A Project on

House Price Prediction Model

For

Feynn Labs

Batch 7-SB3

Submitted by

Pawan Narote **Ranapratap Ghosh**

Tejas Prakash **Vaibhav Poojary**

Machine Learning Intern Engineer

Batch 7-SB3

Team T-1-1

2021-2022

Abstract

In this, we will implement a Bangalore House Price Prediction model using a Machine Learning algorithm. This model predicts the price of Bangalore's house with the help of a few parameters like availability, size, total square feet, bath, location, etc.

During this Bengaluru House Price prediction using Machine Learning tutorial you will learn several things like, Exploratory data analysis, dealing with a missing values or noisy data, Data preprocessing, create new features from existing features, remove outliers, Data visualization, Splitting data into the training and testing , Train linear regression model and test. We have trained a Bengaluru House Price prediction model using linear regression algorithm.

Regression analysis

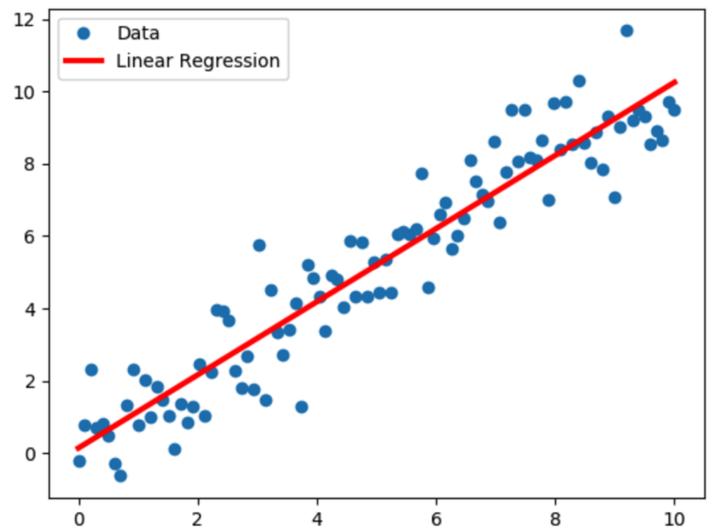
In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships between a dependent variable and one or more independent variables. Regression analysis consists of a set of machine learning methods that allow us to predict a continuous outcome variable (y) based on the value of one or multiple predictor variables (x). It assumes a linear relationship between the outcome and the predictor variables.

Regression analysis is a fundamental concept in the field of machine learning. It falls under supervised learning wherein the algorithm is trained with both input features and output labels. It helps in establishing a relationship among the variables by estimating how one variable affects the other

Linear Regression Model for Machine Learning

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering, and the number of independent variables being used. Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.

In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.



$$y = \theta_1 + \theta_2 . x$$

While training the model we are given :

x: input training data (univariate – one input variable(parameter))

y: labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.

θ1: intercept

θ2: coefficient of x

Once we find the best θ1 and θ2 values, we get the best fit line. So, when we are finally using our model for prediction, it will predict the value of y for the input value of x.

Making Predictions with Linear Regression

Given the representation is a linear equation, making predictions is as simple as solving the equation for a specific set of inputs.

Let's make this concrete with an example. Imagine we are predicting weight (y) from height (x). Our linear regression model representation for this problem would be:

y = B0 + B1 * x1

or

weight =B0 +B1 * height

Where B0 is the bias coefficient and B1 is the coefficient for the height column. We use a learning technique to find a good set of coefficient values. Once found, we can plug in different height values to predict the weight.

For example, let's use B0 = 0.1 and B1 = 0.5. Let's plug them in and calculate the weight (in kilograms) for a person with the height of 182 centimeters.

weight = 0.1 + 0.5 * 182

weight = 91.1

You can see that the above equation could be plotted as a line in two-dimensions. The B0 is our starting point regardless of what height we have. We can run through a bunch of heights from 100 to 250 centimeters and plug them to the equation and get weight values, creating our line.

Preparing Data for Linear Regression

Linear regression is been studied at great length, and there is a lot of literature on how your data must be structured to make best use of the model. As such, there is a lot of sophistication when talking about these requirements and expectations which can be intimidating. In practice, you can use these rules more as rules of thumb when using Ordinary Least Squares Regression, the most common implementation of linear regression.

Linear Assumption: Linear regression assumes that the relationship between your input and output is linear. It does not support anything else. This may be obvious, but it is good to remember when you have a lot of attributes. You may need to transform data to make the relationship linear (e.g., log transform for an exponential relationship).

Remove Noise: Linear regression assumes that your input and output variables are not noisy. Consider using data cleaning operations that let you better expose and clarify the signal in your data. This is most important for the output variable and you want to remove outliers in the output variable (y) if possible.

Remove Collinearity: Linear regression will over-fit your data when you have highly correlated input variables. Consider calculating pairwise correlations for your input data and removing the most correlated.

Gaussian Distributions: Linear regression will make more reliable predictions if your input and output variables have a Gaussian distribution. You may get some benefit using transforms (e.g., log or BoxCox) on your variables to make their distribution more Gaussian looking.

Rescale Inputs: Linear regression will often make more reliable predictions if you rescale input variables using standardization or normalization.

Dataset

What are the things that a potential home buyer considers before purchasing a house? The location, the size of the property, vicinity to offices, schools, parks, restaurants, hospitals or the stereotypical white picket fence? What about the most important factor — the price?

For example, for a potential homeowner, over 9,000 apartment projects and flats for sale are available in the range of ₹42-52 lakh, followed by over 7,100 apartments that are in the ₹52-62 lakh budget segment, says a report by property website Makaan. According to the study, there are over 5,000 projects in the ₹15-25 lakh budget segment followed by those in the ₹34-43 lakh budget category.

Buying a home, especially in a city like Bengaluru, is a tricky choice. While the major factors are usually the same for all metros, there are others to be considered for the Silicon Valley of India. With its millennial crowd, vibrant culture, great climate and a slew of job opportunities, it is difficult to ascertain the price of a house in Bengaluru.

Raw Dataset

area_type	availability	location	size	society	total_sqft	bath	balcony	price
Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2	1	39.07
Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5	3	120
Built-up Area	Ready To Move	Uttarahalli	3 BHK		1440	2	3	62
Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3	1	95
Super built-up Area	Ready To Move	Kothanur	2 BHK		1200	2	1	51
Super built-up Area	Ready To Move	Whitefield	2 BHK	DuenaTa	1170	2	1	38
Super built-up Area	18-May	Old Airport Road	4 BHK	Jaades	2732	4		204
Super built-up Area	Ready To Move	Rajaji Nagar	4 BHK	Brway G	3300	4		600
Super built-up Area	Ready To Move	Marathahalli	3 BHK		1310	3	1	63.25
Plot Area	Ready To Move	Gandhi Bazar	6 Bedroom		1020	6		370
Super built-up Area	18-Feb	Whitefield	3 BHK		1800	2	2	70

Importing Data sources

Common step is to load all the required libraries and load the Bengaluru house data set using the Pandas function read_csv() and display the top five rows of the data set using the head() method.

 # Importing all the libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

 # Importing the dataset

df = pd.read_csv('Bengaluru_House_Dataset.csv')

df.head()

- Numpy we have imported for the performing mathematics calculation.
- Matplotlib is for plotting the graph, and pandas are for managing the dataset.
- Seaborn is for data visualization library, it is based on matplotlib.

df.head()

	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

Data Pre-Processing

Now perform an Exploratory Data Analysis. In EDA, Check the shape of the data set using the shape method. It displays the number of rows and number of columns. Then display the percentage of null values like how much percent it contains NULL values. Then check the value count of the area_type column. Then drop some features (columns) which are of no use to train our model. The features which we are going to drop are availability, area_type, society, balcony. Now display the data set.

df.shape

```
(13320, 9)
```

df.isnull().mean()*100

```
area_type        0.000000
availability     0.000000
location         0.007508
size             0.120120
society         41.306306
total_sqft       0.000000
bath             0.548048
balcony          4.572072
price            0.000000
dtype: float64
```

df['area_type'].value_counts()

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

Removing the duplicate and unwanted data

Then again check if there are Null values or not. So, you can see there are some null values. Then we drop all the rows which contain null values using the method dropna(). Then check the shape of the data set and display the top 5 rows of the data set.

df.drop(columns=["availability","area_type","society","balcony"],axis=1,inplace=True)

df.head()

	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

Now check the unique values of size feature and you can see there are different types of values like in BHK, bedrooms etc. So, we write a function to extract only the starting integer values from the size feature and store it into a new bhk feature. And now you can see the size feature of the data set. Now drop the size feature which is of no use now.

df['size'].unique()

```
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)
```

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

	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

df.drop(columns=["size"],axis=1,inplace=True)

df.shape

```
(13246, 5)
```

Now it's time to remove the outliers from the BHK. firstly, check the BHK greater than 22. If it's greater than 22 which means, it's outlier. Now check the unique values of total_sqft which contain integer values (Like 2000), range values (2000-3000) and mixed data type values (2000Sq Meter).

df[df.bhk>22]

	location	total_sqft	bath	price	bhk
1718	2Electronic City Phase II	8000	27.0	230.0	27
4684	Munnekollal	2400	40.0	660.0	43

df.total_sqft.unique()

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

Now create a user defined function is_float() with the the total_sqft as an argument and return all the floating (function convert integer values into float). Then we apply a function on the total_sqft feature. But we apply this function using a tilt(~) symbol which returns all values except floating type. It means, it returns a range and mixed data type values as you can see in the below output.

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

df[~df['total_sqft'].apply(is_float)].head(10)

	location	total_sqft	bath	price	bhk
30	Yelahanka	2100 - 2850	4.0	186.000	4
122	Hebbal	3067 - 8156	4.0	477.000	4
137	8th Phase JP Nagar	1042 - 1105	2.0	54.005	2
165	Sarjapur	1145 - 1340	2.0	43.490	2
188	KR Puram	1015 - 1540	2.0	56.800	2
410	Kengeri	34.46Sq. Meter	1.0	18.500	1

Now implement a convert_sqft_into_number() function which takes a total_sqft feature as an argument and if the type of value if integer then simply convert into float and return, if the type of value is range then take an average of both and return, if the type of value is mixed data type then return None because this type of value is only one in total_sqft feature. Then apply it on the total_sqft feature.

Then create a new feature price_per_sqft from the existing feature price and total_sqft. And display the data.

```
def convert_sqft_into_number(x):
    token = x.split('-')
    if len(token) == 2:
        return (float(token[0]) + float(token[1])) / 2
    try:
        return float(x)
    except:
        return None
df1 = df.copy()
df1['total_sqft'] = df1['total_sqft'].apply(convert_sqft_into_number)
df2 = df1.copy()
df2['price_per_sqft'] = df2['price']*100000 / df2['total_sqft']
df2.head()
```

	location	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	1056.0	2.0	39.07	2	3699.810606
1	Chikka Tirupathi	2600.0	5.0	120.00	4	4615.384615
2	Uttarahalli	1440.0	2.0	62.00	3	4305.555556
3	Lingadheeranahalli	1521.0	3.0	95.00	3	6245.890861
4	Kothanur	1200.0	2.0	51.00	2	4250.000000

Now display the value counts of the location feature and create an anonymous function to remove the spaces from the left side and right side. After removing the spaces, you can see the count of location. Before Removing the spaces, the count was 1304 and after removing the spaces, the count was 1293.

df2['location'].value_counts()

```
Whitefield                                534
Sarjapur  Road                            392
Electronic City                           302
Kanakpura Road                            266
Thanisandra                               233
                                         ...
Escorts Colony                              1
Nagarbhavi  BDA Complex                     1
Bande Nallasandra                           1
RMV extension stage 2, rmv extension        1
MEI layout, Bagalgunte                      1
Name: location, Length: 1304, dtype: int64
```

df2['location'] = df2['location'].apply(lambda x: x.strip())

df2.location.value_counts()

```
Whitefield               535
Sarjapur  Road           392
Electronic City          304
Kanakpura Road           266
Thanisandra              236
                        ...
Amrita Nagar               1
Deepanjali Nagar           1
KHB Colony Extension       1
1 Giri Nagar               1
NR Colony                  1
Name: location, Length: 1293, dtype: int64
```

Create a new variable loc_less_than_10. It contains locations which are less than 10.

loc_stats[loc_stats<=10]

loc_less_than_10 = loc_stats[loc_stats<=10]

loc_less_than_10

```
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
```

Then create an anonymous function which applies to the location. This function returns all the locations where the count of location is greater than 10, if the count of location is less than 10 then return 'other'. Now the unique location becomes 242 from 1293. Now remove outliers from the bhk features. All bhk removed from where bhk less than 300.

df2.location = df2.location.apply(lambda x: 'other' if x in loc_less_than_10 else x)

df2.head()

	location	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	1056.0	2.0	39.07	2	3699.810606
1	Chikka Tirupathi	2600.0	5.0	120.00	4	4615.384615
2	Uttarahalli	1440.0	2.0	62.00	3	4305.555556
3	Lingadheeranahalli	1521.0	3.0	95.00	3	6245.890861
4	Kothanur	1200.0	2.0	51.00	2	4250.000000

len(df2.location.unique())

242

df2[(df2.total_sqft / df2.bhk < 300)].head()

	location	total_sqft	bath	price	bhk	price_per_sqft
9	other	1020.0	6.0	370.0	6	36274.509804
45	HSR Layout	600.0	9.0	200.0	8	33333.333333
58	Murugeshpalya	1407.0	4.0	150.0	6	10660.980810
68	Devarachikkanahalli	1350.0	7.0	85.0	8	6296.296296
70	other	500.0	3.0	100.0	3	20000.000000

Now describe a price_per_sqft feature and in this, you can see the outlier. House price is 176470. Lakh which is not possible according to location and total square feet. So, create a function remove_outlier_from_price_per_sqft(). It takes a dataset and uses a Standard Deviation technique to remove outliers. After applying this function.

```
df3 = df2[ ~(df2.total_sqft / df2.bhk < 300) ]
df3.shape
```

```
(12502, 6)
```

```
df3.price_per_sqft.describe()
```

```
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
```

```
def remove_outlier_from_price_per_sqft(df):
    df_out = pd.DataFrame()
    for key,sub in df.groupby('location'):
        m = np.mean( sub.price_per_sqft )
        st = np.std( sub.price_per_sqft )
        reduce_df = sub[( sub.price_per_sqft>(m-st) ) & ( sub.price_per_sqft<=(m+st) ) ]
        df_out = pd.concat( [df_out, reduce_df],ignore_index=True )
    return df_out
df4 = remove_outlier_from_price_per_sqft(df3)
```

df4.shape

(10241, 6)

df4.describe()

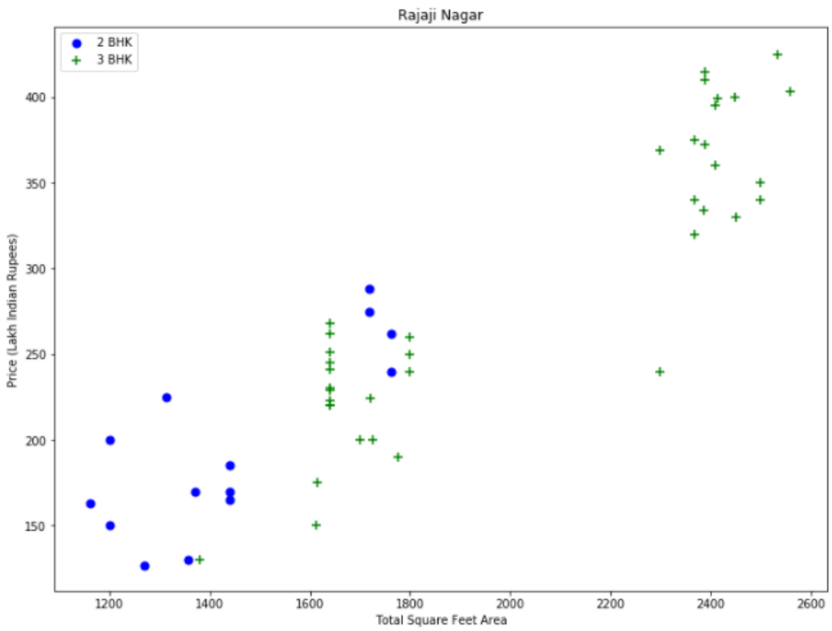
	total_sqft	bath	price	bhk	price_per_sqft
count	10241.000000	10241.000000	10241.000000	10241.000000	10241.000000
mean	1503.877034	2.474075	90.982730	2.572210	5657.702572
std	876.716232	0.981338	86.147549	0.896219	2266.476980
min	300.000000	1.000000	10.000000	1.000000	1250.000000
25%	1108.000000	2.000000	49.000000	2.000000	4244.762955
50%	1282.000000	2.000000	67.000000	2.000000	5172.413793
75%	1650.000000	3.000000	100.000000	3.000000	6426.099852
max	30400.000000	16.000000	2200.000000	16.000000	24509.803922

Analysis

Visualizing location wise using scatter chart

Now visualize the “Rajaji Nagar” location with 2 bhk and 3 bhk. 2 bhk is in blue color and 3 bhk is in green color. So, you can see in the below graph that the 3 bhk house price is less than the 2 bhk house price.

```
def plot_scatter_chart(df,location):
    bhk2 = df[(df.location==location) & (df.bhk==2)]
    bhk3 = df[(df.location==location) & (df.bhk==3)]
    plt.rcParams['figure.figsize'] = (12,9)
    plt.scatter(bhk2.total_sqft,bhk2.price,color='blue',label='2 BHK', s=50)
    plt.scatter(bhk3.total_sqft,bhk3.price,marker='+', color='green',label='3 BHK', s=50)
    plt.xlabel("Total Square Feet Area")
    plt.ylabel("Price (Lakh Indian Rupees)")
    plt.title(location)
    plt.legend()
plot_scatter_chart(df4,"Rajaji Nagar")
```

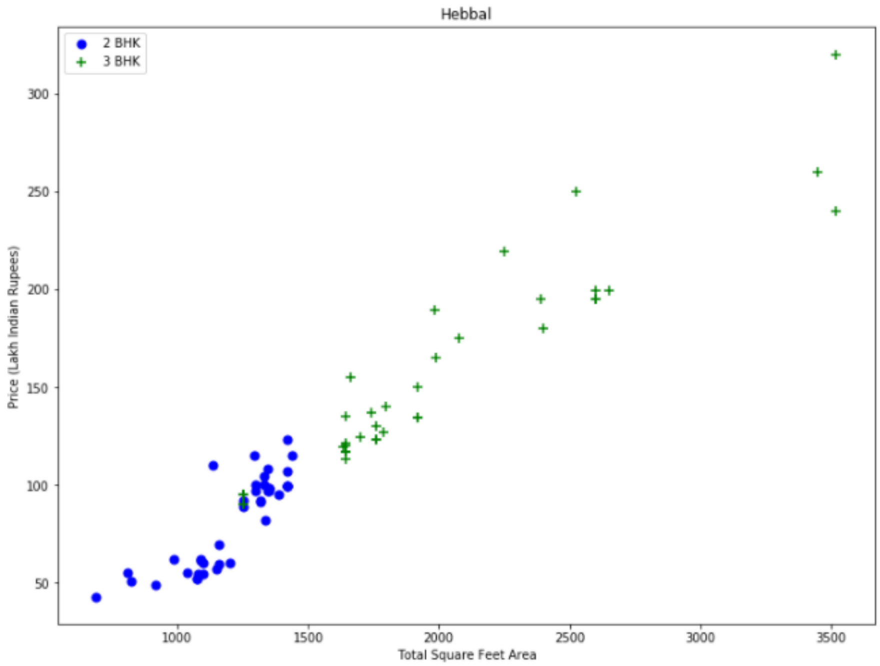


No again use a Standard Deviation technique to remove the outliers from the price_per_sqft.

```
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['mean'])].index.values)
    return df.drop(exclude_indices,axis='index')
df5 = remove_bhk_outliers(df4)
df5.shape
```

Now again visualize the same graph and now you can see that 3 bhk house price is higher than the 2 bhk house price. Some 3 bhk house prices can be less than the 2 bhk price because of the location.

plot_scatter_chart(df5,"Hebbal")



Now checking the unique values of the bath and you can see; it contains a 16 bath in one house which makes no sense. Now display the houses who have greater than 10 baths.

df5.bath.unique()

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

df5[df5.bath>10]

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

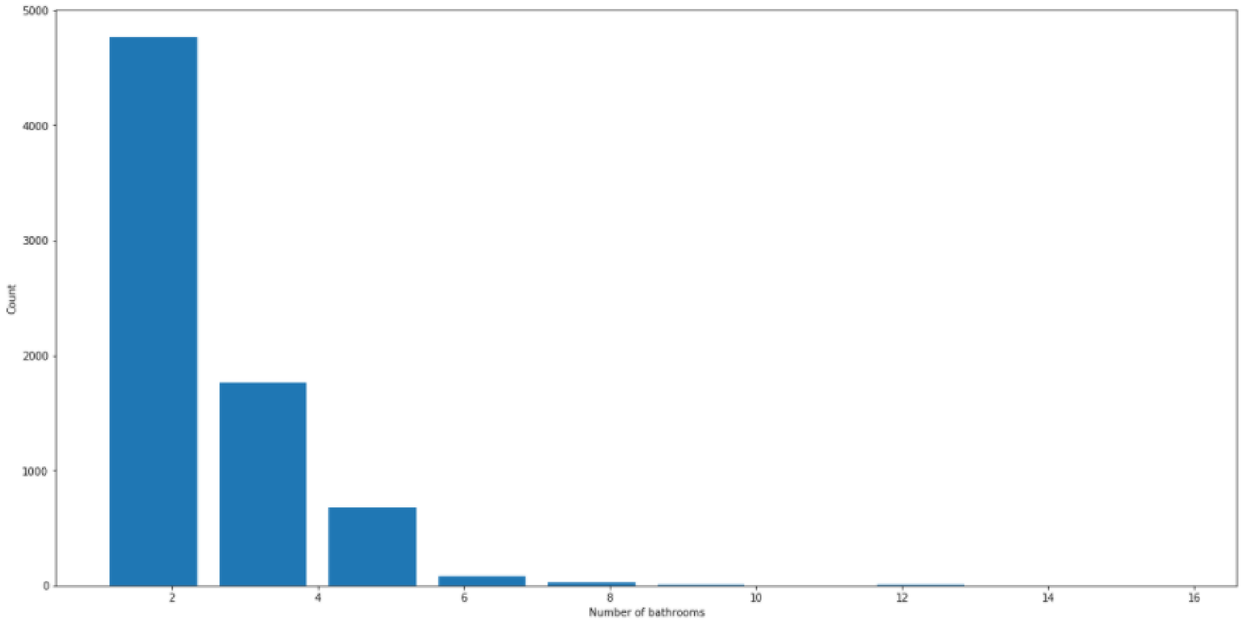
Visualizing flat wise using Histogram

Now visualize the number of baths using a histogram graph.

plt.hist(df5.bath,rwidth=0.8)

plt.xlabel("Number of bathrooms")

plt.ylabel("Count")



Applying Hot Encoding

Keep only those houses who have only less than bhk-1. For example: if a house is of 4 bhk, then it contains only 3 baths (bhk-1). Now check the shape of the data and now the data set contains 7325 rows and 6 columns.

Now drop a price_per_sqft which is of no use and display the final data and now it still contains a categorical feature (location).

df5[(df5.bath > df5.bhk+2)]

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

df6 = df5[~(df5.bath > df5.bhk+2)]

df6.head()

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

df7 = df6.drop(['price_per_sqft'],axis='columns')

df7.head()

	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

Now apply a one hot encoding to convert a categorical feature into numeric feature. And store into a "dummies" data set.

Now concat dummies data set with our final data set and remove a "other" column from "dummies" data set. We can identify a "other" location like if all locations are "0" then automatically "other" is "1".

dummies = pd.get_dummies(df7.location)

dummies.head()

	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	Vittasandra	Whitefield	Yelachenahal
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

5 rows × 242 columns

df8 = pd.concat([df7,dummies.drop('other',axis='columns')],axis='columns')

df8.head()

	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	Vishwapriya Layout	Vittasandra	V
0	1st Block Jayanagar	2850.0	4.0	428.0	4	1	0	0	0	0	...	0	0	0	0	
1	1st Block Jayanagar	1630.0	3.0	194.0	3	1	0	0	0	0	...	0	0	0	0	
2	1st Block Jayanagar	1875.0	2.0	235.0	3	1	0	0	0	0	...	0	0	0	0	
3	1st Block Jayanagar	1200.0	2.0	130.0	3	1	0	0	0	0	...	0	0	0	0	
4	1st Block Jayanagar	1235.0	2.0	148.0	2	1	0	0	0	0	...	0	0	0	0	

5 rows × 246 columns

Then see the final data set. But it contains a location feature which is of no use now. So, drop the location and display the final preprocessed data set. Then check the shape of the final data set.

df8.drop('location',axis='columns',inplace=True)

df8.head()

	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	Vishwapriya Layout	Vittasandra	Whit
0	2850.0	4.0	428.0	4	1	0	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	0	
2	1875.0	2.0	235.0	3	1	0	0	0	0	0	...	0	0	0	0	
3	1200.0	2.0	130.0	3	1	0	0	0	0	0	...	0	0	0	0	
4	1235.0	2.0	148.0	2	1	0	0	0	0	0	...	0	0	0	0	

5 rows × 245 columns

Training and Testing Model

Now it's time to prepare the data set. Data set is split into the independent and dependent features and stored into the “x” and “y” data set. And check the shape of “x” and “y” as you can see below.

Then split the data set into the training and testing using the train_test_split() method which returns 4 data sets as you can see in the below image. Then check the shape of all four data sets.

Now define our linear regression model and train the model using the training data set and check the score of the model using the validation data sets.

x = df8.drop('price',axis=1)

y = df8['price']

x.shape

(7325, 244)

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=101)

X_train.shape, X_test.shape, y_train.shape, y_test.shape

((5860, 244), (1465, 244), (5860,), (1465,))

from sklearn.linear_model import LinearRegression

lr = LinearRegression()

lr.fit(X_train,y_train)

lr.score(X_test,y_test)

0.8629898728935371

Now test the model using the testing data set and after testing you can see our model predicts below values and you can also see the actual values.

pred = lr.predict(X_test)

pred

y_test

```
7892     41.745
3357    380.000
126      75.000
3767    175.000
4871     80.000
          ...
9870    120.000
9802     87.000
2955    113.000
917      65.000
748      59.520
Name: price, Length: 1465, dtype: float64
```

Create a function to test the model on a custom data set which takes the location, sqft, bath, bhk, etc. So, I tested a model on 3 custom data sets as you can see in the below image. Now save a model using a joblib library with the name “banglore house price prediction model.pkl”.

```
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.predict([X])[0]
```

```
predict_price('1st Phase JP Nagar',1000, 2, 2)
```

85.2974569797724

```
predict_price('1st Phase JP Nagar',1000, 2, 3)
```

81.70512816315643

```
predict_price('Indira Nagar',1400, 2, 3)
```

217.40708528156847

```
import joblib

joblib.dump(lr, "banglore house price prediction model.pkl")
```

References

- *Hastie, Friedman, and Tibshirani, The Elements of Statistical Learning, 2001*
- *Bishop, Pattern Recognition and Machine Learning, 2006*
- *Ripley, Pattern Recognition and Neural Networks, 1996*
- *Hastie et al, Bishop, and Duda et al. all have chapters on LDA, logistic regression, and other linear classifiers.*
- *www.kaggle.com*
- *www.google.com*
- *www.youtube.com*