# **PREDICTING DELHI HOUSE PRICING USING THE CONCEPT OF LINEAR REGRESSION**

**PROJECT BASED LAB**

**SUB. CODE: MDS-227**

Submitted by

BHUMIKA GUPTA(214104016)

PRANAVA P(214104015)

GOVIND GUPT(214104017)

Under the guidance

# DR. C.K.VERMA SIR



DEPARTMENT OF MATHEMATICS, BIOINFORMATICS AND COMPUTER APPLICATION

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY

BHOPAL(MP)-462003

***ABSTRACT:***

Linear regression is a widely used machine learning algorithm used for predicting continuous target variables. In this project, we will implement a simple linear regression model from scratch using Python libraries like numpy, pandas, matplotlib, and seaborn and apply it on the Delhi house pricing dataset to predict the price of a house based on its features. The dataset contains information about different features of houses in Delhi like area, location, BHK, etc. and their prices. The aim of this project is to build a model that can predict the price of a house based on its features.

***INTRODUCTION:***

**Regression:**

There are situations in daily life where we want to know the relationship between various factors for example if the price of petrol increases would it affect the sales of cars or does change the location of the house will it affect the price. The process of finding this relationship between multitudes of factors is known as regression in more formal words regression refers to the study of the nature of the relationship between the variables so that one may be able to predict the unknown value of one variable for a known value of another variable.

To better understand the regression we need to be familiar with two terms

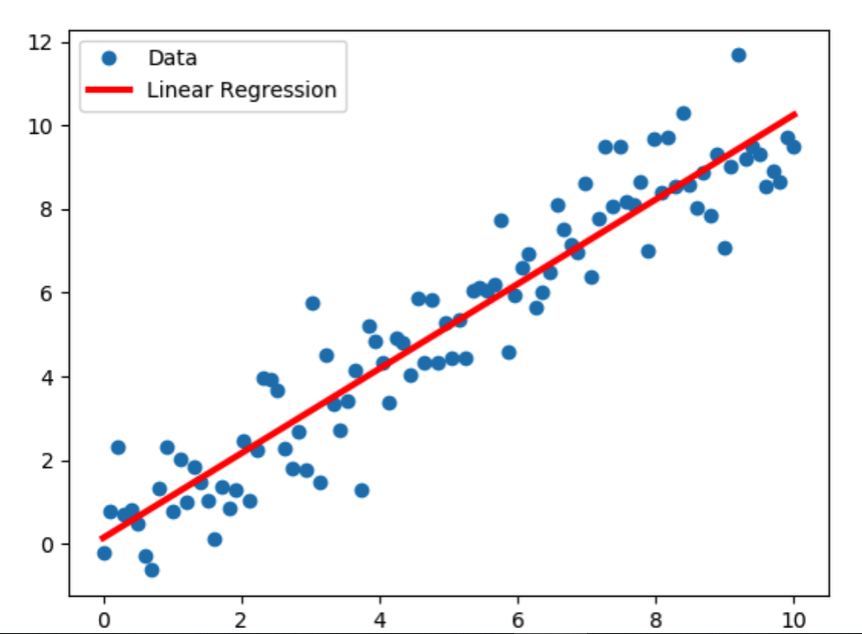
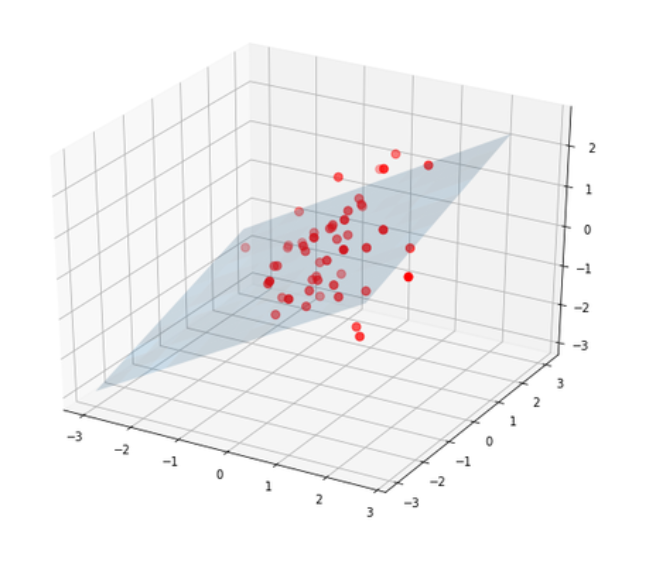
**Independent variable:-** These refer to the factors or the variables based upon which the situation is analyzed. These variables don't change and are used to indicate details about the dependent variable.

**Dependent variable**:- It refers to the result i.e. the sales of cars or the price of the house. In formal terms, it is the factor which is affected based on the independent variables.

# **Linear Regression:**

# On a dataset, if the end goal is to find some sort of linear relationship between the independent variable so that when an unknown point is given a prediction can be made. In the case where there is only 1 independent feature, it is called Uni-variate linear regression and if there are multiple features it is called Multiple linear regression

In the case of Univariate linear regression a line is formed on a 2-D plane that shows the linear relationship between the independent and dependent variables. While in the case of multivariate linear regression a hyperplane is formed.



A regression model tries to find a function which would suitably fit the training data and predict with accuracy the training data

In the case of linear regression it is a linear function of the form:

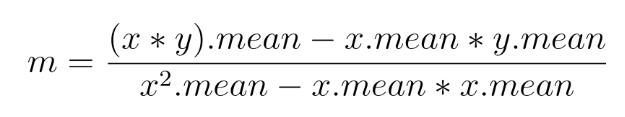
y= m1x1 + m2x2 + m3x3 ..... + b

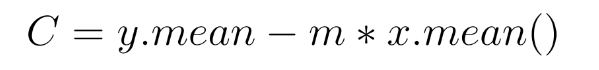
* Where y denotes the predicted value (dependent variable)
* b is called the bias term or the offset
* m1, m2 ,m3 are called the modal parameters (these are the values which our modal tries to learn using the training data)
* x1, x2, x3 are called the feature values (independent variables)

# **How do we decide best-fit line?**

what does the best fit line actually represent? It acts as the line which tells us for a particular value of x what might be the value of y. So the best fit line is y = mx + c and the regression model needs to calculate the value of m and c using the training data set

But it is possible that the actual value of y might not be equal to the predicted value in that case the difference between the predicted value and the actual value is called **residuals**. So the best fit line is a line which divides the data into equal parts along with minimum residuals. The values of m and c are found using differentiation and the results are:





Now we know what are the values of m and c in best fit line so we can try to code linear regression ourselves

***PROCESS:***

#*First import the libraries*

import numpy as np

import pandas as pd

#*now read the csv file*

[*https://drive.google.com/file/d/16FBEZz4XLnVRP\_d-M6g7bL6rCws7rxFk/view?usp=share\_link*](https://drive.google.com/file/d/16FBEZz4XLnVRP_d-M6g7bL6rCws7rxFk/view?usp=share_link)

df=pd.read\_csv(‘ ’)

*# here it will read the csv file*

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1259 entries, 0 to 1258

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Area 1259 non-null float64

1 BHK 1259 non-null int64

2 Bathroom 1257 non-null float64

3 Furnishing 1254 non-null object

4 Locality 1259 non-null object

5 Parking 1226 non-null float64

6 Price 1259 non-null int64

7 Status 1259 non-null object

8 Transaction 1259 non-null object

9 Type 1254 non-null object

10 Per\_Sqft 1018 non-null float64

dtypes: float64(4), int64(2), object(5)

memory usage: 108.3+ KB

*#checking null values*

df.isnull().sum()

Area 0

BHK 0

Bathroom 2

Furnishing 5

Locality 0

Parking 33

Price 0

Status 0

Transaction 0

Type 5

Per\_Sqft 241

dtype: int64

# **Exploratory Data Analysis:**

import seaborn as sns

import matplotlib.pyplot as plt

### *#So lets start the Data Analysis:*

df['Area'].value\_counts()

900.0 67

1500.0 50

1800.0 48

1000.0 42

1600.0 38

..

530.0 1

972.0 1

324.0 1

4800.0 1

11050.0 1

Name: Area, Length: 315, dtype: int64

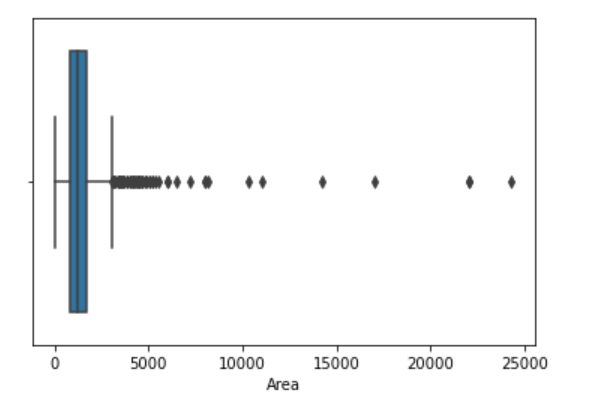
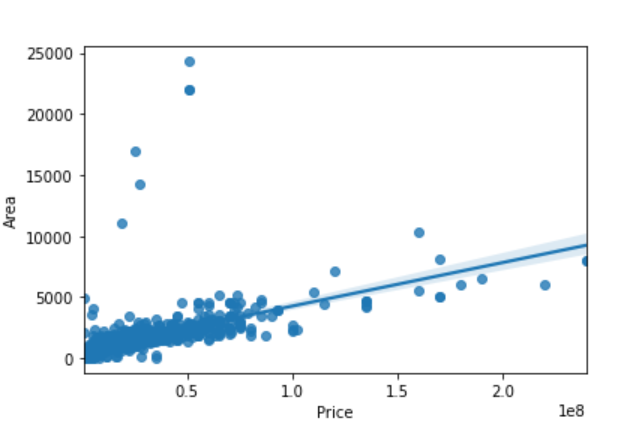
## To get a clear view about Area, we can see these plots below.

## Which clearly says that Area values below 3000 sqft are more frequent.

sns.regplot(y='Area',x='Price',data=df)

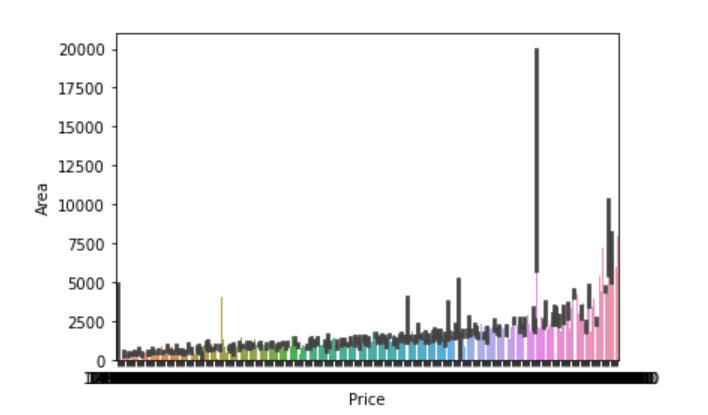
sns.boxplot(df["Area"])

plt.show()



sns.barplot(df["Price"],df['Area'])

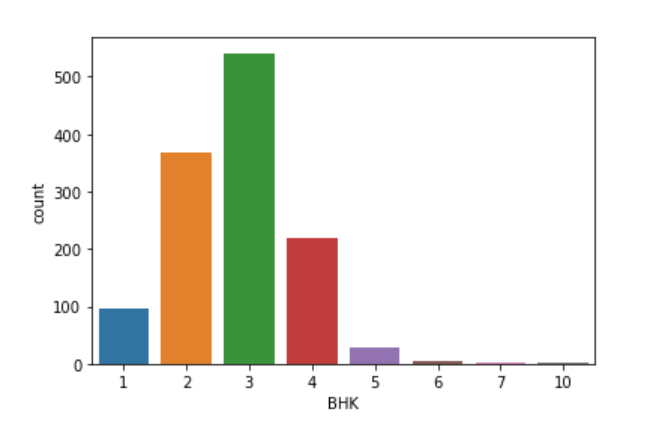
plt.show()



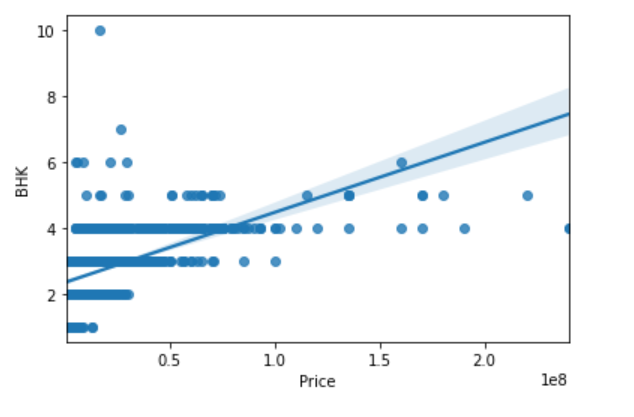
#now doing for BHK:

sns.countplot(df["BHK"])

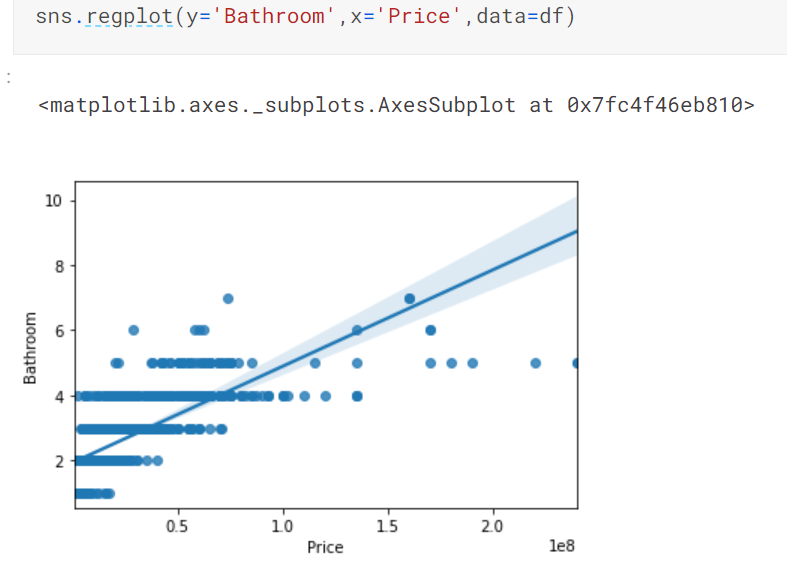
plt.show()



sns.regplot(y='BHK',x='Price',data=df)

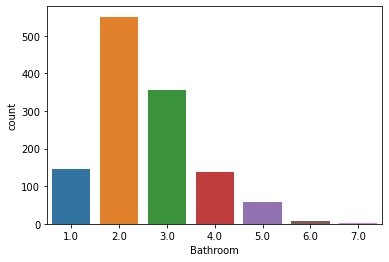


### **From these above plots above we can understand that BHK above 5 are not significant to predict Price. Lets do the same for Bathrooms.**



sns.countplot(df["Bathroom"])

plt.show()



### **Now this analysis implies that No. of Bathrooms above 5 really don't have any significant effect on Price Prediction**

sns.boxplot(df["Parking"])

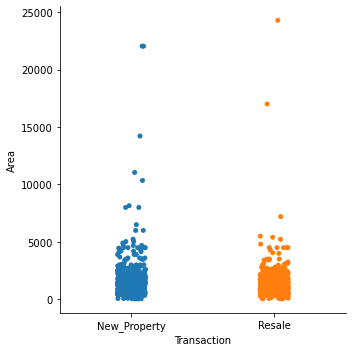
plt.show()



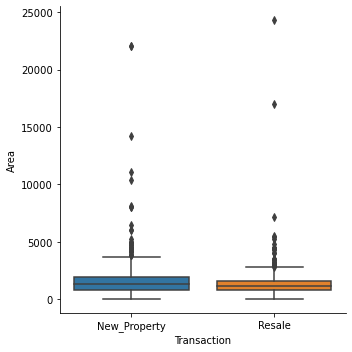
### **No. of Parking is mainly distributed about 1 and 2. This will be discussed later.**

## ***For now let us analyse the Catagorical Data:-***

sns.catplot(x="Transaction", y="Area", data=df)



sns.catplot(x="Transaction", y="Area", kind="box", data=df)



### ***Area is the biggest factor to predict price, as we all know. And seemingly Box Plot is a better way to understand the distribution of these Catagorical Data over Area. These plots as well say that MAX of Area is around 2500 sqft.***

### **Lets see what the other data are signifying--**

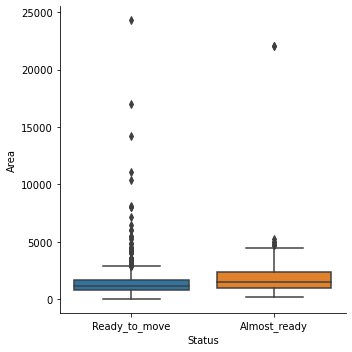
df['Status'].value\_counts()

Ready\_to\_move 1184

Almost\_ready 75

Name: Status, dtype: int64

sns.catplot(x="Status", y="Area", kind="box", data=df)



df['Furnishing'].value\_counts()

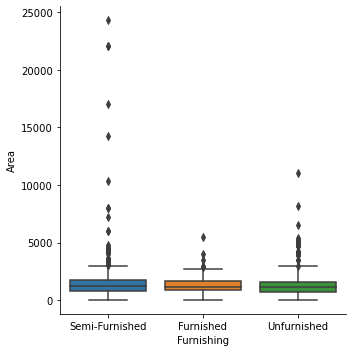
Semi-Furnished 708

Unfurnished 363

Furnished 183

Name: Furnishing, dtype: int64

sns.catplot(x="Furnishing", y="Area", kind="box", data=df)



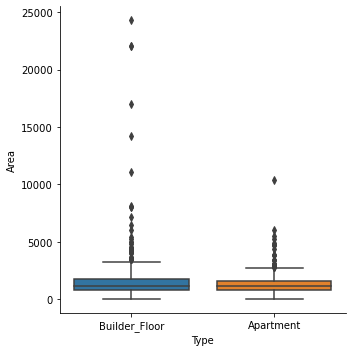
df['Type'].value\_counts()

Builder\_Floor 661

Apartment 593

Name: Type, dtype: int64

sns.catplot(x="Type", y="Area", kind="box", data=df)



### **These are also making us to take Area values below 3000 sqft. Lets see what the Area itself says**

### **Locality Column is a mess actually, 365 different localities are there. Though its evident. So we actually have nothing to do with this column.**

# ***FILTERING:***

## **We have done the Analysis. So now we will filter out the Dataframe. And we will workout the above plans.**

## 1. Filtering out Area Column-->

### Some more codes to get the correct Area range:

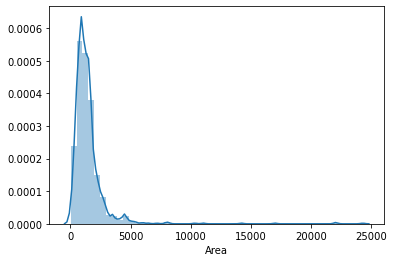
area\_df=df['Area'].value\_counts().head(40)

*# Areas that occured atleast 10 times are taken:--*

area\_df

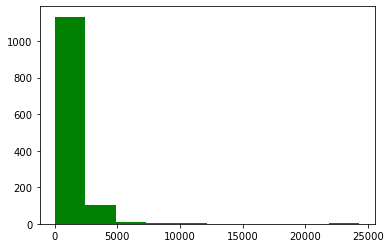
*# among top 36 most occured(atleast 10 times) Area values, MAX is 2700.*

sns.distplot(df['Area'])



plt.hist(df["Area"],color='green')

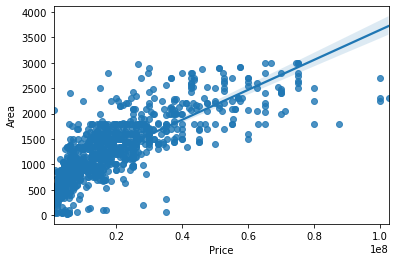
plt.show()



## **Lets keep AREA<=3000 sqft**

df=df[df['Area']<=3000]

sns.regplot(y='Area',x='Price',data=df)



## 2. Filtering out BHK Column-->

df=df[df['BHK']<6]

## 3. Filtering out Bathroom Column-->

df=df[~(df['Bathroom']==6)]

#now as we have obtained our new data frame, we’ll further proceed:

correlation=df.corr()

correlation

|  | Area | BHK | Bathroom | Parking | Price | Per\_Sqft |
| --- | --- | --- | --- | --- | --- | --- |
| Area | 1.000000 | 0.736270 | 0.792406 | -0.038374 | 0.796127 | 0.139027 |
| BHK | 0.736270 | 1.000000 | 0.751536 | -0.088360 | 0.594771 | 0.134648 |
| Bathroom | 0.792406 | 0.751536 | 1.000000 | -0.047850 | 0.742585 | 0.151435 |
| Parking | -0.038374 | -0.088360 | -0.047850 | 1.000000 | -0.020703 | -0.019978 |
| Price | 0.796127 | 0.594771 | 0.742585 | -0.020703 | 1.000000 | 0.267190 |
| Per\_Sqft | 0.139027 | 0.134648 | 0.151435 | -0.019978 | 0.267190 | 1.000000 |

### \*\*Actully there are some Duplicate rows in our df. So dropping them.

df.shape

(1181, 11)

new=df.drop\_duplicates()

new.shape

(1104, 11)

# NULL value Removing:

new.isnull().sum()

Area 0

BHK 0

Bathroom 1

Furnishing 5

Locality 0

Parking 30

Price 0

Status 0

Transaction 0

Type 5

Per\_Sqft 222

dtype: int64

### There are still some NULL values in our Dataset. Lets find out ways to deal with them.

### *First concern is for Per\_Sqft.*

plt.hist(new["Per\_Sqft"],color='brown')

plt.show()



### **There are 200 odd NULL values in Per\_Sqft. I think the best way to fill them will be to fill them by the ratio of corresponding Price and Area.**

compare=(new['Price']/new['Area'])-new['Per\_Sqft']

compare

0 NaN

1 -0.333333

2 9648.789474

3 333.000000

4 2871.461538

...

1252 1528.444444

1253 834.000000

1255 -1011.238095

1256 7084.000000

1257 -1299.838384

Length: 1104, dtype: float64

compare.median()

0.22222222222262644

### You can see Per\_Sqft value is quite close to that ratio. So we will stick to this plan.

new['Per\_Sqft']=new['Per\_Sqft'].fillna(value=new['Price']/new['Area'] )

new.head(50)

new.isnull().sum()

Area 0

BHK 0

Bathroom 1

Furnishing 5

Locality 0

Parking 30

Price 0

Status 0

Transaction 0

Type 5

Per\_Sqft 0

dtype: int64

new[new['Bathroom'].isnull()]

new['Bathroom']=new['Bathroom'].fillna(value=new['Bathroom'].median())

new.isnull().sum()

Area 0

BHK 0

Bathroom 0

Furnishing 5

Locality 0

Parking 30

Price 0

Status 0

Transaction 0

Type 5

Per\_Sqft 0

dtype: int64

new['Furnishing'].value\_counts()

Semi-Furnished 618

Unfurnished 315

Furnished 166

Name: Furnishing,

dtype: int64

new['Type'].value\_counts()

Builder\_Floor 591

Apartment 508

Name: Type, dtype: int64

new[new['Furnishing'].isnull()]

# LABEL ENCODING:

# *>----to convert Furnishing, Status, Transaction and Type into Numerical Data----<*

from sklearn.preprocessing import LabelEncoder

linkcode

furnishing\_encoder=LabelEncoder()

status\_encoder=LabelEncoder()

transaction\_encoder=LabelEncoder()

type\_encoder=LabelEncoder()

new['Furnishing']=furnishing\_encoder.fit\_transform(new['Furnishing'].astype('str'))

new['Status']=status\_encoder.fit\_transform(new['Status'])

new['Transaction']=transaction\_encoder.fit\_transform(new['Transaction'])

new['Type']=type\_encoder.fit\_transform(new['Type'].astype('str'))

new.head(30)

new.corr()['Price']

Area 0.795794

BHK 0.601112

Bathroom 0.750205

Furnishing -0.025317

Parking -0.022254

Price 1.000000

Status -0.098475

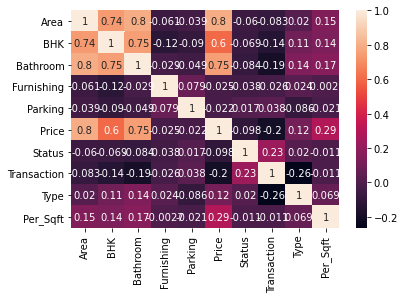
Transaction -0.196361

Type 0.116256

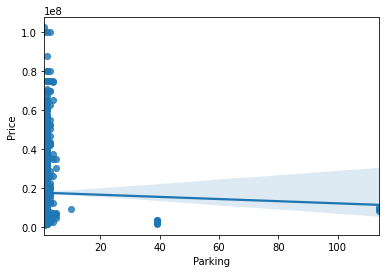
Per\_Sqft 0.290413

Name: Price, dtype: float64

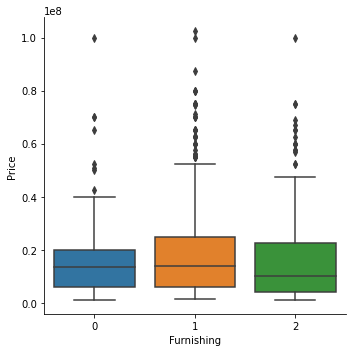
sns.heatmap(new.corr(),annot=True)



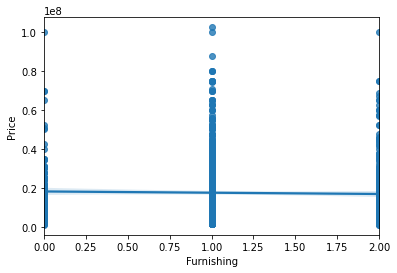
sns.regplot(x="Parking", y="Price", data=new)



sns.catplot(x="Furnishing", y="Price", kind="box", data=new)



sns.regplot(x="Furnishing", y="Price", data=new)



## **Well we can understand from both Correlation Chart and the above plot that Parking Column is Good for Nothing for Price-Prediction. "A correlation of 0.02 is nothing". So is with Furnishing.**

## So these are dropped off along with Locality....

final= new.drop(columns=['Furnishing','Parking','Locality'])

Final

| Area | BHK | Bathroom | Price | Status | Transaction | Type | Per\_Sqft |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 222 | 2700.0 | 4 | 3 | 60000000 | 1 | 0 | 1 | 22222.000000 |
| 645 | 1850.0 | 3 | 2 | 15500000 | 1 | 1 | 0 | 8378.378378 |
| 637 | 1550.0 | 3 | 2 | 13200000 | 1 | 1 | 0 | 8516.129032 |
| 641 | 1470.0 | 3 | 2 | 14500000 | 1 | 1 | 0 | 9863.945578 |
| 642 | 1500.0 | 3 | 2 | 14900000 | 1 | 1 | 0 | 9933.333333 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 332 | 747.0 | 2 | 2 | 6500000 | 1 | 1 | 1 | 8701.000000 |
| 339 | 713.0 | 3 | 1 | 8000000 | 1 | 1 | 0 | 11220.000000 |
| 355 | 700.0 | 2 | 1 | 16000000 | 1 | 0 | 1 | 22857.000000 |
| 1010 | 540.0 | 2 | 1 | 1500000 | 1 | 1 | 1 | 2778.000000 |
| 349 | 400.0 | 2 | 2 | 9000000 | 1 | 1 | 1 | 14550.000000 |

1104 rows × 8 columns

final['Bathroom'].value\_counts()

2 511

3 321

1 142

4 98

5 32

Name: Bathroom, dtype: int64

### "And we are left with a pure numerical dataset."

# MODEL Building:

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,train\_size=0.9,random\_state=2)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

(993, 13)

(111, 13)

(993,)

(111,)

## **(1). Linear Regression:--**

from sklearn.linear\_model import LinearRegression

lr=LinearRegression()

lr.fit(X\_train,y\_train)

LinearRegression()

y\_pred=lr.predict(X\_test)

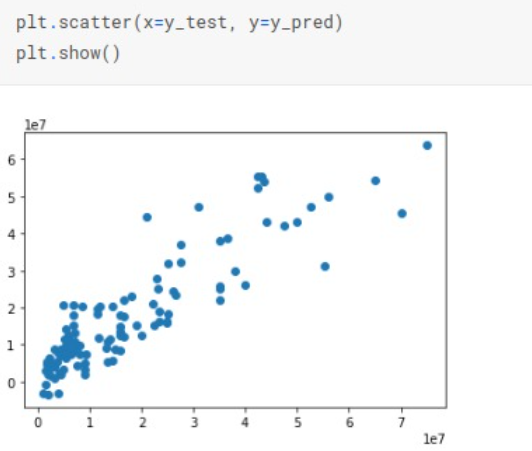
y\_pred

Now finding the accuracy:

from sklearn.metrics import r2\_score

print(f”accuracy of model is {r2\_score(y\_test,y\_pred)\*100}%”)

accuracy of model is 79.34806001571548%



***RESULT:***

The model is trained on the training dataset and evaluated on the testing dataset using mean squared error metric.The model is then applied on the Delhi house pricing dataset and the mean squared error is obtained. The obtained accuracy is **79.34806001571548%** which indicates that the model is performing reasonably well on the dataset.

***REFERENCE:***

* <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>
* <https://numpy.org/doc/stable/>
* <https://pandas.pydata.org/docs/>
* <https://matplotlib.org/stable/contents.html>
* <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html>