Real Estate price estimation model

Firstly we will train a model and import it using pickle into a file. We then will write a python flask server to consume this pickle file and expose various http endpoints to various requests and make http get and post box.
For constructing the module we will be using following concepts:

- Feature Engineering
- Data Cleaning
- One Hot Encoding
- Outlier Detection
- Dimensionality Reduction
- GridSearchCV

1. Data Cleaning: Post importing all the needed libraries. Use groupby function to categorize according to area_type.

```
8]: df.groupby('area_type')['area type'].agg('count')

8]: area_type
    Built-up  Area          2418
    Carpet  Area              87
    Plot  Area              2025
    Super built-up  Area    8790
    Name: area_type, dtype: int64

 ]:
```

Assuming, Few features like area_type, society, balcony, availability aren't important(this may come in feature engineering). Data cleaning starts with handling null cells. Remove the rows with null values if they are in very less number comapred to total no. of rows.

```
In [15]: df1.isnull().sum()

Out[15]: location       1
         size          16
         total_sqft     0
         bath          73
         price          0
         dtype: int64

In [ ]:
```

As the total number of null values are quite less compared to total number of rows/records.
Hence, its safe to delete them. Now tokenizing the size column(as bedrooms and BHK both strings are used) which may result in model not being able to understand the column.

1.

In [21]:
```
df2['size'].unique()
```

Out[21]:
```
array(['2 BHK', '4 Bedroom', '3 BHK', '4 BHK', '6 Bedroom', '3 Bedroom',
       '1 BHK', '1 RK', '1 Bedroom', '8 Bedroom', '2 Bedroom',
       '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',
       '9 BHK', '9 Bedroom', '27 BHK', '10 Bedroom', '11 Bedroom',
       '10 BHK', '19 BHK', '16 BHK', '43 Bedroom', '14 BHK', '8 BHK',
       '12 Bedroom', '13 BHK', '18 Bedroom'], dtype=object)
```

In [22]:
```
df2['bhk'] = df2['size'].apply(lambda x: int(x.split(' ')[0]))
```

```
C:\Users\91940\AppData\Local\Temp\ipykernel_10404\1142257054.py:1: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
rsus-a-copy
  df2['bhk'] = df2['size'].apply(lambda x: int(x.split(' ')[0]))
```

tokenized size table will look as follows:

```
df2
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600	5.0	120.00	4
2	Uttarahalli	3 BHK	1440	2.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521	3.0	95.00	3
4	Kothanur	2 BHK	1200	2.0	51.00	2
...	...	...	...	...	...	...
13315	Whitefield	5 Bedroom	3453	4.0	231.00	5
13316	Richards Town	4 BHK	3600	5.0	400.00	4
13317	Raja Rajeshwari Nagar	2 BHK	1141	2.0	60.00	2
13318	Padmanabhanagar	4 BHK	4689	4.0	488.00	4
13319	Doddathoguru	1 BHK	550	1.0	17.00	1

13246 rows × 6 columns

In [27]:
```
df3.total_sqft.unique()
```

Out[27]:
```
array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)
```

now to remove or modify that 1133 - 1384 types of data in normal numbered columns, one must use concept of exception handling in python.

1.

```
Out[27]: array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
               dtype=object)

In [28]: def is_float(x):
             try:
                 float(x)
             except:
                 return False
             return True

In [29]: ##Now there are certain 1133-1384 types of values in total_sqft column, now to seperate and analyse them

         df3[~df3.total_sqft.apply(is_float)] ##~is negation of the statement it is predecessor of

Out[29]:
```

	location	total_sqft	bath	price	bhk
30	Yelahanka	2100 - 2850	4.0	186.000	4
122	Hebbal	3067 - 8156	4.0	477.000	4
137	8th Phase JP Nagar	1042 - 1105	2.0	54.005	2
165	Sarjapur	1145 - 1340	2.0	43.490	2
188	KR Puram	1015 - 1540	2.0	56.800	2
...	...	...	...	...	...
12975	Whitefield	850 - 1060	2.0	38.190	2
12990	Talaghattapura	1804 - 2273	3.0	122.000	3
13059	Harlur	1200 - 1470	2.0	72.760	2
13265	Hoodi	1133 - 1384	2.0	59.135	2
13299	Whitefield	2830 - 2882	5.0	154.500	4

190 rows × 5 columns

Now to take average of these ranges and replace them with it:

```
In [50]: def convert_sqft_to_num(x):
             tokens = x.split('-')
             if len(tokens)==2:
                 return (float(tokens[0])+float(tokens[1]))/2
             try:
                 return float(x)
             except:
                 return None

In [51]: convert_sqft_to_num('2600')

Out[51]: 2600.0

In [52]: df5 = df3.copy()
         df5['total_sqft'] = df5['total_sqft'].apply(convert_sqft_to_num)
```

2. Feature Engineering and Dimensionality reduction techniques: An important part of machine learning that is process of using domain knowledge about the data to create features that work for ML algorithms. Choice of features has a significant impact on performance of model. Feature Engineering involves:
   - Identifying the features that are most relevant to the problem at hand.
   - Extracting and/or constructing new features from existing ones(one hot encoding or dummies)
   - Selecting the most useful features to input into the model.

- Feature Engineering involves a wide range of technniques, including:
  - Encoding categorial Variables
  - Normalizing and Standardizing numerical variables.
  - Handling missing values.

- Creating new features by combining and transforming existing ones.

Including a extra column of price per sqft

```
In [59]: df6 = df5.copy()

In [60]: df6['price_per_sqft'] = df6['price']*100000/df6['total_sqft']

In [61]: df6
Out[61]:
```

	location	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	1056.0	2.0	39.07	2	3699.810606
1	Chikka Tirupathi	2600.0	5.0	120.00	4	4615.384615
2	Uttarahalli	1440.0	2.0	62.00	3	4305.555556
3	Lingadheeranahalli	1521.0	3.0	95.00	3	6245.890861
4	Kothanur	1200.0	2.0	51.00	2	4250.000000
...	...	...	...	...	...	...
13315	Whitefield	3453.0	4.0	231.00	5	6689.834926
13316	Richards Town	3600.0	5.0	400.00	4	11111.111111
13317	Raja Rajeshwari Nagar	1141.0	2.0	60.00	2	5258.545136
13318	Padmanabhanagar	4689.0	4.0	488.00	4	10407.336319
13319	Doddathoguru	550.0	1.0	17.00	1	3090.909091

13246 rows × 6 columns

As location is a categorial column, but the number of categories is 1304

4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250.00000

```
In [36]: len(df5.location.unique())
Out[36]: 1304
```

hence its not practical to create dummy columns for it as this may lead to huge increase in column numbers. So instead lets categorize areas having less than 10 houses(in the list) as 'other areas'.

For that following procedure:

13246 rows × 6 columns

In [80]:
```
len(df6.location.unique())
```
Out[80]: 242

In [63]:
```
len(df6.location.unique())
```
Out[63]: 1304

In [65]:
```
df6.location = df6.location.apply(lambda x: x.strip())
```

In [70]:
```
location_stats = df6.groupby('location')['location'].agg('count').sort_values(ascending = True)
```

In [71]:
```
location_stats
```
Out[71]:
```
location
1 Annasandrapalya      1
Kudlu Village,         1
Kumbhena Agrahara      1
Kuvempu Layout         1
LIC Colony             1
                    ...
Thanisandra          236
Kanakpura Road       266
Electronic City      304
Sarjapur  Road       392
Whitefield           535
Name: location, Length: 1293, dtype: int64
```

In [72]:
```
len(location_stats[location_stats<=10])
```
Out[72]: 1052

In [74]:
```
location_stats_less_than_10 = location_stats[location_stats<=10]
location_stats_less_than_10
```
Out[74]:
```
location
1 Annasandrapalya       1
Kudlu Village,          1
Kumbhena Agrahara       1
Kuvempu Layout          1
LIC Colony              1
                       ..
Kalkere                10
Naganathapura          10
Sector 1 HSR Layout    10
Basapura               10
BTM 1st Stage          10
Name: location, Length: 1052, dtype: int64
```

In [77]:
```
df6.location = df6.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)
```

In [78]:
```
df6
```
Out[78]:

	location	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	1056.0	2.0	39.07	2	3699.810606
1	Chikka Tirupathi	2600.0	5.0	120.00	4	4615.384615
2	Uttarahalli	1440.0	2.0	62.00	3	4305.555556
3	Lingadheeranahalli	1521.0	3.0	95.00	3	6245.890861
4	Kothanur	1200.0	2.0	51.00	2	4250.000000

3. Outlier Removal: Outliers are data points that are significantly different from the rest of the data in a dataset. They can have a significant impact on the results of statistical analyses and modeling, and can even distort the overall pattern of the data. Hence it is necessary to remove the outliers.
    1. Outliers can affect the accuracy of statistical analyses.
    2. Outliers can distort overall pattern of the data
    3. They are usually result of mistakes or errors
    4. they can be caused by unusual or extreme events

   Now analysing and asking expert the minimum sqft per bhk value (supposing its 300), theen removing every record whose total_sqft per bhk is less than 300 as this maybe an anomaly or error.

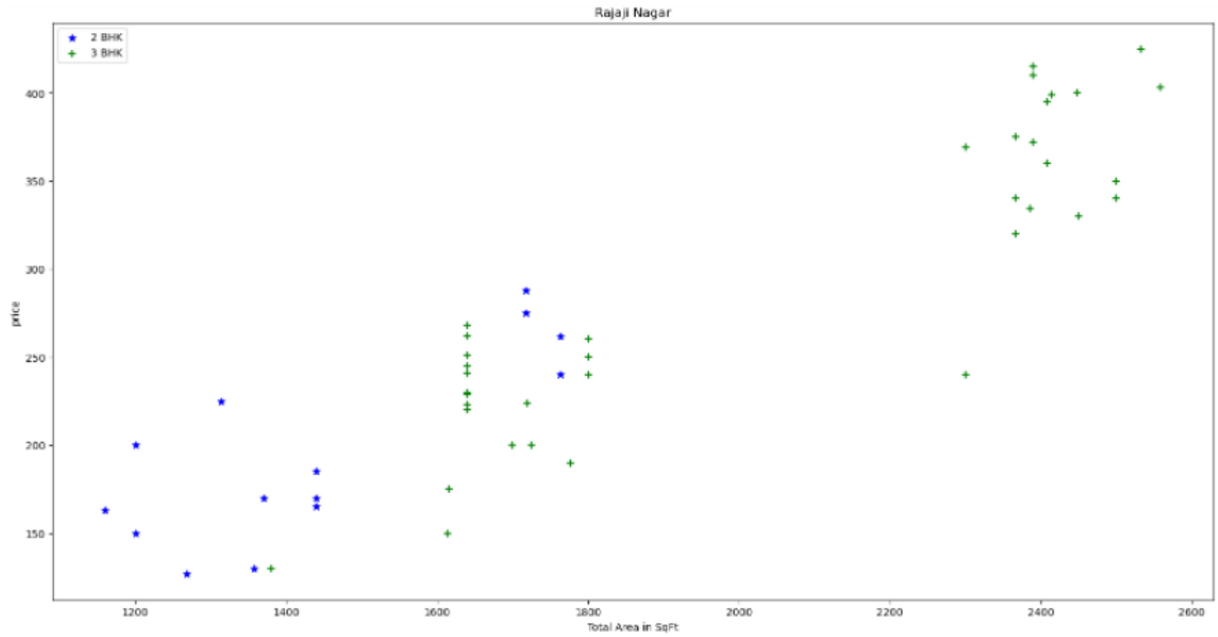
   We need to filter out extreme cases of price per sqft per location as well. Hence removing records whose price per sqft is not between (mean-std,mean+std) per location. Method is as follows:

In [86]:
```
##removing the price_per_sqft outliers

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

df8 = removing_outliers_pps(df7)
```

In [87]:
```
df8
```

In some cases 2 bedroom flat of same sqft area has more price than 3 bedroom flat, hence we need to remove these inefficiencies. To denote such inefficiencies lets plot its scatter plot.

```
47]: def plot_scatter_plot(df,location):
    bhk2 = df[(df.location == location) & (df.bhk == 2)]
    bhk3 = df[(df.location == location) & (df.bhk == 3)]
    matplotlib.rcParams['figure.figsize'] == (15,30)
    plt.scatter(bhk2.total_sqft, bhk2.price, color = 'blue',marker = '*', label = '2 BHK', s = 50)
    plt.scatter(bhk3.total_sqft, bhk3.price, color = 'green', marker =  '+',  label = '3 BHK', s = 50)
    plt.xlabel('Total Area in SqFt')
    plt.ylabel('price')
    plt.title(location)
    plt.legend()

plot_scatter_plot(df8, 'Rajaji Nagar')
```



Now to remove such outliers we create dictonaries for each bhk in each location with its mean, std and count. And we condition the points to remain in the data set it's price should more than mean of (it's bhk-1).

We should also remove properties where for same location, the price of (for example) 3 bedroom apartment is less than 2 bedroom apartment (with same square ft area). What we will do is for a given location, we will build a dictionary of stats per bhk, i.e.

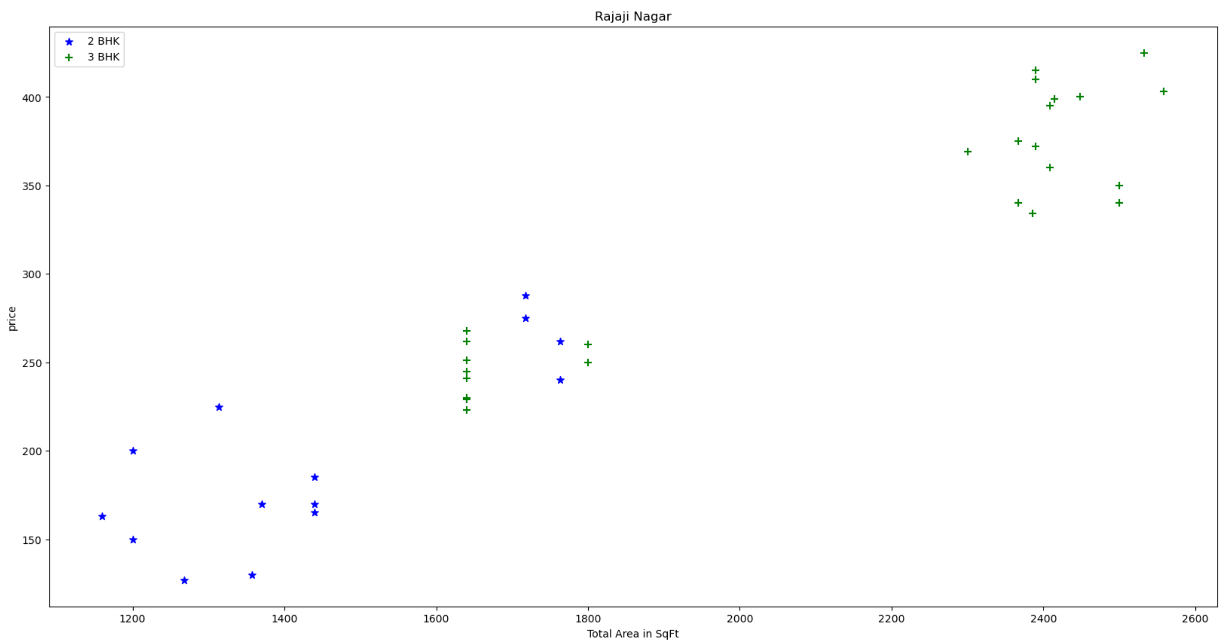
```
{
    '1' : {
        'mean': 4000,
        'std: 2000,
        'count': 34
    },
    '2' : {
        'mean': 4300,
        'std: 2300,
        'count': 22
    },
}
```

Now we can remove those 2 BHK apartments whose price_per_sqft is less than mean price_per_sqft of 1 BHK apartment

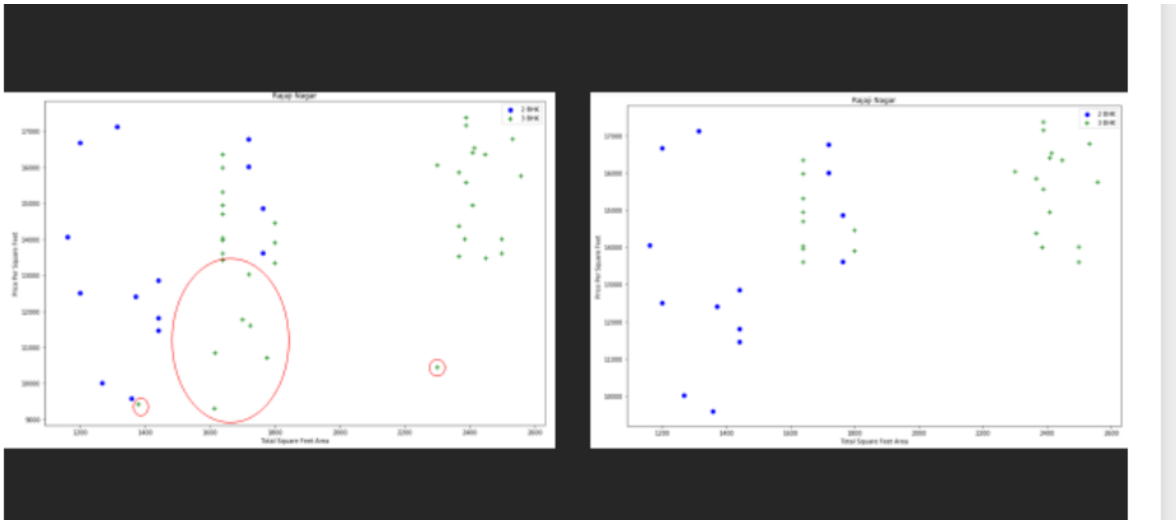
Now we can remove those 2 BHK apartments whose price_per_sqft is less than mean price_per_sqft of 1 BHK apartment

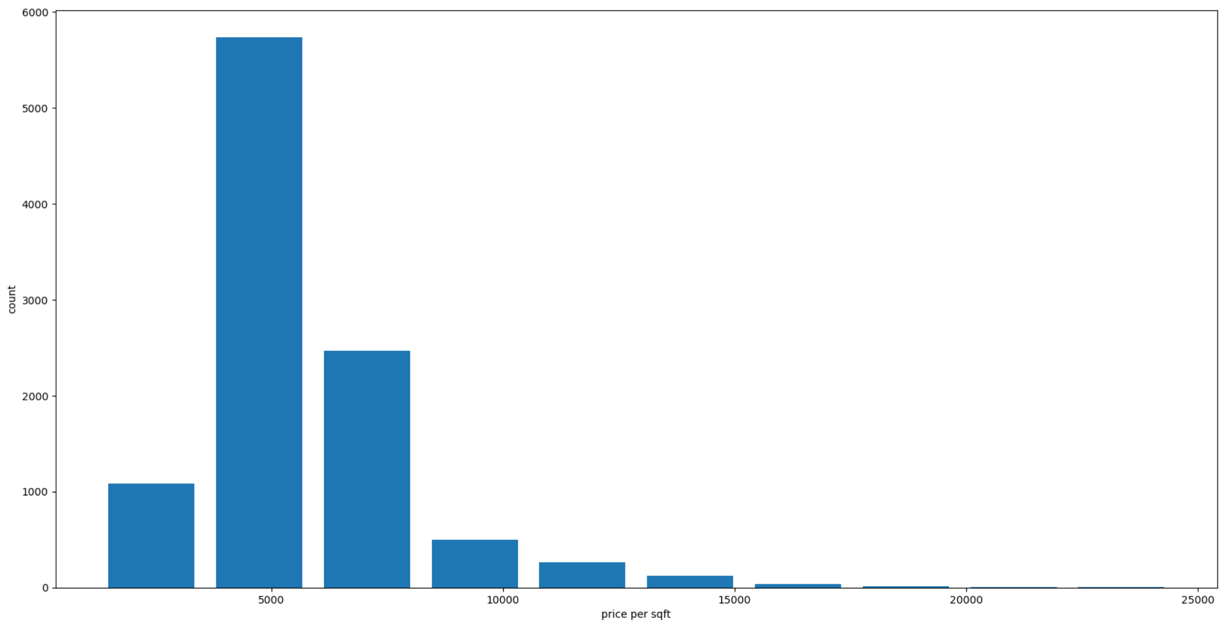
In [38]:
```
def remove_bhk_outliers(df):
    exclude_indices = np.array([])
    for location, location_df in df.groupby('location'):
        bhk_stats = {}
        for bhk, bhk_df in location_df.groupby('bhk'):
            bhk_stats[bhk] = {
                'mean': np.mean(bhk_df.price_per_sqft),
                '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
```

Out[38]: (7317, 7)



As we can see the data points of 3 bhk whose sqft area is same as 2 bhk but its price is less than the latter are removed.





As you can see now the price per sqft is a normal distribution(bell curve).

Now analysing the bathroom feature:

Considering your real estate expert stated that bathrooms > bhk + 2 is unusual and hence can be considered as an outlier

In [82]:
```
##now for bathroom outliers

df10 = df9[~(df9.bath>df9.bhk+2)]
```

In [83]:
```
df10
```

Out[83]:

location	total_sqft	bath	price	bhk	price_per_sqft

As the outlier removal is completed we can drop the usless features post this step, which is price_per_sqft

4. Model building:

   1. Applying One Hot Encoding to location:

```
In [96]: dummies = pd.get_dummies(df10.location)
         dummies
```

Out[96]:

5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar	9th Phase JP Nagar	...	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield	Yelachenahalli	Yelahanka	Yelahanka New Town	Yelenahalli	Yeshwanthpur
0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

```
In [98]: df11 = pd.concat([df10, dummies.drop('other', axis = 'columns')], axis = 'columns')
```

   We only need n-1 columns so removing the 'other' column.

   2. Following general procedure of differentiating data set into features and result datasets

```
In [98]: df11 = pd.concat([df10, dummies.drop('other', axis = 'columns')], axis = 'columns')

In [99]: df12 = df11.drop('location', axis = 'columns')

In [100]: ##model selection

          X = df12.drop('price', axis = 'columns')

In [101]: y = df12.price

In [ ]: 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)
```

   and spliting train test datasets.

   3. Using Linear Regression

```
In [105]: ##Using Logistic Regression model

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

Out[105]: 0.9014569230701958
```

4. Using cross_val_score

```
[107]: ##using Cross validation
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

cv = ShuffleSplit(n_splits = 5, test_size = 0.2, random_
cvs = cross_val_score(LinearRegression(), X, y, cv = cv)
```

```
[108]: cvs.mean()
```

```
[108]: 0.9258219539389323
```

5.

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

Using gridsearchcv to include the regression models and there parameters on which we are going to run our train, test data to find the best model.

6.

```
        })

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

find_best_model_using_gridsearchcv(X,y)
```

```
In [60]: 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]
```

```
In [61]: predict_price('1st Phase JP Nagar',1000, 2, 2)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearReg
ression was fitted with feature names
  warnings.warn(

Out[61]: 83.8657025831206
```

```
In [62]: predict_price('1st Phase JP Nagar',1000, 3, 3)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearReg
ression was fitted with feature names
  warnings.warn(

Out[62]: 86.0806228498683
```

7. It is found that that linear regression is the best model with parameters as shown in figure below:

```
ut[59]:
```

	model	best_score	best_params
0	linear_regression	0.847796	{'normalize': False}
1	lasso	0.726833	{'alpha': 2, 'selection': 'random'}
2	decision_tree	0.709552	{'criterion': 'friedman_mse', 'splitter': 'best'}

```
n [60]: def predict_price(location,sqft,bath,bhk):
            loc_index = np.where(X.columns==location)[0][0]
```

8.


```
        })

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

find_best_model_using_gridsearchcv(X,y)
```

```
In [60]: 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]
```

```
In [61]: predict_price('1st Phase JP Nagar',1000, 2, 2)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearReg
ression was fitted with feature names
  warnings.warn(

Out[61]: 83.8657025831206
```

```
In [62]: predict_price('1st Phase JP Nagar',1000, 3, 3)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearReg
ression was fitted with feature names
  warnings.warn(

Out[62]: 86.0806228498683
```


defining a function to predict the the price given the loaction, sqft, bath, bhk

9.

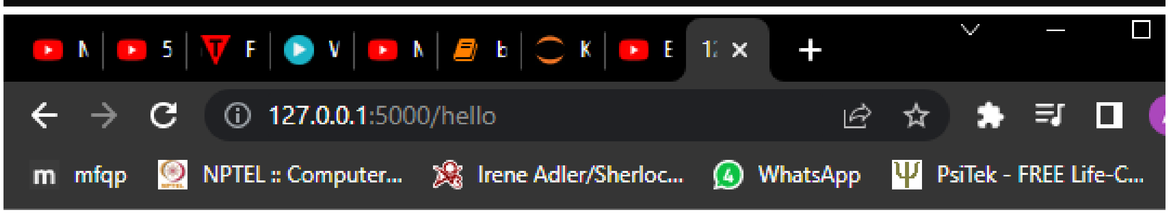
```
In [65]: import pickle
         with open('banglore_home_prices_model.pickle','wb') as f:
             pickle.dump(lr_clf,f)

In [66]: import json
         columns = {
             'data_columns' : [col.lower() for col in X.columns]
         }
         with open("columns.json","w") as f:
             f.write(json.dumps(columns))

In [ ]:
```

Exporting the trained model into a pickle file. And the Data is cleansed and sent through model to train it.

5. Python Flask Server: Writing a python program that can serve http request and respond to it. Basically it is a web framework that provides libraries to build light weight web applications in python and helps in creating servers. It is a micro framework. Firstly, import the flask module and create an app as shown below:

```python
from flask import flask, request, jsonify

app = Flask(__name__)

if __name__== "__main__":
    print("Starting Python Flask Server for home price prediction")
    app.run()
```

```
\__init__.py)
PS C:\Users\91940\Desktop\Data Science\Kaggle proojects\Real_estate_price_predictor\server> python flask_server.py
Starting Python Flask Server for home price prediction
 * Serving Flask app 'flask_server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [03/Jan/2023 22:27:32] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [03/Jan/2023 22:27:33] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [03/Jan/2023 22:28:08] "GET /hello HTTP/1.1" 200 -
```



Hi

Now starting with a function that will return all the location names and importing it in the flask app(using the jsonify method). Creating new file 'util' that will contain all the core routines, one of which will be get_location_name. Which will read columns.json and return the names of location from 4th columns as first 3 are just the features. Firstly, preparing the utility python file i.e. util.py. Importing all the neceaary libraries of pickle, json and numpy. Then initialising the variables of locations, data columns and model as none.

5.
```python
__locations = None
__data_columns = None
__model = None
```

initialising the necessary functions of a. get_estimated_price(with parameters location, sqft, bhk, bath) also considering the exceptions as follows(for the location).

```python
def get_estimated_price(location,sqft,bhk,bath):
    try:
        loc_index = __data_columns.index(location.lower())
    except:
        loc_index = -1

    x = np.zeros(len(__data_columns))
    x[0] = sqft
    x[1] = bath
    x[2] = bhk
    if loc_index>=0:
        x[loc_index] = 1

    return round(__model.predict([x])[0],2)
```

Now initialising the 2nd function, to load both locations and trained pickle model into global variables initialised above

```python
def load_saved_artifacts():
    print("loading saved artifacts...start")
    global  __data_columns
    global __locations

    with open("./artifacts/columns.json", "r") as f:
        __data_columns = json.load(f)['data_columns']
        __locations = __data_columns[3:]  # first 3 columns are sqft, bath, bhk

    global __model
    if __model is None:
        with open('./artifacts/banglore_home_prices_model.pickle', 'rb') as f:
            __model = pickle.load(f)
    print("loading saved artifacts...done")
```

5. Functions to return locations and data columns with some examples of prediction

```python
def get_location_names():
    return __locations

def get_data_columns():
    return __data_columns

if __name__ == '__main__':
    load_saved_artifacts()
    print(get_location_names())
    print(get_estimated_price('1st Phase JP Nagar',1000, 3, 3))
    print(get_estimated_price('1st Phase JP Nagar', 1000, 2, 2))
    print(get_estimated_price('Kalhalli', 1000, 2, 2)) # other location
    print(get_estimated_price('Ejipura', 1000, 2, 2))  # other location
```

Now importing this Util.py file into flask_server.py along with necessary flask libraries as follows

```python
from flask import Flask, request, jsonify
import util

app = Flask(__name__)
```

@app.route for the app routing which means mapping the URLs to a specific fucntion that will handle the logic for that URL, along with GET method for get_location_names function and GET and POST methods for predict_home_price function.

5.

```
@app.route('/get_location_names',
methods=['GET'])
def get_location_names():
    response = jsonify({
        'locations': util.
        get_location_names()
    })
    response.headers.add
    ('Access-Control-Allow-Origin',
    '*')

    return response


@app.route('/predict_home_price',
methods=['GET', 'POST'])
def predict_home_price():
    total_sqft = float(request.form
    ['total_sqft'])
    location = request.form
    ['location']
    bhk = int(request.form['bhk'])
    bath = int(request.form['bath'])

    response = jsonify({
        'estimated_price': util.
        get_estimated_price
        (location, total_sqft, bhk,
        bath)
    })
    response.headers.add
    ('Access-Control-Allow-Origin',
    '*')

    return response
```

request.form is an object that contains all the data sent from the client to server.

```
if __name__ == "__main__":
    print("Starting Python Flask
    Server For Home Price
    Prediction...")
    util.load_saved_artifacts()
    app.run()
```

loading saved artifacts function in util.py file will import both the required pickle and json file. And runs desired app when we run the file flask_server.py in terminal and copy paste the url in postman application that will run it as we require.

Challenges Faced:

Initially I was unable to find the correct dataset as I couldn't understand what the problem statement meant by content dataset, but taking advice of Mr. Solomon Eko Sir I was able to choose this real estate dataset. After choosing a dataset, performing EDA on the obtained dataset was challenging but with the help of few resources I was able to carry it out as well. Post which the decision as to which model would be ideal to be used on the cleaned dataset was an issue, which I tackled using gridsearchCV. Created a flask to deploy the server on postman API.

Key Findings:

The factors affecting the prices of real estates. Various steps the dataset needed to be used to be able to predict prices of future real estate(if the necessary variables are present). Correct model and its parameters to be found with the help of GridSearchCV. Creating and deploying a server using python flask.