IT17157674 – V.Aperame

MLCS (Machine Learning for Cyber Security)

Assignment – Price Prediction for Real Estate

The real estate sector is a sector whose reach is vast, it is therefore affected by many social, political and economic factors which means that there is a huge amount of complex data available. The success of data science in real estate is however dependent on the quality of the data and how it can be accessed. In the near future, it will be able to not only improve a business strategy but also improve the way and quality of our lives. Models that allow this have been developed by using a variety of public data and market information such as prices per square metre on past transactions, number of bedrooms and the quality of the neighbourhood. This provides clients with much wider parameters than could be provided by a single estate agent and allows the estate agent to offer greater accuracy in the information. The real estate sector is widely expanded with the help of Information Technology, Data Science, Machine Learning etc. which also helps the customers not only to do the real estate business in a particular place, also helps to do this real estate business as a world-wide.

I have done the assignment which is based on real life, Price Prediction for Real Estate. To do the prediction for the real estate, I got the data set from "Kaggle.com". The data sets for Real Estate in "Kaggle.com" are found in places like United States, Australia, India, United Kingdom etc. I searched real estate data sets for Sri Lanka. As there aren't any data sets available, I got the data sets for Bengaluru (India) and I did the price prediction for the Bengaluru Real Estate.

I searched for "Real Estate Price Prediction in India" in Kaggle.com, and I found the results as "Help with Real Estate Closed Price Model", "Riga real estate dataset", "Real Estate", "Bengaluru real estate data" etc. In the above searched results, I selected the "Bengaluru real estate data" in order to get the data set which is needed for to do my project. As there are many data set available for real estate in India, I selected the "Bengaluru real estate data", and with the help of the data set which I got, I predicted the estimated price for the real estate.

The "Real Estate" consists of property consisting of land and the buildings on it, along with its natural resources. The major categories of Real Estate are Land, Buildings, Houses etc. The following are the overview of the Price Prediction for Real Estate Project. In this I have selected "Bengaluru Real Estate dataset" from "Kaggle.com". This is shown in the following (Figure 1).

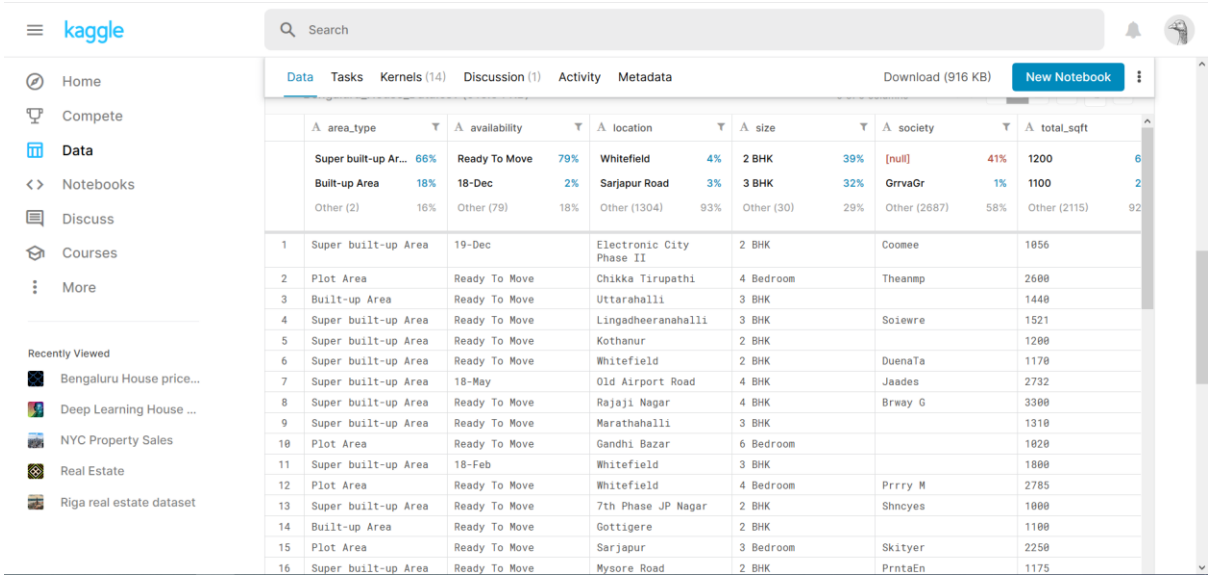


Figure 1: Bengaluru Real Estate Dataset from "Kaggle.com"

With the help of the data sets, which I gained, I built a machine learning model. Through this I gained concepts like Data Cleaning, Feature Engineering, One Hot Encoding, Outlier Detection, Dimensionality Reduction, GridSearchCV etc. Brief understanding of the concepts which I mentioned above are as follows.

- **Data Cleaning:** Process of identifying the incorrect, incomplete, inaccurate, irrelevant or missing part of the data and then modifying, replacing or deleting them according to the necessity.
- **Feature Engineering:** Process of transforming raw data into features which better represents the underlying problem to the predictive models, which results in improved model accuracy on unseen data. The main usage of Feature Engineering helps to turn the inputs into things which the algorithm can understand.
- **One Hot Encoding:** It is a process by which categorical variables are converted into a form that could be provided to Machine Learning algorithms for a better prediction.
- **Outlier Detection:** It is a process of detecting and subsequently excluding outliers from a given set of data.
- **Dimensionality Reduction:** It is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.
- **GridSearchCV:** It is a library function that is a member of sklearn's model selection package. It helps to loop through predefined hyper parameters and it fit the estimator of the built model on the training set.

After I built the model I exported it into a "Pickle" file, and with the help of the Python flask server I predicted the price. I used "Python" as the programming language. Also "Pandas" is used for Data Cleaning, "Matplotlib" is used for Data Visualisation, "Sklearn" for model building, "Python Flask Server" as the backend server, "HTML" (HyperText Markup Language), "CSS" (Cascading Style Sheets) and "JavaScript" for the development of the Real Estate Price Prediction website.

In this Real Estate Price Prediction, I used the Bengaluru Price Prediction dataset which I downloaded from “Kaggle.com” (Already mentioned above). Then, I clicked the “Download” Button, and downloaded the .csv file. This is shown in (Figure 2).

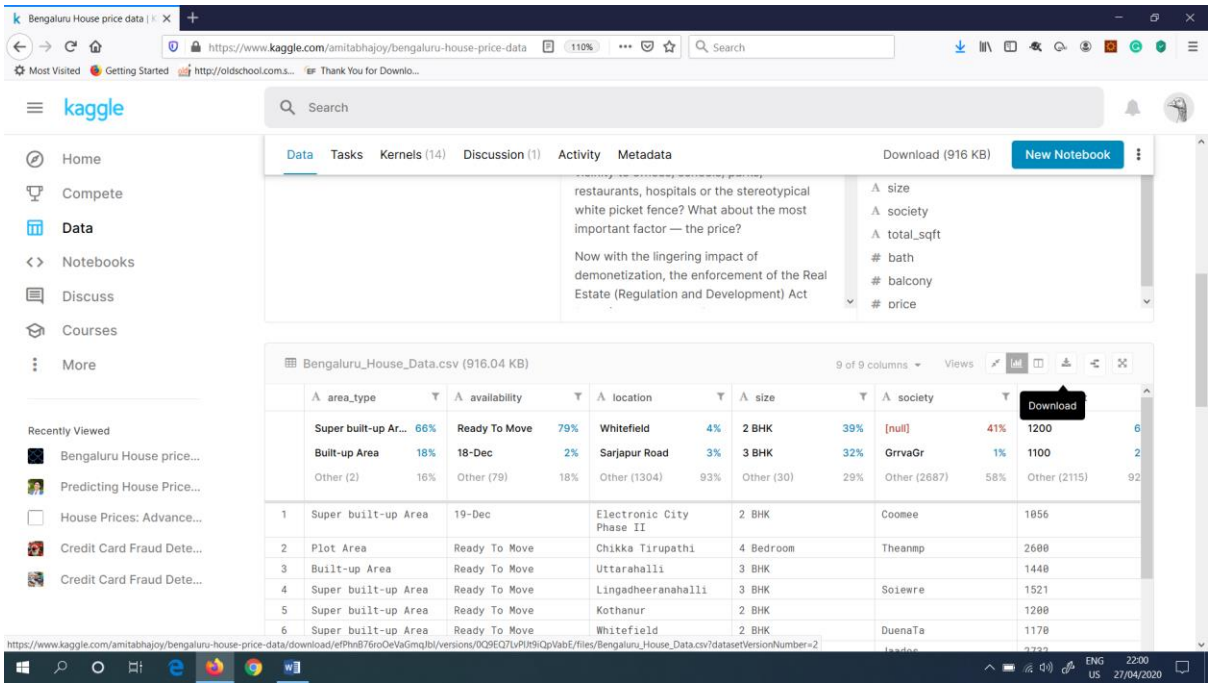


Figure 2: Clicked the “Download” Button to download the .csv file

(Figure 3) shows the downloaded data set for the Real Estate Price Prediction.

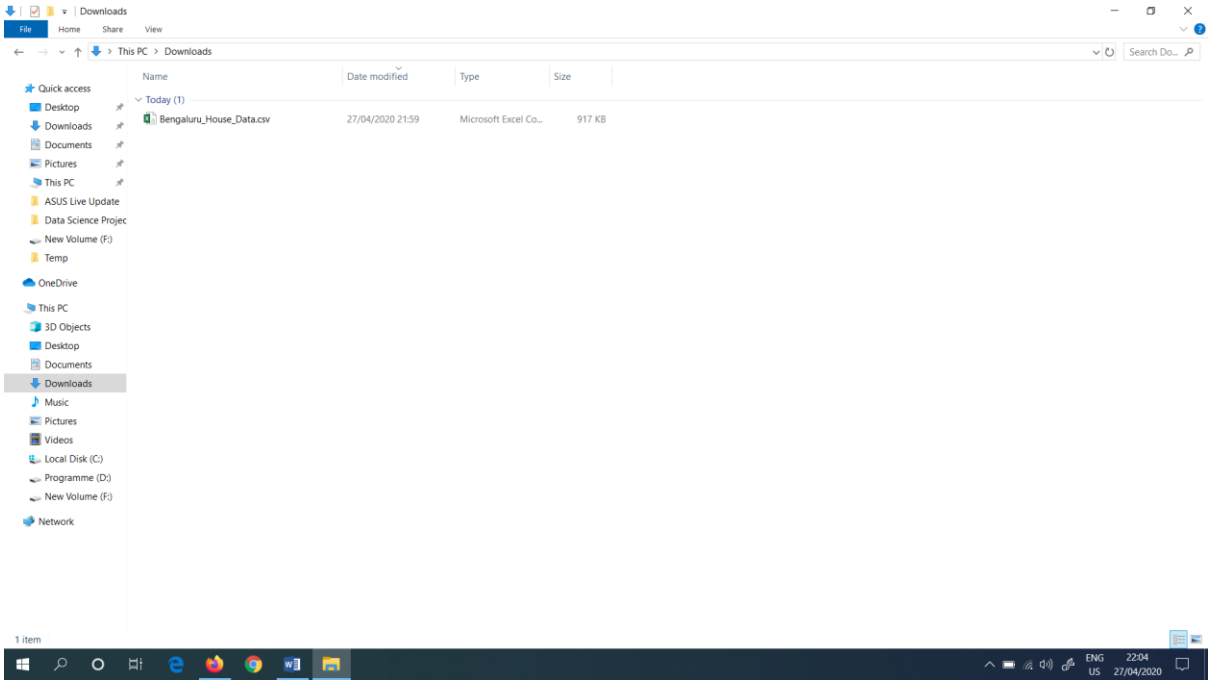
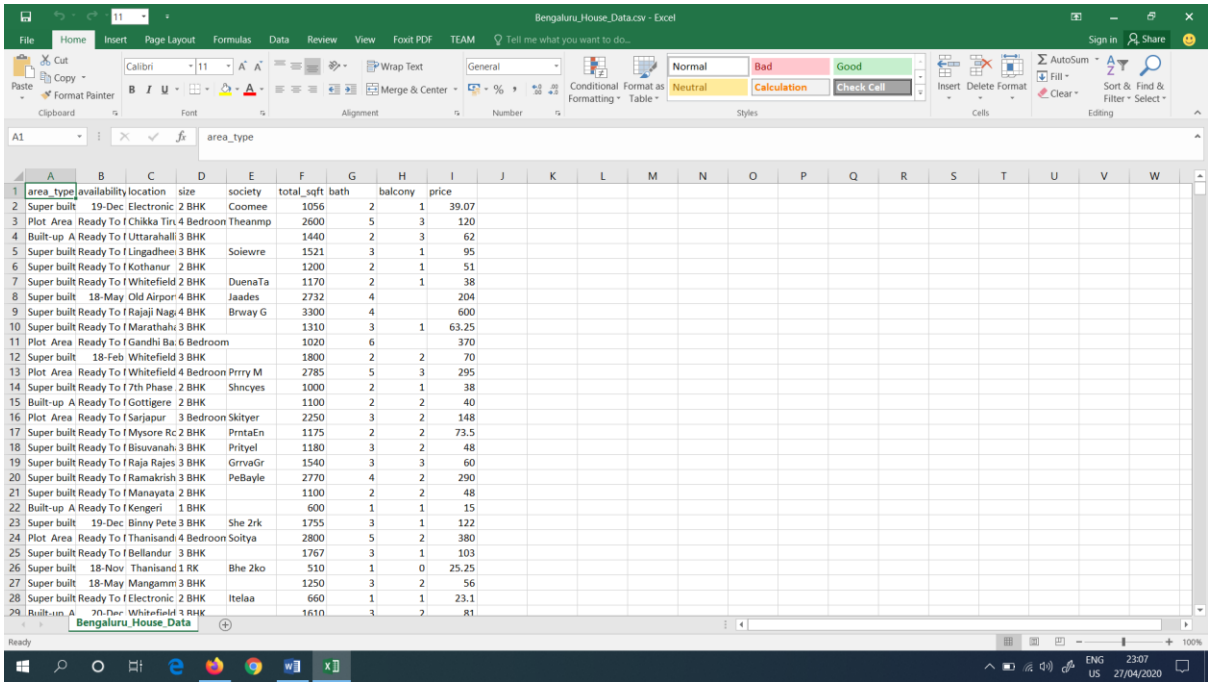


Figure 3: Downloaded .csv file

Inside the .csv file which I downloaded the following are the independent variables. Namely, area type, availability, location, size, society, total sqft (Square feet), bath and balcony. And

the dependent variable is price. So, I planned to predict this dependent variable price. It is presented in Indian Lakh Rupees.

The following (Figure 4) shows the available fields presented in the downloaded csv file.



area_type	availability	location	size	society	total_sqft	bath	balcony	price
Super built	19-Dec	Electronic	2 BHK	Coomee	1056	2	1	39.07
Plot Area	Ready To	Chikka Tir	4 Bedroom	Theanmp	2600	5	3	120
Built-up A	Ready To	Uttarahall	3 BHK		1440	2	3	62
Super built	Ready To	Lingadhee	3 BHK	Soiewre	1521	3	1	95
Super built	Ready To	Kothanur	2 BHK		1200	2	1	51
Super built	Ready To	Whitefield	2 BHK	DuenaTa	1170	2	1	38
Super built	18-May	Old Airpor	4 BHK	Jaades	2732	4		204
Super built	Ready To	Rajaji Nag	4 BHK	Brway G	3300	4		600
Super built	Ready To	Marathah	3 BHK		1310	3	1	63.25
Plot Area	Ready To	Gandhi Ba	6 Bedroom		1020	6		370
Super built	18-Feb	Whitefield	3 BHK		1800	2	2	70
Plot Area	Ready To	Whitefield	4 Bedroom	Prrry M	2785	5	3	295
Super built	Ready To	7th Phase	2 BHK	Shncyes	1000	2	1	38
Built-up A	Ready To	Gottigere	2 BHK		1100	2	2	40
Plot Area	Ready To	Sarjapur	3 Bedroom	Skityer	2250	3	2	148
Super built	Ready To	Mysore Rc	2 BHK	PrntaEn	1175	2	2	73.5
Super built	Ready To	Bisuvanah	3 BHK	Prityel	1180	3	2	48
Super built	Ready To	Raja Rajes	3 BHK	GrrvaGr	1540	3	3	60
Super built	Ready To	Ramakrish	3 BHK	PeBayle	2770	4	2	290
Super built	Ready To	Manayata	2 BHK		1100	2	2	48
Built-up A	Ready To	Kengeri	1 BHK		600	1	1	15
Super built	19-Dec	Binny Pete	3 BHK	She 2rk	1755	3	1	122
Plot Area	Ready To	Thanisandi	4 Bedroom	Soitya	2800	5	2	380
Super built	Ready To	Bellandur	3 BHK		1767	3	1	103
Super built	18-Nov	Thanisand	1 RK	Bhe 2ko	510	1	0	25.25
Super built	18-May	Mangamm	3 BHK		1250	3	2	56
Super built	Ready To	Electronic	2 BHK	Itelaa	660	1	1	23.1
Built-up A	20-Dec	Whitefield	3 BHK		1610	3	2	81

Figure 4: Fields presented in the downloaded .csv file

The Machine Learning technique that I used for this project is Supervised Learning. It allows to collect data or produce a data output from the previous experience. For this project, I have already tagged all the labelled data sets. So, as I am using the supervised learning, I have predefined input as well as output value, based on the values, which helps to predict the price for the real estate.

Next, I installed “Anaconda” for the PC (Personal Computer). The following (Figure 5) shows the website which I downloaded the “Anaconda3-2020.02-Windows-x86_64.exe” file, which is used to install the software named as “Anaconda”.

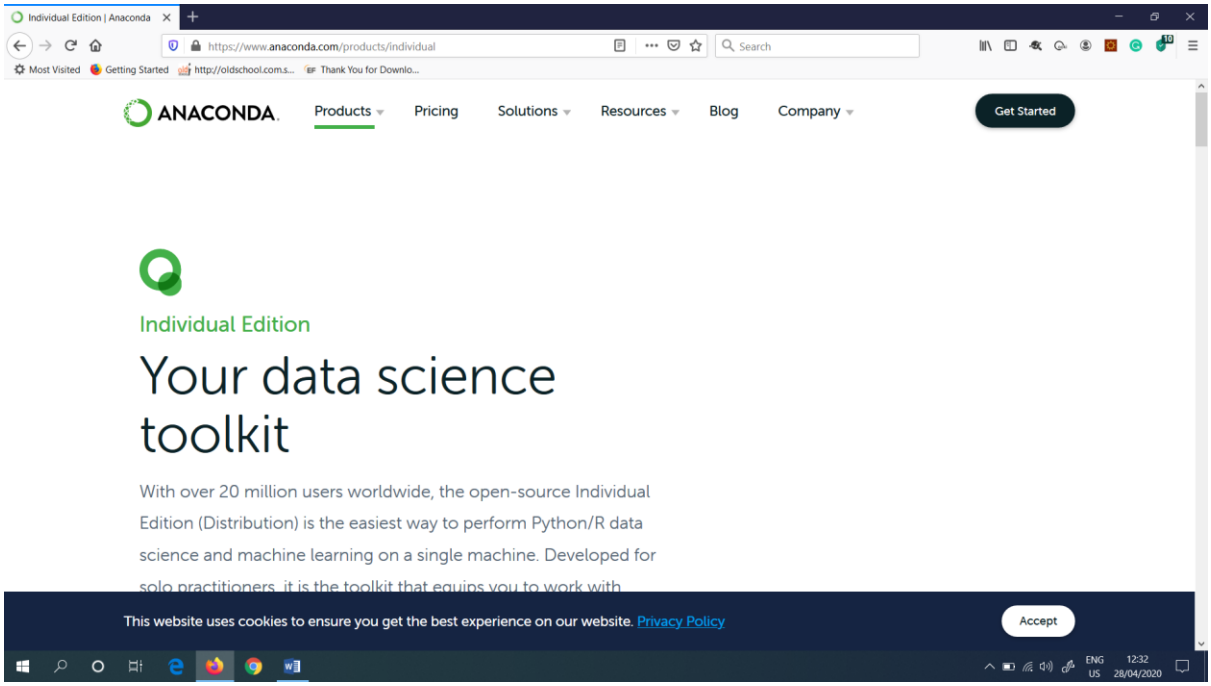


Figure 5: Website to download the Anaconda exe file

As my PC Operating System is Windows and as it is 64-bit PC, I selected the Windows and 64-bit option and downloaded the “Anaconda.exe file”. This is shown in (Figure 6).

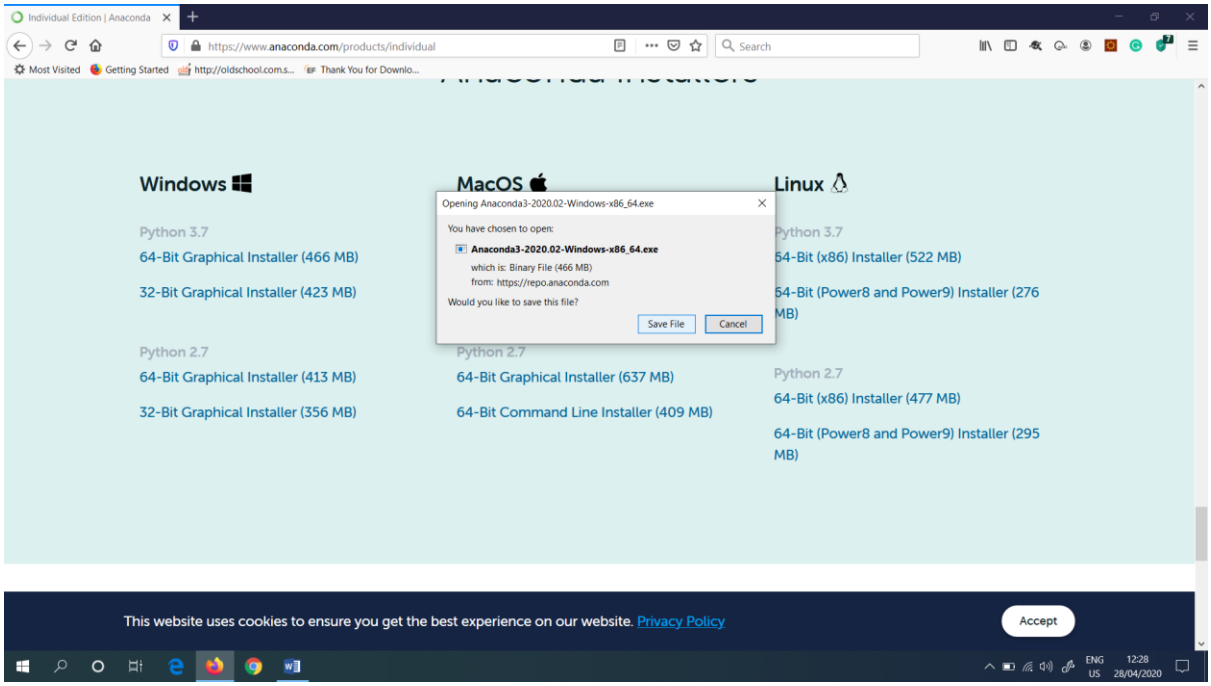


Figure 6: Downloaded Anaconda exe file

After I downloaded the “Anaconda.exe” file, it is shown under the Downloads section in the PC, which is shown in (Figure 7).

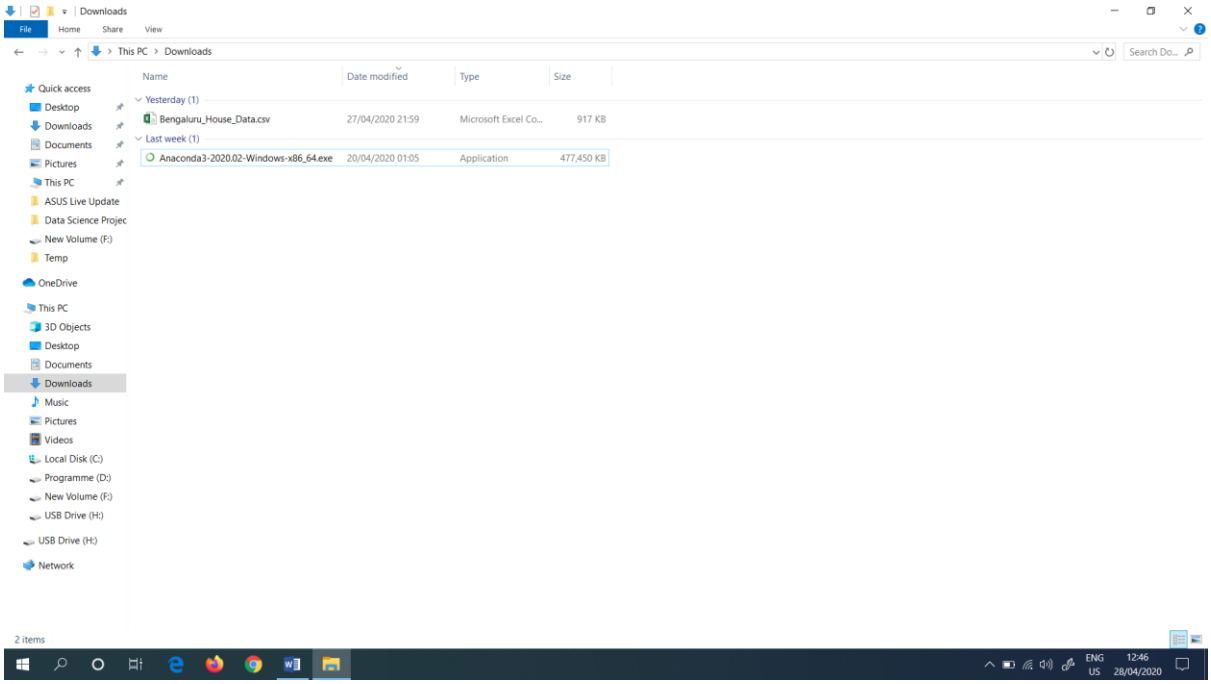


Figure 7: Downloaded Anaconda.exe file which is presented under the Downloads

Then I opened the “Anaconda Prompt” in the search bar (Figure 8).

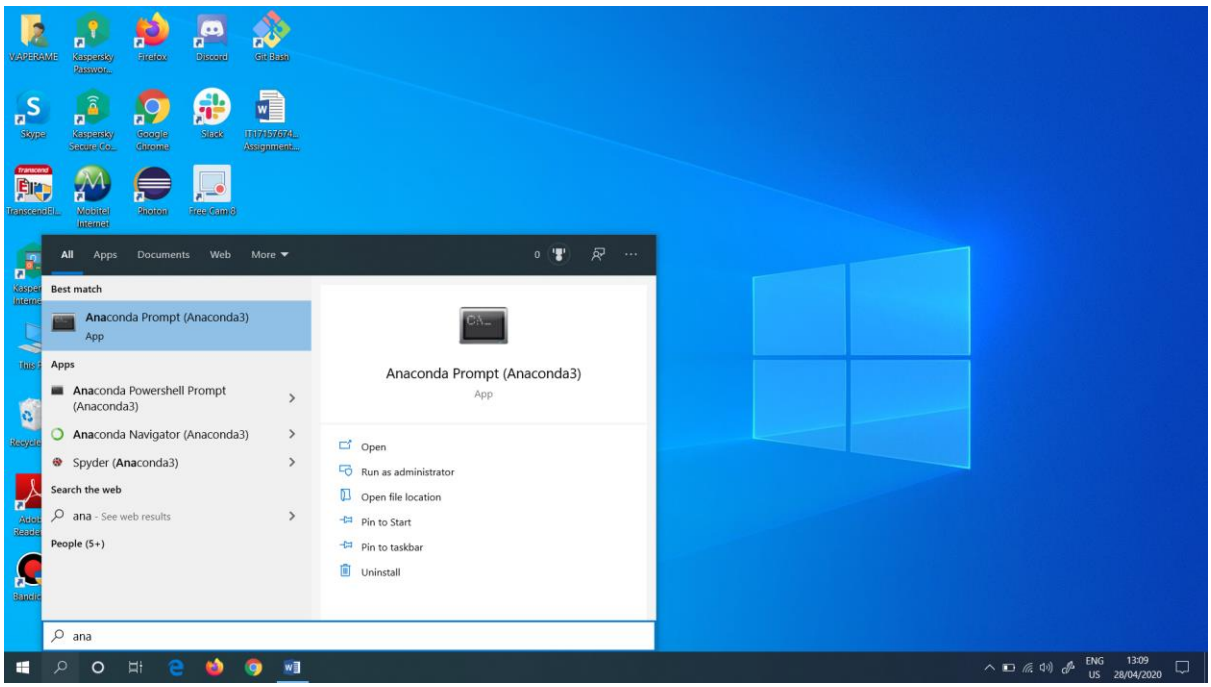


Figure 8: Opened the Anaconda Prompt

Next I checked the Python version, in the Anaconda prompt. So I typed as “python –version”, I have also seen the version of conda. So I typed as “conda info” to get the version of conda. These are shown in (Figure 9).

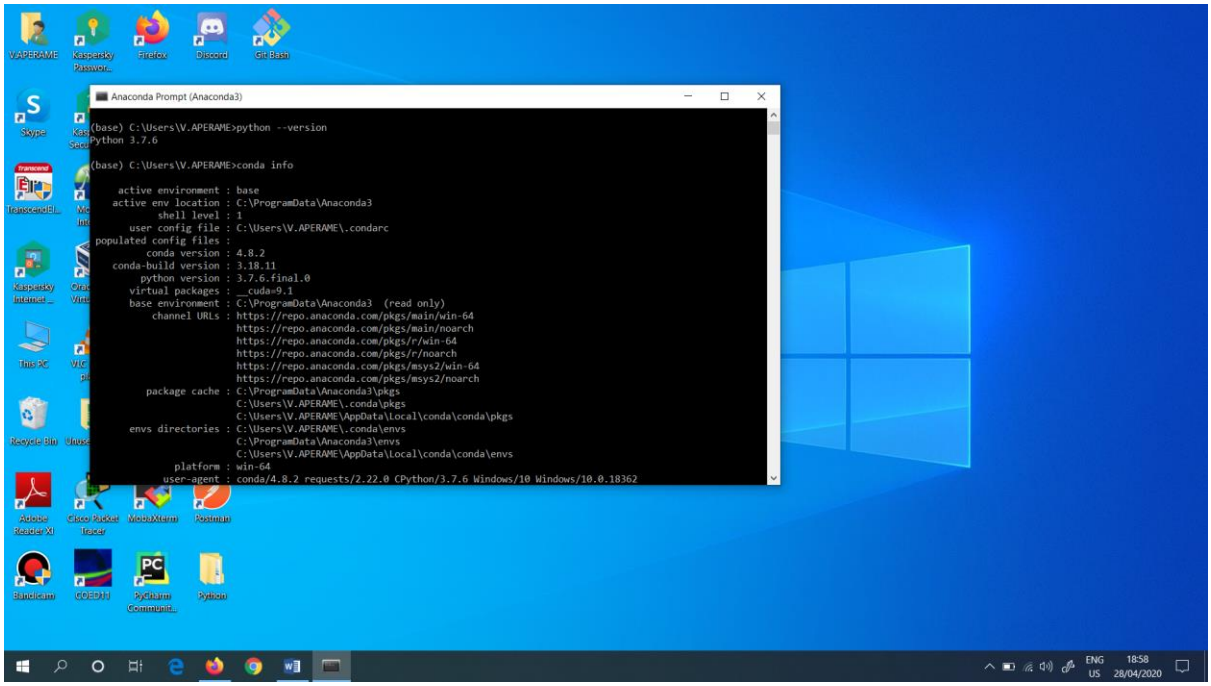


Figure 9: Checked the Python and conda version in Anaconda prompt

"Conda" is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies.

Then I typed as "Jupyter Notebook" in the Anaconda Prompt (Figure 10) and opened the Jupyter Notebook in the local host in any browser.

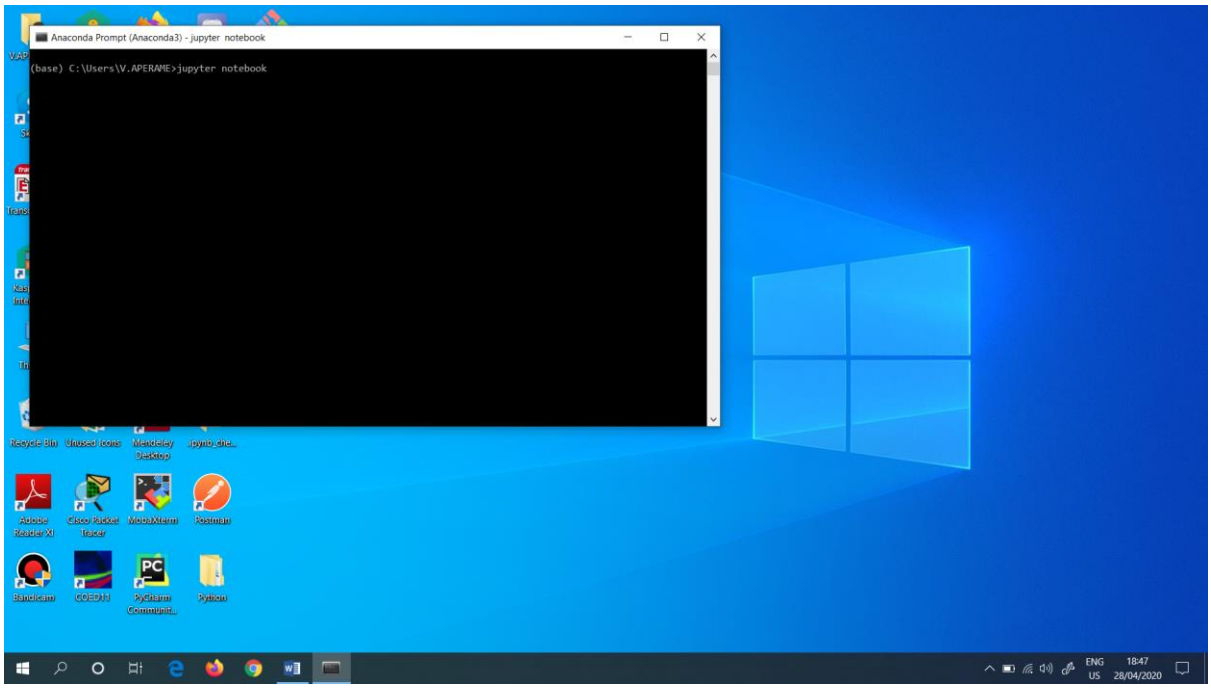


Figure 10: Opened the Jupyter Notebook

So, I opened the Jupyter Notebook in Google Chrome. This is shown in (Figure 11).

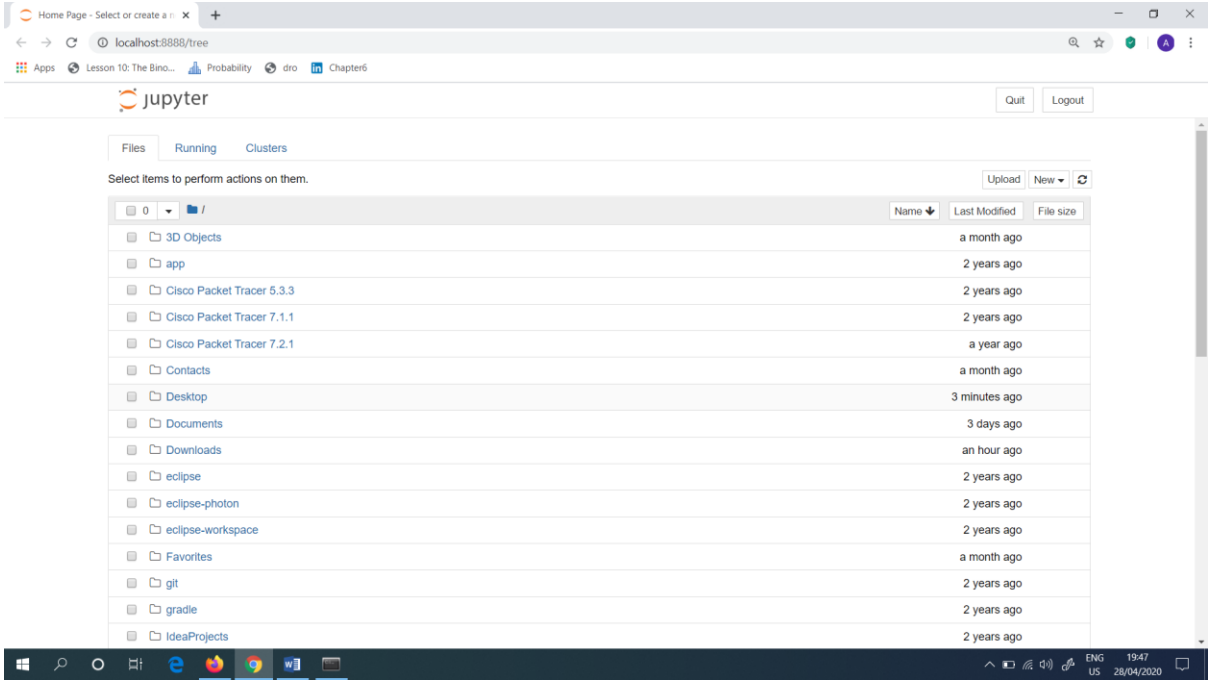


Figure 11: Opened the Jupyter Notebook in Google Chrome

Then I selected the “Desktop” folder, and selected the folder named as “MLCS_Real Estate Project”. This folder consists of all the related files which are needed for this project. To open the Jupyter Notebook file, I selected the “Real Estate Machine Learning Model.ipynb”. This is shown in (Figure 12).

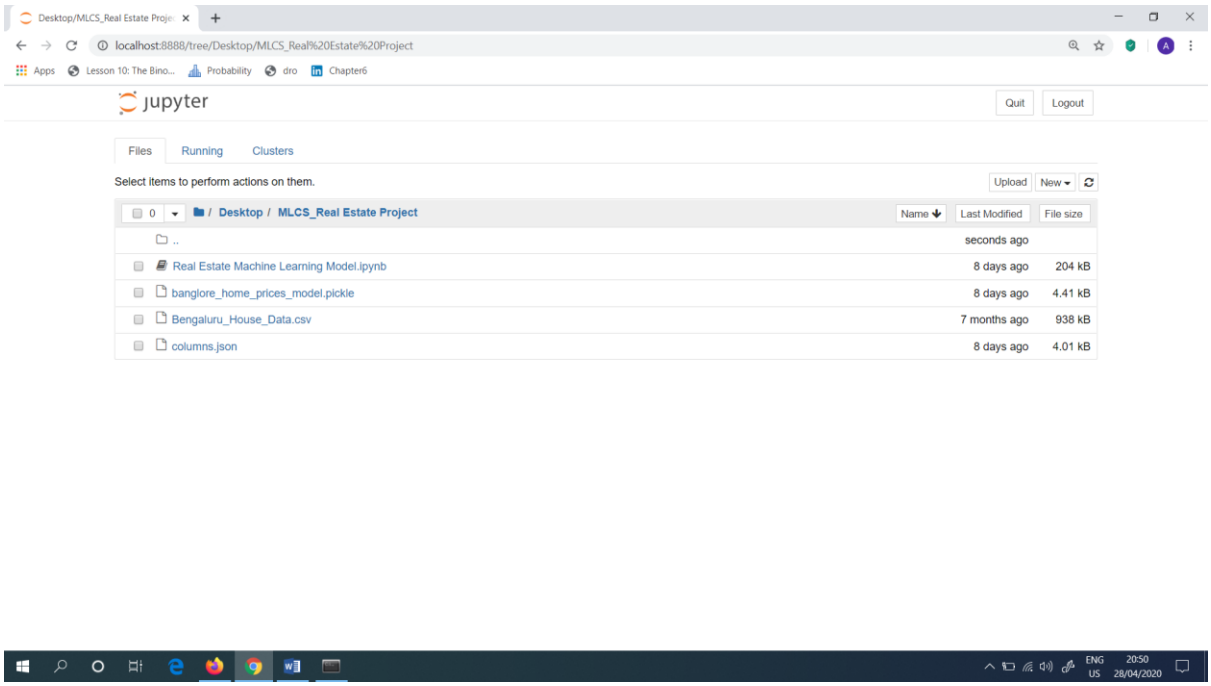


Figure 12: Opened the MLCS_Real Estate Project Folder

After I clicked the “Real Estate Machine Learning Model.ipynb” file, it prompts a python interface, Jupyter Notebook which is shown in (Figure 13).

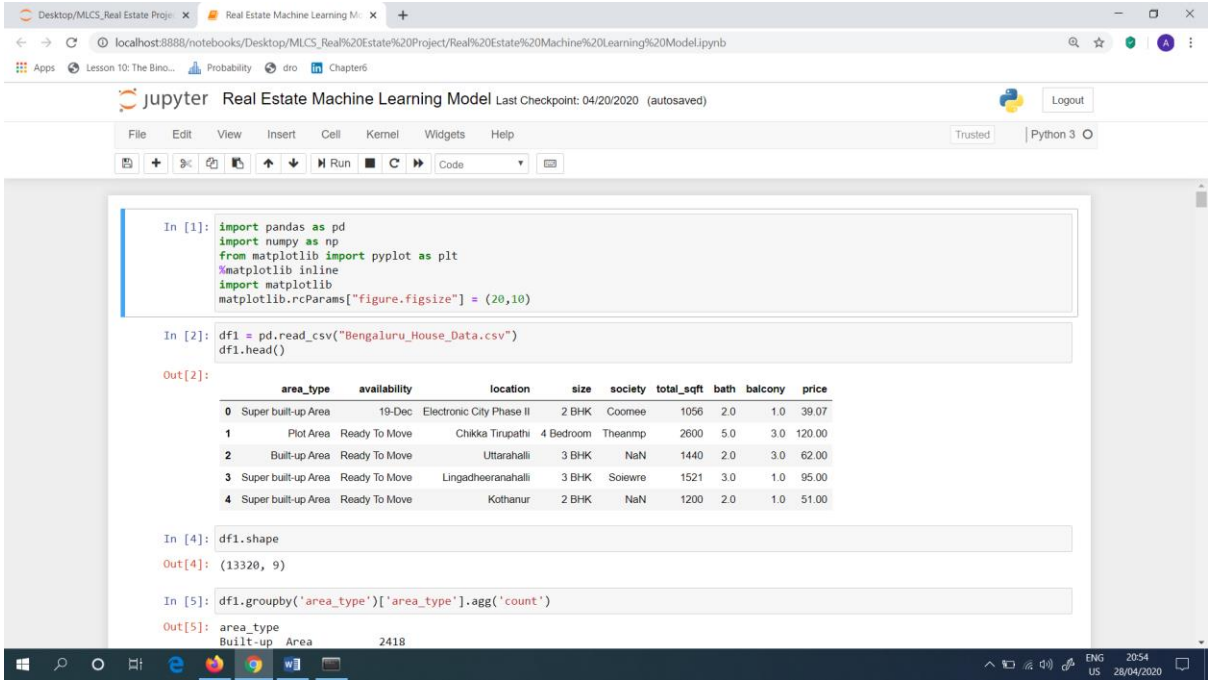


Figure 13: Interface for the Real Estate Machine Learning Model.ipynb

Then I imported some libraries which are needed in order to run this project. Like pandas, numpy, matplotlib etc. After I imported all the necessary libraries, it read the .csv files which I opened into the Pandas Data frame. Quickly all the data are loaded from the csv file.

In order to read the csv file, the command that I used is as follows.

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

df1.head()

In the above written command, which helps to load the csv file, the df1 denotes the Data Frame. This is shown in (Figure 14).

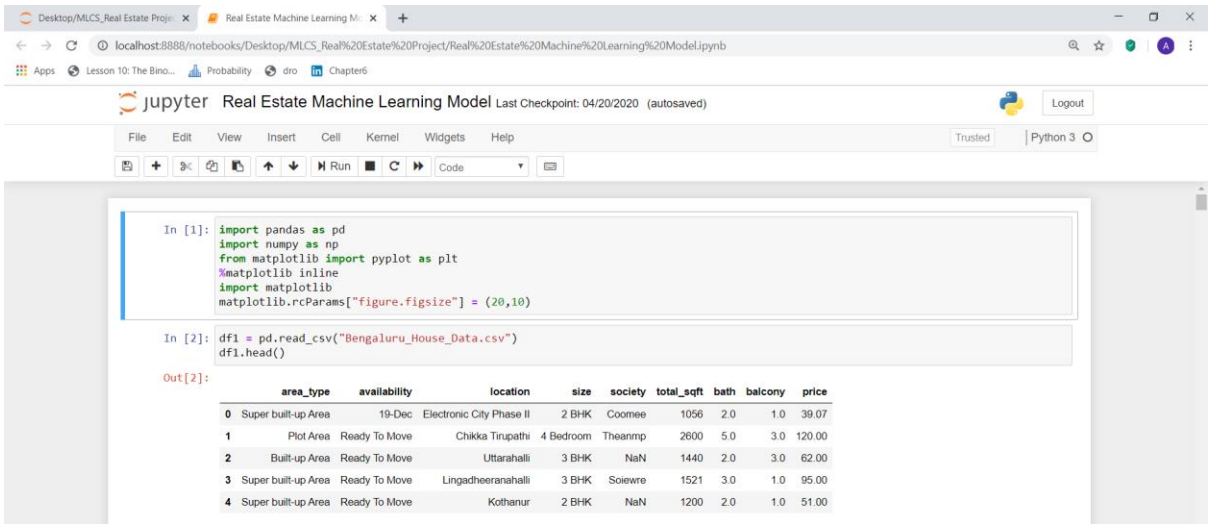


Figure 14: Display the data presented in the csv file

Next, in order to know how many rows and columns available in the loaded csv file, I typed the command as "df1.shape()". It prompts me an output like (13 320, 9). From this above

mentioned output, I came to know that there are 13 320 rows as well as 9 columns are available. This is shown in (Figure 15).

```
In [4]: df1.shape
Out[4]: (13320, 9)
```

Figure 15: Display the number of rows and columns available in the .csv file

Then, to identify the "area type" feature, I typed the command as shown in (Figure 16). In order to do this, I grouped the data set by the area_type and used the function called as aggregate for count (Figure 16).

```
In [5]: df1.groupby('area_type')['area_type'].agg('count')
Out[5]: area_type
        Built-up  Area          2418
        Carpet  Area              87
        Plot  Area              2025
        Super built-up  Area    8790
        Name: area_type, dtype: int64
```

Figure 16: Display the area type

With the help of the above mentioned output (Figure 16), I came to know that, 2418 samples are Built-up Area, 87 samples are Carpet Area, 2025 samples are Plot Area etc.

Next, in order to keep the Machine Learning model simple, I thought to drop certain columns from the data frame. So I assumed that when deciding the final price of the Real Estate, columns like area_type, availability, society etc are unwanted. So, I dropped all the above mentioned columns in order to keep my model simple. The (Figure 17) shows the present available columns available, after I dropped the above mentioned unwanted columns.

To drop the unwanted columns, the written input commands and the output produced are all can be seen in (Figure 17).

```
In [6]: df2 = df1.drop(['area_type','society','balcony','availability'],axis='columns')
        df2.head()
Out[6]:
```

	location	size	total_sqft	bath	price
0	Electronic City Phase II	2 BHK	1056	2.0	39.07
1	Chikka Tirupathi	4 Bedroom	2600	5.0	120.00
2	Uttarahalli	3 BHK	1440	2.0	62.00
3	Lingadheeranahalli	3 BHK	1521	3.0	95.00
4	Kothanur	2 BHK	1200	2.0	51.00

Figure 17: Final output after I dropped the unwanted columns

Then, the Data Cleaning process starts. To do this, isnull() function is used. So, I typed the command as "df2.isnull().sum()". This command shows that, the number of rows where particular column value is zero. For example, the bath has 73 rows, where the value for bath is not available. So, I planned to drop all the zero values as mentioned below. This is shown in (Figure 18).

```
In [9]: df2.isnull().sum()
Out[9]: location       1
        size          16
        total_sqft     0
        bath          73
        price          0
        dtype: int64
```

Figure 18: Data Cleaning Process

But, if I don't want to drop any rows, I can do as follows. For example, without bath the rows are 73. So, If I don't want to drop these 73 rows, I can take the median for this bath value, and fill the zero value with that median value.

So in this project there are 13 320 rows available, and for the zero values like bath, there are 73 rows as well as for size there are 16 rows. So, I planned to drop those rows, due to small in number for these rows. After dropping these rows, I created a new data frame called as "df3". In the newly created df3 data frame, I can able to see that there aren't any rows available. For dropping the rows "dropna()" function is used.

All these above mentioned procedures are shown in (Figure 19).

```
In [10]: df3 = df2.dropna()
         df3.isnull().sum()

Out[10]: location      0
         size          0
         total_sqft    0
         bath          0
         price         0
         dtype: int64
```

Figure 19: New Data frame df3 is created

After this, I again checked that currently how many rows and columns available. To do this, I typed the command as "df3.shape". Now, it prompts me that there are 13 246 rows and 5 columns available. This is shown in (Figure 20).

```
In [11]: df3.shape

Out[11]: (13246, 5)
```

Figure 20: Checked the current rows and columns available

So, in the data cleaning process there are about 74 rows got cleared during this above process. I.e.: 13 320 – 13 246 = 74.

Next, I am going to explore the feature known as "size". Because under the "size" column some values are 2BHK (Bedroom, Hall, Kitchen), 4 Bedroom etc. So, in the written command "df3['size']. unique()", "df3['size']" denotes the Pandas series, as well as the "unique()" function gives all the unique values. The (Figure 21) depicts this.

```
In [12]: df3['size'].unique()

Out[12]: 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)
```

Figure 21: All the unique values

In this above output (Figure 21), 4 Bedrooms and 4 BHK values are same. So, to resolve this kind of problems, I created a new column called as "BHK". The way I used to create a new column is shown in (Figure 22).

```
In [13]: df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0]))

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

Figure 22: To create a new column

In this above mentioned command, the created new column is named as ‘bhk’. This column is created based on the column named as “size” and in this “size” column I applied some function like “lambda x”. After this, I again checked the data frame. This is shown in Figure 23. Here, it shows a new column named as “bhk”.

```
In [14]: df3.head()
```

Out[14]:

	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

Figure 23: Checked the new data frame

Then, to see the unique number of bhk, I typed as shown in (Figure 24). In this I came to know that, there are homes with 43 bedrooms, hall and kitchen.

```
In [15]: df3['bhk'].unique()
Out[15]: array([ 2,  4,  3,  6,  1,  8,  7,  5, 11,  9, 27, 10, 19, 16, 43, 14, 12,
        13, 18], dtype=int64)
```

Figure 24: Checked the unique number of bhk

So, I’d like to see what kind of home which has 43 bedrooms, hall and kitchen. To do this, I wrote the command as shown in (Figure 25), where I asked to filter the command as bhk > 20. As a result, it showed me two output.

```
In [16]: df3[df3.bhk>20]
```

Out[16]:

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

Figure 25: Saw which real estate are above bhk>20

But, there can’t be 43 Bedrooms with 2400 total square feet. So, in order to remove this error, I have done the following mechanisms. Next, I checked all the unique total square feet available. In this, I noticed that “1133 – 1384” it shows a range of values (Figure 26).

```
In [17]: df3.total_sqft.unique()
Out[17]: array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
       dtype=object)
```

Figure 26: Range of values

So, in order to convert the range of values into a single number I thought that the average of that particular range of numbers will be the best option. So, at first most I need to identify whether the given range of value is float or not. So, for this I typed the command as shown in (Figure 27).

```
In [19]: def is_float(x):
             try:
                 float(x)
             except:
                 return False
             return True
```

```
In [20]: df3[~df3['total_sqft'].apply(is_float)].head()
```

Out[20]:

	location	size	total_sqft	bath	price	bhk
30	Yelahanka	4 BHK	2100 - 2850	4.0	186.000	4
122	Hebbal	4 BHK	3067 - 8156	4.0	477.000	4
137	8th Phase JP Nagar	2 BHK	1042 - 1105	2.0	54.005	2
165	Sarjapur	2 BHK	1145 - 1340	2.0	43.490	2
188	KR Puram	2 BHK	1015 - 1540	2.0	56.800	2

Figure 27: Identified whether the given range of value is float or not

Then, in order to get only the first 10 records, I wrote the command as shown in (Figure 28).

```
In [21]: df3[~df3['total_sqft'].apply(is_float)].head(10)
```

Out[21]:

	location	size	total_sqft	bath	price	bhk
30	Yelahanka	4 BHK	2100 - 2850	4.0	186.000	4
122	Hebbal	4 BHK	3067 - 8156	4.0	477.000	4
137	8th Phase JP Nagar	2 BHK	1042 - 1105	2.0	54.005	2
165	Sarjapur	2 BHK	1145 - 1340	2.0	43.490	2
188	KR Puram	2 BHK	1015 - 1540	2.0	56.800	2
410	Kengeri	1 BHK	34.46Sq. Meter	1.0	18.500	1
549	Hennur Road	2 BHK	1195 - 1440	2.0	63.770	2
648	Arekere	9 Bedroom	4125Perch	9.0	265.000	9
661	Yelahanka	2 BHK	1120 - 1145	2.0	48.130	2
672	Bettahalsoor	4 Bedroom	3090 - 5002	4.0	445.000	4

Figure 28: To get the first 10 records

In the above (Figure 28), some values for total_sqft are Meter, some are Perch. So, the datasets are not unified, they are unstructured, contains outliers etc. To handle these above mentioned errors in a particular range of numbers, which are presented in total_sqft, I tried to take the average of it. The values which are in Square Meter and Perch, I just ignored those rows. So, I am going to take only the range of values.

To do this, I am going to write a Python function, which takes the range of number as an input, and it returns the average for that range of number as an output value. So, If I wrote the function as mentioned in (Figure 29), it returns the value as a float. This is shown in (Figure 29).

```
In [22]: def convert_sqft_to_num(x):
             tokens = x.split('-')
             if len(tokens) == 2:
                 return (float(tokens[0])+float(tokens[1]))/2
             try:
                 return float(x)
             except:
                 return None
```

```
In [23]: convert_sqft_to_num('2166')
```

```
Out[23]: 2166.0
```

Figure 29: Function returns the float value for a number

Then, If I wrote the same function for the range of value, it takes the average of the two numbers and returns the float value (Figure 30).

```
In [24]: convert_sqft_to_num('2100 - 2850')
Out[24]: 2475.0
```

Figure 30: Function returns the float value for the range of value

Next, If I wrote for the Sq. Meter, it didn't return anything. This is shown in (Figure 31).

```
In [25]: convert_sqft_to_num('34.465q. Meter')
```

Figure 31: Function returns nothing

Now, I applied the function to the "total_sqft" column, and I created a new data frame. In the data processing pipeline at every stage I am creating a new data frame. In this process, I am going to create a new data frame known as "df4", which is the copy of df3. This can be seen in (Figure 32).

```
In [26]: df4 = df3.copy()
         df4['total_sqft'] = df4['total_sqft'].apply(convert_sqft_to_num)
         df4.head(3)
Out[26]:
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3

Figure 32: Created a new data frame (df4)

To, access the index 30, I wrote the command as shown in (Figure 33). In this command the "loc" defines the location. And if I get the average for the index 30, it shows the value as 2475.0.

```
In [27]: df4.loc[30]
Out[27]: location      Yelahanka
         size              4 BHK
         total_sqft         2475
         bath                  4
         price               186
         bhk                   4
         Name: 30, dtype: object

In [28]: (2100+2850)/2
Out[28]: 2475.0
```

Figure 33: Accessing the index 30, Average for index 30

Up to now, I have cleaned up the "total_sqft" column. And now the data frame looks as shown in the (Figure 34).

```
In [29]: df4.head(3)
Out[29]:
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3

Figure 34: The Data Frame

Then, now I am going to create a new data frame (df5), which is a copy of the data frame, (df4) to produce a new column named as "price_per_sqft".

In the real estate market, as the price per square feet feature is really important, I planned to create a new column which was named as "price_per_sqft", which is a feature in Featured

Engineering, which helps for outlier detection and removal. The following (Figure 35) shows the above procedure.

```
In [30]: df5 = df4.copy()
         df5['price_per_sqft'] = df5['price']*100000/df5['total_sqft']
         df5.head()
```

Out[30]:

	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

Figure 35: New column "price_per_sqft" is created

In order to check how many number of unique locations are available, I typed the command as shown in (Figure 36).

```
In [31]: df5.location.unique()

Out[31]: array(['Electronic City Phase II', 'Chikka Tirupathi', 'Uttarahalli', ...,
                '12th cross srinivas nagar banshankari 3rd stage',
                'Havanur extension', 'Abshot Layout'], dtype=object)
```

Figure 35: Number of unique locations available

Next, to get the count of all the locations available, I typed the command as shown in (Figure 36).

```
In [32]: len(df5.location.unique())

Out[32]: 1304
```

Figure 36: Total count for the available locations

According to the above output, as 1304 is a very large number, which causes dimensionality problems. There are some techniques available to reduce these kinds of problems. Among that, one of the technique is to produce a new category known as "Other" category. Which means, when there are 1304 locations, can able to find there are many locations available with one/ two data points.

So, as the next step, I wrote the command, in order to find out how many data points are available for the particular locations. The command for this procedure is shown in (Figure 37).

```
In [33]: df5.location = df5.location.apply(lambda x: x.strip())

         location_stats = df5.groupby('location')['location'].agg('count')
         location_stats

Out[33]: location
         1 Annasandrapalya                                  1
         1 Giri Nagar                                       1
         1 Immadihalli                                      1
         1 Ramamurthy Nagar                                 1
         12th cross srinivas nagar banshankari 3rd stage    1
                                                           ..
         t.c palya                                          1
         tc.palya                                           4
         vinayakanagar                                      1
         white field,kadugodi                               1
         whitefiled                                         1
         Name: location, Length: 1293, dtype: int64
```

Figure 37: Data Points available for the particular locations

Then, I would like to sort the locations by number of data points. To do that, “sort_values” function is used, which gives the output in descending order (Figure 38).

```
In [34]: df5.location = df5.location.apply(lambda x: x.strip())

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

Out[34]: location
         Whitefield           535
         Sarjapur  Road       392
         Electronic City      304
         Kanakpura Road       266
         Thanisandra          236
                             ...
         LIC Colony             1
         Kuvempu Layout         1
         Kumbhena Agrahara      1
         Kudlu Village,         1
         1 Annasandrapalya      1
         Name: location, Length: 1293, dtype: int64
```

Figure 38: Location in Descending Order

In the above output, I came to know that “Whitefield” location has maximum number of data points in the .csv file, which I used for this project.

And also, I came to know, many locations have only 1 data point. So, from this I came to a conclusion that, any location which has less than 10 data points will come under “Other Locations”. This is shown in (Figure 39).

```
In [35]: len(location_stats[location_stats<=10])

Out[35]: 1052
```

Figure 39: Location which has less than 10 data points

From this there are 1052 locations, out of 1293 which comes under the category “Other Locations”.

Then, I checked what are the locations, which has less than 10 data points. To do this, I typed the command as shown in (Figure 40).

```
In [36]: location_stats_less_than_10 = location_stats[location_stats<=10]
         location_stats_less_than_10

Out[36]: location
         BTM 1st Stage          10
         Basapura               10
         Sector 1 HSR Layout    10
         Naganathapura          10
         Kalkere                10
                                ..
         LIC Colony              1
         Kuvempu Layout          1
         Kumbhena Agrahara       1
         Kudlu Village,          1
         1 Annasandrapalya       1
         Name: location, Length: 1052, dtype: int64
```

Figure 40: Total locations which are less than 10 data points

Then, I print to see the total number of unique locations available, and I saw that there are about 1293 (Figure 41).

```
In [37]: len(df5.location.unique())

Out[37]: 1293
```

Figure 41: Total number of unique locations available

Next, I applied the transformation into the data frame. Then I also applied the “Lambda” function. After the transformation is finished all the locations get converted to “Other”. Then If I print the unique location value, I found that there are only 242 locations.

To get sorted only the unique number of locations from the above, I wrote the command as shown in (Figure 42), which gives the output for unique locations as 242.

```
In [38]: df5.location = df5.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)
         len(df5.location.unique())

Out[38]: 242
```

Figure 42: Number of unique locations available

To check, whether the particular locations have been converted as “Other”, I typed the command as shown in (Figure 43) and saw the output from the data frame.

```
In [39]: df5.head(10)
```

Out[39]:

	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	other	6 Bedroom	1020.0	6.0	370.00	6	36274.509804

Figure 43: The locations are get converted to “Other”

Now I planned to do outlier detection and removal. The definition for the “Outliers” are, the data points which have data errors. Sometimes they aren’t data errors, but they represent the extreme variations in a particular data set. So, it is a must to remove out from the data sets, because they produce some unwanted errors in a defined data set.

There are many techniques available to remove the outliers such as Standard Deviation/ Any simple knowledge in any domain.

Basically, in the real estate, any data rows where the square feet per bedroom is less than some particular threshold. But, the typical square feet per bedroom is, around 300.

As an example, if there is 600 square feet home, and the house consists of totally 6 bedrooms. Then if I divide it (600/6 = 100) it produces 100 (Figure 44). This is very unusual because, 300 square feet per bedroom, is the typical threshold.

```
In [40]: 600/6

Out[40]: 100.0
```

Figure 44: Example

So, I decided to remove the data sets where the threshold value (As mentioned above) is not met. The following (Figure 45) represents the command I wrote to remove the data sets.

```
In [41]: df5[df5.total_sqft/df5.bhk<300].head()
```

Out[41]:

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

Figure 45: Removed Data sets

According to (Figure 45), 1020 total_sqft real estate with 6 BHK is unusual. As same as, 600 total_sqft real estate with 8 BHK is also unusual. These are anomalies/ data errors/ outliers which are found in the data frame. So, I planned to remove the data errors.

To do this, first I saw how many rows are there in the data frame. It shows the result as 13 246 rows in (Figure 46).

```
In [42]: df5.shape
Out[42]: (13246, 7)
```

Figure 46: Rows available in the data frame

Then I removed the outliers, by created a new data frame (df6), and I wrote the command as shown in (Figure 47).

```
In [43]: df6 = df5[~(df5.total_sqft/df5.bhk<300)]
         df6.shape
Out[43]: (12502, 7)
```

Figure 47: Removed the outliers

Then I removed some rows. Because, I compared and saw that earlier there were 13 246 rows and now there are 12 502 rows.

Next, I checked the price_per_sqft. To check this, I typed the command as shown in (Figure 48). From this criteria, I need to check whether the price_per_sqft is extremely high/ low.

```
In [44]: df6.price_per_sqft.describe()
Out[44]: count     12456.000000
         mean       6308.502826
         std        4168.127339
         min         267.829813
         25%        4210.526316
         50%        5294.117647
         75%        6916.666667
         max      176470.588235
         Name: price_per_sqft, dtype: float64
```

Figure 48: Checked the price_per_sqft

From the above mentioned output, to purchase a real estate property in Bengaluru, 267.829 Indian Lakh Rupees per square feet, is very unlikely. Because the price is very low. Similarly, in Bengaluru 176 470.588 Indian Lakh Rupees per square feet, is extremely high. So, I planned to remove all these outliers, as I am going to build a generic model.

In order to remove this extreme cases, I planned to write a function that can remove these extreme cases, based on standard deviation. So, I wrote a function which can remove price_per_sqft outliers, per locations. Because some locations will have high price, as well as some locations will have low price. So, I need to find mean and standard deviation per

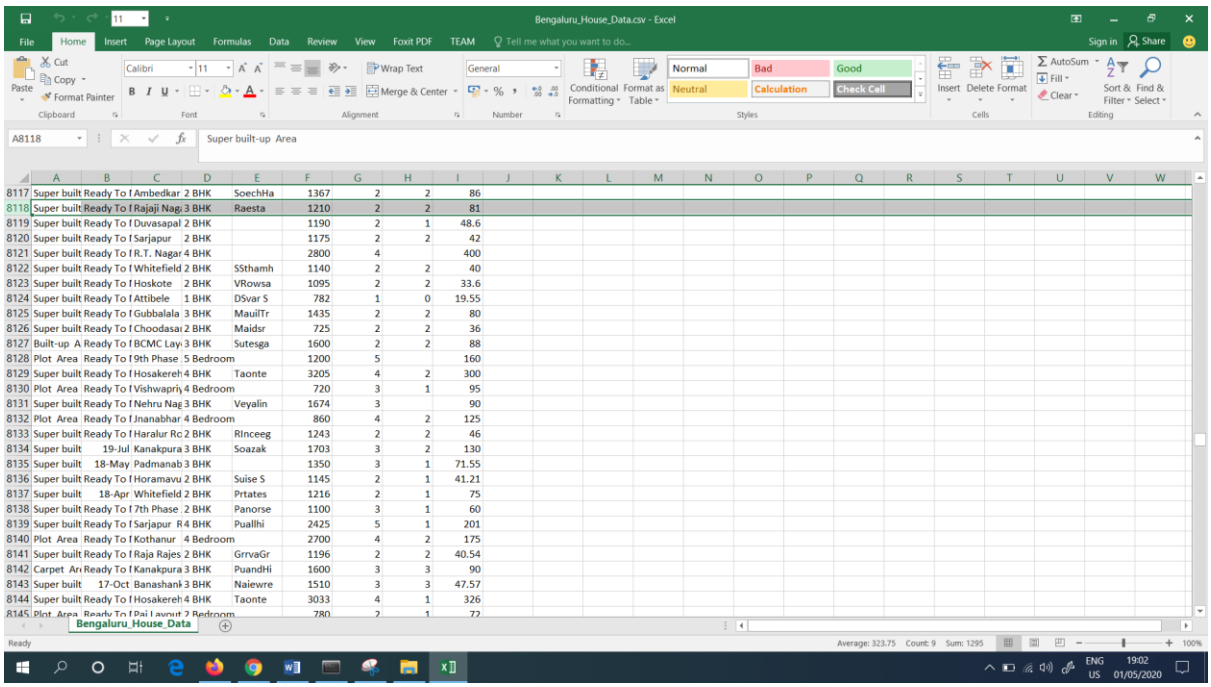
location and filter the data points which are beyond one standard deviation. The function which I wrote looks as shown in (Figure 49).

```
In [46]: def remove_pps_outliers(df):
             df_out = pd.DataFrame()
             for key, subdf in df.groupby('location'):
                 m = np.mean(subdf.price_per_sqft)
                 st = np.std(subdf.price_per_sqft)
                 reduced_df = subdf[(subdf.price_per_sqft>(m-st)) & (subdf.price_per_sqft<=(m+st))]
                 df_out = pd.concat([df_out,reduced_df],ignore_index=True)
             return df_out

         df7 = remove_pps_outliers(df6)
         df7.shape

Out[46]: (10241, 7)
```

Figure 49: Data frame (df7)

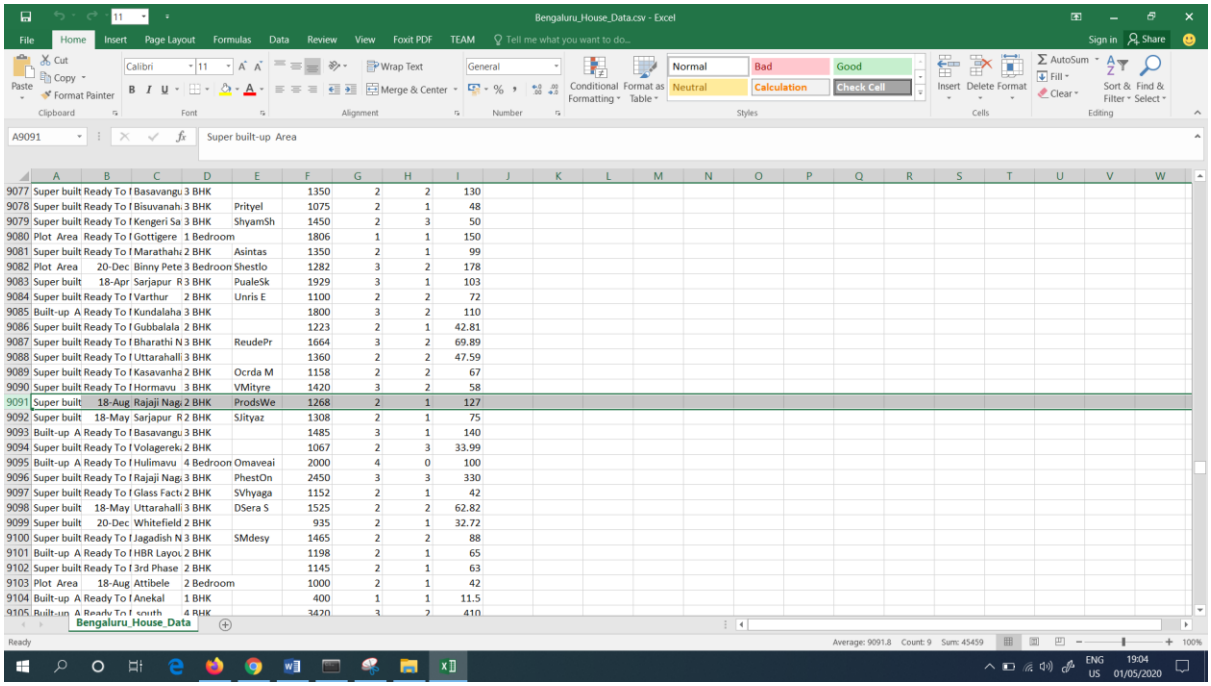
In the (Figure 49), I took the data frame as input, and I grouped them by location. And, per location I got the sub data frame for which I calculated m (Mean) and st (Standard Deviation). Then I filtered all the data points which are beyond the standard deviation. So, I called this function in df7. This df7 function, removed all the price_per_sqft outliers from df6. And I got a new data frame (df7), which has 10 241 data points. So, nearly compared to the previous output, I removed at about 2000 outliers from the data sets. (12 502 – 10 241 = 2 261)

Next, I need to check whether the property price for 3 bedrooms' real estate are more than the property price for 2 bedrooms' real estates or not, with the same square feet area. As an example which is shown in (Figure 50 and Figure 51) the property price for 3 bedrooms' price is 81 Indian Lakh Rupees, as well as 2 Bedroom price is 127 Indian Lakh Rupees. But, both have same square feet area like 1200 square feet.



8117	Super built	Ready To I	Ambedkar	2 BHK	SoechHa	1367	2	2	86
8118	Super built	Ready To I	Rajaji Nag	3 BHK	Raesta	1210	2	2	81
8119	Super built	Ready To I	Duvasapal	2 BHK		1190	2	1	48.6
8120	Super built	Ready To I	Sarjapur	2 BHK		1175	2	2	42
8121	Super built	Ready To I	R.T. Nagar	4 BHK		2800	4		400
8122	Super built	Ready To I	Whitefield	2 BHK	SSthamh	1140	2	2	40
8123	Super built	Ready To I	Hoskote	2 BHK	VRowsa	1095	2	2	33.6
8124	Super built	Ready To I	Attibele	1 BHK	DSvar S	782	1	0	19.55
8125	Super built	Ready To I	Gubbalala	3 BHK	MauilTr	1435	2	2	80
8126	Super built	Ready To I	Choodasar	2 BHK	Maidsr	725	2	2	36
8127	Built-up A	Ready To I	BCMC Lay	3 BHK	Sutesga	1600	2	2	88
8128	Plot Area	Ready To I	9th Phase	5 Bedroom		1200	5		160
8129	Super built	Ready To I	Hosakereh	4 BHK	Taonte	3205	4	2	300
8130	Plot Area	Ready To I	Vishwapriy	4 Bedroom		720	3	1	95
8131	Super built	Ready To I	Nehru Nag	3 BHK	Veyalin	1674	3		90
8132	Plot Area	Ready To I	Jnanabhar	4 Bedroom		860	4	2	125
8133	Super built	Ready To I	Haralur Ro	2 BHK	RInceeg	1243	2	2	46
8134	Super built	19-Jul	Kanakpura	3 BHK	Soazak	1703	3	2	130
8135	Super built	18-May	Padmanab	3 BHK		1350	3	1	71.55
8136	Super built	Ready To I	Horamavu	2 BHK	Suise S	1145	2	1	41.21
8137	Super built	18-Apr	Whitefield	2 BHK	Prtates	1216	2	1	75
8138	Super built	Ready To I	7th Phase	2 BHK	Panorse	1100	3	1	60
8139	Super built	Ready To I	Sarjapur R	4 BHK	Puallhi	2425	5	1	201
8140	Plot Area	Ready To I	Kothanur	4 Bedroom		2700	4	2	175
8141	Super built	Ready To I	Raja Rajes	2 BHK	GrrvaGr	1196	2	2	40.54
8142	Carpet Ar	Ready To I	Kanakpura	3 BHK	PuandHi	1600	3	3	90
8143	Super built	17-Oct	Banashank	3 BHK	Naiewre	1510	3	3	47.57
8144	Super built	Ready To I	Hosakereh	4 BHK	Taonte	3033	4	1	326
8145	Plot Area	Ready To I	Pai Layout	2 Bedroom		780	2	1	72

Figure 50: Property with 3 Bedrooms



9077	Super built	Ready To I	Basavangu	3 BHK		1350	2	2	130
9078	Super built	Ready To I	Bisuvanah	3 BHK	Prityel	1075	2	1	48
9079	Super built	Ready To I	Kengeri Sa	3 BHK	ShyamSh	1450	2	3	50
9080	Plot Area	Ready To I	Gottigere	1 Bedroom		1806	1	1	150
9081	Super built	Ready To I	Marathahi	2 BHK	Asintas	1350	2	1	99
9082	Plot Area	20-Dec	Binny Pete	3 Bedroom	Shestlo	1282	3	2	178
9083	Super built	18-Apr	Sarjapur R	3 BHK	PualeSk	1929	3	1	103
9084	Super built	Ready To I	Varthur	2 BHK	Unris E	1100	2	2	72
9085	Built-up A	Ready To I	Kundalaha	3 BHK		1800	3	2	110
9086	Super built	Ready To I	Gubbalala	2 BHK		1223	2	1	42.81
9087	Super built	Ready To I	Bharathi N	3 BHK	ReudePr	1664	3	2	69.89
9088	Super built	Ready To I	Uttarahalli	3 BHK		1360	2	2	47.59
9089	Super built	Ready To I	Kasavanha	2 BHK	Ocrda M	1158	2	2	67
9090	Super built	Ready To I	Hormavu	3 BHK	VMityre	1420	3	2	58
9091	Super built	18-Aug	Rajaji Nag	2 BHK	ProdsWe	1268	2	1	127
9092	Super built	18-May	Sarjapur R	2 BHK	SJityaz	1308	2	1	75
9093	Built-up A	Ready To I	Basavangu	3 BHK		1485	3	1	140
9094	Super built	Ready To I	Volagereki	2 BHK		1067	2	3	33.99
9095	Built-up A	Ready To I	Hulimavu	4 Bedroom	Omaveai	2000	4	0	100
9096	Super built	Ready To I	Rajaji Nag	3 BHK	PhestOn	2450	3	3	330
9097	Super built	Ready To I	Glass Facto	2 BHK	SVhyaga	1152	2	1	42
9098	Super built	18-May	Uttarahalli	3 BHK	DSera S	1525	2	2	62.82
9099	Super built	20-Dec	Whitefield	2 BHK		935	2	1	32.72
9100	Super built	Ready To I	Jagadish N	3 BHK	SMdesy	1465	2	2	88
9101	Built-up A	Ready To I	HBR Layou	2 BHK		1198	2	1	65
9102	Super built	Ready To I	3rd Phase	2 BHK		1145	2	1	63
9103	Plot Area	18-Aug	Attibele	2 Bedroom		1000	2	1	42
9104	Built-up A	Ready To I	Anekal	1 BHK		400	1	1	11.5
9105	Built-up A	Ready To I	south	4 BHK		3420	3	2	410

Figure 51: Property with 2 Bedrooms

Although the property with 2 bedrooms cost high price due to many reasons like location, any special reasons etc. So, I planned to do a visualization with the help of the scatter plot in order to identify these types of special cases. The (Figure 52) denotes the function which I wrote in order to identify these special cases with the help of the scatter plot.

```
In [47]: def plot_scatter_chart(df,location):
             bhk2 = df[(df.location==location) & (df.bhk==2)]
             bhk3 = df[(df.location==location) & (df.bhk==3)]
             matplotlib.rcParams['figure.figsize'] = (15,10)
             plt.scatter(bhk2.total_sqft,bhk2.price,color='blue',label='2 BHK', s=50)
             plt.scatter(bhk3.total_sqft,bhk3.price,marker='+', color='green',label='3 BHK', s=50)
             plt.xlabel("Total Square Feet Area")
             plt.ylabel("Price")
             plt.title(location)
             plt.legend()

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

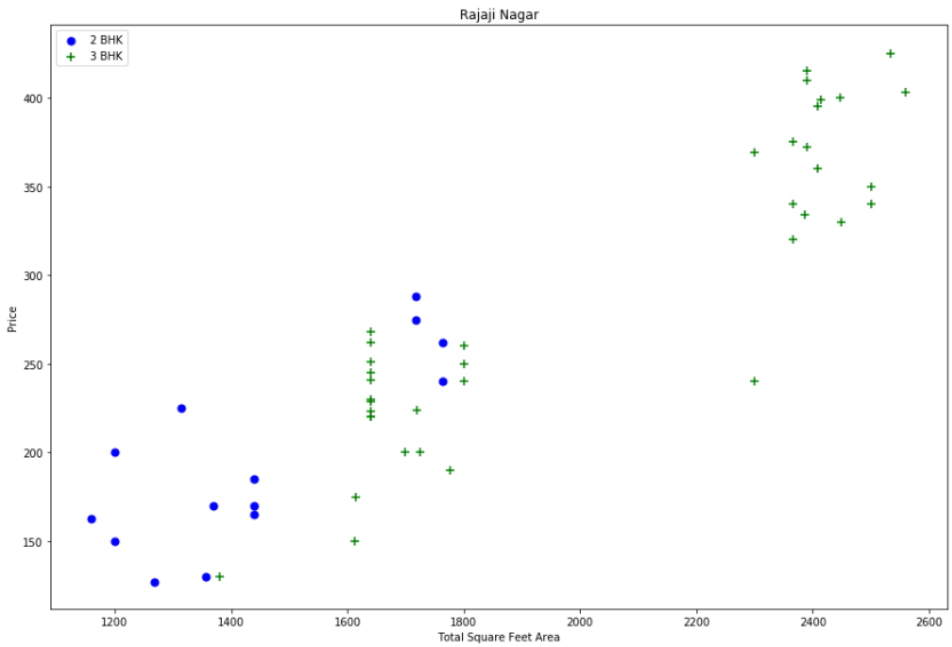


Figure 52: Scatter Plot for the location – Rajaji Nagar

The command which I wrote in (Figure 52) depicts that, it is used to draw a scatter plot, in which It was plot for 2 Bedrooms and for 3 Bedrooms (BHK). And also this function took the data frame (df7) and location (Rajaji Nagar) as the input.

In the scatter plot (Figure 52) the blue points are 2 bedrooms' real estate and green points depicts 3 BHK (Bedroom, Hall, Kitchen) properties. X Axis is depicted as Total Square Feet Area, and Y Axis is depicted as Price. In this scatter plot I found that around 1700 vertical square feet area, 2 Bedrooms real estate prices are higher than 3 Bedrooms. So, I planned to remove some of the outliers.

Then, again I ran the same function to another different location named as Hebbal. This is shown in (Figure 53).

```
In [48]: def plot_scatter_chart(df,location):
             bhk2 = df[(df.location==location) & (df.bhk==2)]
             bhk3 = df[(df.location==location) & (df.bhk==3)]
             matplotlib.rcParams['figure.figsize'] = (15,10)
             plt.scatter(bhk2.total_sqft,bhk2.price,color='blue',label='2 BHK', s=50)
             plt.scatter(bhk3.total_sqft,bhk3.price,marker='+', color='green',label='3 BHK', s=50)
             plt.xlabel("Total Square Feet Area")
             plt.ylabel("Price")
             plt.title(location)
             plt.legend()

         plot_scatter_chart(df7,"Hebbal")
```

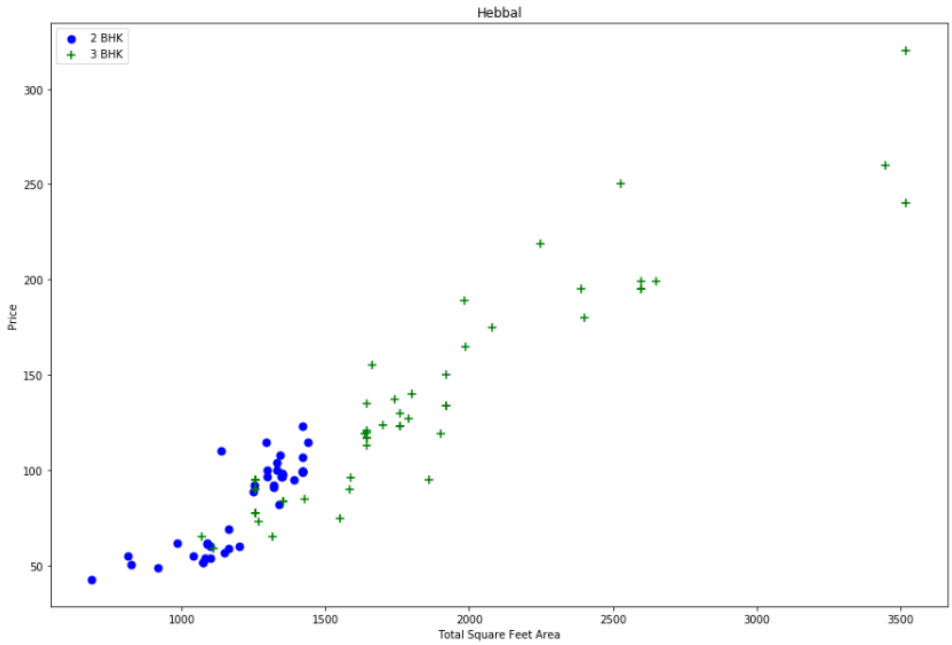


Figure 53: Scatter Plot for the location - Hebbal

There also I found that 2 Bedrooms prices are higher than 3 Bedrooms prices (Figure 53).

Next, I wrote the function which need to work as, which need to show more price for 3 Bedrooms compared to 2 Bedrooms property. So, I wrote a function in order to comply the above criteria (Figure 54).

In (Figure 54) function, first I went through, by grouping the location, then I went through every location data frame, and every location data frame, I again created a new data frame which are based on BHK. And for BHK data frame, I computed the mean, standard deviation and count. Once the "for loop" is over, I again ran the for loop again, and I tried to exclude

the data points where the price_per_sqft < mean. Next, I created a new data frame (df8), where it removes many outliers and showed me only 7329 rows.

```
In [49]: def remove_bhk_outliers(df):
             exclude_indices = np.array([])
             for location, location_df in df.groupby('location'):
                 bhk_stats = {}
                 for bhk, bhk_df in location_df.groupby('bhk'):
                     bhk_stats[bhk] = {
                         'mean': np.mean(bhk_df.price_per_sqft),
                         'std': np.std(bhk_df.price_per_sqft),
                         'count': bhk_df.shape[0]
                     }
                 for bhk, bhk_df in location_df.groupby('bhk'):
                     stats = bhk_stats.get(bhk-1)
                     if stats and stats['count']>5:
                         exclude_indices = np.append(exclude_indices, bhk_df[bhk_df.price_per_sqft<(stats['mean'])].index.values)
             return df.drop(exclude_indices,axis='index')

         df8 = remove_bhk_outliers(df7)
         df8.shape

Out[49]: (7329, 7)
```

Figure 54: Rows presented after the removal of outliers

Then, I again plot the scatter plot, to saw what kind of improvement had been done in the previous scatter plot. So I typed the command as shown in (Figure 55) and it produce a new scatter plot.

```
In [50]: plot_scatter_chart(df8,"Hebbal")
```

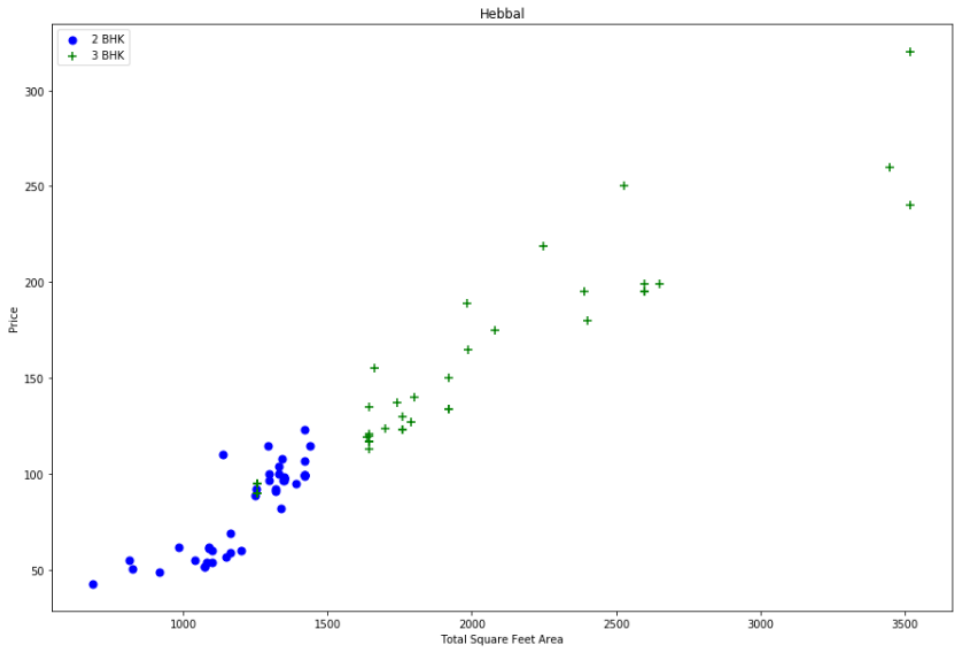


Figure 55: New Scatter Plot

In the above scatter plot (Figure 55) I found that the green coloured data points got disappeared. And also 3 Bedrooms real estate have higher value than 2 Bedrooms real estate. Even though the green coloured data points are removed, it is hard to remove those kind of abnormalities. So, it is still presented in the data sets.

Once the outliers are removed, then I plotted a histogram to see how many properties are available per square feet area. So, I wrote the command as shown in (Figure 56), and I found that the X Axis denotes the Price per square feet, and Y Axis denotes the Count. In the Histogram, I found that 0 – 10 000 Indian Lakh Rupees per Square feet range, I found that

there are majority of data points. The Histogram looks like a normal distribution, as well as it looks like a bell curve.

```
In [51]: import matplotlib
         matplotlib.rcParams["figure.figsize"] = (20,10)
         plt.hist(df8.price_per_sqft,rwidth=0.8)
         plt.xlabel("Price Per Square Feet")
         plt.ylabel("Count")

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

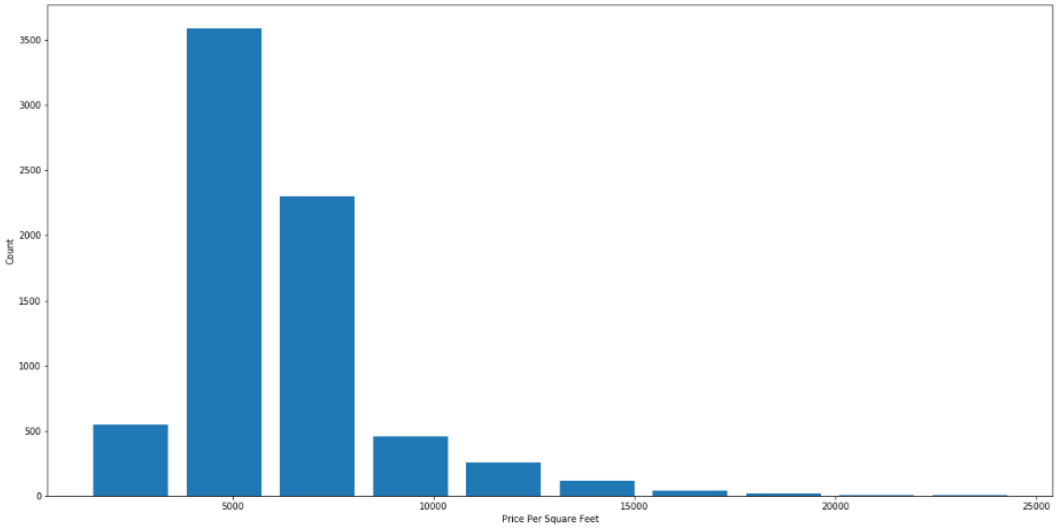


Figure 56: Histogram

To explore the unique value for the bathroom feature, I typed the command as shown in (Figure 57).

```
In [52]: df8.bath.unique()

Out[52]: array([ 4.,  3.,  2.,  5.,  8.,  1.,  6.,  7.,  9., 12., 16., 13.])
```

Figure 57: Bathroom feature

I found that, there are properties with 13 bathrooms from the above output.

But, In Bengaluru there are properties with greater than 10 bathrooms (Figure 58).

```
In [53]: df8[df8.bath>10]

Out[53]:
```

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

Figure 58: Properties with greater than 10 bathrooms

Again, I plot the histogram (Figure 59). In this, by calling the “plt.hist”, the Histogram can be plotted. “rwidth” defines the width of the bar.

```
In [54]: plt.hist(df8.bath,rwidth=0.8)
         plt.xlabel("Number of bathrooms")
         plt.ylabel("Count")
Out[54]: Text(0, 0.5, 'Count')
```

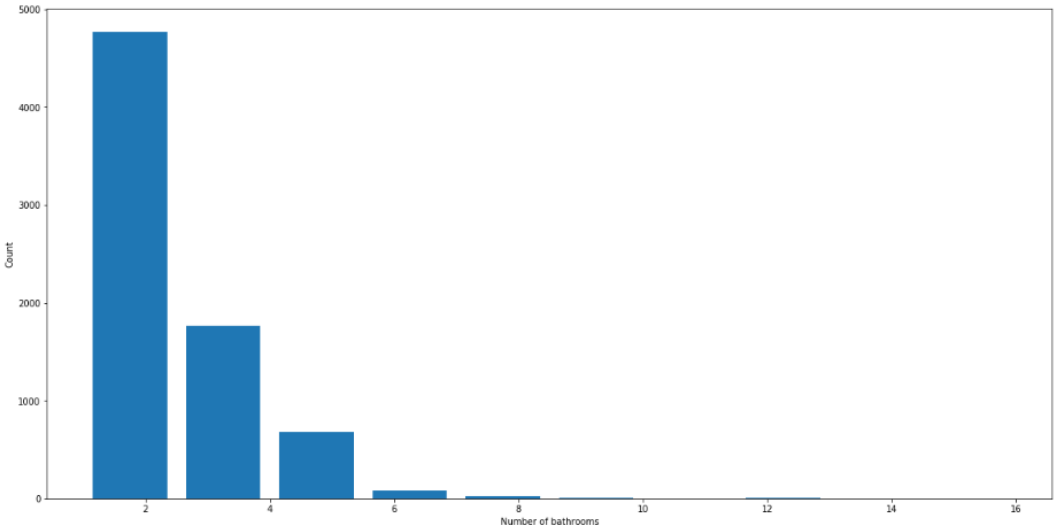


Figure 59: Plot the Histogram again

As a result of the histogram, most of the properties are having 2 bathrooms and few outliers which had more number of bathrooms.

From this above result, I came to a decision that, whenever the number of bathrooms > number of BHK, then, I removed that also as an outlier. The following are the outliers (Figure 60), which I removed.

```
In [55]: df8[df8.bath>df8.bhk+2]
Out[55]:
```

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

Figure 60: Removed the outliers

Finally, I removed all the outliers and saw how many rows are available (Figure 61).

```
In [56]: df9 = df8[df8.bath<df8.bhk+2]
         df9.shape
Out[56]: (7251, 7)
```

Figure 61: Checked the number of rows available

From the above output, I removed all the outliers, and I got a new data frame (df9), which has 7251 rows.

To do the machine learning training, I dropped some unnecessary features like price_per_sqft and size. Because, for size, there is another feature called as BHK. So, I dropped the “size” feature. “price_per_sqft” feature also I dropped it, because it is only used for outlier detection. So, again I created a new data frame, where I dropped the above mentioned features. The following (Figure 62) depicts it.

```
In [57]: df10 = df9.drop(['size','price_per_sqft'],axis='columns')
         df10.head(3)
```

Out[57]:

	location	total_sqft	bath	price	bhk
0	1st Block Jayanagar	2850.0	4.0	428.0	4
1	1st Block Jayanagar	1630.0	3.0	194.0	3
2	1st Block Jayanagar	1875.0	2.0	235.0	3

Figure 62: New columns available after dropping

As Machine Learning model cannot interpret text data, the "location" column need to be converted. Because, it is text column. So, I converted that column into numeric. For the conversion I used Pandas, Dummies method. The following (Figure 63) shows it.

```
In [58]: pd.get_dummies(df10.location)
```

Out[58]:

	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar	9th Phase JP Nagar	...	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield	Yelach
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10232	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
10233	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
10236	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
10237	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
10240	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

7251 rows × 242 columns

Figure 63: One hot encoding (Convert the text column into numerical column)

In above (Figure 63), I hot encoded the column named as "location", and the function I used is pd.get_dummies(). While I ran, it created a new column for each of the location. Example, for 1st Block Jayanagar, for that it created a new column and it set the value to 1 and the rest are as 0.

Next, to store the dummy column into a separate data frame, I wrote the command as shown in (Figure 64).

```
In [59]: dummies = pd.get_dummies(df10.location)
         dummies.head(3)
```

Out[59]:

	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar	9th Phase JP Nagar	...	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield	Yelachenah
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

3 rows × 242 columns

Figure 64: Stored the dummy column into a separate data frame

Next, I append this into the main data frame. For this I created a new data frame (df11), and I concatenate (df10) and the dummies. Next I dropped the last column named as "other". This is shown in (Figure 65).

```
In [60]: df11 = pd.concat([df10,dummies.drop('other',axis='columns')],axis='columns')
         df11.head(3)
```

Out[60]:

	location	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefiel
0	1st Block Jayanagar	2850.0	4.0	428.0	4	1	0	0	0	0	...	0	0	0	0	
1	1st Block Jayanagar	1630.0	3.0	194.0	3	1	0	0	0	0	...	0	0	0	0	
2	1st Block Jayanagar	1875.0	2.0	235.0	3	1	0	0	0	0	...	0	0	0	0	

3 rows × 246 columns

Figure 65: Append to the main data frame and removed the "other" column

It has all the necessary features such as total_sqft, bath, price etc and all the location columns are dummy encoded.

Then as the "location" column is already included, I dropped that column, and created a new data frame known as (df12). This is shown in (Figure 66).

```
In [61]: df12 = df11.drop('location',axis='columns')
         df12.head(2)
```

Out[61]:

	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield
0	2850.0	4.0	428.0	4	1	0	0	0	0	0	...	0	0	0	0	0
1	1630.0	3.0	194.0	3	1	0	0	0	0	0	...	0	0	0	0	0

2 rows × 245 columns

Figure 66: Dropped "location" column and created a new data frame (df12)

So, for this project the pipeline went up to df12. Next, in order to build the model, first I checked the rows and columns (Figure 67). This shows that there are 7251 rows and 245 columns.

```
In [62]: df12.shape
Out[62]: (7251, 245)
```

Figure 67: Current rows and columns

As "price" is the dependent variable, I dropped it from the data frame. But, the X variable, which I created is an independent variable. So, I dropped the dependent variable price from the data frame. This is shown in (Figure 68).

```
In [63]: X = df12.drop('price',axis='columns')
         X.head()
```

Out[63]:

	total_sqft	bath	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield
0	2850.0	4.0	4	1	0	0	0	0	0	0	...	0	0	0	0	0
1	1630.0	3.0	3	1	0	0	0	0	0	0	...	0	0	0	0	0
2	1875.0	2.0	3	1	0	0	0	0	0	0	...	0	0	0	0	0
3	1200.0	2.0	3	1	0	0	0	0	0	0	...	0	0	0	0	0
4	1235.0	2.0	2	1	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 244 columns

Figure 68: Dropped the dependent variable "price"

Next, I saw the Y axis (Figure 69).

```
In [64]: y = df12.price
         y.head()

Out[64]: 0    428.0
         1    194.0
         2    235.0
         3    130.0
         4    148.0
         Name: price, dtype: float64
```

Figure 69: Y Axis

So, the X and Y axis got finished. Train and Test data sets are the two types of data sets. To train the model, train data sets are used. Test data sets are used to evaluate the model performance.

So, I imported train_test_split method from sklearn.model_selection and the test_size = 0.2. Which means, I wanted 20% of my samples needed to be as test samples, as well as the remaining 80% is used for model training. Shown in (Figure 70).

```
In [65]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=10)
```

Figure 70: Train and Test Data sets

In (Figure 71) I had created a linear regression model, then I called the fit() method for X train and Y train. Once the model is trained, it need to evaluate the score of the created model. So, as the result it produce the score as 0.8452.

```
In [66]: from sklearn.linear_model import LinearRegression
         lr_clf = LinearRegression()
         lr_clf.fit(X_train,y_train)
         lr_clf.score(X_test,y_test)

Out[66]: 0.8452277697874276
```

Figure 71: Score for the model

Next, I used K Fold Cross Validation method (Figure 72). Here, I imported some needed methods. Next, I created a ShuffleSplit for cross validation. ShuffleSplit will helped to randomize my sample. So that, there will be an equal distribution of data samples. Then with the help of cross validation I got the scores as shown in (Figure 72).

```
In [67]: from sklearn.model_selection import ShuffleSplit
         from sklearn.model_selection import cross_val_score

         cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

         cross_val_score(LinearRegression(), X, y, cv=cv)

Out[67]: array([0.82430186, 0.77166234, 0.85089567, 0.80837764, 0.83653286])
```

Figure 72: Score for the model above 80%

Mostly, I got 80% score all the time. Except, 0.77166 (i.e.: 70%).

There are regression techniques like “Lasso” regression, “Decision tree regression” etc are available. In order to find out which algorithm gives the best score, I used a method called as “Grid Search CV”.

Other than “Linear Regression” I also tried “Lasso” and “Decision Tree Regression”. Then I also wrote a function for “Grid Search CV”. This is shown in (Figure 73). I had specified the algorithms as well as the parameters. So, “Grid Search CV” not only tells the best algorithm

selection, but also it helps to get the best parameter selection. This is known as “Hyper parameter tuning”.

```
In [68]: from sklearn.model_selection import GridSearchCV

         from sklearn.linear_model import Lasso
         from sklearn.tree import DecisionTreeRegressor

         def find_best_model_using_gridsearchcv(X,y):
             algos = {
                 'linear_regression' : {
                     'model': LinearRegression(),
                     'params': {
                         'normalize': [True, False]
                     }
                 },
                 'lasso': {
                     'model': Lasso(),
                     'params': {
                         'alpha': [1,2],
                         'selection': ['random', 'cyclic']
                     }
                 },
                 'decision_tree': {
                     'model': DecisionTreeRegressor(),
                     'params': {
                         'criterion' : ['mse','friedman_mse'],
                         'splitter': ['best','random']
                     }
                 }
             }
             scores = []
             cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
             for algo_name, config in algos.items():
                 gs = GridSearchCV(config['model'], config['params'], cv=cv, return_train_score=False)
                 gs.fit(X,y)
                 scores.append({
                     'model': algo_name,
                     'best_score': gs.best_score_,
                     'best_params': gs.best_params_
                 })

             return pd.DataFrame(scores,columns=['model','best_score','best_params'])

         find_best_model_using_gridsearchcv(X,y)
```

Out[68]:

	model	best_score	best_params
0	linear_regression	0.818354	{'normalize': False}
1	lasso	0.687429	{'alpha': 1, 'selection': 'cyclic'}
2	decision_tree	0.723212	{'criterion': 'friedman_mse', 'splitter': 'best'}

Figure 73: Model and Score

In the (Figure 73), the ShuffleSplit helps to randomly shuffle the sample, which helped to get better results. Then I initialized the GridSearch CV object, with model as well as with parameter. For Cross Validation I used the “cv” object. The fit() method is used to append the scores into the score list. Finally, the resultant score will be returned as the data frame.

As a result, from the above (Figure 73), the Linear Regression had the most score compared to the other models. I.e.: 0.818354. And also the best parameters for the Linear Regression are normalize and false. So, I concluded that Linear Regression Model is the best model.

If I need to find the X.columns I wrote the command as shown in (Figure 74).

```
In [69]: X.columns

Out[69]: Index(['total_sqft', 'bath', 'bhk', '1st Block Jayanagar',
                '1st Phase JP Nagar', '2nd Phase Judicial Layout',
                '2nd Stage Nagarbhavi', '5th Block Hbr Layout', '5th Phase JP Nagar',
                '6th Phase JP Nagar',
                ...
                'Vijayanagar', 'Vishveshwarya Layout', 'Vishwapriya Layout',
                'Vittasandra', 'Whitefield', 'Yelachenahalli', 'Yelahanka',
                'Yelahanka New Town', 'Yelenahalli', 'Yeshwanthpur'],
               dtype='object', length=244)
```

Figure 74: X.columns

If I need to know the location for 2nd Phase Judicial Layout, I typed as shown in (Figure 75). As the output it gives the column index as 5.

```
In [70]: np.where(X.columns=='2nd Phase Judicial Layout')[0][0]
Out[70]: 5
```

Figure 75: Column Index for 2nd Phase Judicial Layout

To get the predict price prediction, I wrote the command as shown in (Figure 76).

```
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 lr_clf.predict([x])[0]
```

Figure 76: Price Prediction

Finally, to check the price prediction for the model, I gave the inputs as shown in (Figure 77), where 1st Phase JP Nagar, sqft_area is 1000, 2 BHK and 2 bath. So, as the estimated price I got 83. 49904 Indian Lakh Rupees.

```
In [72]: predict_price('1st Phase JP Nagar',1000, 2, 2)
Out[72]: 83.49904677167729
```

Figure 77: Prediction Price for the real estate at 1st Phase JP Nagar

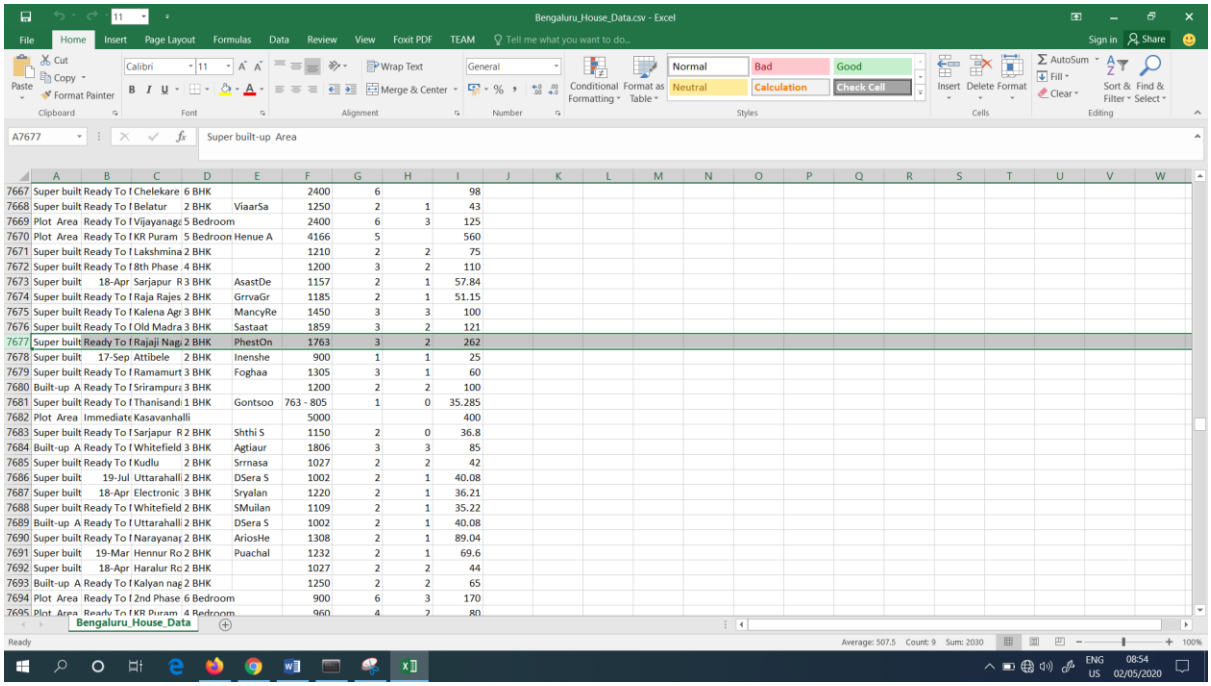
Then I typed the command as shown in (Figure 78).

```
In [76]: predict_price('1st Phase JP Nagar',1000, 2,3)
Out[76]: 81.7261690074302
```

Figure 78: Prediction Price for the real estate at 1st Phase JP Nagar with different BHK

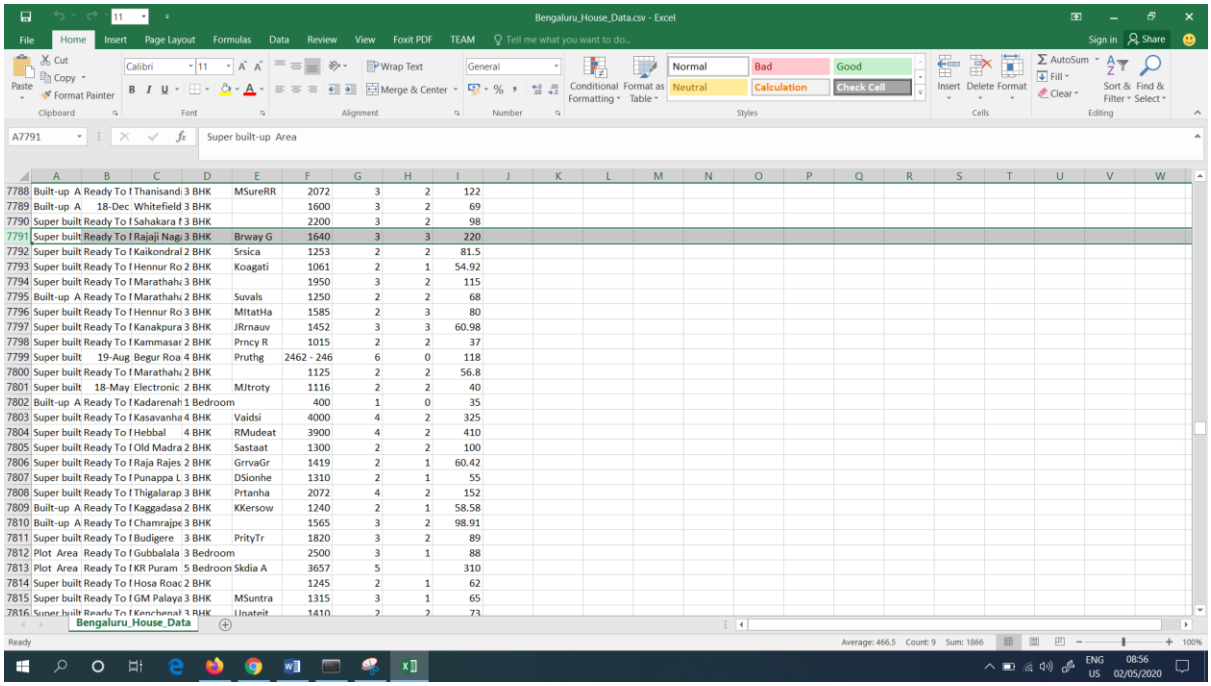
In the above (Figure 78), I got little less price compared to the previous output. When the BHK goes higher the price also need to go higher as usual. But, it didn't happen. Also in the data set for the same square feet area, when there are 3 bedrooms they cost less, compared to 2 bedrooms. The reasons can be if there are only 2 bedrooms then it can be big enough, compared to 3 bedrooms. Another reason is because as the data was distributed sometimes there aren't enough information on why 3 bedrooms cost less compared to 2 bedrooms.

For example, as shown in the following (Figure 79 and Figure 80), In "Rajaji Nagar" there are 2 Bedrooms' and 3 Bedrooms' real estate, where the square feet is little big but the price of 2 Bedrooms' real estate is 42 Indian Lakh Rupees is higher than the 3 Bedrooms' real estate.



7667	Super built	Ready To f	Chelekare	6 BHK		2400	6	98
7668	Super built	Ready To f	Belatur	2 BHK	ViaarSa	1250	2	1	43
7669	Plot Area	Ready To f	Vijayanaga	5 Bedroom		2400	6	3	125
7670	Plot Area	Ready To f	KR Puram	5 Bedroom	Henue A	4166	5		560
7671	Super built	Ready To f	Lakshmina	2 BHK		1210	2	2	75
7672	Super built	Ready To f	8th Phase	4 BHK		1200	3	2	110
7673	Super built	18-Apr	Sarjapur R	3 BHK	AsastDe	1157	2	1	57.84
7674	Super built	Ready To f	Raja Rajes	2 BHK	GrrvaGr	1185	2	1	51.15
7675	Super built	Ready To f	Kalena Agr	3 BHK	MancyRe	1450	3	3	100
7676	Super built	Ready To f	Old Madra	3 BHK	Sastaat	1859	3	2	121
7677	Super built	Ready To f	Rajaji Nagi	2 BHK	PhestOn	1763	3	2	262
7678	Super built	17-Sep	Attibele	2 BHK	Inenshe	900	1	1	25
7679	Super built	Ready To f	Ramamurt	3 BHK	Foghaa	1305	3	1	60
7680	Built-up A	Ready To f	Srirampura	3 BHK		1200	2	2	100
7681	Super built	Ready To f	Thanisandi	1 BHK	Gontsoo	763 - 805	1	0	35.285
7682	Plot Area	Immediate	Kasavanhalli			5000			400
7683	Super built	Ready To f	Sarjapur R	2 BHK	Shthi S	1150	2	0	36.8
7684	Built-up A	Ready To f	Whitefield	3 BHK	Agtiaur	1806	3	3	85
7685	Super built	Ready To f	Kudlu	2 BHK	Srrnasa	1027	2	2	42
7686	Super built	19-Jul	Uttarahalli	2 BHK	DSera S	1002	2	1	40.08
7687	Super built	18-Apr	Electronic	3 BHK	Sryalan	1220	2	1	36.21
7688	Super built	Ready To f	Whitefield	2 BHK	SMuilan	1109	2	1	35.22
7689	Built-up A	Ready To f	Uttarahalli	2 BHK	DSera S	1002	2	1	40.08
7690	Super built	Ready To f	Narayanap	2 BHK	AriosHe	1308	2	1	89.04
7691	Super built	19-Mar	Hennur Ro	2 BHK	Puachal	1232	2	1	69.6
7692	Super built	18-Apr	Haralur Ro	2 BHK		1027	2	2	44
7693	Built-up A	Ready To f	Kalyan nag	2 BHK		1250	2	2	65
7694	Plot Area	Ready To f	2nd Phase	6 Bedroom		900	6	3	170
7695	Plot Area	Ready To f	KR Puram	4 Bedroom		960	4	2	80

Figure 79: Price of 2 Bedroom real estate



7788	Built-up A	Ready To f	Thanisandi	3 BHK	MSureRR	2072	3	2	122
7789	Built-up A	18-Dec	Whitefield	3 BHK		1600	3	2	69
7790	Super built	Ready To f	Sahakara f	3 BHK		2200	3	2	98
7791	Super built	Ready To f	Rajaji Nagi	3 BHK	Brway G	1640	3	3	220
7792	Super built	Ready To f	Kaikondral	2 BHK	Srsica	1253	2	2	81.5
7793	Super built	Ready To f	Hennur Ro	2 BHK	Koagati	1061	2	1	54.92
7794	Super built	Ready To f	Marathahi	3 BHK		1950	3	2	115
7795	Built-up A	Ready To f	Marathahi	2 BHK	Suvals	1250	2	2	68
7796	Super built	Ready To f	Hennur Ro	3 BHK	MItatHa	1585	2	3	80
7797	Super built	Ready To f	Kanakpura	3 BHK	JRrnauv	1452	3	3	60.98
7798	Super built	Ready To f	Kammasar	2 BHK	Prncy R	1015	2	2	37
7799	Super built	19-Aug	Begur Roa	4 BHK	Pruthg	2462 - 246	6	0	118
7800	Super built	Ready To f	Marathahi	2 BHK		1125	2	2	56.8
7801	Super built	18-May	Electronic	2 BHK	MJtroty	1116	2	2	40
7802	Built-up A	Ready To f	Kadarenah	1 Bedroom		400	1	0	35
7803	Super built	Ready To f	Kasavanha	4 BHK	Vaidsi	4000	4	2	325
7804	Super built	Ready To f	Hebbal	4 BHK	RMudeat	3900	4	2	410
7805	Super built	Ready To f	Old Madra	2 BHK	Sastaat	1300	2	2	100
7806	Super built	Ready To f	Raja Rajes	2 BHK	GrrvaGr	1419	2	1	60.42
7807	Super built	Ready To f	Punappa L	3 BHK	DSionhe	1310	2	1	55
7808	Super built	Ready To f	Thigalarap	3 BHK	Prtanha	2072	4	2	152
7809	Built-up A	Ready To f	Kaggadasa	2 BHK	KKersow	1240	2	1	58.58
7810	Built-up A	Ready To f	Chamrajpe	3 BHK		1565	3	2	98.91
7811	Super built	Ready To f	Budigere	3 BHK	PrityTr	1820	3	2	89
7812	Plot Area	Ready To f	Gubbalala	3 Bedroom		2500	3	1	88
7813	Plot Area	Ready To f	KR Puram	5 Bedroom	Skdia A	3657	5		310
7814	Super built	Ready To f	Hosa Roac	2 BHK		1245	2	1	62
7815	Super built	Ready To f	GM Palaya	3 BHK	MSuntra	1315	3	1	65
7816	Super built	Ready To f	Kenchenah	3 BHK	Unatait	1410	2	2	73

Figure 80: Price of 3 Bedroom real estate

Next, I wrote the command and checked for 3 bedrooms and 3 bathrooms. This is shown in (Figure 81).

```
In [73]: predict_price('1st Phase JP Nagar',1000, 3, 3)

Out[73]: 86.805193951943
```

Figure 81: Prediction Price for 1st Phase JP Nagar with 3 bath and 3 BHK

And also I checked the price for Indira Nagar. This is shown in (Figure 82).

```
In [74]: predict_price('Indira Nagar',1000, 2,2)
Out[74]: 181.27815484007027
```

Figure 82: Prediction Price for Indira Nagar with 2 BHK and 2 bath

Without changing the location, I changed only the BHK and bath value as 3 (Figure 83).

```
In [75]: predict_price('Indira Nagar',1000, 3,3)
Out[75]: 184.58430202033597
```

Figure 83: Prediction Price for Indira Nagar with 3 BHK and 3 bath

Up to the above mentioned steps, I build the model. Finally, I export this model to a pickle file. In the (Figure 84), first I import the pickle file, and wrote the function pickle.dump(). Then I passed the model and classifier as the arguments. And when I execute the command the pickle file is exported and it is found in the file as shown in (Figure 85).

```
In [77]: import pickle
         with open('banglore_home_prices_model.pickle','wb') as f:
             pickle.dump(lr_clf,f)
```

Figure 84: Command used to export the pickle file

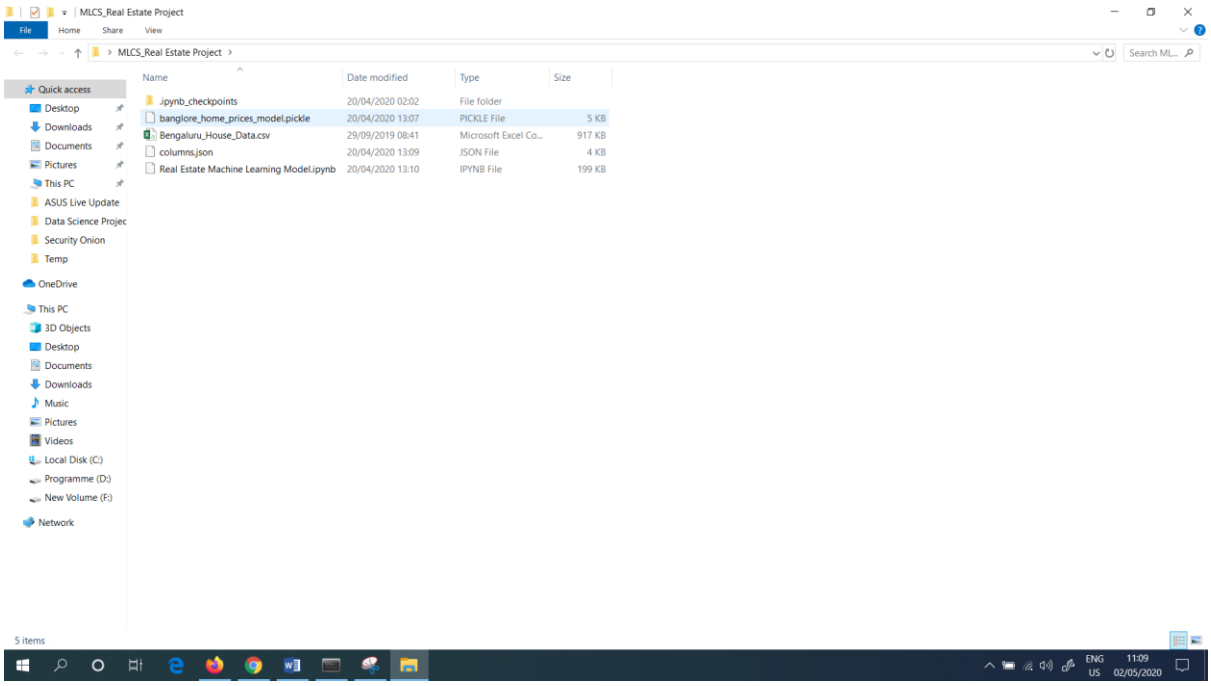


Figure 85: Pickle file in the directory

As the above mentioned pickle file doesn't have the actual data, the file size is only 5 KB. I.e.: The file size is very small.

Finally, I need the column information. Because, the way they are structured and how they are indexed into the list are important in order to make the prediction. To do this, I exported the information into a "json" file. In the command written, which is shown in (Figure 86), I converted the words, into lower case. Because Normally all the words are written in Upper and Lower case. So, I thought it will be in uniformity, when I changed all into lower case.

```
In [78]: import json
         columns = {
             'data_columns' : [col.lower() for col in X.columns]
         }
         with open("columns.json","w") as f:
             f.write(json.dumps(columns))
```

Figure 86: Command written to export the json file

When I executed the written command it is found in the file as shown in (Figure 87).

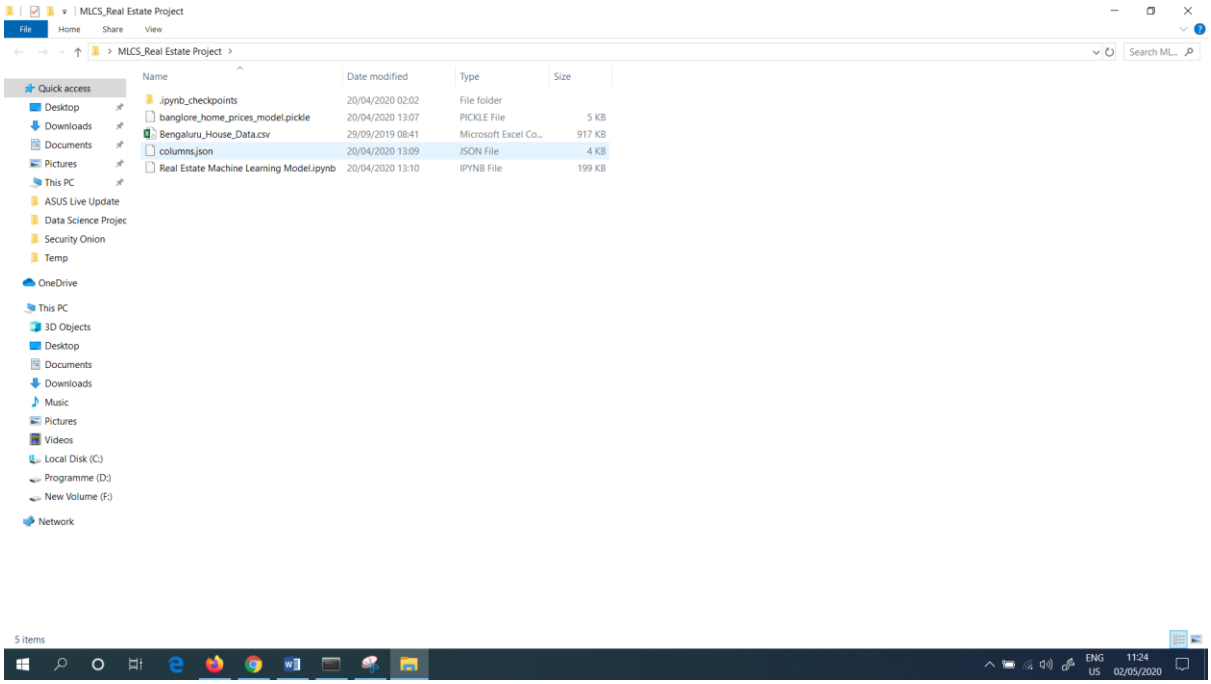


Figure 87: Exported json file

Then I clicked the json file and opened it with the help of "Notepad++", where it shows all the available columns (Figure 88).

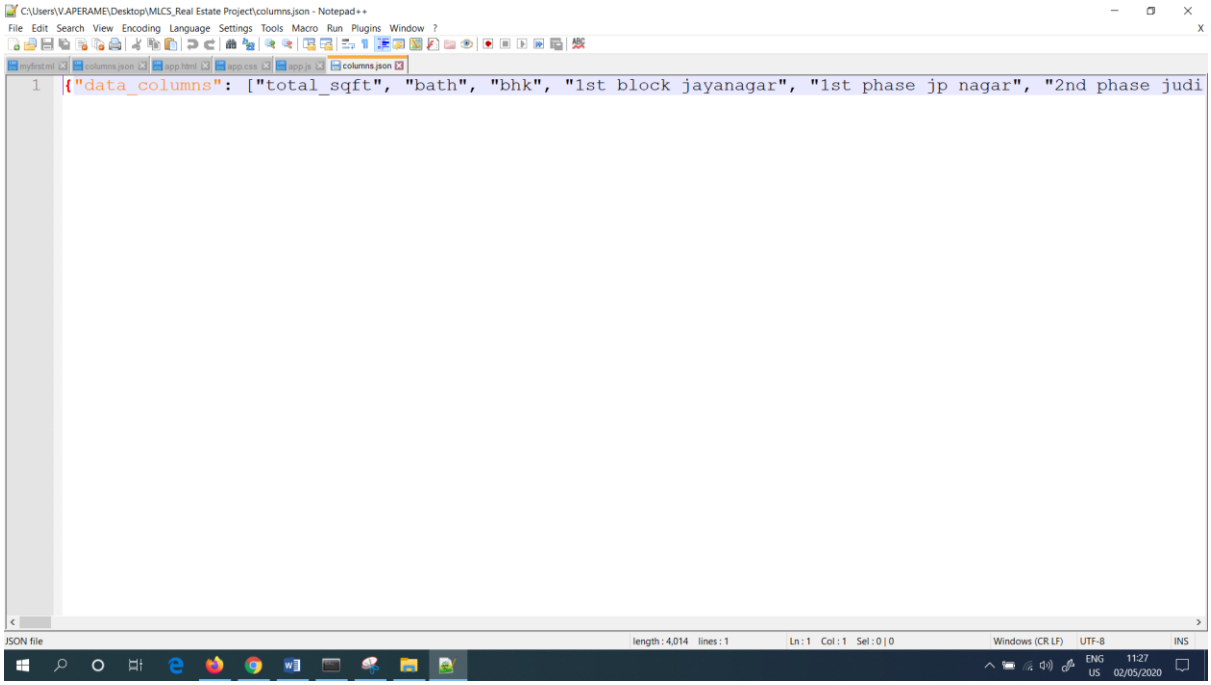


Figure 88: json file opened in Notepad++

json file and pickle model are used in the Python Flask Server. Next, I wrote the Python flask server which is used for HTTP request made from the UI (User Interface). The Python Flask Server is used as the backend for the UI Application.

To write the Python Flask Server I used the software known as “PyCharm”. It is a community edition. So I downloaded it for free and I installed it in my PC (Personal Computer). And the version I used is 2019.3.3. Also it is a Python editor.

Next, I clicked the “File” and then I clicked the “Open” option. Then, I selected the file named as “BHP” which I saved it in my desktop. This is shown in (Figure 89).

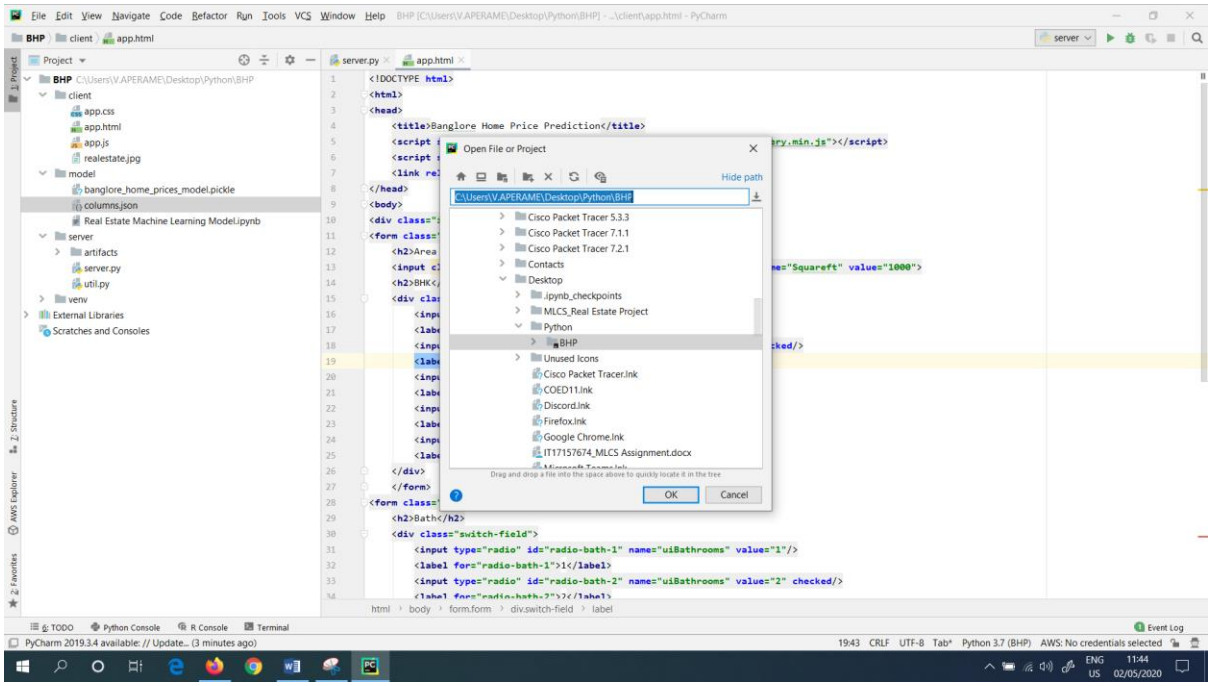


Figure 89: Opened the Project file in “PyCharm”

Next, I have created 3 folders namely client, model and server under the folder named as “BHP”. Under the client folder, the UI Application have presented. Under the server folder, the Python Flask Server have presented. And the model folder contains the jupyter notebook and the artifacts (Pickle file and the columns.json file).

Then, in the PyCharm, under the server folder I created a file with the name as “server.py”. This is shown in (Figure 90).

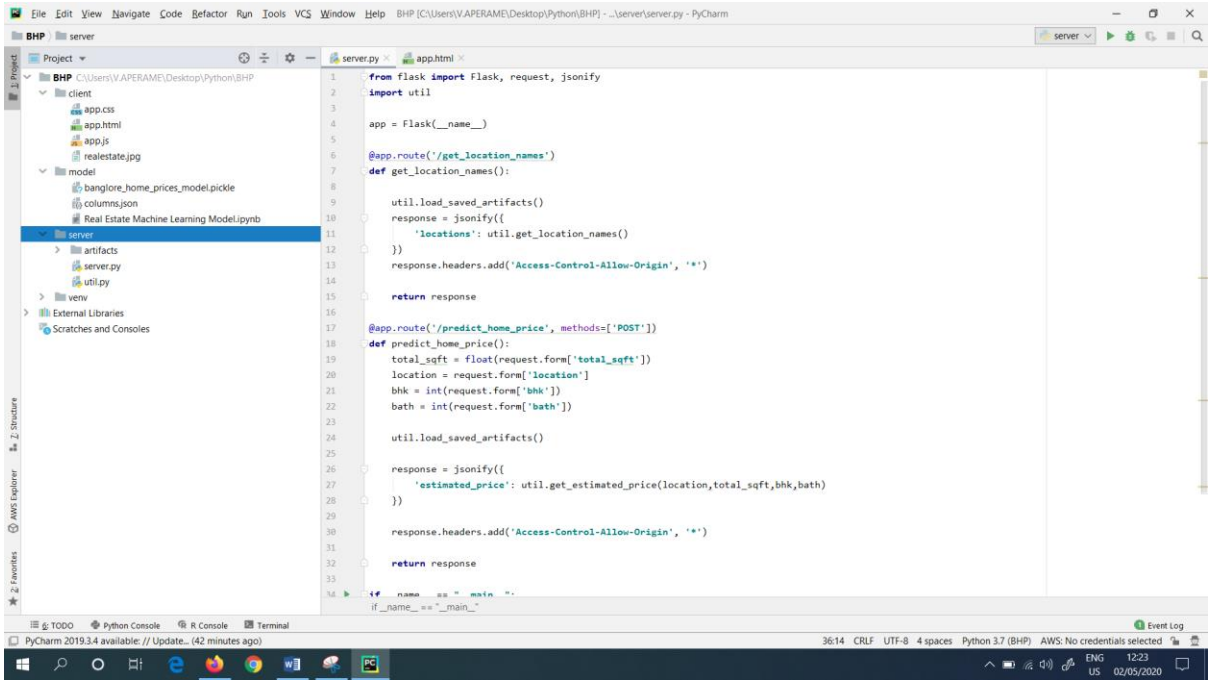


Figure 90: server.py file in "PyCharm"

Next, I configured the Interpreter to "Anaconda". Because Anaconda normally comes with Flask. So, first I went to "File". Then clicked the "Settings". Next, I searched for the project interpreter. This is shown in (Figure 91).

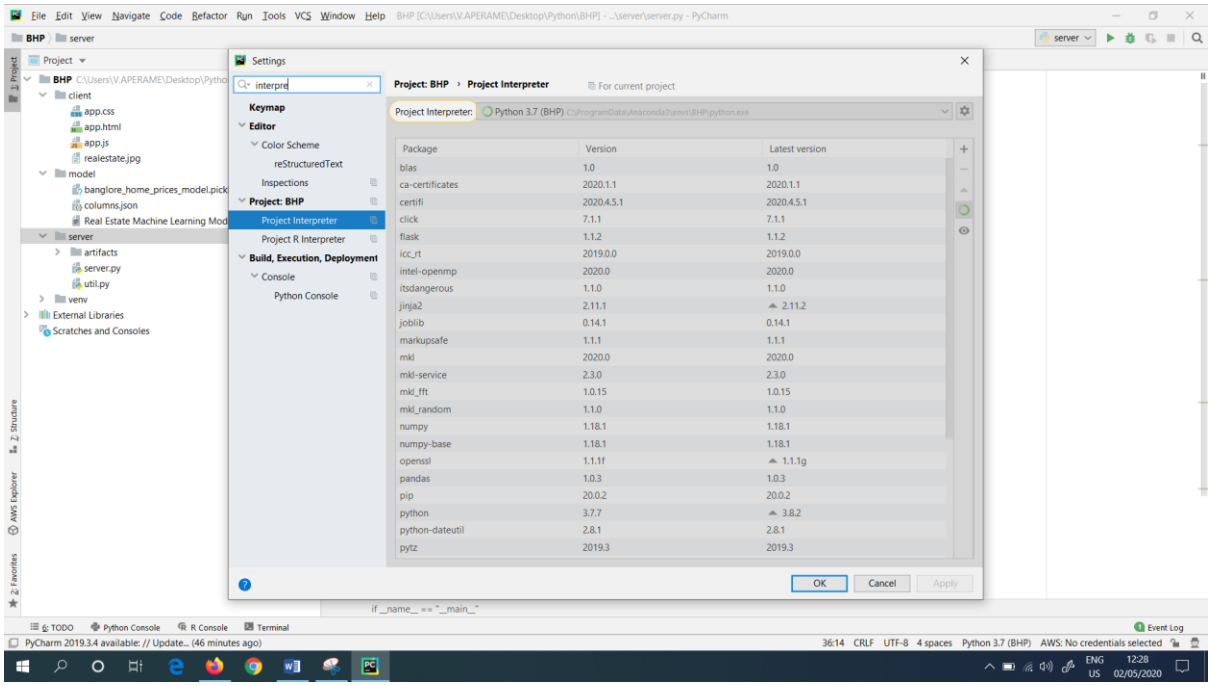


Figure 91: Configured the Interpreter to "Anaconda"

"Server.py" is the main server file, where I imported the flask module. There are routines available, and the routine should return the locations in Bengaluru. The "columns.json" file consists all the locations available.

Next, I created a sub directory named as “artifacts”, under the main folder named as “server”. There I copied the pickle file and the columns.json file, inside the “artifacts” folder (Figure 92).

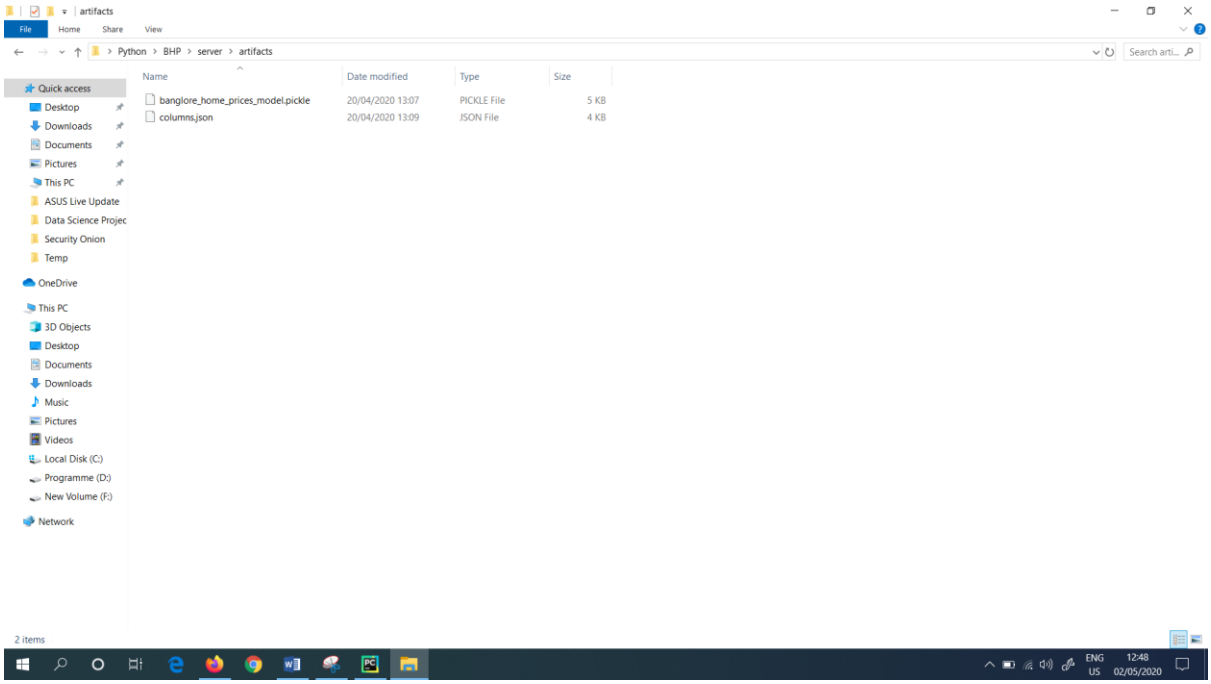


Figure 92: Sub folder named as “artifacts”

In the “server.py” file, the “get_location_names()” function returns all the locations available. The “jsonify” method helps to return all the locations. Next for the locations I created a new file named as “util”. Util contains all the core routines whereas the server will do the routing of the request and response. Under the “util.py” it contains a function named as “get_location_names()”. This is shown in (Figure 93).

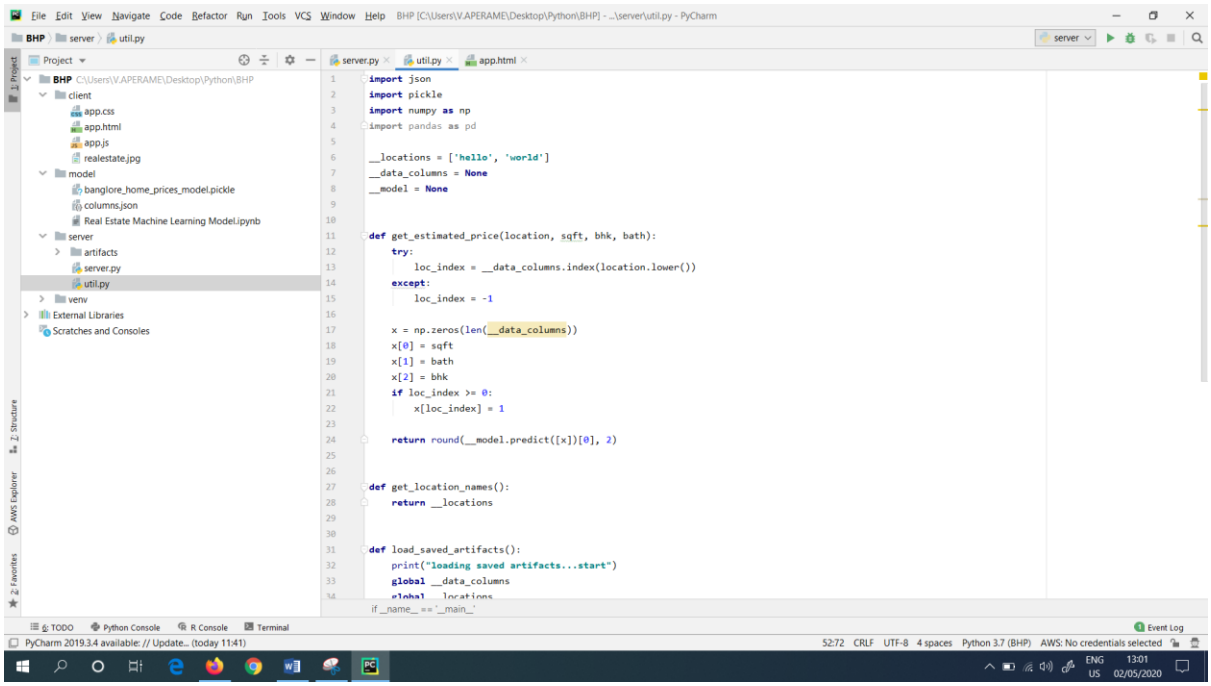


Figure 93: “util.py”

“load_saved_artifacts()” method helps to load the “columns.json” file as well as the saved pickle file. Under the “util.py” file I created three global variables namely “__locations”, “__data_columns” and “__model”. Then I opened the “columns.json” file. Also I used “Python Index Slicing” to get the elements from the list, starting from number 3. I.e.: [3:]. I also load the saved pickle model.

Next, I also wrote a function inorder to get the estimated price. So, the function I wrote is: get_estimated_price(). For this the arguments I used are “location”, “sqft”, “bhk” and “bath”. So, the input was a 2 Dimensional Array. I.e.: I used “x” as the numpy array.

Finally, the “x” gives the estimated price in Indian Lakh Rupees. And also, as this is a float number I rounded off the number to two decimal places. As a result I called the get_estimated_price() function, which produced the estimated price of the real estate. Next, I ran the util.py. And it showed me the result as shown in (Figure 94).

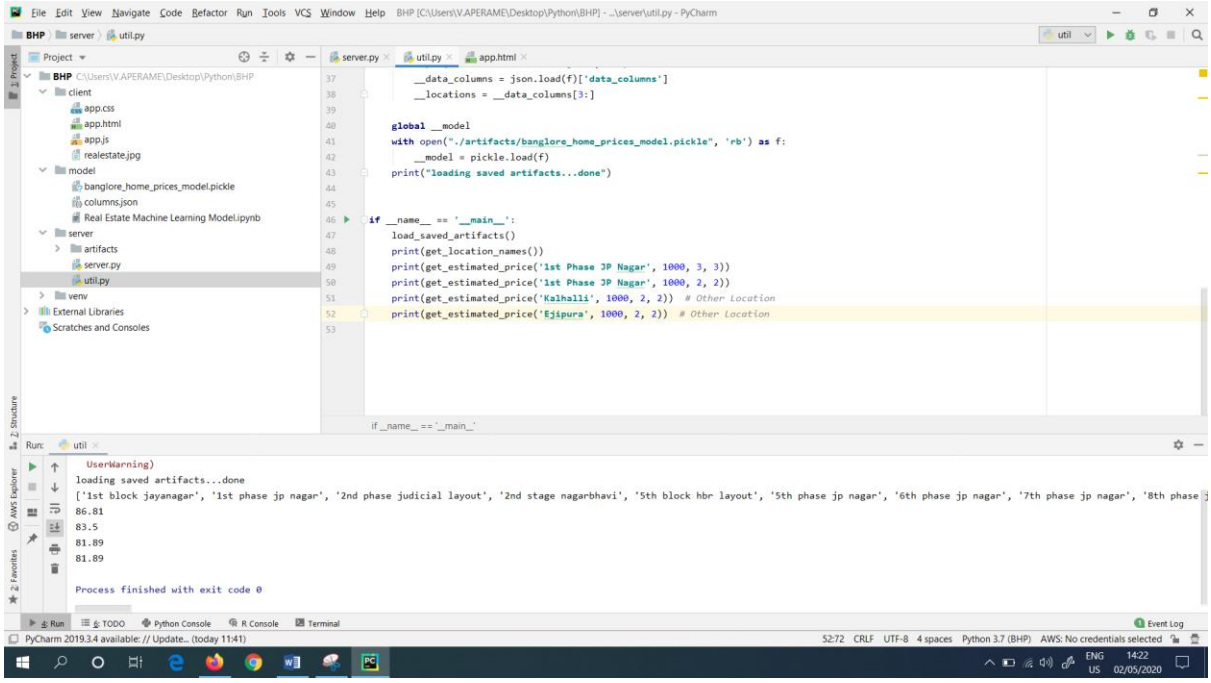


Figure 94: Ran the “util.py”

Now, the “utils.py” methods were ready. In the “server.py” I created another end point which was called as “predict_home_price()”. This takes the HTTP “Post” method. In the “server.py” I have imported the “request” class, and whenever I make the HTTP call from the HTML Application, I got all the inputs in requests.form. They are looked as shown in (Figure 95).

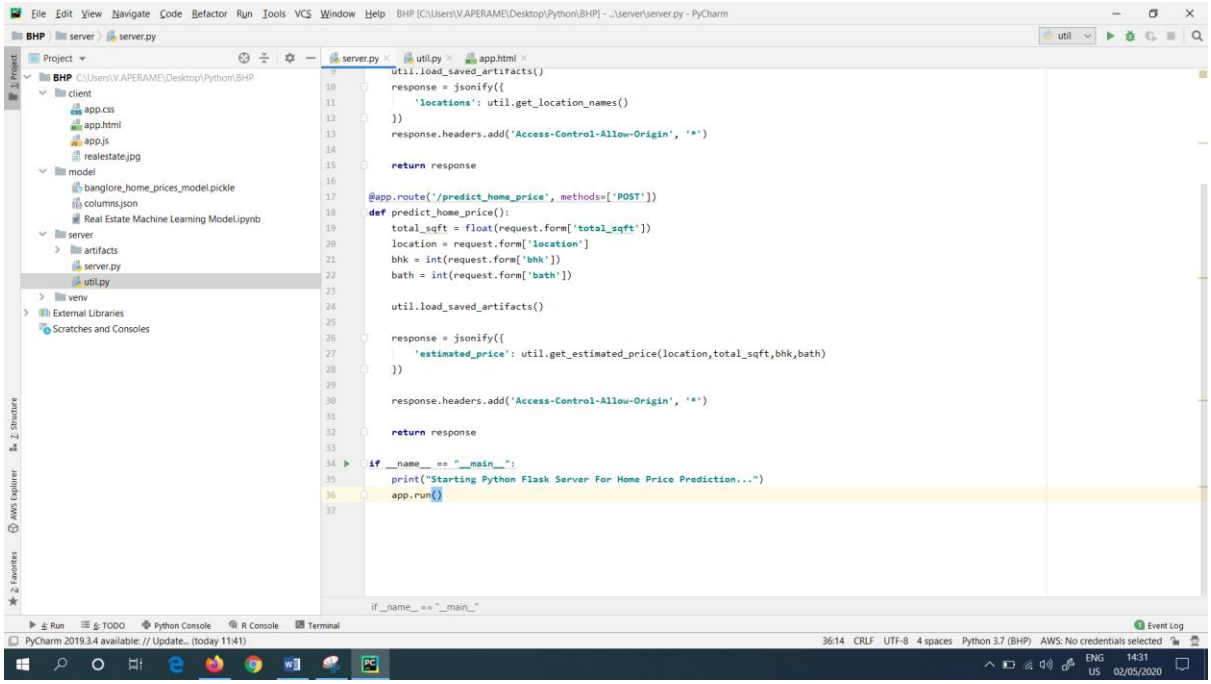


Figure 95: Requests in "server.py"

Next, I called the util.get_estimated_price() function, which returned the estimated price as a result.

To do the testing, I downloaded the "Postman" Application for free. The following (Figure 96) shows the website, where I downloaded the "Postman" Application. And I installed it.

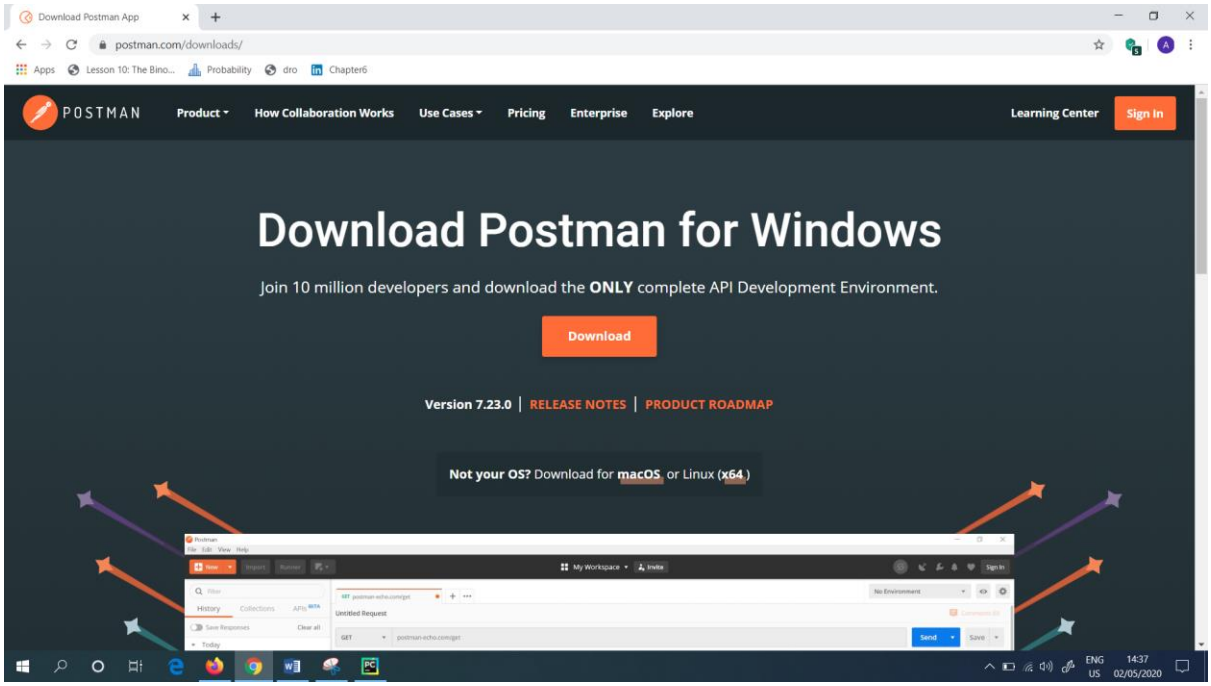


Figure 96: Downloaded the "Postman" Application

The above mentioned "Postman" Application allowed to test the HTTP requests. Including GET and POST requests.

In order to get the output for all the endpoint, at first most I ran the server named as (server.py). To do this, I clicked the “Run” button. This output is shown in (Figure 97).

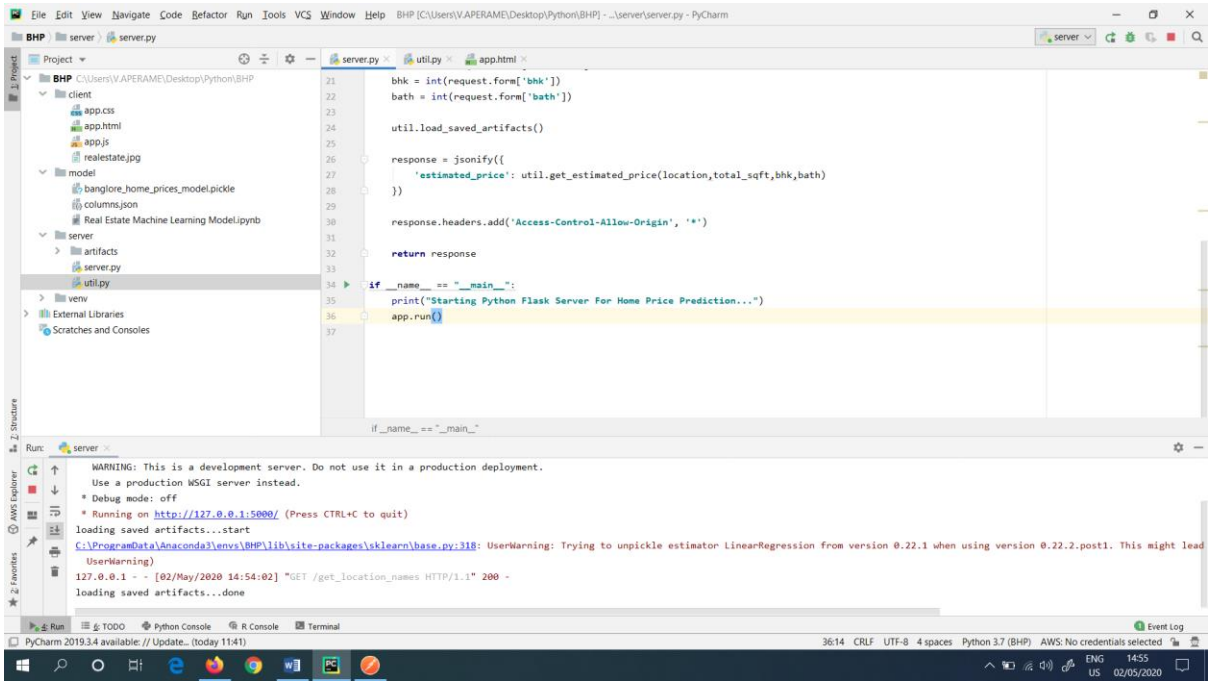


Figure 97: Ran the “server.py”

Next, first, I tested the “get_location_names” function in the Postman Application. To do this, first I changed the request type as “GET” and then I typed the URL and mentioned the HTTP end point. I.e.: get_location_names. The output is shown in (Figure 98).

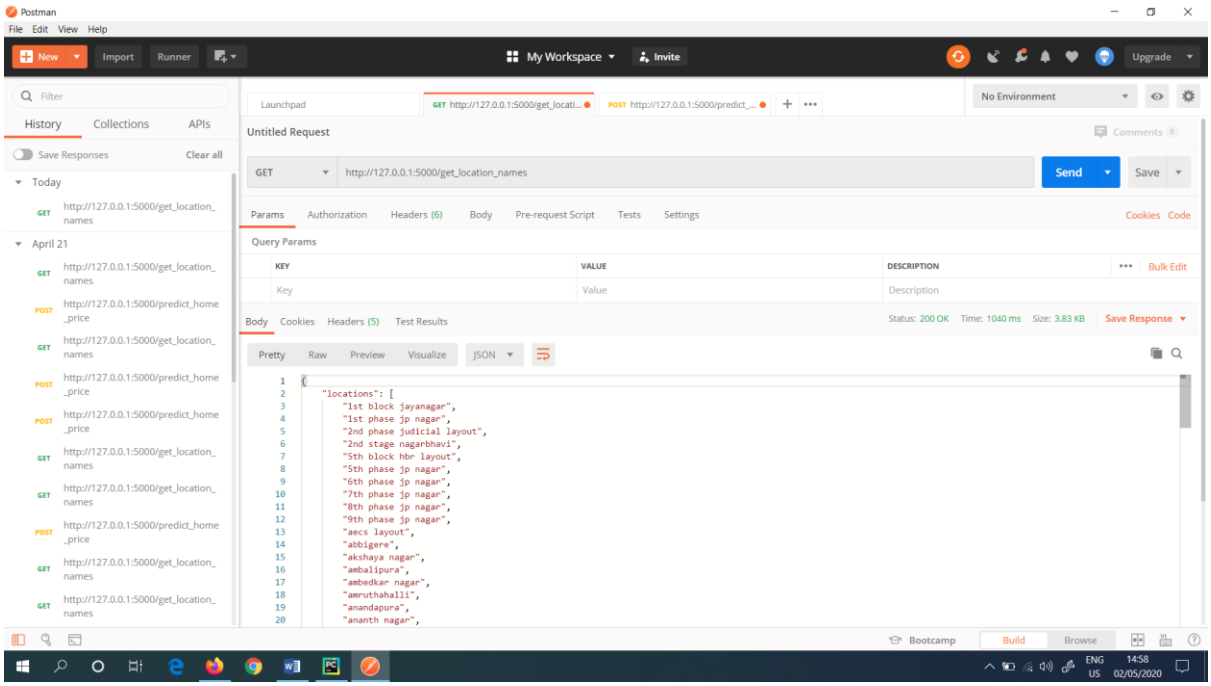


Figure 98: Tested “get_location_names” in the Postman Application

This produces all the available locations.

In the same way, I also checked for “predict_home_price”. In the “body” as I used the json form, I clicked the option known as “form-data” where it specified all the keys and values. Under the “Key” the options are total_sqft, location, bhk and bath. Also I input the values to the Key options namely, 1000, 1st Phase JP Nagar, 2, 2. And to get the output, I clicked the button known as “Send” which showed the output as in (Figure 99).

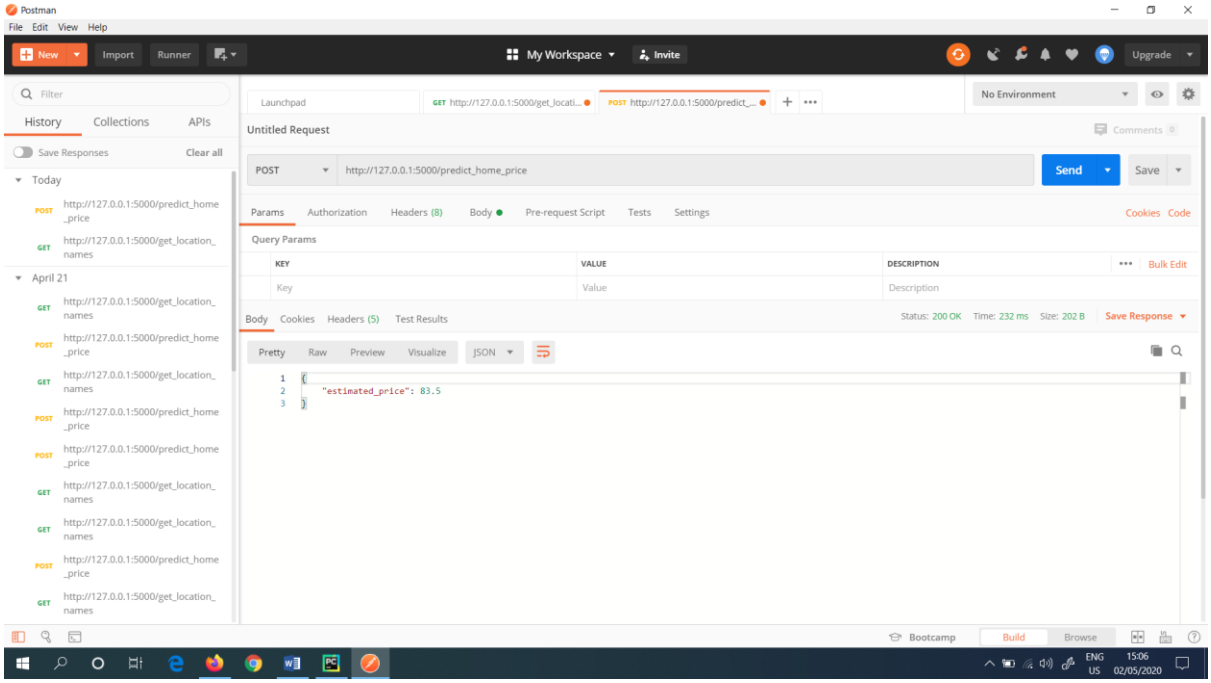


Figure 99: Tested “predict_home_price” in the Postman Application

It returned 83 Indian Lakh Rupees as the estimated price.

Upto now, the HTTP (Hypertext Transfer Protocol) end points and the Python Flask Server were ready. And next, I built the UI (User Interface) for this Real Estate Project. The UI were developed with the help of HTML, CSS (Cascading Style Sheets) and Java Script.

As to build the website, first I created 3 files under the “client” folder namely, “app.css”, “app.html” and “app.js”. This is showed in (Figure 100).

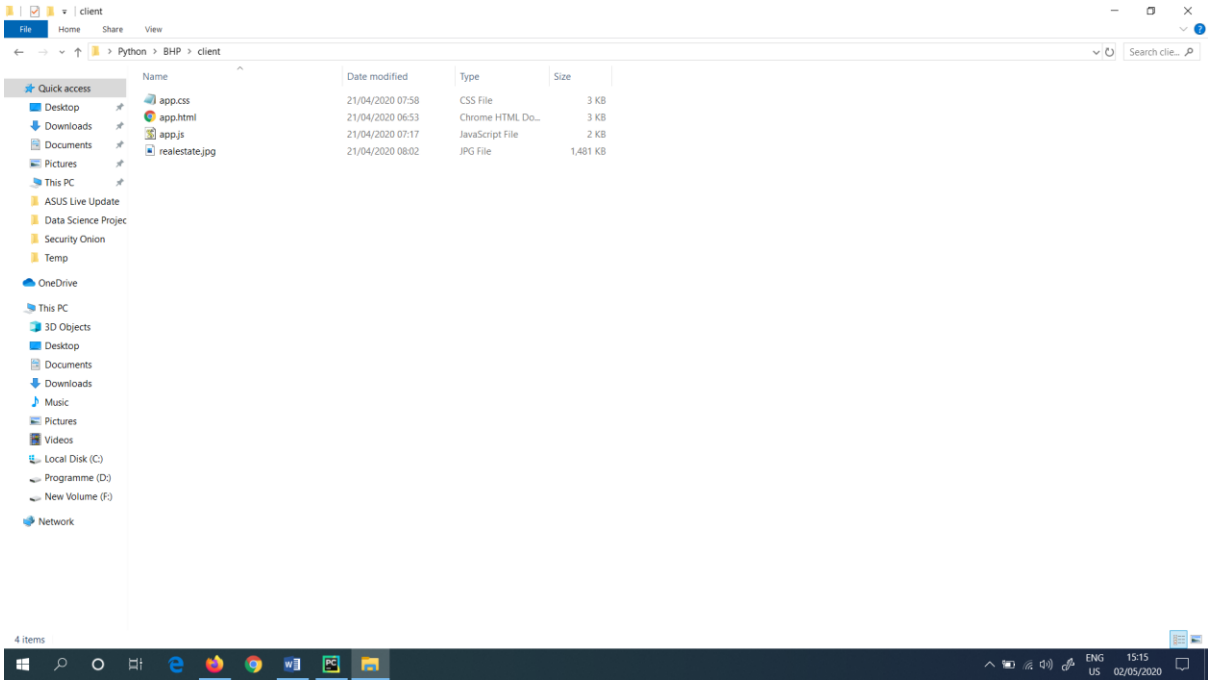


Figure 100: Files created for the website development

HTML contains the structure of the UI, CSS contains the colour and look of the website, JavaScript is used to retrieve the data from the backend. This is mostly used to communicate with the server.

Next, I opened the above mentioned 3 files, in Notepad++ and wrote the UI code. In the "app.html" file, it consists of 2 sections namely head and body. Where there is a title for this project, next I used jquery for HTTP calls. So, I imported the "jquery.js" file. Also I imported "app.css" and "app.js".

In the body section, I implemented segments like Area (Square Feet), BHK, Bath, Location etc. "Area" is an input entry as well as "BHK" and "Bath" are buttons. So, the user can select the values according to his/ her needs. The "BHK" and "Bath" have 5 options. Then the "Location" is a drop down list. The following (Figure 101) shows the "app.html" file.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Banglore Real Estate Price Prediction</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    <script src="app.js"></script>
    <link rel="stylesheet" href="app.css">
</head>
<body>
<div class="img"></div>
<form class="form">
    <h2>Area (Square Feet)</h2>
    <input class="area"  type="email" id="uiSqft" class="floatLabel" name="Squareft" value="1000">
    <h2>BHK</h2>
    <div class="switch-field">
        <input type="radio" id="radio-bhk-1" name="uiBHK" value="1"/>
        <label for="radio-bhk-1">1</label>
        <input type="radio" id="radio-bhk-2" name="uiBHK" value="2" checked/>
        <label for="radio-bhk-2">2</label>
        <input type="radio" id="radio-bhk-3" name="uiBHK" value="3"/>
        <label for="radio-bhk-3">3</label>
        <input type="radio" id="radio-bhk-4" name="uiBHK" value="4"/>
        <label for="radio-bhk-4">4</label>
        <input type="radio" id="radio-bhk-5" name="uiBHK" value="5"/>
        <label for="radio-bhk-5">5</label>
    </div>
    </form>
<form class="form">
```

Figure 101: “app.html” file

Next, I defined the “location” class into “app.css” file. And I typed the code as shown in GitHub for “app.css” file. In the css file, for example, the “app.css” file consists the UI elements like background colour, font, outline, border, margins, padding etc. The (Figure 102) shows the following.

```css
  border-radius: 5px;
}

.location{
  font-family: "Roboto", sans-serif;
  outline: 0;
  background: #f2f2f2;
  width: 76%;
  border: 0;
  margin: 0 0 10px;
  padding: 10px;
  box-sizing: border-box;
  font-size: 15px;
  height: 40px;
  border-radius: 5px;
}

.submit{
  background: #a5dc86;
  width: 76%;
  border: 0;
  margin: 25px 0 10px;
  box-sizing: border-box;
  font-size: 15px;
    height: 35px;
    text-align: center;
    border-radius: 5px;
}
```

Figure 102: “app.css” file

The function which I wrote on “app.js” file, i.e.: onPageLoad() load all the locations available, when the page is loaded. Next, I used the URL, which I already tested in the “Postman” Application. This is shown in (Figure 103).

```javascript
        bath: bathrooms,
        location: location.value
    },function(data, status) {
        console.log(data.estimated_price);
        estPrice.innerHTML = "<h2>" + data.estimated_price.toString() + " Lakh</h2>";
        console.log(status);
    });
}

function onPageLoad() {
    console.log( "document loaded" );
    var url = "http://127.0.0.1:5000/get_location_names";
    $.get(url,function(data, status) {
        console.log("got response for get_location_names request");
        if(data) {
            var locations = data.locations;
            var uiLocations = document.getElementById("uiLocations");
            $('#uiLocations').empty();
            for(var i in locations) {
                var opt = new Option(locations[i]);
                $('#uiLocations').append(opt);
            }
        }
    });
}

window.onload = onPageLoad;
```

Figure 103: “app.js” file

Next, I implemented a function for the button named as “Estimate Price”. This javascript function is implemented with onClickedEstimatePrice() function. Inside this function I wrote some more functions in order to get the values from the UI. Such as, value for bath, BHK etc. For this, the URL I used to be, I already tested in the Postman Application. This is a POST call. (Figure 104) shows it.

```javascript
        }
    }
    return -1; // Invalid Value
}

function onClickedEstimatePrice() {
    console.log("Estimate price button clicked");
    var sqft = document.getElementById("uiSqft");
    var bhk = getBHKValue();
    var bathrooms = getBathValue();
    var location = document.getElementById("uiLocations");
    var estPrice = document.getElementById("uiEstimatedPrice");

    var url = "http://127.0.0.1:5000/predict_home_price";


    $.post(url, {
        total_sqft: parseFloat(sqft.value),
        bhk: bhk,
        bath: bathrooms,
        location: location.value
    },function(data, status) {
        console.log(data.estimated_price);
        estPrice.innerHTML = "<h2>" + data.estimated_price.toString() + " Lakh</h2>";
        console.log(status);
    });
}
```

Figure 104: onClickedEstimatePrice() in “app.js” file

All the codes including for the Machine Learning Model as well as the code written for web application/ web page development are presented in GitHub.

Then, finally I tested my project in the Localhost. For this, In PyCharm first I ran the “server.py” in order to up the server. Next, I ran the “app.html” which I wrote in Pycharm earlier.

Rather than this, I also imported the Python Flask Server as well as I installed the Python (All these steps are done before).

Finally, in order to run the website, I clicked the “Google Chrome” icon which is presented in the right hand corner side, in the “app.html”. (Figure 105) shows it.

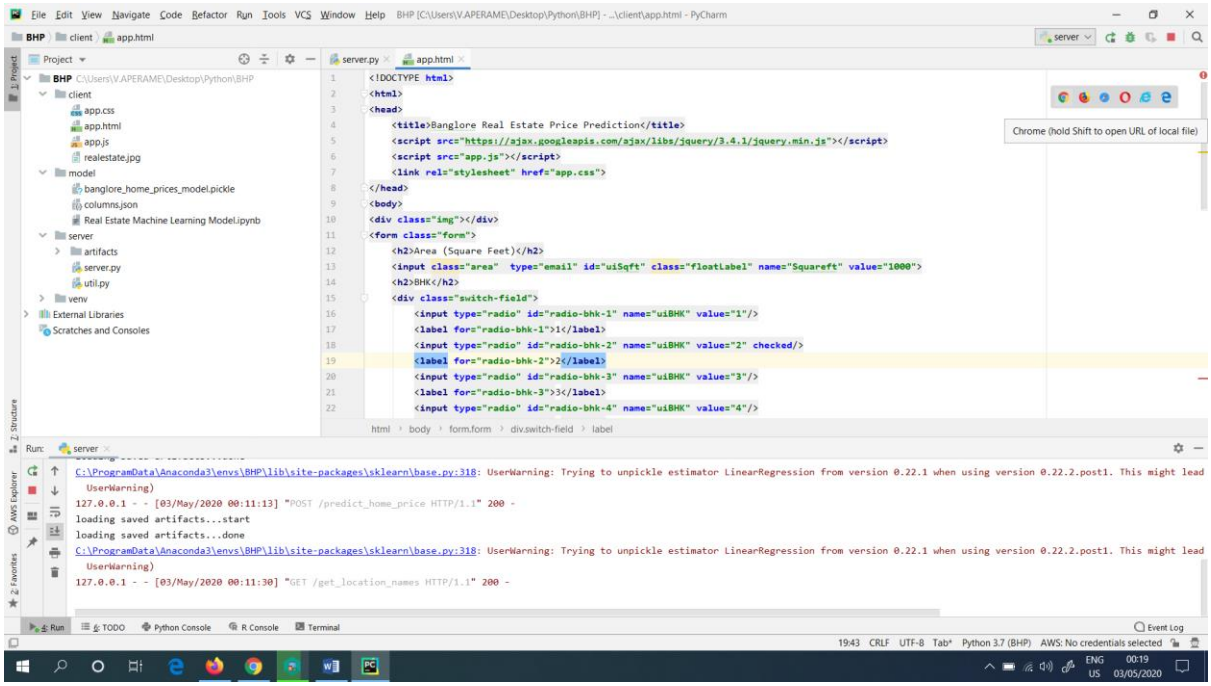


Figure 105: Clicked the Google Chrome icon in “app.html” file

OR

There is another method (Second Method) presented, in order to run the project. This is explained below.

Directly I went to the “app.html” file, which I saved it in Desktop, and I double clicked it in the file. It also prompted me the similar website which I got as shown below. This is shown in (Figure 106). This is directly hosted in the local host of the PC (Personal Computer).

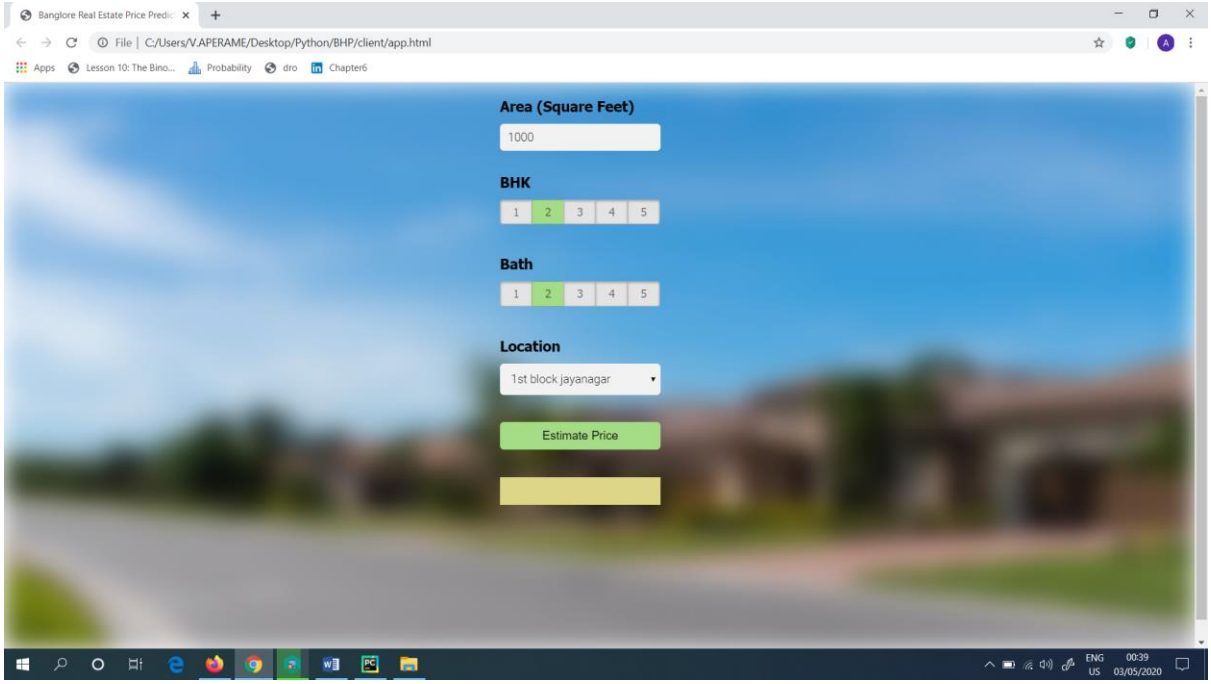


Figure 106: Got the same website by double clicking the “app.html” file

Next, when I continued from the first method which I showed above, suddenly, in the Local host of PyCharm, the website which I created was get hosted as shown in (Figure 107). This is hosted with the help of the PyCharm.

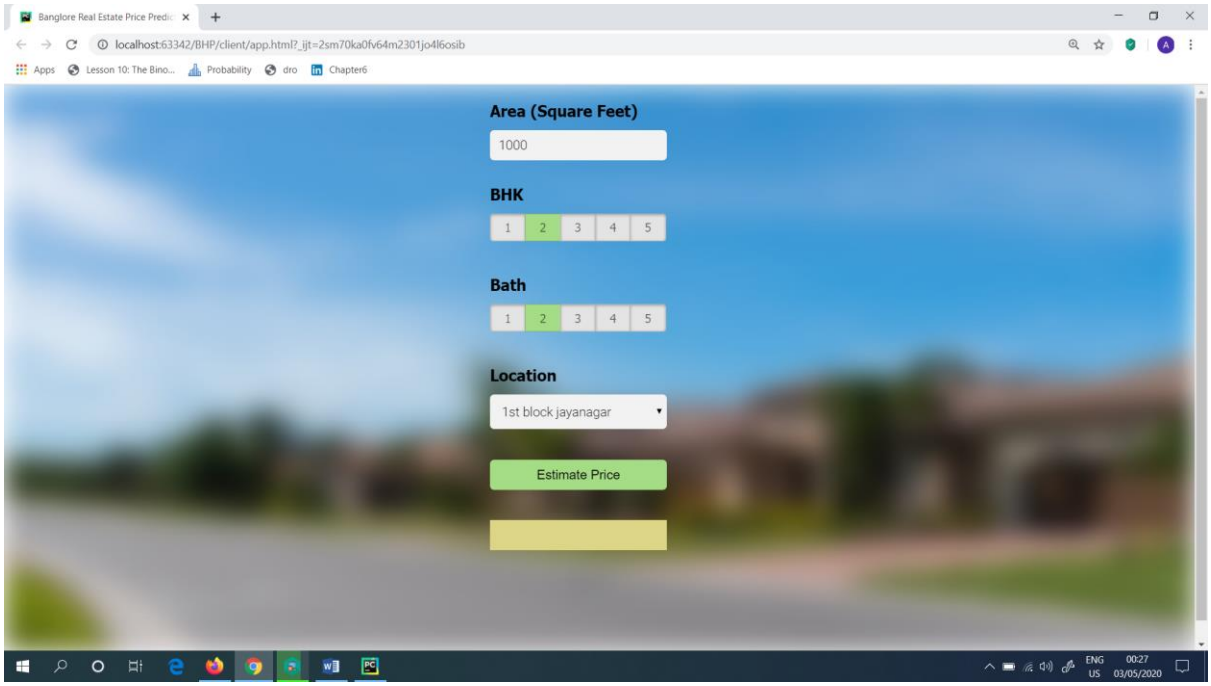


Figure 107: Website hosted in the localhost

Then, I thought to check the model, whether it is working fine or not. So, as the input for the website I gave the following as shown in (Figure 108) and it returned the estimated price as 219.14 Indian Lakh Rupees.

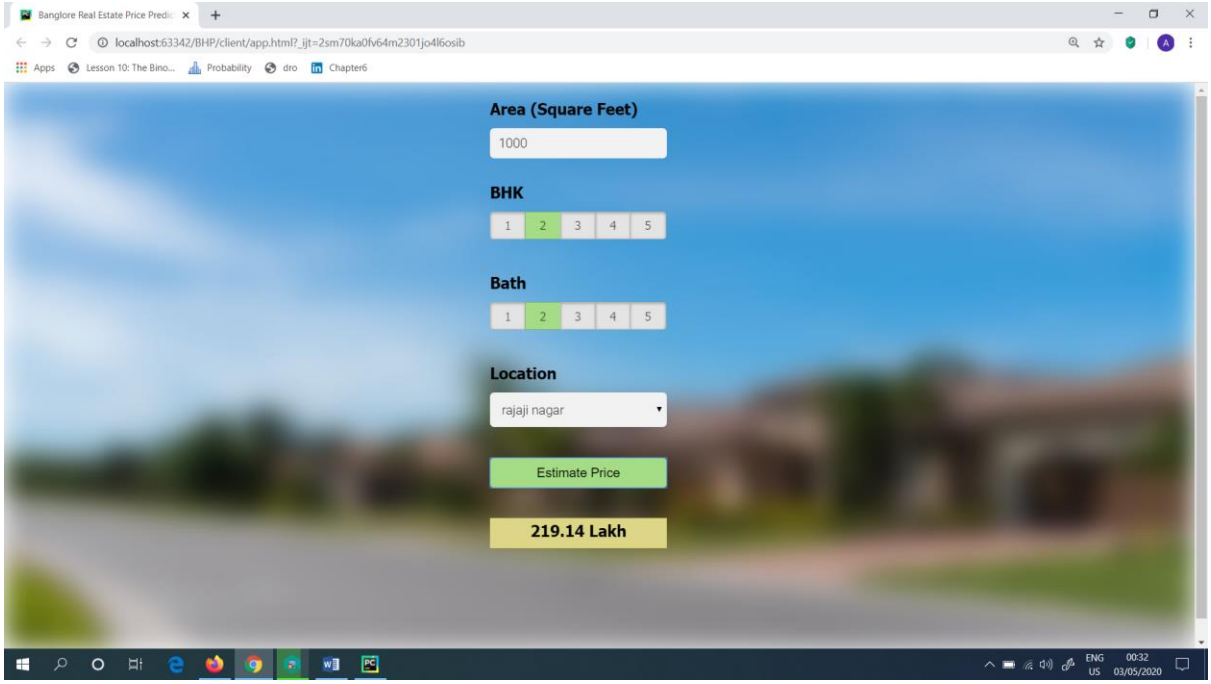


Figure 108: Got the estimated price

Next, I also tried the Estimated Price by giving different inputs. The model was worked fine and right without any errors. The Estimated Price got varied according to the user inputs as well as the Location.

Steps Followed (Summary)

1. Import the Data
2. Clean the Data
3. Split the Data into Training/ Test Sets
4. Create a Model
5. Train the Model
6. Make Predictions
7. Evaluate and Improve

Advantages of the product produced

- This product reduces the paper work dramatically.
- Required output given by the product is more accurate.
- The results are produced with the less amount of time.
- The data sets can be updated easily.
- The results are produced relevantly for the search.
- The product can be accessed from anywhere in the world, without the help of the manual records.
- As a result, data is logically structured, securely stored, not replicated, easy to retrieve and analyse.
- The data can be easily sort out, remove the unwanted data and can be updated according to the needs.
- With the help of the data pattern produced which are drawn from historical sales records, it helps to identify the most demanded property types.

- The information produced are useful for salespeople and brokers.
- There is very low possibility of error when handling data sets.
- Uses a small space to keep records.
- Open Source.
- Work repeatedly without getting bored or tired.
- Reduced man power with the help of the product.

References

- https://azati.ai/artificial-intelligence-and-machine-learning-for-real-estate/
- https://www.realestate.bnpparibas.com/how-data-science-transforming-real-estate
- https://towardsdatascience.com/my-capstone-project-real-estate-prices-venues-data-analysis-of-london-c936c0bc4b1d
- https://www.youtube.com/watch?v=_uQrJ0TkZlc
- https://www.youtube.com/watch?v=juldrxDrSH0&list=PLhW3qG5bs-L-oT0GenwPLcJAPD_SiFK3C
- https://www.youtube.com/watch?v=rnaa5jl8pVg
- https://www.youtube.com/watch?v=5mDYijMfSzs&t=498s
- https://www.youtube.com/watch?v=otmWEEFysms&t=30s