Data Science Regression Project: Predicting Home Prices in Banglore

Outline

1. Data Cleaning
2. Feature Engineering
3. Outlier Removal
4. Model Building

Dataset is downloaded from here: [https://www.kaggle.com/amitabhajoy/bengaluru-house-price-data (https://www.kaggle.com/amitabhajoy/bengaluru-house-price-data)](https://www.kaggle.com/amitabhajoy/bengaluru-house-price-data)

In [2]:

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

Data Load: Load banglore home prices into a dataframe

In [3]:

```
df1 = pd.read_csv("bengaluru_house_prices.csv")
df1.head()
```

Out[3]:

	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 [4]:

```
df1.shape
```

Out[4]:

```
(13320, 9)
```

In [5]:

```
df1.columns
```

Out[5]:

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

In [6]:

```
df1['area_type'].unique()
```

Out[6]:

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

In [7]:

```
df1['area_type'].value_counts()
```

Out[7]:

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

Drop features that are not required to build our model

In [9]:

```
df2 = df1.drop(['area_type','society','balcony','availability'],axis='columns')
df2.shape
```

Out[9]:

```
(13320, 5)
```

1. Data Cleaning: Handle NA values

In [10]:

```
df2.isnull().sum()
```

Out[10]:

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

In [11]:

```
df2.shape
```

Out[11]:

```
(13320, 5)
```

In [12]:

```
df3 = df2.dropna()
df3.isnull().sum()
```

Out[12]:

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

In [13]:

```
df3.shape
```

Out[13]:

```
(13246, 5)
```

In size column we see 4 BHK and 4 bedrooms which are the same exist. By using unique method we can see all the unique values in the column.

In [15]:

```
df3['size'].unique()
```

Out[15]:

```
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 want to create a new column and get the first element of size as an integer number with following procedure:

1. Apply a function on size column by using lambda
2. Split the first element on size column
3. Convert it into integer.

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

In [16]:

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

```
<ipython-input-16-681cf3aca53d>: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-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0]))
```

Out[16]:

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

I want to check the bhk>20

In [17]:

```
df3[df3.bhk>20]
```

Out[17]:

	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

43 Bedrooms?!?=> This looks like an error with 2400 total_sqft

I want to check the values on total_sqft

In [18]:

```
df3['total_sqft'].unique()
```

Out[18]:

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

The value 1133-1384 is not a single number it is a range & I want to convert it into single value. One way is to do that I can take an average of two numbers.

First we want to see what kind of variations we have. In order to do that we can detect the number in total_sqft is float or not.

We will define a function is called is_float. The way this function work I will try to convert each value into float and if it is not a valid value such as this range (1133-1384) it will come into this except block and it returns False.

Explore total_sqft feature

In [19]:

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

Now we can apply is_float function into df3['total_sqft']. In order to see the value that it is not a float number we can do ~ opperation.

In [20]:

```python
df3[~df3['total_sqft'].apply(is_float)].head(10)
```

Out[20]:

	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 (e.g. 2100-2850). For such case we can just take average of min and max value in the range. There are other cases such as 34.46Sq. Meter which one can convert to square ft using unit conversion. I am going to just drop such corner cases to keep things simple

In order to clean this data we want to take an average value of the ranges one. Also, about some data like 34.46Sq. Mete or 4125Perch you can change the unit from Perch to sqft but here for simplicity we just ignore that data.

In [21]:

```python
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 [22]:

```
df4 = df3.copy()
df4.total_sqft = df4.total_sqft.apply(convert_sqft_to_num)
df4 = df4[df4.total_sqft.notnull()]
df4.head(5)
```

Out[22]:

	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

For below row, it shows total_sqft as 2475 which is an average of the range 2100-2850

In [23]:

```
df4.loc[30]
```

Out[23]:

```
location      Yelahanka
size              4 BHK
total_sqft         2475
bath                  4
price               186
bhk                   4
Name: 30, dtype: object
```

In [24]:

```
(2100+2850)/2
```

Out[24]:

```
2475.0
```

2. Feature Engineering

In [25]:

```
df5 = df4.copy()
df5.head()
```

Out[25]:

	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

we want to create a new column which is called price_per_sqft. This value is really important in price market. This feature will help us do some outlier cleaning in the later stage. So we are doing some feature engineering which can be helpful fot outlier detection and removal.

Add new feature called price per square feet

In [26]:

```
df5['price_per_sqft'] = df5['price']*100000/df5['total_sqft']
df5.head()
```

Out[26]:

	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

Because price is in lagropies and in order to convert it to dollar we have to multiply it by 100000.

In [29]:

```
df5_stats = df5['price_per_sqft'].describe()
df5_stats
```

Out[29]:

```
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
```

In [30]:

```
df5.to_csv("bhp.csv",index=False)
```

We'll now explore location column. I want to check how many locations are there and how many rows are available in my dataset for location.

Location is a categorical feature and it is text data. If you have too many locations it creates problem.

In [27]:

```
# First check how many locations
len(df5.location.unique())
```

Out[27]:

```
1298
```

The number 1304 is too big and you have too many features. There are techniques to reduce the dimensions. One of the effective techniques is to come up with other category. Other category means when you have 1304 locasions you will fine there are many locations which have only 1 or 2 datapoints.

In [28]:

```
df5.location.unique()
```

Out[28]:

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

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

In [31]:

```
# First: Remove space between or end of the characters by using strip
df5.location = df5.location.apply(lambda x: x.strip())
location_stats = df5['location'].value_counts(ascending=False)
location_stats
```

Out[31]:

```
Whitefield                      533
Sarjapur  Road                  392
Electronic City                 304
Kanakpura Road                  264
Thanisandra                     235
                               ... 
Ramamohanapuram                   1
Amco Colony                       1
Popular Colony                    1
mvj engineering college           1
2Electronic City Phase II         1
Name: location, Length: 1287, dtype: int64
```

In [32]:

```
location_stats.values.sum()
```

Out[32]:

```
13200
```

In [33]:

```
len(location_stats[location_stats>10])
```

Out[33]:

```
240
```

In [34]:

```
len(location_stats)
```

Out[34]:

```
1287
```

In [35]:

```
len(location_stats[location_stats<=10])
```

Out[35]:

```
1047
```

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

```
Out[36]: Basapura                     10
         Nagadevanahalli              10
         Dairy Circle                 10
         Sadashiva Nagar              10
         Thyagaraja Nagar             10
                                      ..
         Ramamohanapuram               1
         Amco Colony                   1
         Popular Colony                1
         mvj engineering college       1
         2Electronic City Phase II     1
         Name: location, Length: 1047, dtype: int64
```

```
In [37]: len(df5.location.unique())
```

```
Out[37]: 1287
```

```
In [38]: df5.location = df5.location.apply(lambda x: 'other' if x in location_stats_less_t
         len(df5.location.unique())
```

```
Out[38]: 241
```

By using this function any value <=10 is other and the others which are >10 keep as x. As you see the number is reduced from 1287 to 241. This is great because when I want to convert it to One Hot Encoding I will only have 283 DataFrames.

```
In [39]: df5.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

3. Outlier Removal Using Business Logic

Outliers are the data points which could be either data errors or represent the extreme variations is datasets. They are valid but it makes sense to remove them. You couls use standard deviation or domin knowledge to remove for them. For example, the sqft per bedrooms should have a logical number and if it is out of this number this should be removed.

As a data scientist when you have a conversation with your business manager (who has expertise in real estate), he will tell you that normally square ft per bedroom is 300 (i.e. 2 bhk apartment is minimum 600 sqft. If you 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 thresold per bhk to be 300 sqft

In [40]:

```
df5[df5.total_sqft/df5.bhk<300].head()
```

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

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 [41]:

```
df5.shape
```

Out[41]:

```
(13200, 7)
```

In [42]:

```
# Remove Outlier for total_sqft
df6 = df5[~(df5.total_sqft/df5.bhk<300)]
df6.shape
```

Out[42]:

```
(12456, 7)
```

Outlier Removal Using Standard Deviation and Mean

In [43]:

```
df6.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
```

Here we find that min price per sqft is 267 rs/sqft whereas max is 12000000, this shows a wide variation in property prices. We should remove outliers per location using mean and one standard deviation

e are going to write a function that can remove the extreme cases based on std. If our dataset has a normal distribution almost 68% of the dataset is around mean +- std. So we are going to filter out anything which is beyond std.

In [44]:

```
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<
        df_out = pd.concat([df_out,reduced_df],ignore_index=True)
    return df_out
df7 = remove_pps_outliers(df6)
df7.shape
```

Out[44]:

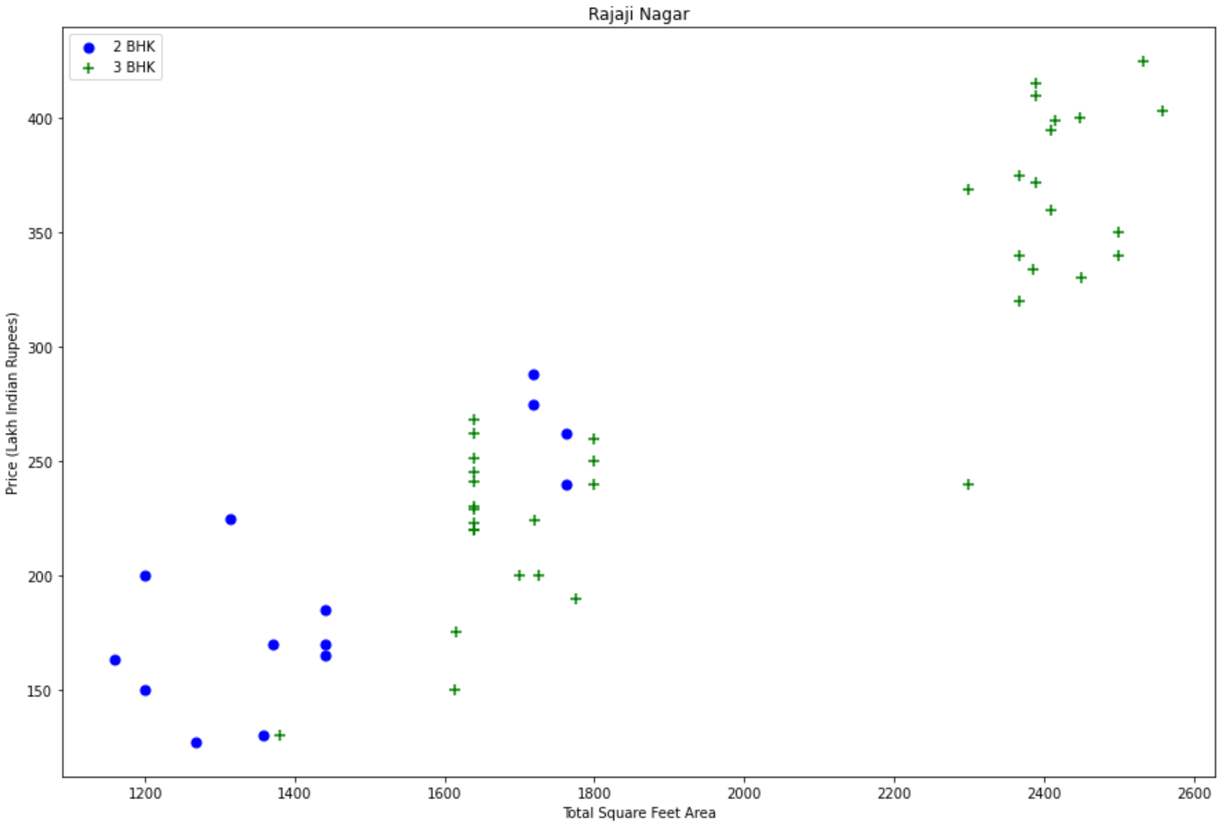
```
(10242, 7)
```

With the function we are keeping anything above m-st or below m+st in my reduced dataframe and then we'll keep on appending into my output dataframe.

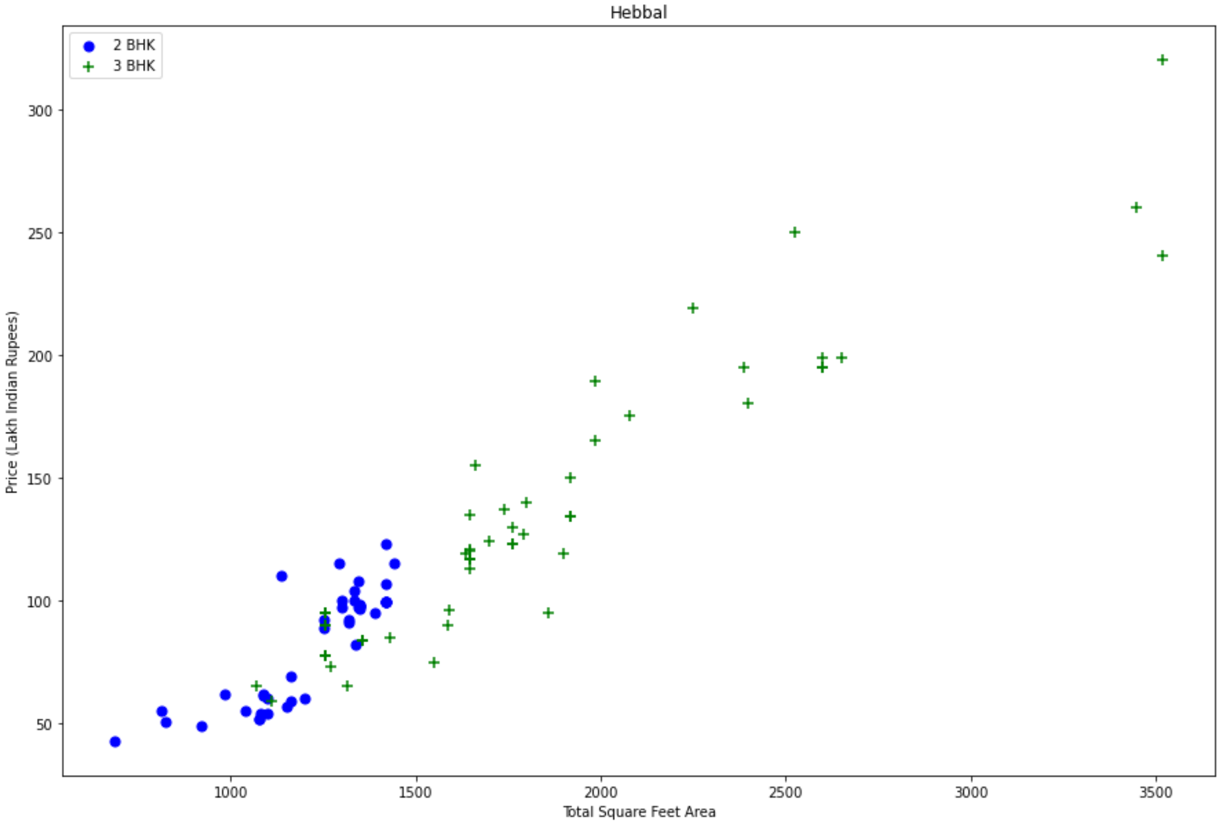
Let's check if for a given location how does the 2 BHK and 3 BHK property prices look like

```
In [45]: 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
             plt.xlabel("Total Square Feet Area")
             plt.ylabel("Price (Lakh Indian Rupees)")
             plt.title(location)
             plt.legend()

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



```
In [46]: plot_scatter_chart(df7,"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 [47]:

```
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_
    return df.drop(exclude_indices,axis='index')
df8 = remove_bhk_outliers(df7)
# df8 = df7.copy()
df8.shape
```

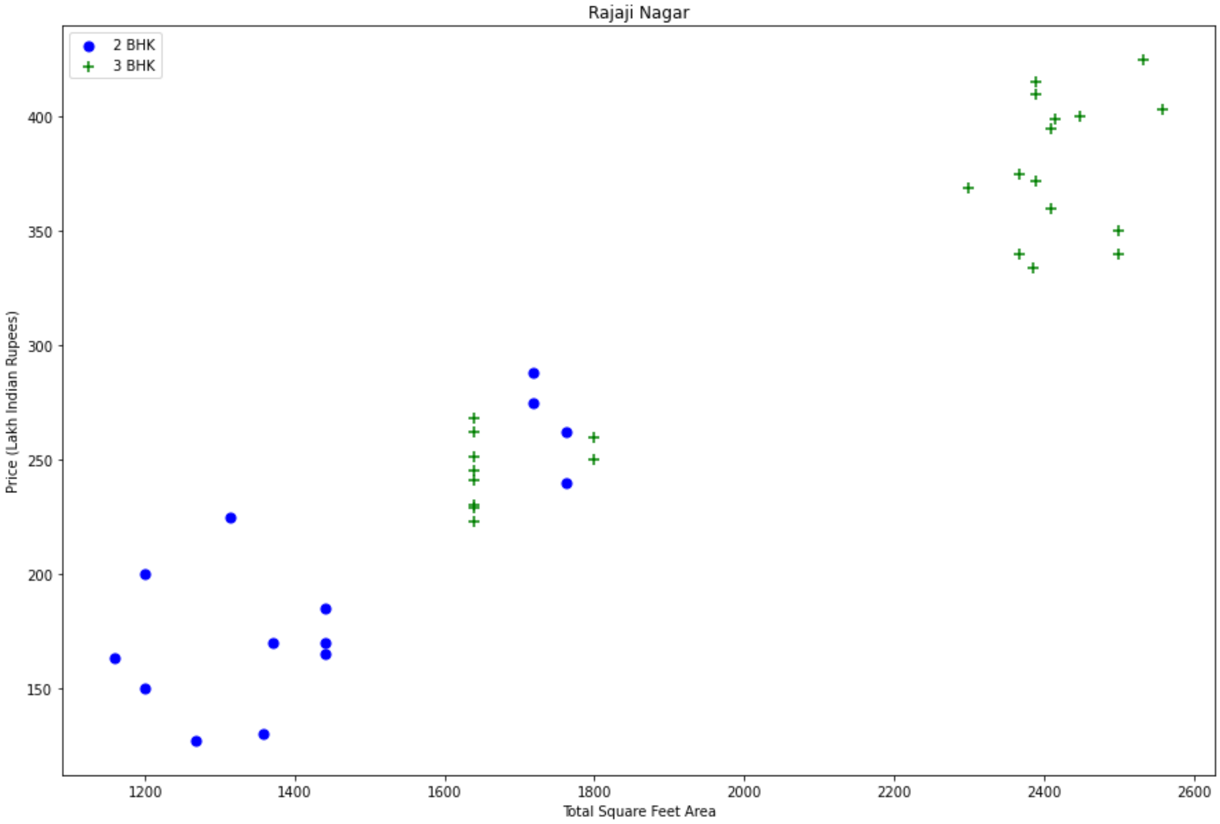
Out[47]:

```
(7317, 7)
```

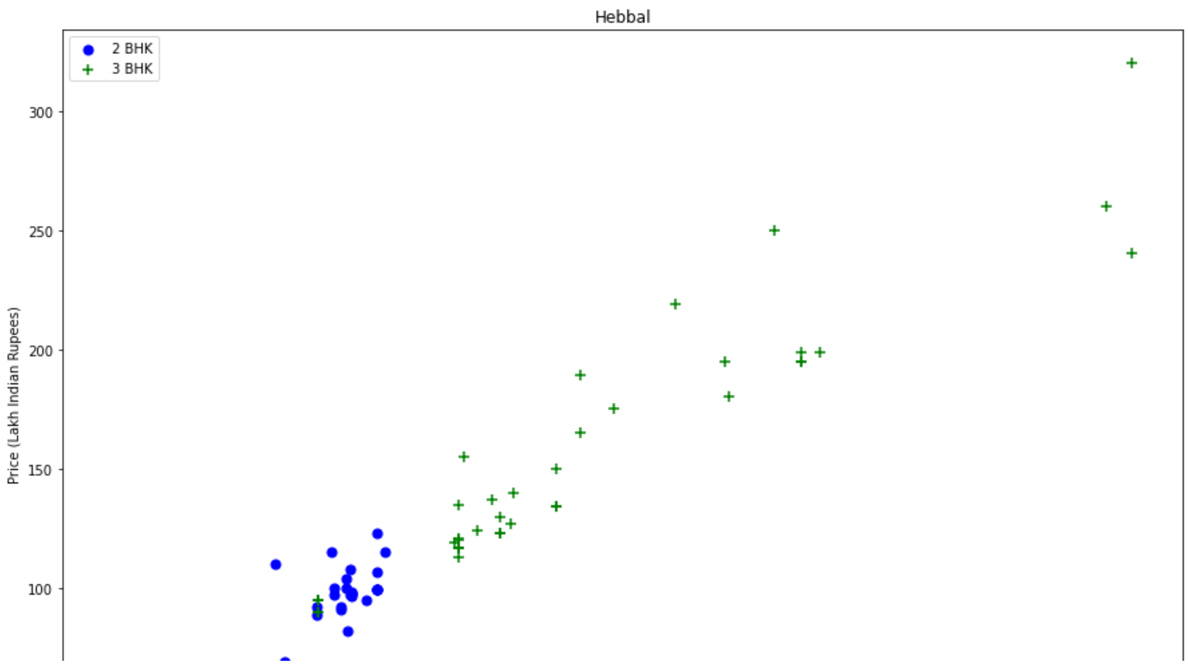
Function Explanation: First I'm doing location groupby going through every location, and for every location dataframe I am creating new dataframe based on bhk. In this new dataframe I am calculating mean, std, and count. Once this for loop is done I am running another for loop trying to exclude those datapoints whose value(price_per_sqft) is less than the mean of the previous bhk.

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

In [48]:
```
plot_scatter_chart(df8,"Rajaji Nagar")
```



In [49]:
```
plot_scatter_chart(df8,"Hebbal")
```



Cleaning is very great even though there are some data whose not clean completely but that's ok.

Based on above charts we can see that data points highlighted in red below are outliers and they are being removed due to remove_bhk_outliers function

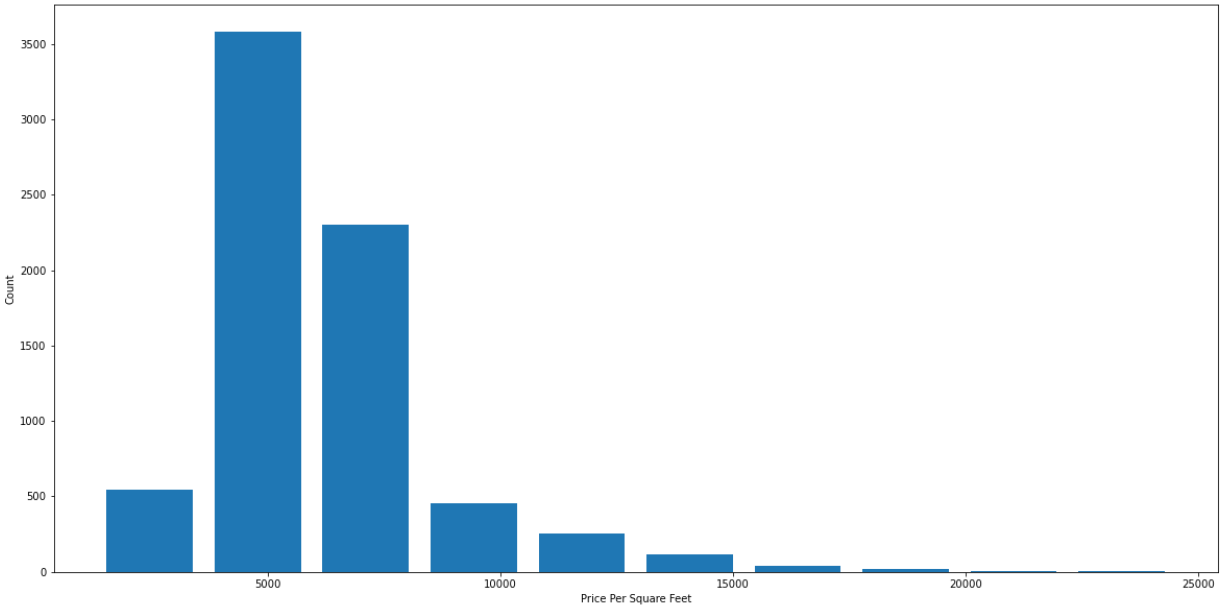
Plot a histogram to see how many apartments I have in per sqft area.

In [50]:

```
import matplotlib
matplotlib.rcParams["figure.figsize"] = (20,10)
plt.hist(df8.price_per_sqft,rwidth=0.8)
plt.xlabel("Price Per Square Feet")
plt.ylabel("Count")
```

Out[50]:

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



As you see the majority of our data points are between 0 to 10000 price_per_sqft.

Outlier Removal Using Bathrooms Feature

In [51]:

```
df8.bath.unique()
```

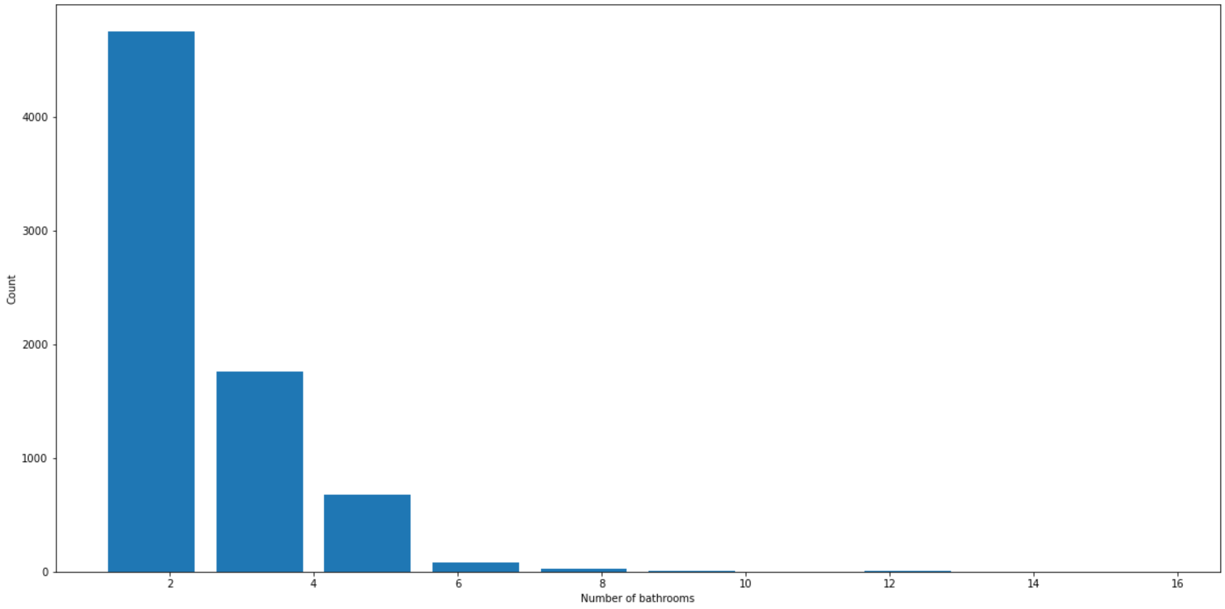
Out[51]:

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

As you see there are some apartment with 12, 13, or 16 bathrooms?!?

In [52]:
```
plt.hist(df8.bath,rwidth=0.8)
plt.xlabel("Number of bathrooms")
plt.ylabel("Count")
```

Out[52]:
```
Text(0, 0.5, 'Count')
```



In [53]:
```
df8[df8.bath>10]
```

Out[53]:

	location	size	total_sqft	bath	price	bhk	price_per_sqft
5277	Neeladri Nagar	10 BHK	4000.0	12.0	160.0	10	4000.000000
8483	other	10 BHK	12000.0	12.0	525.0	10	4375.000000
8572	other	16 BHK	10000.0	16.0	550.0	16	5500.000000
9306	other	11 BHK	6000.0	12.0	150.0	11	2500.000000
9637	other	13 BHK	5425.0	13.0	275.0	13	5069.124424

Rule of Thump: It is unusual to have 2 more bathrooms than number of bedrooms in a

home

In [54]:
```
df8[df8.bath>df8.bhk+2]
```

Out[54]:

	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

Again the business manager has a conversation with you (i.e. a data scientist) that if you have 4 bedroom home and even if you have bathroom in all 4 rooms plus one guest bathroom, you will have total bath = total bed + 1 max. Anything above that is an outlier or a data error and can be removed

In [55]:
```
df9 = df8[df8.bath<df8.bhk+2]
df9.shape
```

Out[55]:
```
(7239, 7)
```

In [56]:
```
df9.head(2)
```

Out[56]:

	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

Now my dataframe almost clean, so I can start preparing it for machine learning training. For that, I have to drop some unnecessary features. size is unnecessary because for size, we already have bhk. price_per_sqft can also be dropped because this data is aditional and we use it for outlier detection.

In [57]:
```
df10 = df9.drop(['size','price_per_sqft'],axis='columns')
df10.head(3)
```

Out[57]:

	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

Use One Hot Encoding For Location

Machine Learning model cannot interpret text column so we have to convert location into numeric column. One of the ways to convert a text column into numeric is to use One Hot Encoding.

In [58]:

```
dummies = pd.get_dummies(df10.location)
dummies.head(3)
```

Out[58]:

	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	...	Vish
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 rows × 241 columns

In [59]:

```
# append dummies into data
# We do not need other data from dummies
df11 = pd.concat([df10,dummies.drop('other',axis='columns')],axis='columns')
df11.head()
```

Out[59]:

	location	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	...	Vij
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

In [60]:

```
# Now we can drop location columns because we already covered it with dummies
df12 = df11.drop('location',axis='columns')
df12.head(2)
```

Out[60]:

	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	...	Vija
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 rows × 244 columns

4. Building a Model

Now our data is clean and is ready for model building. In this part we are going to build a Machine Learnig model and then use K fold Cross Validation and gridsearch CV to come up with the best algorithm as well as the best parameters.

In [61]:

```
df12.shape
```

Out[61]:

```
(7239, 244)
```

In [62]:

```
X = df12.drop(['price'],axis='columns')
X.head(3)
```

Out[62]:

	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 rows × 243 columns

In [63]:

```
X.shape
```

Out[63]:

```
(7239, 243)
```

In [64]:

```
y = df12.price
y.head(3)
```

Out[64]:

```
0    428.0
1    194.0
2    235.0
Name: price, dtype: float64
```

In [65]:

```
len(y)
```

Out[65]:

```
7239
```

train test split

In [66]:

```
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_stat
```

In [67]:

```
from sklearn.linear_model import LinearRegression
lr_clf = LinearRegression()
lr_clf.fit(X_train,y_train)
lr_clf.score(X_test,y_test)
```

Out[67]:

```
0.8629132245229442
```

Use K Fold cross validation to measure accuracy of LinearRegression model

In [68]:

```
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[68]:

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

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 [69]:

```python
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
        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[69]:

	model	best_score	best_params
0	linear_regression	0.847796	{'normalize': False}
1	lasso	0.726823	{'alpha': 2, 'selection': 'random'}
2	decision_tree	0.717227	{'criterion': 'friedman_mse', 'splitter': 'best'}

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 [70]: X.columns
```

```
Out[70]: Index(['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',
                'Vittasandra', 'Whitefield', 'Yelachenahalli', 'Yelahanka',
                'Yelahanka New Town', 'Yelenahalli', 'Yeshwanthpur'],
               dtype='object', length=243)
```

```
In [71]: np.where(X.columns=='bhk')[0][0]
```

```
Out[71]: 2
```

It returns the location of 'bhk'.

Because Linear Regression is the winner we use this model which we already created lr_clf for prediction. Because we have almost 200 locations we have to specify our location for prediction like above example. once you have location index you can set that particular index value to be 1.

```
In [73]: 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 [74]: predict_price('1st Phase JP Nagar',1000, 2, 2)
```

```
Out[74]: 83.86570258312275
```

```
In [75]: predict_price('1st Phase JP Nagar',1000, 3, 3)
```

```
Out[75]: 86.08062284987054
```

```
In [76]: predict_price('Indira Nagar',1000, 2, 2)
```

```
Out[76]: 193.31197733179937
```

In [77]:
```
predict_price('Indira Nagar',1000, 3, 3)
```

Out[77]:
```
195.52689759854715
```

Export the tested model to a pickle file

Now you have come this long way with this aesome model and it is ready for production so you need to export it into a pickle file.

In [78]:
```
import pickle
with open('banglore_home_prices_model.pickle','wb') as f:
    pickle.dump(lr_clf,f)
```

The file size of pickle file is 1 KB, this is because this picle file doesnt have the actual data, it has just model, coefficients.

Export column information into a Jason file

Other than the model we also need the columns information. For example, in my predict_price function I have X.columns. These columns and their index and their structures are important for making a prediction.

In [79]:
```
import json
columns = {
    'data_columns' : [col.lower() for col in X.columns]
}
with open("columns.json","w") as f:
    f.write(json.dumps(columns))
```

Date	Author
11-21-2021	**Ehsan Zia**