Mini-Project Report

Department:	Mathematical and Computational Sciences
Specialization:	Computational and Data Science
Subject:	Machine Learning
Topic:	House Price Prediction Using Machine Learning
Course Instructor:	Dr. Jidesh P.
Date:	14/05/2021

Team Number 2

- Gavali Deshabhakt Nagnath - 202CD005
- Mohammad Ahsan - 202CD016
- Shimpi Mayur Anil - 202CD027

Index

Chapter - I

Introduction and Methodology

Problem Statement:

Building machine learning model for predicting house price from a dataset of houses with different features and sizes.

Introduction:

This project is about building a machine learning model which can be used for prediction of price of houses. We have taken data set from Kaggle for Bangalore city of India and using that data set we have developed a machine learning model which can predict prices for houses. While building the model we have used some of important data science concepts such as Data load and cleaning, Feature Engineering, Outlier detection and removal, One hot encoding, dimensionality reduction etc.

For doing this project we have used Python as programming language. We have used some of the libraries of python as per our requirement. Such as Numpy and Pandas library for data cleaning, Matplotlib for data visualization, scikit-learn for model building etc. In the last part of the project, we have made GUI to make Input and output user friendly. Methodology part will explain in brief the steps we have followed to reach the final destination.

Methodology:

Step 1: Downloading Database

We have taken data set from Kaggle for Bangalore city of India. The database containing useful attributes of Bangalore houses which help in developing a machine learning model for house prediction.

Step 2: Data preprocessing

In this step we have removed some unnecessary features from the data set which won't be useful for predicting price. We have handled NA values. We have converted the range of property size (such as 2100-3250) into an average of min and max. We have also performed certain other operations so that we can use the data for further analysis.

Step 3: Feature Engineering

After cleaning of database, the most important part of our data pipeline is to create some useful features to have better analysis of the data.

Step 4: Removing Outliers

In this step we have done outliers detection and removal. Outliers are data errors or sometimes they are data points which represents some extreme variations in our datasets and which can make problems is analysis. With the help of some statistical techniques, we have detected and removed those outliers.

Step 6: Performing One-Hot-encoding

As we know that machine learning model cannot interpret text data so in this step, we have performed one-hot-encoding on location column to convert it into numeric column. It will create separate new columns with unique name of locations and assign '1' to rows containing that particular location and '0' to other rows for each new location columns. This is required to train our machine learning model using supervised learning technique

Step 7: Building Machine learning Model

In this step we have built a machine learning model to train and test our data. To use machine learning we have to first train our model with database. After training our model we again have to test it on some database. So, in order to achieve this, we have split our database in train and testing sets. We have selected linear regression technique for training the data among linear regression, lasso regression and decision tree regression.

Step 8: Making of GUI

After preprocessing our database contained around 243 locations, so it became very hard to type in location name while predicting the price of house in that particular location. So, in order to overcome this difficult we made a simple graphical user interface using ‘gradio’ library which enables us to give 4 inputs – area/size (in sqft), BHK, bathrooms, location and displays cost corresponding to input parameters in output section. This made Input and Output more user friendly.

Chapter - II

Downloading Database

The data base we used in the project is available on ‘Kaggle’ website.

The database containing useful attributes of Bangalore houses which help in developing a machine learning model for house prediction.

The database contains attributes as columns such as-

‘area_ type’: 'Super built-up ', 'Plot', 'Built-up ', 'Carpet' areas.

‘location’: All important locations of Bangalore where house is available.

‘size’: It describe the type of apartment it is like: 2BHK, 3BHK, 2 Bedrooms etc.

‘baths’: This column describes the number of bathrooms that house have, which helps in removing the outliers.

‘total_sqft’: It describe the area size of the house.

‘price’: Finally, the house price which act as dependent variable for training and testing of our machine learning model.

‘Others’: There are some unnecessary columns such as ‘availability’, ‘society’, ’balcony’, which we removed as they do not help in deciding the price of the house.

The link to download the Bangalore-house database is available here:

https://www.kaggle.com/amitabhajoy/bengaluru-house-price-data

Data Pre-processing

We have done our project using 'Jupyter Notebook' in python programming, which includes importing libraries such as 'pandas', 'numpy', 'matplotlib', etc.

Importing Libraries:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import matplotlib
matplotlib.rcParams["figure.figsize"] = (20,10)
```

Pandas: Pandas helps in data manipulation and analysis.

Numpy: Numpy library comes handy while doing large sized matrix operations along with some important mathematical tools.

Matplotlib: This library provides sufficient functions to plot graphs which is very much required for analysis and understanding of large problem sets.

Loading 'Bangalore_House_Price' database:

```
df1 = pd.read_csv("bengaluru_house_prices.csv")
df1.head()
```

We here import our 'bangaluru_house_price.csv' database to our python program, using pandas.read_csv().

Structure of the imported database:

```
df1.head()
```

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

```
df1.shape
```

```
(13320, 9)
```

```
df1.columns
```

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

As you can clearly see here that there are '13320 rows' and '9 columns' available with our database, which even contains some of the unnecessary columns and thus required cleaning of the data to reduce the dimension and speed the processing capability.

Dropping unnecessary columns:

```
df2 = df1.drop(['area_type','society','balcony','availability'],axis='columns')
df2.shape
```

```
(13320, 5)
```

Here we dropped columns such as 'area_type', 'society', 'balcony', 'availability' from our database, because these were not needed for analysis.

Checking for 'Null' values in database:

```
df2.isnull().sum()
```

```
location       1
size          16
total_sqft     0
bath          73
price          0
dtype: int64
```

```
df3 = df2.dropna()
df3.isnull().sum()
```

```
df3.shape
```

```
(13246, 5)
```

As we can see that there exist some 'Null' values in respective columns. As they are few in number hence can be safely removed from database.

Feature Engineering

After cleaning of database, the most important part of our data pipeline is to create some useful features to have better analysis of the data. So, in this chapter we have discussed the processes of generation of useful features.

Adding new feature 'bhk':

As far of analysis we found that the 'size' column have alpha-numeric format and the numerical values determine the size of the apartment/house. So from this we extract the numeric values and assigned a new column for that. i.e. 'bhk'.

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

	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

Here we separated the bhk (number) from size column using simple lambda function.

Updating the 'total_sqft' column values to 'float' data type:

As the values present in each cell of the 'total_sqft' column contain numeric value in 'string' format. Thus, we are required to convert it to float so as to make some mathematical analysis over that column when required.

```
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
```

```
df4 = df3.copy()
df4.total_sqft = df4.total_sqft.apply(convert_sqft_to_num)
df4 = df4[df4.total_sqft.notnull()]
df4.head(2)
```

	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

For this purpose we use 'convert_sqft_to_num()' to convert it to float data type and updated the column and formed a new data frame by the name of 'df4' where do not exist any 'null' value in 'total_sqft' column.

Adding a new column 'price_per_sqft':

We are required this column to remove some outliers whose 'price_per_sqft' is less than the threshold value i.e., '300' (in rupees) (our assumption).

For that we used this formula: price_per_sqft= (price/total_sqft)

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

	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

<u>Checking for the all-possible unique locations available in database:</u>

```
df5.location = df5.location.apply(lambda x: x.strip())
location_stats = df5['location'].value_counts(ascending=False)
location_stats
```

```
Whitefield                    533
Sarjapur  Road                392
Electronic City               304
Kanakpura Road                264
Thanisandra                   235
Yelahanka                     210
Uttarahalli                   186
Hebbal                        176
Marathahalli                  175
Raja Rajeshwari Nagar         171
Bannerghatta Road             151
Hennur Road                   150
```

```
                                                                  ...
Ckikkakammana Halli                                                1
Neelasandra                                                        1
Gangondanahalli                                                    1
Agara Village                                                      1
Sundara Nagar                                                      1
Binny Mills Employees Colony                                       1
Adugodi                                                            1
Uvce Layout                                                        1
Kenchanehalli R R Nagar                                            1
Whietfield,                                                        1
manyata                                                            1
Air View Colony                                                    1
Thavarekere                                                        1
Muthyala Nagar                                                     1
Haralur Road,                                                      1
Manonarayanapalya                                                  1
GKW Layout                                                         1
Marathalli bridge                                                  1
Banashankari 6th Stage ,Subramanyapura                             1
anjananager magdi road                                             1
akshaya nagar t c palya                                            1
Indiranagar HAL 2nd Stage                                          1
Maruthi HBCS Layout                                                1
Gopal Reddy Layout                                                 1
High grounds                                                       1
CMH Road                                                           1
Chambenahalli                                                      1
Sarvobhogam Nagar                                                  1
Ex-Servicemen Colony Dinnur Main Road R.T.Nagar                    1
Bilal Nagar                                                        1
Name: location, Length: 1287, dtype: int64
```

So you can see here that there are exactly ‘1287’ unique locations with their respective counts of number of available houses to each location. If we have to do analysis and train our ML model we are required to create ‘1287’ new columns while doing ‘one-hot-encoding’. Thus, it will create high dimensional complexity and consume huge resources to process such a large dataset.

Thus, it’s the prime requirement to reduce look for some dimensionality reduction option. And for that we are putting all such location for which count is less than ‘10’ into a separate category i.e. ‘others’.

Performing dimensionality reduction over 'location' column:

```
location_stats_less_than_10 = location_stats[location_stats<=10]
location_stats_less_than_10
```

```
BTM 1st Stage                                10
Sector 1 HSR Layout                          10
Ganga Nagar                                  10
Naganathapura                                10
1st Block Koramangala                        10
Thyagaraja Nagar                             10
Dairy Circle                                 10
Nagadevanahalli                              10
Sadashiva Nagar                              10
Gunjur Palya                                 10
```

These are some of the columns showing the locations which has house counts less than or equal to 10. As these locations contains lesser number of houses, we put all these location into 'other' category. Doing this we will reduce number of columns significantly.

```
df5.location = df5.location.apply(lambda x: 'other' if x in location_stats_less_tha
len(df5.location.unique())
```

```
241
```

```
df5.head(10)
```

	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

So, using above functions we reduce the locations to just '241' from '1287'.

Removing Outliers

1. Removing outliers based on 'price per square feet':

As we previously made assumption that 'price_per_sqft' can't be less than 300 Rs. Thus, we need to remove any such values present in 'price_per_sqft' column.

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

	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

Illustration of such houses whose price_per_sqft are less than 300.

```
df5.shape
```

```
(13200, 7)
```

```
df6 = df5[~(df5.total_sqft/df5.bhk<300)]
df6.shape
```

```
(12456, 7)
```

So, you can see that some of the outliers corresponding (price_per_sqft) are removed. And we saved to new data frame 'df6'.

2. Removing outliers using 'statistical technique':

```
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
```

```
(10242, 7)
```

It is not possible to have huge variation in 'price_per_sqft' values for houses at same location. So we find mean and standard deviation grouping the data frame location-wise and removing any such data which having variation of more than (mean + standard deviation) and lesser than (mean – standard deviation) from mean value of the particular location.

So, after following the above defined function you can see we arrived at just 10k rose from 12.5k and the result is finally stored to new data frame 'df7'.

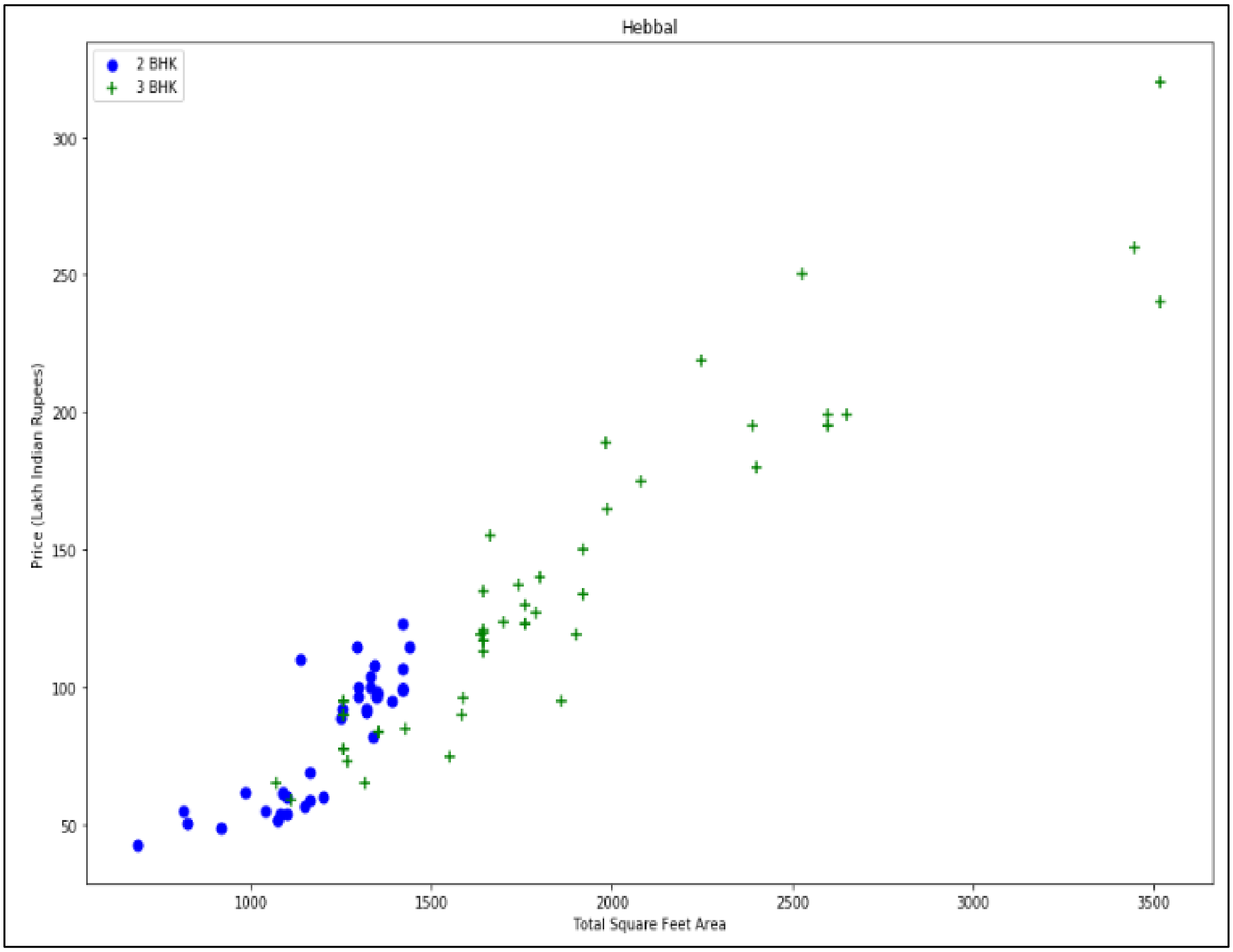
3. Detecting the outlier and their removal by plotting as scatter plot:

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

```
plot_scatter_chart(df7,"Hebbal")
```

We can follow from the scatter plot shown on next page, and can see the comparison of 2-bhk and 3-bhk houses at same location.

We observe that some of the 3-bhk house at same location with same square feet have price lower than 2-bhk houses , thus we treat them as outlier as such situation is not possible. And it can decrease our machine learning model accuracy.



Scatter plot comparing price of 2-bhk and 3-bhk at different Total Square Feet Area

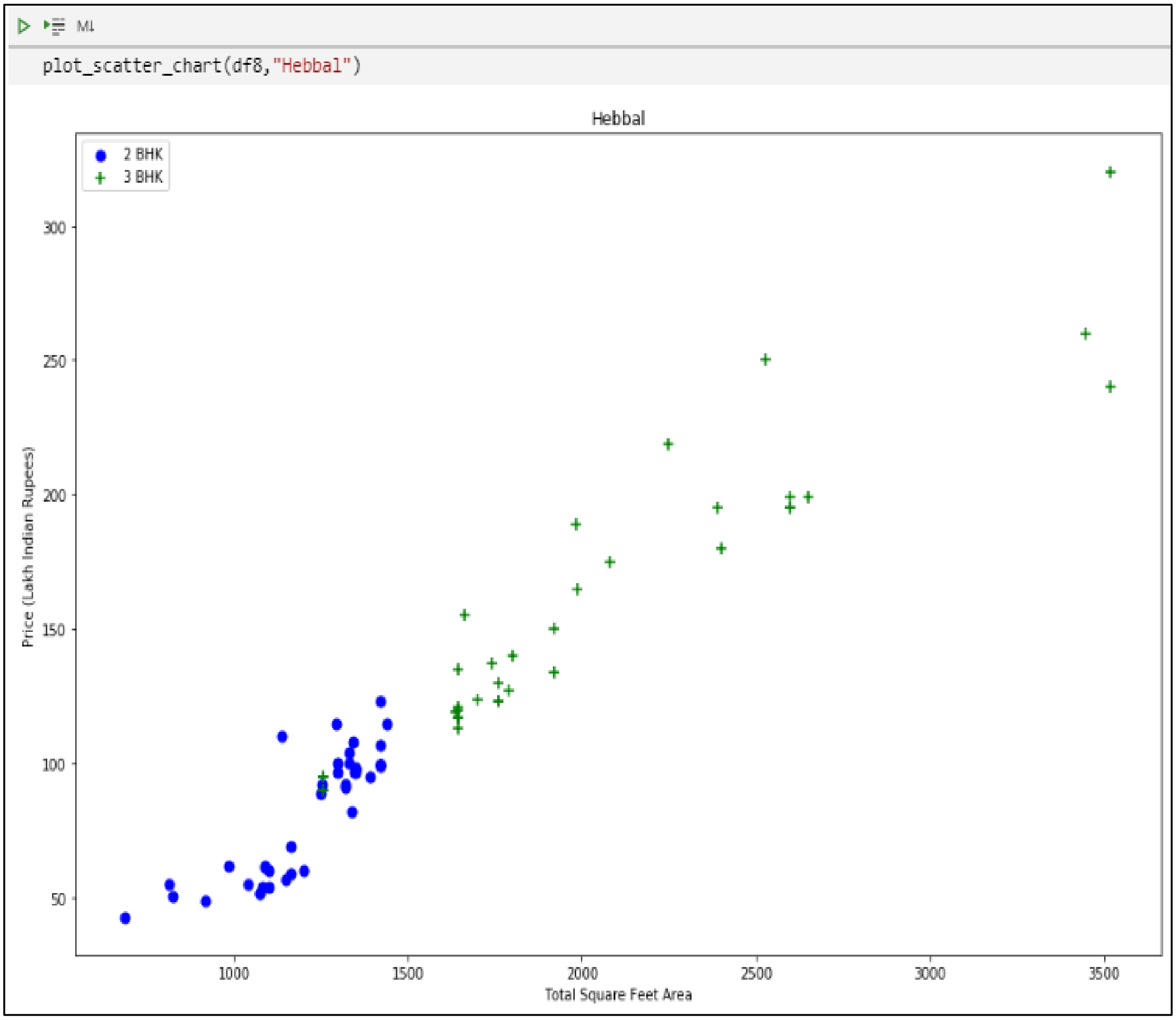
```python
def remove_bhk_outliers(df):
    exclude_indices = np.array([])
    for location, location_df in df.groupby('location'):
        bhk_stats = {}
        for bhk, bhk_df in location_df.groupby('bhk'):
            bhk_stats[bhk] = {
                'mean': np.mean(bhk_df.price_per_sqft),
                '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 = df7.copy()
df8.shape
```

```
(7317, 7)
```

Here as you can find that we grouped the data location-wise and for each sub-data frame we are finding the mean, standard-deviation and count. And later comparing the house with one less bhk value to the current bhk value.

If the price per square feet for current bhk is found to be less than '1 bhk' house in the same location. Then we consider such rows as outliers and removing such rows later.

So at the end we can see that only '7317' rows are now available with us as some of the outliers are removed to make a good ML model.



Scatter-Plot after the removal of the outlier of same location (Hebbal)

4. Removing outliers using ‘bath’ feature:

It is very unlikely to have a greater number of bathroom than bedroom.
So, for such condition we have to remove such data giving condition where number of bathrooms are more than two than number of bedrooms.
Mathematically saying, we will remove such data points where, (bath) > (bhk+2).

```
df8[df8.bath>df8.bhk+2]
```

	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
8408	other	6 BHK	11338.0	9.0	1000.0	6	8819.897689

The rows shown above need to be removed from our data frame.

```
df9 = df8[df8.bath<df8.bhk+2]
df9.shape
```

```
(7239, 7)
```

```
df9.head(2)
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	1st Block Jayanagar	4 BHK	2850.0	4.0	428.0	4	15017.543860
1	1st Block Jayanagar	3 BHK	1630.0	3.0	194.0	3	11901.840491

Now after removal of such rows, we only have ‘7239’ rows.

Dropping ‘size’ and ‘price_per_sqft’ columns:

Dropping ‘size’ and ‘price_per_sqft’ as they are not required anymore.

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

	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

Performing One-Hot-Encoding

We need to perform one-hot-encoding on location columns. It will create separate new columns with unique name of locations and assign '1' to rows containing that particular location and '0' to other rows for each new location columns.

This is required to train our machine learning model using supervised learning technique.

We do it by 'get_dummies()' of pandas applied on 'location' column of our data frame. You can see in the image below.

```
dummies = pd.get_dummies(df10.location)
dummies.head(3)
```

	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	...
0	1	0	0	0	0	0	0	0	0	0	...
1	1	0	0	0	0	0	0	0	0	0	...
2	1	0	0	0	0	0	0	0	0	0	...

3 rows × 241 columns

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

	location	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	...
0	1st Block Jayanagar	2850.0	4.0	428.0	4	1	0	0	0	0	...
1	1st Block Jayanagar	1630.0	3.0	194.0	3	1	0	0	0	0	...
2	1st Block Jayanagar	1875.0	2.0	235.0	3	1	0	0	0	0	...
3	1st Block Jayanagar	1200.0	2.0	130.0	3	1	0	0	0	0	...
4	1st Block Jayanagar	1235.0	2.0	148.0	2	1	0	0	0	0	...

5 rows × 245 columns

Now concatenating the newly created ‘dummies’ data frame to ‘df10’ using pandas.concat() as you can see here.

<u>Dropping ‘location’ column:</u>

```
df12 = df11.drop('location',axis='columns')
df12.head(2)
```

	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	...
0	2850.0	4.0	428.0	4	1	0	0	0	0	0	...
1	1630.0	3.0	194.0	3	1	0	0	0	0	0	...

2 rows × 244 columns

From here onwards we are ready to train and test our model using this data frame containing ‘7239 rows’ and ‘244 columns’.

Chapter - VII

Building Machine Learning Model

To use machine learning we first to train our model with database. After training our model we again have to test it on some database. So, in order to achieve this, we have split our database in train and testing sets.

Splitting database into training and testing database:

We need to divide our database into two different portion of database.

One on which we can train the machine learning model and other is used to test the model. We choses 20% of data points to test our models.

```
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)
```

Applying Linear Regression technique to out training examples:

```
from sklearn.linear_model import LinearRegression
lr_clf = LinearRegression()
lr_clf.fit(X_train,y_train)
lr_clf.score(X_test,y_test)
```

```
0.8629132245229449
```

We used here ‘scikit learn’ library to implement Linear Regression model on our training dataset and then cross-validating our result, which give around 86% accuracy, which can be one of the potential models for predicting the house price.

Finding best fitting model for our created database:

Here we used Linear Regression, Lasso Regression, Decision Tree Regression as three different machine learning models and comparing their result using GridSearchCV available in sklearn module, to find the best fitting model.

```python
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']
            }
        }
    }
```

```python
    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'])
```

```python
find_best_model_using_gridsearchcv(X,y)
```

	model	best_score	best_params
0	linear_regression	0.847796	{'normalize': False}
1	lasso	0.726738	{'alpha': 2, 'selection': 'cyclic'}
2	decision_tree	0.716064	{'criterion': 'friedman_mse', 'splitter': 'best'}

Based on the above result, we can say that Liner Regression model is giving best result among all three. And thus, we will use Linear Regression model as our machine learning model to predict the house price.

Testing our model by giving few properties:

```
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]
```

For this purpose, we created a function taking ‘Location’, ’Area’, ’Bathroom’, ’BHK’ as parameter to estimate the house price.

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

```
86.08062284985995
```

The example here shows that with the given parameter the expected house price is around 86 Lakh.

Also, some of the predictions are as follows,

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

```
83.86570258312172
```

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

```
86.08062284986931
```

```
predict_price('Indira Nagar',1000, 2, 2)
```

```
193.311977331799
```

```
predict_price('Indira Nagar',1000, 3, 3)
```

```
195.52689759854664
```

Making GUI

As our database contains around 243 locations (after preprocessing), so, it becomes very hard to type in location name while predicting the price of house in that particular location. So, in order to overcome this difficult we made a simple graphical user interface using 'gradio' library which enables us to give 4 inputs – area/size (in sqft), BHK, bathrooms, location and displays cost corresponding to input parameters in output section. But first we have to do following things.

1. Storing trained model in ML file:

Also, as every time we open our program we need to run/train our model which becomes a tedious task. So, to avoid this we stored our model in a 'pickle file' using '_pickle' library of python.

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

Using this code, we stored our trained ML-model in pickle file.

We can read our model from above stored file using following code.

```
model = None
with open('banglore_home_prices_model.pickle','rb') as file:
    model = pickle.load(file)
file.close()
```

2. Storing column name information in text file:

We have created the ML model for predicting house prices in which we're supposed to give location of house as one of the inputs. So, in order to retain location names from our database we stored those in a json file. Doing this we can then simply load this file to get back all location names. Also, as we did one hot encoding on locations, we formed column corresponding to each location name. So, for retaining location we can simply store column names.

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

Here we can see following is data stored in our ‘data_column.json’ file

ML-Programs > ML-Project > columns.json > ...


```
{"data_columns": ["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", "7th phase jp nagar", "8th phase jp nagar", "9th phase jp nagar", "aecs
layout", "abbigere", "akshaya nagar", "ambalipura", "ambedkar nagar", "amruthahalli", "anandapura", "ananth nagar", "anekal", "anjanapura",
"ardendale", "arekere", "attibele", "beml layout", "btm 2nd stage", "btm layout", "babusapalaya", "badavala nagar", "balagere", "banashankari",
"banashankari stage ii", "banashankari stage iii", "banashankari stage v", "banashankari stage vi", "banaswadi", "banjara layout", "bannerghatta",
"bannerghatta road", "basavangudi", "basaveshwara nagar", "battarahalli", "begur", "begur road", "bellandur", "benson town", "bharathi nagar",
"bhoganhalli", "billekahalli", "binny pete", "bisuvanahalli", "bommanahalli", "bommasandra", "bommasandra industrial area", "bommenahalli",
"brookefield", "budigere", "cv raman nagar", "chamrajpet", "chandapura", "channasandra", "chikka tirupathi", "chikkabanavar", "chikkalasandra",
"choodasandra", "cooke town", "cox town", "cunningham road", "dasanapura", "dasarahalli", "devanahalli", "devarachikkanahalli", "dodda nekkundi",
"doddaballapur", "doddakallasandra", "doddathoguru", "domlur", "dommasandra", "epip zone", "electronic city", "electronic city phase ii",
"electronics city phase 1", "frazer town", "gm palaya", "garudachar palya", "giri nagar", "gollarapalya hosahalli", "gottigere", "green glen
layout", "gubbalala", "gunjur", "hal 2nd stage", "hbr layout", "hrbr layout", "hsr layout", "haralur road", "harlur", "hebbal", "hebbal
kempapura", "hegde nagar", "hennur", "hennur road", "hoodi", "horamavu agara", "horamavu banaswadi", "hormavu", "hosa road", "hosakerehalli",
"hoskote", "hosur road", "hulimavu", "isro layout", "itpl", "iblur village", "indira nagar", "jp nagar", "jakkur", "jalahalli", "jalahalli east",
"jigani", "judicial layout", "kr puram", "kadubeesanahalli", "kadugodi", "kaggadasapura", "kaggalipura", "kaikondrahalli", "kalena agrahara",
"kalyan nagar", "kambipura", "kammanahalli", "kammasandra", "kanakapura", "kanakpura road", "kannamangala", "karuna nagar", "kasavanhalli",
"kasturi nagar", "kathriguppe", "kaval byrasandra", "kenchenahalli", "kengeri", "kengeri satellite town", "kereguddadahalli", "kodichikkanahalli",
"kodigehaali", "kodigehalli", "kodihalli", "kogilu", "konanakunte", "koramangala", "kothannur", "kothanur", "kudlu", "kudlu gate", "kumaraswami
layout", "kundalahalli", "lb shastri nagar", "laggere", "lakshminarayana pura", "lingadheeranahalli", "magadi road", "mahadevpura", "mahalakshmi
layout", "mallasandra", "malleshpalya", "malleshwaram", "marathahalli", "margondanahalli", "marsur", "mico layout", "munnekollal",
"murugeshpalya", "mysore road", "ngr layout", "nri layout", "nagarbhavi", "nagasandra", "nagavara", "nagavarapalya", "narayanapura", "neeladri
nagar", "nehru nagar", "ombr layout", "old airport road", "old madras road", "padmanabhanagar", "pai layout", "panathur", "parappana agrahara",
"pattandur agrahara", "poorna pragna layout", "prithvi layout", "r.t. nagar", "rachenahalli", "raja rajeshwari nagar", "rajaji nagar", "rajiv
nagar", "ramagondanahalli", "ramamurthy nagar", "rayasandra", "sahakara nagar", "sanjay nagar", "sarakki nagar", "sarjapur", "sarjapur  road",
"sarjapura - attibele road", "sector 2 hsr layout", "sector 7 hsr layout", "seegehalli", "shampura", "shivaji nagar", "singasandra", "somasundara
palya", "sompura", "sonnenahalli", "subramanyapura", "sultan palaya", "tc palaya", "talaghattapura", "thanisandra", "thigalarapalya",
"thubarahalli", "tindlu", "tumkur road", "ulsoor", "uttarahalli", "varthur", "varthur road", "vasanthapura", "vidyaranyapura", "vijayanagar",
"vishveshwarya layout", "vishwapriya layout", "vittasandra", "whitefield", "yelachenahalli", "yelahanka", "yelahanka new town", "yelenahalli",
"yeshwanthpur"]}
```


We can read above json file using following code.

```
column_heads = None
locations = None
with open("columns.json","r") as f:
    column_heads = json.load(f)['data_columns']
    locations = column_heads[3:]
f.close()
```

3. **<u>Making Predict price function for price prediction:</u>**

This function takes area size (in sqft), number of bathrooms, number of bhk and location as input and returns predicted price as output.

```
def predict_price(Area,BHK,Bathrooms,Location):
    if(BHK==0 or BHK>5 or Bathrooms==0 or Bathrooms>5):
        return 'Invalid input'


    loc_index = column_heads.index(Location.lower())

    x = np.zeros(len(column_heads))
    x[0] = Area
    x[1] = Bathrooms
    x[2] = BHK
    if loc_index >= 0:
        x[loc_index] = 1

    return model.predict([x])[0]
```

4. **<u>Making GUI:</u>**

As discussed earlier, Making GUI is crucial for making price prediction easy and user friendly. So, we made a GUI using gradio library.

The code for GUI is as below,

```
inp = [gr.inputs.Number(label='Total Area (in square ft)'),
       gr.inputs.Number(label='BHK (1-5)'),
       gr.inputs.Number(label='Bathrooms (1-5)'),
       gr.inputs.Dropdown(locations,label='Select Location')
     ]

oup = gr.outputs.Textbox()
gr.Interface(fn=predict_price, inputs=inp, outputs=oup, capture_session=True).launch()
```

The ‘inp’ variable contains a list of gradio inputs. Those are as follows,

1. Total Area in square ft
2. Number of BHK
3. Number of Bathrooms
4. Location (can be selected from a drop-down menu)

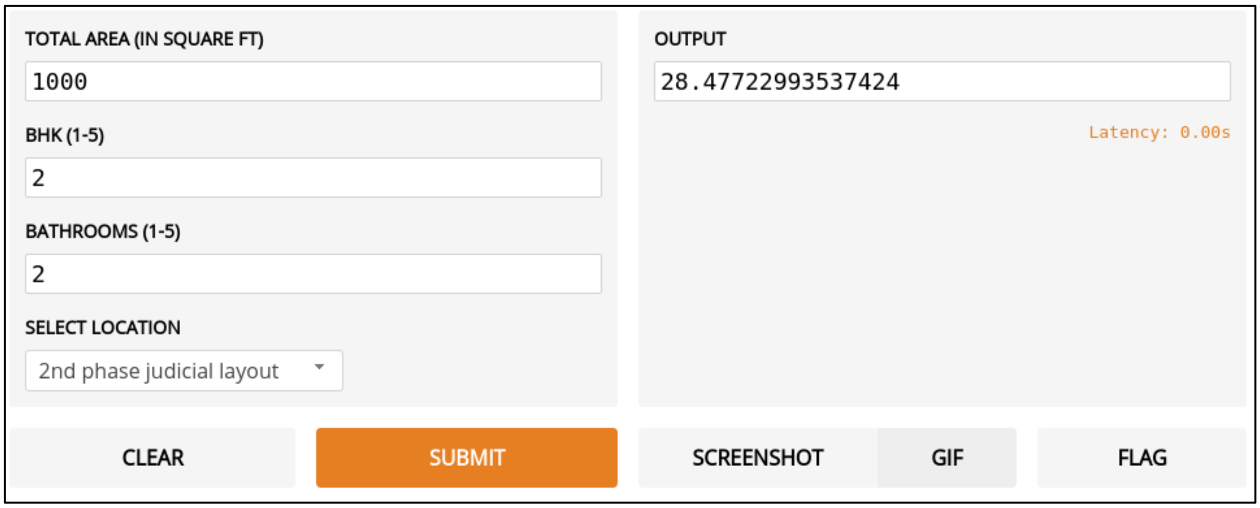
This list is passed as input to interface function of gradio library.

The ‘oup’ is output variable which will collect predicted price (in lakhs).

The interface function takes 3 main parameters. Those are as follows,

1. fn – function that is supposed to be used to perform operation of given inputs
2. inputs – variable to store inputs to be taken from user
3. outputs – variable to store output

Our GUI looks like this,



*The output is in lakhs.

References

1. Kaggle
2. Pandas Documentation
3. Scikit learn Documentation
4. Matplotlib Documentation
5. Gradio Documentation