import pickle

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

Step#8: Export any other important information

We export location and column information to a file that will be useful later in our prediction application.

```
In [99]: import json
columns = {
    'data_columns' : [col.lower() for col in X.columns]
}
with open("columns.json","w") as f:
    f.write(json.dumps(columns))
```

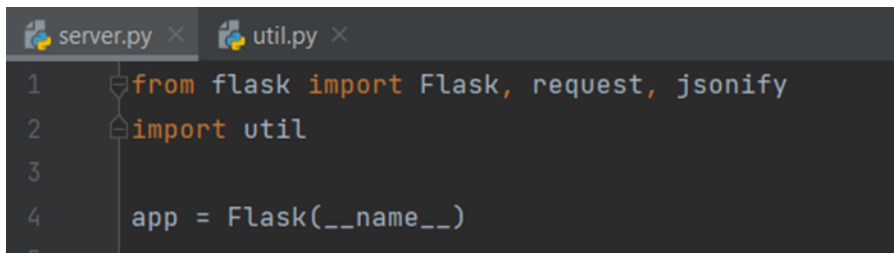
After building up and testing the model to predict the house prices.

We imported model into a pickle a file and other important information needed for our application as seen above.

NEXT STEP – PYTHON FLASK SERVER

The next step was to write a Python flask server which can serve http requests made from the UI (User Interface) and can predict the home prices. The Python Flask server will be used as our backend for the UI application.

We create a “server” python file and import the required libraries.



```
server.py x util.py x
1 from flask import Flask, request, jsonify
2 import util
3
4 app = Flask(__name__)
5
```

We need two routings:

1. The first routine was to return the location in “Bangalore city.”



```
@app.route('/get_location_names', methods=['GET'])
def get_location_names():
    response = jsonify({
        'locations': util.get_location_names()
    })
    response.headers.add('Access-Control-Allow-Origin', '*')
    return response
```

For ‘locations’ we create a new python file call “Utils” and Utils will contain all the core routines whereas the “server „ will just do the routing of request and responds

We import the required libraries, variables and call the “get_location_names” routines which reads the “columns. Json” and return the list of location from the fourth column.

```
server.py x util.py x columns.json x
import pickle
import json
import numpy as np

__locations = None
__data_columns = None
__model = None

def get_location_names():
    return __locations
```

“Column.json”

```
server.py x util.py x columns.json x
{"data_columns": ["total_sqft", "bath", "bhk", "1st block jayanagar", "1st phase jp nagar", "2nd phase judicial layout", "2nd
```

Next, we write a function called “load_saved_artifacts”, this method will load the saved artifacts which are the “columns. json” and “bangalore home prices”; then we store both into a global variable and load the “bangalore_home_prices_model. pickle”

```
def load_saved_artifacts():
    print("loading saved artifacts...start")
    global __data_columns
    global __locations

    with open("./artifacts/columns.json", "r") as f:
        __data_columns = json.load(f)['data_columns']
        __locations = __data_columns[3:] # first 3 columns are sqft, bath, bhk

    global __model
    if __model is None:
        with open('./artifacts/bangalore_home_prices_model.pickle', 'rb') as f:
            __model = pickle.load(f)
    print("loading saved artifacts...done")
```

2. The second routine will be to write a function which can return an estimated price given the location, square foot area, BHK and bathroom.

```
def get_estimated_price(location, sqft, bhk, bath):  
    try:  
        loc_index = __data_columns.index(location.lower())  
    except:  
        loc_index = -1  
  
    x = np.zeros(len(__data_columns))  
    x[0] = sqft  
    x[1] = bath  
    x[2] = bhk  
    if loc_index >= 0:  
        x[loc_index] = 1  
  
    return round(__model.predict([x])[0], 2)
```

That ends the “util.py” file.

Returning on the “server.py”

We create another end point call “predict_home_price” which takes the post and get method, responds to the user request.

```
@app.route('/predict_home_price', methods=['GET', 'POST'])  
def predict_home_price():  
    total_sqft = float(request.form['total_sqft'])  
    location = request.form['location']  
    bhk = int(request.form['bhk'])  
    bath = int(request.form['bath'])  
  
    response = jsonify({  
        'estimated_price': util.get_estimated_price(location, total_sqft, bhk, bath)  
    })  
    response.headers.add('Access-Control-Allow-Origin', '*')  
  
    return response  
  
if __name__ == "__main__":  
    print("Starting Python Flask Server For Home Price Prediction...")  
    util.load_saved_artifacts()  
    app.run()
```

WEBSITE SCREENSHOT

Area (Square Feet)

BHK

1 2 3 4 5

Bath

1 2 3 4 5

Location

Choose a Location ▼

Estimate Price

PROJECT LIMITATION

- We find that min price per sqft is 267 rs/sqft whereas max is 12000000, this shows a wide variation in property prices. We removed outliers per location using mean and one standard deviation.
- Minimum threshold of square ft per bedroom is 300 sqft(i.e. 2 bhk apartment is minimum 600 sqft.) . Anything that is not in line with this, was removed as an outlier.
- We 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).
- Every house has only one more bathroom than number of bedrooms. Anything above that is an outlier or a data error and was removed.

OBSERVATION/CONCLUSION

- There are locations that the prices higher even with same number of BHK and bathroom. This could be because of the higher standard of living in those locations.
- There are many samples in our data where for a given location and square ft area, three bedroom apartment cost less compared to a two bedroom apartment. There could be different reasons for this;
 - The sizes of each rooms of the two bedroom apartment is bigger than that of the three bedroom apartment.
 - Because our data is distributed, sometimes we don't have enough information on why three bedroom apartment would cost less compared to two bedroom apartment.

REFERENCES:

Pedro Marcelino, Comprehensive data exploration with Python, Kaggle, February 2017. Accessed on: April 19, 2021. [Online]

Available:

<https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python>

J. Ade-Ojo, Predicting House Prices with Machine Learning, Towards Data Science, January 8, 2021.

Accessed on: April 19, 2021. [Online]

Available:

<https://towardsdatascience.com/predicting-house-prices-with-machine-learning-62d5bcd0d68f>

House Prices - Advanced Regression Techniques, Kaggle,

Accessed on: April 19, 2021. [Online]

Available: <https://www.kaggle.com/c/house-prices-advanced-regression-techniques>

House Prices EDA, Kaggle,

Accessed on: April 19, 2021. [Online]

Available: <https://www.kaggle.com/dgawlik/house-prices-eda>

Kaaviya Modi, 2021. <https://kaaviyamodi.medium.com/real-estate-price-prediction-2c0c845f3884>

K G Prajwal,

2020. <https://towardsdatascience.com/a-data-science-web-app-to-predict-real-estate-price-d2366df2a4fd>

Janne Kuskeri, Partitioning web applications between the server and the client.

https://www.researchgate.net/publication/221001791_Partitioning_web_applications_between_the_server_and_the_client