# HOUSE PRICE PREDICTION

After the demonetization in India the prices of the houses in the real estate field dropped a lot especially in metropolitan cities like Mumbai, Chennai, Banglore and Hyderabad. This pose a challenge to predict the house prices in the different areas of the city.

Considering Banglore city. The attributes of the dataset are:

1. Area_type
2. Location, size
3. Total square feet
4. Number of bathrooms
5. Bed rooms
6. Society,
7. Availability,
8. Balcony
9. Price

It's time for the **Model**:

By taking the leverage of the total sqft, number of bedrooms and bath, one can classify the houses depending on their location, price and type of bed & bath. The classification techniques like linear regression, decision tree are appropriate for the dataset. The appropriate analysis from the dataset are to predict the price of the house and perform analysis on the two bed room vs three bed room in specific locations.

```
#Loading the data into the dataframe.
df = pd.read_csv("House_Data.csv")
df.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 |

```
# Handling the non-uniform data of total_sqft column using the below function.
def average_sqft(x):
  values = x.split('-')
  if len(values) == 2:
    return (float(values[0]) + float(values[1]))/2
  try:
    return float(x)
  except:
    return None
```

```
#Creating a new data frame and calculating the price per square.ft.
df4 = df3.copy()
df4['price_per_sqft'] = df4['price']*100000 / df4['total_sqft']
df4.head()
```

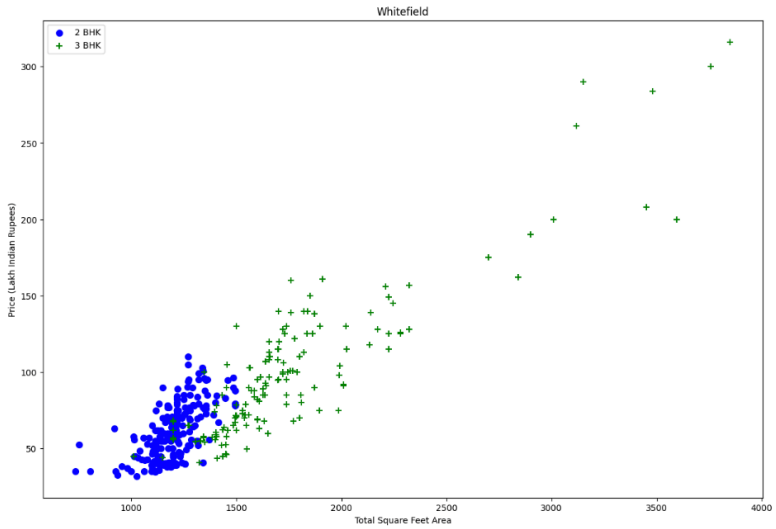
| | location | size | total_sqft | bath | price | bedroom | price_per_sqft |
|---|---|---|---|---|---|---|---|
| 0 | Electronic City Phase II | 2 BHK | 1056.0 | 2.0 | 39.07 | 2 | 3699.810606 |
| 1 | Chikka Tirupathi | 4 Bedroom | 2600.0 | 5.0 | 120.00 | 4 | 4615.384615 |
| 2 | Uttarahalli | 3 BHK | 1440.0 | 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 |

Data pre-processing in which elimination of nulls and anamolies took place enhanced the data quality and suits for training the model.

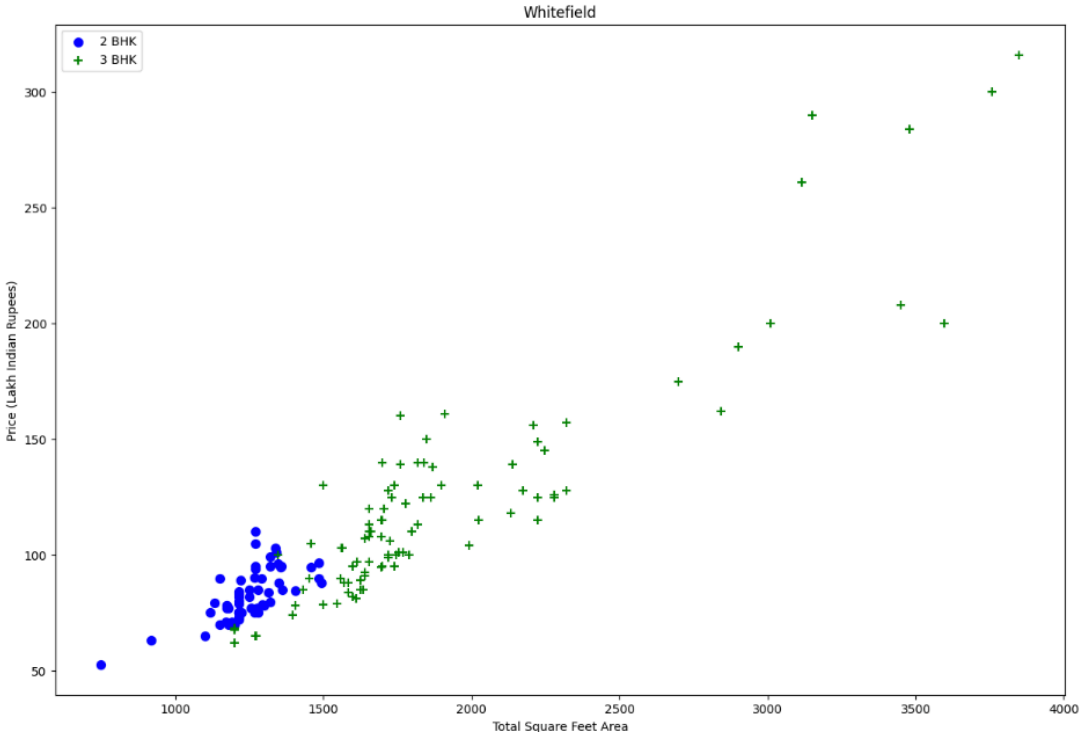
```
# Function to remove price per sqft outliers.
def eliminate_pps(df):
    df_out = pd.DataFrame()
    for key, subdf in df.groupby('location'):
        m = np.mean(subdf.price_per_sqft)
        st = np.std(subdf.price_per_sqft)
        reduced_df = subdf[(subdf.price_per_sqft>(m-st)) & (subdf.price_per_sqft<=(m+st))]
        df_out = pd.concat([df_out,reduced_df],ignore_index=True)
    return df_out
df6 = eliminate_pps(df5)
df6.shape
```

```
(10241, 7)
```

**Data With Anomalies:**



This dataset is more suitable to get more insights of the house prediction and the demand for the two bedroom, three bedroom layouts in the city, which made to opt for this specific analysis. After performing the data cleaning by removing the null values and anomalies in the dataset the data looks linear.




Since the data looks linear. To determine the best algorithm and best parameters, Gridsearchcv is used.

```
[38] # GridSearchCV it will not only give best algorithm, for that particular algorithm, it will also tell you the best parameters
     # This is called hyper parameter tuning.
     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):
         algorithms = {
             'linear_regression' : {
                 'model' : LinearRegression(),
                 'params' : {
                     'fit_intercept': [True, False],
                     'copy_X': [True, False],
                     'n_jobs': [1, 2, 4, -1],
                     'positive': [True, False]
                 }
             },
             'lasso' : {
                 'model' : Lasso(),
                 'params' : {
                     'alpha' : [1, 2],
                     'selection' : ['random', 'cyclic']
                 }
             },
             'decision_tree' : {
                 'model' : DecisionTreeRegressor(),
                 'params' : {
                     'criterion' : ['mse', 'friedman_mse'],
                     'splitter' : ['best', 'random']
                 }
             }
         }
         # Shufflesplit randomly shuffles sample for cross validation
         scores = []
         cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
         for algo_name, config in algorithms.items():
             gs = GridSearchCV(config['model'], config['params'], cv=cv, return_train_score=False)
             gs.fit(X,Y)
             scores.append({
                 'model' : algo_name,
                 'best_score' : gs.best_score_,
                 'best_params' : gs.best_params_
             })

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

     find_best_model_using_gridsearchcv(X, Y)
```

Linear Regression stands out as a best model:

| | model | best_score | best_params |
|---|---|---|---|
| 0 | linear_regression | 0.821725 | {'copy_X': True, 'fit_intercept': True, 'n_job... |
| 1 | lasso | 0.699286 | {'alpha': 1, 'selection': 'random'} |
| 2 | decision_tree | 0.774266 | {'criterion': 'friedman_mse', 'splitter': 'best'} |

The data mining technique linear regression was implemented to classify different regions of the city and to predict the house price with Jupiter notebook, python. 80% of the data from the data set is used to train and the remaining data for evaluating the model. The accurate prediction of the house price was about 85% that showed the model is an approved model for the house prediction. Different types of parameters, optimizers and loss functions could improve the analysis.

```
[27] # Building the model using test train split.
     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)
```

```
[28] # Applying linear regression model.
     from sklearn.linear_model import LinearRegression
     lr_clf = LinearRegression()
     lr_clf.fit(X_train, Y_train)
     lr_clf.score(X_test, Y_test)

     0.8452277697873658
```

Once the model is ready, the we can export the model and necessary columns as a pickle file.

It's time to prepare a **Server**:

Using POST, GET methods develop http endpoints using Flask and evaluate these methods using PostMan.

Troubleshooting tips:

- If port already in use error, try with different ports
- Check what is using that port : lost -i:portnumber.
- If it is 'Control Center". Control Centre on Mac gives you quick access to key macOS settings such as volume, brightness, Wi-Fi or Focus — and indicates when your Mac is using a camera or microphone. You can customise Control Centre to add other items, such as accessibility shortcuts or fast user switching.
- If you wanted to kill the process, you can do that as well.

**Python-Flask:**

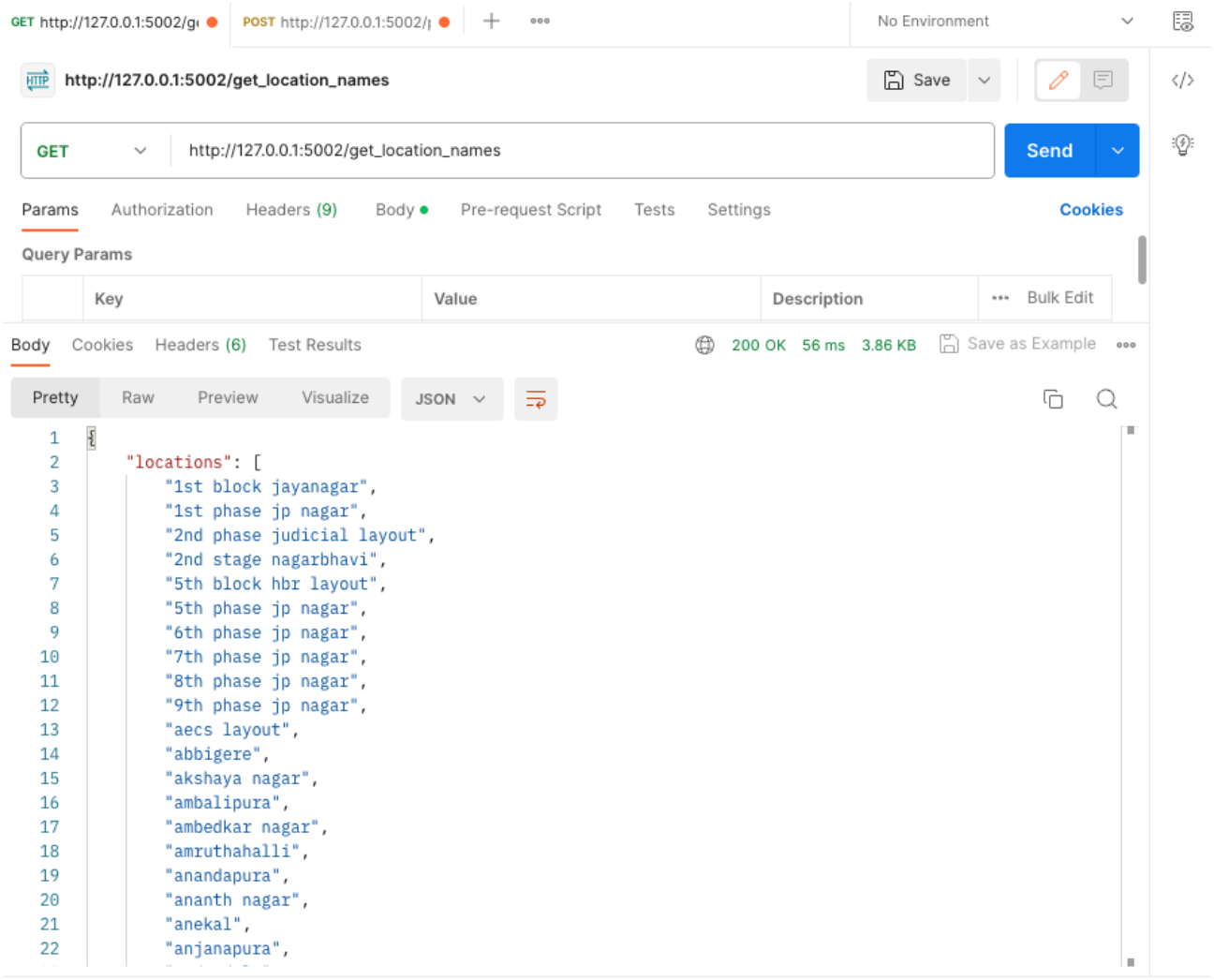
```
@app.route( rule: '/get_location_names', methods=['GET', 'POST'])
def get_location_names():
    response = jsonify({
        'locations': util.get_location_names()
    })
    response.headers.add( _key: 'Access-Control-Allow-origin', _value: '*')
    return response


new *
@app.route( rule: '/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( _key: "Access-control-allow-origin", _value: '*')
    return response
```

**POSTMAN:**

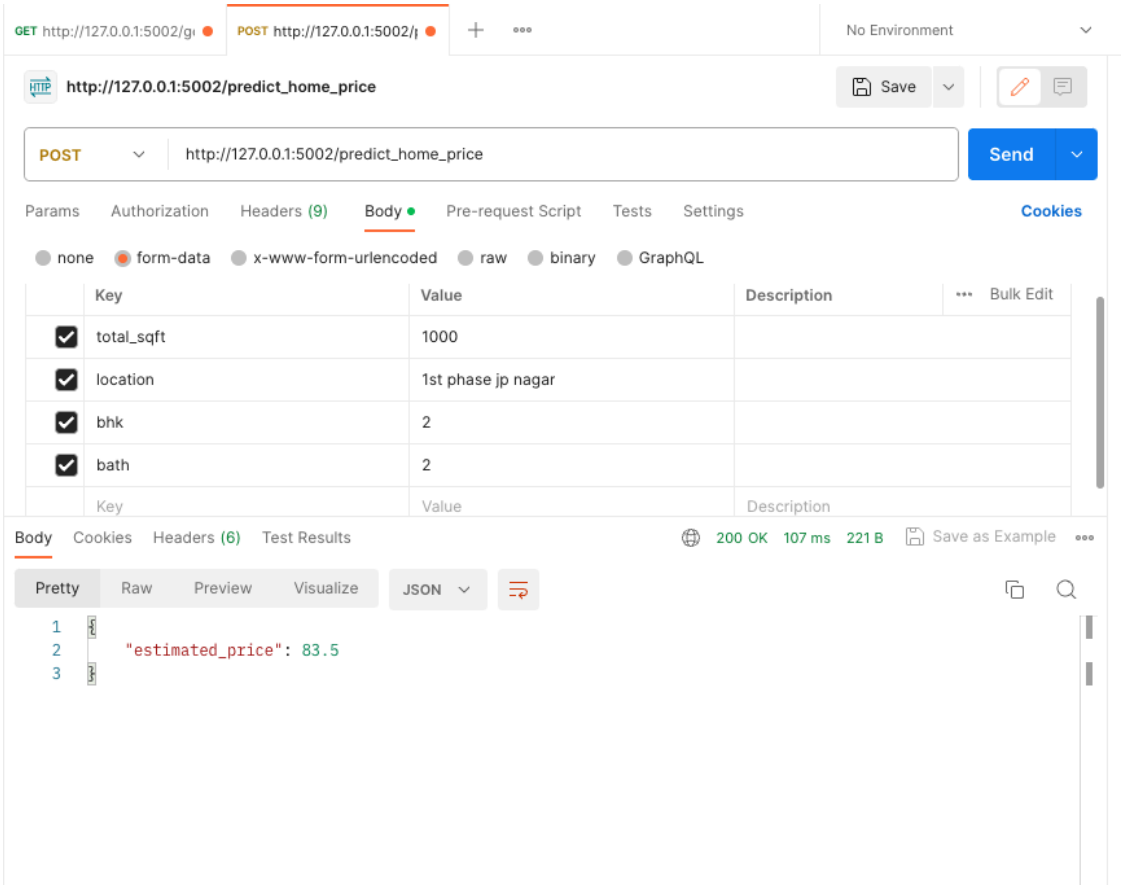
**GET** Method Evaluation:

Testing the get method using postman. It gave all the list of locations that are in the dataset as well as Banglore. This assures my end point is working as expected.



**POST** Method Evaluation:

Testing the post method using postman. It takes 4 different parameters that we have to pass in the body as a key value pair which in return gave the price predicted This assures my end point is working as expected.



Its time for the **Client:**

Creating a front-end user interface (UI) that combines HTML, CSS, and JavaScript for a price prediction project is a great way to present your work effectively. Below is a sample snippet of a front-end JavaScript code for price prediction.

```
function onPageLoad() {
    console.log( "document loaded" );
    var url = "http://127.0.0.1:5002/get_location_names"; // Use this if you are NOT using nginx which is first 7 tutorials
    $.get(url,function(data, status) {
        console.log("got response for get_location_names request");
        if(data) {
            var locations = data.locations;
            var uiLocations = document.getElementById("uiLocations");
            $('#uiLocations').empty();
            for(var i in locations) {
                var opt = new Option(locations[i]);
                $('#uiLocations').append(opt);
            }
        }
    });
  }

window.onload = onPageLoad;
```

**Sample Output:**

**Parameters = 1000sqft, 2 bhk, 2 bath, 1st phase Jp nagar**