CSE4020 Machine Learning

J Component

Slot: F1/L37+L38

Faculty: prof. Balamurugan R

Topic

House Price Prediction Model

Team

Bukkasamudram Maheedhar Reddy (19BCE0121)

Vadarevu Hemanth Sai Sri Harsha (19BCE2237)

TABLE OF CONTENTS

ABSTRACT :

Machine learning plays a major role in spam reorganization, image detection and also in speech to text transformation from the past few years and now the advanced machine learning algorithms are used to build prediction models. The prediction models are used in various fields like medical and commerce to provide the end users with a better understanding on how certain steps taken could result in a better overall life whether it might be health or wealth.

In this project we are trying to build a house price prediction model which uses certain factors like the location, size, the society it is a part of and also the number of bathrooms and balconies present. What are the things that a potential home buyer considers before purchasing a house? The location, the size of the property, vicinity to offices, schools, parks, restaurants, hospitals or the stereotypical white picket fence? What about the most important factor — the price? Now with the lingering impact of demonetization, the enforcement of the Real Estate (Regulation and Development) Act (RERA), and the lack of trust in property developers in the city, housing units sold across India in 2017 dropped by 7 percent. Addition to this, Bangalore is the silicon valley of India and its growing economy and diverse culture makes its house price prediction difficult. We took this as a challenge and used a large dataset with various attributes to achieve the maximum accuracy for our model.

1. INTRODUCTION-

Machine learning develops algorithms and builds models from data, and uses them to predict on new data. The main difference with traditional algorithm is that a model is built from inputs data rather than just execute a series of instructions. Supervised learning uses data with result labeled, while unsupervised learning using unlabeled data. There are a few common machine learning algorithms, such as regression, classification, neural network and deep learning. Machine learning plays a major role from past years in image detection, spam reorganization, normal speech command, product recommendation and medical diagnosis.

1.1. Aim-

The aim of this project is to explore different machine learning algorithms and build a house price prediction model with the highest accuracy algorithm.

House price prediction is about building a model using machine learning algorithms which predicts future house prices based on the features like area type, location, size, total square feet, availability, bath, balcony etc.

1.2. Purpose-

House price prediction are expected to help people who plan to buy a house so they can know the price range in the future, then they can plan their finance well. In addition, house price predictions are also beneficial for property investors to know the trend of housing prices in a certain location.

It provides a description of prediction prices, and also the current prices which are useful in understanding the market which helps in making useful predictions. Thus, there is a need to predict the efficient house pricing for real estate customers with respect to their budgets and priorities.

2. LITERATURE REVIEW-

Most of the literature study we had gone through were comparing different machine learning algorithms suitable for the house price prediction model based on the accuracy. Here is the summary of few of those papers-

House Price Prediction by Ahmad Abdulal Nawar Aghi

Summary-

This study proposes a performance comparison between machine learning regression algorithms and Artificial Neural Network (ANN).

This study attempts to analyse the correlation between variables to determine the most important factors that affect house prices in Malmö, Sweden.

This thesis attempts to show that Lasso gives the best score among other algorithms when using the public dataset in training.

The accuracy of the prediction is evaluated by checking the root square and root mean square error scores of the training model.

Proposed Model and Results-

The regression algorithms used in this study are Multiple linear, Least Absolute Selection Operator (Lasso), Ridge, Random Forest.

It is shown that ANN differs from the other algorithms by having a worse prediction despite that it has the best RMSE score which proves that R2 score decreases when training the ANN model whereas regression algorithms show better accuracy in both R2 and RMSE.

The practical results show that Lasso is the most accurate algorithm among other algorithms, and it has the best performance after evaluating both RMSE and R2.

Issues-

The prediction accuracy of these algorithms depends heavily on the given data when training the model.

If the data is in bad shape, the model will be overfitted and inefficient, which means that data pre-processing is an important part of this experiment and will affect the final results.

Thus, multiple combinations of pre-processing methods need to be tested before getting the data ready to be used in training.

A Hybrid Regression Technique for House Prices Prediction by Sifei Lu, Zengxiang Li, Zheng Qin, Xulei Yang, Rick Siow Mong Goh

Summary-

This paper demonstrates how to make machine learning more useful in normal life by taking house price prediction as an example.

The proposed approach has recently been deployed as the key kernel for Kaggle Challenge "House Prices: Advanced Regression Techniques".

Proposed Model and Results-

With limited dataset and data features, a practical and composite data pre-processing, creative feature engineering method is examined in this paper.

The paper also proposes a hybrid, Lasso and Gradient boosting regression model to predict individual house price.

Based on the result, the hybrid regressions are better than one from Ridge, Lasso or Gradient boosting regression.

Lasso is very useful for feature selection, and one more benefit of removing useless features is it could decrease Ridge, Gradient boosting regression MSE result.

Improvements-

Random forest is an advanced regression algorithm; it may help to improve prediction accuracy.

If data are available, a good idea is to introduce more features, for example income, salary, population, local amenities, cost of living, annual property tax, school, crime, marketing data.

House Price Prediction Using Machine Learning by G. Naga Satish, Ch. V. Raghavendran, M.D.Sugnana Rao, Ch.Srinivasulu

Summary-

This study utilizes machine learning algorithms as a research method that develops housing price prediction models.

The paper discusses about the selection of prediction methods and comparing and exploring various prediction methods for the house price prediction.

Proposed Model and results-

The proposed models are simple linear regression, multiple regression, lasso regression and gradient boosting algorithm.

The examinations exhibit that lasso regression algorithm, in view of accuracy, reliably outperforms alternate models in the execution of housing cost prediction.

Issues-

The model depends on correctness of datasets.

Performing the computations sequentially, and utilizing various processors and parallel the computations involved, which might possibly decrease the preparation time and prediction period.

3. PROBLEM FORMULATION:

3.1. Problem Statement:

The major objective of the project is to find out how certain parameters regarding the house like the location it is located in, the size of the house, the availability, demand and etc. affects the price of the house.

By analysing the trend on how these parameters affect the price of the house, better knowledge about the pricing of the houses will be available to the potential tenants who can choose the perfect house for the perfect price.

To achieve the Machine Learning model to predict the price the problem is divided in the following way shown below:

(i) **Choose an adept dataset:** The Machine Learning model needs to be trained on the data for it to predict the final outcome based on the attributes given by the user. Thus, a dataset with enough attributes and data values trains the model better which in turn produces better outcomes. The dataset used for building the project is shown below:

https://www.kaggle.com/amitabhajoy/bengaluru-house-price-data?select=Bengaluru_House_Data.csv

The dataset contains 13320 instances and 9 attributes.

Attribute Information:

1. **Area_Type:** This attribute gives the information about the area in which is the house is there like whether it is in a built-up area or a plot area or a super built-up area etc.,.
2. **Availability:** This attributes lets the customer know when the house will be available to move-in.
3. **Location:** This attribute lets the client know about the exact location of the house.
4. **Size:** This attribute displays the size of the house whether it is a 2BHK or a 3BHK etc.
5. **Society:** This attribute gives information about the society in which the house is a part of.
6. **Total_sqft:** As the name suggests, this attributes informs about the total size of the house in sqft.
7. **#bath:** This attribute gives the client the number of bathrooms present in the house.
8. **#balcony:** This attribute informs the customer about the number of balconies present in the house.
9. **#price:** This attribute tells the price of the overall house.

(ii) **Explanatory Data Analysis:** The dataset needs to be studied in depth regarding the attribute types and how they can affect the price to choose the attributes on which the ML model needs to be trained and tested.

(iii) **Prepare the data for the ML Model:** Once, the analysis is done and the attributes to be considered for training the model are confirmed, the data needs to be cleaned and processed for it to be used by the ML model. For this, the null values or the missing values present are either dropped or filled, this process is known as the data pre-processing.

(iv) **Finding Outliers:** Outliers are data points that are far from other data points. In other words, they're unusual values in a dataset. Outliers are problematic for many statistical analyses because they can cause tests to either miss significant findings or distort real results. Thus finding these outliers in our dataset and removing them is necessary for a much accurate ML model.

(v) **Categorical Variable Encoding:** Not all variables or attributes need to be integers or numbers in general and hence the categorical variables need to be encoded into either numbers or integers for them to be trained on the model.

(vi) **ML model Training and Testing:** The Machine Learning model needs to be trained on the dataset and tested on the data to find out the accuracy measure of the model created and for the same reason the dataset needs to be split into training and testing data using feature scaling.

(vii) **Cross Validation:** Once the model created is trained and tested and accuracy measures are found out, we need to cross verify the training accuracy with respect to the testing accuracy to find whether the model is over-fitting, under-fitting or is normal.

(viii) **Hyper Parameter Tuning:** This is one of the important components in the process of building a ML model because it resolves the issue of either under-fitting or over-fitting by tuning certain parameters in the dataset.

(ix) **Testing the final model:** Once the model is done tuning, we can test the model by giving certain attribute values of our own to test whether the model can predict the final output or not. Once the testing is done we can save and load the model using the joblib library as a .pkl file.

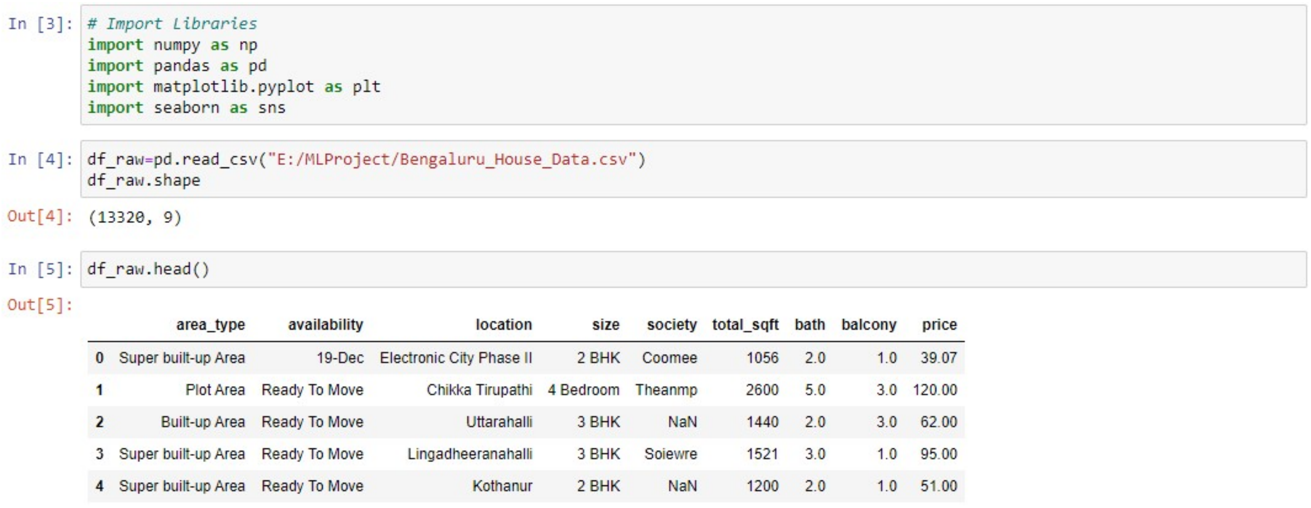
(x) **Model Deployment:** Finally when the model is created and is certified accurate enough, we need to deploy the model for normal end users to use the ML model for their necessary predictions. Hence, a webpage needs to be created and using the Flask server of the Python, we can deploy the ML model created onto the webpage created using the Flask server.

4. RESULTS AND DISCUSSION:-

4.1. Exploratory Data analysis:

For the Data analysis, we need to import necessary libraries of python such as pandas to read and describe the dataset used and matplotlib for visualizing the analysis.

We can read the dataset in the csv form using the pd.read_csv command and when we use the shape command on the dataset it defines the size of the dataset being used like the number of rows and columns. Finally, we can use the .head() command to see certain number of data values if the actual dataset to get a brief idea of how the data is distributed and their types.



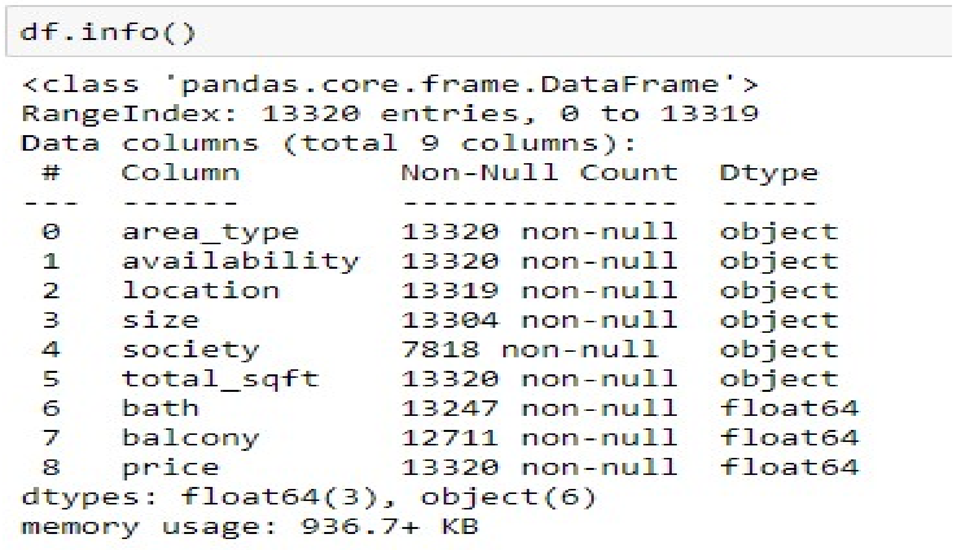
```
In [3]: # Import Libraries
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
```

```
In [4]: df_raw=pd.read_csv("E:/MLProject/Bengaluru_House_Data.csv")
        df_raw.shape

Out[4]: (13320, 9)
```

```
In [5]: df_raw.head()

Out[5]:
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00

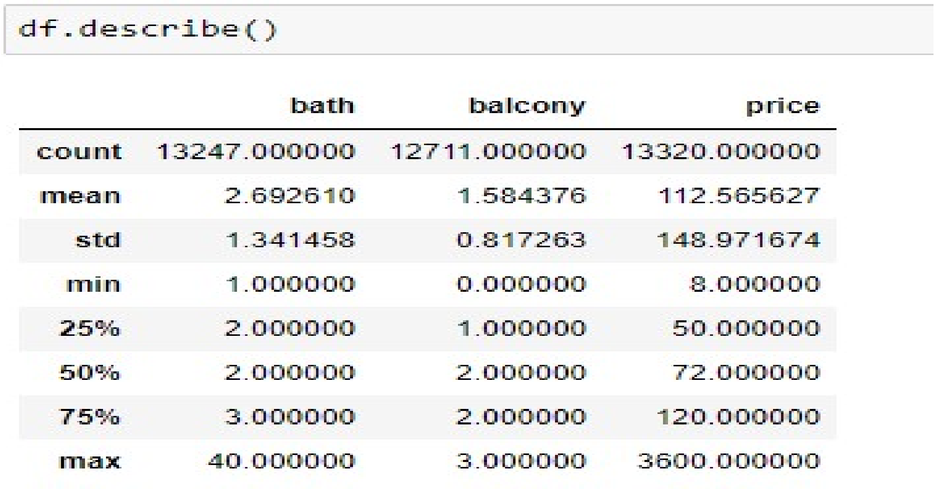
We can use the info() command to get the attribute names along with their types to get the basic information about the attributes of the dataset.



```
df.info()
```

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

The describe command gives the information about the mean, median, count, variance, standard deviation and also the maximum and minimum values so that we can find the outliers.

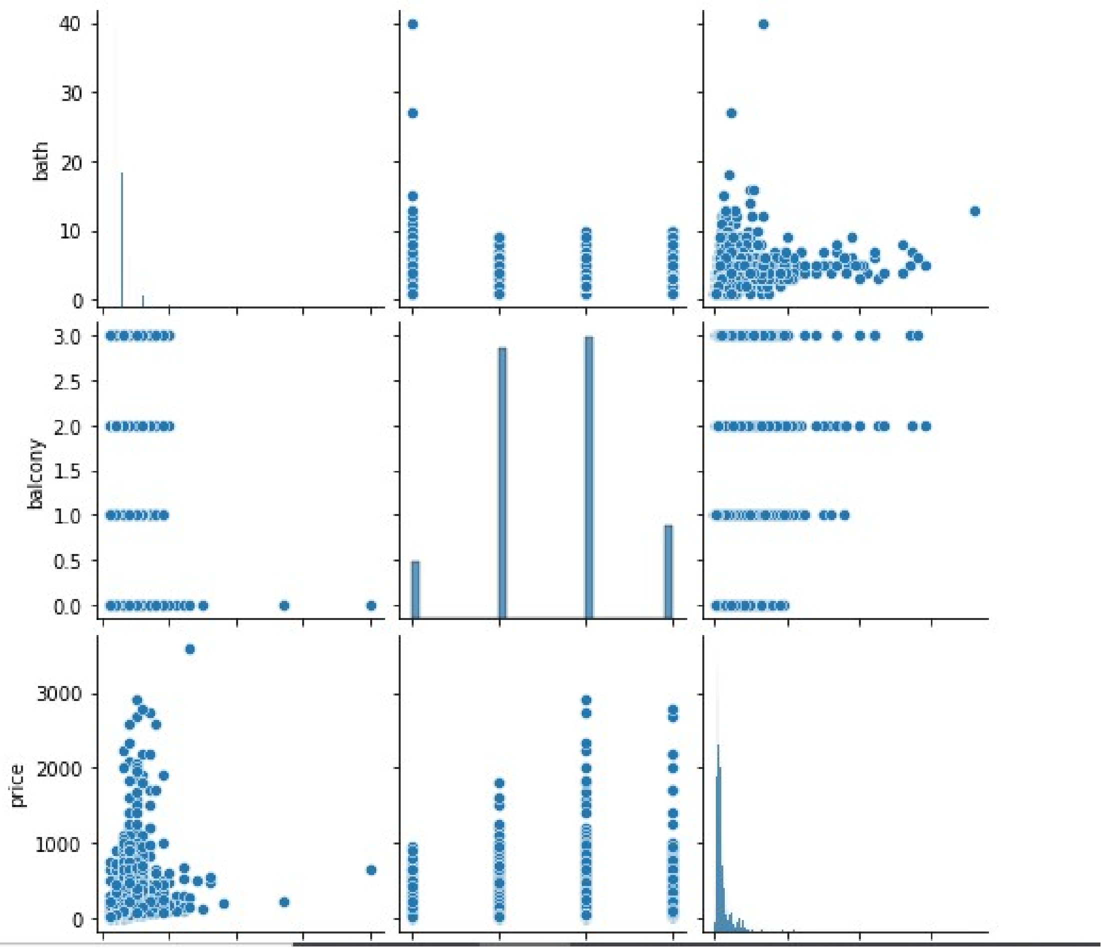


```
df.describe()
```

	bath	balcony	price
count	13247.000000	12711.000000	13320.000000
mean	2.692610	1.584376	112.565627
std	1.341458	0.817263	148.971674
min	1.000000	0.000000	8.000000
25%	2.000000	1.000000	50.000000
50%	2.000000	2.000000	72.000000
75%	3.000000	2.000000	120.000000
max	40.000000	3.000000	3600.000000

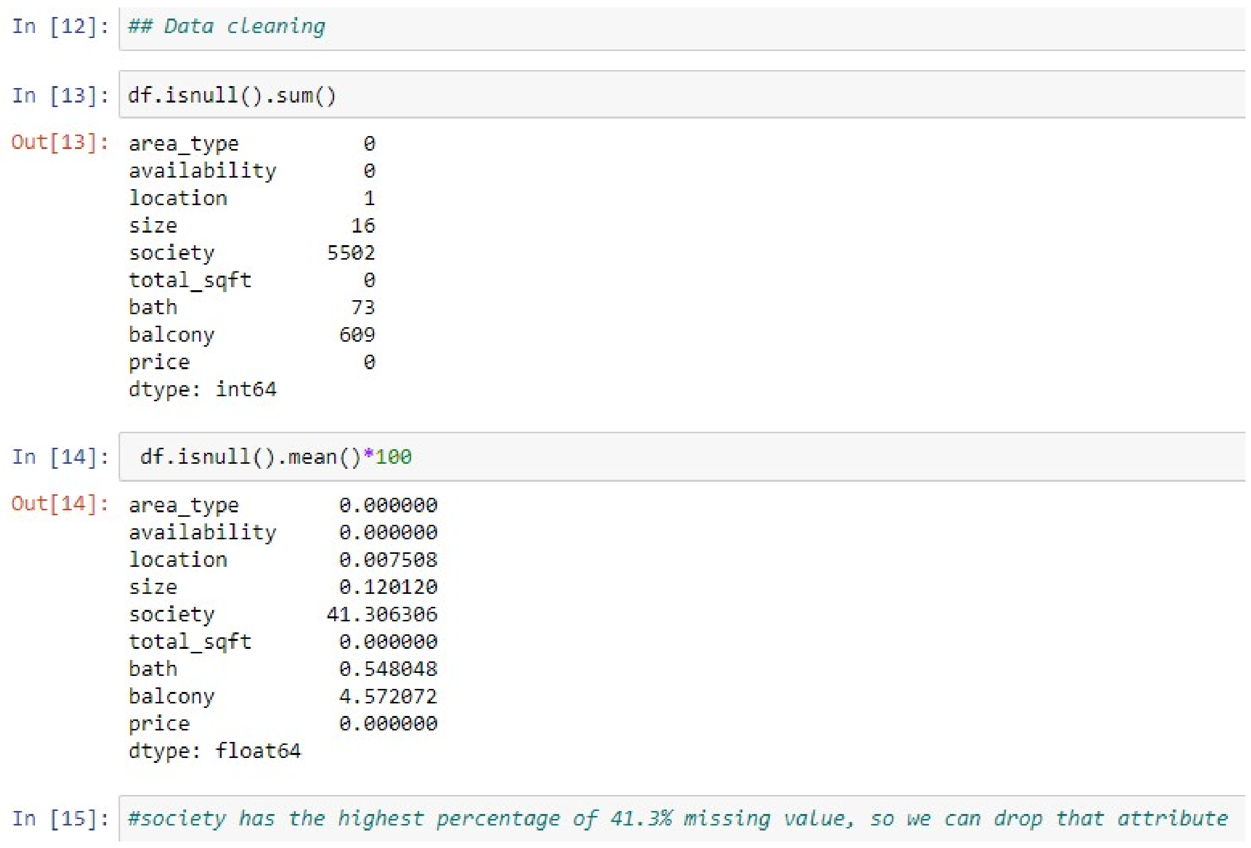
We can use the seaborn library of python to plot the pairplot of all the attributes in the dataset and the points which are way away from the major data points can be taken as outliers.

```
Out[10]: <seaborn.axisgrid.PairGrid at 0x2249e759d00>
```



4.2. Prepare the Data for the ML model:

When the isnull().sum() command is given it gives the number of missing values in each of the attribute present in the dataset.



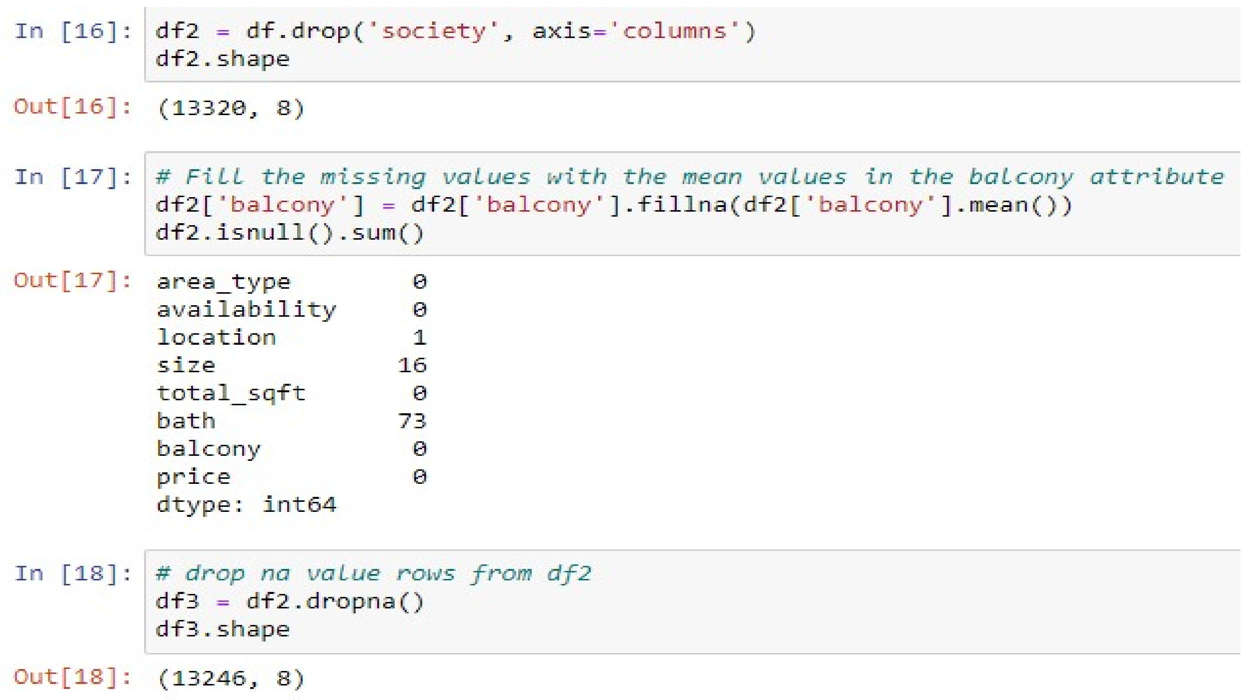
```
In [12]: ## Data cleaning

In [13]: df.isnull().sum()

Out[13]: area_type          0
         availability       0
         location           1
         size              16
         society         5502
         total_sqft         0
         bath              73
         balcony          609
         price              0
         dtype: int64

In [14]:  df.isnull().mean()*100

Out[14]: area_type        0.000000
         availability     0.000000
         location         0.007508
         size             0.120120
         society         41.306306
         total_sqft       0.000000
         bath             0.548048
         balcony          4.572072
         price            0.000000
         dtype: float64

In [15]: #society has the highest percentage of 41.3% missing value, so we can drop that attribute
```

When converted to percentages, the society has a 41.3% rate of missing values and hence it can be dropped.

Once the attribute 'society' is dropped from the data frame, the other attributes with missing values either need to be dropped or filled with mean or median values.

Since the attribute 'balcony' has 4.57% of missing values, the values are filled with mean values of the remaining data.

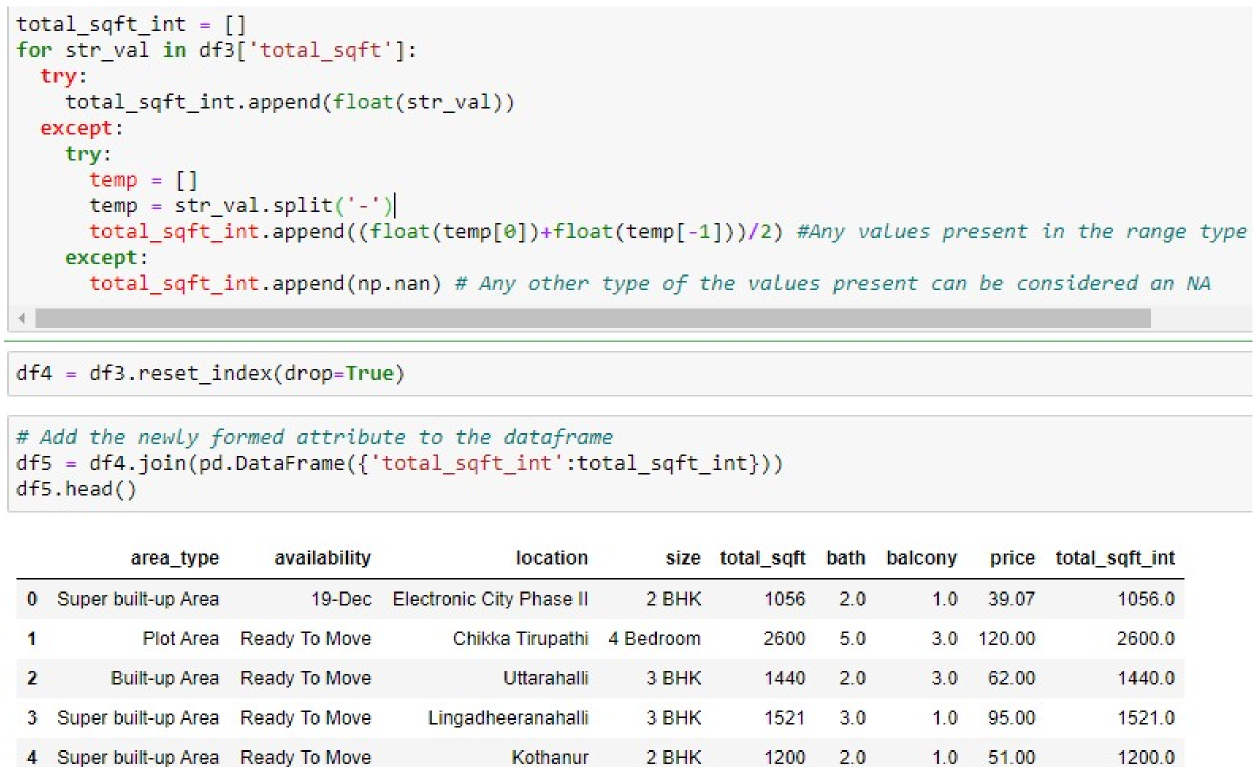
Once the missing values are filled, we check the number of missing values again and if the number is very small compared to the total size of the dataset used, we can drop the remaining missing values and this concludes the process of pre-processing of data.



```
In [16]: df2 = df.drop('society', axis='columns')
         df2.shape

Out[16]: (13320, 8)

In [17]: # Fill the missing values with the mean values in the balcony attribute
         df2['balcony'] = df2['balcony'].fillna(df2['balcony'].mean())
         df2.isnull().sum()

Out[17]: area_type        0
         availability     0
         location         1
         size            16
         total_sqft       0
         bath            73
         balcony          0
         price            0
         dtype: int64

In [18]: # drop na value rows from df2
         df3 = df2.dropna()
         df3.shape

Out[18]: (13246, 8)
```

Feature Engineering:

Once the attribute data values are studied, we can find that the data under total_sqft and size attributes have different formats in which they are present like integer, range values or a combination and thus all these data values need to be converted to integer to train the model.



```
total_sqft_int = []
for str_val in df3['total_sqft']:
  try:
    total_sqft_int.append(float(str_val))
  except:
    try:
      temp = []
      temp = str_val.split('-')
      total_sqft_int.append((float(temp[0])+float(temp[-1]))/2) #Any values present in the range type
    except:
      total_sqft_int.append(np.nan) # Any other type of the values present can be considered an NA
```

```
df4 = df3.reset_index(drop=True)
```

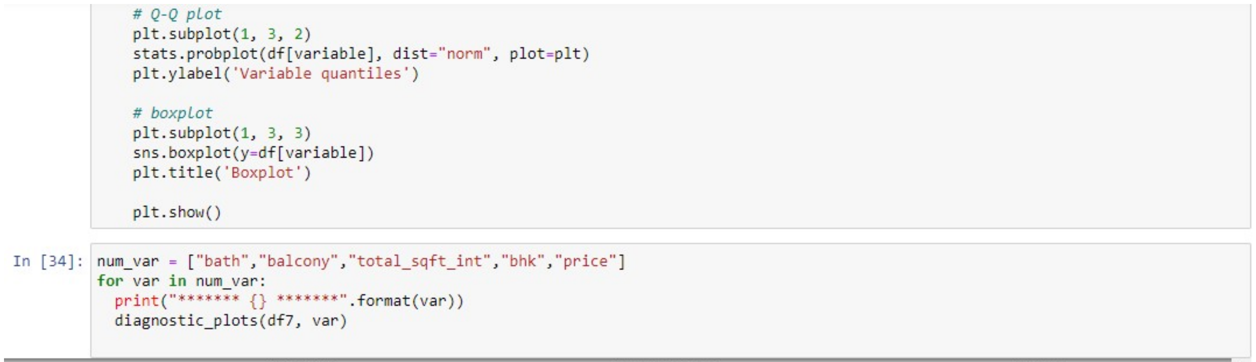
```
# Add the newly formed attribute to the dataframe
df5 = df4.join(pd.DataFrame({'total_sqft_int':total_sqft_int}))
df5.head()
```

	area_type	availability	location	size	total_sqft	bath	balcony	price	total_sqft_int
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	1056	2.0	1.0	39.07	1056.0
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	2600	5.0	3.0	120.00	2600.0
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	1440	2.0	3.0	62.00	1440.0
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	1521	3.0	1.0	95.00	1521.0
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	1200	2.0	1.0	51.00	1200.0

4.3. Finding Outliers and Removal:

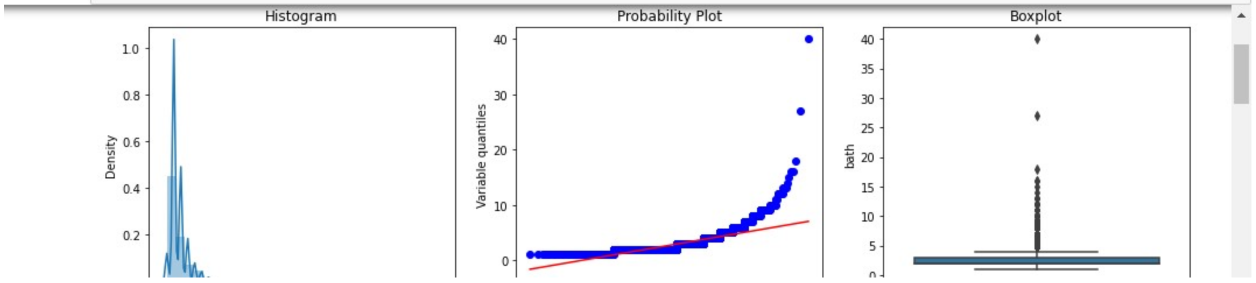
For all the attributes, graphs like histogram, box plots and probability plots are drawn to find out the outliers and these outliers are removed as these are unusual values in the dataset.

These plots can be drawn using seaborn and the Q-Q plot is drawn using the stats from scipy library.



```
    # Q-Q plot
    plt.subplot(1, 3, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    plt.ylabel('Variable quantiles')

    # boxplot
    plt.subplot(1, 3, 3)
    sns.boxplot(y=df[variable])
    plt.title('Boxplot')

    plt.show()
```

```
In [34]: num_var = ["bath","balcony","total_sqft_int","bhk","price"]
         for var in num_var:
           print("******* {} *******".format(var))
           diagnostic_plots(df7, var)
```



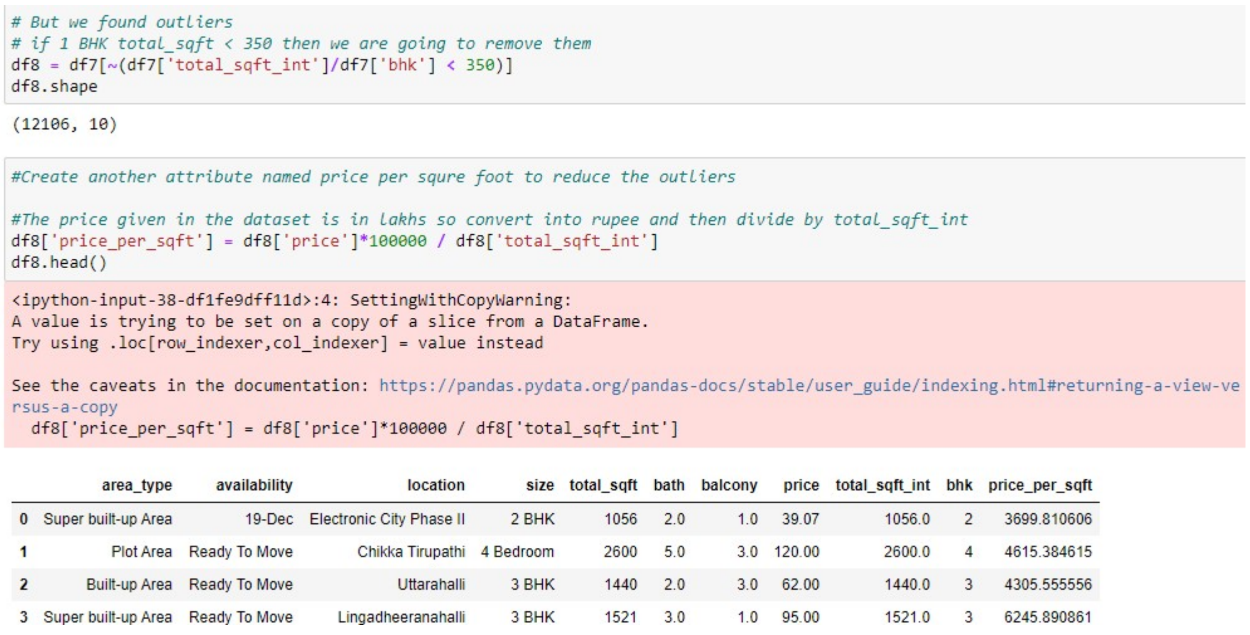
From studying these graphs, we can set parameters to remove these outliers or to not consider these outliers and one such parameter can be to have a minimum of 350sqft for 1BHK houses.

But even after putting this condition into action there were still outliers in the data frame and hence they are dropped.

Now a new attribute called price_per_sqft is created to minimize the outliers in both the attributes price and total_sqft.

After the attribute is created if any outliers are found they are removed by dropping all the outlier values.

After this a scatter plot was drawn and in locations where the price of a 2BHK house is greater than that of a 3BHK house are considered outliers and these are dropped.



```
# But we found outliers
# if 1 BHK total_sqft < 350 then we are going to remove them
df8 = df7[~(df7['total_sqft_int']/df7['bhk'] < 350)]
df8.shape
```

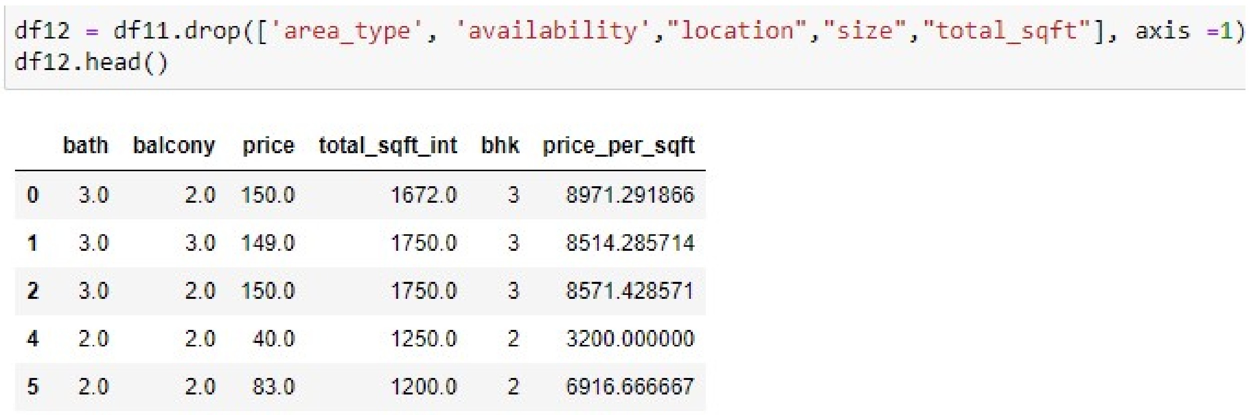
```
(12106, 10)
```

```
#Create another attribute named price per squre foot to reduce the outliers

#The price given in the dataset is in Lakhs so convert into rupee and then divide by total_sqft_int
df8['price_per_sqft'] = df8['price']*100000 / df8['total_sqft_int']
df8.head()
```

```
<ipython-input-38-df1fe9dff11d>:4: 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-ve
rsus-a-copy
  df8['price_per_sqft'] = df8['price']*100000 / df8['total_sqft_int']
```

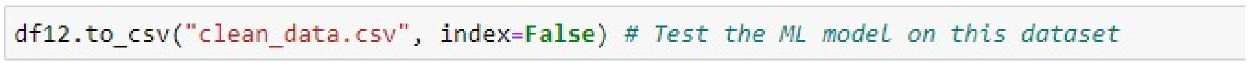
	area_type	availability	location	size	total_sqft	bath	balcony	price	total_sqft_int	bhk	price_per_sqft
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	1056	2.0	1.0	39.07	1056.0	2	3699.810606
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	2600	5.0	3.0	120.00	2600.0	4	4615.384615
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	1440	2.0	3.0	62.00	1440.0	3	4305.555556
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	1521	3.0	1.0	95.00	1521.0	3	6245.890861

Once all the outliers are removed the data is then copied into a new csv file named clean_data.csv.



```
df12 = df11.drop(['area_type', 'availability',"location","size","total_sqft"], axis =1)
df12.head()
```

	bath	balcony	price	total_sqft_int	bhk	price_per_sqft
0	3.0	2.0	150.0	1672.0	3	8971.291866
1	3.0	3.0	149.0	1750.0	3	8514.285714
2	3.0	2.0	150.0	1750.0	3	8571.428571
4	2.0	2.0	40.0	1250.0	2	3200.000000
5	2.0	2.0	83.0	1200.0	2	6916.666667



```
df12.to_csv("clean_data.csv", index=False) # Test the ML model on this dataset
```

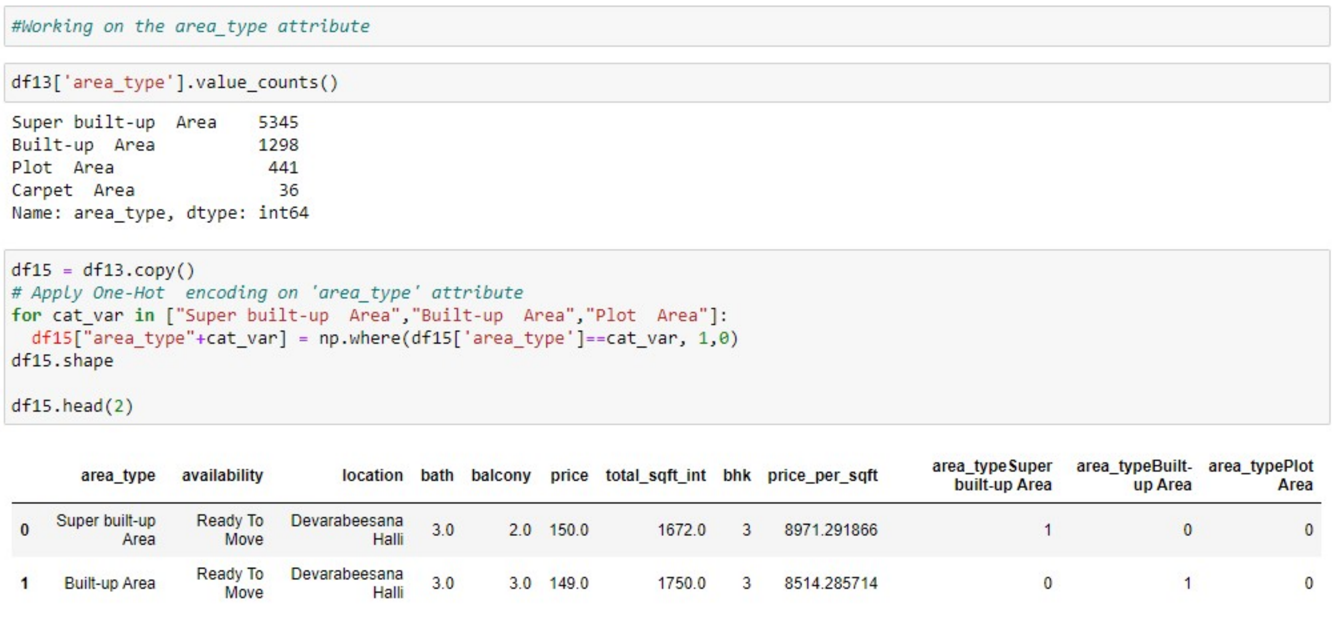
4.4. Categorical Encoding:

Even though all the outliers and the missing values are removed, there are still many categorical attributes which need to be encoded into numbers to train the model.

But attributes like ['area_type','availability','location'] contain multiple classes and if we try and convert them into ONE Encoding, it increases the size of data frame thus, we try to use those classes which are *frequently* present in the categorical variables.

We will encode these categorical variables one by one:

Area_type:



```
#Working on the area_type attribute
```

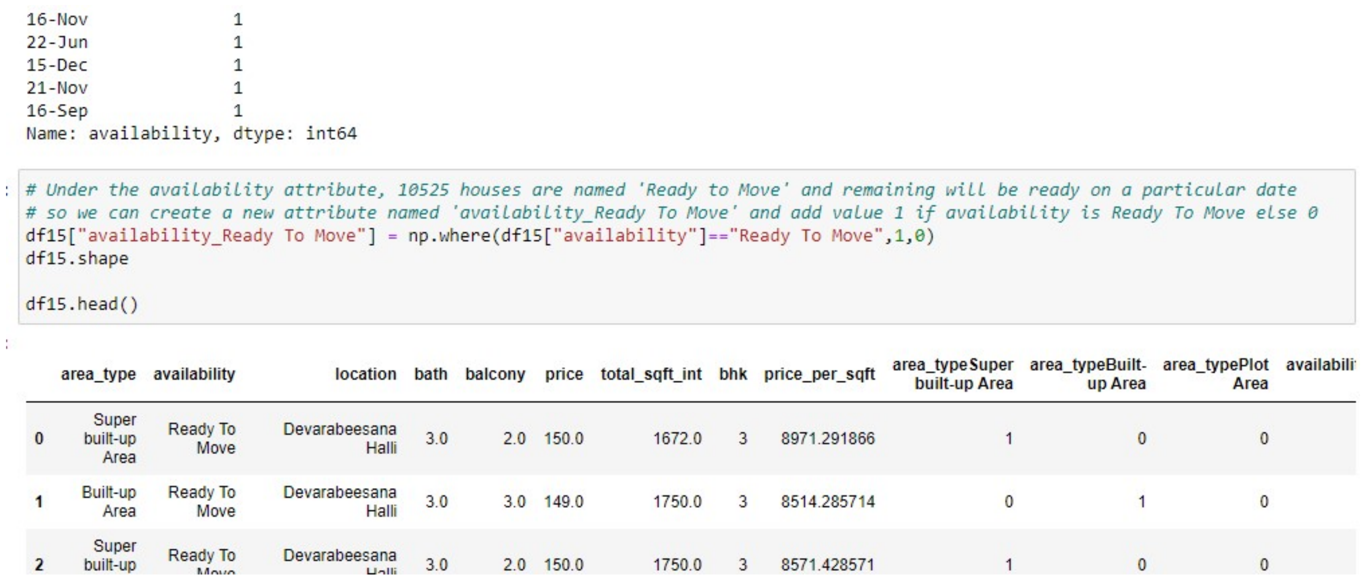
```
df13['area_type'].value_counts()
```

```
Super built-up  Area    5345
Built-up  Area          1298
Plot  Area               441
Carpet  Area              36
Name: area_type, dtype: int64
```

```
df15 = df13.copy()
# Apply One-Hot  encoding on 'area_type' attribute
for cat_var in ["Super built-up  Area","Built-up  Area","Plot  Area"]:
  df15["area_type"+cat_var] = np.where(df15['area_type']==cat_var, 1,0)
df15.shape

df15.head(2)
```

	area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_typeSuper built-up Area	area_typeBuilt-up Area	area_typePlot Area
0	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1672.0	3	8971.291866	1	0	0
1	Built-up Area	Ready To Move	Devarabeesana Halli	3.0	3.0	149.0	1750.0	3	8514.285714	0	1	0

Availability:

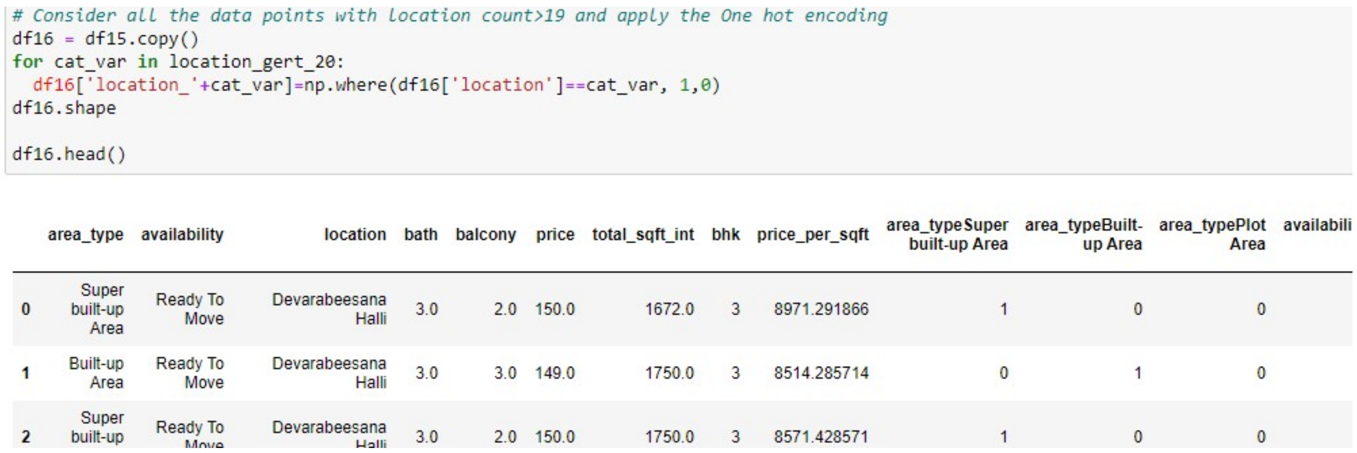


```
16-Nov           1
22-Jun           1
15-Dec           1
21-Nov           1
16-Sep           1
Name: availability, dtype: int64
```

```
# Under the availability attribute, 10525 houses are named 'Ready to Move' and remaining will be ready on a particular date
# so we can create a new attribute named 'availability_Ready To Move' and add value 1 if availability is Ready To Move else 0
df15["availability_Ready To Move"] = np.where(df15["availability"]=="Ready To Move",1,0)
df15.shape

df15.head()
```

	area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_typeSuper built-up Area	area_typeBuilt-up Area	area_typePlot Area	availabili
0	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1672.0	3	8971.291866	1	0	0	
1	Built-up Area	Ready To Move	Devarabeesana Halli	3.0	3.0	149.0	1750.0	3	8514.285714	0	1	0	
2	Super built-up	Ready To	Devarabeesana	3.0	2.0	150.0	1750.0	3	8571.428571	1	0	0	

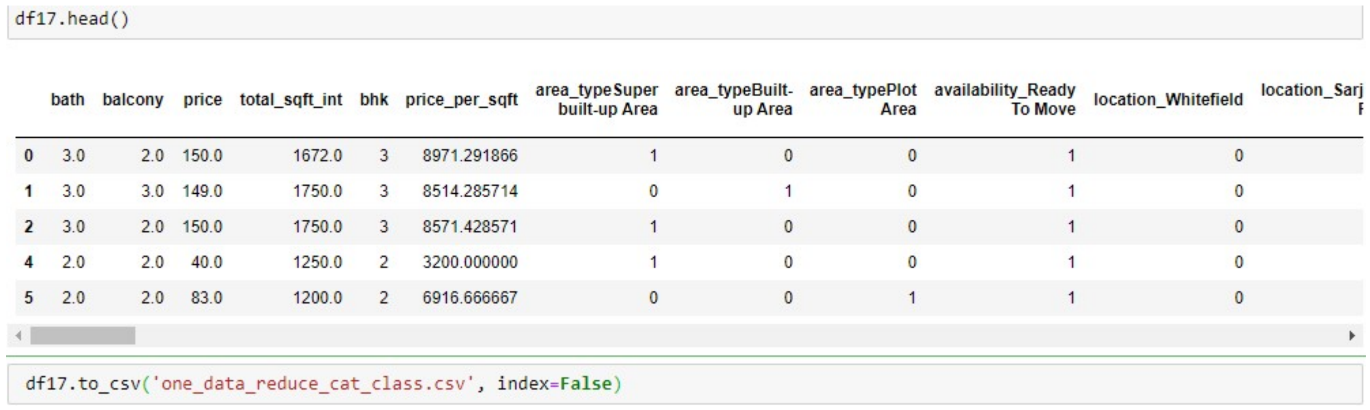
Location:



```
# Consider all the data points with location count>19 and apply the One hot encoding
df16 = df15.copy()
for cat_var in location_gert_20:
  df16['location_'+cat_var]=np.where(df16['location']==cat_var, 1,0)
df16.shape

df16.head()
```

	area_type	availability	location	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_typeSuper built-up Area	area_typeBuilt-up Area	area_typePlot Area	availabili
0	Super built-up Area	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1672.0	3	8971.291866	1	0	0	
1	Built-up Area	Ready To Move	Devarabeesana Halli	3.0	3.0	149.0	1750.0	3	8514.285714	0	1	0	
2	Super built-up	Ready To Move	Devarabeesana Halli	3.0	2.0	150.0	1750.0	3	8571.428571	1	0	0	

After encoding any other remaining categorical data is dropped and the data is saved in a csv file named one_data_reduce_cat_class.csv.

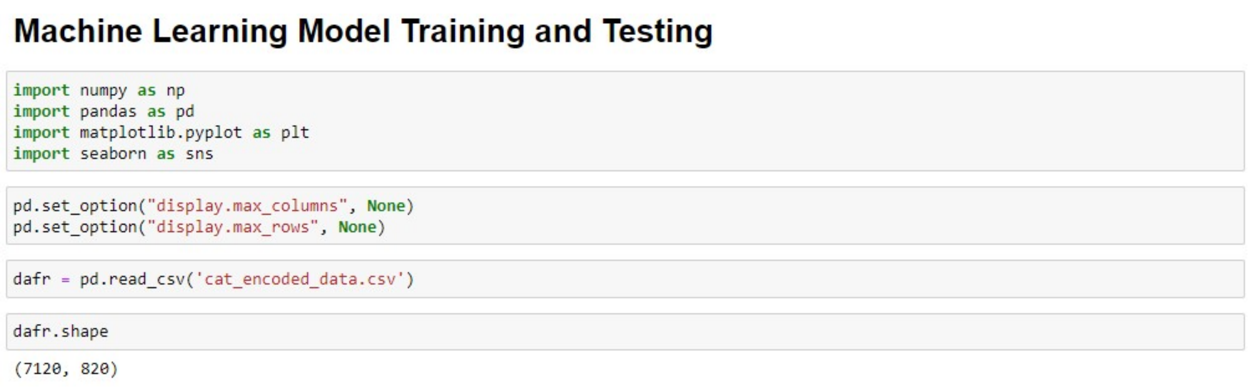


```
df17.head()
```

	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_typeSuper built-up Area	area_typeBuilt-up Area	area_typePlot Area	availability_Ready To Move	location_Whitefield	location_Sarj
0	3.0	2.0	150.0	1672.0	3	8971.291866	1	0	0	1	0	
1	3.0	3.0	149.0	1750.0	3	8514.285714	0	1	0	1	0	
2	3.0	2.0	150.0	1750.0	3	8571.428571	1	0	0	1	0	
4	2.0	2.0	40.0	1250.0	2	3200.000000	1	0	0	1	0	
5	2.0	2.0	83.0	1200.0	2	6916.666667	0	0	1	1	0	

```
df17.to_csv('one_data_reduce_cat_class.csv', index=False)
```

4.5. ML Model Training and Testing:



Machine Learning Model Training and Testing

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

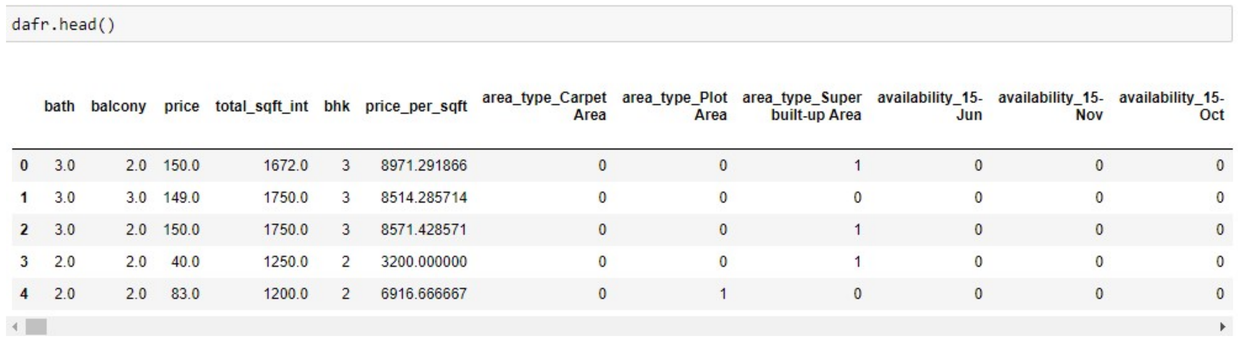
```
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
```

```
dafr = pd.read_csv('cat_encoded_data.csv')
```

```
dafr.shape
```

```
(7120, 820)
```

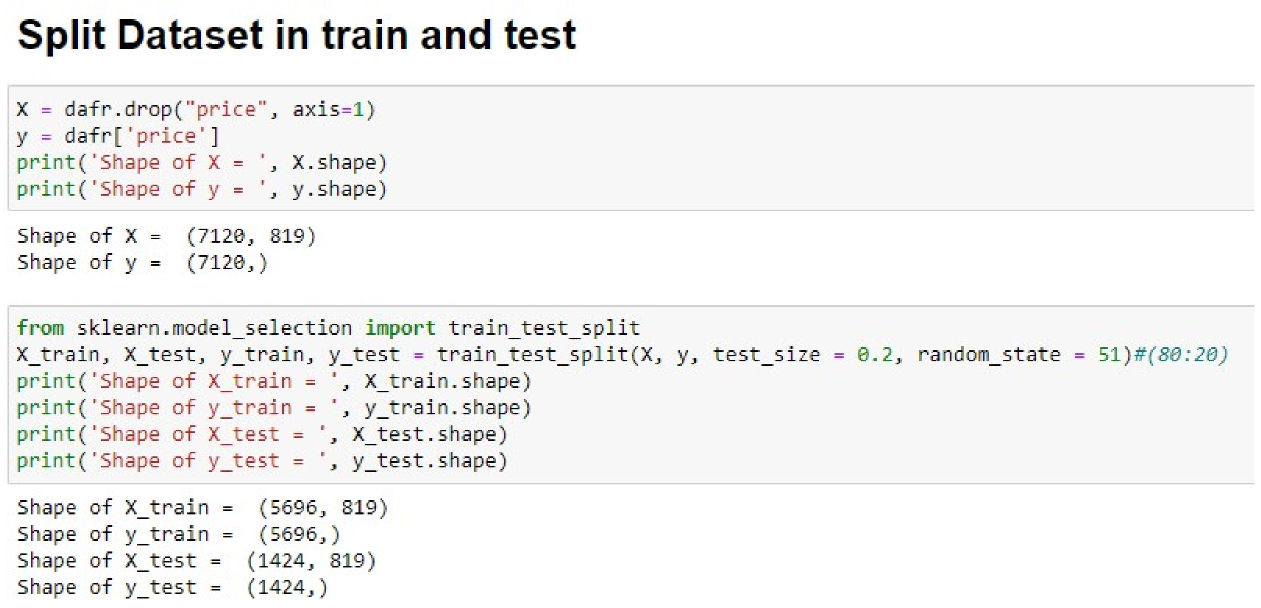
Now again all the required libraries are imported and the dataset created after the categorical encoding is read and used. A sample of the dataset used is shown below:



```
dafr.head()
```

	bath	balcony	price	total_sqft_int	bhk	price_per_sqft	area_type_Carpet Area	area_type_Plot Area	area_type_Super built-up Area	availability_15-Jun	availability_15-Nov	availability_15-Oct
0	3.0	2.0	150.0	1672.0	3	8971.291866	0	0	1	0	0	0
1	3.0	3.0	149.0	1750.0	3	8514.285714	0	0	0	0	0	0
2	3.0	2.0	150.0	1750.0	3	8571.428571	0	0	1	0	0	0
3	2.0	2.0	40.0	1250.0	2	3200.000000	0	0	1	0	0	0
4	2.0	2.0	83.0	1200.0	2	6916.666667	0	1	0	0	0	0

After the dataset is read it is split into training and testing data. Here we have split the data in 80:20 ratios.



Split Dataset in train and test

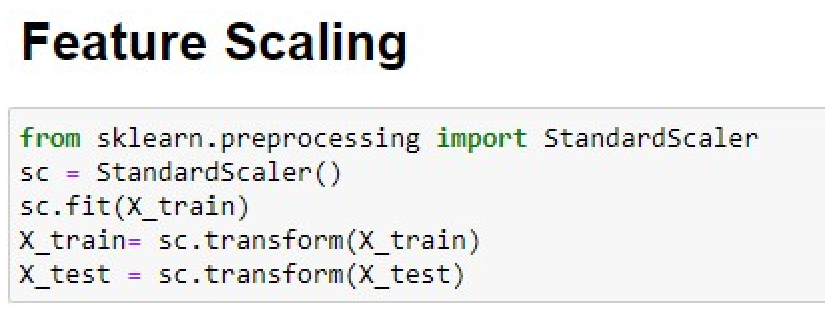
```
X = dafr.drop("price", axis=1)
y = dafr['price']
print('Shape of X = ', X.shape)
print('Shape of y = ', y.shape)
```

```
Shape of X =  (7120, 819)
Shape of y =  (7120,)
```

```
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 = 51)#(80:20)
print('Shape of X_train = ', X_train.shape)
print('Shape of y_train = ', y_train.shape)
print('Shape of X_test = ', X_test.shape)
print('Shape of y_test = ', y_test.shape)
```

```
Shape of X_train =  (5696, 819)
Shape of y_train =  (5696,)
Shape of X_test =  (1424, 819)
Shape of y_test =  (1424,)
```

After the dataset is split feature scaling is applied followed by training the model using various regression models.



Feature Scaling

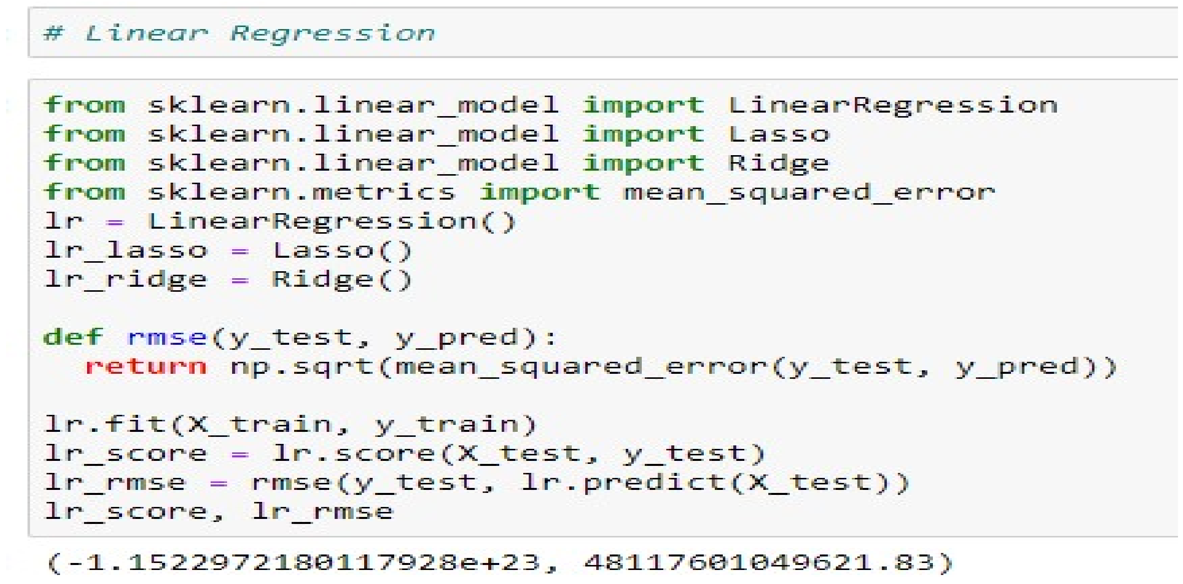
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train= sc.transform(X_train)
X_test = sc.transform(X_test)
```

We used many regression models on the split set and found the accuracy scores to find the best suitable regression equation or model.

Firstly, we used the concept of multiple linear regressions which is imported from the sklearn library of python and the error measure taken into consideration is root mean square.

Multiple Linear Regression:

Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable *x* is associated with a value of the dependent variable *y*. The population regression line for *p* explanatory variables $x_1, x_2, \ldots, x_p$ is defined to be $\mu_y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$.



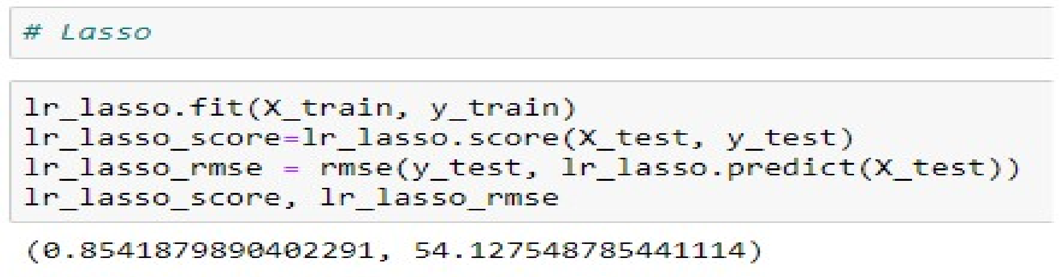
```
# Linear Regression
```

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
lr = LinearRegression()
lr_lasso = Lasso()
lr_ridge = Ridge()

def rmse(y_test, y_pred):
  return np.sqrt(mean_squared_error(y_test, y_pred))

lr.fit(X_train, y_train)
lr_score = lr.score(X_test, y_test)
lr_rmse = rmse(y_test, lr.predict(X_test))
lr_score, lr_rmse
```

```
(-1.1522972180117928e+23, 48117601049621.83)
```

Lasso Regression:

In statistics and machine learning, lasso is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the resulting statistical model. Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models.



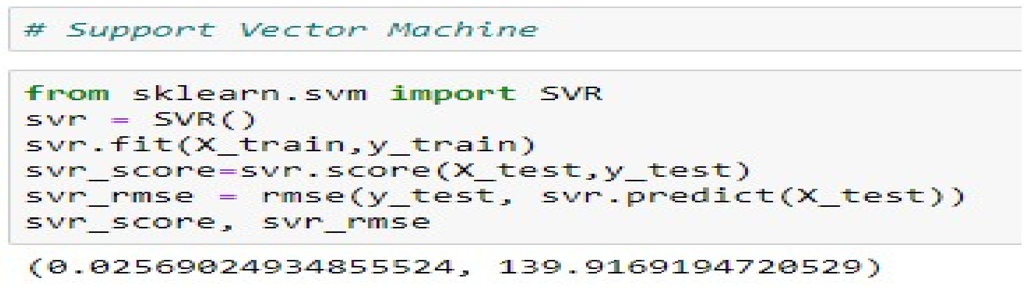
```
# Lasso
```

```
lr_lasso.fit(X_train, y_train)
lr_lasso_score=lr_lasso.score(X_test, y_test)
lr_lasso_rmse = rmse(y_test, lr_lasso.predict(X_test))
lr_lasso_score, lr_lasso_rmse
```

```
(0.8541879890402291, 54.127548785441114)
```

Support Vector Machine (Support Vector Regression):

Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem.



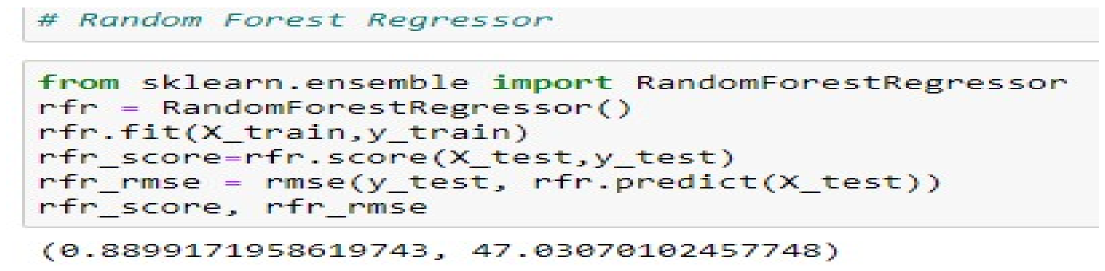
```
# Support Vector Machine
```

```
from sklearn.svm import SVR
svr = SVR()
svr.fit(X_train,y_train)
svr_score=svr.score(X_test,y_test)
svr_rmse = rmse(y_test, svr.predict(X_test))
svr_score, svr_rmse
```

```
(0.02569024934855524, 139.9169194720529)
```

Random Forest Regressor:

A random forest is a Meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.



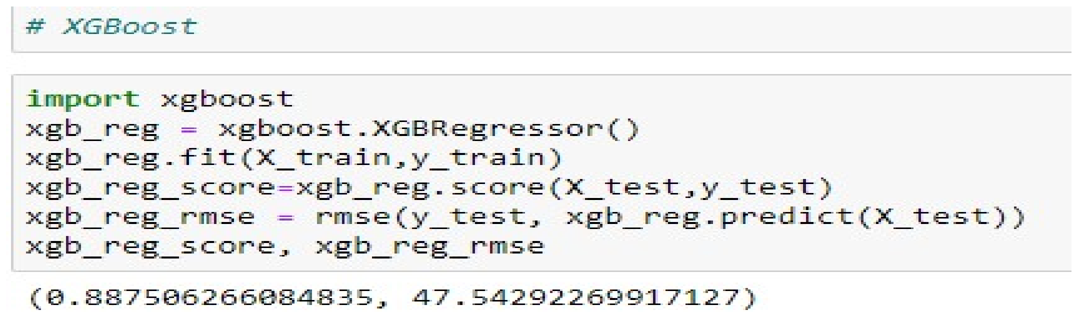
```
# Random Forest Regressor
```

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(X_train,y_train)
rfr_score=rfr.score(X_test,y_test)
rfr_rmse = rmse(y_test, rfr.predict(X_test))
rfr_score, rfr_rmse
```

```
(0.8899171958619743, 47.03070102457748)
```

XGBoost from Gradient Boost:

XGBoost is a powerful approach for building supervised regression models. Ensemble learning involves training and combining individual models (known as base learners) to get a single prediction and XGBoost is one of the ensemble learning methods.

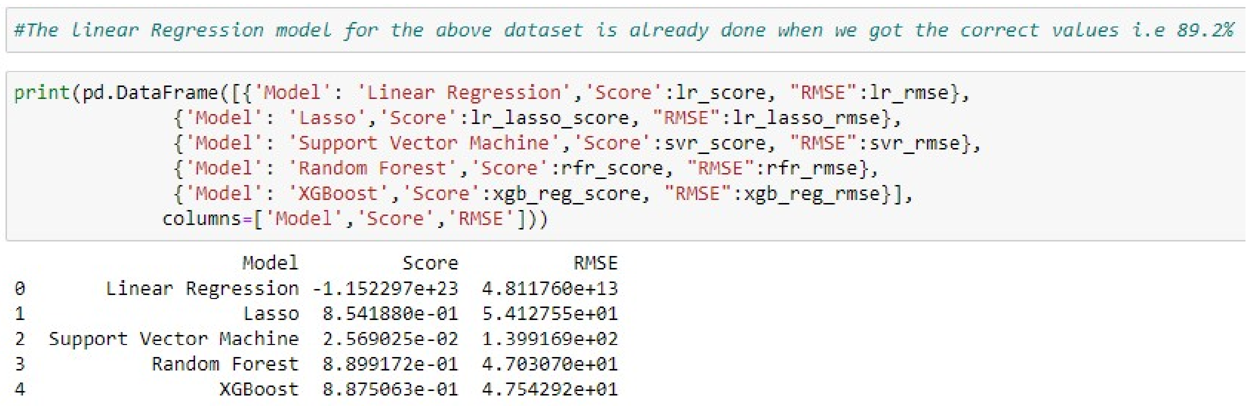


```
# XGBoost
```

```
import xgboost
xgb_reg = xgboost.XGBRegressor()
xgb_reg.fit(X_train,y_train)
xgb_reg_score=xgb_reg.score(X_test,y_test)
xgb_reg_rmse = rmse(y_test, xgb_reg.predict(X_test))
xgb_reg_score, xgb_reg_rmse
```

```
(0.887506266084835, 47.54292269917127)
```

Now the accuracy score for all the 5 models considered vs their root mean square error values is shown:



```
#The linear Regression model for the above dataset is already done when we got the correct values i.e 89.2%
```

```
print(pd.DataFrame([{'Model': 'Linear Regression','Score':lr_score, "RMSE":lr_rmse},
              {'Model': 'Lasso','Score':lr_lasso_score, "RMSE":lr_lasso_rmse},
              {'Model': 'Support Vector Machine','Score':svr_score, "RMSE":svr_rmse},
              {'Model': 'Random Forest','Score':rfr_score, "RMSE":rfr_rmse},
              {'Model': 'XGBoost','Score':xgb_reg_score, "RMSE":xgb_reg_rmse}],
             columns=['Model','Score','RMSE']))
```

```
                    Model         Score          RMSE
0       Linear Regression -1.152297e+23  4.811760e+13
1                   Lasso  8.541880e-01  5.412755e+01
2  Support Vector Machine  2.569025e-02  1.399169e+02
3           Random Forest  8.899172e-01  4.703070e+01
4                 XGBoost  8.875063e-01  4.754292e+01
```

Seeing this we can clearly see that SVR is out of question with only 25% accuracy while all other perform in a similar way.

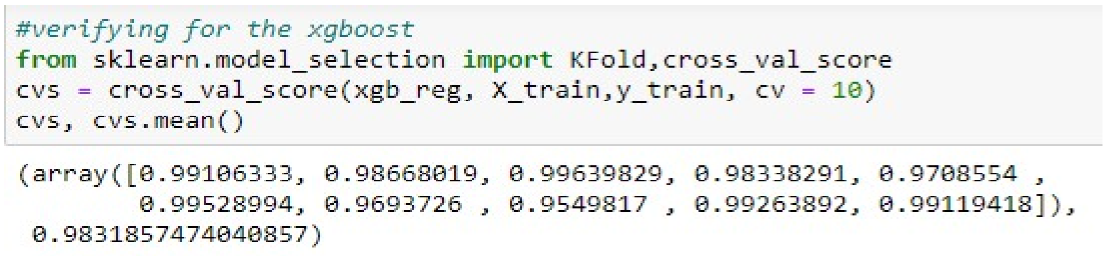
The Random Forest Regressor is a step ahead of other with an accuracy score of almost 88.9% that is almost 89%.

While the XGboost has an accuracy around 88.7% not so less from the RFR while the lasso regression model gives an accuracy score of 85.4% and MLR gives an accuracy score of 89.2% when tested individually but here it gives a different value.

4.6 Cross Validation:

Now all the models' accuracy measure on the training data is measured and verified to check the measure of over-fitting.

XGboost:

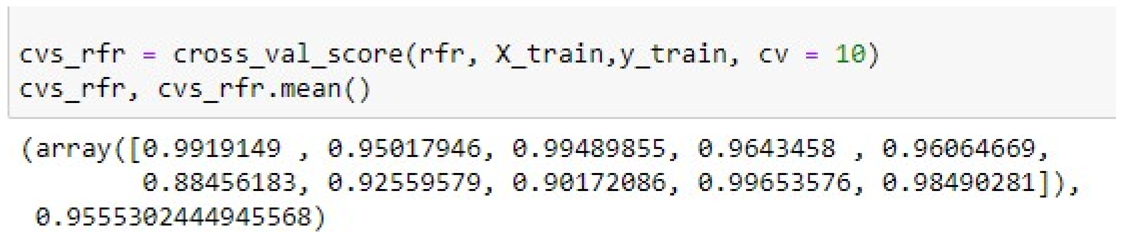


```
#verifying for the xgboost
from sklearn.model_selection import KFold,cross_val_score
cvs = cross_val_score(xgb_reg, X_train,y_train, cv = 10)
cvs, cvs.mean()
```

```
(array([0.99106333, 0.98668019, 0.99639829, 0.98338291, 0.9708554 ,
        0.99528994, 0.9693726 , 0.9549817 , 0.99263892, 0.99119418]),
 0.9831857474040857)
```

98% accuracy.

Random Forest Regressor:



```
cvs_rfr = cross_val_score(rfr, X_train,y_train, cv = 10)
cvs_rfr, cvs_rfr.mean()
```

```
(array([0.9919149 , 0.95017946, 0.99489855, 0.9643458 , 0.96064669,
        0.88456183, 0.92559579, 0.90172086, 0.99653576, 0.98490281]),
 0.9555302444945568)
```

95% accuracy

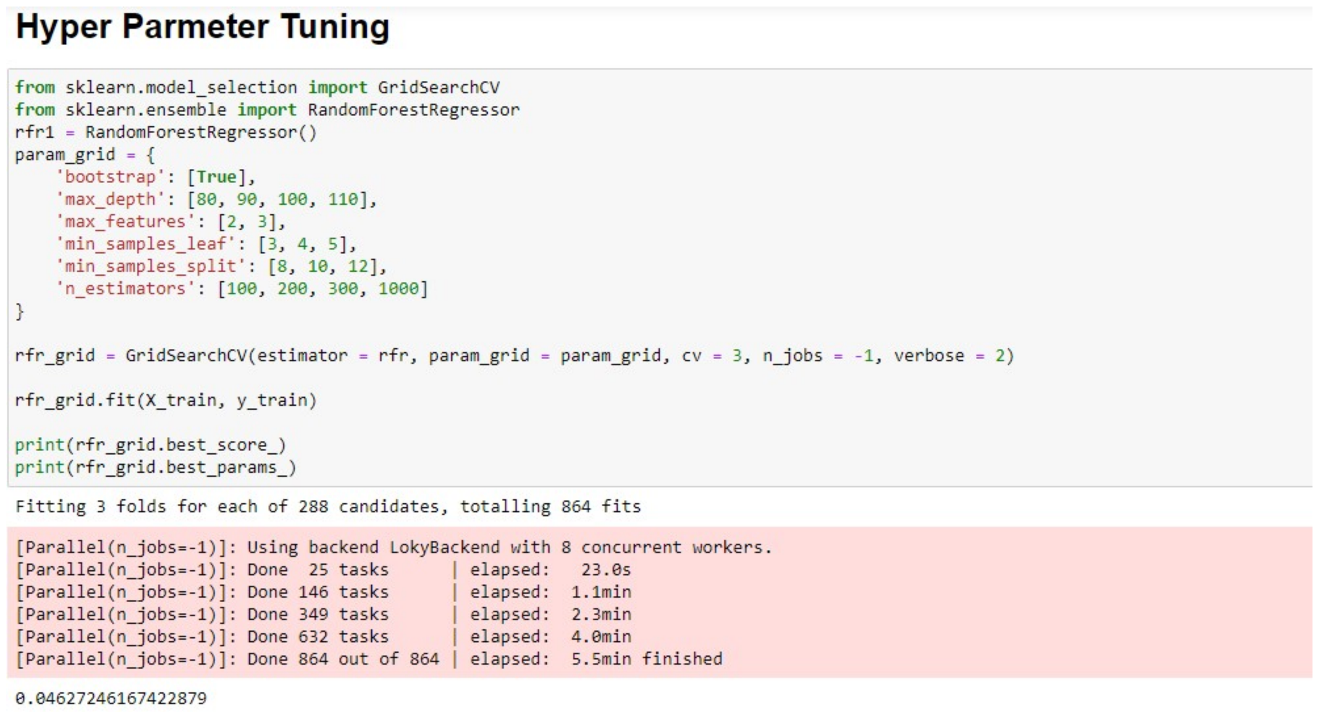
So the model is over-fitting.

We need to tune the model using hyper parameter tuning.

4.7. Hyper Parameter Tuning:

The hyper parameters for the Random Forest Regressor are set and the tuning is done as shown below:

The parameters taken into consideration are bootstrap, max_depth, max_features, min_samples_leaf, min_samples_split and n_estimators.



Hyper Parmeter Tuning

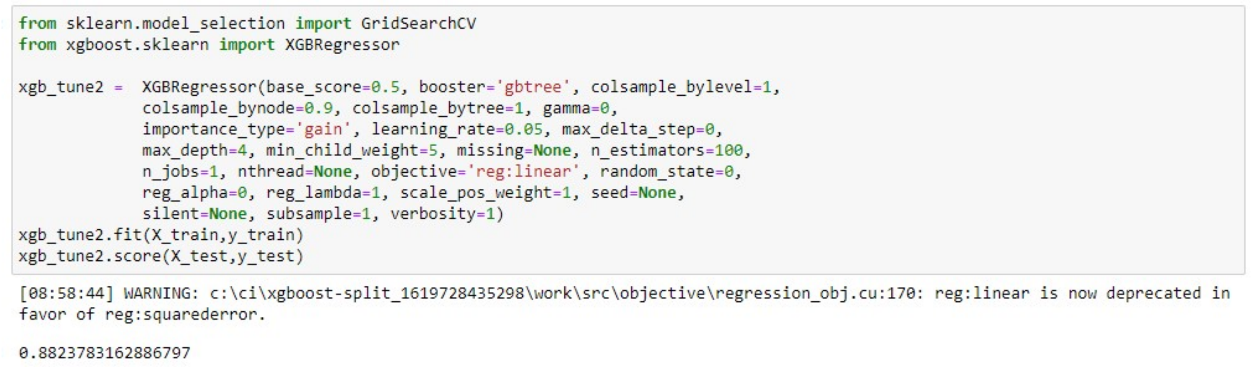
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
rfr1 = RandomForestRegressor()
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}

rfr_grid = GridSearchCV(estimator = rfr, param_grid = param_grid, cv = 3, n_jobs = -1, verbose = 2)

rfr_grid.fit(X_train, y_train)

print(rfr_grid.best_score_)
print(rfr_grid.best_params_)
```

```
Fitting 3 folds for each of 288 candidates, totalling 864 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   23.0s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done 864 out of 864 | elapsed:  5.5min finished
```

```
0.04627246167422879
```

Now the hyper parameter tuning for the XGboost regression is done.



```
from sklearn.model_selection import GridSearchCV
from xgboost.sklearn import XGBRegressor

xgb_tune2 =  XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=0.9, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=4, min_child_weight=5, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
xgb_tune2.fit(X_train,y_train)
xgb_tune2.score(X_test,y_test)
```

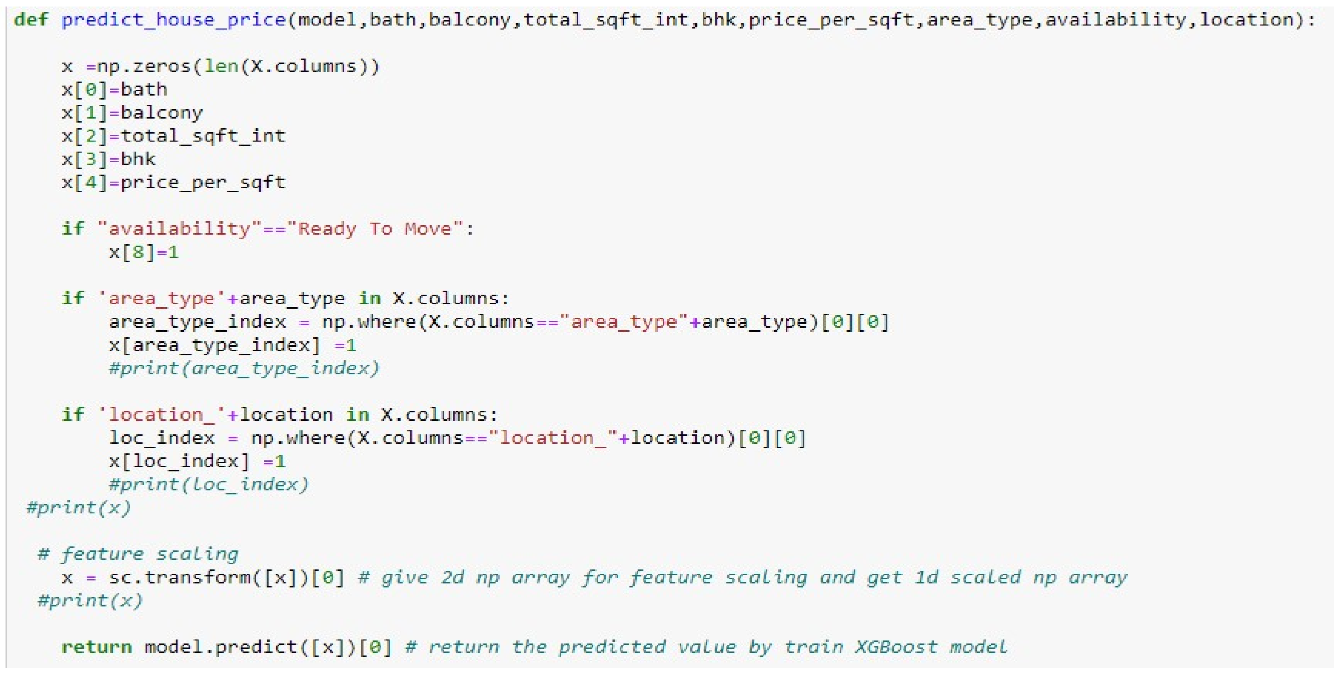
```
[08:58:44] WARNING: c:\ci\xgboost-split_1619728435298\work\src\objective\regression_obj.cu:170: reg:linear is now deprecated in
favor of reg:squarederror.

0.8823783162886797
```

As we can see, even after tuning the XGboost regressor, it is still over-fitting and thus we can use the Random Forest Regression or the general Multiple Linear Regression for our Machine Learning Model.

Since we have decided on the regression model used for our prediction model, we can load the model and test it on some sample inputs.

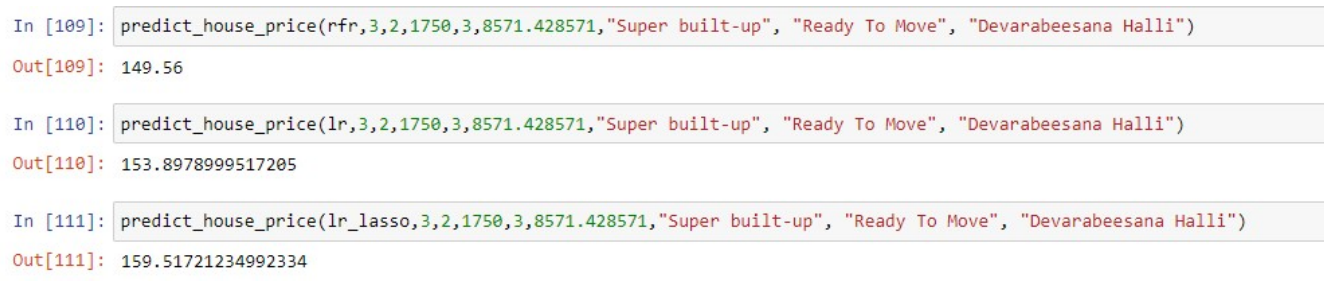
4.8. Testing the Model:

To test, firstly, the data is converted into a list of input values and a testing function is created as shown:



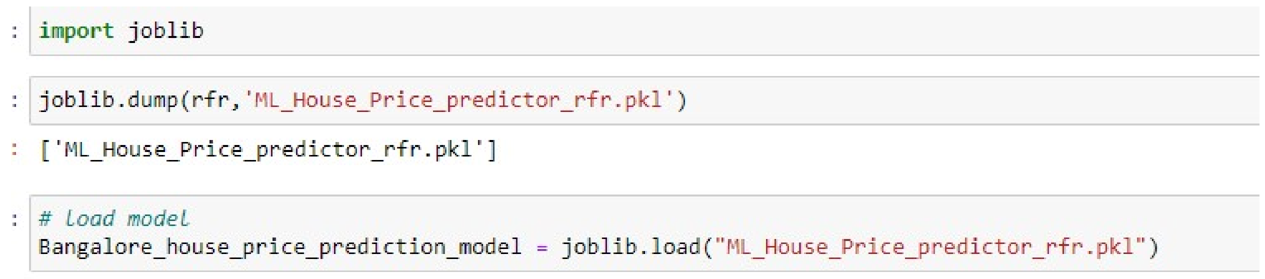
```
def predict_house_price(model,bath,balcony,total_sqft_int,bhk,price_per_sqft,area_type,availability,location):

    x =np.zeros(len(X.columns))
    x[0]=bath
    x[1]=balcony
    x[2]=total_sqft_int
    x[3]=bhk
    x[4]=price_per_sqft

    if "availability"=="Ready To Move":
        x[8]=1

    if 'area_type'+area_type in X.columns:
        area_type_index = np.where(X.columns=="area_type"+area_type)[0][0]
        x[area_type_index] =1
        #print(area_type_index)

    if 'location_'+location in X.columns:
        loc_index = np.where(X.columns=="location_"+location)[0][0]
        x[loc_index] =1
        #print(loc_index)
 #print(x)

  # feature scaling
    x = sc.transform([x])[0] # give 2d np array for feature scaling and get 1d scaled np array
  #print(x)

    return model.predict([x])[0] # return the predicted value by train XGBoost model
```

Once the function is done, we can test the model using some sample inputs:



```
In [109]: predict_house_price(rfr,3,2,1750,3,8571.428571,"Super built-up", "Ready To Move", "Devarabeesana Halli")
Out[109]: 149.56

In [110]: predict_house_price(lr,3,2,1750,3,8571.428571,"Super built-up", "Ready To Move", "Devarabeesana Halli")
Out[110]: 153.8978999517205

In [111]: predict_house_price(lr_lasso,3,2,1750,3,8571.428571,"Super built-up", "Ready To Move", "Devarabeesana Halli")
Out[111]: 159.51721234992334
```

Once tested, we can save and load the ML model as a .pkl file.



```
: import joblib

: joblib.dump(rfr,'ML_House_Price_predictor_rfr.pkl')
: ['ML_House_Price_predictor_rfr.pkl']

: # Load model
  Bangalore_house_price_prediction_model = joblib.load("ML_House_Price_predictor_rfr.pkl")
```

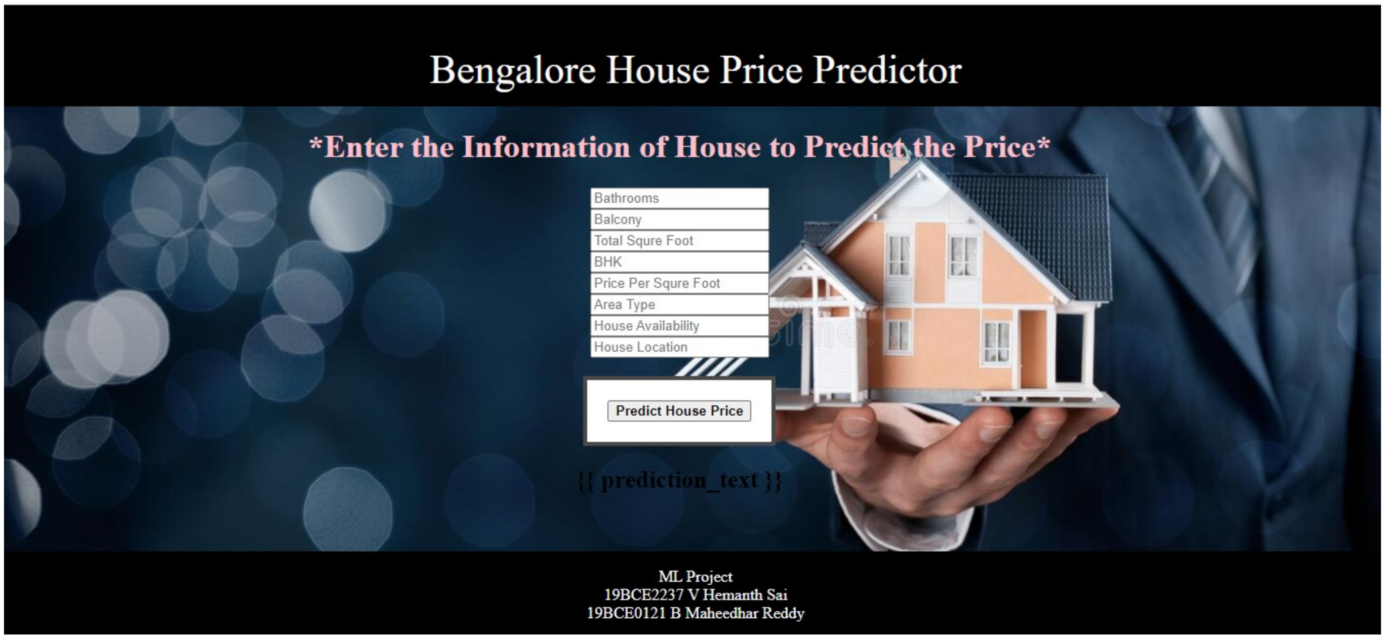
4.9. Model Deployment:

The model creation is done now.

Now the final stage of the project comes to the deployment of the ML model created.

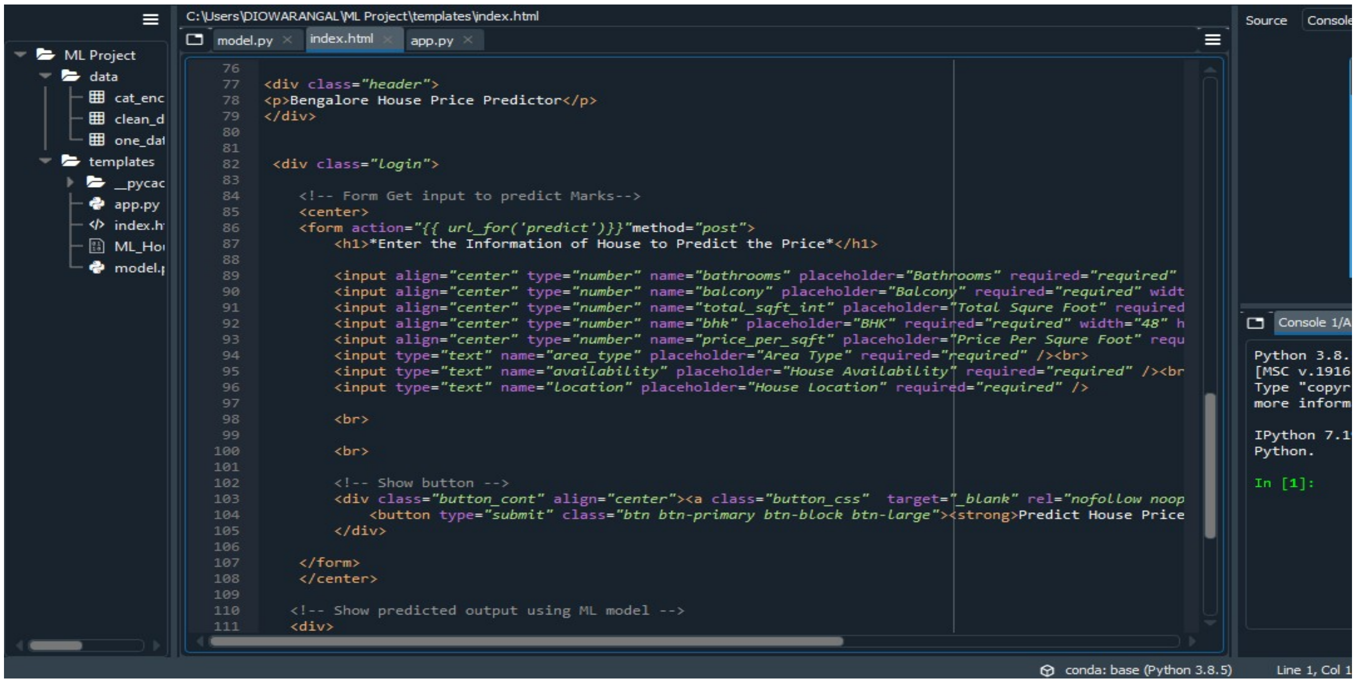
Now deployment consists of creating a webpage visible to the end users and then a server using flask on which we can host the page and then a model.py which contains the function that our ML model performs to predict and hence the flask server takes the input from the html file that is from the user through the web page and then it applies it to the model.py which predicts the value and sends it to the server, the server then sends the output predicted value to the webpage which is then displayed to the end-user.

The webpage created for the ML project is as shown:



As you can see, the user can enter the input values in the boxes displayed and then click the predict house price button which then displays the predicted value below in place of the {{prediction_text}}.

The inputs taken are checked and sent using the html code shown below:

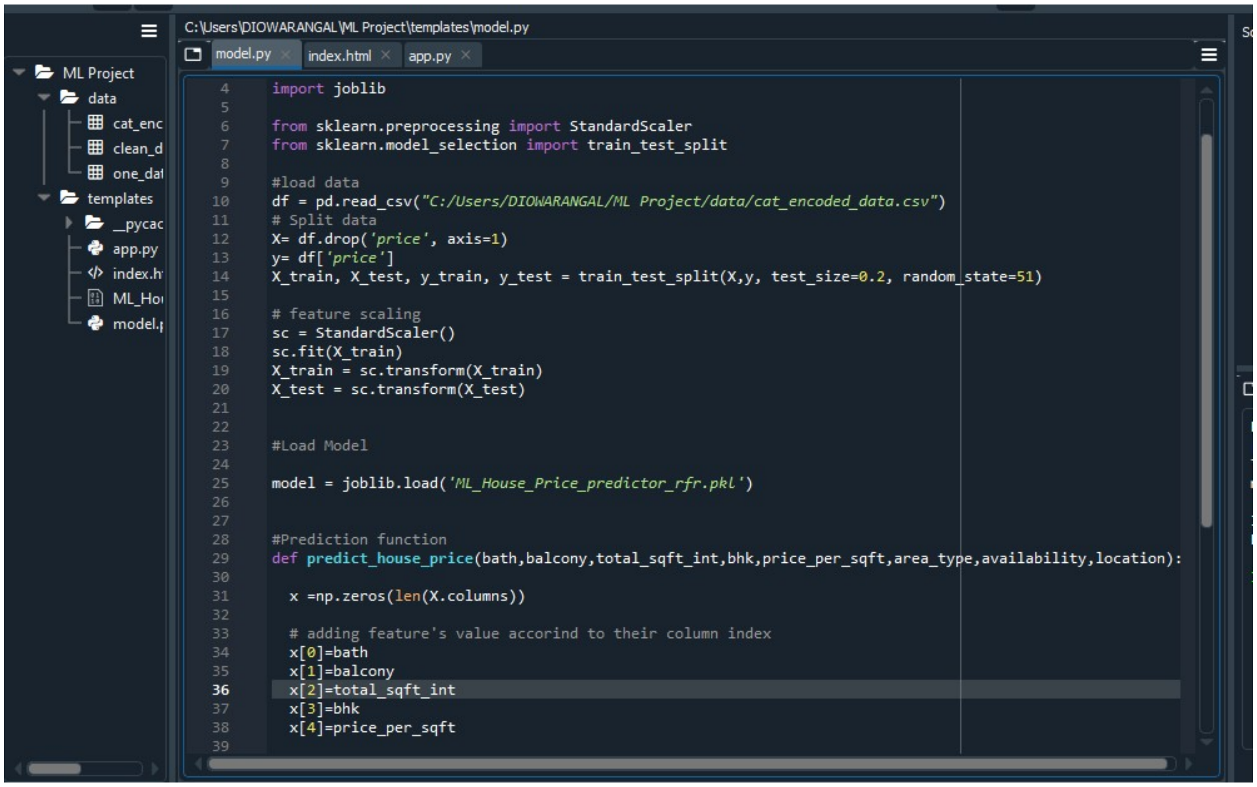


```html

<div class="header">
<p>Bengalore House Price Predictor</p>
</div>


 <div class="login">

    <!-- Form Get input to predict Marks-->
    <center>
    <form action="{{ url_for('predict')}}"method="post">
        <h1>*Enter the Information of House to Predict the Price*</h1>

        <input align="center" type="number" name="bathrooms" placeholder="Bathrooms" required="required"
        <input align="center" type="number" name="balcony" placeholder="Balcony" required="required" widt
        <input align="center" type="number" name="total_sqft_int" placeholder="Total Squre Foot" required
        <input align="center" type="number" name="bhk" placeholder="BHK" required="required" width="48" h
        <input align="center" type="number" name="price_per_sqft" placeholder="Price Per Squre Foot" requ
        <input type="text" name="area_type" placeholder="Area Type" required="required" /><br>
        <input type="text" name="availability" placeholder="House Availability" required="required" /><br
        <input type="text" name="location" placeholder="House Location" required="required" />

        <br>

        <br>

        <!-- Show button -->
        <div class="button_cont" align="center"><a class="button_css"  target="_blank" rel="nofollow noop
            <button type="submit" class="btn btn-primary btn-block btn-large"><strong>Predict House Price
        </div>

    </form>
    </center>

   <!-- Show predicted output using ML model -->
   <div>
```

```
Python 3.8.
[MSC v.1916
Type "copyr
more inform

IPython 7.1
Python.

In [1]:
```

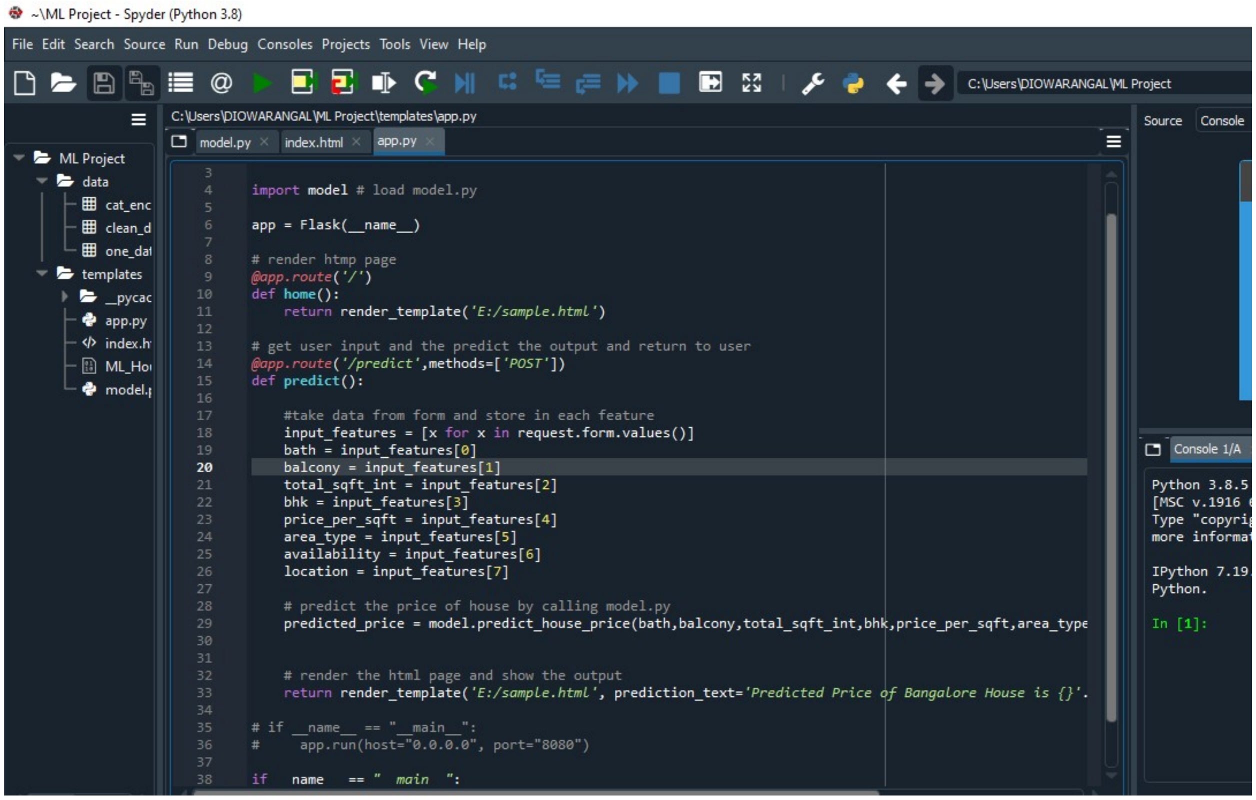
Then the back-end that is the model.py file is created which contains the function that we created as shown above:

Then we will also create the flask server in the file named as app.py and we place all these files in a folder named as templates.



```
import joblib

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#load data
df = pd.read_csv("C:/Users/DIOWARANGAL/ML Project/data/cat_encoded_data.csv")
# Split data
X= df.drop('price', axis=1)
y= df['price']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=51)

# feature scaling
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)


#Load Model

model = joblib.load('ML_House_Price_predictor_rfr.pkl')


#Prediction function
def predict_house_price(bath,balcony,total_sqft_int,bhk,price_per_sqft,area_type,availability,location):

  x =np.zeros(len(X.columns))

  # adding feature's value accorind to their column index
  x[0]=bath
  x[1]=balcony
  x[2]=total_sqft_int
  x[3]=bhk
  x[4]=price_per_sqft
```

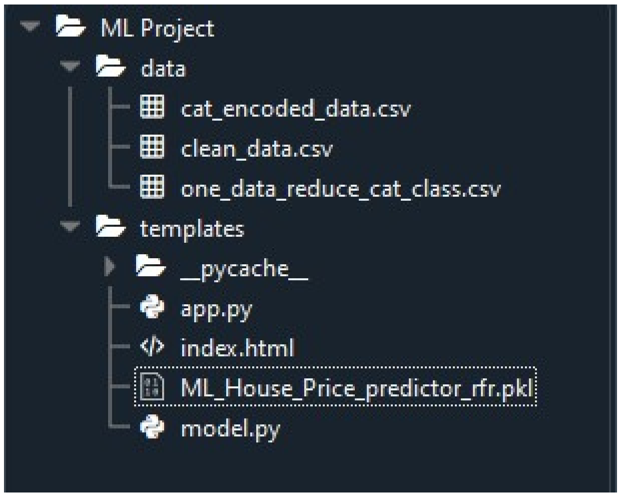
Model.py (above) and app.py (below)


~\ML Project - Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\DIOWARANGAL\ML Project
C:\Users\DIOWARANGAL\ML Project\templates\app.py
model.py
index.html
app.py
Source
Console
Console 1/A

```python

import model # load model.py

app = Flask(__name__)

# render htmp page
@app.route('/')
def home():
    return render_template('E:/sample.html')

# get user input and the predict the output and return to user
@app.route('/predict',methods=['POST'])
def predict():

    #take data from form and store in each feature
    input_features = [x for x in request.form.values()]
    bath = input_features[0]
    balcony = input_features[1]
    total_sqft_int = input_features[2]
    bhk = input_features[3]
    price_per_sqft = input_features[4]
    area_type = input_features[5]
    availability = input_features[6]
    location = input_features[7]

    # predict the price of house by calling model.py
    predicted_price = model.predict_house_price(bath,balcony,total_sqft_int,bhk,price_per_sqft,area_type


    # render the html page and show the output
    return render_template('E:/sample.html', prediction_text='Predicted Price of Bangalore House is {}'.

# if __name__ == "__main__":
#     app.run(host="0.0.0.0", port="8080")

if __name__ == "__main__":
```

```
Python 3.8.5
[MSC v.1916
Type "copyri
more informa

IPython 7.19
Python.

In [1]:
```

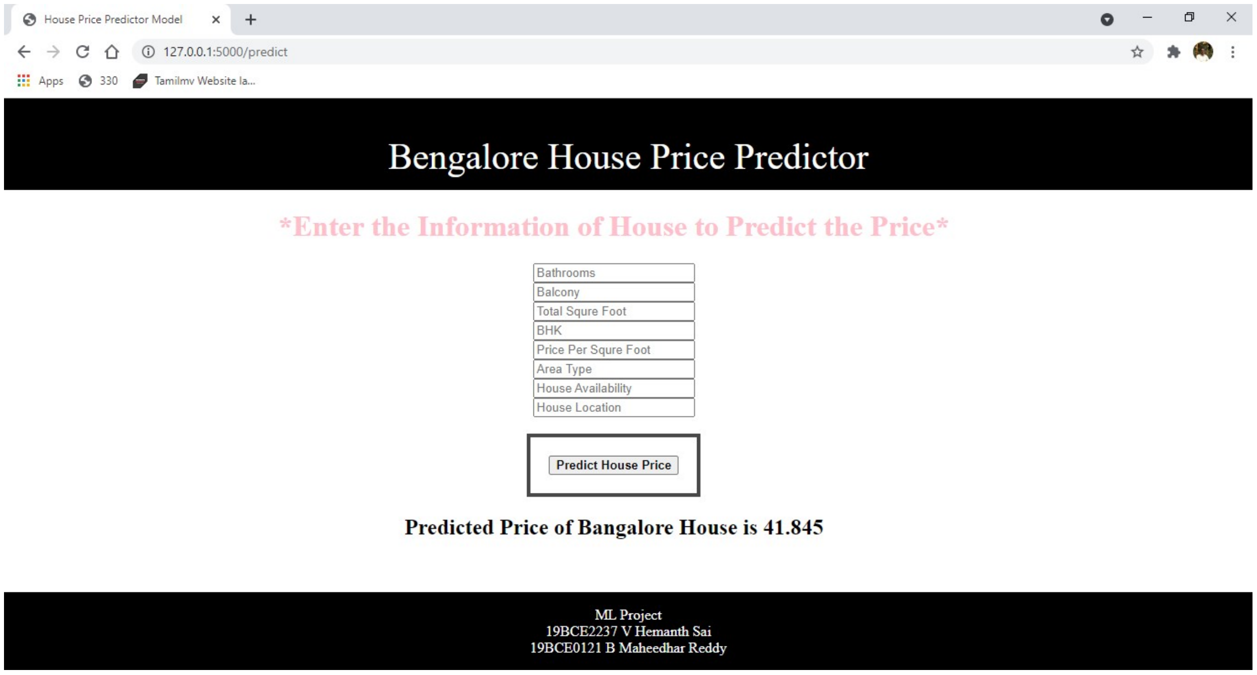
We have also included the .pkl file created earlier and you can also see those csv files we created under the data folder in our project folder.



```
ML Project
  data
    cat_encoded_data.csv
    clean_data.csv
    one_data_reduce_cat_class.csv
  templates
    __pycache__
    app.py
    index.html
    ML_House_Price_predictor_rfr.pkl
    model.py
```

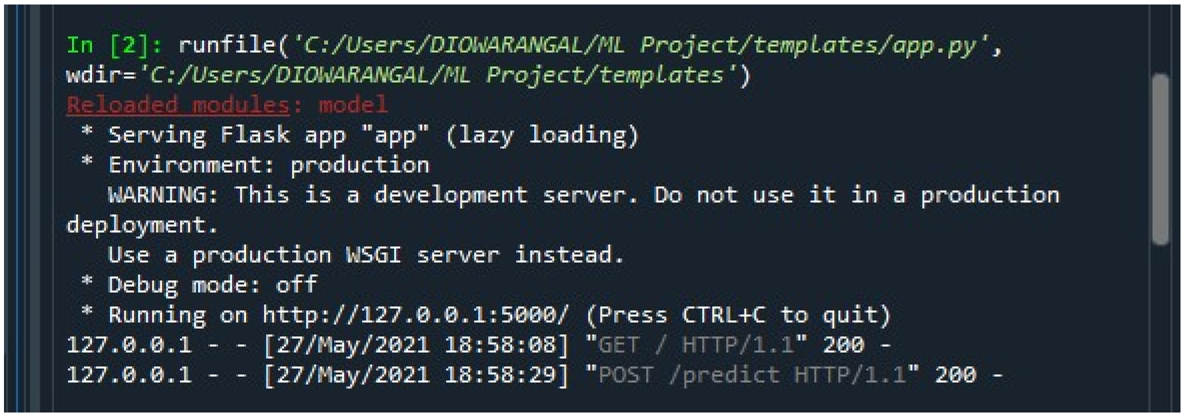
Once all this is done we can run the app.py file which will give us the ip address which when pasted on the browser leads us to the webpage we created and shown above:

For some reason flask is not identifying the background image placed in the html file.



Once the predict house price is clicked, the input is cleared and hence we are not able to see the input values given. The predicted price for the above given data is 41lakhs 84.5 thousand rupees.

Below you can see the ip address being presented and also the actions performed by the user are also recorded as shown:



```
In [2]: runfile('C:/Users/DIOWARANGAL/ML Project/templates/app.py',
wdir='C:/Users/DIOWARANGAL/ML Project/templates')
Reloaded modules: model
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [27/May/2021 18:58:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/May/2021 18:58:29] "POST /predict HTTP/1.1" 200 -
```

CONCLUSIONS AND FUTURE WORK:-

Conclusion-

A regression machine learning model is created to study and predict the house prices in Bangalore, the silicon valley of India, a place with diverse people and a complex house price structure. The data is cleaned processed and feature engineering was used to trace out the relevant parameters affecting the house prices in Bangalore. After the data is processed a regression model with the most accuracy measure which is Random Forest Regressor was used to predict the house prices. The model is built and deployed in a simple but clean webpage which ensures easier user experience to get the required output in a simple and quick manner. All in all, a model to finally bring in the relevance and predict the complex structure of Bangalore is built for tenants to better plan and study before planning to buy a house in which they are going to stay for a long period of time.

Significance of our project:

Most of the house price prediction models are built using Multiple Linear Regression while considering only numerical attributes for predicting the output but we used tuned Random Forest Regressor which uses ensemble learning which is useful both for Regression and classification analysis. We even used One Encoding to encode the categorical attributes which leads to a very specific house type rather than some general house size in a particular area. While using all the attributes present in the dataset and getting 95% accuracy is considered very good for a regression model. Other than that we deployed the ML model to be useful for general users who does not need any knowledge of programming to predict the prices and our server presents all the tasks done by the user to the admin too, thus, creating a complete, efficient and user-friendly ML model project.

Future Work:

Although the model presents 90% accuracy in predicting the values, it gave 94% accuracy on the training data and hence even better hyper parameter tuning may lead to a much accurate model. The webpage is simple and clean but it does not store the input values or the user information. Anyone can access and use it like a calculator but it does not store the information and hence a user login and a database to store all the input values given by a particular user. The model's webpage can also be converted to take in several input values and simultaneously show the predicted house price as and when the inputs are given rather than running the whole server again and again. These are the minimal limitations of the project mainly in the user experience component which can be improved in the future.

REFERENCES:-

1. G. Naga Satish, Ch. V. Raghavendran, M.D.Sugnana Rao, Ch.Srinivasulu, "House Price Prediction Using Machine Learning", International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-8 Issue-9, July 2019.
2. Ahmad Abdulal, Nawar Aghi, "House Price Prediction", Kristian Stad University, Degree Project, Spring Semester, 2020.
3. Sifei Lu, Zengxiang Li, Zheng Qin, Xulei Yang, Rick Siow Mong Goh, "A Hybrid Regression Technique for House Prices Prediction", Institute of High Performance Computing (IHPC), August 2018.
4. Joep Steegmans, Wolter Hassink, "Financial position and house price determination: An empirical study of income and wealth effects", Journal of Housing Economics, vol. 36, pp. 8-24, June 2017.
5. David C. Ling, Joseph T.L. Ooi and Thao T.T. Le, "Explaining house price dynamics: Isolating the role of nonfundamentals", Journal of Money, Credit and Banking, vol. 47, Issue S1, pp. 87-125, March/April 2015.
6. Sean Holly, M. Hashem Pesarana, Takashi Yamagata, "A spatio-temporal model of house prices in the USA", Journal of Econometrics, vol. 158, Issue 1, pp. 160–173, Sep. 2010.
7. Yang Z, Wang S. "Permanent and transitory shocks in owner-occupied housing: A common trend model of price dynamics", Journal of Housing Economics, vol. 21, Issue 24, pp. 336-46, Dec 2012.
8. Landberg N, "The Swedish Housing Market: An empirical analysis of the price development on the Swedish housing market", Master of Science Thesis. Stockholm: KTH, Engineering and Management; 2015.