Data Science Regression Project: Predicting Home Prices in Bangalore

Dataset Link: https://www.kaggle.com/amitabhajoy/bengaluru-house-price-data

Data Load

In [14]:
```
df1 = pd.read_csv("bengaluru_house_prices.csv")
df1.head()
```

Out[14]:

	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

Drop features that are not required to build our model

In [19]:
```
df2 = df1.drop(['area_type','society','balcony','availability'],axis='columns')
df2.shape
```

Out[19]:
```
(13320, 5)
```

Data Cleaning

1. Handle NA values

In [20]:
```
df2.isnull().sum()
```

Out[20]:
```
location       1
size          16
total_sqft     0
bath          73
price          0
dtype: int64
```

In [21]:
```
df2.shape
```

Out[21]:
```
(13320, 5)
```

In [22]:
```
df3 = df2.dropna()
df3.isnull().sum()
```

Out[22]:
```
location      0
size          0
total_sqft    0
bath          0
price         0
dtype: int64
```

In [23]:
```
df3.shape
```

Out[23]:
```
(13246, 5)
```

Feature Engineering

1. Adding a new feature(integer) for bhk (Bedrooms Hall Kitchen)

In [24]:
```
df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0]))
df3.bhk.unique()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py: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: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-v
iew-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[24]:
```
array([ 2,  4,  3,  6,  1,  8,  7,  5, 11,  9, 27, 10, 19, 16, 43, 14, 12,
        13, 18], dtype=int64)
```

2. Exploring total_sqft feature

In [25]:
```
def is_float(x):
    try:
        float(x)
    except:
        return False
    return True
```

In [26]:
```
2+3
```

Out[26]:
```
5
```

In [27]:
```
df3[~df3['total_sqft'].apply(is_float)].head(10)
```

Out[27]:

	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 [28]:
```
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 [29]:
```
df4 = df3.copy()
df4.total_sqft = df4.total_sqft.apply(convert_sqft_to_num)
df4 = df4[df4.total_sqft.notnull()]
df4.head(2)
```

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

3. **Adding new feature called price per square feet**

In [32]:
```
df5 = df4.copy()
df5['price_per_sqft'] = df5['price']*100000/df5['total_sqft']
df5.head()
```

Out[32]:

	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

4. **Examining locations which is a categorical variable. (We need to apply dimensionality reduction technique here to reduce number of locations)**

In [35]:
```
df5.location = df5.location.apply(lambda x: x.strip())
location_stats = df5['location'].value_counts(ascending=False)
location_stats
```

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 [40]:
```
location_stats_less_than_10 = location_stats[location_stats<=10]
location_stats_less_than_10
```

Outlier Removal

5. **Using Business Logic**

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 [44]:
```
df5[df5.total_sqft/df5.bhk<300].head()
```

Out[44]:

	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 [46]:
```
df6 = df5[~(df5.total_sqft/df5.bhk<300)]
df6.shape
```

Out[46]: (12456, 7)

6. Using Standard Deviation and Mean

In [47]:
```
df6.price_per_sqft.describe()
```

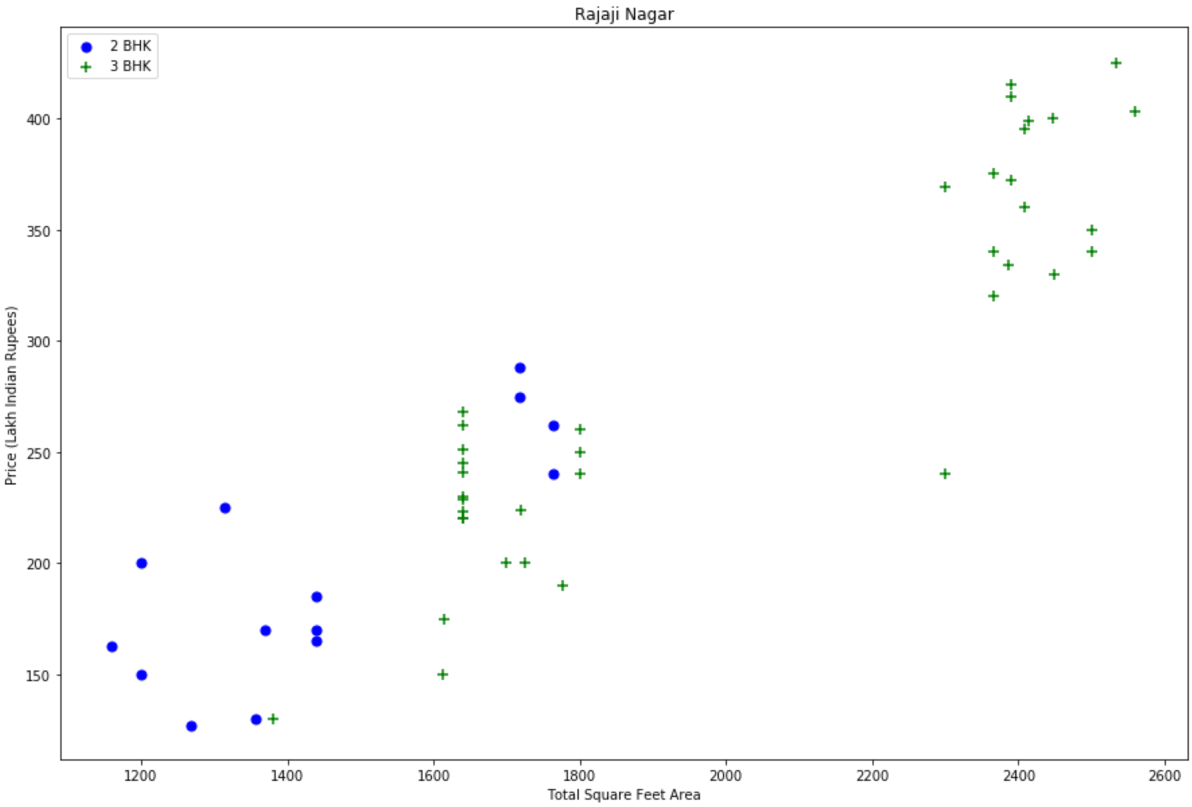
Out[47]:
```
count     12456.000000
mean       6308.502826
std        4168.127339
min         267.829813
25%        4210.526316
50%        5294.117647
75%        6916.666667
max      176470.588235
Name: price_per_sqft, dtype: float64
```

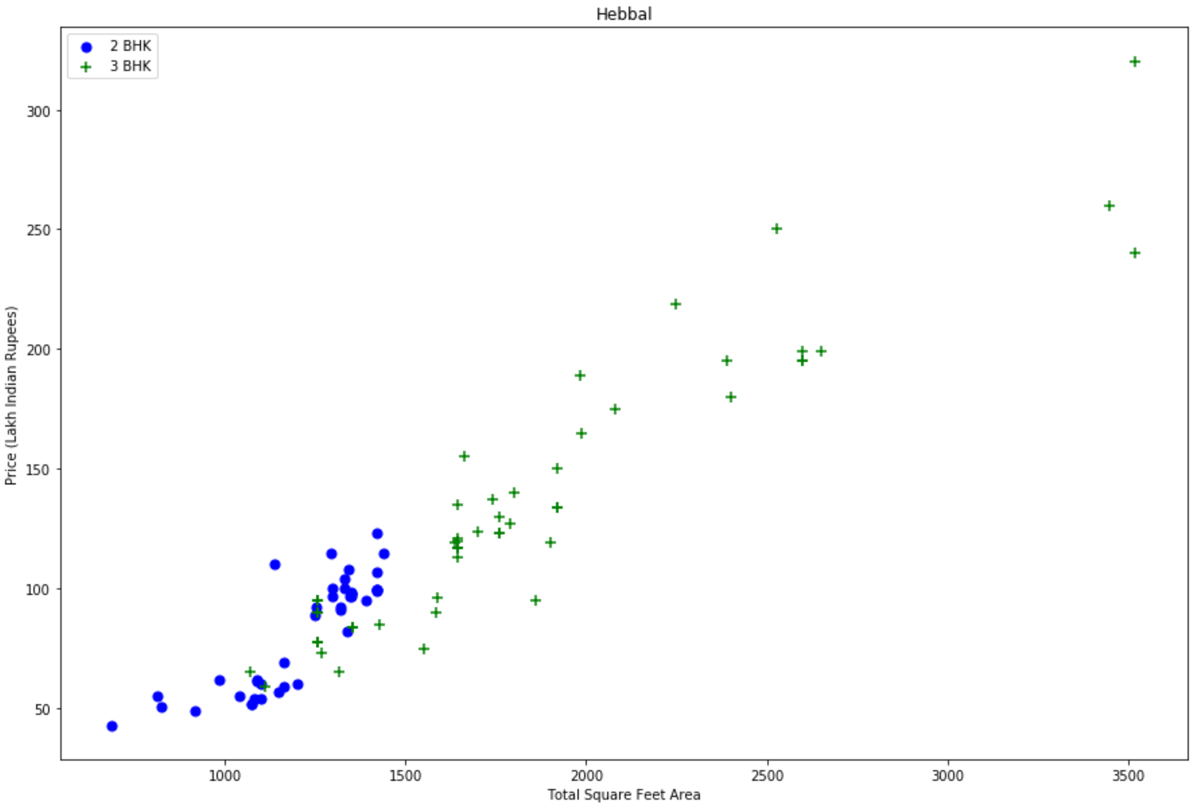
Here we find that min price per sqft is 267 rs/sqft whereas max is 176470, this shows a wid e variation in property prices. We should remove outliers per location using mean and one standard deviation

In [48]:
```
def remove_pps_outliers(df):
    df_out = pd.DataFrame()
    for key, subdf in df.groupby('location'):
        m = np.mean(subdf.price_per_sqft)
        st = np.std(subdf.price_per_sqft)
        reduced_df = subdf[(subdf.price_per_sqft>(m-st)) & (subdf.price_per_sqft<=(m+st))]
        df_out = pd.concat([df_out,reduced_df],ignore_index=True)
    return df_out
df7 = remove_pps_outliers(df6)
df7.shape
```

Out[48]: (10242, 7)

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





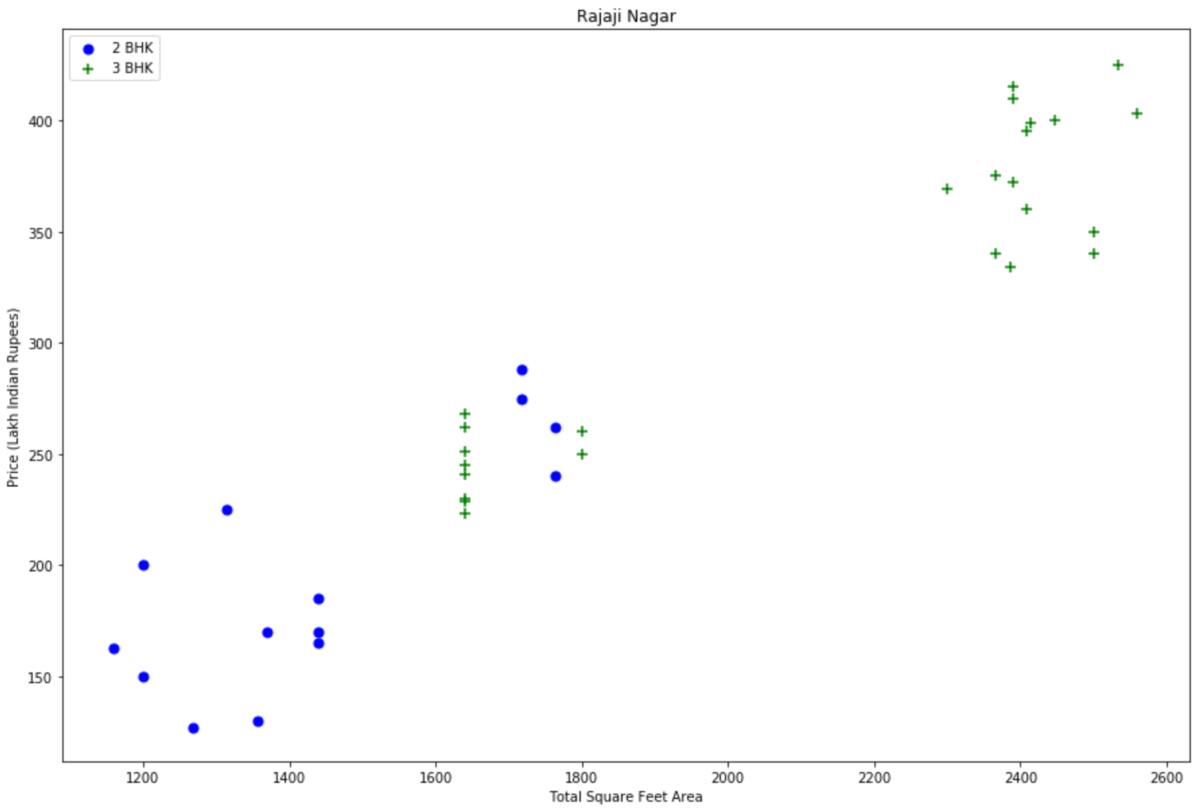
We should also remove properties where for same location, the price of (for example) 3 bedroom apartment is less than 2 bedroom apartment (with same square ft area). What we will do is for a given location

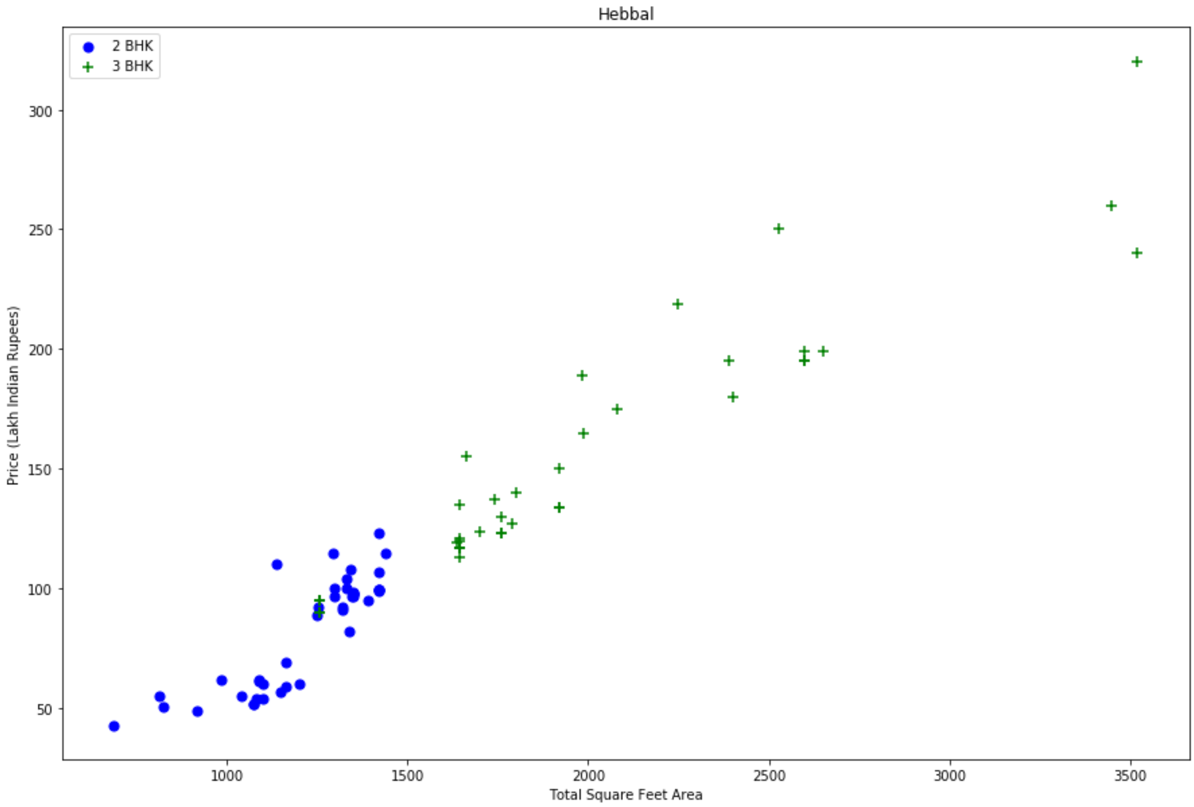
Now we can remove those 2 BHK apartments whose price_per_sqft is less than mean price_per_sqft of 1 BHK apartment

In [51]:
```
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['mea
n'])].index.values)
    return df.drop(exclude_indices,axis='index')
df8 = remove_bhk_outliers(df7)
# df8 = df7.copy()
df8.shape
```

Out[51]: (7317, 7)

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





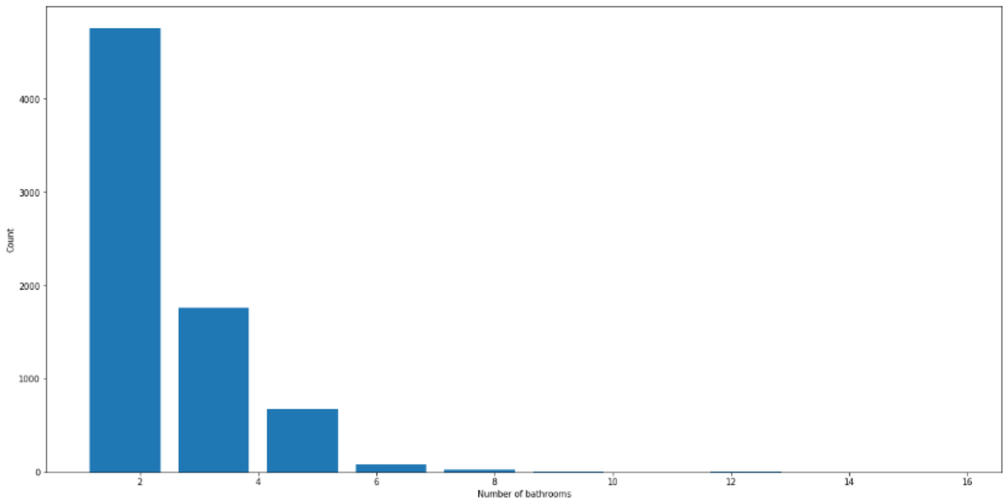
7. Using Bathrooms Feature

In [55]:
```
df8.bath.unique()
```

Out[55]: `array([ 4.,  3.,  2.,  5.,  8.,  1.,  6.,  7.,  9., 12., 16., 13.])`

In [56]:
```
plt.hist(df8.bath,rwidth=0.8)
plt.xlabel("Number of bathrooms")
plt.ylabel("Count")
```

Out[56]: `Text(0, 0.5, 'Count')`



In [57]:
```
df8[df8.bath>10]
```

Out[57]:

	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

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

In [58]:
```
df8[df8.bath>df8.bhk+2]
```

Out[58]:

	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

If we have 4 bedroom home and even if we have bathroom in all 4 rooms plus one guest bathroom, we will have total bath = total bed + 1 max. Anything above that is an outlier or a data error and can be removed

In [59]:
```
df9 = df8[df8.bath<df8.bhk+2]
df9.shape
```

Out[59]: (7239, 7)

In [60]:
```
df9.head(2)
```

Out[60]:

	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

In [61]:
```
df10 = df9.drop(['size','price_per_sqft'],axis='columns')
df10.head(3)
```

Out[61]:

	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

Using One Hot Encoding For Location

In [62]:
```
dummies = pd.get_dummies(df10.location)
dummies.head(3)
```

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	...	Vishveshwarya Layout	Vishwapriya Layout
0	1	0	0	0	0	0	0	0	0	0	...	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0

3 rows × 241 columns

In [63]:
```
df11 = pd.concat([df10,dummies.drop('other',axis='columns')],axis='columns')
df11.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	...	Vijayanagar	Vishveshwarya Layout
0	1st Block Jayanagar	2850.0	4.0	428.0	4	1	0	0	0	0	...	0	0
1	1st Block Jayanagar	1630.0	3.0	194.0	3	1	0	0	0	0	...	0	0
2	1st Block Jayanagar	1875.0	2.0	235.0	3	1	0	0	0	0	...	0	0
3	1st Block Jayanagar	1200.0	2.0	130.0	3	1	0	0	0	0	...	0	0
4	1st Block Jayanagar	1235.0	2.0	148.0	2	1	0	0	0	0	...	0	0

5 rows × 245 columns

In [64]:
```
df12 = df11.drop('location',axis='columns')
df12.head(2)
```

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	...	Vijayanagar	Vishveshwarya Layout	Vi La
0	2850.0	4.0	428.0	4	1	0	0	0	0	0	...	0	0	0
1	1630.0	3.0	194.0	3	1	0	0	0	0	0	...	0	0	0

2 rows × 244 columns

Build a Model

In [57]:

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

...

In [73]:

```
Y = df12.price
Y=Y.astype('int')
Y.head(3)
```

...

In [74]:

```
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 [66]:

```
# Algorithms
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
```

Stochastic Gradient Descent (SGD):

In [91]:

```
sgd = linear_model.SGDClassifier(max_iter=5, tol=None)
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)

sgd.score(X_train, Y_train)

acc_sgd = round(sgd.score(X_test, Y_test) * 100, 2)
```

Linear Regression

In [92]:

```
linreg = LinearRegression()
linreg.fit(X_train,Y_train)
Y_pred = linreg.predict(X_test)

linreg.score(X_train, Y_train)


acc_linreg = round(linreg.score(X_test, Y_test) * 100, 2)
```

Logistic Regression

In [93]:

```
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

Y_pred = logreg.predict(X_test)

acc_log = round(logreg.score(X_test, Y_test) * 100, 2)
```

```
C:\Users\NIKHIL SALUNKHE\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will b
e changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\NIKHIL SALUNKHE\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: FutureWarning: Default multi_class w
ill be changed to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

KNN

In [94]:

```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_test, Y_test) * 100, 2)
```

Gaussian Naive Bayes

In [95]:
```
gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_test, Y_test) * 100, 2)
```

Perecepton

In [96]:
```
perceptron = Perceptron(max_iter=5)
perceptron.fit(X_train, Y_train)

Y_pred = perceptron.predict(X_test)

acc_perceptron = round(perceptron.score(X_test, Y_test) * 100, 2)
```

```
C:\Users\NIKHIL SALUNKHE\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarning: Maxim
um number of iteration reached before convergence. Consider increasing max_iter to improve the fit.
  ConvergenceWarning)
```

Support Vector Machine

In [97]:
```
linear_svc = LinearSVC()
linear_svc.fit(X_train, Y_train)

Y_pred = linear_svc.predict(X_test)

acc_linear_svc = round(linear_svc.score(X_test, Y_test) * 100, 2)
```

```
C:\Users\NIKHIL SALUNKHE\Anaconda3\lib\site-packages\sklearn\svm\base.py:929: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Decision Tree

In [98]:
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_test, Y_test) * 100, 2)
```

Random Forest

In [99]:
```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)

Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_test, Y_test) * 100, 2)
```

Best Model ?

In [100]:
```
results = pd.DataFrame({
    'Model': ['Linear Regression','Logistic Regression','KNN','Support Vector Machines',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent',
              'Decision Tree'],
    'Score': [acc_linreg, acc_log, acc_knn, acc_linear_svc,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_decision_tree]})
result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df.head(9)
```

click to expand output; double click to hide output

Score	Model
86.30	Linear Regression
14.09	Random Forest
14.09	Decision Tree
10.50	KNN
6.35	Logistic Regression
1.80	Perceptron
1.59	Naive Bayes
0.62	Support Vector Machines
0.28	Stochastic Gradient Decent

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

In [103]:
```
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[103]: array([0.82708703, 0.86040393, 0.85340127, 0.84375035, 0.85496201])

Find best model using GridSearchCV

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit


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[102]:

	model	best_score	best_params
0	linear_regression	0.847796	{'normalize': False}
1	lasso	0.726860	{'alpha': 2, 'selection': 'random'}
2	decision_tree	0.715907	{'criterion': 'friedman_mse', 'splitter': 'best'}