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

In [2]:
```
The line matplotlib.rcParams["figure.figsize"] = (20, 10) in Python's Matplotlib li

matplotlib.rcParams is a dictionary-like structure that holds various configuration
"figure.figsize" is the key in this dictionary that specifies the default dimension
(20, 10) represents the width and height of the figure in inches.
Matplotlib's rcParams (runtime configuration parameters) allows you to control vari
```

```
  Cell In[2], line 1
    The line matplotlib.rcParams["figure.figsize"] = (20, 10) in Python's Matplotl
ib library is used to set the default size of all figures (plots) generated during
that session.
                                                                               ^
SyntaxError: unterminated string literal (detected at line 1)
```

In [3]:
```
df = pd.read_csv("C:\\Users\\P.Navya Sree\\OneDrive\\Documents\\ML\\benghse.csv")
df.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]:
```
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
```

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

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['size'].head()
```

Out[8]:
```
0        2 BHK
1    4 Bedroom
2        3 BHK
3        3 BHK
4        2 BHK
Name: size, dtype: object
```

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)
```

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

```
C:\Users\P.Navya Sree\AppData\Local\Temp\ipykernel_15992\2533156592.py:1: SettingW
ithCopyWarning:
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/stabl
e/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]:
```
#So here it is defientely an error of having a 43 bedrooms with total sqft of 2400
#actually uncorrelated data hence we need to change it !!
```

In [15]:
```
df2.total_sqft.unique()
```

Out[15]:
```
array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)
```

In [16]:
```
#first to alter them we have range of values where the sqft ranges but here we don'
#that hence we take the average of the values in this
```

In [17]:
```
def is_float(x):
    try:
        float(x)
    except:
        return False
    return True
```

In [18]:
```
df2[df2['total_sqft'].apply(is_float)]
```

Out[18]:

	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
...	...	...	...	...	...	...
13315	Whitefield	5 Bedroom	3453	4.0	231.00	5
13316	Richards Town	4 BHK	3600	5.0	400.00	4
13317	Raja Rajeshwari Nagar	2 BHK	1141	2.0	60.00	2
13318	Padmanabhanagar	4 BHK	4689	4.0	488.00	4
13319	Doddathoguru	1 BHK	550	1.0	17.00	1

13056 rows × 6 columns

In [19]:
```
#so here we got the values which are exactly float but in order to get the vlues wh
#aren't float so we negate the given values
```

In [20]:
```
df2[~df2['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

In [21]:
```
#so here we got the sqft values where they are not floats or where the is_float met
#return false
#hence we make the average of this values!!
```

In [22]:
```
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 [23]:
```
convert_sqft_to_num('2166')
```

Out[23]:
```
2166.0
```

In [24]:
```
convert_sqft_to_num('789mtr')
```

In [25]:
```
df3 = df2.copy()
df3['total_sqft'] = df3['total_sqft'].apply(convert_sqft_to_num)
df3.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

In [26]:
```
df3.loc[30]
```

Out[26]:
```
location      Yelahanka
size              4 BHK
total_sqft       2475.0
bath                4.0
price             186.0
bhk                   4
Name: 30, dtype: object
```

In [27]:
```
df2.loc[30]
```

Out[27]:
```
location        Yelahanka
size                4 BHK
total_sqft    2100 - 2850
bath                  4.0
price               186.0
bhk                     4
Name: 30, dtype: object
```

In [28]:
```
#here we get the average of two values in this in df3 data frame!!
```

In [29]:
```
df3.head()
```

Out[29]:

	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

In [30]:
```
df4 = df3.copy()
```

In [31]:
```
df4.head()
```

Out[31]:

	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

In [32]:
```
#So here we are  doing feature engineering and we create a column for outlier detec
```

In [33]:
```
df4['price_per_sqft'] = df4['price']*100000/df4['total_sqft']
df4.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

In [34]:
```
len(df4.location.unique())
```

Out[34]:
```
1304
```

In [35]:
```
len(df4.location.unique())
```

Out[35]:
```
1304
```

In [36]:
```
# Grouping Data:
The primary idea is to group data by one or more columns. For example, if you have

2. Aggregation:
After grouping, you often perform aggregation to summarize the grouped data. This c

Counting the number of items in each group.
Averaging values within each group.
Summing values within each group.
Finding the maximum or minimum value in each group.
3. Transformation:
Sometimes, instead of just aggregating, you might transform the grouped data. For e

4. Feature Engineering:
In machine learning, you often use "group by" for feature engineering, where you gr

USED FOR  :

    1. Preproccesing
    2. Handling Imbalance data
    3.Feature Aggregation

    Check in gpt for syntax
```

```
  Cell In[36], line 21
    3.Feature Aggregation
     ^
SyntaxError: invalid decimal literal
```

In [37]:
```
df4.location = df4.location.apply(lambda x: x.strip())
location_stats = df4.groupby('location')['location'].agg('count').sort_values(ascer
location_stats
```

Out[37]:
```
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 [38]:
```
#handling more numbered unique values like locations giri nagar,onwards
# so this locations may inscrease the dimensionaliity of data to learn by ML models
# not only that it as these unique values are reallly smaller and only single value
#then it might cause noise in the data by these data points
# so we try to reduce its dimensionality by grouping them as one category or taking
```

In [39]:
```
Instead of removing rare locations entirely, you might:

Aggregate or Group Rare Locations: You could group these rare locations into an "Ot
Use Target Encoding: You can replace each location with the average target value (I
```

```
  Cell In[39], line 1
    Instead of removing rare locations entirely, you might:
             ^
SyntaxError: invalid syntax
```

In [40]:
```
#S here we try to remove the locations_stats which are less than 10
#becoz highest location_stats are upto 200 to 500
```

In [41]:
```
len(location_stats[location_stats<=10])
```

Out[41]:
```
1052
```

In [ ]:
```

```

In [42]:
```
location_stats_less_than_10 =location_stats[location_stats<=10]
location_stats_less_than_10
```

Out[42]:
```
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 [43]:
```
len(df4.location.unique())
```

Out[43]:
```
1293
```

In [44]:
```
df4.location = df4.location.apply(lambda x :'other' if x in location_stats_less_tha
```

In [45]:
```
df4.location.unique()
```

Out[45]:
```
array(['Electronic City Phase II', 'Chikka Tirupathi', 'Uttarahalli',
       'Lingadheeranahalli', 'Kothanur', 'Whitefield', 'Old Airport Road',
       'Rajaji Nagar', 'Marathahalli', 'other', '7th Phase JP Nagar',
       'Gottigere', 'Sarjapur', 'Mysore Road', 'Bisuvanahalli',
       'Raja Rajeshwari Nagar', 'Kengeri', 'Binny Pete', 'Thanisandra',
       'Bellandur', 'Electronic City', 'Ramagondanahalli', 'Yelahanka',
       'Hebbal', 'Kasturi Nagar', 'Kanakpura Road',
       'Electronics City Phase 1', 'Kundalahalli', 'Chikkalasandra',
       'Murugeshpalya', 'Sarjapur  Road', 'HSR Layout', 'Doddathoguru',
       'KR Puram', 'Bhoganhalli', 'Lakshminarayana Pura', 'Begur Road',
       'Varthur', 'Bommanahalli', 'Gunjur', 'Devarachikkanahalli',
       'Hegde Nagar', 'Haralur Road', 'Hennur Road', 'Kothannur',
       'Kalena Agrahara', 'Kaval Byrasandra', 'ISRO Layout',
       'Garudachar Palya', 'EPIP Zone', 'Dasanapura', 'Kasavanhalli',
       'Sanjay nagar', 'Domlur', 'Sarjapura - Attibele Road',
       'Yeshwanthpur', 'Chandapura', 'Nagarbhavi', 'Devanahalli',
       'Ramamurthy Nagar', 'Malleshwaram', 'Akshaya Nagar', 'Shampura',
       'Kadugodi', 'LB Shastri Nagar', 'Hormavu', 'Vishwapriya Layout',
       'Kudlu Gate', '8th Phase JP Nagar', 'Bommasandra Industrial Area',
       'Anandapura', 'Vishveshwarya Layout', 'Kengeri Satellite Town',
       'Kannamangala', 'Hulimavu', 'Mahalakshmi Layout', 'Hosa Road',
       'Attibele', 'CV Raman Nagar', 'Kumaraswami Layout', 'Nagavara',
       'Hebbal Kempapura', 'Vijayanagar', 'Pattandur Agrahara',
       'Nagasandra', 'Kogilu', 'Panathur', 'Padmanabhanagar',
       '1st Block Jayanagar', 'Kammasandra', 'Dasarahalli', 'Magadi Road',
       'Koramangala', 'Dommasandra', 'Budigere', 'Kalyan nagar',
       'OMBR Layout', 'Horamavu Agara', 'Ambedkar Nagar',
       'Talaghattapura', 'Balagere', 'Jigani', 'Gollarapalya Hosahalli',
       'Old Madras Road', 'Kaggadasapura', '9th Phase JP Nagar', 'Jakkur',
       'TC Palaya', 'Giri Nagar', 'Singasandra', 'AECS Layout',
       'Mallasandra', 'Begur', 'JP Nagar', 'Malleshpalya', 'Munnekollal',
       'Kaggalipura', '6th Phase JP Nagar', 'Ulsoor', 'Thigalarapalya',
       'Somasundara Palya', 'Basaveshwara Nagar', 'Bommasandra',
       'Ardendale', 'Harlur', 'Kodihalli', 'Narayanapura',
       'Bannerghatta Road', 'Hennur', '5th Phase JP Nagar', 'Kodigehaali',
       'Billekahalli', 'Jalahalli', 'Mahadevpura', 'Anekal', 'Sompura',
       'Dodda Nekkundi', 'Hosur Road', 'Battarahalli', 'Sultan Palaya',
       'Ambalipura', 'Hoodi', 'Brookefield', 'Yelenahalli', 'Vittasandra',
       '2nd Stage Nagarbhavi', 'Vidyaranyapura', 'Amruthahalli',
       'Kodigehalli', 'Subramanyapura', 'Basavangudi', 'Kenchenahalli',
       'Banjara Layout', 'Kereguddadahalli', 'Kambipura',
       'Banashankari Stage III', 'Sector 7 HSR Layout', 'Rajiv Nagar',
       'Arekere', 'Mico Layout', 'Kammanahalli', 'Banashankari',
       'Chikkabanavar', 'HRBR Layout', 'Nehru Nagar', 'Kanakapura',
       'Konanakunte', 'Margondanahalli', 'R.T. Nagar', 'Tumkur Road',
       'Vasanthapura', 'GM Palaya', 'Jalahalli East', 'Hosakerehalli',
       'Indira Nagar', 'Kodichikkanahalli', 'Varthur Road', 'Anjanapura',
       'Abbigere', 'Tindlu', 'Gubbalala', 'Parappana Agrahara',
       'Cunningham Road', 'Kudlu', 'Banashankari Stage VI', 'Cox Town',
       'Kathriguppe', 'HBR Layout', 'Yelahanka New Town',
       'Sahakara Nagar', 'Rachenahalli', 'Yelachenahalli',
       'Green Glen Layout', 'Thubarahalli', 'Horamavu Banaswadi',
       '1st Phase JP Nagar', 'NGR Layout', 'Seegehalli', 'BEML Layout',
       'NRI Layout', 'ITPL', 'Babusapalaya', 'Iblur Village',
       'Ananth Nagar', 'Channasandra', 'Choodasandra', 'Kaikondrahalli',
       'Neeladri Nagar', 'Frazer Town', 'Cooke Town', 'Doddakallasandra',
       'Chamrajpet', 'Rayasandra', '5th Block Hbr Layout', 'Pai Layout',
       'Banashankari Stage V', 'Sonnenahalli', 'Benson Town',
       '2nd Phase Judicial Layout', 'Poorna Pragna Layout',
       'Judicial Layout', 'Banashankari Stage II', 'Karuna Nagar',
       'Bannerghatta', 'Marsur', 'Bommenahalli', 'Laggere',
       'Prithvi Layout', 'Banaswadi', 'Sector 2 HSR Layout',
       'Shivaji Nagar', 'Badavala Nagar', 'Nagavarapalya', 'BTM Layout',
       'BTM 2nd Stage', 'Hoskote', 'Doddaballapur', 'Sarakki Nagar',
```

```
'Thyagaraja Nagar', 'Bharathi Nagar', 'HAL 2nd Stage',
'Kadubeesanahalli'], dtype=object)
```

In [46]:
```
len(df4.location.unique())
# SO unique we got an "other" as other distnict value oflocation
```

Out[46]:
```
242
```

In [47]:
```
df4.head(10)
```

Out[47]:

	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

In [48]:
```
#removing outliers



#like having 2bhk for total sqft of 600 this data is invariant!!
# as the average sqft of bhk for a house lies starts  from 400 sqft to 2500 sqft or

#And this can be done using standard deviation method or using domain knowledge lik
```

In [49]:
```
#thats why we remove the values having average bhk values less than 300
```

In [50]:
```
df4[df4.total_sqft/df4.bhk<300].head()
```

Out[50]:

	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

In [51]:
```
df4.shape
```

Out[51]:
```
(13246, 7)
```

In [52]:
```
df5 = df4[~(df4.total_sqft/df4.bhk<300)]
```

In [53]:
```
df5
```

Out[53]:

	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
...	...	...	...	...	...	...	...
13315	Whitefield	5 Bedroom	3453.0	4.0	231.00	5	6689.834926
13316	other	4 BHK	3600.0	5.0	400.00	4	11111.111111
13317	Raja Rajeshwari Nagar	2 BHK	1141.0	2.0	60.00	2	5258.545136
13318	Padmanabhanagar	4 BHK	4689.0	4.0	488.00	4	10407.336319
13319	Doddathoguru	1 BHK	550.0	1.0	17.00	1	3090.909091

12502 rows × 7 columns

In [54]:
```
df5.shape
```

Out[54]:
```
(12502, 7)
```

In [55]:
```
#now we removed them !!
```

In [56]:
```
#other outliers like price per sqft or any other features
```

In [57]:
```
df5.price_per_sqft.describe()
```

Out[57]:
```
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 [58]:
```
# no look at this the minimum price is of 267 then a flat where price per sqft 247
#in bengaluru is defiently an outlier!! so we remove them and take the only values
#from mean to the median of the price_per_sqft values
```

In [59]:
```
df6.shape
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[59], line 1
----> 1 df6.shape

NameError: name 'df6' is not defined
```

In [ ]:
```
#and in the data we check that 3 bedroom is for 81 lakhs
#and in other area 3 bedroom is for 327 lakhs
```

```
# so it might be due to the area where the flat is present or something else hence
#we need to somevisualizations for this cases! and remve outliers
```

In [70]:
```
def remove_pps_outlier(df):
    df_out = pd.DataFrame()
    for key, subdf  in df5.groupby('location'):
        m = np.mean(subdf.price_per_sqft)
        std = np.std(subdf.price_per_sqft)
        reduced_df = subdf[(subdf.price_per_sqft>(m-std)) & (subdf.price_per_sqft<=
        df_out = pd.concat([df_out,reduced_df],ignore_index = True)
    return df_out

df6 = remove_pps_outlier(df5)
df6.head()
```

Out[70]:

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

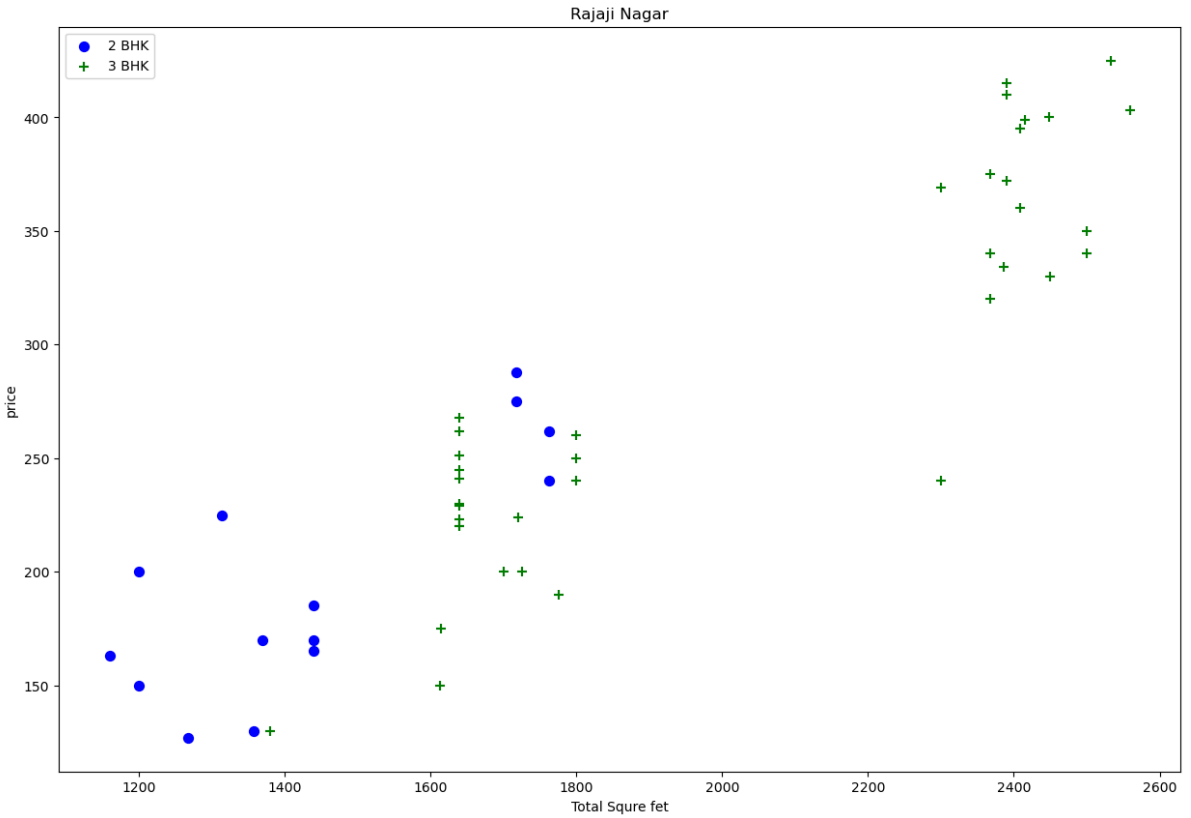
In [71]:
```
""" In your code, the rows that fall within the specified range based on the mean a

For each location, you calculate the mean (m) and standard deviation (std) of the p

Then, you filter the rows to keep only those where the price_per_sqft falls within

This condition keeps rows where the price_per_sqft is reasonably close to the mean,

The filtered rows for each location (i.e., those that fit the condition) are combir

Any row that does not satisfy this condition is not included in the final dataframe

So, the final result in df6 contains only the rows where price_per_sqft falls withi
"""
```

Out[71]:
```
' In your code, the rows that fall within the specified range based on the mean an
d standard deviation of price_per_sqft are kept, and the outliers (those that fall
outside this range) are effectively removed. Here’s how it works step by step:\n\n
For each location, you calculate the mean (m) and standard deviation (std) of the
price_per_sqft values.\n\nThen, you filter the rows to keep only those where the p
rice_per_sqft falls within one standard deviation of the mean:\n\nThis condition k
eeps rows where the price_per_sqft is reasonably close to the mean, removing outli
ers.\n\nThe filtered rows for each location (i.e., those that fit the condition) a
re combined into a new dataframe (df_out).\n\nAny row that does not satisfy this c
ondition is not included in the final dataframe, so they are effectively remove
d.\n\nSo, the final result in df6 contains only the rows where price_per_sqft fall
s within the range defined by the mean and standard deviation, and \n'
```

In [72]:
```
#Now again we still see the inconstencies in data as :
# if we have same sqft and same less no of bedrooms also we have higher price held
#flats so these are inconsistent hence they are removed by plotting the graphs and
#at the correct data points which fall in therange
```

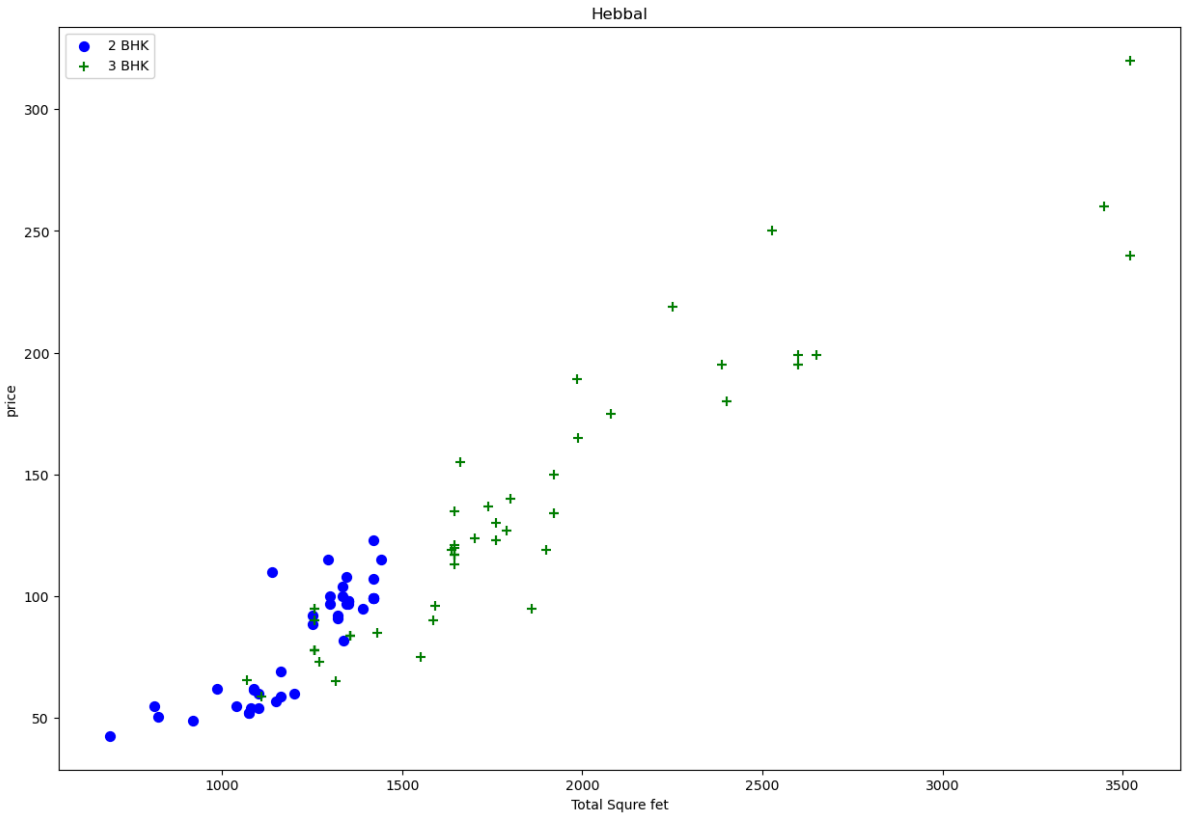
In [73]:
```
#So we use scatter plot here to observe the data
```

In [74]:
```
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
    plt.xlabel("Total Squre fet")
    plt.ylabel("price")
    plt.title(location)
    plt.legend()

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



In [75]:
```
# now check in this grapgh for square ft around 1600 to 1800 (around 1670) the pric
#for 2BHk is higher than the price of the 3 BHk hence it is outlier so we remove it
```

In [76]:
```
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
    plt.xlabel("Total Squre fet")
    plt.ylabel("price")
    plt.title(location)
    plt.legend()

plot_scatter_chart(df6,"Hebbal")
```



In [77]:
```
# So here are many such cases
```

In [78]:
```
We should also remove properties where for same location, the price of (for example

{
    '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 pri
```

```
  Cell In[78], line 6
    'std: 2000,
    ^
SyntaxError: unterminated string literal (detected at line 6)
```

In [82]:
```
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_pe
```

```
    return df.drop(exclude_indices,axis='index')
df7 = remove_bhk_outliers(df6)
# df8 = df7.copy()
df7.shape
```

Out[82]:
```
(7329, 7)
```

In [83]:
```
What happens if you remove location_df and bhk_df?
Removing location_df:
python
Copy code
for location in df.groupby('location'):
    for bhk, bhk_df in df.groupby('bhk'):
        # Do something
Error: Removing location_df would raise an error because the second for loop (for b
No context for location: You lose the context of grouping by location, so you'll er
Logic breakdown: Without location_df, the code will no longer properly group by loc
```

```
  Cell In[83], line 2
    Removing location_df:
             ^
SyntaxError: invalid syntax
```

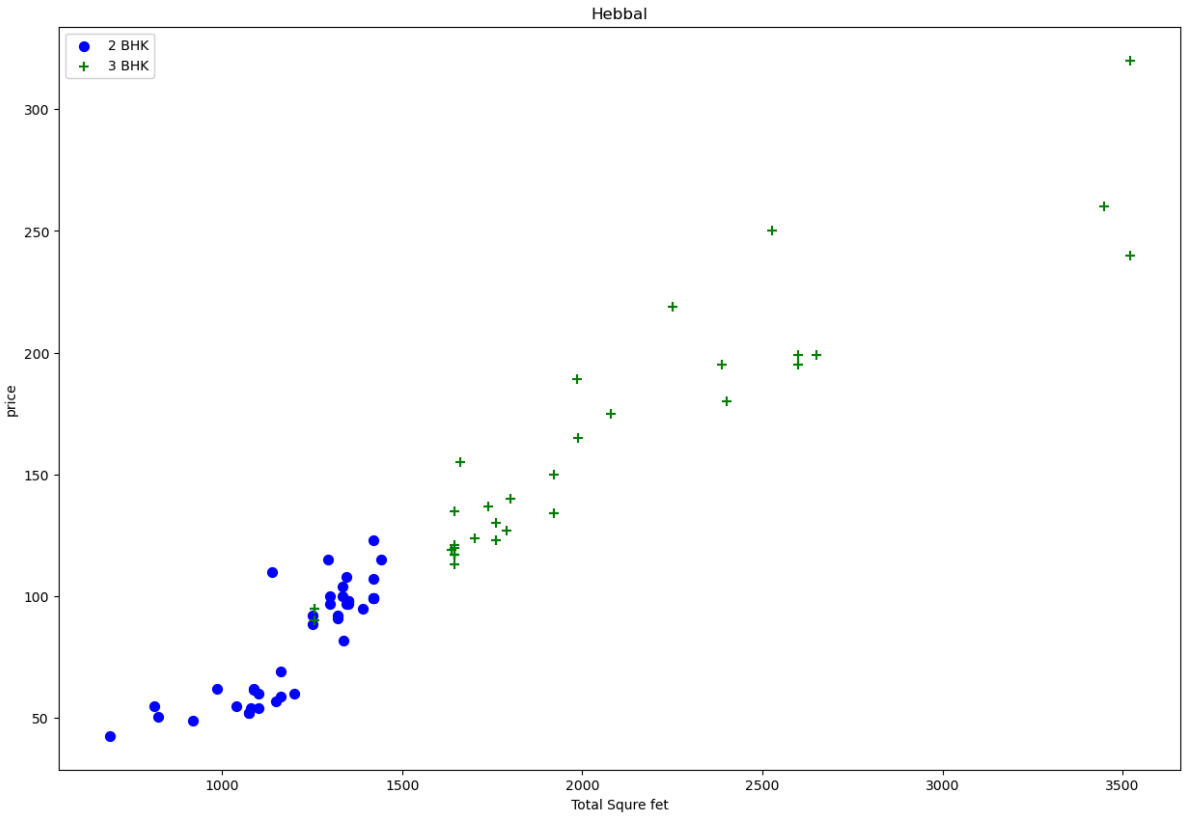
In [84]:
```
If you remove location_df and bhk_df, the code will break because it relies on thos
```

```
  Cell In[84], line 1
    If you remove location_df and bhk_df, the code will break because it relies on
those dataframes to group and filter the data properly. Removing them will lead to
errors or incorrect logic that fails to properly analyze the data based on locatio
n and BHK. These variables are essential for the correct functioning of the outlie
r removal process.
       ^
SyntaxError: invalid syntax
```

In [85]:
```
# as we see the removed the values which don't fall in the correct range so we mini
#(outlier removal)
```

In [86]:
```
#thn again we scatter plot it to see the difference
```

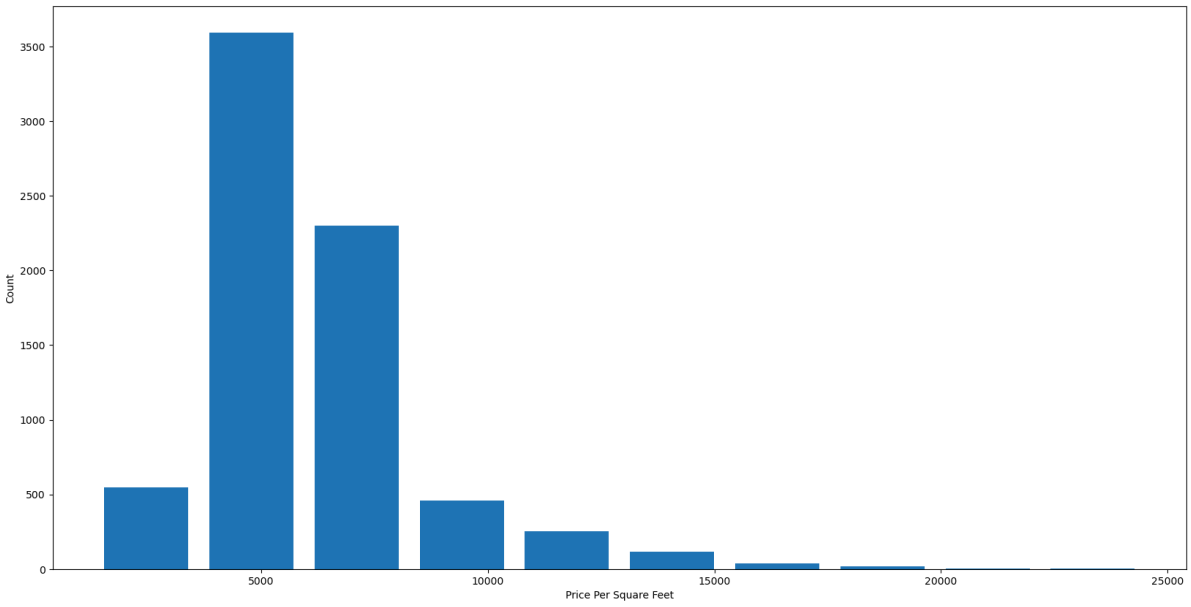
In [87]:
```
plot_scatter_chart(df7,"Hebbal")
```



In [88]:
```
#yes this is the change broo!
# still there are green but it is difficutl to precisely remove them so we leave it
#like that as our data may have some inconsistent values
```

In [89]:
```
import matplotlib
matplotlib.rcParams['figure.figsize'] = (20,10)
plt.hist(df7.price_per_sqft,rwidth=0.8)
plt.xlabel("Price Per Square Feet")
plt.ylabel("Count")
```

Out[89]:
```
Text(0, 0.5, 'Count')
```



In [90]:
```
df7.bath.unique()
```

Out[90]:
```
array([ 4.,  3.,  2.,  5.,  8.,  1.,  6.,  7.,  9., 12., 16., 13.])
```

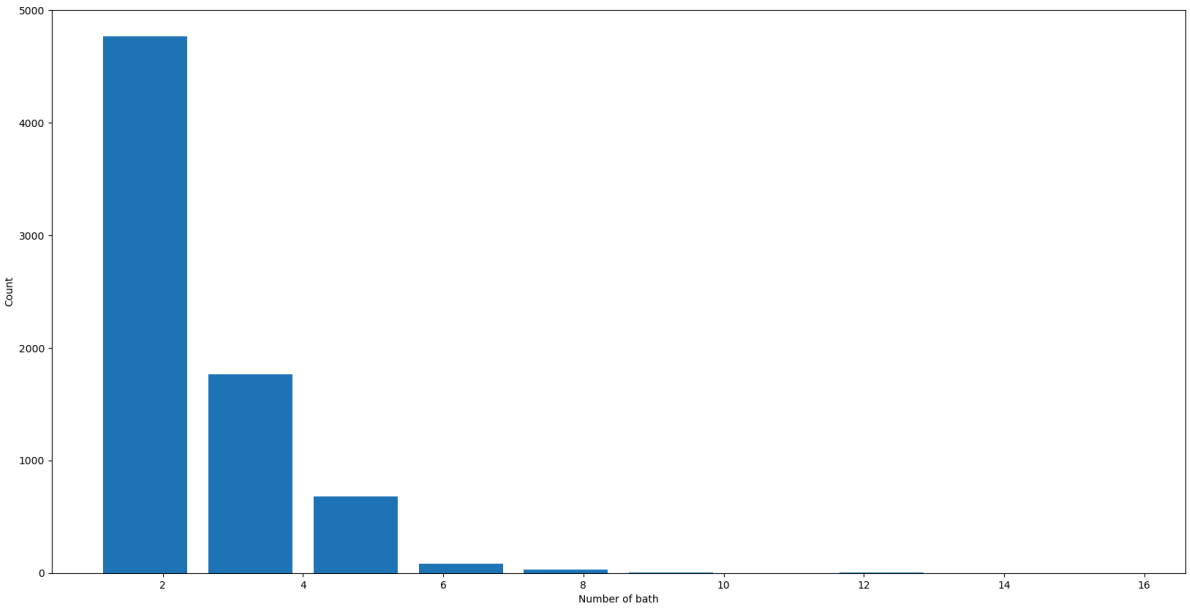
In [91]:
```
df7[df7.bath>10]
```

Out[91]:

	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 [92]:
```
plt.hist(df7.bath,rwidth=0.8)
plt.xlabel("Number of bath")
plt.ylabel("Count")
```

Out[92]:
```
Text(0, 0.5, 'Count')
```



In [93]:
```
#so here we have least number of bath as outliers (like having bathrooms more than
#So we remove here
```

In [94]:
```
df8 = df7[df7.bath<df7.bhk+2]
df8
#this is sample criteria forbedrooms ansd bathrooms
# we wil have diff criteria in real life
```

Out[94]:

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	1st Block Jayanagar	4 BHK	2850.0	4.0	428.0	4	15017.543860
1	1st Block Jayanagar	3 BHK	1630.0	3.0	194.0	3	11901.840491
2	1st Block Jayanagar	3 BHK	1875.0	2.0	235.0	3	12533.333333
3	1st Block Jayanagar	3 BHK	1200.0	2.0	130.0	3	10833.333333
4	1st Block Jayanagar	2 BHK	1235.0	2.0	148.0	2	11983.805668
...	...	...	...	...	...	...	...
10232	other	2 BHK	1200.0	2.0	70.0	2	5833.333333
10233	other	1 BHK	1800.0	1.0	200.0	1	11111.111111
10236	other	2 BHK	1353.0	2.0	110.0	2	8130.081301
10237	other	1 Bedroom	812.0	1.0	26.0	1	3201.970443
10240	other	4 BHK	3600.0	5.0	400.0	4	11111.111111

7251 rows × 7 columns

In [95]:
```
df8.shape
```

Out[95]:
```
(7251, 7)
```

In [96]:
```
#So now we maximum tried to remove our outliers and then we try to use ML model
```

In [97]:
```
#MODEL PREDICTION
```

In [98]:
```
#first we can drop uneccasary columns here like size as we already have bhk
# and here we can also remove price per sqft
```

In [99]:
```
df9 = df8.drop(['size','price_per_sqft'],axis='columns')
```

In [100…
```
df9.head()
```

Out[100]:

	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

In [101…
```
dummies = pd.get_dummies(df9.location).astype(int)
dummies
```

Out[101]:

	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	...
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	...
...	...	...	...	...	...	...	...	...	...	...	...
10232	0	0	0	0	0	0	0	0	0	0	...
10233	0	0	0	0	0	0	0	0	0	0	...
10236	0	0	0	0	0	0	0	0	0	0	...
10237	0	0	0	0	0	0	0	0	0	0	...
10240	0	0	0	0	0	0	0	0	0	0	...

7251 rows × 242 columns

In [102…

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

Out[102]:

	location	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	...	Vi
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 rows × 246 columns

In [103…

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

Out[103]:

	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	...	Vijay
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 × 245 columns

In [104…
```
df11.shape
```

Out[104]:
```
(7251, 245)
```

In [105…
```
X = df11.drop('price',axis='columns')
X
```

Out[105]:

	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	...
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	...
...	...	...	...	...	...	...	...	...	...	...	...
10232	1200.0	2.0	2	0	0	0	0	0	0	0	...
10233	1800.0	1.0	1	0	0	0	0	0	0	0	...
10236	1353.0	2.0	2	0	0	0	0	0	0	0	...
10237	812.0	1.0	1	0	0	0	0	0	0	0	...
10240	3600.0	5.0	4	0	0	0	0	0	0	0	...

7251 rows × 244 columns

In [106…
```
Y=df11.price
Y
```

Out[106]:
```
0        428.0
1        194.0
2        235.0
3        130.0
4        148.0
         ...
10232     70.0
10233    200.0
10236    110.0
10237     26.0
10240    400.0
Name: price, Length: 7251, dtype: float64
```

In [107…
```
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 [108…
```
from sklearn.linear_model import LinearRegression
lr_clf = LinearRegression()
lr_clf.fit(X_train,Y_train)
lr_clf.score(X_test,Y_test)
```

Out[108]:
```
0.845227769787429
```

In [109…
```
#yes it is a decent score but we try to also come up with different models for more
```

In [110…
```
#it is optional!! Check for code basis for grid search CV for this model or K folds
#anyhow more accurate is linear regression
```

In [113…
```
#testing the models :
X.columns
np.where(X.columns=='2nd Phase Judicial Layout')[0][0]
```

Out[113]:
```
5
```

In [114…
```
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 [115…
```
predict_price('1st Phase JP Nagar',1000,2,2)
```

```
C:\Users\P.Navya Sree\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but LinearRegression was fitted with featu
re names
  warnings.warn(
```

Out[115]:
```
83.49904677172415
```

In [116…
```
predict_price('1st Phase JP Nagar',1000,3,3)
```

```
C:\Users\P.Navya Sree\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but LinearRegression was fitted with featu
re names
  warnings.warn(
```

Out[116]:
```
86.80519395199
```

In [118…
```
predict_price('Indira Nagar',1000,2,3)
```

```
C:\Users\P.Navya Sree\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but LinearRegression was fitted with featu
re names
  warnings.warn(
```

Out[118]:
```
179.5052770758238
```

In [119…
```
"""Let's take an example dataset for predicting house prices, where the features ar

sqft (square footage),
bath (number of bathrooms),
bhk (bedrooms, halls, and kitchens),
location (one-hot encoded locations, like location_A, location_B, location_C).
Here is a sample dataset:

sqft    bath    bhk     location_A      location_B      location_C
1200    2       2       1       0       0
1500    3       3       0       1       0
900     1       1       0       0       1
The locations are one-hot encoded, meaning only one location is active (1), and the
Now, let's go step by step through your method, assuming we are using this dataset:

Method: predict_price
python
Copy code
def predict_price(location, sqft, bath, bhk):
    loc_index = np.where(X.columns == location)[0][0]  # Find the index for the loc

    x = np.zeros(len(X.columns))  # Create an array of zeros with the same length a
    x[0] = sqft  # Set the sqft value
    x[1] = bath  # Set the bath value
    x[2] = bhk   # Set the bhk value
    if loc_index >= 0:
        x[loc_index] = 1  # One-hot encode the location

    return lr_clf.predict([x])[0]  # Make the prediction using the model
Step-by-Step Breakdown:
Let's assume we want to predict the price for a house with these details:

location = 'location_B'
sqft = 1400
bath = 2
bhk = 3
1. Find Location Index:
The function first finds the index of location_B in X.columns:

python
Copy code
loc_index = np.where(X.columns == location)[0][0]
Suppose X.columns = ['sqft', 'bath', 'bhk', 'location_A', 'location_B', 'location_C

python
Copy code
loc_index = np.where(X.columns == 'location_B')[0][0]  # This returns loc_index = 4
2. Initialize the Array with Zeros:
Next, it initializes an array x with zeros:

python
Copy code
```

```
x = np.zeros(len(X.columns))  # If len(X.columns) is 6, this gives x = [0. 0. 0. 0.
3. Set sqft, bath, and bhk Values:
Now, the method sets the first three elements of the array x:

python
Copy code
x[0] = sqft  # x[0] = 1400
x[1] = bath  # x[1] = 2
x[2] = bhk   # x[2] = 3
After this, the array looks like:

python
Copy code
x = [1400. 2. 3. 0. 0. 0.]
4. One-Hot Encode the Location:
Now, the method sets the value at the loc_index (which is 4 for location_B) to 1, i

python
Copy code
if loc_index >= 0:
    x[loc_index] = 1  # Set the index for location_B to 1
After this step, the array x looks like:

python
Copy code
x = [1400. 2. 3. 0. 1. 0.]  # location_B is encoded as 1, all other locations remai
5. Predict Using the Model:
Finally, the method sends this feature array to the predict method of the model:

python
Copy code
return lr_clf.predict([x])[0]
The x array is wrapped in another list to form a 2D array [[1400, 2, 3, 0, 1, 0]],
```

Out[119]:
```
"Let's take an example dataset for predicting house prices, where the features ar
e:\n\nsqft (square footage),\nbath (number of bathrooms),\nbhk (bedrooms, halls, a
nd kitchens),\nlocation (one-hot encoded locations, like location_A, location_B, l
ocation_C).\nHere is a sample dataset:\n\nsqft\tbath\tbhk\tlocation_A\tlocation_B
\tlocation_C\n1200\t2\t2\t1\t0\t0\n1500\t3\t3\t0\t1\t0\n900\t1\t1\t0\t0\t1\nThe lo
cations are one-hot encoded, meaning only one location is active (1), and the othe
rs are inactive (0).\nNow, let's go step by step through your method, assuming we
are using this dataset:\n\nMethod: predict_price\npython\nCopy code\ndef predict_p
rice(location, sqft, bath, bhk):\n    loc_index = np.where(X.columns == location)
[0][0]  # Find the index for the location\n    \n    x = np.zeros(len(X.columns))
# Create an array of zeros with the same length as the number of columns\n    x[0]
= sqft  # Set the sqft value\n    x[1] = bath  # Set the bath value\n    x[2] = bh
k   # Set the bhk value\n    if loc_index >= 0:\n        x[loc_index] = 1  # One-h
ot encode the location\n        \n    return lr_clf.predict([x])[0]  # Make the pr
ediction using the model\nStep-by-Step Breakdown:\nLet's assume we want to predict
the price for a house with these details:\n\nlocation = 'location_B'\nsqft = 1400
\nbath = 2\nbhk = 3\n1. Find Location Index:\nThe function first finds the index o
f location_B in X.columns:\n\npython\nCopy code\nloc_index = np.where(X.columns ==
location)[0][0]\nSuppose X.columns = ['sqft', 'bath', 'bhk', 'location_A', 'locati
on_B', 'location_C']. Then:\n\npython\nCopy code\nloc_index = np.where(X.columns =
= 'location_B')[0][0]  # This returns loc_index = 4\n2. Initialize the Array with
Zeros:\nNext, it initializes an array x with zeros:\n\npython\nCopy code\nx = np.z
eros(len(X.columns))  # If len(X.columns) is 6, this gives x = [0. 0. 0. 0. 0. 0.]
\n3. Set sqft, bath, and bhk Values:\nNow, the method sets the first three element
s of the array x:\n\npython\nCopy code\nx[0] = sqft  # x[0] = 1400\nx[1] = bath  #
x[1] = 2\nx[2] = bhk   # x[2] = 3\nAfter this, the array looks like:\n\npython\nCo
py code\nx = [1400. 2. 3. 0. 0. 0.]\n4. One-Hot Encode the Location:\nNow, the met
hod sets the value at the loc_index (which is 4 for location_B) to 1, indicating t
hat this location is selected:\n\npython\nCopy code\nif loc_index >= 0:\n    x[loc
_index] = 1  # Set the index for location_B to 1\nAfter this step, the array x loo
ks like:\n\npython\nCopy code\nx = [1400. 2. 3. 0. 1. 0.]  # location_B is encoded
as 1, all other locations remain 0\n5. Predict Using the Model:\nFinally, the meth
od sends this feature array to the predict method of the model:\n\npython\nCopy co
de\nreturn lr_clf.predict([x])[0]\nThe x array is wrapped in another list to form
a 2D array [[1400, 2, 3, 0, 1, 0]], which is suitable for the model's input."
```

In [120…
```
"""np.where returns all the indices where the condition X.columns == location is Tr
```

Out[120]:
```
'np.where returns all the indices where the condition X.columns == location is Tru
e. The [0][0] part ensures that you only get the first matching index. However, if
there are multiple matching values for the location, the method will always take t
he first match and ignore the others. This works fine if your columns are unique,
as typically expected in one-hot encoding (where only one location should be 1 at
any time).'
```

In [121…
```
#Generation of pickle files :
```

In [122…
```
import pickle
with open('banglorehomeprice_predmodel.pickle','wb') as f:
    pickle.dump(lr_clf,f)
```

In [123…
```
#so here we are creating a pickle file to store the coefficents, intercept
#for this linear regression model !!
#Apart from this we need to store the values of X.columns as we need to have data
#of all the columnsof locations so we try to store them in JSON files
```

In [126…
```
import json
columns={
    'data_columns' : [col.lower() for col in X.columns]
}
with open("columns.json","w") as f:
    f.write(json.dumps(columns))
```

```
#created a json file
```

In [ ]: