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)
```

In [3]:

```
df1 = pd.read_csv("Bengaluru_House_Data.csv")
```

In [4]:

```
df1.head()
```

Out[4]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	pric
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.0
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.0
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.0
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.0
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.0

In [5]:

```
df1.groupby('area_type')['area_type'].agg('count') #counting each sub parts of area type
```

Out[5]:

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

In [5]:

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

In [7]:

```
df2.head()
```

Out[7]:

	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 [8]:

```
df2.isnull().sum() #finding no of rows where the values are NA
```

Out[8]:

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

In [9]:

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

Out[9]:

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

In [10]:

```
df3.shape
```

Out[10]:

```
(13246, 5)
```

In [11]:

```
df3['BHK'] = df3['size'].apply(lambda x: int(x.split(' ')[0])) #creating a row o store only
```

```
<ipython-input-11-c3e7b9c1045d>: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://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  df3['BHK'] = df3['size'].apply(lambda x: int(x.split(' ')[0])) #creating a
row o store only the bhk values in a single format.
```

In [12]:

```
df3.head()
```

Out[12]:

	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 [13]:

```
df3.total_sqft.unique()
```

Out[13]:

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

In [14]:

```
def is_float(x): #to find the rows under total_sqft which are there as ranges
    try:
        float(x)
    except:
        return False
    return True
```

In [15]:

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

Out[15]:

	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
...	...	...	...	...	...	...
12975	Whitefield	2 BHK	850 - 1060	2.0	38.190	2
12990	Talaghattapura	3 BHK	1804 - 2273	3.0	122.000	3
13059	Harlur	2 BHK	1200 - 1470	2.0	72.760	2
13265	Hoodi	2 BHK	1133 - 1384	2.0	59.135	2
13299	Whitefield	4 BHK	2830 - 2882	5.0	154.500	4

190 rows × 6 columns

In [16]:

```
def convert_sqft_to_num(x):  #converting the ranges under total_sqft into single float numb
    tokens = x.split('-')
    if len(tokens) == 2:
        return (float(tokens[0])+float(tokens[1]))/2
    try:
        return float(x)
    except:
        return None
```

In [17]:

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

Out[17]:

	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 [18]:

```
# for accessing some particular axis/rows:
df4.loc[120] #row/axis no 120
```

Out[18]:

```
location      Thanisandra
size                3 BHK
total_sqft           1427
bath                    3
price                 120
BHK                     3
Name: 120, dtype: object
```

In [19]:

```
# creating another data frame and adding a row: price per sqft
df5 = df4.copy()
df5['price_per_sqft'] = df5['price']*100000/df5['total_sqft']
df5.head()
```

Out[19]:

	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 [20]:

```
# finding the no of data points for each location
df5.location = df5.location.apply(lambda x: x.strip())

location_stats = df5.groupby('location')['location'].agg('count').sort_values(ascending=Fal
location_stats
```

Out[20]:

```
location
Whitefield           535
Sarjapur  Road       392
Electronic City      304
Kanakpura Road       266
Thanisandra          236
                    ... 
LIC Colony             1
Kuvempu Layout         1
Kumbhena Agrahara      1
Kudlu Village,         1
1 Annasandrapalya      1
Name: location, Length: 1293, dtype: int64
```

In [21]:

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

Out[21]:

```
location
BTM 1st Stage          10
Basapura               10
Sector 1 HSR Layout    10
Naganathapura          10
Kalkere                10
                       ..
LIC Colony              1
Kuvempu Layout          1
Kumbhena Agrahara       1
Kudlu Village,          1
1 Annasandrapalya       1
Name: location, Length: 1052, dtype: int64
```

In [22]:

```
# putting the locations which have less than 10 data points under 'other' section
df5.location = df5.location.apply(lambda x: 'other' if x in location_stats_less_than_10 els
len(df5.location.unique())
```

Out[22]:

```
242
```

In [24]:

```
df5.head()
```

Out[24]:

	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 [27]:

```
## outlier removal
# listing out the houses based on a threshold sq ft( 300 sqft per bed room)
df5[df5.total_sqft/df5.BHK<300].head()
```

Out[27]:

	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 [28]:

```
# removing these data as these are irrelevant using the negate (~) function
df6 = df5[~(df5.total_sqft/df5.BHK<300)]
```

In [29]:

```
df6.shape
```

Out[29]:

```
(12502, 7)
```

In [31]:

```
# removing price per sqft outliers
def remove_pps_outlier(df):
    df_out = pd.DataFrame()
    for key, subdf in df.groupby('location'):
        m = np.mean(subdf.price_per_sqft)
        st = np.std(subdf.price_per_sqft)
        reduced_df = subdf[(subdf.price_per_sqft>(m-st)) & (subdf.price_per_sqft<=(m+st))]
        df_out = pd.concat([df_out, reduced_df],ignore_index=True)
    return df_out
df7 = remove_pps_outlier(df6)
df7.shape
```

Out[31]:

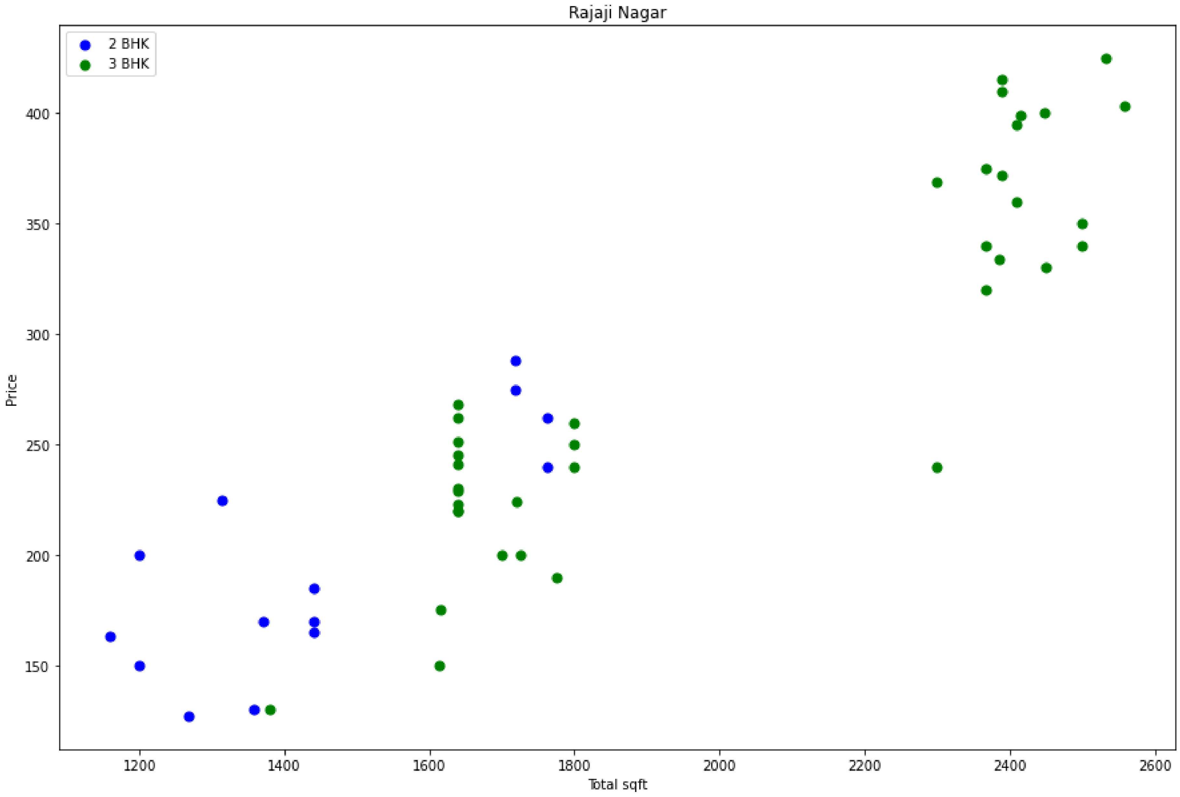
```
(10241, 7)
```

In [36]:

```
# making a scatter plot to show price of almost same sqft 2 and 3 bhk flats at a specific l
# we will then remove the outliers, i.e, price of 2 bhk more than 3 bhk for same sqft.

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,color = 'green', label = '3 BHK', s=50)
    plt.xlabel('Total sqft')
    plt.ylabel('Price')
    plt.title(location)
    plt.legend()

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



In [41]:

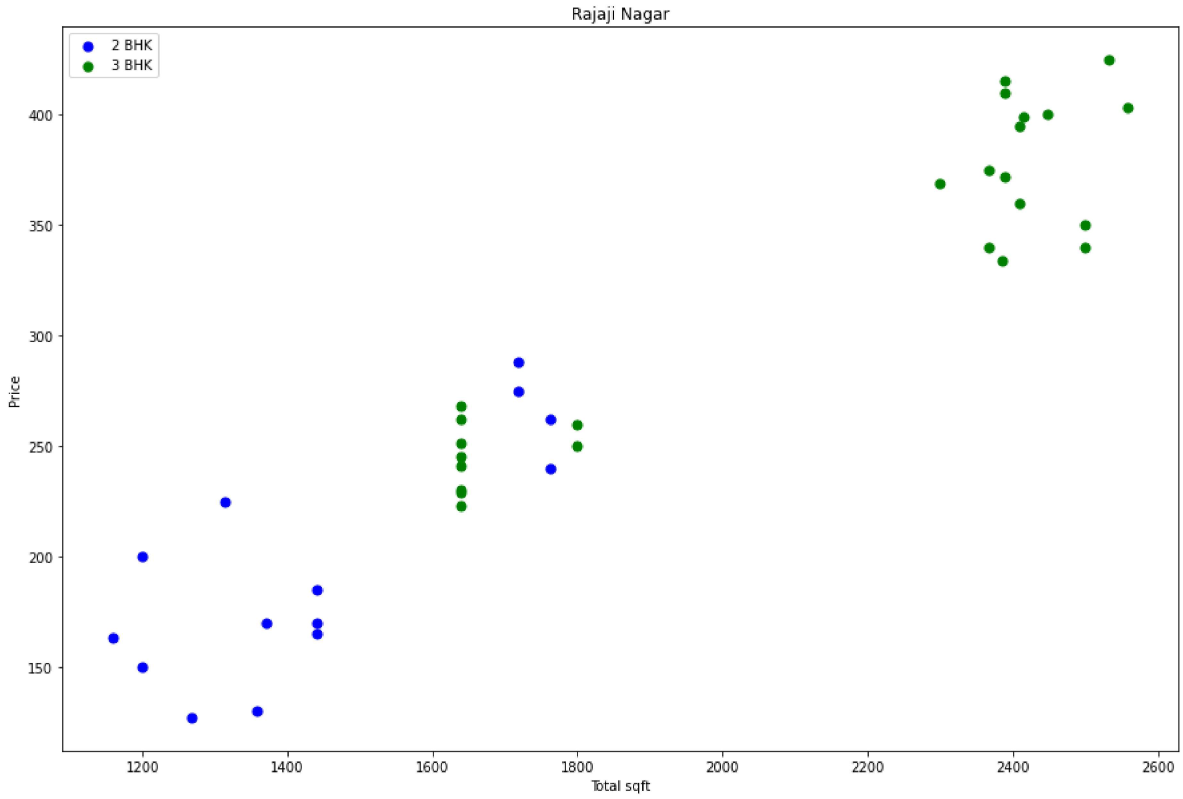
```
## removing these outliers
def remove_bhk_outliers(df):
    exclude_indices = np.array([])
    for location, location_df in df.groupby('location'):
        BHK_stats = {}
        for BHK, BHK_df in location_df.groupby('BHK'):
            BHK_stats[BHK] = {
                'mean': np.mean(BHK_df.price_per_sqft),
                '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<(
    return df.drop(exclude_indices,axis='index')
df8 = remove_bhk_outliers(df7)
# df8 = df7.copy()
df8.shape
```

Out[41]:

```
(7329, 7)
```

In [42]:

```
plot_scatter_chart(df8,"Rajaji Nagar")
```

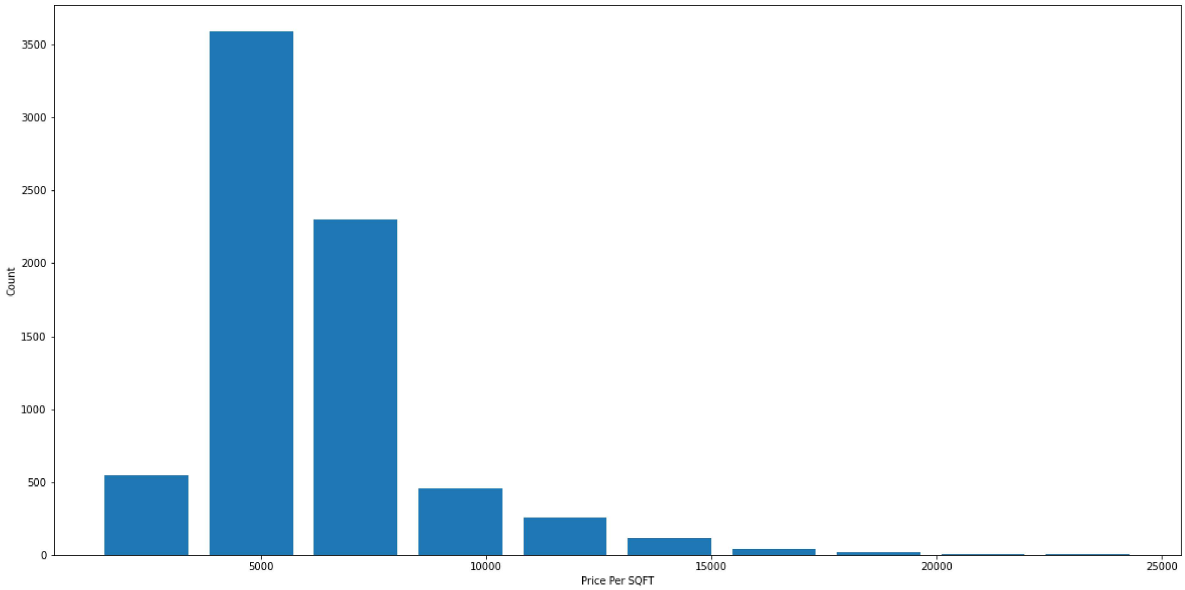


In [43]:

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

Out[43]:

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



In [44]:

```
# removing bathroom outliers
df8.bath.unique()
```

Out[44]:

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

In [45]:

```
df8[df8.bath>10]
```

Out[45]:

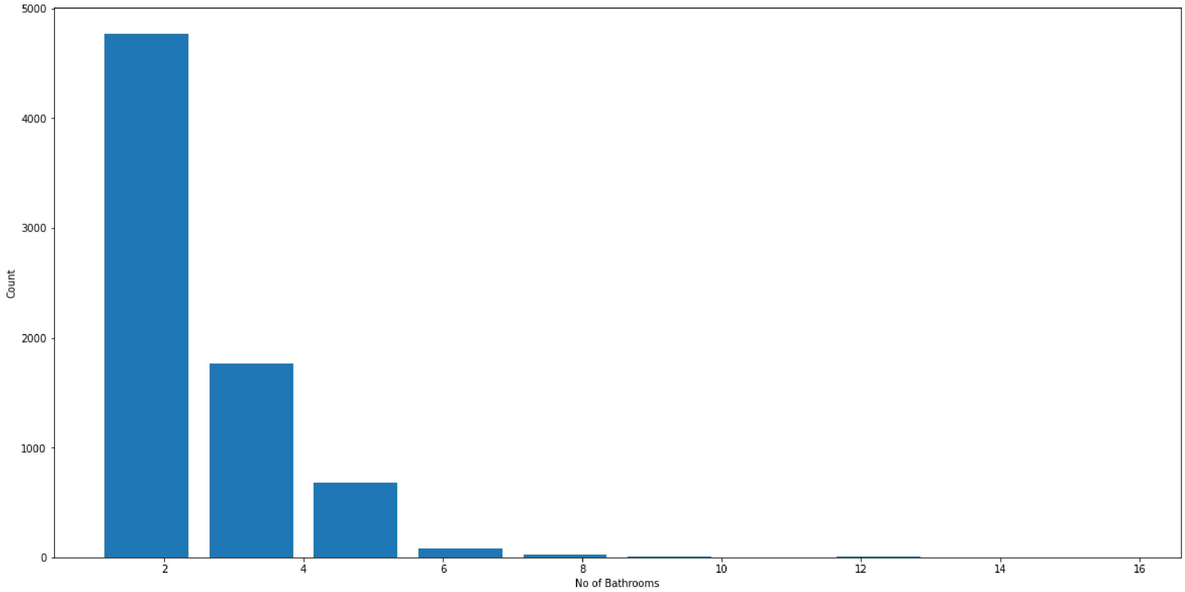
	location	size	total_sqft	bath	price	BHK	price_per_sqft
5277	Neeladri Nagar	10 BHK	4000.0	12.0	160.0	10	4000.000000
8486	other	10 BHK	12000.0	12.0	525.0	10	4375.000000
8575	other	16 BHK	10000.0	16.0	550.0	16	5500.000000
9308	other	11 BHK	6000.0	12.0	150.0	11	2500.000000
9639	other	13 BHK	5425.0	13.0	275.0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

In [46]:

```
plt.hist(df8.bath,rwidth = 0.8)  # rwidth is the width of the bar
plt.xlabel("No of Bathrooms")
plt.ylabel("Count")
```

Out[46]:

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



In [47]:

```
df8[df8.bath>df8.BHK+2]
```

Out[47]:

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

In [48]:

```
df9 = df8[df8.bath<df8.BHK+2]
df9.shape
```

Out[48]:

```
(7251, 7)
```

In [58]:

```
# removing the columns which are not needed for the machine learning model
df10 = df9.drop(['size','price_per_sqft'],axis = 'columns')
df10.head(3)
```

Out[58]:

	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

In [61]:

```
# converting the location into a dummies column as ML can only support numeric values
## ONE HOT ENCODING for Location
dummies = pd.get_dummies(df10.location)
dummies.head(3)
```

Out[61]:

	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 rows × 242 columns

In [63]:

```
df11 = pd.concat([df10,dummies.drop('other',axis = 'columns')],axis = 'columns')
df11.head(3)
```

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	...
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 [64]:

```
# dropping the location column as we already have the dummy columns for all the locations
df12 = df11.drop('location',axis = 'columns')
df12.head(3)
```

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	...	V
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 rows × 245 columns

In [65]:

```
# creating a variable to store all the independent values
## dropping price column as its dependent
X = df12.drop('price', axis = 'columns')
X.head(3)
```

Out[65]:

	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 rows × 244 columns

In [66]:

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

Out[66]:

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

In [67]:

```
# dividing the dataset into training and test dataset
# Training dataset - For model training
# Test dataset - To evaluate the performance of the model
# test size = 0.2 means 20% of the dataset is used for test sample and remaining 80% as tra
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 [69]:

```
from sklearn.linear_model import LinearRegression
lr_clf = LinearRegression()
lr_clf.fit(X_train,y_train)
lr_clf.score(X_test,y_test) #finding the accuracy of our model
```

Out[69]:

```
0.8452277697874312
```

In [71]:

```
# K Fold Cross Validation method
# Shufflesplit will randomize the dataset such that each fold has equal amount of data

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[71]:

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

In [72]:

```
# using grid search CV to find which regression method is best suited for our dataset
## regressions like linear, lasso,decision-tree

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=Fal
        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[72]:

	model	best_score	best_params
0	linear_regression	0.818354	{'normalize': True}
1	lasso	0.687436	{'alpha': 1, 'selection': 'random'}
2	decision_tree	0.730093	{'criterion': 'friedman_mse', 'splitter': 'best'}

In [ ]: