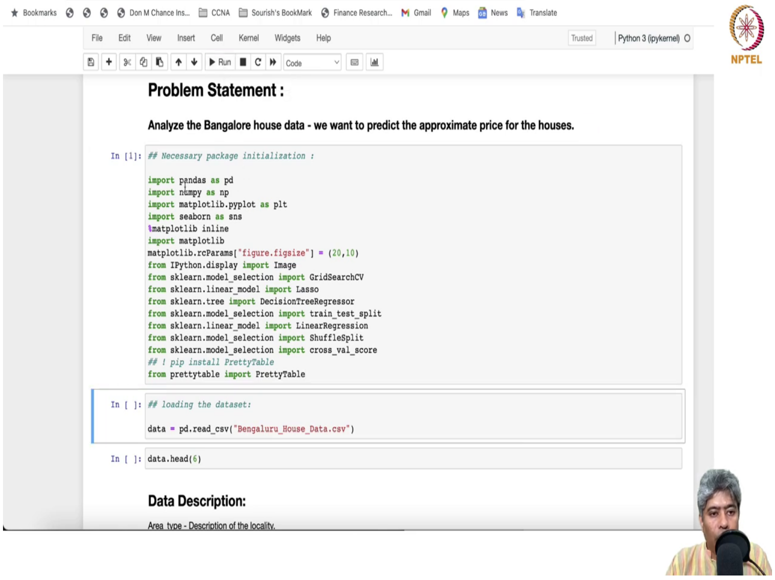
Predictive Analytics - Regression and Classification

Prof. Sourish Das
Department of Mathematics
Chennai Mathematical Institute

Lecture - 54
Hands on with Python: Analysis of Bangalore House Price Data

Hello all. In this video, we are going to do some hands-on. We are going to analyze Bangalore House Price Data using Python. I have shared the Bangalore house price data in the NPTEL platform. You can also get this Bangalore house price data in Kaggle or in internet somewhere. In some GitHub repository also it is available.

(Refer Slide Time: 00:54)



Problem Statement :

Analyze the Bangalore house data - we want to predict the approximate price for the houses.

```
In [1]: ## Necessary package initialization :

        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
        import matplotlib
        matplotlib.rcParams["figure.figsize"] = (20,10)
        from IPython.display import Image
        from sklearn.model_selection import GridSearchCV
        from sklearn.linear_model import Lasso
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LinearRegression
        from sklearn.model_selection import ShuffleSplit
        from sklearn.model_selection import cross_val_score
        ## ! pip install PrettyTable
        from prettytable import PrettyTable
```

```
In [ ]: ## loading the dataset:

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

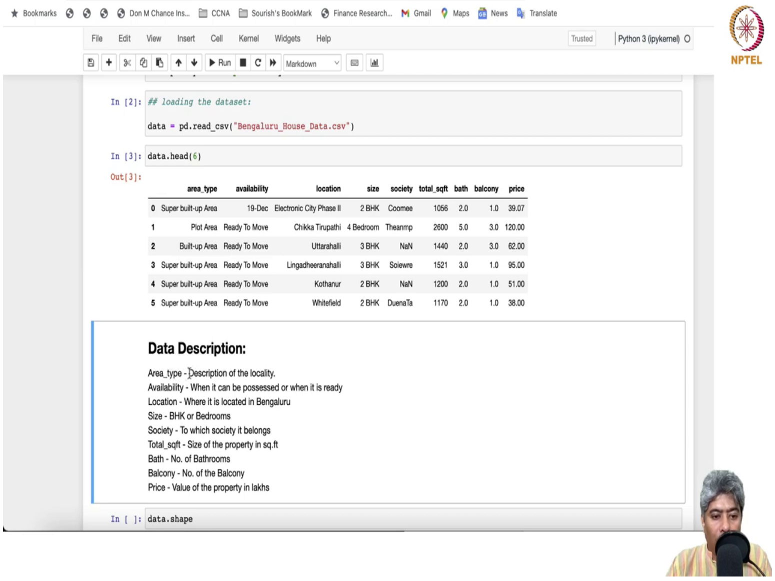
```
In [ ]: data.head(6)
```

Data Description:

Area_type - Description of the locality.

So, in this notebook I prepared the problem statement is we will use this Bangalore house price data and we will try to approximate or estimate a possible price of any house that are kind of part of the Bangalore you know in the vicinity of approximately we will try to estimate a possible price of a possible house. So, first we are going to load this bunch of Python packages.

(Refer Slide Time: 01:28)



```
In [2]: ## loading the dataset:

        data = pd.read_csv("Bengaluru_House_Data.csv")

In [3]: data.head(6)

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
5	Super built-up Area	Ready To Move	Whitefield	2 BHK	DuenaTa	1170	2.0	1.0	38.00

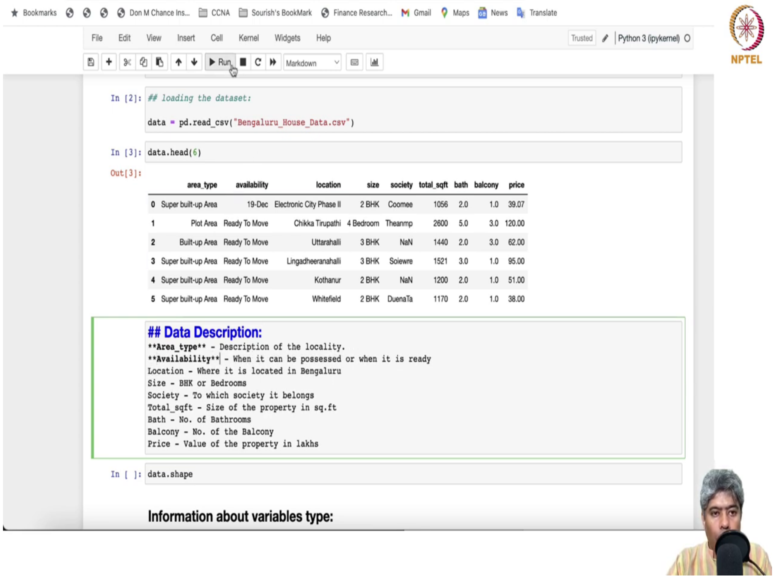
Data Description:

Area_type - Description of the locality.
Availability - When it can be possessed or when it is ready
Location - Where it is located in Bengaluru
Size - BHK or Bedrooms
Society - To which society it belongs
Total_sqft - Size of the property in sq.ft
Bath - No. of Bathrooms
Balcony - No. of the Balcony
Price - Value of the property in lakhs

```
In [ ]: data.shape
```

And, then we are going to call the Bangalore house price data. So, the first I will print this dataset. It has 9 columns; area type, availability, location, size, society, total square feet, bath, a balcony and price. I have also provided the description of each column, area type refers to description of the locality.

(Refer Slide Time: 01:04)



In [2]:
```
## loading the dataset:

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

In [3]:
```
data.head(6)
```

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
5	Super built-up Area	Ready To Move	Whitefield	2 BHK	DuenaTa	1170	2.0	1.0	38.00

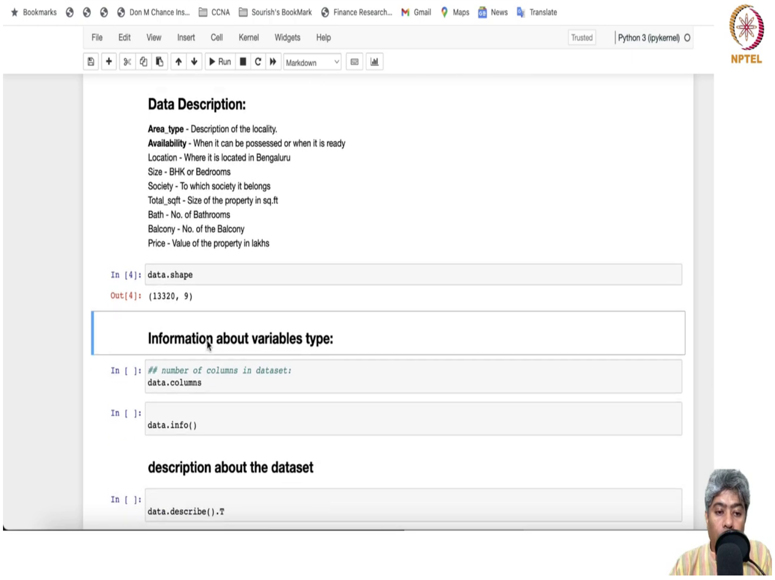
```
## Data Description:
**Area_type** - Description of the locality.
**Availability** - When it can be possessed or when it is ready
Location - Where it is located in Bengaluru
Size - BHK or Bedrooms
Society - To which society it belongs
Total_sqft - Size of the property in sq.ft
Bath - No. of Bathrooms
Balcony - No. of the Balcony
Price - Value of the property in lakhs
```

In [ ]:
```
data.shape
```

Information about variables type:

In fact, I can try this also. So, yeah, you can just put.

(Refer Slide Time: 02:07)



Data Description:

Area_type - Description of the locality.
Availability - When it can be possessed or when it is ready
Location - Where it is located in Bengaluru
Size - BHK or Bedrooms
Society - To which society it belongs
Total_sqft - Size of the property in sq.ft
Bath - No. of Bathrooms
Balcony - No. of the Balcony
Price - Value of the property in lakhs

```
In [4]: data.shape
Out[4]: (13320, 9)
```

Information about variables type:

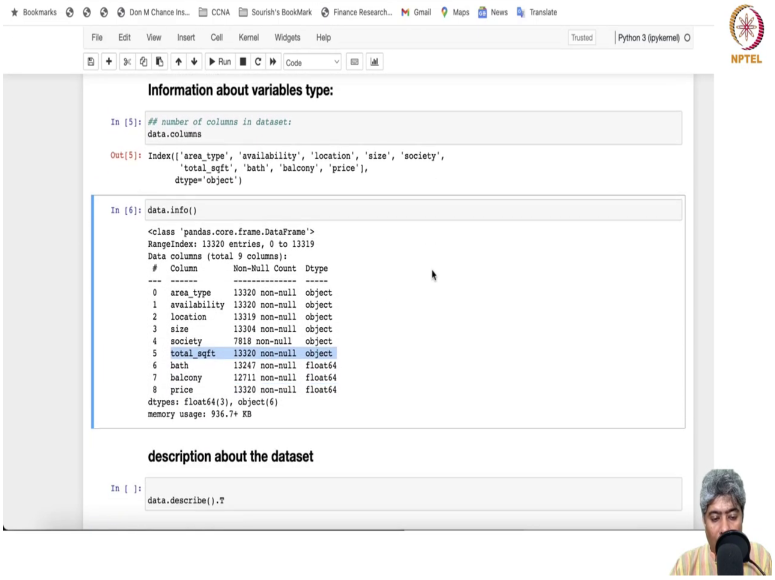
```
In [ ]: ## number of columns in dataset:
        data.columns

In [ ]:
        data.info()
```

description about the dataset

```
In [ ]:
        data.describe().T
```

So, I will do that beautification later and location stands for where it is located in the Bangalore, size stands for BHK or bedrooms like whether it is a 3 BHK or 2 BHK back pay. Society stands for which society looks like there are some in a values. Bath stands for number of bathrooms, the apartment of the house has, balcony is the number of balcony and price stands for the value of the price in rupees lakhs. Data dot shape will give you the shape of the data. It is it has 13,320 rows and 9 columns.

(Refer Slide Time: 03:07)



Information about variables type:

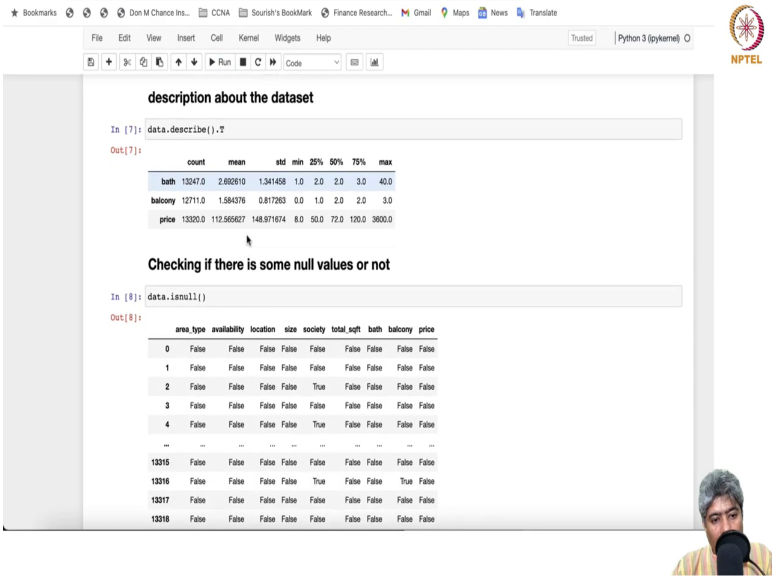
```
In [5]: ## number of columns in dataset:
        data.columns

Out[5]: Index(['area_type', 'availability', 'location', 'size', 'society',
               'total_sqft', 'bath', 'balcony', 'price'],
              dtype='object')

In [6]: data.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 13320 entries, 0 to 13319
        Data columns (total 9 columns):
         #   Column        Non-Null Count  Dtype
        ---  ------        --------------  -----
         0   area_type     13320 non-null  object
         1   availability  13320 non-null  object
         2   location      13319 non-null  object
         3   size          13304 non-null  object
         4   society       7818 non-null   object
         5   total_sqft    13320 non-null  object
         6   bath          13247 non-null  float64
         7   balcony       12711 non-null  float64
         8   price         13320 non-null  float64
        dtypes: float64(3), object(6)
        memory usage: 936.7+ KB
```

description about the dataset

```
In [ ]:
        data.describe().T
```

Data columns, data dot columns will give you the all the columns that are there and data in dot info will give you basic information about each column. Area type, availability location, size, society, square feet, these are objects. Type data, bath, balcony price, these are float. Now, total square feet cannot be object. If you look into the data, total square feets are numeric.

(Refer Slide Time: 03:46)



description about the dataset

```
In [7]: data.describe().T
```

```
Out[7]:
            count        mean         std  min   25%   50%    75%     max
   bath   13247.0    2.692610    1.341458  1.0   2.0   2.0    3.0    40.0
balcony   12711.0    1.584376    0.817263  0.0   1.0   2.0    2.0     3.0
  price   13320.0  112.565627  148.971674  8.0  50.0  72.0  120.0  3600.0
```

Checking if there is some null values or not

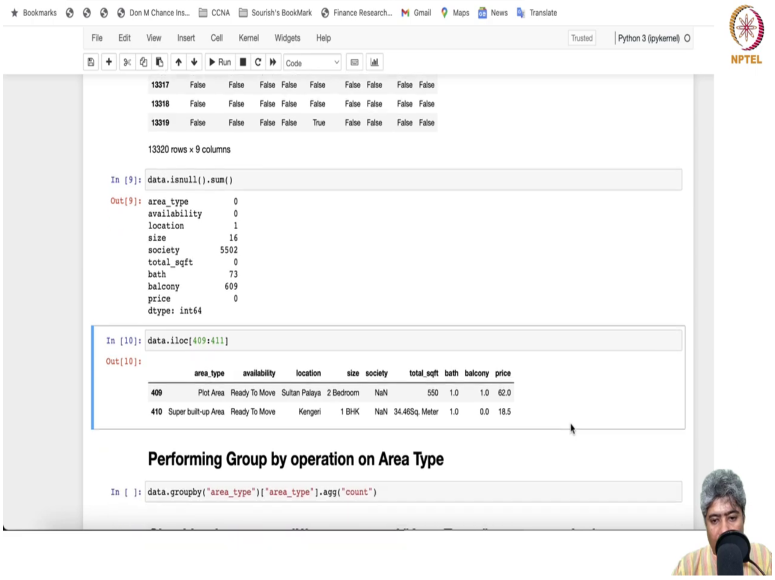
```
In [8]: data.isnull()
```

```
Out[8]:
       area_type  availability  location   size  society  total_sqft   bath  balcony  price
    0      False         False     False  False    False       False  False    False  False
    1      False         False     False  False    False       False  False    False  False
    2      False         False     False  False     True       False  False    False  False
    3      False         False     False  False    False       False  False    False  False
    4      False         False     False  False     True       False  False    False  False
  ...        ...           ...       ...    ...      ...         ...    ...      ...    ...
13315      False         False     False  False    False       False  False    False  False
13316      False         False     False  False     True       False  False     True  False
13317      False         False     False  False    False       False  False    False  False
13318      False         False     False  False    False       False  False    False  False
```

There must be some character that was required that came in the data and that resulted the made it object. We will see how to solve that problem. If we say data dot describes, it will give you basic summary of the float variables which is bath, balcony and price. So, like there are 13,247 instances where the bathroom, bath variable is available average 2.69, minimum 1 bathroom.

There are most of the median is 2, most of the values are here. Most of the apartment or house has 2 bathroom and maximum 40. There could be very big house or mansions which has a very. And, the price is average price is 112 lakhs and median is 72 lakhs whereas, maxima is 3600 lakhs so, near about 36 crore.

(Refer Slide Time: 05:00)



13317	False	False	False	False	False	False	False	False	False
13318	False	False	False	False	False	False	False	False	False
13319	False	False	False	False	True	False	False	False	False

13320 rows × 9 columns

```
In [9]: data.isnull().sum()
```

```
Out[9]: area_type         0
        availability      0
        location          1
        size             16
        society        5502
        total_sqft        0
        bath             73
        balcony         609
        price             0
        dtype: int64
```

```
In [10]: data.iloc[409:411]
```

Out[10]:

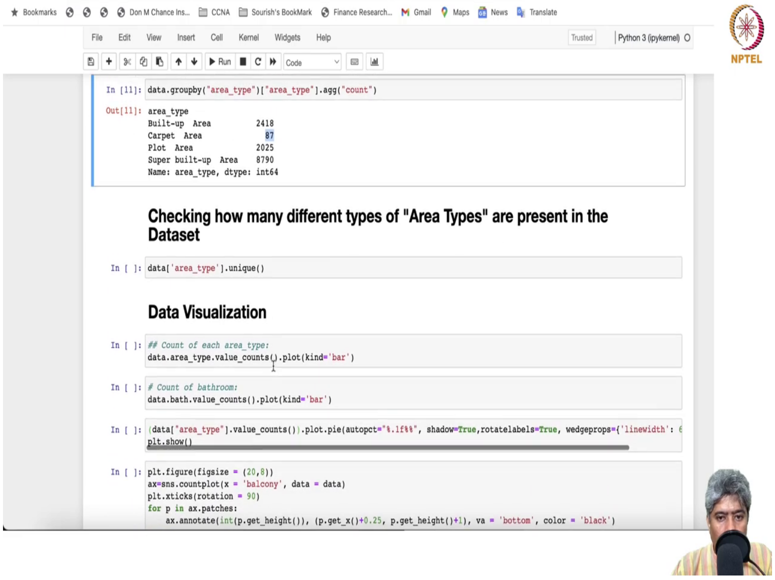
	area_type	availability	location	size	society	total_sqft	bath	balcony	price
409	Plot Area	Ready To Move	Sultan Palaya	2 Bedroom	NaN	550	1.0	1.0	62.0
410	Super built-up Area	Ready To Move	Kengeri	1 BHK	NaN	34.46Sq. Meter	1.0	0.0	18.5

Performing Group by operation on Area Type

```
In [ ]: data.groupby("area_type")["area_type"].agg("count")
```

So, there data dot null, if there are many null values are there and if you do some, then if you see that society has maximum number of null values and balcony also has quite a few null values. So, probably it is better to drop society. And we did some basic searching and we found that in this particular row, total square feet was recorded as 34.46 cube times meter. So, it is a weird way of recorded.

(Refer Slide Time: 05:42)



```
In [11]: data.groupby("area_type")["area_type"].agg("count")

Out[11]: area_type
         Built-up  Area          2418
         Carpet  Area              87
         Plot  Area              2025
         Super built-up  Area    8790
         Name: area_type, dtype: int64
```

Checking how many different types of "Area Types" are present in the Dataset

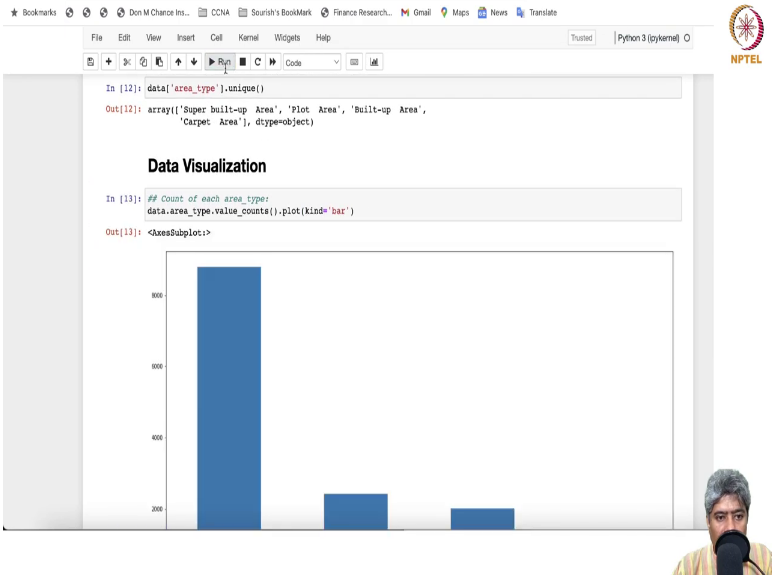
```
In [ ]: data['area_type'].unique()
```

Data Visualization

```
In [ ]: ## Count of each area_type:
        data.area_type.value_counts().plot(kind='bar')

In [ ]: # Count of bathroom:
        data.bath.value_counts().plot(kind='bar')

In [ ]: (data["area_type"].value_counts()).plot.pie(autopct="%.1f%%", shadow=True,rotatelabels=True, wedgeprops={'linewidth': 6
        plt.show()

In [ ]: plt.figure(figsize = (20,8))
        ax=sns.countplot(x = 'balcony', data = data)
        plt.xticks(rotation = 90)
        for p in ax.patches:
            ax.annotate(int(p.get_height()), (p.get_x()+0.25, p.get_height()+1), va = 'bottom', color = 'black')
```

So, we may have to drop this recording and performing group by operation. So, if this will be like area types, these are the count of the area types. There are four kind of area types, Built-up Area, Carpet Area, Plot Area and Super built Area. So, now, Super built Area 8790 instances belongs to Super built Area. Majority of the instances belong to Super built Area. 2000 cases and 2004 and cases belongs to Plot Area or Built-up Area and there are only 87 cases where it is given as a Carpet Area.

(Refer Slide Time: 06:34)



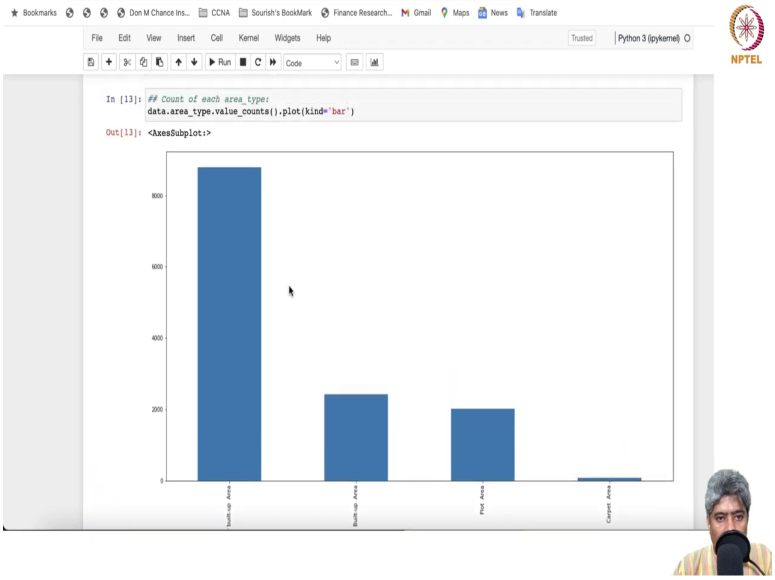
```
In [12]: data['area_type'].unique()

Out[12]: array(['Super built-up  Area', 'Plot  Area', 'Built-up  Area',
                'Carpet  Area'], dtype=object)
```

Data Visualization

```
In [13]: ## Count of each area_type:
         data.area_type.value_counts().plot(kind='bar')

Out[13]: <AxesSubplot:>
```

So, if you run this. So, there are four cases of area types.

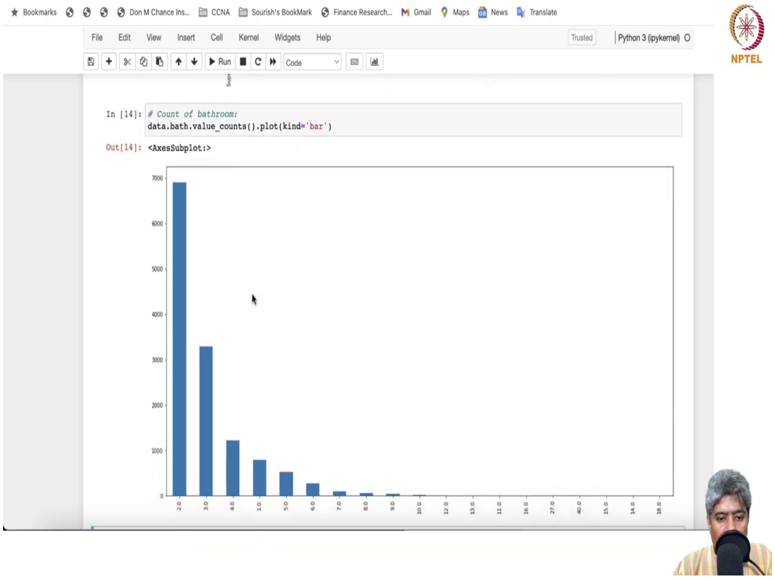
(Refer Slide Time: 06:40)



```
In [13]: ## Count of each area_type:
         data.area_type.value_counts().plot(kind='bar')

Out[13]: <AxesSubplot:>
```

Let us do some visualization. So, there is a bar plot for the area type.

(Refer Slide Time: 06:48)

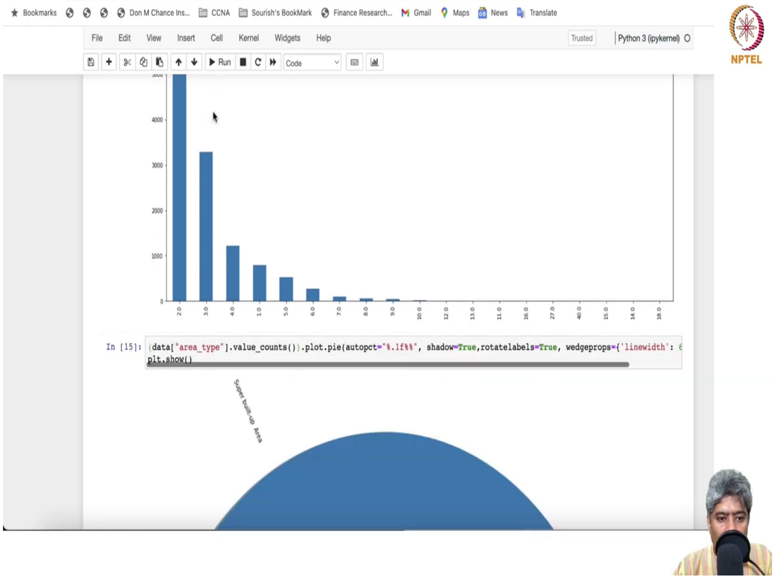


```
In [14]: # Count of bathroom:
         data.bath.value_counts().plot(kind='bar')
```

```
Out[14]: <AxesSubplot:>
```

If you do, we can run this. This is the bathroom has a bar plot.

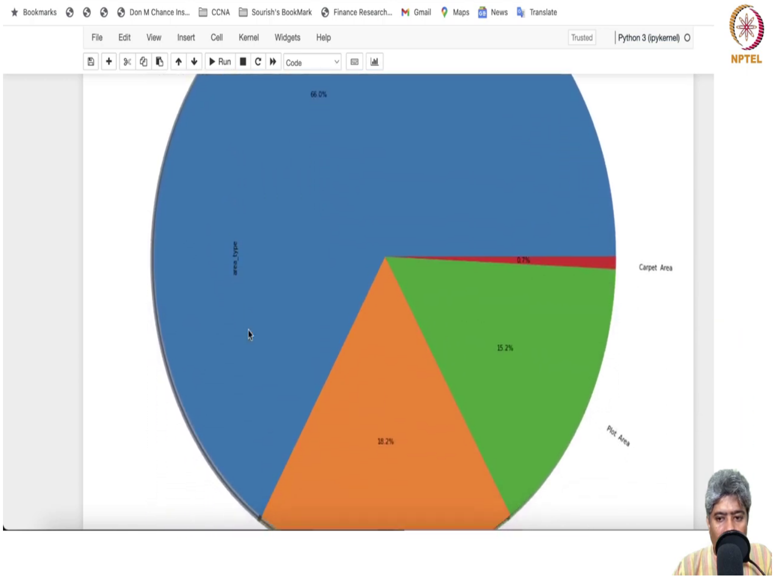
(Refer Slide Time: 06:56)



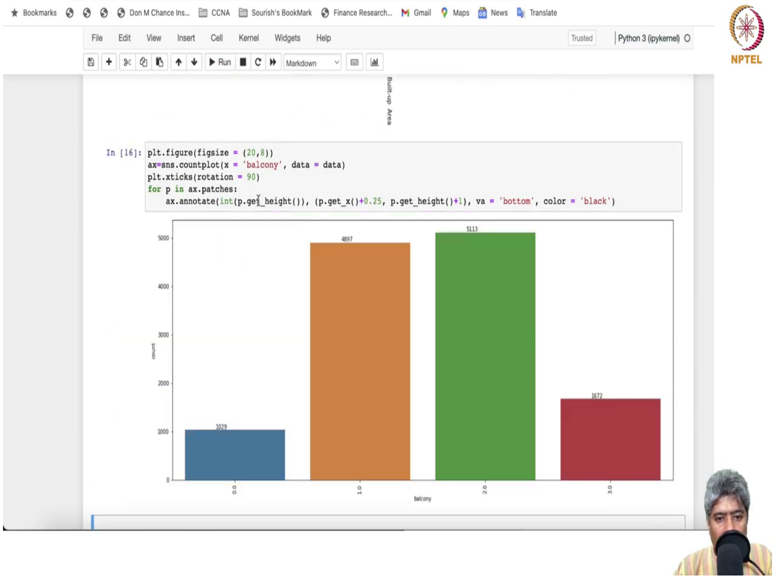
```
In [15]: (data["area_type"].value_counts()).plot.pie(autopct="%.1f%%", shadow=True,rotatelabels=True, wedgeprops={'linewidth': 6
         plt.show()
```

This is the bar plot for the bathroom and you can see there are about to almost 7000 instances had 2 bathroom. So, maximum cases there are 2 bathroom.

(Refer Slide Time: 07:12)



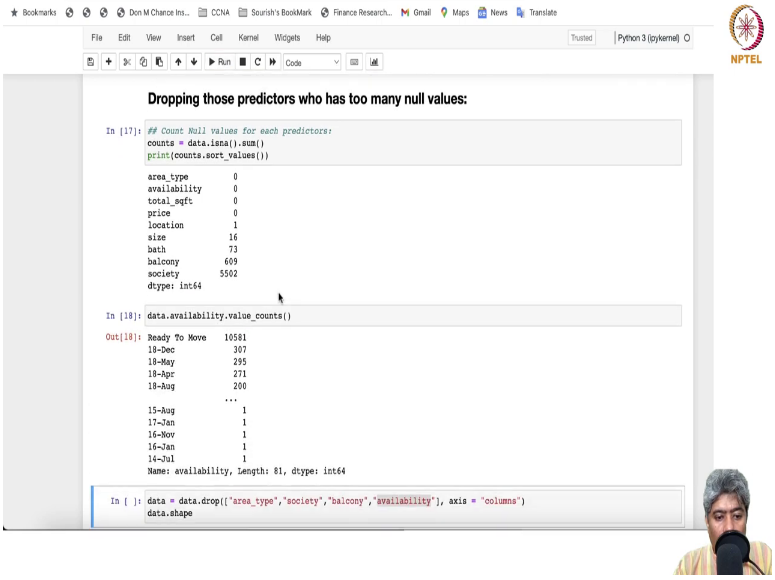
(Refer Slide Time: 07:15)



```
In [16]: plt.figure(figsize = (20,8))
         ax=sns.countplot(x = 'balcony', data = data)
         plt.xticks(rotation = 90)
         for p in ax.patches:
             ax.annotate(int(p.get_height()), (p.get_x()+0.25, p.get_height()+1), va = 'bottom', color = 'black')
```

And, again this is a pie chart for area type.

(Refer Slide Time: 07:21)



Dropping those predictors who has too many null values:

```
In [17]: ## Count Null values for each predictors:
         counts = data.isna().sum()
         print(counts.sort_values())

         area_type          0
         availability       0
         total_sqft         0
         price              0
         location           1
         size              16
         bath              73
         balcony          609
         society         5502
         dtype: int64
```

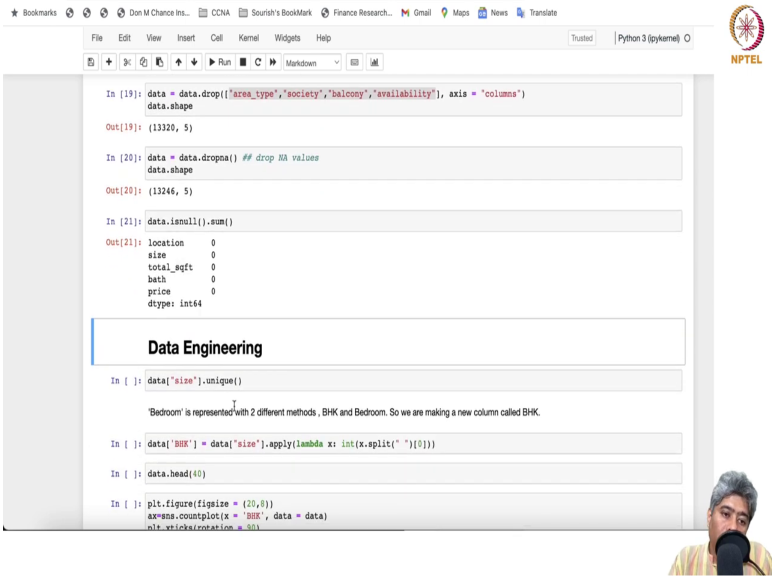
```
In [18]: data.availability.value_counts()

Out[18]: Ready To Move    10581
         18-Dec             307
         18-May             295
         18-Apr             271
         18-Aug             200
                           ...
         15-Aug               1
         17-Jan               1
         16-Nov               1
         16-Jan               1
         14-Jul               1
         Name: availability, Length: 81, dtype: int64
```

```
In [ ]: data = data.drop(["area_type","society","balcony","availability"], axis = "columns")
        data.shape
```

And, balcony there are 0, 1, 2, 3. Majority are either have 1 balcony or 2 balcony. There are few cases there are 3 balcony and quite a few cases there are no balcony. Now, if you count, you can count how many null values are there for each predictor. The society as we found, society as the maximum number of null values, then balcony followed by bath and size.

Price, total square feet, availability, area type does not have any balcony, any null values. We are dropping here area types, society, balcony and availability. Because, if you run the area types, if you actually run this, copy this and.

(Refer Slide Time: 08:50)



```
In [19]: data = data.drop(["area_type","society","balcony","availability"], axis = "columns")
         data.shape

Out[19]: (13320, 5)

In [20]: data = data.dropna() ## drop NA values
         data.shape

Out[20]: (13246, 5)

In [21]: data.isnull().sum()

Out[21]: location      0
         size          0
         total_sqft    0
         bath          0
         price         0
         dtype: int64
```

Data Engineering

```
In [ ]: data["size"].unique()
```

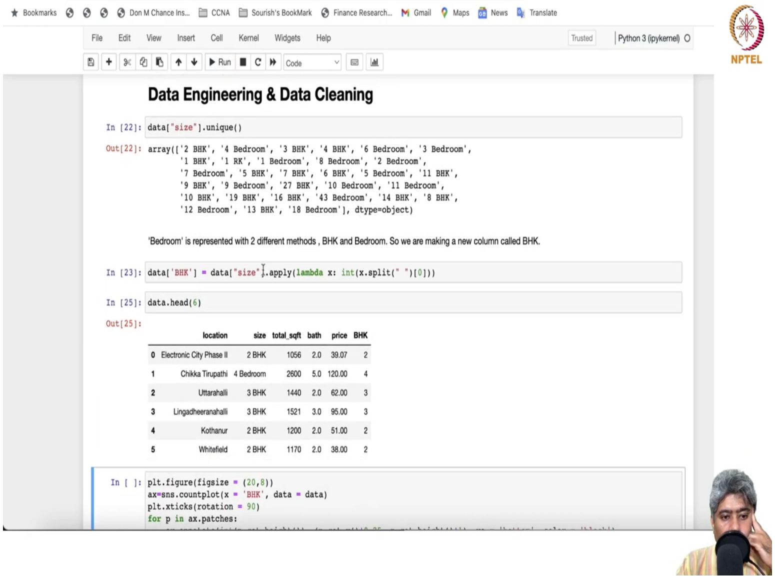
'Bedroom' is represented with 2 different methods , BHK and Bedroom. So we are making a new column called BHK.

```
In [ ]: data['BHK'] = data["size"].apply(lambda x: int(x.split(" ")[0]))

In [ ]: data.head(40)

In [ ]: plt.figure(figsize = (20,8))
        ax=sns.countplot(x = 'BHK', data = data)
        plt.xticks(rotation = 90)
```

So, if you do availability so, you will see that there are 81 different type of availability. Most of the availability goes to ready to move and they stop them at just given a number date without any particular mention. So, it isit is not going to create any help in the model final analysis. And, area type is also not going to available and society has too many null values.

So, that is why we are going to drop these cases. And after we drop, we have now 13,320 cases still there and after dropping these columns. Now, we have 5 columns and now if we drop the NAs, we have still 13,246 now cases. Now, there is no null value in the data set. So, that is a good start to have.

(Refer Slide Time: 09:58)



Data Engineering & Data Cleaning

```
In [22]: data["size"].unique()

Out[22]: 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)
```

'Bedroom' is represented with 2 different methods , BHK and Bedroom. So we are making a new column called BHK.

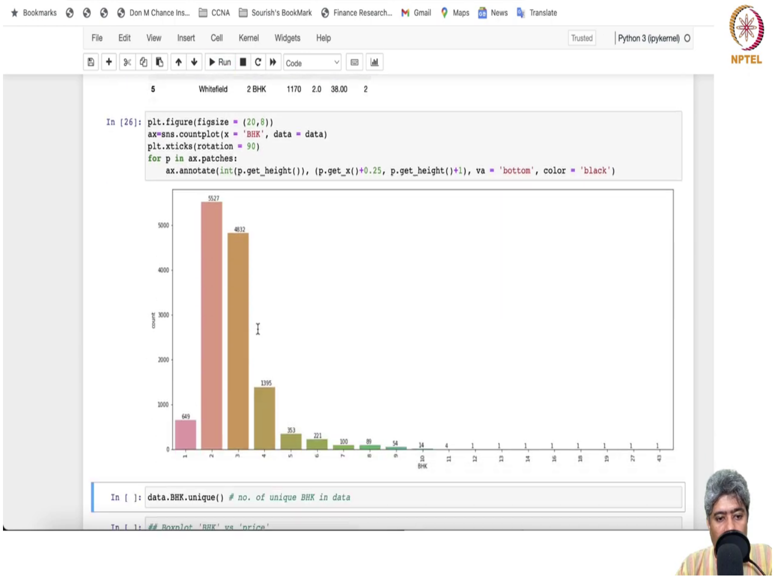
```
In [23]: data['BHK'] = data["size"].apply(lambda x: int(x.split(" ")[0]))

In [25]: data.head(6)

Out[25]:
```

	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
5	Whitefield	2 BHK	1170	2.0	38.00	2

```
In [ ]: plt.figure(figsize = (20,8))
        ax=sns.countplot(x = 'BHK', data = data)
        plt.xticks(rotation = 90)
        for p in ax.patches:
```

So, now, it is a full data, there is no missing data. Now, let us do, we have to may have to do some engineering or data cleaning, not only data engineering, we have to do some and data cleaning ok. Now, first if you run data size, there are you will see that there are 2 BHK, 3 BHK, 4 BHK cases and if you run this, ok. So, there are 2 BHK and then ok, maybe there are 6 alright.

(Refer Slide Time: 10:44)



```
5        Whitefield    2 BHK    1170   2.0   38.00    2
```

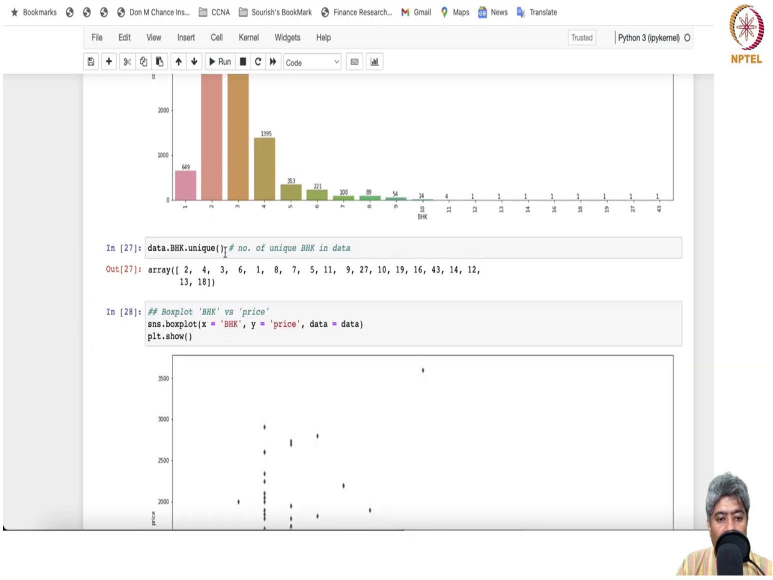
```
In [26]: plt.figure(figsize = (20,8))
         ax=sns.countplot(x = 'BHK', data = data)
         plt.xticks(rotation = 90)
         for p in ax.patches:
             ax.annotate(int(p.get_height()), (p.get_x()+0.25, p.get_height()+1), va = 'bottom', color = 'black')
```

```
In [ ]: data.BHK.unique() # no. of unique BHK in data
```

```
In [ ]: ## Boxplot 'BHK' vs 'price'
```

So, these are the this is the data set that we have. Ok. And, if we drop plot the BHK cases.

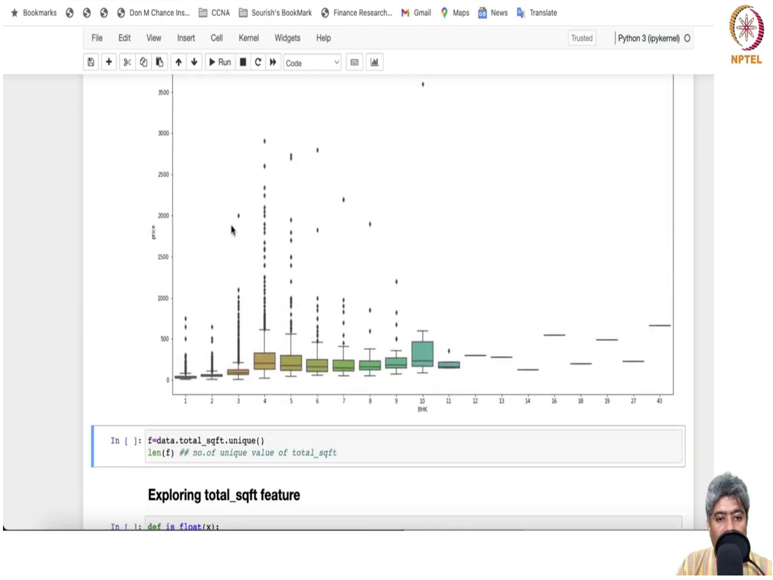
(Refer Slide Time: 10:55)



```
In [27]: data.BHK.unique() # no. of unique BHK in data

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

In [28]: ## Boxplot 'BHK' vs 'price'
         sns.boxplot(x = 'BHK', y = 'price', data = data)
         plt.show()
```

So, these are the different type of 1, 2, 3, 4 different kinds of BHKs and they are plot. So, most of the instances are either 2 BHK house or 3 BHK or 4 BHK houses. There are some 1 BHK and 5 BHK and then other cases are very rare. So, the unique values that we have here.

(Refer Slide Time: 11:25)



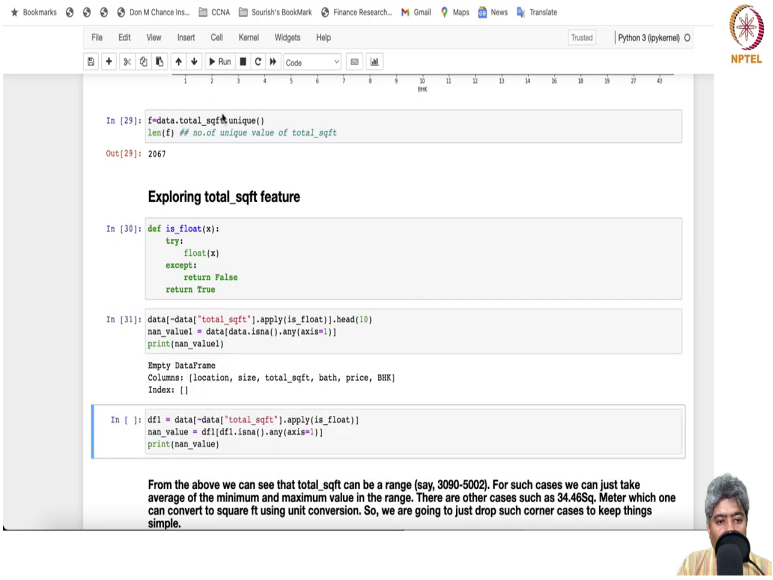
```
In [ ]: f=data.total_sqft.unique()
        len(f) ## no.of unique value of total_sqft
```

Exploring total_sqft feature

```
In [ ]: def is_float(x):
```

And, the BHK versus price box plot, these are like side by side box plot, these are like single value cases.

(Refer Slide Time: 11:33)



```
In [29]: f=data.total_sqft.unique()
         len(f) ## no.of unique value of total_sqft

Out[29]: 2067
```

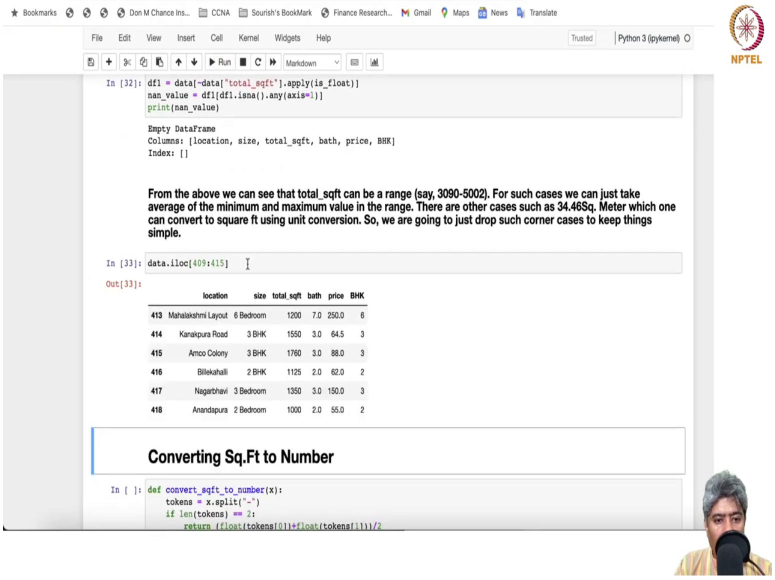
Exploring total_sqft feature

```
In [30]: def is_float(x):
             try:
                 float(x)
             except:
                 return False
             return True

In [31]: data[~data["total_sqft"].apply(is_float)].head(10)
         nan_value1 = data[data.isna().any(axis=1)]
         print(nan_value1)

         Empty DataFrame
         Columns: [location, size, total_sqft, bath, price, BHK]
         Index: []

In [ ]: df1 = data[~data["total_sqft"].apply(is_float)]
        nan_value = df1[df1.isna().any(axis=1)]
        print(nan_value)
```

From the above we can see that total_sqft can be a range (say, 3090-5002). For such cases we can just take average of the minimum and maximum value in the range. There are other cases such as 34.46Sq. Meter which one can convert to square ft using unit conversion. So, we are going to just drop such corner cases to keep things simple.

So, they just put up a dash kind of thing. Now, if we just look into number of unique value in the total square fit, it has 2067 and there are just too many unique values. So, that we have to, there are some character values. So, we have to first check which are the float and which are the not float. So, we can create definition.

(Refer Slide Time: 12:02)



```
In [32]: df1 = data[~data["total_sqft"].apply(is_float)]
         nan_value = df1[df1.isna().any(axis=1)]
         print(nan_value)

         Empty DataFrame
         Columns: [location, size, total_sqft, bath, price, BHK]
         Index: []
```

From the above we can see that total_sqft can be a range (say, 3090-5002). For such cases we can just take average of the minimum and maximum value in the range. There are other cases such as 34.46Sq. Meter which one can convert to square ft using unit conversion. So, we are going to just drop such corner cases to keep things simple.

```
In [33]: data.iloc[409:415]
```

Out[33]:

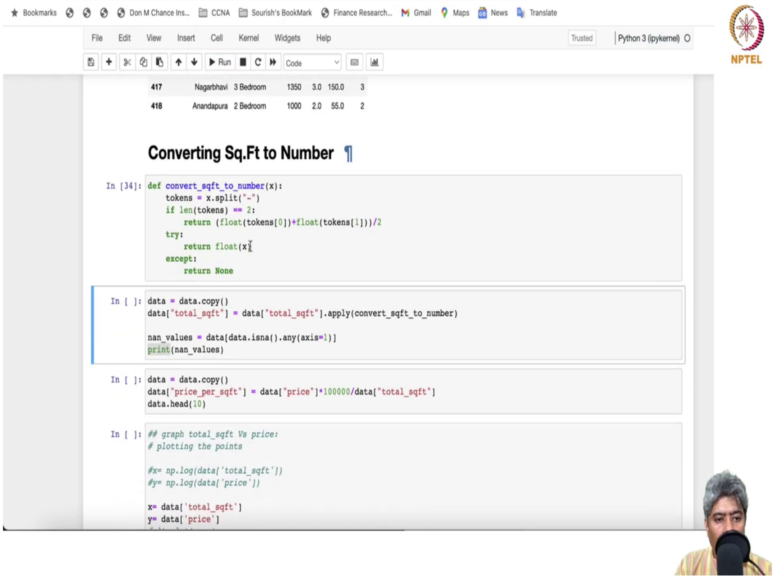
	location	size	total_sqft	bath	price	BHK
413	Mahalakshmi Layout	6 Bedroom	1200	7.0	250.0	6
414	Kanakpura Road	3 BHK	1550	3.0	64.5	3
415	Amco Colony	3 BHK	1760	3.0	88.0	3
416	Billekahalli	2 BHK	1125	2.0	62.0	2
417	Nagarbhavi	3 Bedroom	1350	3.0	150.0	3
418	Anandapura	2 Bedroom	1000	2.0	55.0	2

Converting Sq.Ft to Number

```
In [ ]: def convert_sqft_to_number(x):
            tokens = x.split("-")
            if len(tokens) == 2:
                return (float(tokens[0])+float(tokens[1]))/2
```

And, then where we have float and a. So, empty data frame and then ok. Let us run this. Hopefully, this is fine.

(Refer Slide Time: 12:27)



```
417    Nagarbhavi  3 Bedroom   1350  3.0  150.0  3
418    Anandapura  2 Bedroom   1000  2.0   55.0  2
```

Converting Sq.Ft to Number ¶

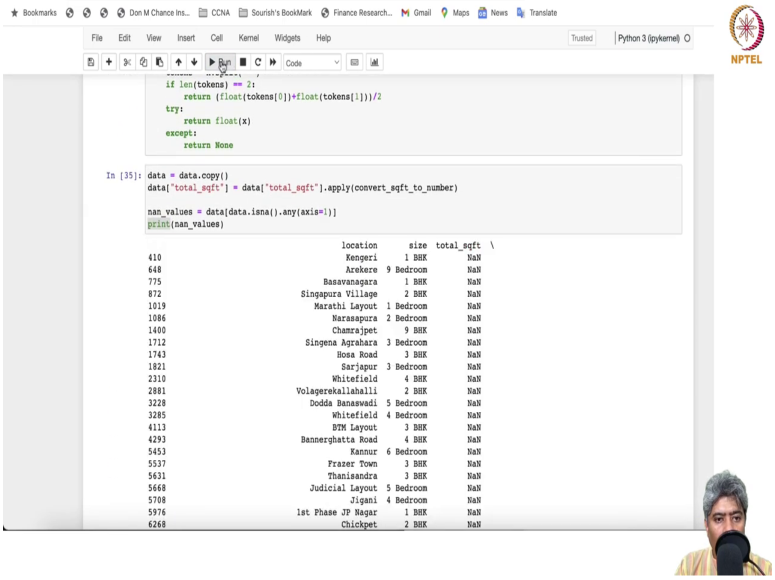
```
In [34]: def convert_sqft_to_number(x):
             tokens = x.split("-")
             if len(tokens) == 2:
                 return (float(tokens[0])+float(tokens[1]))/2
             try:
                 return float(x)
             except:
                 return None
```

```
In [ ]: data = data.copy()
        data["total_sqft"] = data["total_sqft"].apply(convert_sqft_to_number)

        nan_values = data[data.isna().any(axis=1)]
        print(nan_values)
```

```
In [ ]: data = data.copy()
        data["price_per_sqft"] = data["price"]*100000/data["total_sqft"]
        data.head(10)
```

```
In [ ]: ## graph total_sqft Vs price:
        # plotting the points

        #x= np.log(data['total_sqft'])
        #y= np.log(data['price'])

        x= data['total_sqft']
        y= data['price']
```

(Refer Slide Time: 12:30)



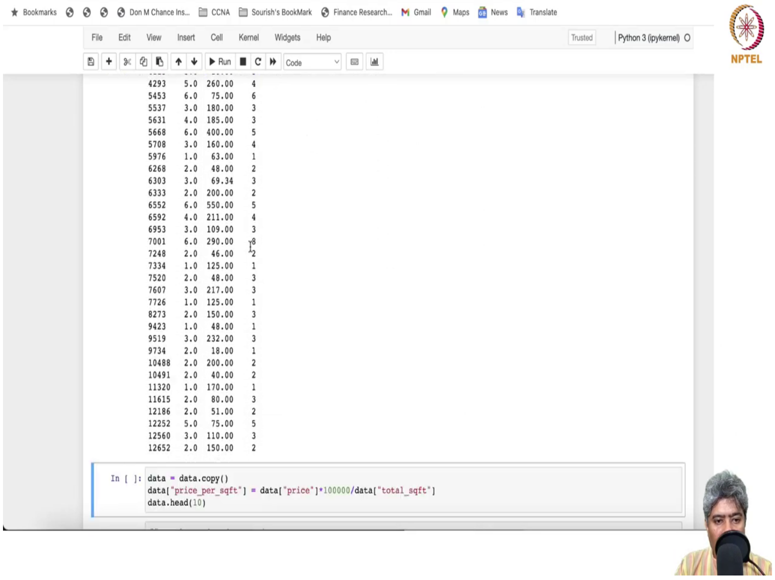
```
    if len(tokens) == 2:
        return (float(tokens[0])+float(tokens[1]))/2
    try:
        return float(x)
    except:
        return None
```

```
In [35]: data = data.copy()
         data["total_sqft"] = data["total_sqft"].apply(convert_sqft_to_number)

         nan_values = data[data.isna().any(axis=1)]
         print(nan_values)
```

```
                               location       size  total_sqft  \
410                             Kengeri      1 BHK         NaN
648                             Arekere  9 Bedroom         NaN
775                         Basavanagara      1 BHK         NaN
872                     Singapura Village      2 BHK         NaN
1019                      Marathi Layout  1 Bedroom         NaN
1086                          Narasapura  2 Bedroom         NaN
1400                          Chamrajpet      9 BHK         NaN
1712                    Singena Agrahara  3 Bedroom         NaN
1743                           Hosa Road      3 BHK         NaN
1821                            Sarjapur  3 Bedroom         NaN
2310                          Whitefield      4 BHK         NaN
2881                  Volagerekallahalli      2 BHK         NaN
3228                      Dodda Banaswadi  5 Bedroom         NaN
3285                          Whitefield  4 Bedroom         NaN
4113                          BTM Layout      3 BHK         NaN
4293                    Bannerghatta Road      4 BHK         NaN
5453                              Kannur  6 Bedroom         NaN
5537                         Frazer Town      3 BHK         NaN
5631                         Thanisandra      3 BHK         NaN
5668                     Judicial Layout  5 Bedroom         NaN
5708                              Jigani  4 Bedroom         NaN
5976                    1st Phase JP Nagar      1 BHK         NaN
6268                            Chickpet      2 BHK         NaN
```

So, there are quite a few.

(Refer Slide Time: 12:38)

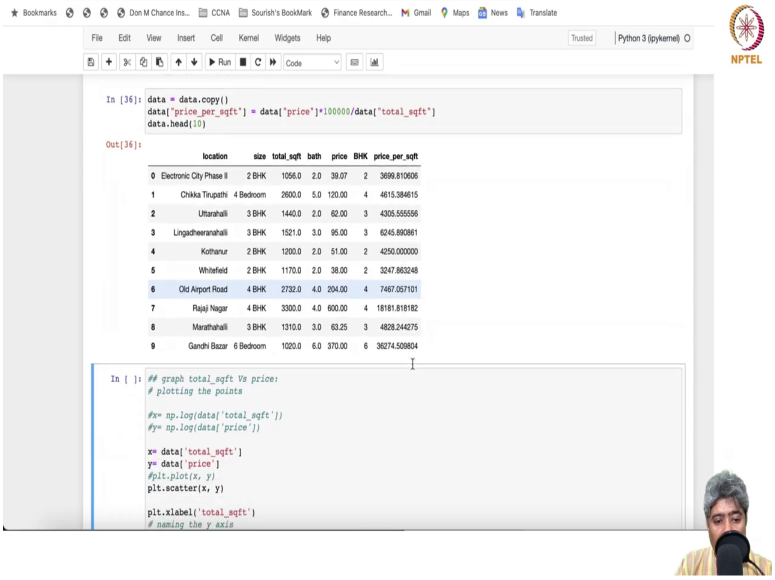


```
4293   5.0  260.00   4
5453   6.0   75.00   6
5537   3.0  180.00   3
5631   4.0  185.00   3
5668   6.0  400.00   5
5708   3.0  160.00   4
5976   1.0   63.00   1
6268   2.0   48.00   2
6303   3.0   69.34   3
6333   2.0  200.00   2
6552   6.0  550.00   5
6592   4.0  211.00   4
6953   3.0  109.00   3
7001   6.0  290.00   8
7248   2.0   46.00   2
7334   1.0  125.00   1
7520   2.0   48.00   3
7607   3.0  217.00   3
7726   1.0  125.00   1
8273   2.0  150.00   3
9423   1.0   48.00   1
9519   3.0  232.00   3
9734   2.0   18.00   1
10488  2.0  200.00   2
10491  2.0   40.00   2
11320  1.0  170.00   1
11615  2.0   80.00   3
12186  2.0   51.00   2
12252  5.0   75.00   5
12560  3.0  110.00   3
12652  2.0  150.00   2
```

```
In [ ]: data = data.copy()
        data["price_per_sqft"] = data["price"]*100000/data["total_sqft"]
        data.head(10)
```

These are the cases where the total square feet you have NAs.

(Refer Slide Time: 12:40)



```
In [36]: data = data.copy()
         data["price_per_sqft"] = data["price"]*100000/data["total_sqft"]
         data.head(10)
```

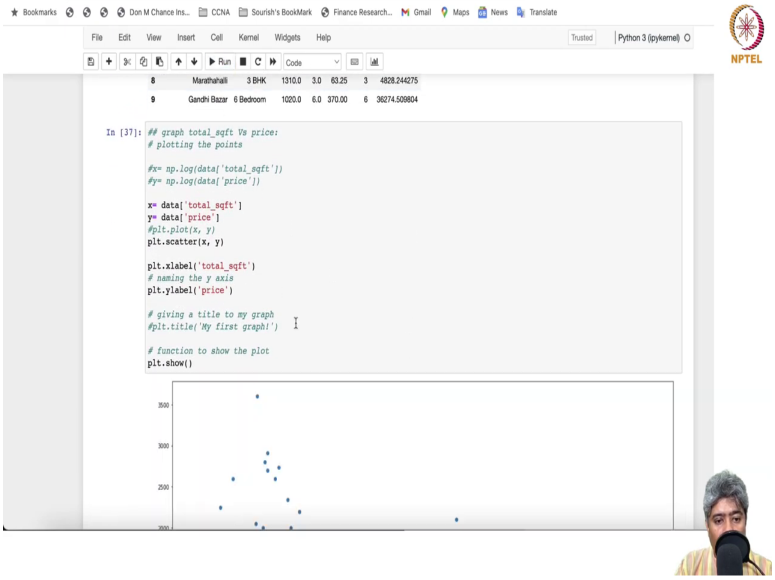
Out[36]:

	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	Gandhi Bazar	6 Bedroom	1020.0	6.0	370.00	6	36274.509804

```
In [ ]: ## graph total_sqft Vs price:
        # plotting the points

        #x= np.log(data['total_sqft'])
        #y= np.log(data['price'])

        x= data['total_sqft']
        y= data['price']
        #plt.plot(x, y)
        plt.scatter(x, y)

        plt.xlabel('total_sqft')
        # naming the y axis
```

And so, these are the, we just created price per total square feet. So, price per square feet.

(Refer Slide Time: 12:59)

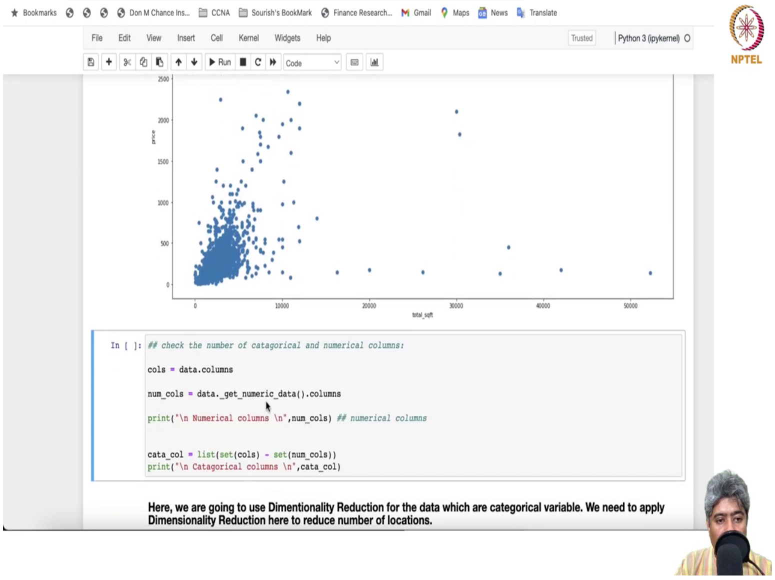


```
8    Marathahalli   3 BHK       1310.0  3.0   63.25   3   4828.244275
9    Gandhi Bazar   6 Bedroom   1020.0  6.0  370.00   6  36274.509804
```

```
In [37]: ## graph total_sqft Vs price:
         # plotting the points

         #x= np.log(data['total_sqft'])
         #y= np.log(data['price'])

         x= data['total_sqft']
         y= data['price']
         #plt.plot(x, y)
         plt.scatter(x, y)

         plt.xlabel('total_sqft')
         # naming the y axis
         plt.ylabel('price')

         # giving a title to my graph
         #plt.title('My first graph!')

         # function to show the plot
         plt.show()
```

How much you are paying for each square feet?

(Refer Slide Time: 13:03)

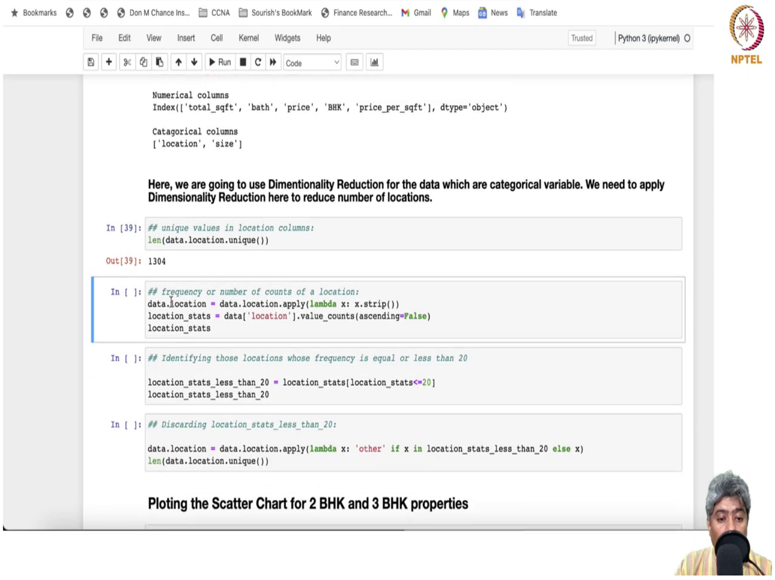


```
In [ ]: ## check the number of catagorical and numerical columns:

        cols = data.columns

        num_cols = data._get_numeric_data().columns

        print("\n Numerical columns \n",num_cols) ## numerical columns


        cata_col = list(set(cols) - set(num_cols))
        print("\n Catagorical columns \n",cata_col)
```

Here, we are going to use Dimentionality Reduction for the data which are categorical variable. We need to apply Dimensionality Reduction here to reduce number of locations.

And now, if you just plot, you can see this on the x axis we are plotting total square per feet versus price.

(Refer Slide Time: 13:12)



```
Numerical columns
Index(['total_sqft', 'bath', 'price', 'BHK', 'price_per_sqft'], dtype='object')

Catagorical columns
['location', 'size']
```

Here, we are going to use Dimentionality Reduction for the data which are categorical variable. We need to apply Dimensionality Reduction here to reduce number of locations.

```
In [39]: ## unique values in location columns:
         len(data.location.unique())

Out[39]: 1304
```

```
In [ ]: ## frequency or number of counts of a location:
        data.location = data.location.apply(lambda x: x.strip())
        location_stats = data['location'].value_counts(ascending=False)
        location_stats
```

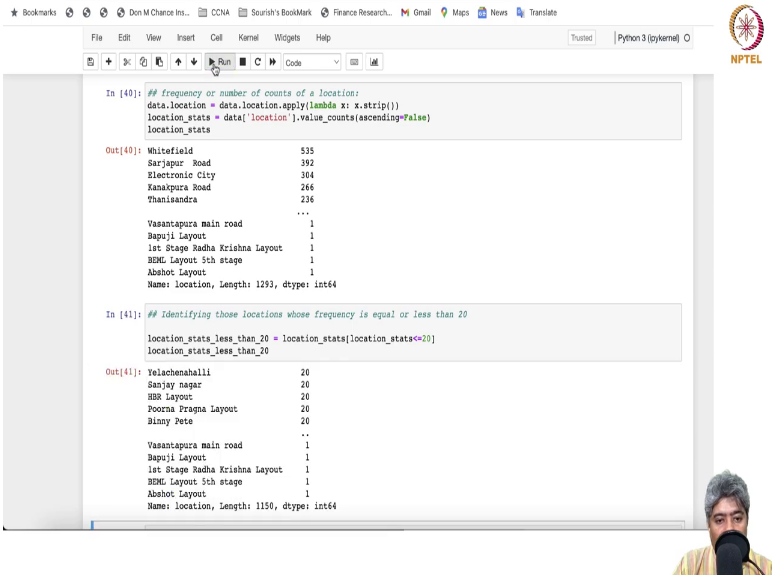
```
In [ ]: ## Identifying those locations whose frequency is equal or less than 20

        location_stats_less_than_20 = location_stats[location_stats<=20]
        location_stats_less_than_20
```

```
In [ ]: ## Discarding location_stats_less_than_20:

        data.location = data.location.apply(lambda x: 'other' if x in location_stats_less_than_20 else x)
        len(data.location.unique())
```

Ploting the Scatter Chart for 2 BHK and 3 BHK properties

And we are here, we are going to, we are checking the categorical variable versus numerical variables, how many categorical, how many numerical, those are which are the. So, these are the numerical variables and these two are categorical, location and size. So, how many new unique locations we have? So, there are 1300 unique locations that we have.

(Refer Slide Time: 13:48)

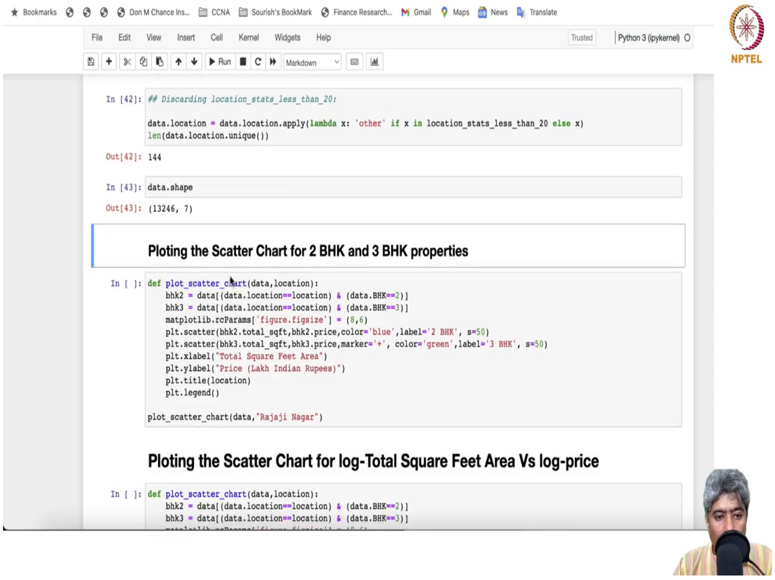


```
In [40]: ## frequency or number of counts of a location:
         data.location = data.location.apply(lambda x: x.strip())
         location_stats = data['location'].value_counts(ascending=False)
         location_stats

Out[40]: Whitefield                      535
         Sarjapur  Road                  392
         Electronic City                 304
         Kanakpura Road                  266
         Thanisandra                     236
                                        ...
         Vasantapura main road             1
         Bapuji Layout                     1
         1st Stage Radha Krishna Layout    1
         BEML Layout 5th stage             1
         Abshot Layout                     1
         Name: location, Length: 1293, dtype: int64

In [41]: ## Identifying those locations whose frequency is equal or less than 20

         location_stats_less_than_20 = location_stats[location_stats<=20]
         location_stats_less_than_20

Out[41]: Yelachenahalli                  20
         Sanjay nagar                    20
         HBR Layout                      20
         Poorna Pragna Layout            20
         Binny Pete                      20
                                         ..
         Vasantapura main road            1
         Bapuji Layout                    1
         1st Stage Radha Krishna Layout   1
         BEML Layout 5th stage            1
         Abshot Layout                    1
         Name: location, Length: 1150, dtype: int64
```

And in the 1300 unique location, we count that and these is the situation, there are quite a few location where you have only single instances. So, if we do that, then there will be little bit of a problem because you see, there are 1300, about 1300 locations, unique location and if we create a dummy variable, if we do a one-hot one encoding and if we do create a dummy variable, for each location, it will create a dummy variable.

And there are instances we have one one-hot encoding and that will make the solution unreliable. In fact, probably you will not have a solution, you can have a solution, but you will not be able to calculate the standard error. So, you need at least 20 locations, a 20 instances for each for a locations, a solution to estimate. So, what we are doing here, we are saying that, ok, let us try to figure out which are the locations you have unique are less than 20 instances ok.

(Refer Slide Time: 15:10)



```
In [42]: ## Discarding location_stats_less_than_20:

         data.location = data.location.apply(lambda x: 'other' if x in location_stats_less_than_20 else x)
         len(data.location.unique())

Out[42]: 144

In [43]: data.shape

Out[43]: (13246, 7)
```

Ploting the Scatter Chart for 2 BHK and 3 BHK properties

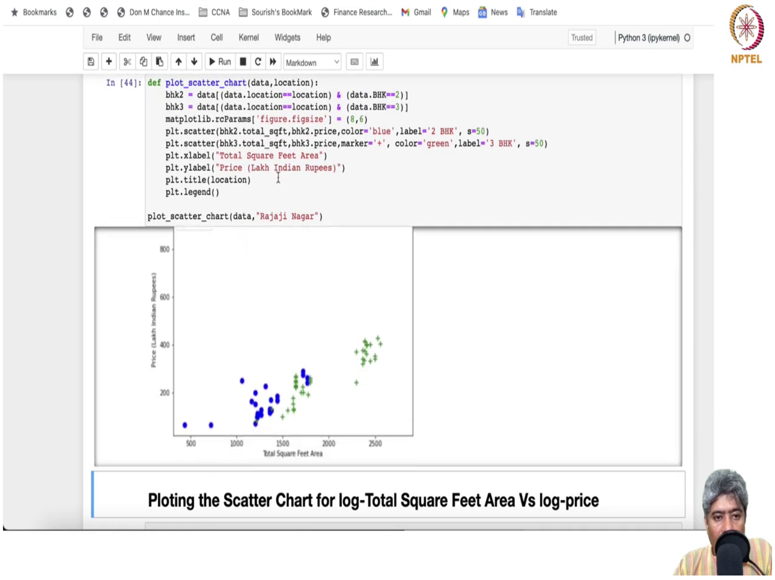
```
In [ ]: def plot_scatter_chart(data,location):
            bhk2 = data[(data.location==location) & (data.BHK==2)]
            bhk3 = data[(data.location==location) & (data.BHK==3)]
            matplotlib.rcParams['figure.figsize'] = (8,6)
            plt.scatter(bhk2.total_sqft,bhk2.price,color='blue',label='2 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(data,"Rajaji Nagar")
```

Ploting the Scatter Chart for log-Total Square Feet Area Vs log-price

```
In [ ]: def plot_scatter_chart(data,location):
            bhk2 = data[(data.location==location) & (data.BHK==2)]
            bhk3 = data[(data.location==location) & (data.BHK==3)]
```

These are the locations where you have less than 20 instances, ok. And, let us drop those and we are discarding those locations. So, there are now 144 unique locations. After dropping 20 location, where we have now have 44 unique locations. And if you just say data dot shape, still you have 13,240, 7, 246 instances you have. So, this is a good thing.

(Refer Slide Time: 16:01)

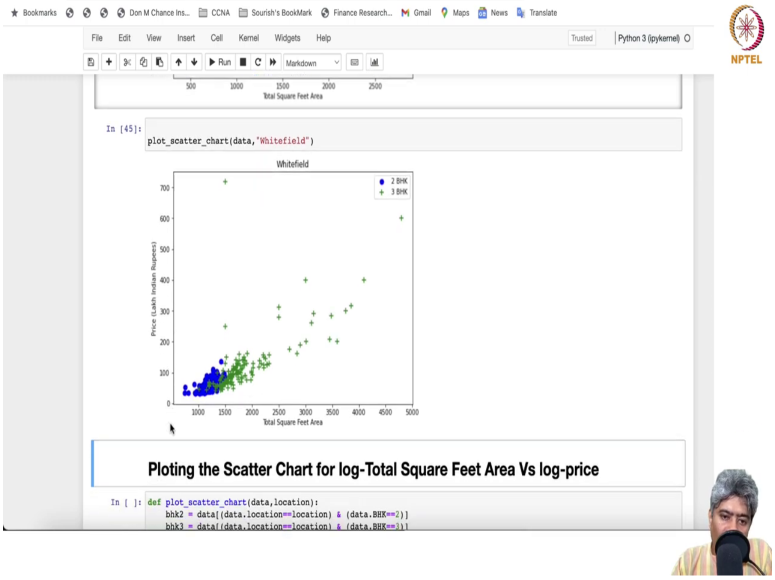


```
In [44]: def plot_scatter_chart(data,location):
             bhk2 = data[(data.location==location) & (data.BHK==2)]
             bhk3 = data[(data.location==location) & (data.BHK==3)]
             matplotlib.rcParams['figure.figsize'] = (8,6)
             plt.scatter(bhk2.total_sqft,bhk2.price,color='blue',label='2 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(data,"Rajaji Nagar")
```

Ploting the Scatter Chart for log-Total Square Feet Area Vs log-price

And now we are going to do some plotting, scattered chart for 2BHK and 3BHK properties. So, say for example, here in the Rajaji Nagar area, you can give a area name and for that area, you will get a 2BHK and 3BHK's price or say Whitefield, if you take Whitefield.

(Refer Slide Time: 16:38)



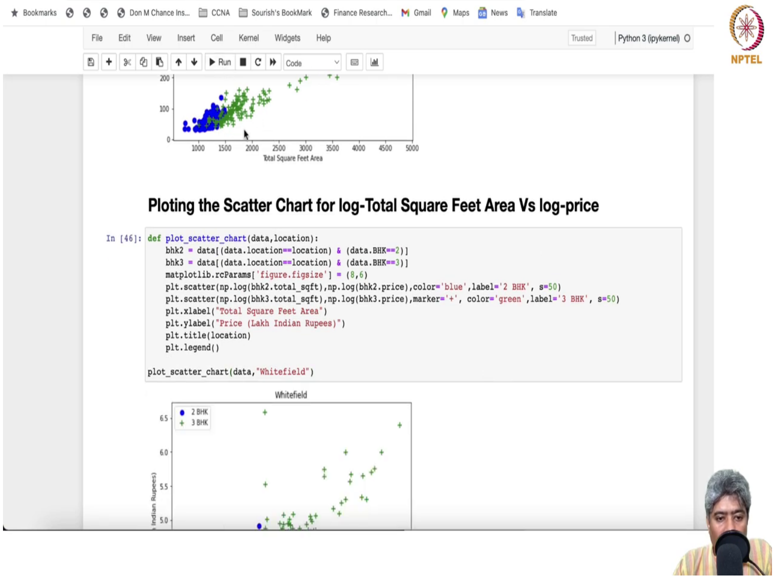
```
In [45]: plot_scatter_chart(data,"Whitefield")
```

Ploting the Scatter Chart for log-Total Square Feet Area Vs log-price

```
In [ ]: def plot_scatter_chart(data,location):
            bhk2 = data[(data.location==location) & (data.BHK==2)]
            bhk3 = data[(data.location==location) & (data.BHK==3)]
```

And you give another place, just a minute. So, let me just copy this here and instead of. So, now, let me just run this. So, in the Whitefield, these are the 2BHK prices and these are the 3BHK. The x axis we plot the total square feet and on the y axis, we have right plotting the price for each of each cases.

(Refer Slide Time: 17:28)

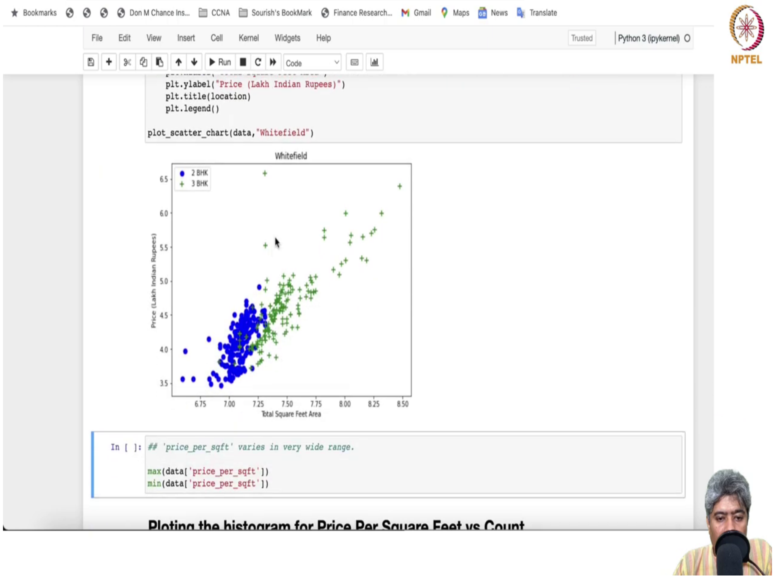


Ploting the Scatter Chart for log-Total Square Feet Area Vs log-price

```
In [46]: def plot_scatter_chart(data,location):
             bhk2 = data[(data.location==location) & (data.BHK==2)]
             bhk3 = data[(data.location==location) & (data.BHK==3)]
             matplotlib.rcParams['figure.figsize'] = (8,6)
             plt.scatter(np.log(bhk2.total_sqft),np.log(bhk2.price),color='blue',label='2 BHK', s=50)
             plt.scatter(np.log(bhk3.total_sqft),np.log(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(data,"Whitefield")
```

Now, we are plotting the chart for a log total, log scale in the total square feet versus log price.

(Refer Slide Time: 17:46)



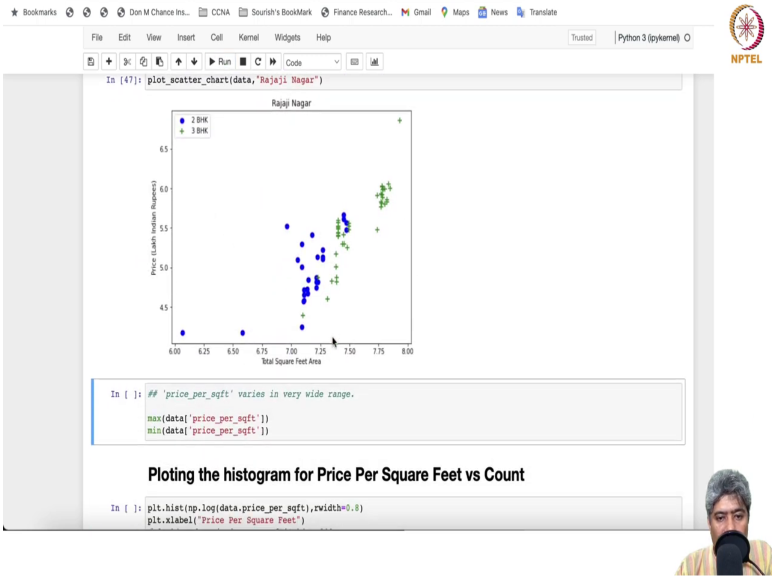
```
    plt.ylabel("Price (Lakh Indian Rupees)")
    plt.title(location)
    plt.legend()

plot_scatter_chart(data,"Whitefield")
```

```
In [ ]: ## 'price_per_sqft' varies in very wide range.

        max(data['price_per_sqft'])
        min(data['price_per_sqft'])
```

Ploting the histogram for Price Per Square Feet vs Count

So, if you plot that, they are somewhat in the much more linear in terms.

(Refer Slide Time: 17:51)



```
In [47]: plot_scatter_chart(data,"Rajaji Nagar")
```

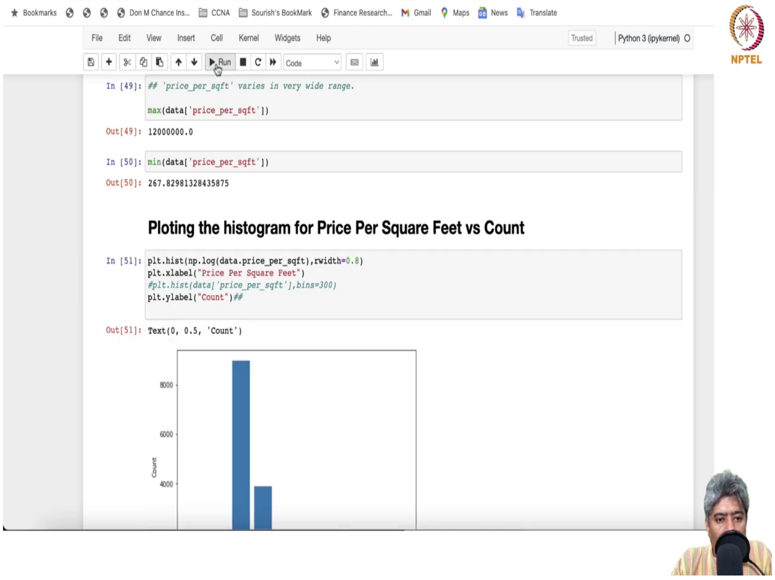
```
In [ ]: ## 'price_per_sqft' varies in very wide range.

        max(data['price_per_sqft'])
        min(data['price_per_sqft'])
```

Ploting the histogram for Price Per Square Feet vs Count

```
In [ ]: plt.hist(np.log(data.price_per_sqft),rwidth=0.8)
        plt.xlabel("Price Per Square Feet")
```

And if you just do the, do it for the Whitefield also. So, this is already for the Whitefield. Let me do it for Rajaji Nagar than. So, this was for the Whitefield. Let me do it for the Rajaji Nagar. Alright. Ok. So, this is less number of instances that you have.

(Refer Slide Time: 18:46)



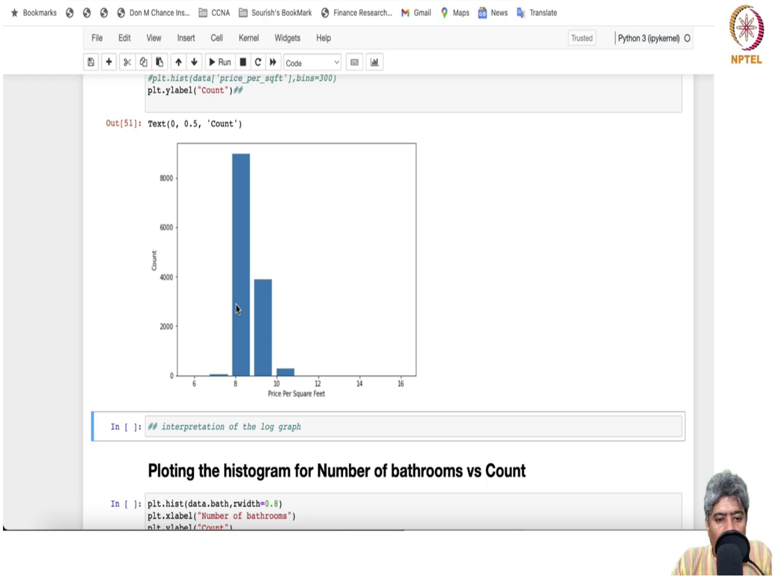
```
In [49]: ## 'price_per_sqft' varies in very wide range.

         max(data['price_per_sqft'])

Out[49]: 12000000.0

In [50]: min(data['price_per_sqft'])

Out[50]: 267.82981328435875
```

Ploting the histogram for Price Per Square Feet vs Count

```
In [51]: plt.hist(np.log(data.price_per_sqft),rwidth=0.8)
         plt.xlabel("Price Per Square Feet")
         #plt.hist(data['price_per_sqft'],bins=300)
         plt.ylabel("Count")##

Out[51]: Text(0, 0.5, 'Count')
```

So, price per square feet, if you see the price per square feet, ok this was, this is the minimum actually. Let me just first plot. This is the really high price per square feet. And, if you create the minimum of it, then the price per square feet is 267. So, the range is very big. That is bit of a weird, I find.

(Refer Slide Time: 19:27)



```
#plt.hist(data['price_per_sqft'],bins=300)
plt.ylabel("Count")##
```

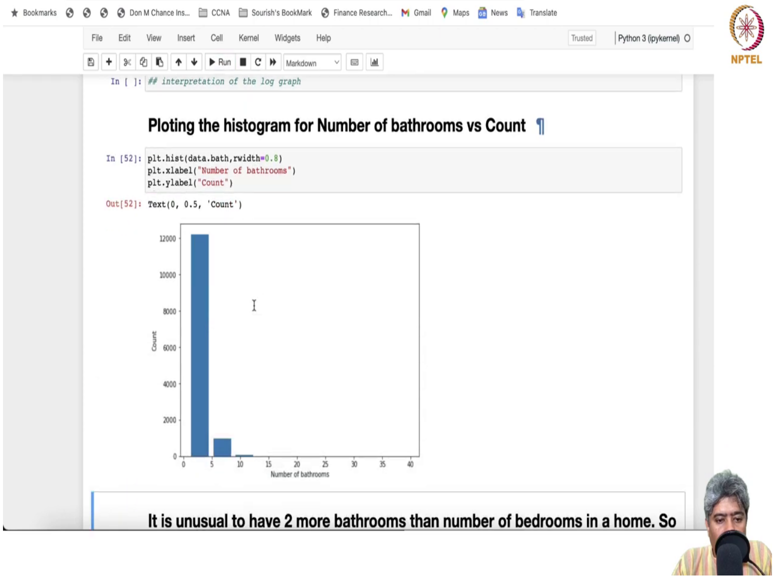
```
Out[51]: Text(0, 0.5, 'Count')
```

```
In [ ]: ## interpretation of the log graph
```

Ploting the histogram for Number of bathrooms vs Count

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

(Refer Slide Time: 19:30)



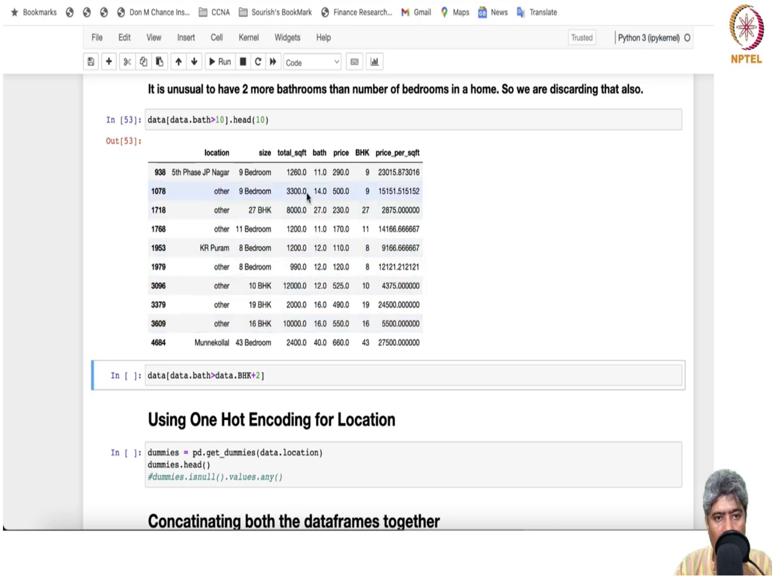
```
In [ ]: ## interpretation of the log graph
```

Ploting the histogram for Number of bathrooms vs Count ¶

```
In [52]: plt.hist(data.bath,rwidth=0.8)
         plt.xlabel("Number of bathrooms")
         plt.ylabel("Count")

Out[52]: Text(0, 0.5, 'Count')
```

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

(Refer Slide Time: 19:36)



It is unusual to have 2 more bathrooms than number of bedrooms in a home. So we are discarding that also.

```
In [53]: data[data.bath>10].head(10)
```

Out[53]:

	location	size	total_sqft	bath	price	BHK	price_per_sqft
938	5th Phase JP Nagar	9 Bedroom	1260.0	11.0	290.0	9	23015.873016
1078	other	9 Bedroom	3300.0	14.0	500.0	9	15151.515152
1718	other	27 BHK	8000.0	27.0	230.0	27	2875.000000
1768	other	11 Bedroom	1200.0	11.0	170.0	11	14166.666667
1953	KR Puram	8 Bedroom	1200.0	12.0	110.0	8	9166.666667
1979	other	8 Bedroom	990.0	12.0	120.0	8	12121.212121
3096	other	10 BHK	12000.0	12.0	525.0	10	4375.000000
3379	other	19 BHK	2000.0	16.0	490.0	19	24500.000000
3609	other	16 BHK	10000.0	16.0	550.0	16	5500.000000
4684	Munnekollal	43 Bedroom	2400.0	40.0	660.0	43	27500.000000

```
In [ ]: data[data.bath>data.BHK+2]
```

Using One Hot Encoding for Location

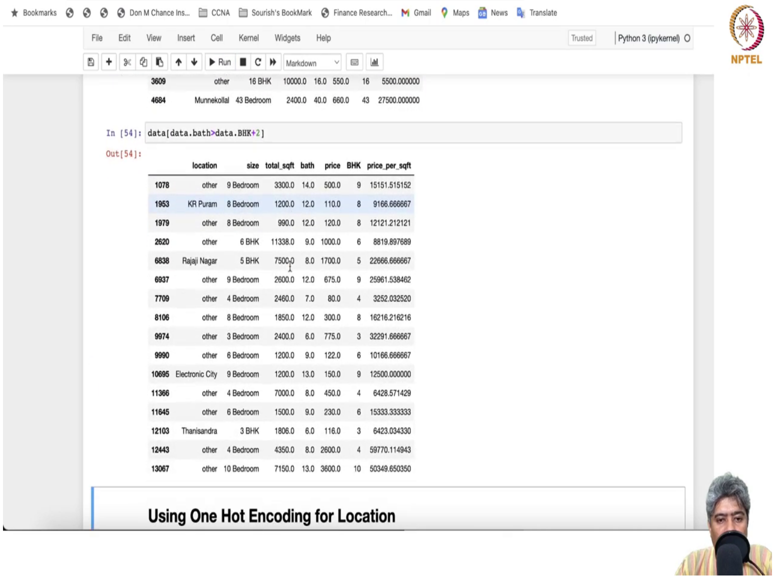
```
In [ ]: dummies = pd.get_dummies(data.location)
        dummies.head()
        #dummies.isnull().values.any()
```

Concatinating both the dataframes together

And, the histogram that you are plotting here for price per square feet and then bathroom for histogram. So, there are number of bedrooms. These are the number of bedrooms and, sorry, number of bathrooms. So, these are the cases where you have number of bedroom is 9 and number of bathroom and number of bedroom is number of bathroom is more than the number of bedroom.

So, these are like really, really big houses with, you know, lot of. So, this could be mansion or something. These are like really big places. So, a really big outlier with all these things so, we can drop this probably.

(Refer Slide Time: 21:04)



3609	other	16 BHK	10000.0	16.0	550.0	16	5500.000000
4684	Munnekollal	43 Bedroom	2400.0	40.0	660.0	43	27500.000000

```
In [54]: data[data.bath>data.BHK+2]
```

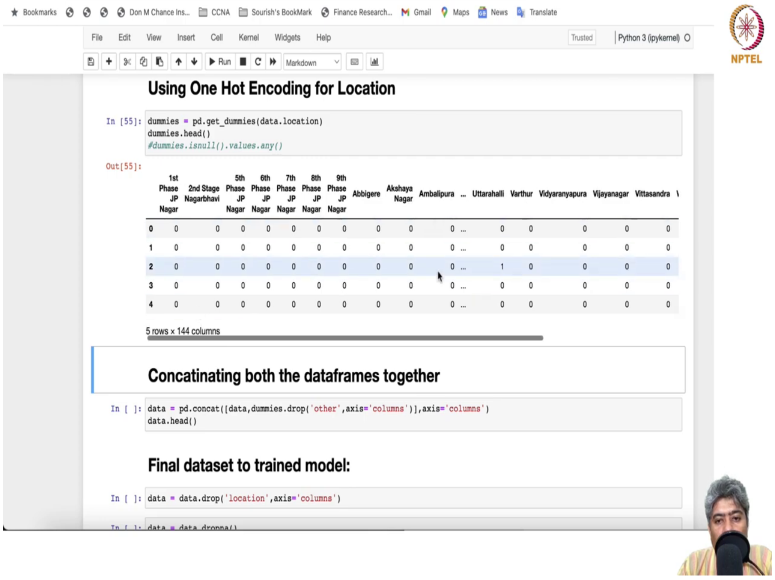
Out[54]:

	location	size	total_sqft	bath	price	BHK	price_per_sqft
1078	other	9 Bedroom	3300.0	14.0	500.0	9	15151.515152
1953	KR Puram	8 Bedroom	1200.0	12.0	110.0	8	9166.666667
1979	other	8 Bedroom	990.0	12.0	120.0	8	12121.212121
2620	other	6 BHK	11338.0	9.0	1000.0	6	8819.897689
6838	Rajaji Nagar	5 BHK	7500.0	8.0	1700.0	5	22666.666667
6937	other	9 Bedroom	2600.0	12.0	675.0	9	25961.538462
7709	other	4 Bedroom	2460.0	7.0	80.0	4	3252.032520
8106	other	8 Bedroom	1850.0	12.0	300.0	8	16216.216216
9974	other	3 Bedroom	2400.0	6.0	775.0	3	32291.666667
9990	other	6 Bedroom	1200.0	9.0	122.0	6	10166.666667
10695	Electronic City	9 Bedroom	1200.0	13.0	150.0	9	12500.000000
11366	other	4 Bedroom	7000.0	8.0	450.0	4	6428.571429
11645	other	6 Bedroom	1500.0	9.0	230.0	6	15333.333333
12103	Thanisandra	3 BHK	1806.0	6.0	116.0	3	6423.034330
12443	other	4 Bedroom	4350.0	8.0	2600.0	4	59770.114943
13067	other	10 Bedroom	7150.0	13.0	3600.0	10	50349.650350

Using One Hot Encoding for Location

These are not really regular cases for sure.

(Refer Slide Time: 21:08)



Using One Hot Encoding for Location

In [55]:
```
dummies = pd.get_dummies(data.location)
dummies.head()
#dummies.isnull().values.any()
```

Out[55]:

	1st Phase JP Nagar	2nd Stage Nagarbhavi	5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar	9th Phase JP Nagar	Abbigere	Akshaya Nagar	Ambalipura	...	Uttarahalli	Varthur	Vidyaranyapura	Vijayanagar	Vittasandra
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 144 columns

Concatinating both the dataframes together

In [ ]:
```
data = pd.concat([data,dummies.drop('other',axis='columns')],axis='columns')
data.head()
```

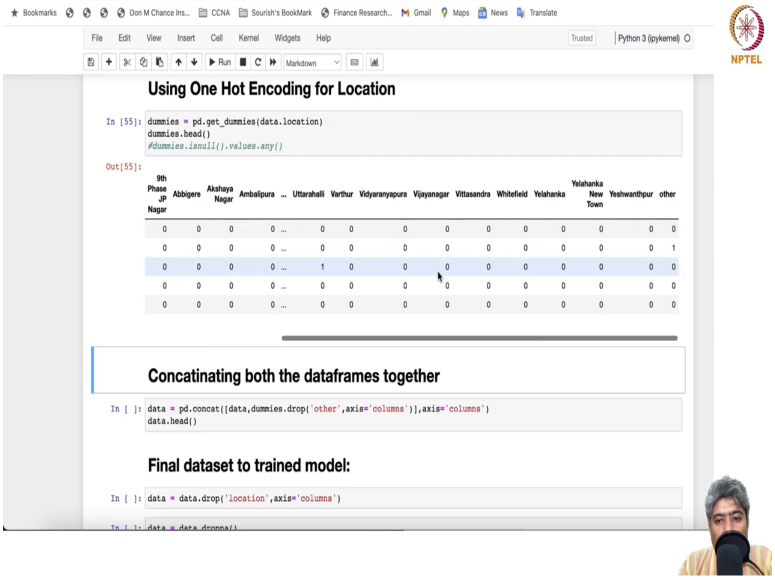
Final dataset to trained model:

In [ ]:
```
data = data.drop('location',axis='columns')
```

In [ ]:
```
data = data.dropna()
```

So, now we are going to do some one-hot encoding for the location. This is very important. So, this gives us one-hot encoding for all the locations.

(Refer Slide Time: 21:31)



Using One Hot Encoding for Location

```
In [55]: dummies = pd.get_dummies(data.location)
         dummies.head()
         #dummies.isnull().values.any()
```

Out[55]:

9th Phase JP Nagar	Abbigere	Akshaya Nagar	Ambalipura	...	Uttarahalli	Varthur	Vidyaranyapura	Vijayanagar	Vittasandra	Whitefield	Yelahanka	Yelahanka New Town	Yeshwanthpur	other
0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
0	0	0	0	...	1	0	0	0	0	0	0	0	0	0
0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Concatinating both the dataframes together

```
In [ ]: data = pd.concat([data,dummies.drop('other',axis='columns')],axis='columns')
        data.head()
```

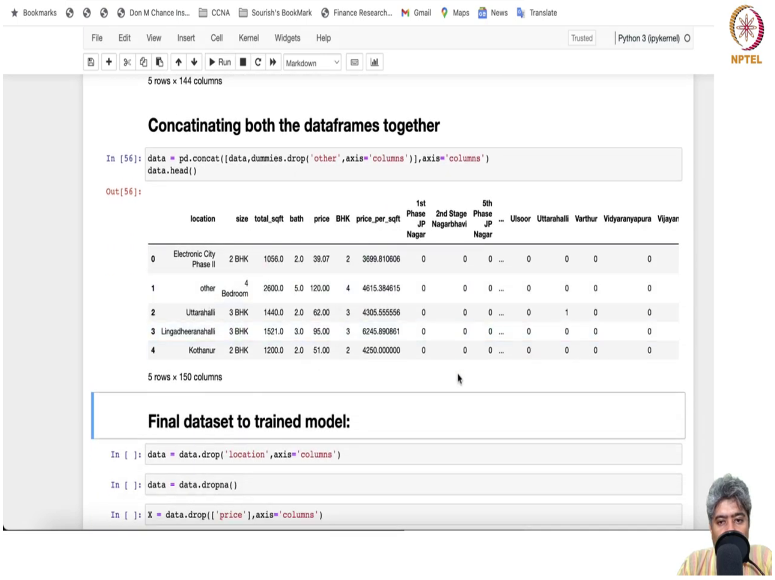
Final dataset to trained model:

```
In [ ]: data = data.drop('location',axis='columns')
```

```
In [ ]: data = data.dropna()
```

5 rows for 144 columns.

(Refer Slide Time: 21:33)



5 rows × 144 columns

Concatinating both the dataframes together

```
In [56]: data = pd.concat([data,dummies.drop('other',axis='columns')],axis='columns')
         data.head()
```

Out[56]:

	location	size	total_sqft	bath	price	BHK	price_per_sqft	1st Phase JP Nagar	2nd Stage Nagarbhavi	5th Phase JP Nagar	...	Ulsoor	Uttarahalli	Varthur	Vidyaranyapura	Vijayan
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699.810606	0	0	0	...	0	0	0	0	
1	other	4 Bedroom	2600.0	5.0	120.00	4	4615.384615	0	0	0	...	0	0	0	0	
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305.555556	0	0	0	...	0	1	0	0	
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245.890861	0	0	0	...	0	0	0	0	
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250.000000	0	0	0	...	0	0	0	0	

5 rows × 150 columns

Final dataset to trained model:

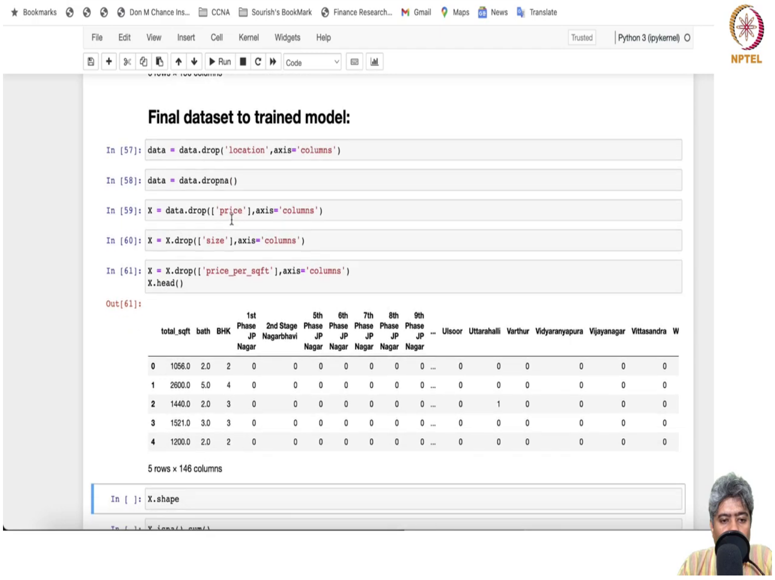
```
In [ ]: data = data.drop('location',axis='columns')
```

```
In [ ]: data = data.dropna()
```

```
In [ ]: X = data.drop(['price'],axis='columns')
```

For each columns, we get the first five rows or all the 144 columns. Ok. And at the end, we get the others. Ok, alright. And, then concatenating with the data frames together, we have to add them, concatenate them. So, we just concatenate with the data frames.

(Refer Slide Time: 21:54)



Final dataset to trained model:

```
In [57]: data = data.drop('location',axis='columns')

In [58]: data = data.dropna()

In [59]: X = data.drop(['price'],axis='columns')

In [60]: X = X.drop(['size'],axis='columns')

In [61]: X = X.drop(['price_per_sqft'],axis='columns')
         X.head()

Out[61]:
```

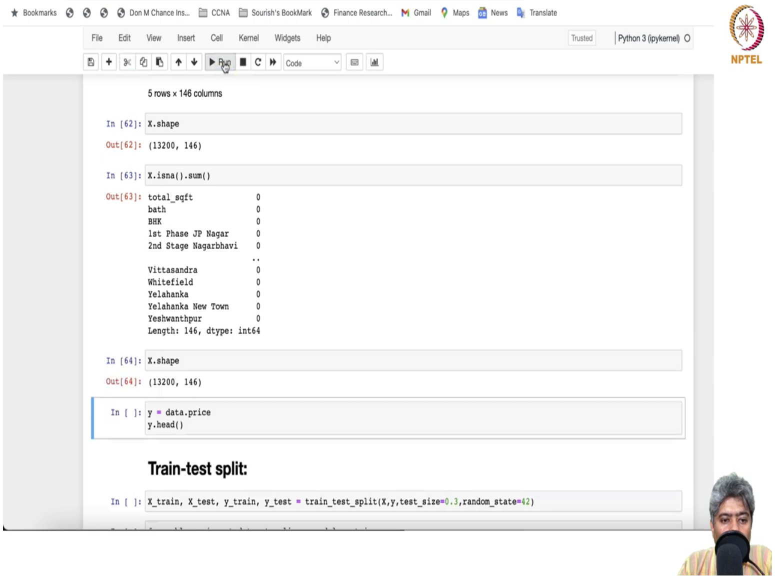
	total_sqft	bath	BHK	1st Phase JP Nagar	2nd Stage Nagarbhavi	5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar	9th Phase JP Nagar	...	Ulsoor	Uttarahalli	Varthur	Vidyaranyapura	Vijayanagar	Vittasandra	W
0	1056.0	2.0	2	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
1	2600.0	5.0	4	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
2	1440.0	2.0	3	0	0	0	0	0	0	0	...	0	1	0	0	0	0	
3	1521.0	3.0	3	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
4	1200.0	2.0	2	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

5 rows × 146 columns

```
In [ ]: X.shape
```

And, then we have to drop the location because now we have for each location, we have done the one-hot encoding, correct? And, we can drop the NAs. Now, we take the, from the, we can drop the price and rest of the, we will become the X matrix. We can drop the size and drop the price per square feet.

(Refer Slide Time: 22:31)



5 rows × 146 columns

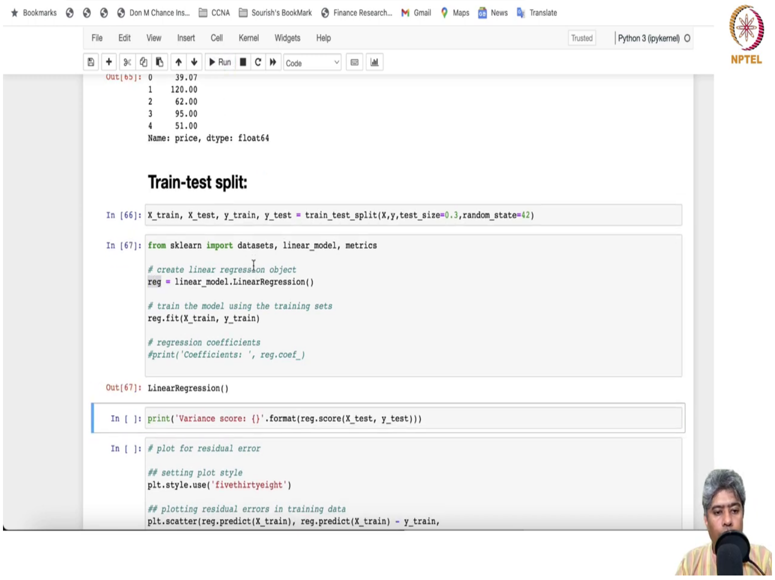
```
In [62]: X.shape

Out[62]: (13200, 146)

In [63]: X.isna().sum()

Out[63]: total_sqft             0
         bath                   0
         BHK                    0
         1st Phase JP Nagar     0
         2nd Stage Nagarbhavi   0
                               ..
         Vittasandra            0
         Whitefield             0
         Yelahanka              0
         Yelahanka New Town     0
         Yeshwanthpur           0
         Length: 146, dtype: int64

In [64]: X.shape

Out[64]: (13200, 146)

In [ ]: y = data.price
        y.head()
```

Train-test split:

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)
```

So, this will be our final X. So, total square feet, bathroom, BHK and the locations. This will be the our thing. So, X shape will be 13200 and 146 columns. So, we have so many columns and yeah. And, there is no NA. Ok. So, now we are going to split the data into train and test.

(Refer Slide Time: 23:09)



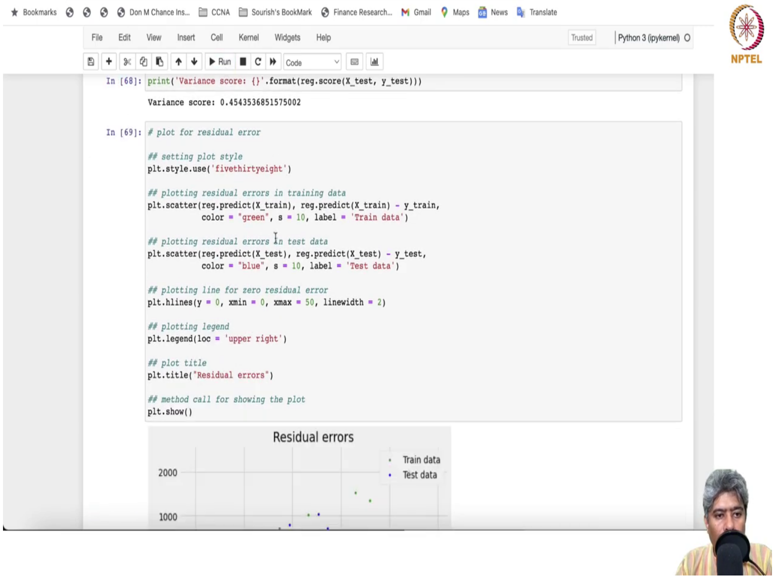
```
Out[65]: 0     39.07
         1    120.00
         2     62.00
         3     95.00
         4     51.00
         Name: price, dtype: float64
```

Train-test split:

```
In [66]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)

In [67]: from sklearn import datasets, linear_model, metrics

         # create linear regression object
         reg = linear_model.LinearRegression()

         # train the model using the training sets
         reg.fit(X_train, y_train)

         # regression coefficients
         #print('Coefficients: ', reg.coef_)

Out[67]: LinearRegression()

In [ ]: print('Variance score: {}'.format(reg.score(X_test, y_test)))

In [ ]: # plot for residual error

        ## setting plot style
        plt.style.use('fivethirtyeight')

        ## plotting residual errors in training data
        plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train,
```

So, we are going to call train test split. Now, if let me just go on the top. Ok. In the, from the sklearn package, scikit-learn package, model selection module, we imported the train and test split here. Ok. So, let us run that. And from there, we are importing the linear model, we linear regression from the linear model of sklearn, we call it regression and then we are called fitting the regression dot fit. Ok.

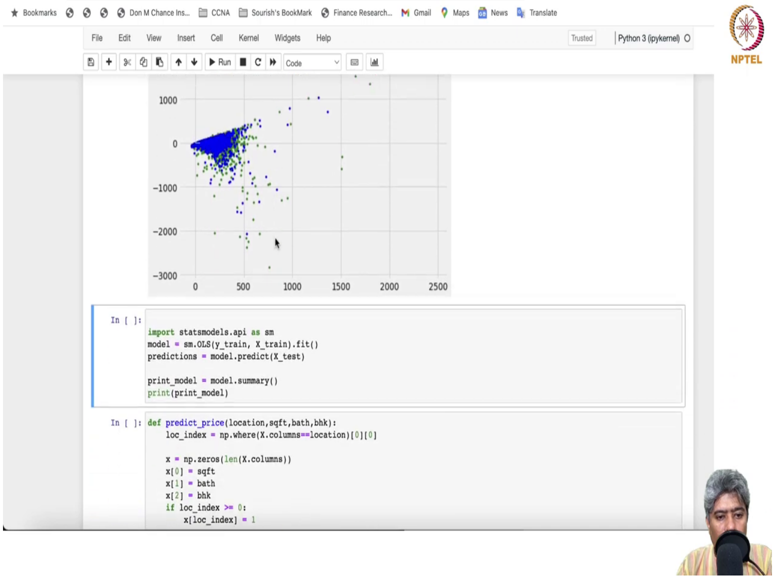
(Refer Slide Time: 24:15)



```
In [68]: print('Variance score: {}'.format(reg.score(X_test, y_test)))

         Variance score: 0.4543536851575002

In [69]: # plot for residual error

         ## setting plot style
         plt.style.use('fivethirtyeight')

         ## plotting residual errors in training data
         plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train,
                     color = "green", s = 10, label = 'Train data')

         ## plotting residual errors in test data
         plt.scatter(reg.predict(X_test), reg.predict(X_test) - y_test,
                     color = "blue", s = 10, label = 'Test data')

         ## plotting line for zero residual error
         plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

         ## plotting legend
         plt.legend(loc = 'upper right')

         ## plot title
         plt.title("Residual errors")

         ## method call for showing the plot
         plt.show()
```

Now so, the variance score is 0.45.

(Refer Slide Time: 24:25)

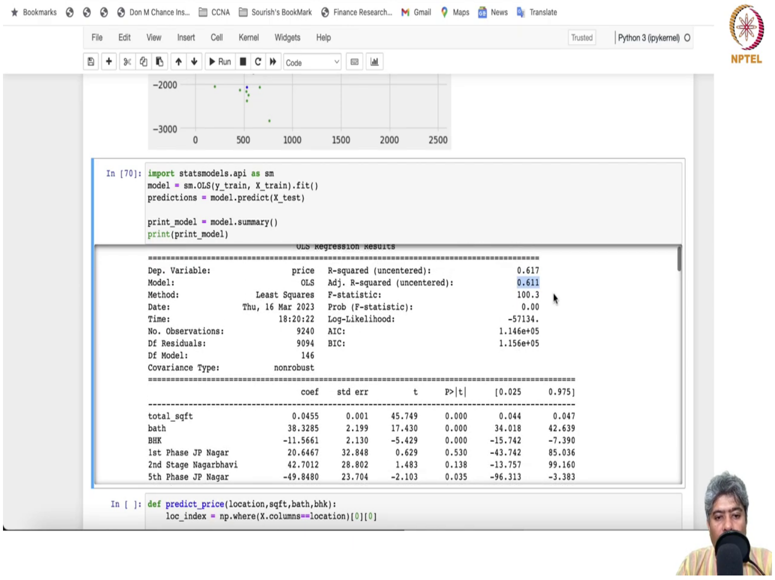


```
In [ ]:
import statsmodels.api as sm
model = sm.OLS(y_train, X_train).fit()
predictions = model.predict(X_test)

print_model = model.summary()
print(print_model)
```

```
In [ ]: 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
```

And, then if we just plot the residual errors for train data versus test data. So, they are somewhat overlap each other. So, which is a good news, which is a good news.

(Refer Slide Time: 24:43)



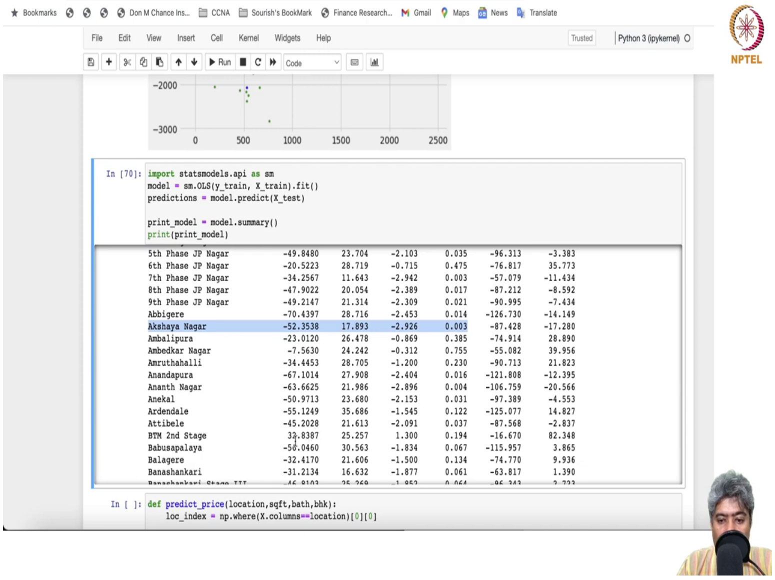
```
In [70]: import statsmodels.api as sm
         model = sm.OLS(y_train, X_train).fit()
         predictions = model.predict(X_test)

         print_model = model.summary()
         print(print_model)
```

```
                            OLS Regression Results
=================================================================================
Dep. Variable:                  price   R-squared (uncentered):             0.617
Model:                            OLS   Adj. R-squared (uncentered):        0.611
Method:                 Least Squares   F-statistic:                        100.3
Date:                Thu, 16 Mar 2023   Prob (F-statistic):                  0.00
Time:                        18:20:22   Log-Likelihood:                   -57134.
No. Observations:                9240   AIC:                            1.146e+05
Df Residuals:                    9094   BIC:                            1.156e+05
Df Model:                         146
Covariance Type:            nonrobust
========================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
total_sqft                 0.0455      0.001     45.749      0.000       0.044       0.047
bath                      38.3285      2.199     17.430      0.000      34.018      42.639
BHK                      -11.5661      2.130     -5.429      0.000     -15.742      -7.390
1st Phase JP Nagar        20.6467     32.848      0.629      0.530     -43.742      85.036
2nd Stage Nagarbhavi      42.7012     28.802      1.483      0.138     -13.757      99.160
5th Phase JP Nagar       -49.8480     23.704     -2.103      0.035     -96.313      -3.383
```

```
In [ ]: def predict_price(location,sqft,bath,bhk):
            loc_index = np.where(X.columns==location)[0][0]
```

And then if we just plot this, the model, if we just fit the model, OLS model. So, adjusted R square is 61. So, 61 percent gets, can be explained by the data.

(Refer Slide Time: 25:05)



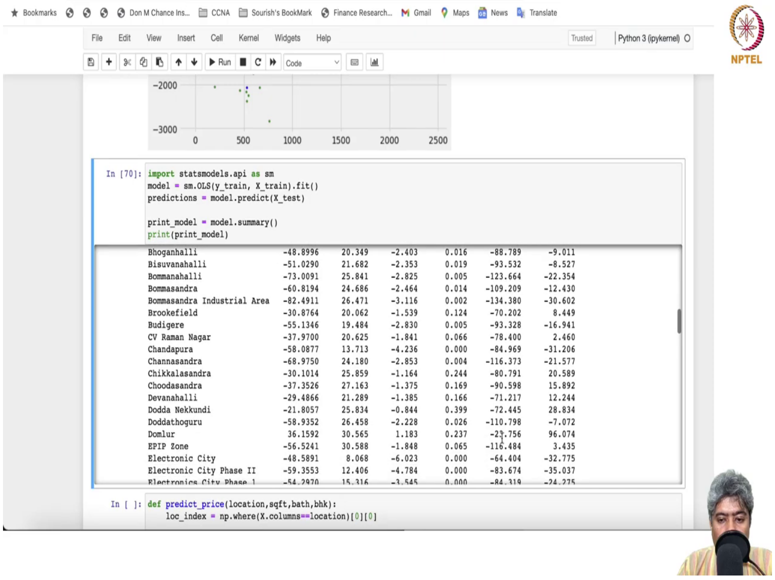
```
In [70]: import statsmodels.api as sm
         model = sm.OLS(y_train, X_train).fit()
         predictions = model.predict(X_test)

         print_model = model.summary()
         print(print_model)
```

```
5th Phase JP Nagar          -49.8480     23.704     -2.103      0.035     -96.313     -3.383
6th Phase JP Nagar          -20.5223     28.719     -0.715      0.475     -76.817     35.773
7th Phase JP Nagar          -34.2567     11.643     -2.942      0.003     -57.079    -11.434
8th Phase JP Nagar          -47.9022     20.054     -2.389      0.017     -87.212     -8.592
9th Phase JP Nagar          -49.2147     21.314     -2.309      0.021     -90.995     -7.434
Abbigere                    -70.4397     28.716     -2.453      0.014    -126.730    -14.149
Akshaya Nagar               -52.3538     17.893     -2.926      0.003     -87.428    -17.280
Ambalipura                  -23.0120     26.478     -0.869      0.385     -74.914     28.890
Ambedkar Nagar               -7.5630     24.242     -0.312      0.755     -55.082     39.956
Amruthahalli                -34.4453     28.705     -1.200      0.230     -90.713     21.823
Anandapura                  -67.1014     27.908     -2.404      0.016    -121.808    -12.395
Ananth Nagar                -63.6625     21.986     -2.896      0.004    -106.759    -20.566
Anekal                      -50.9713     23.680     -2.153      0.031     -97.389     -4.553
Ardendale                   -55.1249     35.686     -1.545      0.122    -125.077     14.827
Attibele                    -45.2028     21.613     -2.091      0.037     -87.568     -2.837
BTM 2nd Stage                32.8387     25.257      1.300      0.194     -16.670     82.348
Babusapalaya                -50.0460     30.563     -1.834      0.067    -115.957      3.865
Balagere                    -32.4170     21.606     -1.500      0.134     -74.770      9.936
Banashankari                -31.2134     16.632     -1.877      0.061     -63.817      1.390
Banashankari Stage III      -46.8103     25.269     -1.852      0.064     -96.343      2.723
```

```
In [ ]: def predict_price(location,sqft,bath,bhk):
            loc_index = np.where(X.columns==location)[0][0]
```

So, the total square feet has a very positive coefficient as the and you know looks, this p value are very small for bathroom and BHK. And, then there are locations for which some p values sometimes are small, small. For example, there is a base line and against the base line. This Akshaya Nagar has a coefficient which is negative 52 and the p value is small. So, that means, it is small enough and the price that you pay is statistically significantly smaller than the base line.

(Refer Slide Time: 26:07)



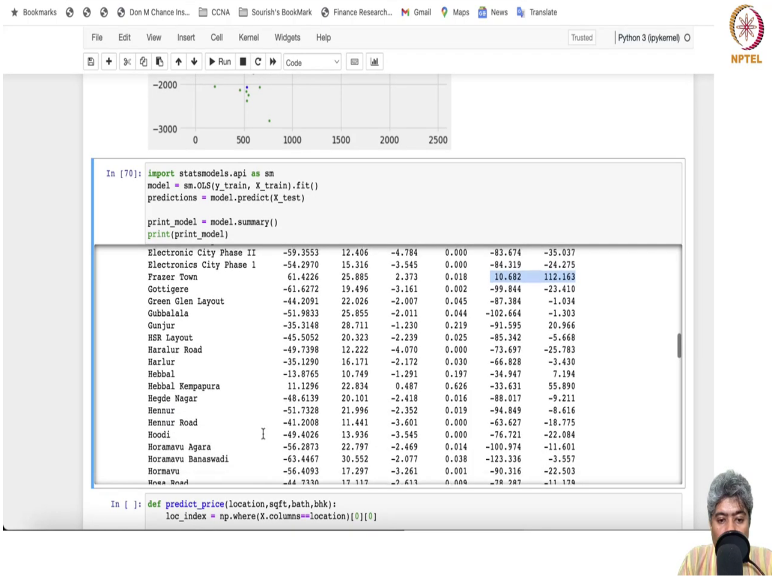
```
In [70]: import statsmodels.api as sm
         model = sm.OLS(y_train, X_train).fit()
         predictions = model.predict(X_test)

         print_model = model.summary()
         print(print_model)
```

```
Bhoganhalli                     -48.8996     20.349     -2.403      0.016     -88.789     -9.011
Bisuvanahalli                   -51.0290     21.682     -2.353      0.019     -93.532     -8.527
Bommanahalli                    -73.0091     25.841     -2.825      0.005    -123.664    -22.354
Bommasandra                     -60.8194     24.686     -2.464      0.014    -109.209    -12.430
Bommasandra Industrial Area     -82.4911     26.471     -3.116      0.002    -134.380    -30.602
Brookefield                     -30.8764     20.062     -1.539      0.124     -70.202      8.449
Budigere                        -55.1346     19.484     -2.830      0.005     -93.328    -16.941
CV Raman Nagar                  -37.9700     20.625     -1.841      0.066     -78.400      2.460
Chandapura                      -58.0877     13.713     -4.236      0.000     -84.969    -31.206
Channasandra                    -68.9750     24.180     -2.853      0.004    -116.373    -21.577
Chikkalasandra                  -30.1014     25.859     -1.164      0.244     -80.791     20.589
Choodasandra                    -37.3526     27.163     -1.375      0.169     -90.598     15.892
Devanahalli                     -29.4866     21.289     -1.385      0.166     -71.217     12.244
Dodda Nekkundi                  -21.8057     25.834     -0.844      0.399     -72.445     28.834
Doddathoguru                    -58.9352     26.458     -2.228      0.026    -110.798     -7.072
Domlur                           36.1592     30.565      1.183      0.237     -23.756     96.074
EPIP Zone                       -56.5241     30.588     -1.848      0.065    -116.484      3.435
Electronic City                 -48.5891      8.068     -6.023      0.000     -64.404    -32.775
Electronic City Phase II        -59.3553     12.406     -4.784      0.000     -83.674    -35.037
Electronics City Phase 1        -54.2970     15.316     -3.545      0.000     -84.319    -24.275
```

```
In [ ]: def predict_price(location,sqft,bath,bhk):
            loc_index = np.where(X.columns==location)[0][0]
```

So, we can look for places where it is bit high or bit low. So, for example, if you see this place has a positive coefficient, but the p value is really not large and the coefficient confidence interval also contains the 0. So that means, is its sort of a base line you know expected according to the expected price, you do not really expect much.

(Refer Slide Time: 26:41)



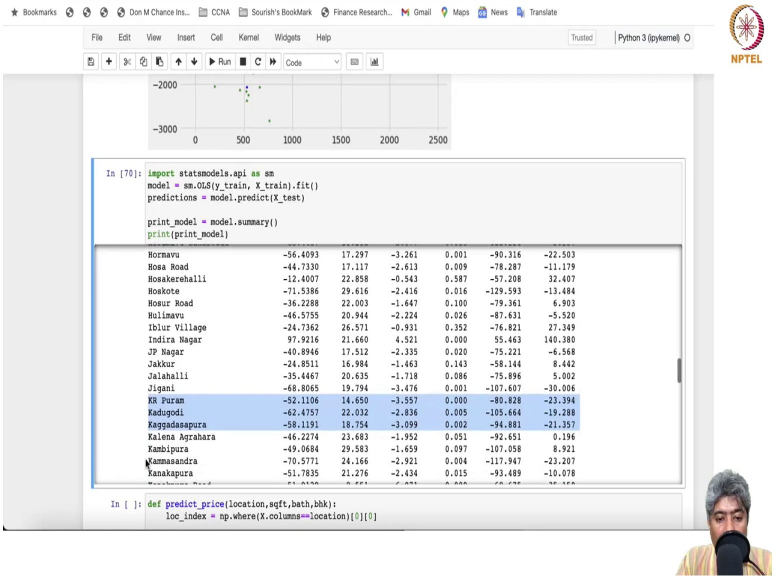
```
In [70]: import statsmodels.api as sm
         model = sm.OLS(y_train, X_train).fit()
         predictions = model.predict(X_test)

         print_model = model.summary()
         print(print_model)
```

```
Electronic City Phase II     -59.3553     12.406     -4.784     0.000    -83.674    -35.037
Electronics City Phase 1     -54.2970     15.316     -3.545     0.000    -84.319    -24.275
Frazer Town                   61.4226     25.885      2.373     0.018     10.682    112.163
Gottigere                    -61.6272     19.496     -3.161     0.002    -99.844    -23.410
Green Glen Layout            -44.2091     22.026     -2.007     0.045    -87.384     -1.034
Gubbalala                    -51.9833     25.855     -2.011     0.044   -102.664     -1.303
Gunjur                       -35.3148     28.711     -1.230     0.219    -91.595     20.966
HSR Layout                   -45.5052     20.323     -2.239     0.025    -85.342     -5.668
Haralur Road                 -49.7398     12.222     -4.070     0.000    -73.697    -25.783
Harlur                       -35.1290     16.171     -2.172     0.030    -66.828     -3.430
Hebbal                       -13.8765     10.749     -1.291     0.197    -34.947      7.194
Hebbal Kempapura              11.1296     22.834      0.487     0.626    -33.631     55.890
Hegde Nagar                  -48.6139     20.101     -2.418     0.016    -88.017     -9.211
Hennur                       -51.7328     21.996     -2.352     0.019    -94.849     -8.616
Hennur Road                  -41.2008     11.441     -3.601     0.000    -63.627    -18.775
Hoodi                        -49.4026     13.936     -3.545     0.000    -76.721    -22.084
Horamavu Agara               -56.2873     22.797     -2.469     0.014   -100.974    -11.601
Horamavu Banaswadi           -63.4467     30.552     -2.077     0.038   -123.336     -3.557
Hormavu                      -56.4093     17.297     -3.261     0.001    -90.316    -22.503
Hosa Road                    -44.7330     17.117     -2.613     0.009    -78.287    -11.179
```

```
In [ ]: def predict_price(location,sqft,bath,bhk):
            loc_index = np.where(X.columns==location)[0][0]
```

So, if you interestingly, Frazer Town is a place which has coefficient positive. That means for Frazer Town, you pay a premium and looks like p value is small and the coefficient confidence interval for the coefficient is also not including 0. So, I do not live in Bangalore. So, the people of Bangalore can tell me whether it makes sense that Frazer Town has a house price which is bit expensives looks like it is expensive place to be.

(Refer Slide Time: 27:26)



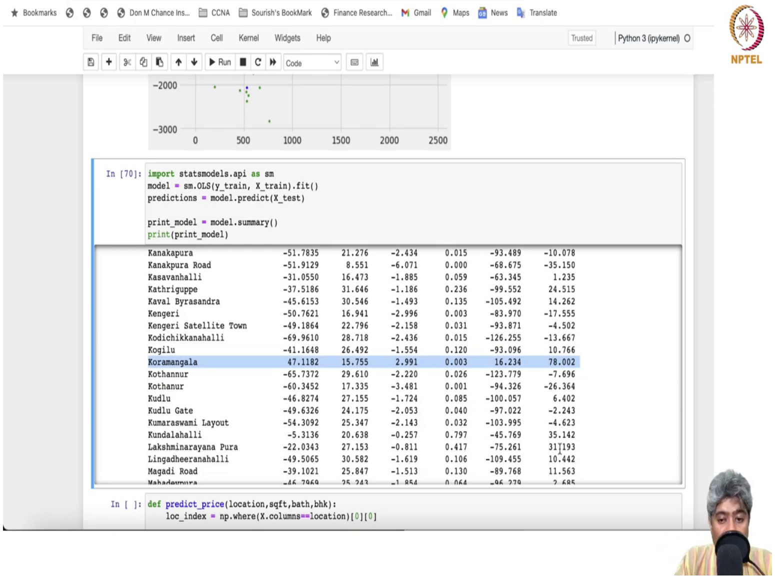
```
In [70]: import statsmodels.api as sm
         model = sm.OLS(y_train, X_train).fit()
         predictions = model.predict(X_test)

         print_model = model.summary()
         print(print_model)
```

```
Hormavu                   -56.4093     17.297     -3.261      0.001     -90.316     -22.503
Hosa Road                 -44.7330     17.117     -2.613      0.009     -78.287     -11.179
Hosakerehalli             -12.4007     22.858     -0.543      0.587     -57.208      32.407
Hoskote                   -71.5386     29.616     -2.416      0.016    -129.593     -13.484
Hosur Road                -36.2288     22.003     -1.647      0.100     -79.361       6.903
Hulimavu                  -46.5755     20.944     -2.224      0.026     -87.631      -5.520
Iblur Village             -24.7362     26.571     -0.931      0.352     -76.821      27.349
Indira Nagar               97.9216     21.660      4.521      0.000      55.463     140.380
JP Nagar                  -40.8946     17.512     -2.335      0.020     -75.221      -6.568
Jakkur                    -24.8511     16.984     -1.463      0.143     -58.144       8.442
Jalahalli                 -35.4467     20.635     -1.718      0.086     -75.896       5.002
Jigani                    -68.8065     19.794     -3.476      0.001    -107.607     -30.006
KR Puram                  -52.1106     14.650     -3.557      0.000     -80.828     -23.394
Kadugodi                  -62.4757     22.032     -2.836      0.005    -105.664     -19.288
Kaggadasapura             -58.1191     18.754     -3.099      0.002     -94.881     -21.357
Kalena Agrahara           -46.2274     23.683     -1.952      0.051     -92.651       0.196
Kambipura                 -49.0684     29.583     -1.659      0.097    -107.058       8.921
Kammasandra               -70.5771     24.166     -2.921      0.004    -117.947     -23.207
Kanakapura                -51.7835     21.276     -2.434      0.015     -93.489     -10.078
```

```
In [ ]: def predict_price(location,sqft,bath,bhk):
            loc_index = np.where(X.columns==location)[0][0]
```

Let us see how it is. For example, Jigani is a place which has a coefficient value negative 68. That means, house price is lower than the expected average prices and then coefficient is also negative, quite low..

(Refer Slide Time: 28:01)



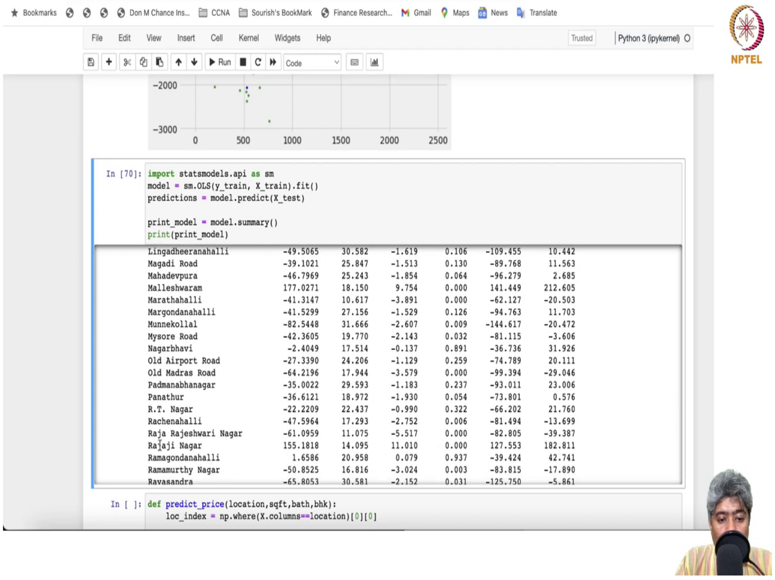
```
In [70]: import statsmodels.api as sm
         model = sm.OLS(y_train, X_train).fit()
         predictions = model.predict(X_test)

         print_model = model.summary()
         print(print_model)
```

```
Kanakapura                -51.7835     21.276     -2.434      0.015    -93.489    -10.078
Kanakpura Road            -51.9129      8.551     -6.071      0.000    -68.675    -35.150
Kasavanhalli              -31.0550     16.473     -1.885      0.059    -63.345      1.235
Kathriguppe               -37.5186     31.646     -1.186      0.236    -99.552     24.515
Kaval Byrasandra          -45.6153     30.546     -1.493      0.135   -105.492     14.262
Kengeri                   -50.7621     16.941     -2.996      0.003    -83.970    -17.555
Kengeri Satellite Town    -49.1864     22.796     -2.158      0.031    -93.871     -4.502
Kodichikkanahalli         -69.9610     28.718     -2.436      0.015   -126.255    -13.667
Kogilu                    -41.1648     26.492     -1.554      0.120    -93.096     10.766
Koramangala                47.1182     15.755      2.991      0.003     16.234     78.002
Kothannur                 -65.7372     29.610     -2.220      0.026   -123.779     -7.696
Kothanur                  -60.3452     17.335     -3.481      0.001    -94.326    -26.364
Kudlu                     -46.8274     27.155     -1.724      0.085   -100.057      6.402
Kudlu Gate                -49.6326     24.175     -2.053      0.040    -97.022     -2.243
Kumaraswami Layout        -54.3092     25.347     -2.143      0.032   -103.995     -4.623
Kundalahalli               -5.3136     20.638     -0.257      0.797    -45.769     35.142
Lakshminarayana Pura      -22.0343     27.153     -0.811      0.417    -75.261     31.193
Lingadheeranahalli        -49.5065     30.582     -1.619      0.106   -109.455     10.442
Magadi Road               -39.1021     25.847     -1.513      0.130    -89.768     11.563
Mahadevpura               -46.7969     25.243     -1.854      0.064    -96.279      2.685
```

```
In [ ]: def predict_price(location,sqft,bath,bhk):
            loc_index = np.where(X.columns==location)[0][0]
```

KR Puram, Kadugodi, these are all places where you have prices to be little on the lower side. Now, here is one place, Koramangala looks like it is a bit expensive place.

(Refer Slide Time: 28:43)



```
In [70]: import statsmodels.api as sm
         model = sm.OLS(y_train, X_train).fit()
         predictions = model.predict(X_test)

         print_model = model.summary()
         print(print_model)
```

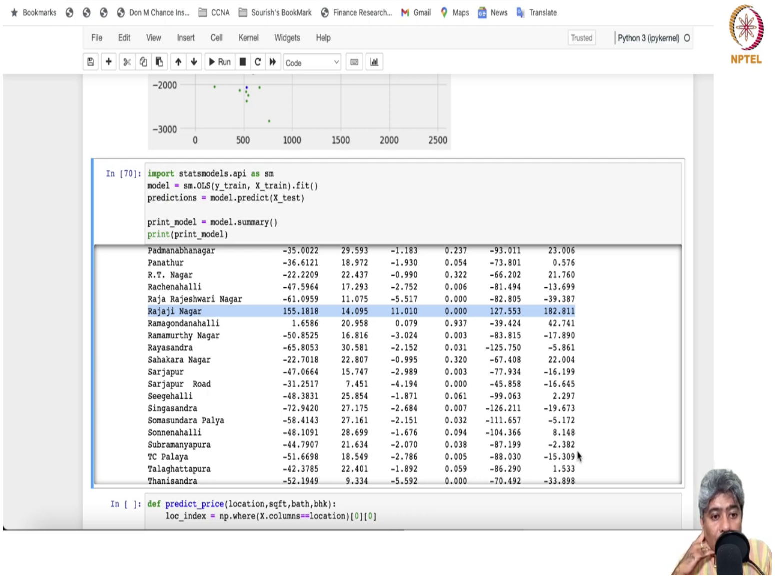
```
Lingadheeranahalli         -49.5065     30.582     -1.619     0.106   -109.455     10.442
Magadi Road                -39.1021     25.847     -1.513     0.130    -89.768     11.563
Mahadevpura                -46.7969     25.243     -1.854     0.064    -96.279      2.685
Malleshwaram               177.0271     18.150      9.754     0.000    141.449    212.605
Marathahalli               -41.3147     10.617     -3.891     0.000    -62.127    -20.503
Margondanahalli            -41.5299     27.156     -1.529     0.126    -94.763     11.703
Munnekollal                -82.5448     31.666     -2.607     0.009   -144.617    -20.472
Mysore Road                -42.3605     19.770     -2.143     0.032    -81.115     -3.606
Nagarbhavi                  -2.4049     17.514     -0.137     0.891    -36.736     31.926
Old Airport Road           -27.3390     24.206     -1.129     0.259    -74.789     20.111
Old Madras Road            -64.2196     17.944     -3.579     0.000    -99.394    -29.046
Padmanabhanagar            -35.0022     29.593     -1.183     0.237    -93.011     23.006
Panathur                   -36.6121     18.972     -1.930     0.054    -73.801      0.576
R.T. Nagar                 -22.2209     22.437     -0.990     0.322    -66.202     21.760
Rachenahalli               -47.5964     17.293     -2.752     0.006    -81.494    -13.699
Raja Rajeshwari Nagar      -61.0959     11.075     -5.517     0.000    -82.805    -39.387
Rajaji Nagar               155.1818     14.095     11.010     0.000    127.553    182.811
Ramagondanahalli             1.6586     20.958      0.079     0.937    -39.424     42.741
Ramamurthy Nagar           -50.8525     16.816     -3.024     0.003    -83.815    -17.890
Ravasandra                 -65.8053     30.581     -2.152     0.031   -125.750     -5.861
```

```
In [ ]: def predict_price(location,sqft,bath,bhk):
            loc_index = np.where(X.columns==location)[0][0]
```

Because, the coefficient is positive 47.112, p value is quite small and the confidence interval for the coefficient is completely right side of the 0. Does not include 0. So, the people from Bangalore tell me if Koramangala is a place where housing price is more than expected. At least the model is saying it is bit expensive place ok. There is one place looks really expensive Rajaji Nagar.

The coefficient is very high, 155.1818. And, then p value is really really small and interesting and the coefficient is completely off. One confidence interval of the coefficient is somewhere between 127 and 182. Guys, this place looks like really expensive. Tell me I do not live in Bangalore, I live in I am from Chennai. I teach at Chennai Math Institute.

(Refer Slide Time: 29:26)



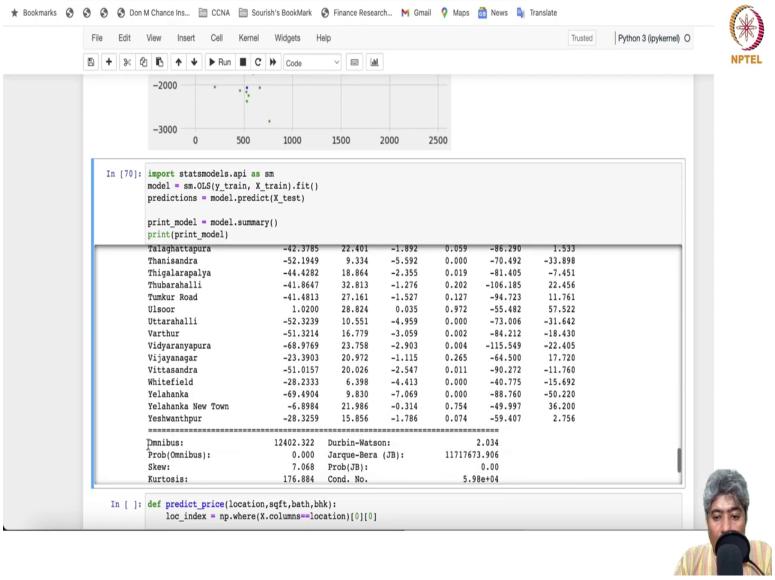
```
In [70]: import statsmodels.api as sm
         model = sm.OLS(y_train, X_train).fit()
         predictions = model.predict(X_test)

         print_model = model.summary()
         print(print_model)
```

```
Padmanabhanagar            -35.0022     29.593     -1.183      0.237     -93.011     23.006
Panathur                   -36.6121     18.972     -1.930      0.054     -73.801      0.576
R.T. Nagar                 -22.2209     22.437     -0.990      0.322     -66.202     21.760
Rachenahalli               -47.5964     17.293     -2.752      0.006     -81.494    -13.699
Raja Rajeshwari Nagar      -61.0959     11.075     -5.517      0.000     -82.805    -39.387
Rajaji Nagar               155.1818     14.095     11.010      0.000     127.553    182.811
Ramagondanahalli             1.6586     20.958      0.079      0.937     -39.424     42.741
Ramamurthy Nagar           -50.8525     16.816     -3.024      0.003     -83.815    -17.890
Rayasandra                 -65.8053     30.581     -2.152      0.031    -125.750     -5.861
Sahakara Nagar             -22.7018     22.807     -0.995      0.320     -67.408     22.004
Sarjapur                   -47.0664     15.747     -2.989      0.003     -77.934    -16.199
Sarjapur  Road             -31.2517      7.451     -4.194      0.000     -45.858    -16.645
Seegehalli                 -48.3831     25.854     -1.871      0.061     -99.063      2.297
Singasandra                -72.9420     27.175     -2.684      0.007    -126.211    -19.673
Somasundara Palya          -58.4143     27.161     -2.151      0.032    -111.657     -5.172
Sonnenahalli               -48.1091     28.699     -1.676      0.094    -104.366      8.148
Subramanyapura             -44.7907     21.634     -2.070      0.038     -87.199     -2.382
TC Palaya                  -51.6698     18.549     -2.786      0.005     -88.030    -15.309
Talaghattapura             -42.3785     22.401     -1.892      0.059     -86.290      1.533
Thanisandra                -52.1949      9.334     -5.592      0.000     -70.492    -33.898
```

```
In [ ]: def predict_price(location,sqft,bath,bhk):
            loc_index = np.where(X.columns==location)[0][0]
```

So, people from Bangalore can tell me whether this place is really expensive place in terms of house price. Ok. So, ok.

(Refer Slide Time: 29:47)



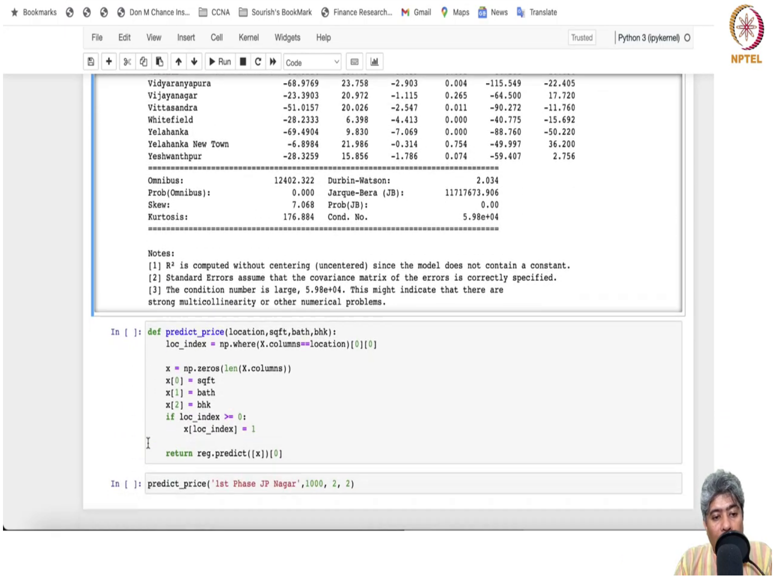
```
In [70]: import statsmodels.api as sm
         model = sm.OLS(y_train, X_train).fit()
         predictions = model.predict(X_test)

         print_model = model.summary()
         print(print_model)
```

```
Talaghattapura             -42.3785     22.401     -1.892      0.059     -86.290       1.533
Thanisandra                -52.1949      9.334     -5.592      0.000     -70.492     -33.898
Thigalarapalya             -44.4282     18.864     -2.355      0.019     -81.405      -7.451
Thubarahalli               -41.8647     32.813     -1.276      0.202    -106.185      22.456
Tumkur Road                -41.4813     27.161     -1.527      0.127     -94.723      11.761
Ulsoor                       1.0200     28.824      0.035      0.972     -55.482      57.522
Uttarahalli                -52.3239     10.551     -4.959      0.000     -73.006     -31.642
Varthur                    -51.3214     16.779     -3.059      0.002     -84.212     -18.430
Vidyaranyapura             -68.9769     23.758     -2.903      0.004    -115.549     -22.405
Vijayanagar                -23.3903     20.972     -1.115      0.265     -64.500      17.720
Vittasandra                -51.0157     20.026     -2.547      0.011     -90.272     -11.760
Whitefield                 -28.2333      6.398     -4.413      0.000     -40.775     -15.692
Yelahanka                  -69.4904      9.830     -7.069      0.000     -88.760     -50.220
Yelahanka New Town          -6.8984     21.986     -0.314      0.754     -49.997      36.200
Yeshwanthpur               -28.3259     15.856     -1.786      0.074     -59.407       2.756
==============================================================================
Omnibus:                    12402.322   Durbin-Watson:                   2.034
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         11717673.906
Skew:                           7.068   Prob(JB):                         0.00
Kurtosis:                     176.884   Cond. No.                     5.98e+04
```

```
In [ ]: def predict_price(location,sqft,bath,bhk):
            loc_index = np.where(X.columns==location)[0][0]
```

So, I think you guys can figure out whether you live in a not so expensive place or not so or really expensive place.

(Refer Slide Time: 29:59)



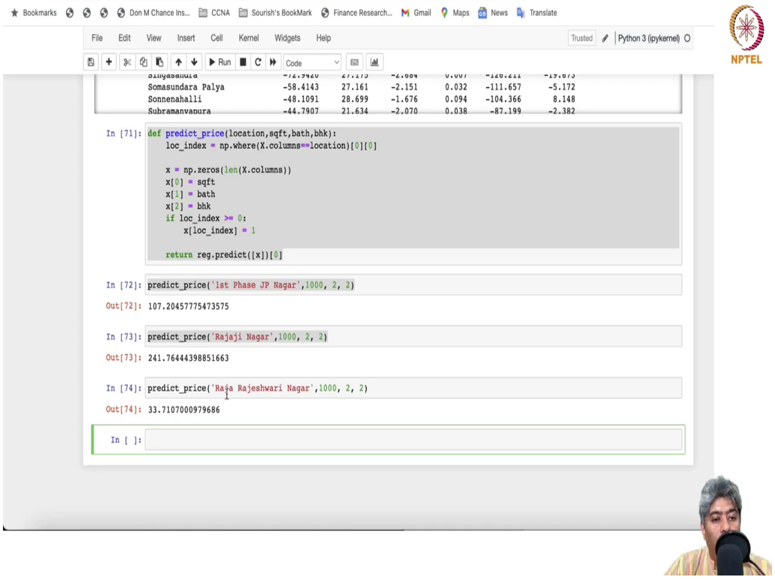
```
Vidyaranyapura             -68.9769     23.758     -2.903      0.004   -115.549    -22.405
Vijayanagar                -23.3903     20.972     -1.115      0.265    -64.500     17.720
Vittasandra                -51.0157     20.026     -2.547      0.011    -90.272    -11.760
Whitefield                 -28.2333      6.398     -4.413      0.000    -40.775    -15.692
Yelahanka                  -69.4904      9.830     -7.069      0.000    -88.760    -50.220
Yelahanka New Town          -6.8984     21.986     -0.314      0.754    -49.997     36.200
Yeshwanthpur               -28.3259     15.856     -1.786      0.074    -59.407      2.756
==============================================================================
Omnibus:                    12402.322   Durbin-Watson:                   2.034
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         11717673.906
Skew:                           7.068   Prob(JB):                         0.00
Kurtosis:                     176.884   Cond. No.                     5.98e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 5.98e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
In [ ]: 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 reg.predict([x])[0]
```

```
In [ ]: predict_price('1st Phase JP Nagar',1000, 2, 2)
```

Alright. Now, suppose we want to place predict price for a place where I will go give location, square feet, bath and BHK and what could be the price. So, this piece of code, this function will write give you the that price.

(Refer Slide Time: 30:30)



```
              Singasandra          -72.9420    27.175    -2.684    0.007    -126.211    -19.673
              Somasundara Palya    -58.4143    27.161    -2.151    0.032    -111.657     -5.172
              Sonnenahalli         -48.1091    28.699    -1.676    0.094    -104.366      8.148
              Subramanyapura       -44.7907    21.634    -2.070    0.038     -87.199     -2.382

In [71]: 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 reg.predict([x])[0]

In [72]: predict_price('1st Phase JP Nagar',1000, 2, 2)

Out[72]: 107.20457775473575

In [73]: predict_price('Rajaji Nagar',1000, 2, 2)

Out[73]: 241.76444398851663

In [74]: predict_price('Raja Rajeshwari Nagar',1000, 2, 2)

Out[74]: 33.7107000979686

In [ ]:
```

And if I just say predict, it will give you the 107 lakhs. So, if it is first phase of JP Nagar, 1000 square feet apartment with 2 BHK, 2 bathroom and 2 BHK. Then, it will cost you 107 lakh almost 1 point which is 1 crore 7 lakh rupees expected price. So, we can play with similarly you can take this say if you go to Rajaji Nagar, I think this was the place which was we found bit expensive.

Let us take Rajaji Nagar ok. For the same apartment for 1000 square feet and 2 bathroom, 2 BHK, it will cost you 2.4 crore or 241 lakh. It is very expensive place to be whereas, if you in some other place maybe Raja Rajeshwari Nagar. Let us see how it displays. You need only 33 lakh rupees to buy apartment in Raja Rajeshwari Nagar.

So, we can see that you know based on the location, the price same apartment can cost very different. Guys, I have no clue whether these values what this model is predicting, it does make sense or not. So, tell me if this makes sense you guys and let me know if it makes sense, then we will know that ok these models these predictive models do work and that will be really fun. So, I will stop here.

Thank you very much. See you in the next video.