House Price Prediction

Submitted by: - Niranjan Ganesan, Student ID - 170963389

Group size: 4

Table of individual contribution by each member of my group, based on my subjective opinion:

Student	Effort
Niranjan Ganesan	25%
Priyank Mishra	25%
Rohit Shindhe	25%
Sahil Gupta	25%

Contents

1. Introduction and Background

Housing prices are an important reflection of the economy, and price ranges are of great interest for both buyers and sellers. Buying a property is an important decision in a person's life. A house is the most valuable asset one will ever own. In this project, house prices for a given location in the city of Bangalore will be predicted given explanatory variables that cover many aspects of residential houses. As continuous house prices, they will be predicted with various regression techniques including Linear Regression, Ridge, Kernel Ridge, Random Forest regression, XGBoosting Regressor and Gradient Boosting Regressor. We compare and assess the performance of these methods by examining R^2 score and RMSE. A cross validation is done to assess the achieved accuracy and thereby examine the scope to improve the accuracy score.

2. Project Aims

The aim of project is to build a predictive model that can predict the price of the house using attributes that shows high correlation value to the output price. Another important aspect of this project is to examine the importance of each predictor in explaining price variation for a given set of housing attributes. If a housing feature is highly correlated with price does it actually means that it influences the increase in price.

The primary approach of this project is to prepare the collected raw data and perform exploratory analysis to understand how well the data is related. Upon visualization and applying few statistical techniques the collected data is pre-processed. Pre-processing includes dealing with missing values, dealing with outliers and cleaning the data to the required format. Once pre processing is done the cleaned data is feature engineered such that its ready for building machine learning models. Then, the data is split into two parts where the first part trains the model and produces an inferred function and the second part of the data tests the accuracy of the model built. Both the datasets are subject to various regression models. Finally, the accuracy score is evaluated across the models and the best model is chosen based for the problem.

3. Literature Review

There are many factors that influence the house price. Prices of real estate property are related to the economic conditions of the state [1]. When the economy escalates, construction and employment in housing Sector grows rapidly to meet excess demand of Real Estate prices. Research [2] states the primary influencers are physical condition, idea and area. Physical conditions include span of the house, quantity of rooms, accessibility of kitchen and garage, accessibility of nursery, the zone of land and structures and the age of the house [3]. Area is a significant factor in forming the cost of a house. This is on grounds that the area decides the common land price [4]. Moreover, area additionally decides the accessibility to nearby schools, grounds, emergency clinics and wellbeing focuses such as shopping centers, restaurants or even a delightful scenery [5] [6]. Building a predictive model to estimate the house price requires exhaustive information and extensive exploration [7]. Prediction models in [7] includes Linear

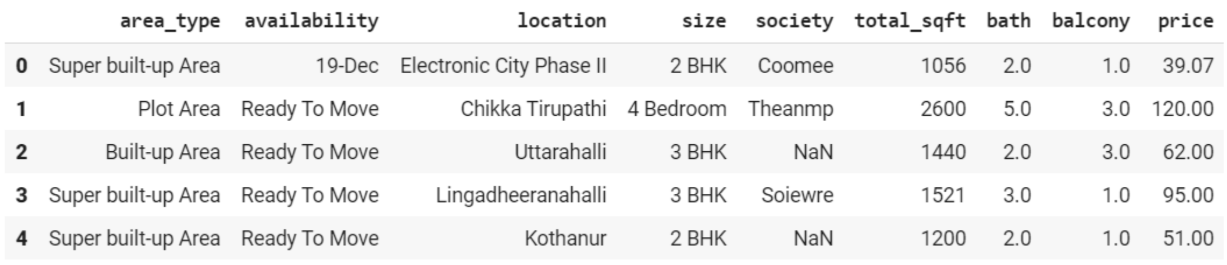
Regression, Support Vector Machine, K-Nearest Neighbors and Random Forest Regression. In [8] epicurean value model and ANN model predicts the house prices. Work in [9] uses classifiers to predict the house prices, collecting data from Multiple Listing Service, verifiable home loans and government funded school evaluations. In [10], the author has considered the most macroeconomic parameters that affect the house price variations using back propagation neural network(BPN) and radial biases function network(RBF) to establish the nonlinear model for real estate's price variation prediction. Overall, accurate forecasting of the evolution path of house prices can be a useful tool both to house market participants and monetary policy authorities.

4. Data retrieval

The data utilized for the analysis was collected from Kaggle. Data retrieval was easy as datasets were available and ready for user download. The dataset obtained was:

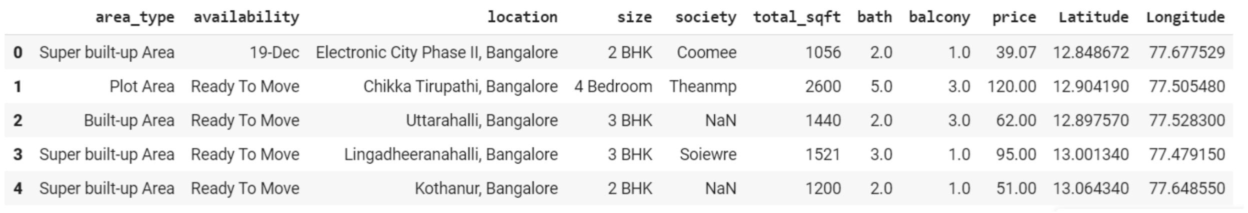
Bengaluru_House_Data.csv: Bangalore house price data containing the details of sold house prices. Features: "area_type", "availability", "location", "size", society", "total_sqft", "bath", "balcony", "price".

Prior to any data manipulation, it is essential to extract the data and transform it into a format that can be easily used in processing stage. The dataset is represented as a Pandas DataFrame and below is the sample of the main dataset containing house price details,



	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

Using the above dataset, we decided to collect more features. So, we collected the latitude and longitude data using 'location' feature for better visual representation. Upon investigating we discovered that without city name incorrect coordinate details were obtained. So, we manually added the city name "Bangalore" to each 'location' record and collected the coordinate details. Likewise, after collecting the coordinates the dataset is represented as a Pandas Dataframe and below is the sample of collected records,



	area_type	availability	location	size	society	total_sqft	bath	balcony	price	Latitude	Longitude
0	Super built-up Area	19-Dec	Electronic City Phase II, Bangalore	2 BHK	Coomee	1056	2.0	1.0	39.07	12.848672	77.677529
1	Plot Area	Ready To Move	Chikka Tirupathi, Bangalore	4 Bedroom	Theanmp	2600	5.0	3.0	120.00	12.904190	77.505480
2	Built-up Area	Ready To Move	Uttarahalli, Bangalore	3 BHK	NaN	1440	2.0	3.0	62.00	12.897570	77.528300
3	Super built-up Area	Ready To Move	Lingadheeranahalli, Bangalore	3 BHK	Soiewre	1521	3.0	1.0	95.00	13.001340	77.479150
4	Super built-up Area	Ready To Move	Kothanur, Bangalore	2 BHK	NaN	1200	2.0	1.0	51.00	13.064340	77.648550

5. Data Representation

Python3 was chosen as the main language to conduct our analysis as it has very useful and powerful libraries and packages for our analytical needs. Specifically, we made use of the following libraries:

- **Pandas:** A prominent open source data manipulation library. It simplifies the retrieval of data from external sources and provides high performance, easy-to-use data structures and data analysis tools. For this project, the main Pandas functionalities utilized are the DataFrame and Series objects.
- **Numpy:** A fundamental package for scientific computing. It provides support for large multidimensional arrays with high-level efficient math functions for operations on these arrays.
- **Matplotlib:** A basic Python package with a variety of functions for data plotting and visualization such as plots, maps, charts etc. It greatly complements Pandas by providing visuals to better explore and understand the data.
- **Seaborn:** A more statistical library with sophisticated visualizations such as heatmaps (used in project) and joining several maps into one.
- **Scikit-learn:** A Python package that provides solid implementations of a range of machine learning algorithms for classification, regression and clustering such as k-means., etc. It naturally complements other libraries like NumPy.
- **Folium:** A powerful data visualization library in Python to visualize geospatial data. The coordinate details are plotted on the map for geospatial representation.
- **Geopy:** A geocoding service in Python to locate the coordinates of addresses, cities, countries and landmark across the globe using third-party geocoders and other data sources.
- **Re:** A Python package called Regular Expression, is a sequence of characters that forms a search pattern. It is used to check if a string contains the specific search pattern.

6. Data Cleaning

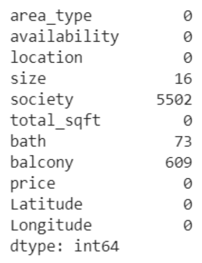
The raw data is unsuitable for analysis. To be useful for predictive modelling and perform the analysis effectively, various data cleaning techniques were carried out. We consider this process to be the backbone of our analysis and a simulating learning experience as the datasets provided are likely to be corrupted with various errors such as incorrect, inconsistent, missing or duplicated data. Moreover, databases can have constructions, arrangements and formatting that may not best suited the purpose of analysis. In addition, using machine learning methodologies with dirty data can prove troublesome to debug and can degrade the model.

The Data cleaning procedure adopted for this analysis consists of the following steps:

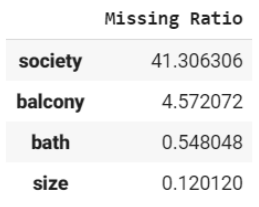
- Dealing with Missing Values.
- Replacing Values and Transforming Data.
- Modifying Data types.

6.1 Dealing with Missing Values

Real world data often comes with missing values. This can be due to human error, fault in the data collection process or due to other reasons. These missing values needs to be treated based on the feature and proportion of data missing. We check for missing values using isnull() and the ratio of missing values in the dataset is calculated as shown below,



```
area_type          0
availability       0
location           0
size              16
society         5502
total_sqft         0
bath              73
balcony          609
price              0
Latitude           0
Longitude          0
dtype: int64
```

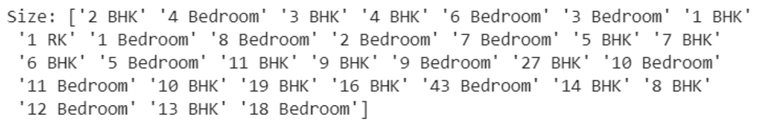


	Missing Ratio
society	41.306306
balcony	4.572072
bath	0.548048
size	0.120120

Since 'society' and balcony has a greater number of missing values we have decided to drop these features as we feel it doesn't add any value to our analysis. Also, 'society' being a categorical feature imputing more records with 'Other' category might lead the data to be biased.
Also, the missing records in 'size', 'bath' and 'balcony' are dropped using dropna().

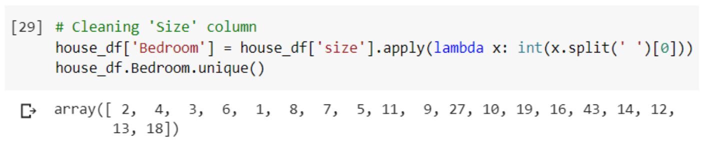
6.2 Replacing Values and Transforming Data.

- Using unique() we listed the unique values in categorical features such as 'Location', 'Size' and 'total_sqft'.
- Bedroom 'size' feature was represented in a single category in multiple ways.



```
Size: ['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']
```

- 3 Bedroom is present as '3 BHK' and '3 Bedroom'. The 'size' feature is cleaned and the result is stored in a new column called 'Bedroom',

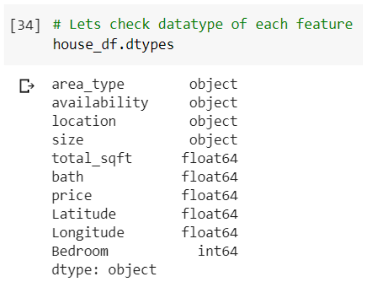


```
[29] # Cleaning 'Size' column
     house_df['Bedroom'] = house_df['size'].apply(lambda x: int(x.split(' ')[0]))
     house_df.Bedroom.unique()

     array([ 2,  4,  3,  6,  1,  8,  7,  5, 11,  9, 27, 10, 19, 16, 43, 14, 12,
            13, 18])
```

- Data in 'total_sqft' feature is not in uniform format. Consists irrelevant values such as '-', 'Sq.Meter', Perch', etc. Measurements in other formats were also present. It is cleaned by taking the average of two values. For e.g if a record is present as '1195 – 1140' we simply take the average of these records. Remaining records which has text in them are dropped.
- The features such as 'size' and 'total_sqft' are cleaned to follow a single format. Other features are already clean and does not require any alteration to make the data meaningful.

6.3 Modifying Data types.

- The ‘Bedroom’ feature is modified to Integer datatype as bedroom size must be a whole integer and not decimal value.
- The ‘total_sqft’ is modified to float data type as the total square foot can contain decimal values.
- Below is the resulting datatype of each features after modification,



```
[34] # Lets check datatype of each feature
     house_df.dtypes

     area_type        object
     availability     object
     location         object
     size             object
     total_sqft      float64
     bath            float64
     price           float64
     Latitude        float64
     Longitude       float64
     Bedroom           int64
     dtype: object
```

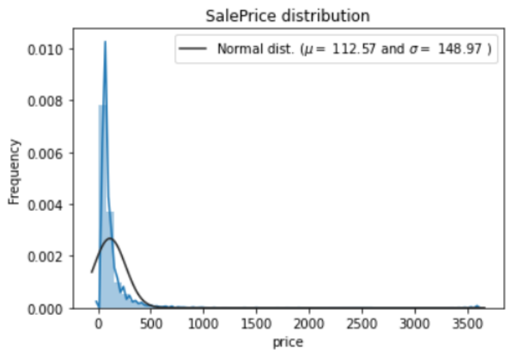
7. Data Exploration and Visualization:

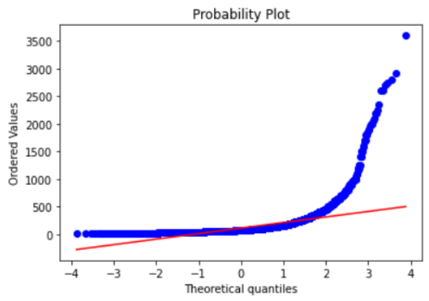
Prior to any modelling, a great amount of attention was devoted to data exploration. We believe that this step was crucial in identifying underlying trends and observe remarkable correlations, simulating questions regarding which area has the most expensive houses and which area has the least. Extensive exploration of the data was carried out as we believe that it is not a wise idea to directly feed the data into a black box and wait for the results. Exploration and visuals helped us understand the data and grasp critical information that could been easily missed – the information that supported our analysis in the long run. All the code for this section of the analysis is documented in the appendices.

7.1 Price Distribution

- Statistical distribution of ‘price’ feature tells us how the data is distributed. To obtain the price distribution we used sns.distplot() and to obtain QQ plot distribution we used stats.probplot().

```
mu = 112.57 and sigma = 148.97
```





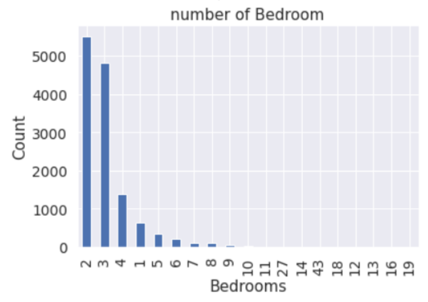
```
Skewness: 8.064469
Kurtosis: 108.166513
```

- From the plot, we observed that the distribution of price is highly right skewed, most of the house price lies below 500 lakhs and there are outliers in the dataset.
- Kurtosis measures whether the dataset is heavy-tailed or light-tailed compared to normal distribution. Our price feature has high kurtosis which means that the price distribution is a heavy tail distribution.

7.2 Bedroom

- Analyzing 'Bedroom' feature using bar plot.

```
<function seaborn.utils.despine>
```



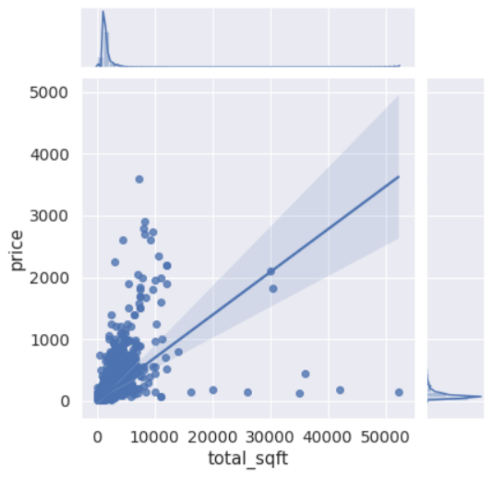
- There are a greater number of 2 and 3-bedroom houses and there are also few houses more than ten bedrooms up to 47. Clearly, we cannot have bedroom size such as 27 or 43 for a house. We can consider these as outliers but we need to first explore its relationship with total sqft of the house.

7.3 Area and Price Relationship

- As the area size of house increases the price of the house increases and a strong correlation is expected. We plotted Joint plot using sns.jointplot() to verify the relationship.

```
[ ]  # Square feet vs Sale Price
     sns.jointplot(x=house_df['total_sqft'], y=house_df['price'], kind='reg')
```

```
<seaborn.axisgrid.JointGrid at 0x7f3f8685cb38>
```

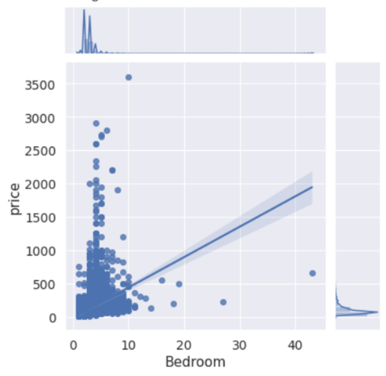


- Features are correlated but outliers are present. The data points in the bottom right are considered as outliers because more the area of house more the price unless it is located far away from the city.

7.4 Bedrooms and Price Relationship

- When bedroom size increases price increases and strong correlation is expected. We plotted a jointplot using sns.jointplot() to verify the relationship.

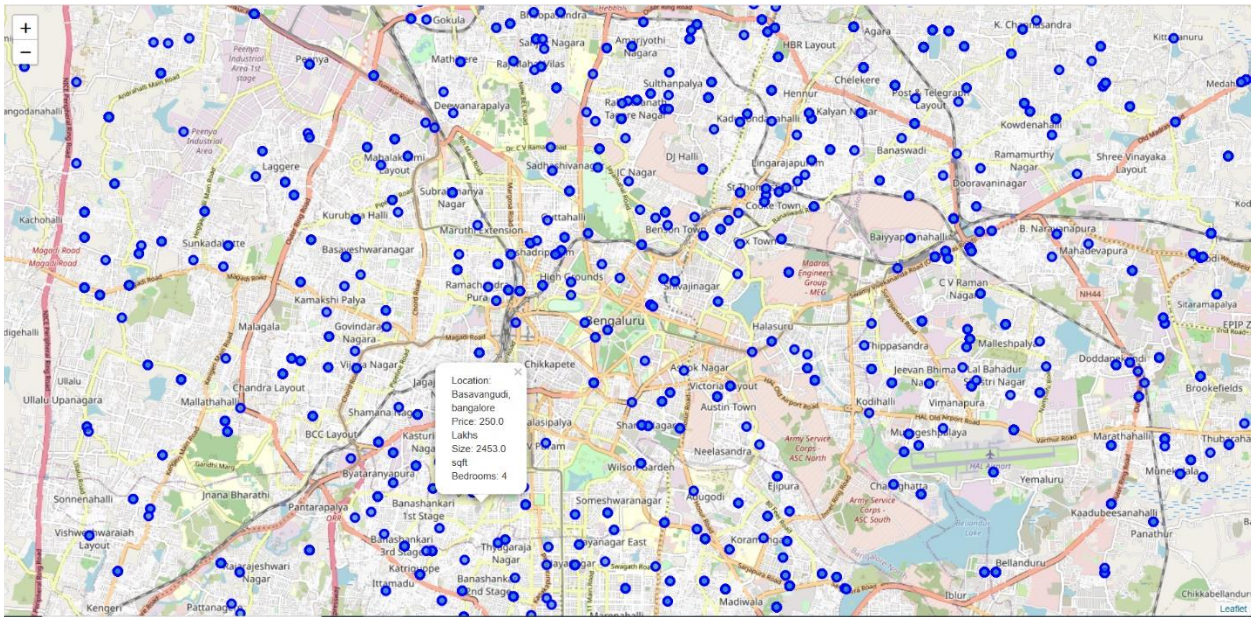
```
<seaborn.axisgrid.JointGrid at 0x7f3f84e12b00>
```



- Mild linear relationship is present but many data points between 0-10 and very few data points above 20 they are considered as outliers but only if it doesn't correlate with total area of the house.

7.5 Geospatial Exploration

- The correlated features Bedrooms, total sqft, price, latitude and longitude are plotted on a visualization map using Folium. Plotting the interest points on a map gives us a visual representation of where exactly the house is located and one can easily decide where to buy a house.

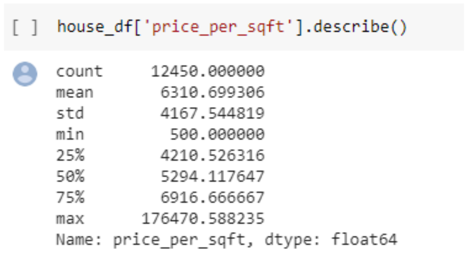


8. Feature Engineering

During data exploration we observed that features such as ‘area_size’, ‘bath’ and ‘bedroom’ are correlated but contain outliers. They are handled using business logic for each feature. Feature engineering is the most important art in machine learning which creates a huge difference between a good model and a bad model.

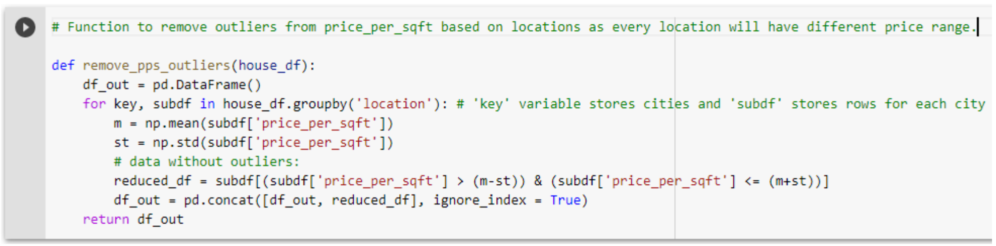
8.1 Handling Outliers - Price and total sqft per location:

The price distribution in QQ Plot is highly right skewed (heavy tail distribution) and is rectified by introducing a new feature called ‘price per sqft’. This feature helps us to retain only the standard distribution of the price data.



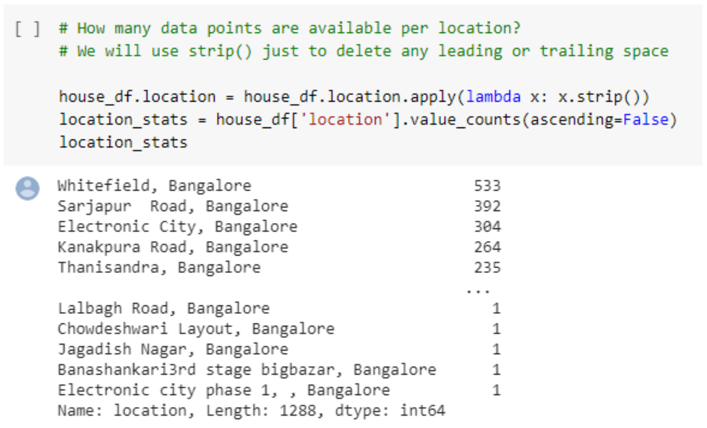
```
[ ]  house_df['price_per_sqft'].describe()

     count     12450.000000
     mean       6310.699306
     std        4167.544819
     min         500.000000
     25%        4210.526316
     50%        5294.117647
     75%        6916.666667
     max      176470.588235
     Name: price_per_sqft, dtype: float64
```

Statistical information shows min price per sqft is 500 Rs/sqft whereas maximum is 176470, this shows a wide variation in property prices. We removed outliers per location using mean and standard deviation. For normal distribution the price values near mean and std are retained. Anything above mean+std-deviation and below mean+std-deviation are outliers. Below is the code,

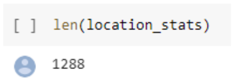

```
# Function to remove outliers from price_per_sqft based on locations as every location will have different price range.

def remove_pps_outliers(house_df):
    df_out = pd.DataFrame()
    for key, subdf in house_df.groupby('location'): # 'key' variable stores cities and 'subdf' stores rows for each city
        m = np.mean(subdf['price_per_sqft'])
        st = np.std(subdf['price_per_sqft'])
        # data without outliers:
        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
```

8.2 Reducing Location dimensions

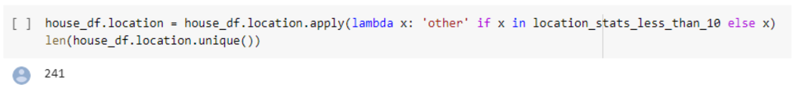
Total number of unique locations are very high. Location being categorical value, without preprocessing we end up in high dimension problem and it must be reduced. This is done by tagging the locations that have less than 10 data points as 'Other'.


```
# How many data points are available per location?
# We will use strip() just to delete any leading or trailing space

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

```
Whitefield, Bangalore                         533
Sarjapur  Road, Bangalore                     392
Electronic City, Bangalore                    304
Kanakpura Road, Bangalore                     264
Thanisandra, Bangalore                        235
                                             ...
Lalbagh Road, Bangalore                         1
Chowdeshwari Layout, Bangalore                  1
Jagadish Nagar, Bangalore                       1
Banashankari3rd stage bigbazar, Bangalore       1
Electronic city phase 1, , Bangalore            1
Name: location, Length: 1288, dtype: int64
```


```
len(location_stats)
```

```
1288
```

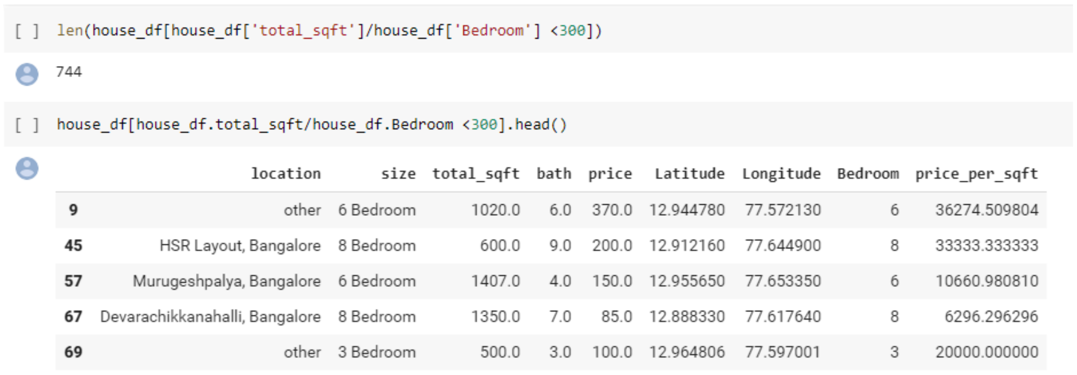

```
house_df.location = house_df.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)
len(house_df.location.unique())
```

```
241
```

Using this method, we reduced the number of dimensions from 1288 to 241. Later we use one hot encoding to convert the location feature to categorical values.

8.3 Handling Outliers – Bedroom

Assuming size of 1-Bedroom as 300 sqft, the total square foot area of the house must be proportional to this value. Any values that doesn't fall in this range are considered as outliers. Checking the disproportionate locations,



```
[ ] len(house_df[house_df['total_sqft']/house_df['Bedroom'] <300])
```

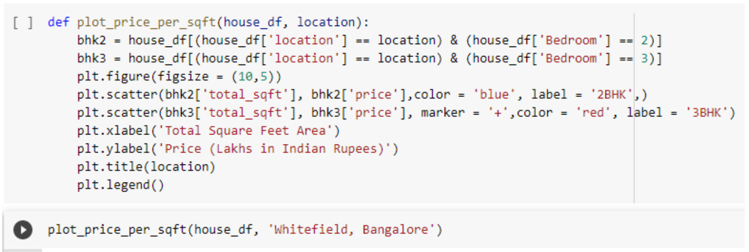
```
744
```

```
[ ] house_df[house_df.total_sqft/house_df.Bedroom <300].head()
```

	location	size	total_sqft	bath	price	Latitude	Longitude	Bedroom	price_per_sqft
9	other	6 Bedroom	1020.0	6.0	370.0	12.944780	77.572130	6	36274.509804
45	HSR Layout, Bangalore	8 Bedroom	600.0	9.0	200.0	12.912160	77.644900	8	33333.333333
57	Murugeshpalya, Bangalore	6 Bedroom	1407.0	4.0	150.0	12.955650	77.653350	6	10660.980810
67	Devarachikkanahalli, Bangalore	8 Bedroom	1350.0	7.0	85.0	12.888330	77.617640	8	6296.296296
69	other	3 Bedroom	500.0	3.0	100.0	12.964806	77.597001	3	20000.000000

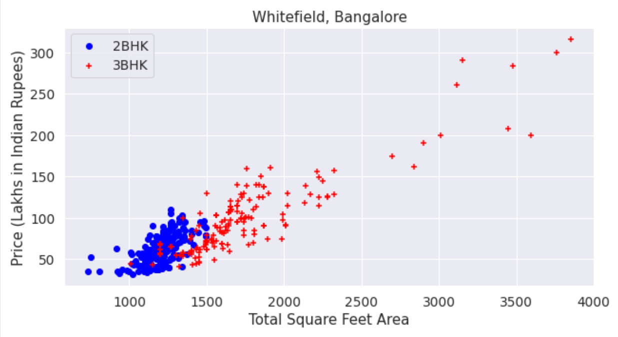
One house has 8 bedrooms but total area is 800 sqft which is not acceptable. So, these 744 records are considered as outliers and are removed.

Another scenario is when the sqft area of 2-Bedroom house is more than sqft area of 3-Bedroom then the price of 2-Bedroom house being more than 3 is justifiable. But, when 2-bedroom house is more than 3-bedroom house even with same sqft area or less, this could be because of many reasons like 2-Bedroom house being in a prime location. We investigated this condition by a scatter plot of location vs Bedrooms. For a given location the 2 and 3-Bedroom properties looks like,



```
[ ] def plot_price_per_sqft(house_df, location):
        bhk2 = house_df[(house_df['location'] == location) & (house_df['Bedroom'] == 2)]
        bhk3 = house_df[(house_df['location'] == location) & (house_df['Bedroom'] == 3)]
        plt.figure(figsize = (10,5))
        plt.scatter(bhk2['total_sqft'], bhk2['price'],color = 'blue', label = '2BHK',)
        plt.scatter(bhk3['total_sqft'], bhk3['price'], marker = '+',color = 'red', label = '3BHK')
        plt.xlabel('Total Square Feet Area')
        plt.ylabel('Price (Lakhs in Indian Rupees)')
        plt.title(location)
        plt.legend()
```

```
plot_price_per_sqft(house_df, 'Whitefield, Bangalore')
```



From the plot the price of 2-bedroom houses are more than 3-bedroom for same location (Whitefield, Bangalore) and for same square feet (at 1000 and 1500). These outliers are removed by checking the mean price per sqft area. If the mean price per sqft area of 2-bedrooms is less than mean price per sqft of 1-bedroom the records are removed.

```
[ ] # user defined function to remove the outliers using mean and std deviation
    def remove_bhk_outliers(house_df):
        exclude_indices = np.array([])
        for location, location_df in house_df.groupby('location'):
            bhk_stats = {}
            for bhk, bhk_df in location_df.groupby('Bedroom'):
                bhk_stats[bhk] = {
                    'mean': np.mean(bhk_df.price_per_sqft),
                    'std': np.std(bhk_df.price_per_sqft),
                    'count': bhk_df.shape[0] #shape would have given RowsXColumns, we want only number of rows, so shape[0]
                }
            for bhk, bhk_df in location_df.groupby('Bedroom'):
                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 house_df.drop(exclude_indices,axis='index')
```

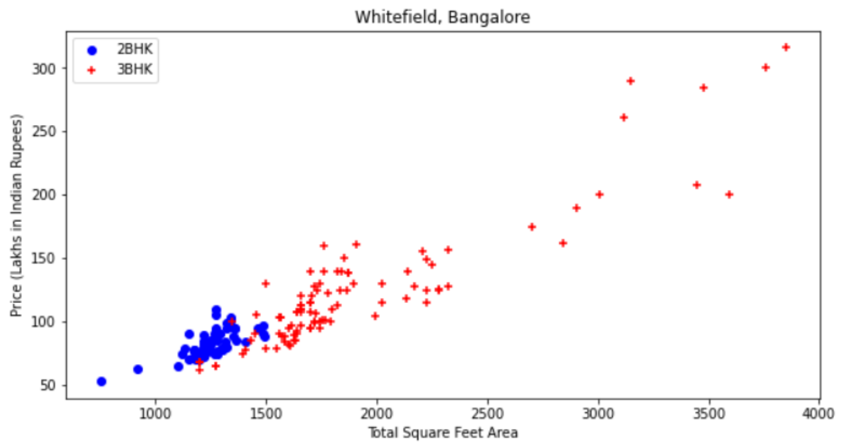
```
[ ] # Shape of the dataset before removing outliers
    house_df.shape
```

```
(10240, 9)
```

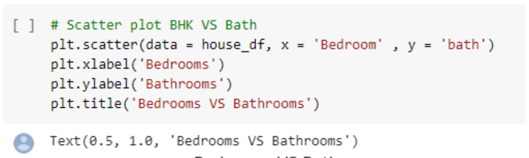
```
[ ] # Shapre of dataset after removing outliers
    house_df = remove_bhk_outliers(house_df)
    house_df.shape
```

```
(7317, 9)
```

```
[ ] # Bedroom size vs Price plot per location after removing outliers
    plot_price_per_sqft(house_df, 'Whitefield, Bangalore')
```

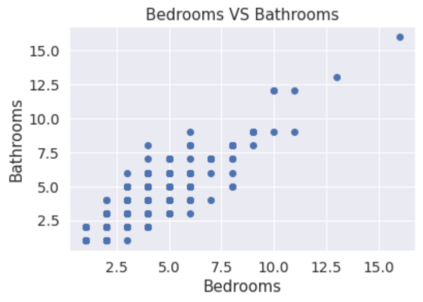


8.4 Handling Outliers - Bathroom

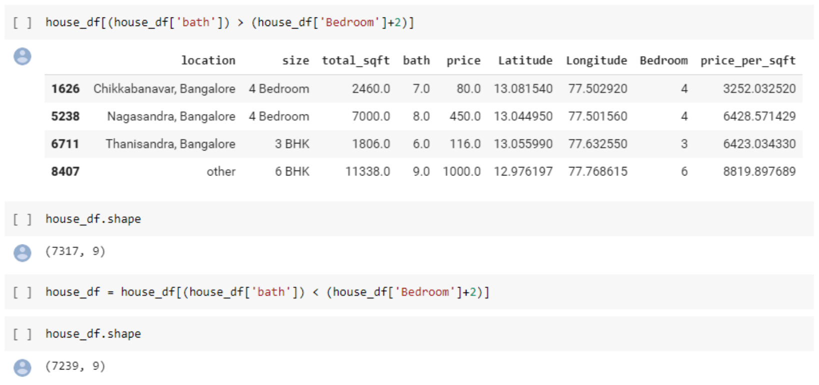


```
[ ]  # Scatter plot BHK VS Bath
     plt.scatter(data = house_df, x = 'Bedroom' , y = 'bath')
     plt.xlabel('Bedrooms')
     plt.ylabel('Bathrooms')
     plt.title('Bedrooms VS Bathrooms')
```

```
Text(0.5, 1.0, 'Bedrooms VS Bathrooms')
```



Few records having more Bathrooms than Bedrooms. For 4-bedroom house we have around 8 bathrooms which doesn't look right. It is unusual to have 2 more bathrooms than number of bedrooms in a house. These values are considered as outliers and are removed.



```
[ ]  house_df[(house_df['bath']) > (house_df['Bedroom']+2)]
```

	location	size	total_sqft	bath	price	Latitude	Longitude	Bedroom	price_per_sqft
1626	Chikkabanavar, Bangalore	4 Bedroom	2460.0	7.0	80.0	13.081540	77.502920	4	3252.032520
5238	Nagasandra, Bangalore	4 Bedroom	7000.0	8.0	450.0	13.044950	77.501560	4	6428.571429
6711	Thanisandra, Bangalore	3 BHK	1806.0	6.0	116.0	13.055990	77.632550	3	6423.034330
8407	other	6 BHK	11338.0	9.0	1000.0	12.976197	77.768615	6	8819.897689

```
[ ]  house_df.shape
```

```
(7317, 9)
```

```
[ ]  house_df = house_df[(house_df['bath']) < (house_df['Bedroom']+2)]
```

```
[ ]  house_df.shape
```

```
(7239, 9)
```

9. Modelling

9.1 Feature Selection:

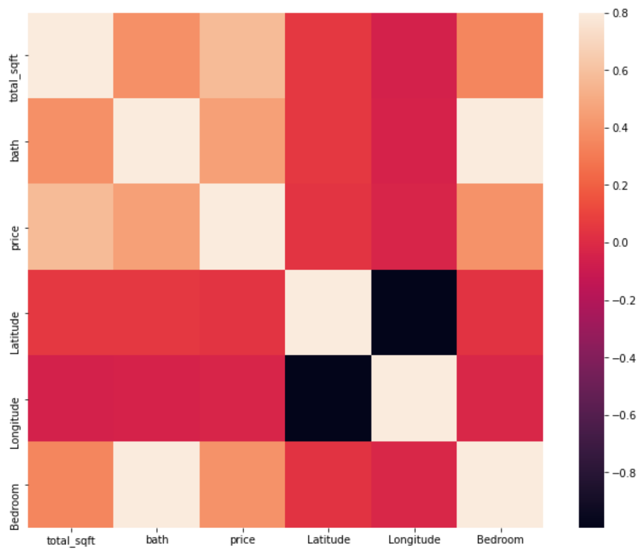
To create an accurate predictive model, we chose features that will give us better accuracy whilst using less data.

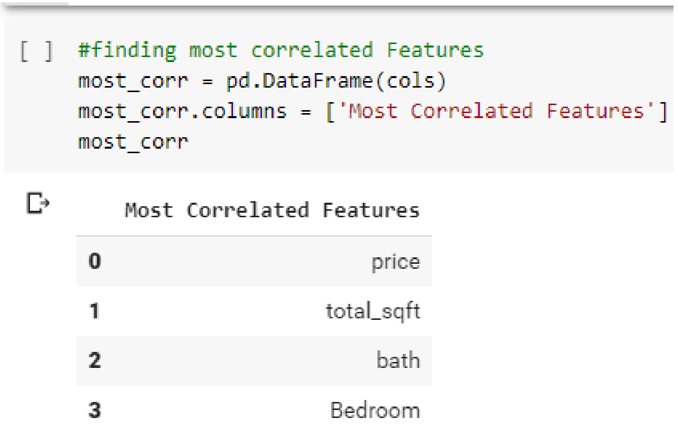
Benefits of feature selection:

- Fast Training
- Reduces complexity of our models
- Improves accuracy
- Reduces Overfitting

Using correlation matrix with heatmap we determine the highly correlated features and verify the correlation with price using regression scatter plot. Below is the heatmap and highly correlated features.

```
[ ] # Correlation Matrix Heatmap
    corrmat = house_df.corr()
    f, ax = plt.subplots(figsize=(12, 9))
    sns.heatmap(corrmat, vmax=.8, square=True);
```

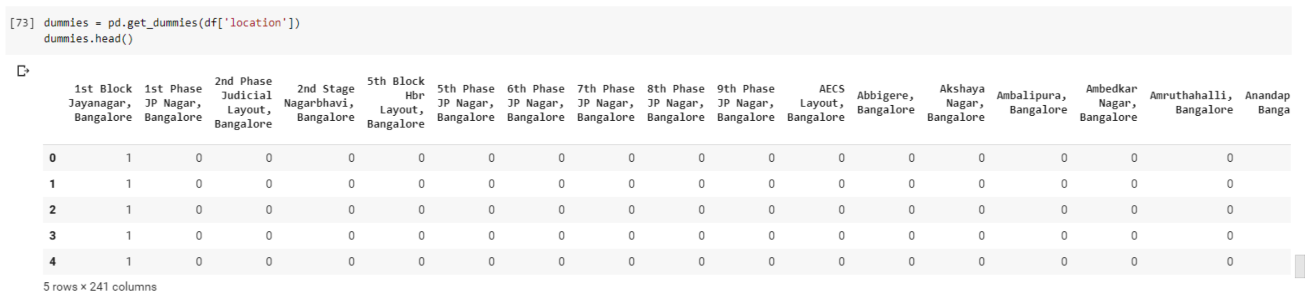




```
[ ] #finding most correlated Features
    most_corr = pd.DataFrame(cols)
    most_corr.columns = ['Most Correlated Features']
    most_corr
```

	Most Correlated Features
0	price
1	total_sqft
2	bath
3	Bedroom

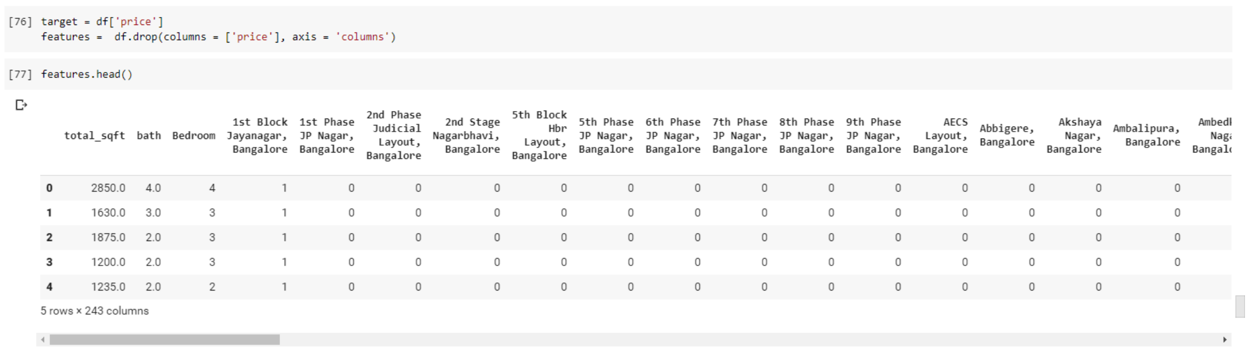
We chose ‘Bedroom’, ‘bath’, ‘total_sqft’ and location as input variables and ‘price’ as output variables for regression model. We performed one-hot-encoding on ‘location’ feature.



```
[73] dummies = pd.get_dummies(df['location'])
     dummies.head()
```

	1st Block Jayanagar, Bangalore	1st Phase JP Nagar, Bangalore	2nd Phase Judicial Layout, Bangalore	2nd Stage Nagarbhavi, Bangalore	5th Block Hbr Layout, Bangalore	5th Phase JP Nagar, Bangalore	6th Phase JP Nagar, Bangalore	7th Phase JP Nagar, Bangalore	8th Phase JP Nagar, Bangalore	9th Phase JP Nagar, Bangalore	AECS Layout, Bangalore	Abbigere, Bangalore	Akshaya Nagar, Bangalore	Ambalipura, Bangalore	Ambedkar Nagar, Bangalore	Amruthahalli, Bangalore	Anandap Banga
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

5 rows × 241 columns

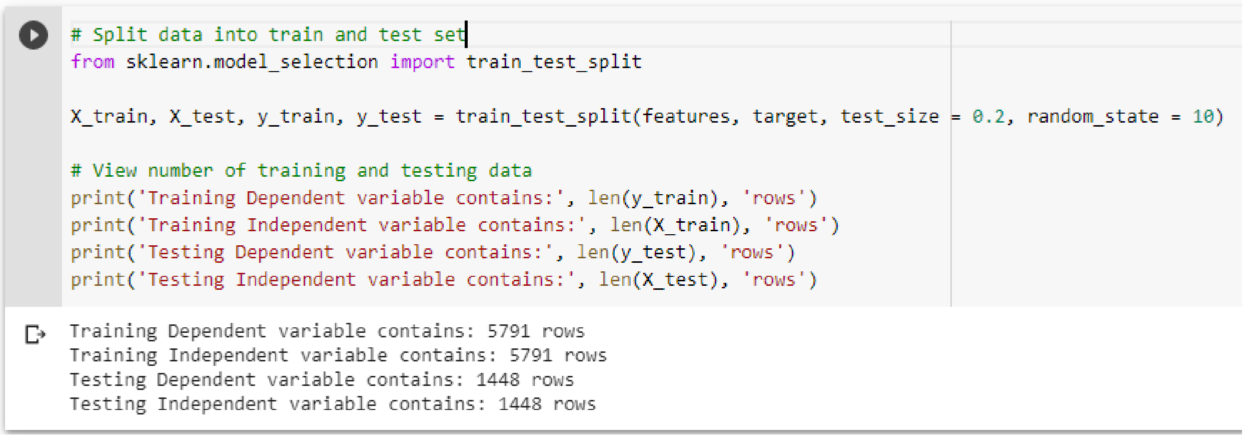


```
[76] target = df['price']
     features =  df.drop(columns = ['price'], axis = 'columns')

[77] features.head()
```

	total_sqft	bath	Bedroom	1st Block Jayanagar, Bangalore	1st Phase JP Nagar, Bangalore	2nd Phase Judicial Layout, Bangalore	2nd Stage Nagarbhavi, Bangalore	5th Block Hbr Layout, Bangalore	5th Phase JP Nagar, Bangalore	6th Phase JP Nagar, Bangalore	7th Phase JP Nagar, Bangalore	8th Phase JP Nagar, Bangalore	9th Phase JP Nagar, Bangalore	AECS Layout, Bangalore	Abbigere, Bangalore	Akshaya Nagar, Bangalore	Ambalipura, Bangalore	Ambedl Naga Bangal
0	2850.0	4.0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1630.0	3.0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	1875.0	2.0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	1200.0	2.0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	1235.0	2.0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	

5 rows × 243 columns

For modelling we chose 80 percent of data for training and 20 percent for testing.



```
# Split data into train and test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size = 0.2, random_state = 10)

# View number of training and testing data
print('Training Dependent variable contains:', len(y_train), 'rows')
print('Training Independent variable contains:', len(X_train), 'rows')
print('Testing Dependent variable contains:', len(y_test), 'rows')
print('Testing Independent variable contains:', len(X_test), 'rows')
```

```
Training Dependent variable contains: 5791 rows
Training Independent variable contains: 5791 rows
Testing Dependent variable contains: 1448 rows
Testing Independent variable contains: 1448 rows
```

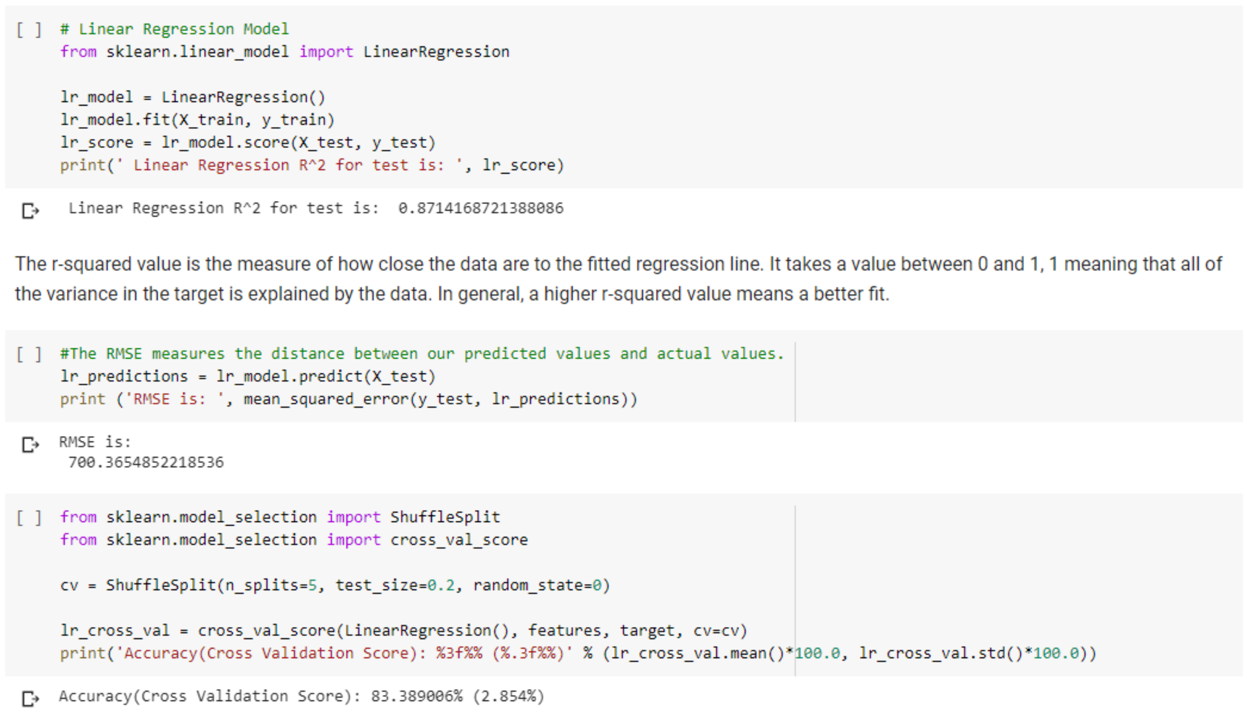
Following prediction model building steps are applicable to all the methods: -

- Once the model is fitted on the training data, we calculate the R^2 score and RMSE is obtained.
- Cross-validation is done to acquire the cross-validation score.
- Finally, for the benchmark model we obtain the residual plot.

9.2 Our Models

9.2.1 Linear Regression

In the initial stage we use linear regression which is basic predictive analysis.



```
# Linear Regression Model
from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_score = lr_model.score(X_test, y_test)
print(' Linear Regression R^2 for test is: ', lr_score)
```

```
 Linear Regression R^2 for test is:  0.8714168721388086
```

The r-squared value is the measure of how close the data are to the fitted regression line. It takes a value between 0 and 1, 1 meaning that all of the variance in the target is explained by the data. In general, a higher r-squared value means a better fit.

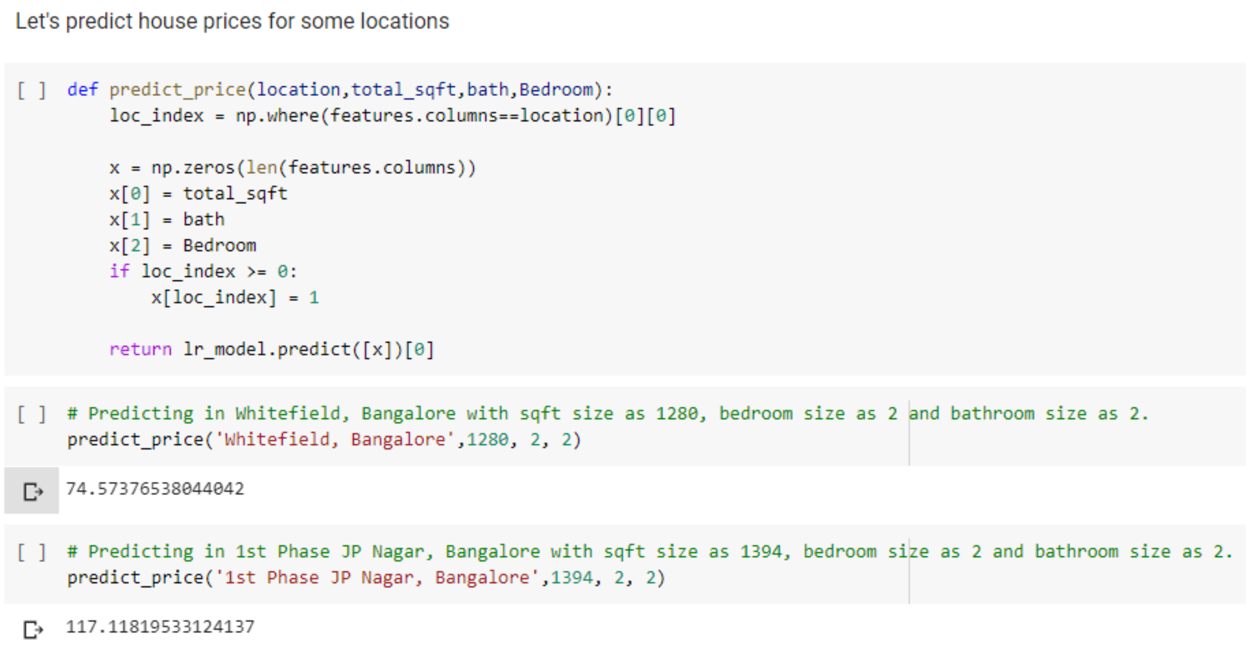
```
#The RMSE measures the distance between our predicted values and actual values.
lr_predictions = lr_model.predict(X_test)
print ('RMSE is: ', mean_squared_error(y_test, lr_predictions))
```

```
RMSE is: 
 700.3654852218536
```

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

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

lr_cross_val = cross_val_score(LinearRegression(), features, target, cv=cv)
print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (lr_cross_val.mean()*100.0, lr_cross_val.std()*100.0))
```

```
Accuracy(Cross Validation Score): 83.389006% (2.854%)
```

The cross-validation results give us 83.38 % accuracy. We also predicted the house prices of some locations,



Let's predict house prices for some locations

```
def predict_price(location,total_sqft,bath,Bedroom):
    loc_index = np.where(features.columns==location)[0][0]

    x = np.zeros(len(features.columns))
    x[0] = total_sqft
    x[1] = bath
    x[2] = Bedroom
    if loc_index >= 0:
        x[loc_index] = 1

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

```
# Predicting in Whitefield, Bangalore with sqft size as 1280, bedroom size as 2 and bathroom size as 2.
predict_price('Whitefield, Bangalore',1280, 2, 2)
```

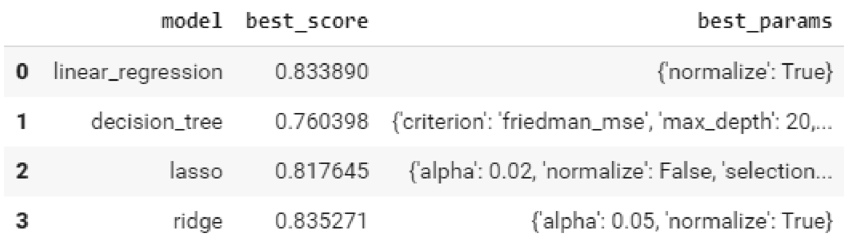
```
74.57376538044042
```

```
# Predicting in 1st Phase JP Nagar, Bangalore with sqft size as 1394, bedroom size as 2 and bathroom size as 2.
predict_price('1st Phase JP Nagar, Bangalore',1394, 2, 2)
```

```
117.11819533124137
```

Grid Search CV:

Several others models were searched using Grid Search CV method and below are the results,



	model	best_score	best_params
0	linear_regression	0.833890	{'normalize': True}
1	decision_tree	0.760398	{'criterion': 'friedman_mse', 'max_depth': 20,...
2	lasso	0.817645	{'alpha': 0.02, 'normalize': False, 'selection...
3	ridge	0.835271	{'alpha': 0.05, 'normalize': True}

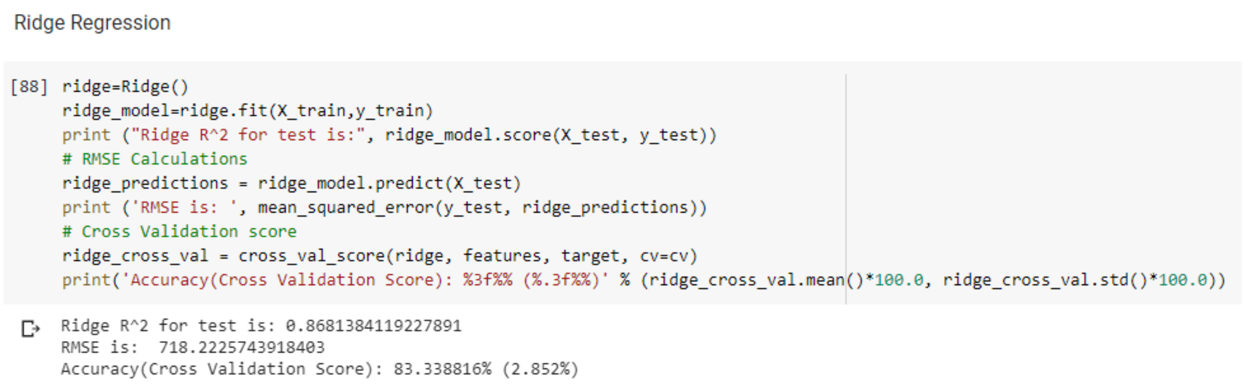
From the result Ridge regression gives better accuracy.

Advanced Regression Techniques

While the linear regression model is too simple to accurately build a prediction model, there are many fundamental concepts in linear regression that many other regression techniques build upon. Further to improve the accuracy, advance regression techniques are used to form the prediction model.

9.2.2 Ridge Regression

Ridge Regression is a technique used when the data suffers from multicollinearity (independent variables are highly correlated). The bedroom size and square foot area are correlated. In such scenario if the least square estimates are unbiased, their variances are large that results in greater difference between observed values and true values. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.



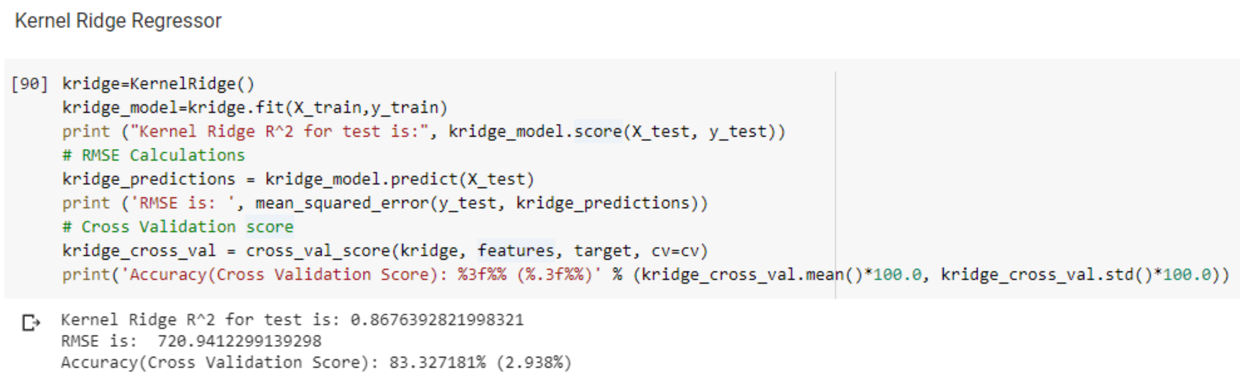
Ridge Regression

```
[88] ridge=Ridge()
     ridge_model=ridge.fit(X_train,y_train)
     print ("Ridge R^2 for test is:", ridge_model.score(X_test, y_test))
     # RMSE Calculations
     ridge_predictions = ridge_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, ridge_predictions))
     # Cross Validation score
     ridge_cross_val = cross_val_score(ridge, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (ridge_cross_val.mean()*100.0, ridge_cross_val.std()*100.0))
```

```
Ridge R^2 for test is: 0.8681384119227891
RMSE is:  718.2225743918403
Accuracy(Cross Validation Score): 83.338816% (2.852%)
```

9.2.3 Kernel Ridge

Kernel ridge regression combines Ridge Regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. The form of the model learned by Kernel Ridge is identical to support vector regression (SVR).

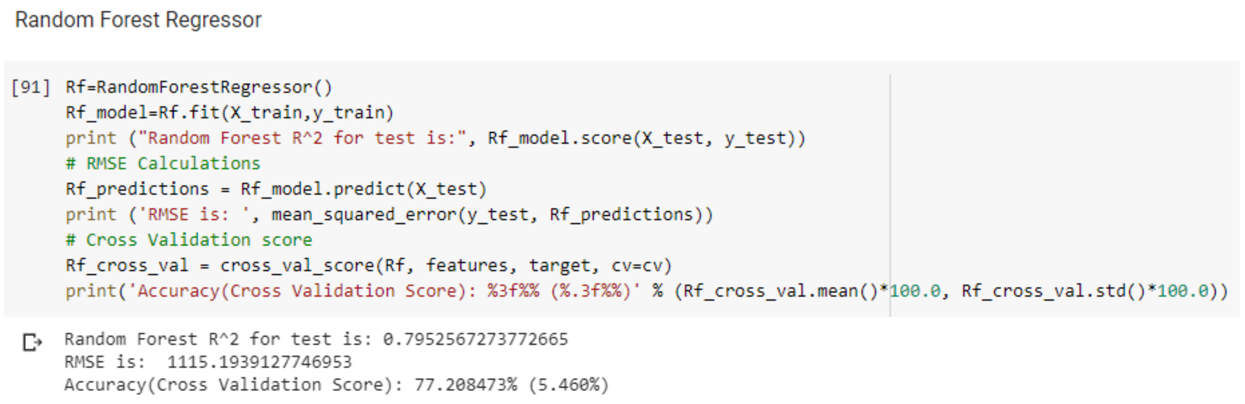

Kernel Ridge Regressor

```
[90] kridge=KernelRidge()
     kridge_model=kridge.fit(X_train,y_train)
     print ("Kernel Ridge R^2 for test is:", kridge_model.score(X_test, y_test))
     # RMSE Calculations
     kridge_predictions = kridge_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, kridge_predictions))
     # Cross Validation score
     kridge_cross_val = cross_val_score(kridge, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (kridge_cross_val.mean()*100.0, kridge_cross_val.std()*100.0))
```

```
Kernel Ridge R^2 for test is: 0.8676392821998321
RMSE is:  720.9412299139298
Accuracy(Cross Validation Score): 83.327181% (2.938%)
```

9.2.4 Random Forest

Random Forest is a trademark term for an ensemble of decision trees. To classify a new object based on attributes, each tree gives a classification as if the tree votes for that class. The forest chooses the classification having the most vote.

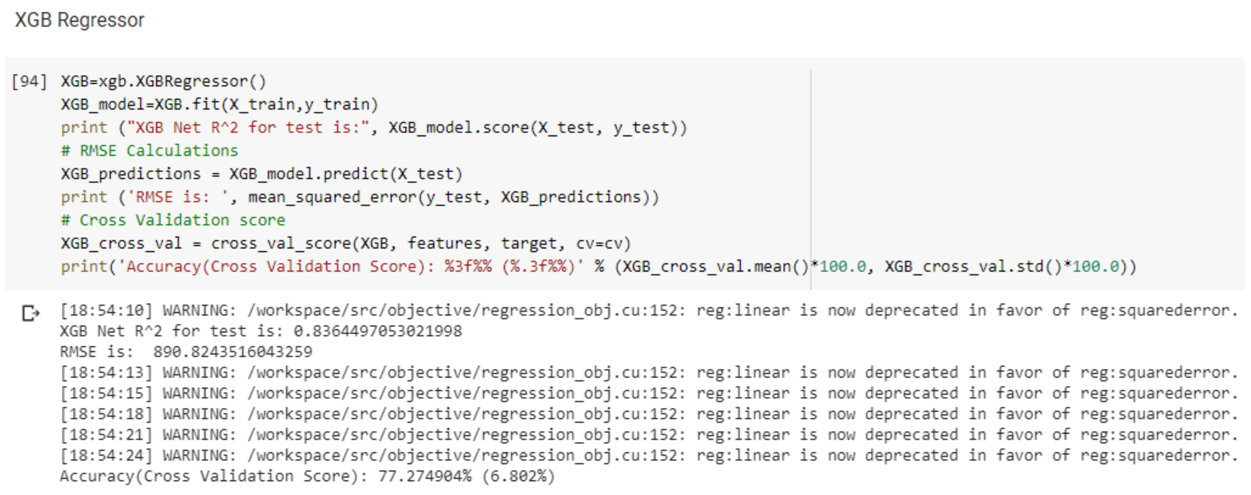

Random Forest Regressor

```
[91] Rf=RandomForestRegressor()
     Rf_model=Rf.fit(X_train,y_train)
     print ("Random Forest R^2 for test is:", Rf_model.score(X_test, y_test))
     # RMSE Calculations
     Rf_predictions = Rf_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, Rf_predictions))
     # Cross Validation score
     Rf_cross_val = cross_val_score(Rf, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (Rf_cross_val.mean()*100.0, Rf_cross_val.std()*100.0))
```

```
Random Forest R^2 for test is: 0.7952567273772665
RMSE is:  1115.1939127746953
Accuracy(Cross Validation Score): 77.208473% (5.460%)
```

9.2.5 XGB Regressor

XGBoost is an advanced implementation of gradient boosting algorithm. Gradient boosting is a machine learning technique for regression and classification problems that produces a prediction model in the form of an ensemble of weak prediction models.
It includes variety of regularization which reduces overfitting and improves overall performance.

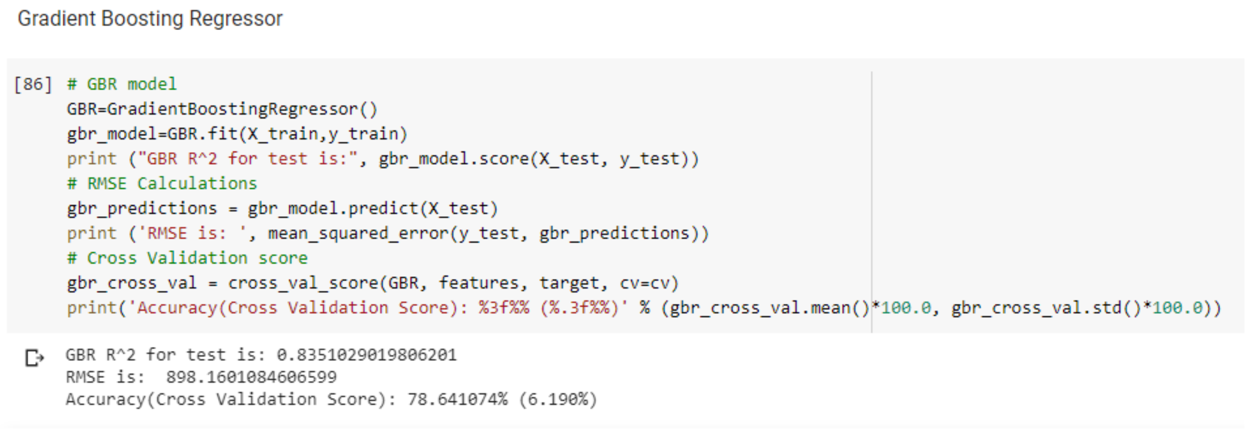

XGB Regressor

```
[94] XGB=xgb.XGBRegressor()
     XGB_model=XGB.fit(X_train,y_train)
     print ("XGB Net R^2 for test is:", XGB_model.score(X_test, y_test))
     # RMSE Calculations
     XGB_predictions = XGB_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, XGB_predictions))
     # Cross Validation score
     XGB_cross_val = cross_val_score(XGB, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (XGB_cross_val.mean()*100.0, XGB_cross_val.std()*100.0))
```

```
[18:54:10] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGB Net R^2 for test is: 0.8364497053021998
RMSE is:  890.8243516043259
[18:54:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[18:54:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[18:54:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[18:54:21] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[18:54:24] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Accuracy(Cross Validation Score): 77.274904% (6.802%)
```

9.2.6 Gradient Boosting Regressor

Gradient Boosting Regressor converts weak learners into strong learners. It produces a prediction model in the form of an ensemble of weak prediction model typically decision trees.


Gradient Boosting Regressor

```
[86] # GBR model
     GBR=GradientBoostingRegressor()
     gbr_model=GBR.fit(X_train,y_train)
     print ("GBR R^2 for test is:", gbr_model.score(X_test, y_test))
     # RMSE Calculations
     gbr_predictions = gbr_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, gbr_predictions))
     # Cross Validation score
     gbr_cross_val = cross_val_score(GBR, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (gbr_cross_val.mean()*100.0, gbr_cross_val.std()*100.0))
```

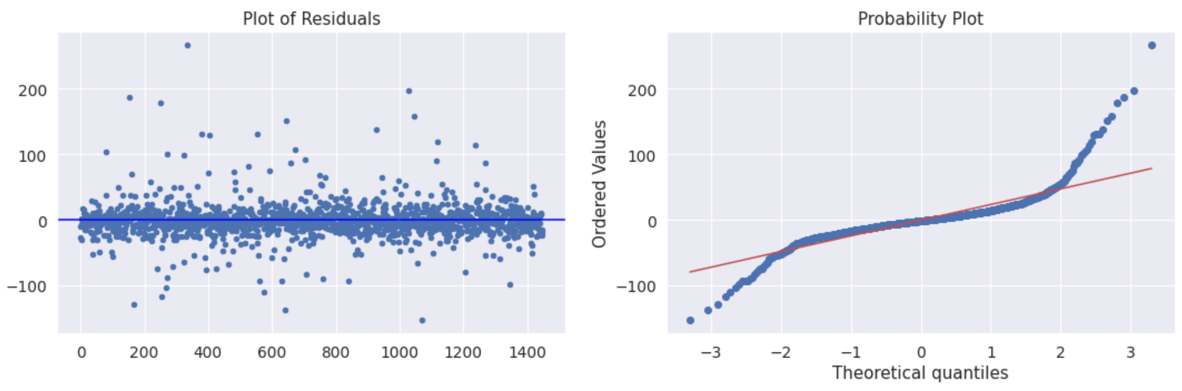
```
GBR R^2 for test is: 0.8351029019806201
RMSE is:  898.1601084606599
Accuracy(Cross Validation Score): 78.641074% (6.190%)
```

Residual Plotting

Residual is the difference between actual value and predicted value. With the final model obtained the residuals are plotted to show their effective behavior.
The residuals roughly formed a “horizontal band” around the 0 line, this suggest that the variances of the error terms are equal.

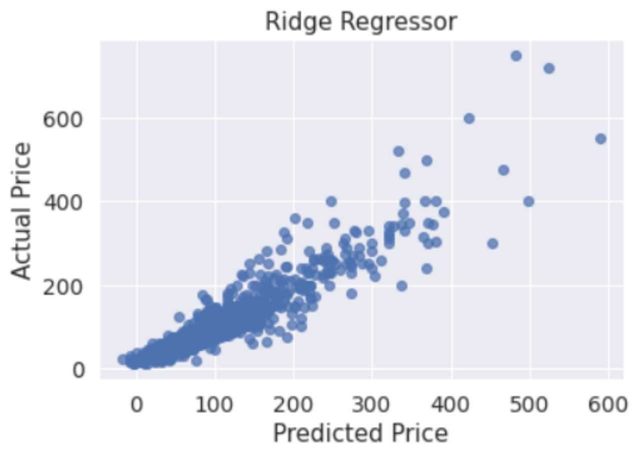
Residual Plot

```
[101] # calculate the residuals
      ridge_preds = pd.DataFrame(ridge_predictions)
      y_test = y_test.reset_index(drop=True)
      residuals = y_test - ridge_preds[0]

      # Plotting residual and probability graph
      plt.figure(figsize=(18,5))
      plt.subplot(1,2,1)
      plt.axhline(0, color='blue')
      plt.title('Plot of Residuals')
      plt.scatter(residuals.index, residuals, s=20)

      plt.subplot(1,2,2)
      plt.title('Probability Plot')
      stats.probplot(residuals, dist='norm', plot=plt)
      plt.show()
```



Actual Price vs Predicted Price

```
[ ] # Plot of variation between predicted price and actual price
    actual_values = y_test
    plt.scatter(ridge_predictions, actual_values, alpha=.75,
                color='b') #alpha helps to show overlapping data
    plt.xlabel('Predicted Price')
    plt.ylabel('Actual Price')
    plt.title('Ridge Regressor')
    plt.show()
```



The graph of predicted price versus actual price show good relation between actual house prices and predicted house prices.

10. Evaluation Metrics

This project uses two different evaluation metrics to test the hypotheses: R square score and RMSE.

- R square is the goodness of fit of the predictions to actual values. It is the explained variation divided by the total variation.

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}.$$

- MSE (Mean Square Error is the average of sum of the squares of the difference between the predicted value and true value.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

- RMSE (Root Mean Square Error) is the square root of the average of all the error. It is simply the square root of the mean square error (Metrics).

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

11.Evaluation Results

Regression methods	RMSE	R^2 Score	Accuracy
Linear Regression	700.365	0.871	83.39%
Ridge Regression	718.222	0.868	83.33%
Kernel Ridge	720.941	0.867	83.32%
Random Forest Regressor	1109.696	0.796	76.99%
XGBoosting Regressor	890.824	0.836	77.27%
Gradient Boosting Regressor	898.16	0.835	78.64%

12.Concluding Remarks

The goal of the project was to build a house price prediction model for a given area in the city of Bangalore. Upon implementing multiple machine learning models, we decided that Ridge Regression gives the best model performance. We also believe with proper hyperparameter tuning the performance of XGBoosting, Gradient Boosting regressor methods can be improved. Since only few features were chosen simple regression algorithms gave better accuracy but when the features increase, we believe the advanced regression methods yield better performance.

According to our analysis square feet area, bedroom size and bathroom size for a given location have the greatest statistical significance in predicting a house's sale price.

13.Future Work

- The effective dataset used in this coursework is around 8000 rows after preprocessing. We still felt more records were required.
- Various parameter optimization techniques can be carried out to improve the model performance.
- Dimensionality reduction methods such as PCA can be done to reduce the dimensions and improve the prediction accuracy.
- The dataset collected has moderate number of features. Ideally more features such as neighborhood crime ratings, access to public transport, condition of the house, proximity to school., etc. can be considered.
- After gathering more features a neural network regression model can be implemented.
- There are quite different metrics that can be derived with the current model. This can indicate a better result.

14. Bayesian network structure learning analysis

The information generated in NetBeans after the execution of step of 4:

_____________________________ Evaluation ______________________________

Nodes: 8

Sample size: 768

TrueDAG arcs: 9

TrueDAG independencies: 27

LearnedDAG arcs: 8

LearnedDAG independencies: 28

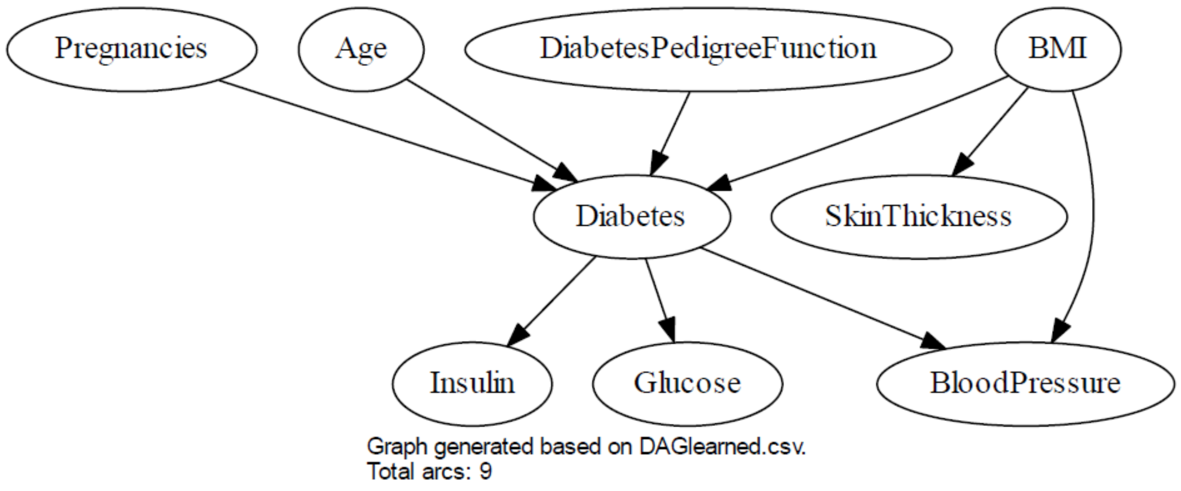
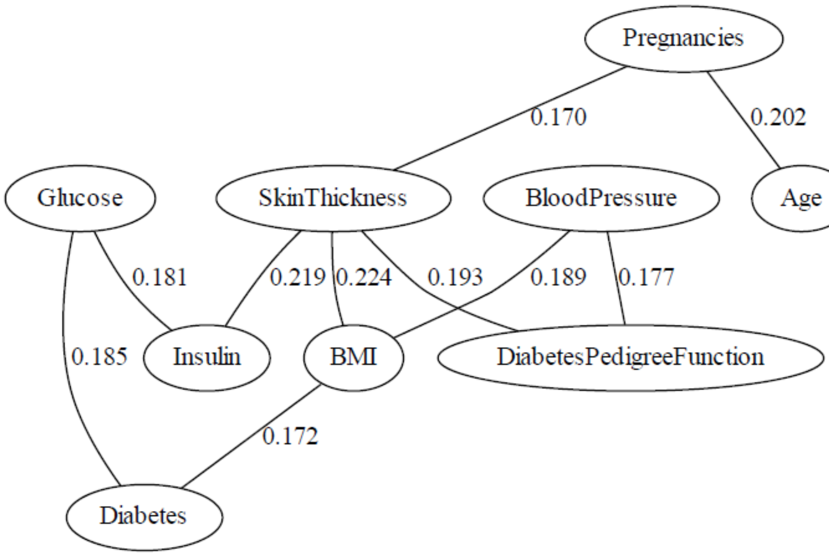
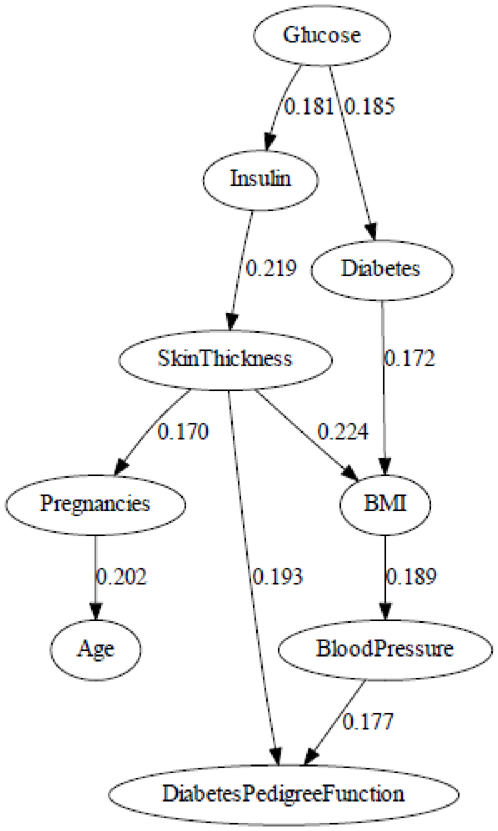


Fig : The Knowledge based graph

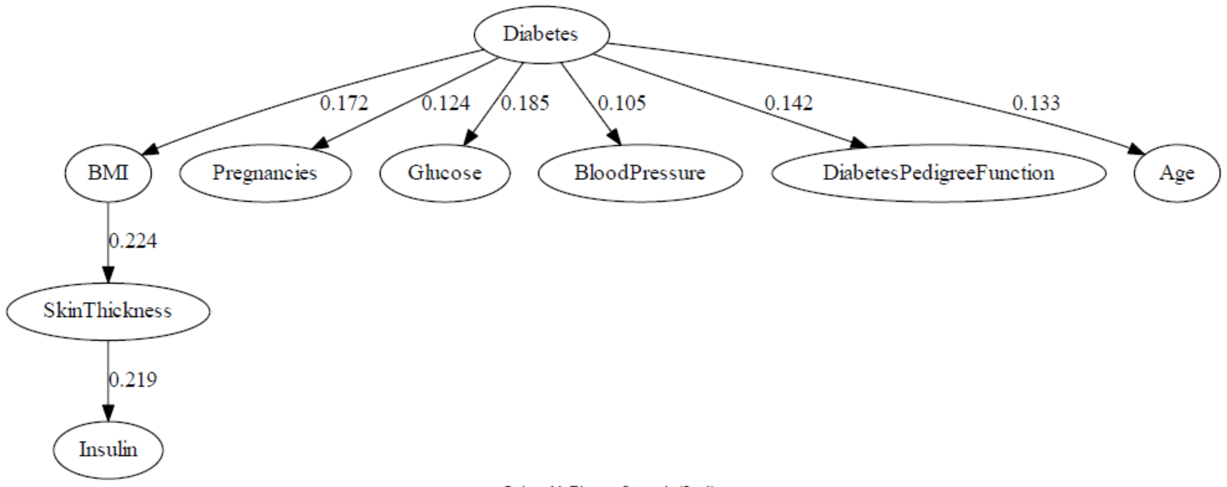
Answer 1 : First we did some discussion, and found we did not have enough knowledge, so we searched on the internet about the causal relationships, during this we found a research paper(Mohammadi et al. 2015) in which researchers tried to model this very topic using a Bayesian Network. One variable "Overweight" was extra in that paper for which we did not have the data. Since BMI index can be used to determine whether a person is overweight or not, we all agreed to replace the "Overweight" variable with "BMI".



SaiyanH_Phase_1 EMSG graph.
Total edges: 10



SaiyanH_Phase_2 EMSG directed graph.
Total arcs: 10



SaiyanH_Phase_3 graph (final).
Total arcs: 8

Answer 2: Phase1 graph has only undirected-edges whereas Phase2 graph has only directed-edges. Phase2 graph is generated by performing conditional independence tests on all sets of triples and they are classified into conditional dependence, independence or insignificance and the edges are oriented accordingly. Then BIC scoring function is used, edge is oriented if the BIC score increases otherwise it is undirected. The undirected-edges are re-assessed using do-calculus which orients the edges such that it maximizes the number of nodes affected by a causal link. Undirected-edges at this point are re-assessed using BIC scoring. Cyclic graphs are avoided by reversing the process.

Answer 3: Phase2 and Phase3 graphs are both directed but Phase3 has highest BIC. The output of Phase2 is used as starting graph for Phase3, then Hill-Climbing explores neighbouring graphs in which an edge is either added,reversed or removed(avoiding cyclic or multiple-independent-component graphs). If neighbour graph has BIC greater than current graph, make neighbour as current graph. Repeat until no neighbour increases BIC. local maximum avoided using TABU. If phase2 and phase3 graphs are identical it means we already got the graph in phase2 which has the highest BIC in comparison to all its possible neighbours and neighbour's neighbours.

Answer 4:

marginalDep.csv : 36

conditionalInsignificance.csv : 251

conditionalIndep.csv : 1

conditionalDep.csv : 0

The "marginalDep" is generated during Phase1 , which contains MMD score for all possible 2 node combinations out of 9 nodes present in network. Therefore it has C(9,2)=9!/(2!(9-2)!)=9x4=36 rows. In Phase2 the other 3 files are generated. In Phase2 , 2 nodes are selected and conditioned on all the other nodes one by one, since we have 9 nodes, total number of possibilities is C(9,2)x7=9x4x7=252 . These 252 combinations are then evaluated using dependency rules and then segregated into three files.

Answer 5: The F1, SHD and BSF scores of our dataset are mostly higher, lower and higher respectively than those shown in figure 2 which implies that the graph generated of our dataset is more accurate. In general, accuracy tends to improve with the increase in sample size. We were expecting the same results because there is less number of nodes that we used in our dataset which did not generate any independent graphs. Since the complexity increases with the increase in number of nodes which in turn increases the number of independent subgraphs and it leads to the decrease in accuracy.

Answer 6: The elapsed time of our structure learning is 1 minute 14 seconds which is not very consistent as compared to the results shown in Table 2 of the research paper. The reason behind this is the time complexity on a single-core (turbo-boost) with a speed of 4.7GHz mentioned in table 2, while the processor we are working on i5 processor having a speed of 2.3GHz, which affects the runtime. To cross verify this result, I executed the same process on a different system with i3 processor at speed of 1.90GHz, which increased the runtime to 1 minute 37 seconds.

Answer 7:

Step3 BIC= -79265.445

Step 4 BIC = -14947.865

Step 3 has Ground-Truth graph and Step 4 has Learned graph which the algorithm learned from data.

We understand that there will be some difference between the two as machine learning approaches don't guarantee perfect solution.

Another possibility is that knowledge graph is not accurate which causes BIC in step3 to be very low(lower than Learned Graph BIC). Also the Ground-Truth graph has a greater number of free parameters than Learned Graph which decreases BIC.

This was not expected, we hoped to achieve a score closer to Ground-Truth graph.

Answer 8:

no. of free parameters in step 3 = 13729

no. of free parameters in step 4 = 221

The Step3 graph is the Ground-Truth graph, and in our graph we have a node “Diabetes” which has 4 parents, and this increases the number of free parameters greatly.

In the learned graph each node has a single parent and thus the number of free parameters is relatively low.

We expected numbers to be close, the directions of a lot of nodes is opposite to our Ground-Truth graph, due to which the number of free parameters is reduced drastically in output.

15.References

[1] R.J. Shiller, “Understanding recent trends in house prices and home ownership,” National Bureau of Economic Research, Working Paper 13553, Oct. 2007. DOI: 10.3386/w13553.

[2] R.A. Rahadi, S.K. Wiryono, D.P. Koesrindartotoor, and I.B. Syamwil, —Factors influencing the price of housing in Indonesia,‖ Int. J. Hous. Mark. Anal., vol. 8, no. 2, pp. 169–188, 2015.

[3] V. Limsombunchai, —House price prediction: Hedonic price model vs. artificial neural network,‖ Am.J. ..., 2004.

[4] D.X. Zhu and K.L. Wei, —The Land Prices and Housing Prices —Empirical Research Based on Panel Data of 11 Provinces and Municipalities in Eastern China,‖ Int. Conf. Manag. Sci. Eng., no. 2009, pp. 2118–2123, 2013.

[5] S. Kisilevich, D. Keim, and L. Rokach, —A GIS-based decision support system for hotel room rate estimation and temporal price prediction: The hotel brokers’ context,‖ Decis. Support Syst., vol. 54, no. 2, pp. 1119–1133, 2013.

[6] C.Y. Jim and W.Y. Chen, —Value of scenic views: Hedonic assessment of private housing in Hong Kong,‖ Landsc. Urban Plan., vol. 91, no. 4, pp. 226–234, 2009.

[7] Limsombunchai, Visit. "House price prediction: hedonic price model vs. artificial neural network."New Zealand Agricultural and Resource Economics Society Conference. 2004.

[8] Park, Byeonghwa, and Jae Kwon Bae. "Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data."Expert Systems with Applications 42.6 (2015): 2928-2934.

[9] Bhuriya, Dinesh, et al. "Stock market predication using a linear regression." Electronics, Communication and Aerospace Technology (ICECA), 2017 International conference of.Vol. 2.IEEE, 2017.

[10] Li, Li, and Kai-Hsuan Chu. "Prediction of real estate price variation based on economic parameters." Applied System Innovation (ICASI), 2017 International Conference on.IEEE, 2017.

[11] Using Bayesian Network for the Prediction and Diagnosis of Diabetes

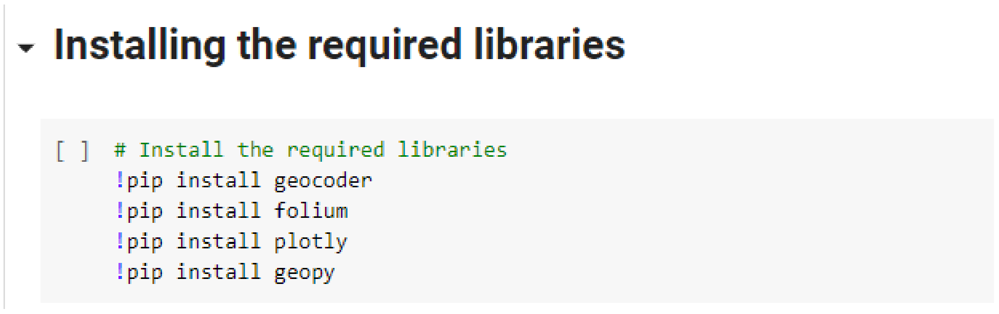
[12] Anthony C. Constantinou “Learning Bayesian Networks that enable full propagation of evidence”

16.Appendices

Code: The full notebook is available on Github through the following links:

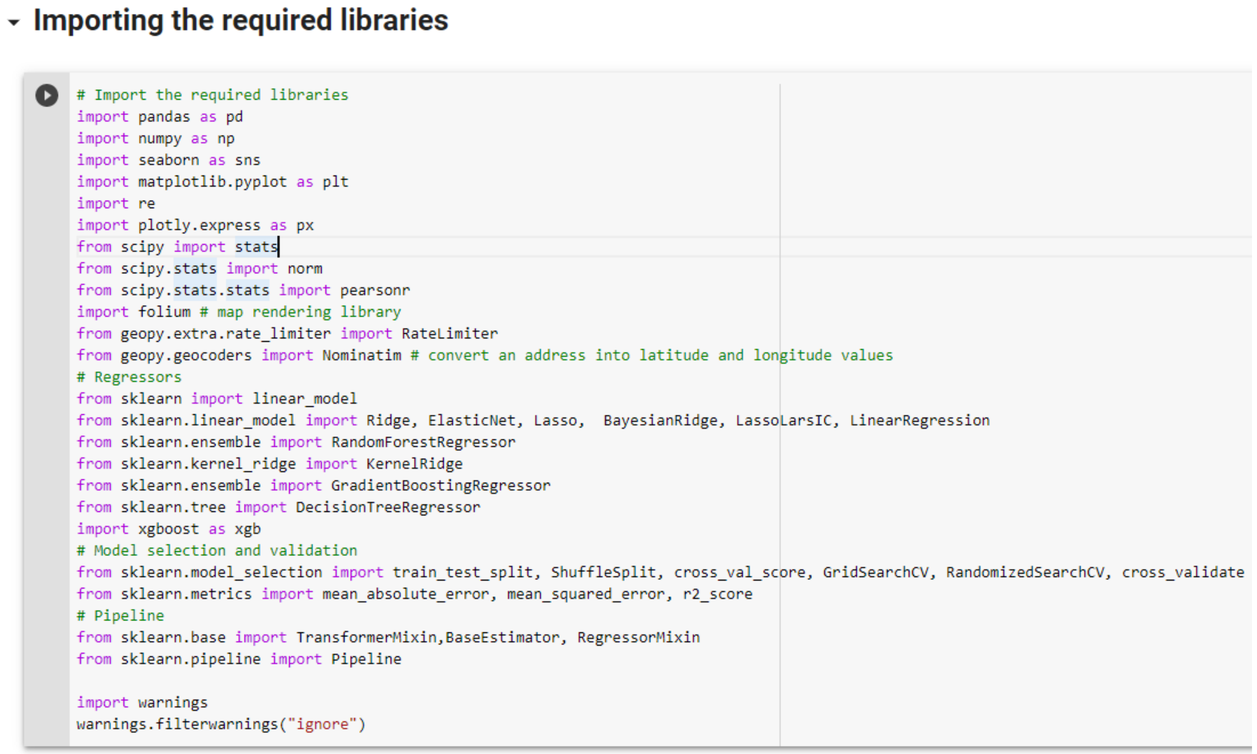
Notebook get Latitude & Longitude details(Data Collection) - https://github.com/niranjanganesan/Data-Analytics-Coursework/blob/master/Bangalore_House_Price_Prediction_GetLatLon.ipynb

Notebook to perform analysis and model building(Rest of the code) - https://github.com/niranjanganesan/Data-Analytics-Coursework/blob/master/Bangalore_House_Price_Prediction.ipynb



Installing the required libraries

```
[ ]  # Install the required libraries
     !pip install geocoder
     !pip install folium
     !pip install plotly
     !pip install geopy
```



Importing the required libraries

```
# Import the required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import re
import plotly.express as px
from scipy import stats
from scipy.stats import norm
from scipy.stats.stats import pearsonr
import folium # map rendering library
from geopy.extra.rate_limiter import RateLimiter
from geopy.geocoders import Nominatim # convert an address into latitude and longitude values
# Regressors
from sklearn import linear_model
from sklearn.linear_model import Ridge, ElasticNet, Lasso,  BayesianRidge, LassoLarsIC, LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
import xgboost as xgb
# Model selection and validation
from sklearn.model_selection import train_test_split, ShuffleSplit, cross_val_score, GridSearchCV, RandomizedSearchCV, cross_validate
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Pipeline
from sklearn.base import TransformerMixin,BaseEstimator, RegressorMixin
from sklearn.pipeline import Pipeline

import warnings
warnings.filterwarnings("ignore")
```

Creating Dataframe and viewing the structure of Bangalore House Dataset

```
# Import the kaggle dataset
house_df = pd.read_csv("Bengaluru_House_Data.csv")
house_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

Getting Location Coordinates

```
[ ] # Add city name to location column to get the latitude and longitude.
    house_df['location'] = house_df['location'].astype(str) + ', Bangalore'
```

```
[ ] # User defined function to get the location coordinates
    def get_latlng(location):
        # initialize your variable to None
        lat_lng_coords = None
        # loop until you get the coordinates
        while(lat_lng_coords is None):
            g = geocoder.arcgis('{}'.format(location))
            lat_lng_coords = g.latlng
        return lat_lng_coords
```

```
[ ] # Get the latitude and longitude coordinates
    coords = [ get_latlng(location) for location in house_df["location"].tolist() ]
```

```
[ ] # create temporary dataframe to populate the coordinates into Latitude and Longitude
    df_coords = pd.DataFrame(coords, columns=['Latitude', 'Longitude'])
```

```
[ ] # merge the coordinates into the original dataframe
    house_df['Latitude'] = df_coords['Latitude']
    house_df['Longitude'] = df_coords['Longitude']
```

```
[ ] # Save the data to a new file for further processing
    house_df.to_csv(r'Bengaluru_data.csv', index = False)
```

Structure of Bangalore house dataset along with coordinates

```
[ ] # Import dataset along with coordinates
    house_df = pd.read_csv("Bengaluru_data.csv")
    house_df.head()
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price	Latitude	Longitude
0	Super built-up Area	19-Dec	Electronic City Phase II, Bangalore	2 BHK	Coomee	1056	2.0	1.0	39.07	12.848672	77.677529
1	Plot Area	Ready To Move	Chikka Tirupathi, Bangalore	4 Bedroom	Theanmp	2600	5.0	3.0	120.00	12.904190	77.505480
2	Built-up Area	Ready To Move	Uttarahalli, Bangalore	3 BHK	NaN	1440	2.0	3.0	62.00	12.897570	77.528300
3	Super built-up Area	Ready To Move	Lingadheeranahalli, Bangalore	3 BHK	Soiewre	1521	3.0	1.0	95.00	13.001340	77.479150
4	Super built-up Area	Ready To Move	Kothanur, Bangalore	2 BHK	NaN	1200	2.0	1.0	51.00	13.064340	77.648550

Dataset structure and Data types of each features

```
[ ] # Check the no.of rows and columns
    house_df.shape
```

```
(13320, 11)
```

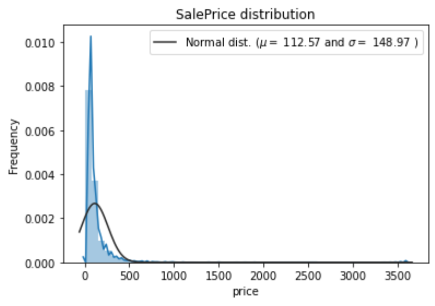
```
[ ] house_df.dtypes
```

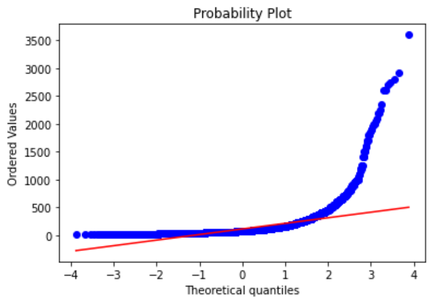
```
area_type        object
availability     object
location         object
size             object
society          object
total_sqft       object
bath            float64
balcony         float64
price           float64
Latitude        float64
Longitude       float64
dtype: object
```

Probability distribution of House Sale Price

```
[8] # Lets check the price distribution
    # Plot Histogram
    sns.distplot(house_df['price'] , fit=norm);

    # Get the fitted parameters used by the function
    (mu, sigma) = norm.fit(house_df['price'])
    print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
    plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
                loc='best')
    plt.ylabel('Frequency')
    plt.title('SalePrice distribution')

    fig = plt.figure()
    res = stats.probplot(house_df['price'], plot=plt)
    plt.show()

    print("Skewness: %f" % house_df['price'].skew())
    print("Kurtosis: %f" % house_df['price'].kurt())
```

```
mu = 112.57 and sigma = 148.97
```





```
Skewness: 8.064469
Kurtosis: 108.166513
```

```
[109] # Drop un-necessary columns such as availability and society, balcony and area type
      house_df.drop(['availability','society', 'area_type', 'balcony'], inplace=True, axis='columns')
      # Verify whether the columns are dropped
      house_df.columns
```

```
Index(['location', 'size', 'total_sqft', 'bath', 'price', 'Latitude',
       'Longitude'],
      dtype='object')
```

```
[110] # Removing rows that has missing values in each feature
      house_df = house_df.dropna()
      house_df.isnull().sum()
```

```
location      0
size          0
total_sqft    0
bath          0
price         0
Latitude      0
Longitude     0
dtype: int64
```

```
# Check any irrelevant records are present in these features

print("Location:", house_df['location'].unique(), "\n")
print("Size:", house_df['size'].unique(), "\n")
print("total_sqft:", house_df['total_sqft'].unique(), "\n")
print("bath:", house_df['bath'].unique(), "\n")
print("Price:", house_df['price'].unique())
```

```
Location: ['Electronic City Phase II, Bangalore' 'Chikka Tirupathi, Bangalore'
 'Uttarahalli, Bangalore' ...
 '12th cross srinivas nagar banshankari 3rd stage, Bangalore'
 'Havanur extension, Bangalore' 'Abshot Layout, Bangalore']

Size: ['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']

total_sqft: ['1056' '2600' '1440' ... '1133 - 1384' '774' '4689']

bath: [ 2.  5.  3.  4.  6.  1.  9.  8.  7. 11. 10. 14. 27. 12. 16. 40. 15. 13.
 18.]

Price: [ 39.07 120.    62.   ...  40.14 231.   488.  ]
```

Cleaning Bedroom Size feature

```
# Cleaning 'Size' column
house_df['Bedroom'] = house_df['size'].apply(lambda x: int(x.split(' ')[0]))
house_df.Bedroom.unique()
```

```
array([ 2,  4,  3,  6,  1,  8,  7,  5, 11,  9, 27, 10, 19, 16, 43, 14, 12,
       13, 18])
```

Cleaning House Area Size feature

```
[113] # Total_sqft feature is of data type object. Lets check for discrepencies in this feature.
      # Creating a function that will convert every valid values to float ex. 1056,
      # but if an unvalid value like 1133 - 1384 is fed, then it should not convert it into float and instead return false
      def is_float(x):
          try:
              float(x)
          except:
              return False
          return True
```

```
# Checking for vivid entries in total_sqft column
house_df[~house_df['total_sqft'].apply(is_float)].head(10)
```

	location	size	total_sqft	bath	price	Latitude	Longitude	Bedroom
30	Yelahanka, Bangalore	4 BHK	2100 - 2850	4.0	186.000	13.099310	77.592590	4
122	Hebbal, Bangalore	4 BHK	3067 - 8156	4.0	477.000	13.049810	77.589030	4
137	8th Phase JP Nagar, Bangalore	2 BHK	1042 - 1105	2.0	54.005	12.862699	77.573583	2
165	Sarjapur, Bangalore	2 BHK	1145 - 1340	2.0	43.490	12.860750	77.783550	2
188	KR Puram, Bangalore	2 BHK	1015 - 1540	2.0	56.800	12.997058	77.717327	2
410	Kengeri, Bangalore	1 BHK	34.46Sq. Meter	1.0	18.500	12.908700	77.487140	1
549	Hennur Road, Bangalore	2 BHK	1195 - 1440	2.0	63.770	13.031069	77.643897	2
648	Arekere, Bangalore	9 Bedroom	4125Perch	9.0	265.000	12.885680	77.596680	9
661	Yelahanka, Bangalore	2 BHK	1120 - 1145	2.0	48.130	13.099310	77.592590	2
672	Bettahalsoor, Bangalore	4 Bedroom	3090 - 5002	4.0	445.000	13.155020	77.605010	4

```
[115] # User defined function to take average of two values
      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
```

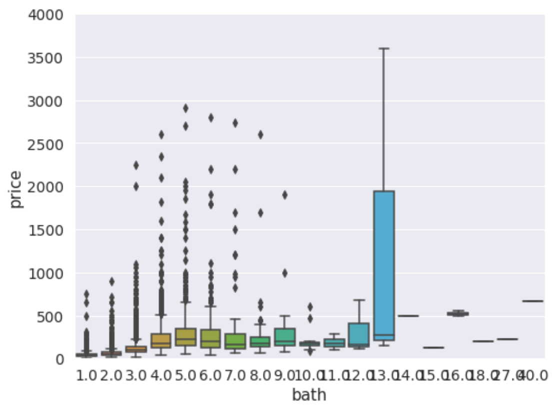
```
[116] # Cleaning 'total_sqft' feature
      house_df.total_sqft = house_df.total_sqft.apply(convert_sqft_to_num)
      house_df = house_df[house_df.total_sqft.notnull()]
      house_df.head(5)
```

	location	size	total_sqft	bath	price	Latitude	Longitude	Bedroom
0	Electronic City Phase II, Bangalore	2 BHK	1056.0	2.0	39.07	12.848672	77.677529	2
1	Chikka Tirupathi, Bangalore	4 Bedroom	2600.0	5.0	120.00	12.904190	77.505480	4
2	Uttarahalli, Bangalore	3 BHK	1440.0	2.0	62.00	12.897570	77.528300	3
3	Lingadheeranahalli, Bangalore	3 BHK	1521.0	3.0	95.00	13.001340	77.479150	3
4	Kothanur, Bangalore	2 BHK	1200.0	2.0	51.00	13.064340	77.648550	2

Data Visualization

Box Plot of Bathroom size vs Price

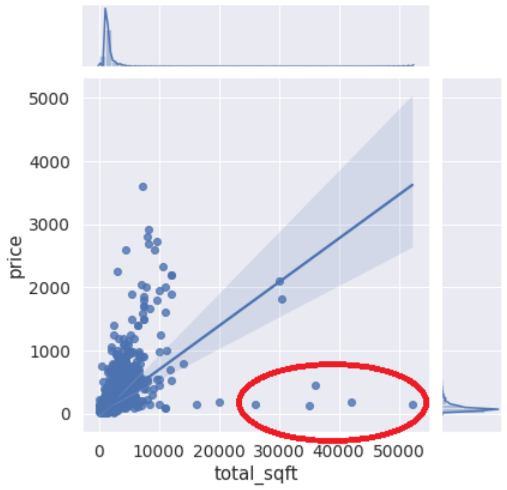
```
[25] # bath vs  Price
     var = 'bath'
     data = pd.concat([house_df['price'], house_df[var]], axis=1)
     f, ax = plt.subplots(figsize=(8, 6))
     fig = sns.boxplot(x=var, y="price", data=data)
     fig.axis(ymin=0, ymax=4000);
```



Regression Plot of Total sqft area of house vs Price

```
[26] # Square feet vs Sale Price
     sns.jointplot(x=house_df['total_sqft'], y=house_df['price'], kind='reg')
```

```
<seaborn.axisgrid.JointGrid at 0x7f0824c15278>
```

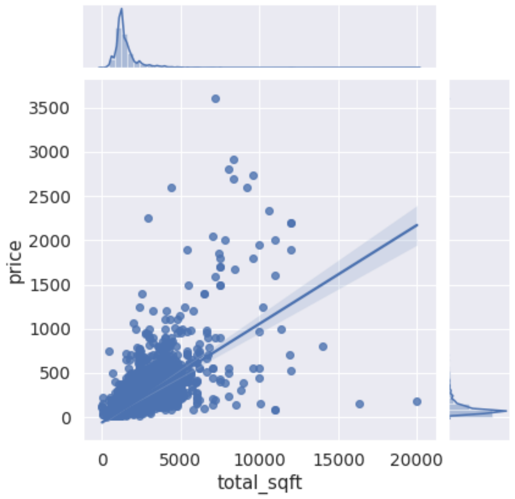


Removing outliers by observing the visual representation. The data points in the bottom right corner are considered as outliers

```
[27] # Removing outliers manually (Two points in the bottom right)
     house_df = house_df.drop(house_df[(house_df['total_sqft']>20000)
                                 & (house_df['price']<3000)].index).reset_index(drop=True)
```

```
[28] # Regression plot after removing outliers
     sns.jointplot(x=house_df['total_sqft'], y=house_df['price'], kind='reg')
```

```
<seaborn.axisgrid.JointGrid at 0x7f082490c908>
```

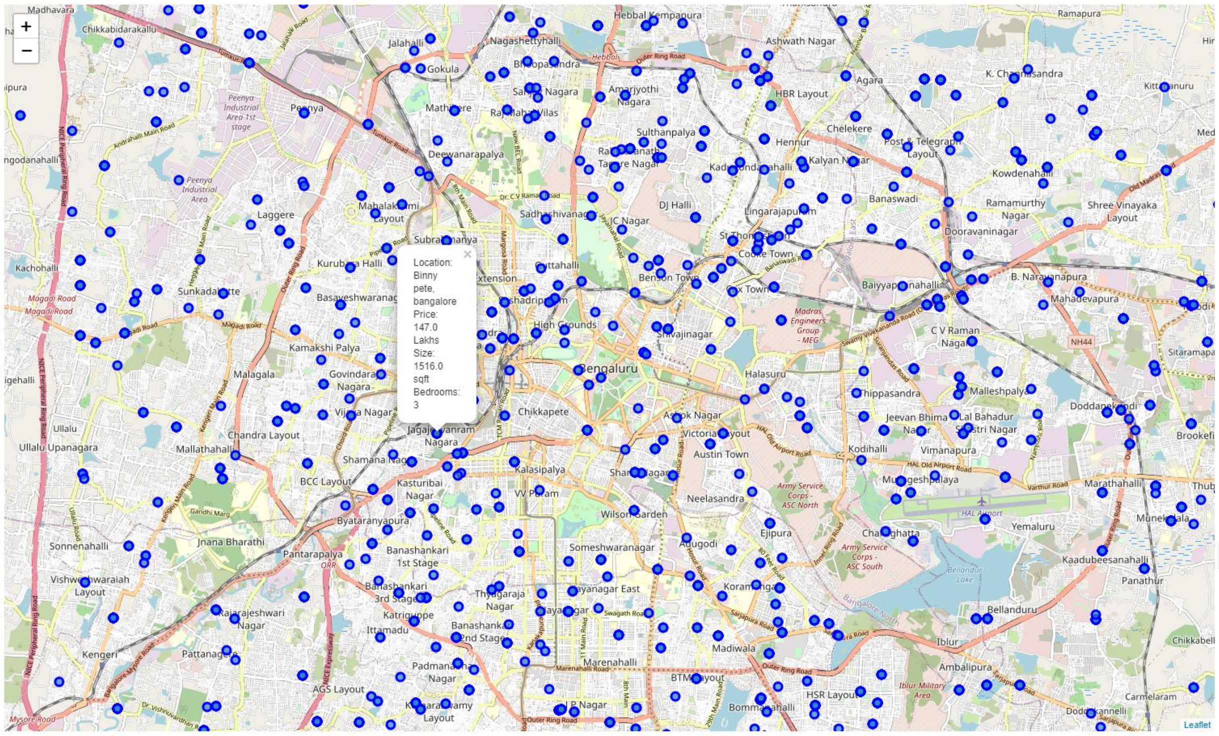


Data Visualization using Folium Map

```
[66] # get the coordinates of Bangalore
     address = 'Bangalore, India'

     geolocator = Nominatim(user_agent="my-application")
     location = geolocator.geocode(address)
     latitude = location.latitude
     longitude = location.longitude
     print('The geograpical coordinate of Bangalore, India {}, {}.'.format(latitude, longitude))
```

```
The geograpical coordinate of Bangalore, India 12.9791198, 77.5912997.
```

```
[67] house_map = folium.Map(location=[latitude, longitude], zoom_start=11)

     for lat, lon, location, price, sqft, bedroom in zip(house_df['Latitude'], house_df['Longitude'], house_df['location'], house_df['price'], house_df['total_sqft'], house_df['Bedroom']):
         folium.CircleMarker(
             [lat, lon],
             radius=5,
             popup = ('Location: ' + str(location).capitalize() + '<br>'
                      'Price: ' + str(price) + ' Lakhs' + '<br>'
                      'Size: ' + str(sqft) + ' sqft' + '<br>'
                      'Bedrooms: ' + str(bedroom)
                     ),
             color='blue',
             # key_on = color,
             # threshold_scale=[0,1,2,3],
             fill_color='#3186cc',
             fill=True,
             fill_opacity=0.7
             ).add_to(house_map)
     house_map
```



Each data points are viualized in map containg basic details of a property

Saving the map results in a html file

```
[68] # Save the results in an html file.
     house_map.save('house_prices.html')
```

Feature Engineering

Dimensionality Reduction - Dealing with location feature

Any location having less than 10 data points is tagged as 'other' location. By this way the number of categories can be reduced by huge amount. Later one hot encoding is done which has fewer dummy columns

```
[130] len(house_df['location'].unique())
```

```
1299
```

```
[131] # How many data points are available per location?
      # We will use strip() just to delete any leading or trailing space

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

```
Whitefield, Bangalore                        533
Sarjapur  Road, Bangalore                    392
Electronic City, Bangalore                   304
Kanakpura Road, Bangalore                    264
Thanisandra, Bangalore                       235
                                            ...
Puttappa Layout, Bangalore                     1
3rd Block HBR Layout, Bangalore                1
Gayathri Nagar, Bangalore                      1
Kasthuri Nagar East Of NGEF, Bangalore         1
Mathikere SBM colony, Bangalore                1
Name: location, Length: 1288, dtype: int64
```

```
[132] # Total number of locations in the dataset
      location_stats.values.sum()
```

```
13194
```

```
[133] # Total number of unique location having more than 10 houses
      len(location_stats[location_stats > 10])
```

```
240
```

```
[134] # Total number of unique locations
      len(location_stats)
```

```
1288
```

```
[135] # Total number of unique locations having less than 10 houses
      len(location_stats[location_stats <= 10])
```

```
1048
```

```
[136] # Unique locations having less than 10 houses
      location_stats_less_than_10 = location_stats[location_stats <= 10]
      location_stats_less_than_10
```

```
Basapura, Bangalore                         10
Sector 1 HSR Layout, Bangalore              10
1st Block Koramangala, Bangalore            10
Nagadevanahalli, Bangalore                  10
Nagappa Reddy Layout, Bangalore             10
                                            ..
Puttappa Layout, Bangalore                   1
3rd Block HBR Layout, Bangalore              1
Gayathri Nagar, Bangalore                    1
Kasthuri Nagar East Of NGEF, Bangalore       1
Mathikere SBM colony, Bangalore              1
Name: location, Length: 1048, dtype: int64
```

```
[137] # tagging the locations having less than 10 houses as 'other'
      house_df.location = house_df.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)
      len(house_df.location.unique())
```

```
241
```

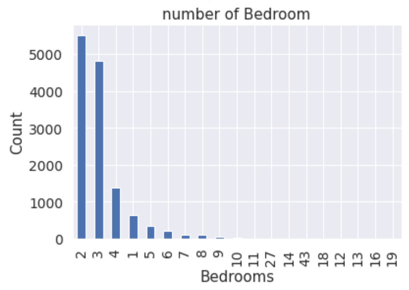
```
[138] house_df.head()
```

	location	size	total_sqft	bath	price	Latitude	Longitude	Bedroom	price_per_sqft
0	Electronic City Phase II, Bangalore	2 BHK	1056.0	2.0	39.07	12.848672	77.677529	2	3699.810606
1	Chikka Tirupathi, Bangalore	4 Bedroom	2600.0	5.0	120.00	12.904190	77.505480	4	4615.384615
2	Uttarahalli, Bangalore	3 BHK	1440.0	2.0	62.00	12.897570	77.528300	3	4305.555556
3	Lingadheeranahalli, Bangalore	3 BHK	1521.0	3.0	95.00	13.001340	77.479150	3	6245.890861
4	Kothanur, Bangalore	2 BHK	1200.0	2.0	51.00	13.064340	77.648550	2	4250.000000

Removing outliers from 'Bedroom' feature

```
[139] # Bar plot of Bedroom size
      house_df['Bedroom'].value_counts().plot(kind='bar')
      plt.title('number of Bedroom')
      plt.xlabel('Bedrooms')
      plt.ylabel('Count')
      sns.despine
```

```
<function seaborn.utils.despine>
```



Considering that the normal size of a bedroom as 300square ft, if the total area of the house is less than the sum of size of bedrooms then we consider them as outliers. For eg., If you have 400sqft house with 2 bedrooms they are considered as outliers as size of a single bedroom is considered to be 300 square ft.

```
[140] # Number of houses with area size less than the standard bedroom size.
      len(house_df[house_df['total_sqft']/house_df['Bedroom'] <300])
```

```
744
```

```
[141] # Few records from Bedroom outliers
      house_df[house_df.total_sqft/house_df.Bedroom <300].head()
```

	location	size	total_sqft	bath	price	Latitude	Longitude	Bedroom	price_per_sqft
9	other	6 Bedroom	1020.0	6.0	370.0	12.944780	77.572130	6	36274.509804
45	HSR Layout, Bangalore	8 Bedroom	600.0	9.0	200.0	12.912160	77.644900	8	33333.333333
57	Murugeshpalya, Bangalore	6 Bedroom	1407.0	4.0	150.0	12.955650	77.653350	6	10660.980810
67	Devarachikkanahalli, Bangalore	8 Bedroom	1350.0	7.0	85.0	12.888330	77.617640	8	6296.296296
69	other	3 Bedroom	500.0	3.0	100.0	12.964806	77.597001	3	20000.000000

Here we can seee that size of 8 bedroom house is 600 sqft which is not acceptable. Hence they are considered as outliers.

```
[142] # Removing the bedroom outliers
      house_df = house_df[~(house_df.total_sqft/house_df.Bedroom < 300)]
```

```
[143] # Shape of the dataset after removing outliers
      house_df.shape
```

```
(12450, 9)
```

Removing outliers from data related to price and total sqft per location.

```
[128] # Creating a new feature called 'price_pre_sqft' to remove outliers from data related to price and total sqft per location.
      house_df['price_per_sqft'] = house_df['price']*100000/house_df['total_sqft']
      house_df.head()
```

	location	size	total_sqft	bath	price	Latitude	Longitude	Bedroom	price_per_sqft
0	Electronic City Phase II, Bangalore	2 BHK	1056.0	2.0	39.07	12.848672	77.677529	2	3699.810606
1	Chikka Tirupathi, Bangalore	4 Bedroom	2600.0	5.0	120.00	12.904190	77.505480	4	4615.384615
2	Uttarahalli, Bangalore	3 BHK	1440.0	2.0	62.00	12.897570	77.528300	3	4305.555556
3	Lingadheeranahalli, Bangalore	3 BHK	1521.0	3.0	95.00	13.001340	77.479150	3	6245.890861
4	Kothanur, Bangalore	2 BHK	1200.0	2.0	51.00	13.064340	77.648550	2	4250.000000

```
[144] # Checking the statistics of 'price_per_sqft' feature
      house_df['price_per_sqft'].describe()
```

```
count     12450.000000
mean       6310.699306
std        4167.544819
min         500.000000
25%        4210.526316
50%        5294.117647
75%        6916.666667
max      176470.588235
Name: price_per_sqft, dtype: float64
```

Here we can see that min price per sqft is 500 and max price per sqft is 176470. which shows wide variation in property prices. We remove outliers per location using mean and std deviation.

```
[145] # Function to remove outliers from price_per_sqft based on locations as every location will have different price range.

      def remove_pps_outliers(house_df):
          df_out = pd.DataFrame()
          for key, subdf in house_df.groupby('location'): # 'key' variable stores cities and 'subdf' stores rows for each city
              m = np.mean(subdf['price_per_sqft'])
              st = np.std(subdf['price_per_sqft'])
              # data without outliers:
              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
```

```
[146] # Removing the outliers and checking the shape of the dataset
      house_df = remove_pps_outliers(house_df)
      house_df.shape
```

```
(10240, 9)
```

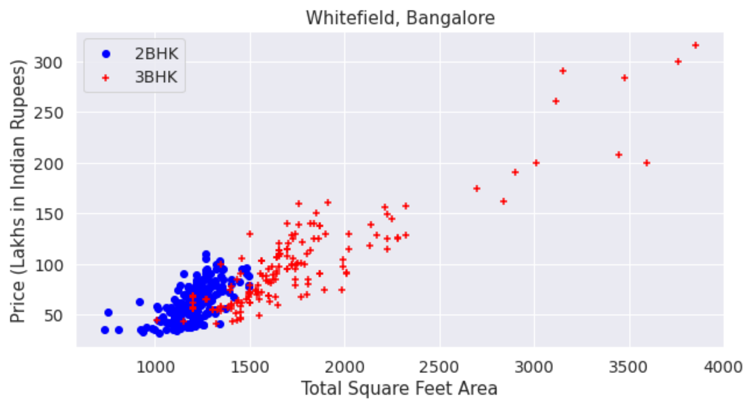
Removing outliers related to bedroom size and location

If the sqft area of 2-Bedroom house is more than sqft area of 3Bedroom house, then the price of 2-Bedroom being more than 3-Bedroom is justifiable. But, in some cases, the price of 2-Bedroom houses are more than 3-Bedroom houses even with the same sqft area or less, this could be because of many reasons like the 2-Bedroom house can be in some prime location and thats the reason it might be costly. We will investigate by scatter plot of location vs price vs Bedroom size

Let's check for a given location how does the 2 BHK and 3 BHK property prices look like

```
# User defined function to plot 2 and 3 bedrooms
def plot_price_per_sqft(house_df, location):
    bhk2 = house_df[(house_df['location'] == location) & (house_df['Bedroom'] == 2)]
    bhk3 = house_df[(house_df['location'] == location) & (house_df['Bedroom'] == 3)]
    plt.figure(figsize = (10,5))
    plt.scatter(bhk2['total_sqft'], bhk2['price'],color = 'blue', label = '2BHK',)
    plt.scatter(bhk3['total_sqft'], bhk3['price'], marker = '+',color = 'red', label = '3BHK')
    plt.xlabel('Total Square Feet Area')
    plt.ylabel('Price (Lakhs in Indian Rupees)')
    plt.title(location)
    plt.legend()
```

```
[51] # Plot of 2 and 3 Bedroom size vs Price for a location
     plot_price_per_sqft(house_df, 'Whitefield, Bangalore')
```



Now we remove those 2 Bedroom houses whose price_per_sqft is less than mean price_per_sqft of 1 Bedroom house.

```
[53] # user defined function to remove the outliers using mean and std deviation
     def remove_bhk_outliers(house_df):
         exclude_indices = np.array([])
         for location, location_df in house_df.groupby('location'):
             bhk_stats = {}
             for bhk, bhk_df in location_df.groupby('Bedroom'):
                 bhk_stats[bhk] = {
                     'mean': np.mean(bhk_df.price_per_sqft),
                     'std': np.std(bhk_df.price_per_sqft),
                     'count': bhk_df.shape[0] #shape would have given RowsXColumns, we want only number of rows, so shape[0]
                 }
             for bhk, bhk_df in location_df.groupby('Bedroom'):
                 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 house_df.drop(exclude_indices,axis='index')
```

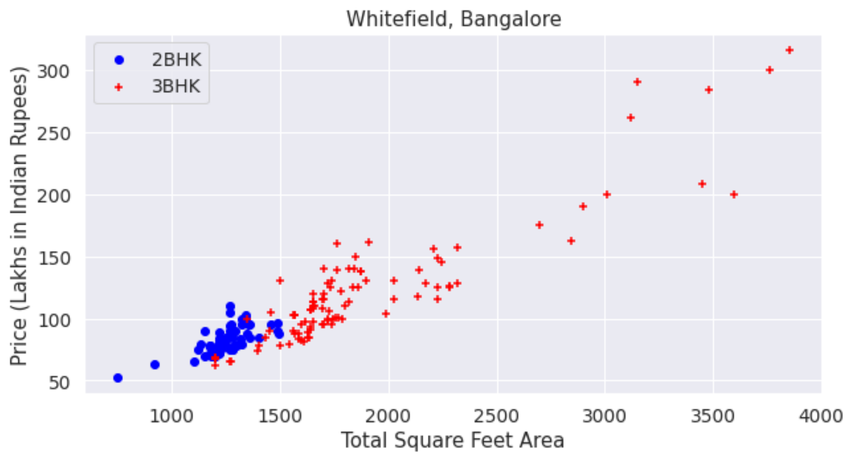
```
[54] # Shape of the dataset before removing outliers
     house_df.shape
```

```
(10240, 9)
```

```
[55] # Shapre of dataset after removing outliers
     house_df = remove_bhk_outliers(house_df)
     house_df.shape
```

```
(7317, 9)
```

```
[56] # Bedroom size vs Price plot per location after removing outliers
     plot_price_per_sqft(house_df, 'Whitefield, Bangalore')
```



Removing Outliers in Number of bathroom feature.

```
[57] # Checking the number of bathrooms in a house.
     house_df['bath'].unique()
```

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

```
[58] # Number of Bedrooms and Bathrooms
     house_df[['Bedroom','bath']]
```

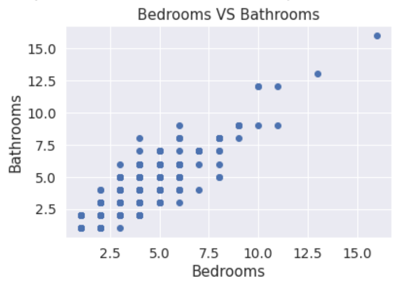
	Bedroom	bath
0	4	4.0
1	3	3.0
2	3	2.0
3	3	2.0
4	2	2.0
...	...	...
10231	2	2.0
10232	1	1.0
10235	2	2.0
10236	1	1.0
10239	4	5.0

7317 rows × 2 columns

Investigating outlier using scatter plot.

```
[59] # Scatter plot BHK VS Bath
     plt.scatter(data = house_df, x = 'Bedroom' , y = 'bath')
     plt.xlabel('Bedrooms')
     plt.ylabel('Bathrooms')
     plt.title('Bedrooms VS Bathrooms')
```

```
Text(0.5, 1.0, 'Bedrooms VS Bathrooms')
```



From scatter plot it can be seen that we have more number of bathrooms than bedrooms in a house. For example for 4 Bedroom house we have 6,7 and 8 bathrooms which doesn't look right. Having 2 more bathrooms than number of bedrooms in a house is unusual.

For a 4 Bedroom house there could be 4+2 bathrooms. Anything above these are outliers in the dataset.

```
[60] # Checking the records which have 2 plus more bathrooms than bedrooms.
     house_df[(house_df['bath']) > (house_df['Bedroom']+2)]
```

	location	size	total_sqft	bath	price	Latitude	Longitude	Bedroom	price_per_sqft
1626	Chikkabanavar, Bangalore	4 Bedroom	2460.0	7.0	80.0	13.081540	77.502920	4	3252.032520
5238	Nagasandra, Bangalore	4 Bedroom	7000.0	8.0	450.0	13.044950	77.501560	4	6428.571429
6711	Thanisandra, Bangalore	3 BHK	1806.0	6.0	116.0	13.055990	77.632550	3	6423.034330
8407	other	6 BHK	11338.0	9.0	1000.0	12.976197	77.768615	6	8819.897689

```
[61] # Shape of dataset before removing outliers
     house_df.shape
```

```
(7317, 9)
```

```
[62] # Removing the outliers
     house_df = house_df[(house_df['bath']) < (house_df['Bedroom']+2)]
```

```
[63] # Shape of dataset after removing outliers
     house_df.shape
```

```
(7239, 9)
```

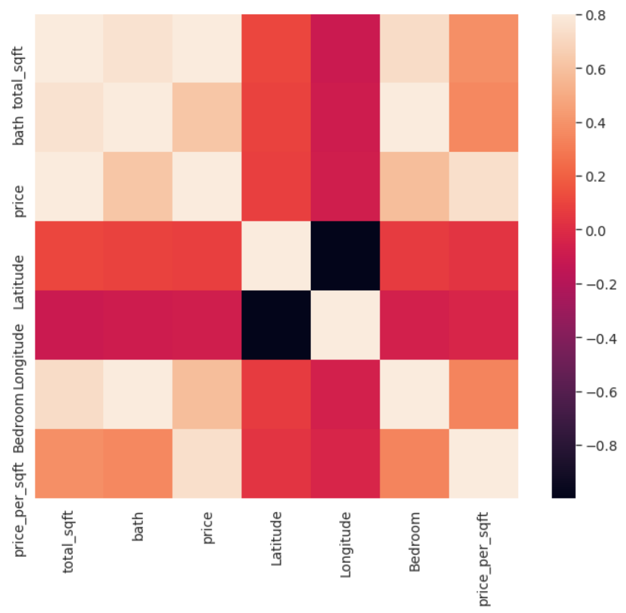
```
[64] # Few records after removing these outliers
     house_df.head()
```

	location	size	total_sqft	bath	price	Latitude	Longitude	Bedroom	price_per_sqft
0	1st Block Jayanagar, Bangalore	4 BHK	2850.0	4.0	428.0	12.9395	77.58981	4	15017.543860
1	1st Block Jayanagar, Bangalore	3 BHK	1630.0	3.0	194.0	12.9395	77.58981	3	11901.840491
2	1st Block Jayanagar, Bangalore	3 BHK	1875.0	2.0	235.0	12.9395	77.58981	3	12533.333333
3	1st Block Jayanagar, Bangalore	3 BHK	1200.0	2.0	130.0	12.9395	77.58981	3	10833.333333
4	1st Block Jayanagar, Bangalore	2 BHK	1235.0	2.0	148.0	12.9395	77.58981	2	11983.805668

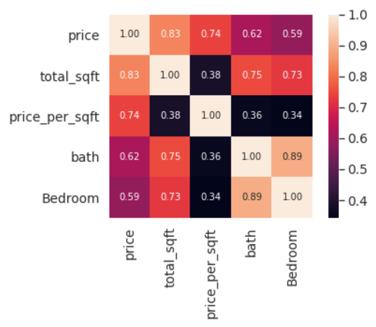
Modelling

Feature Selection using Correlation Matrix heatmap

```python
# Correlation Matrix Heatmap
corrmat = house_df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
```



```python
[72] # Top Heatmap
     k = 5 #number of variables for heatmap
     cols = corrmat.nlargest(k, 'price')['price'].index
     cm = np.corrcoef(house_df[cols].values.T)
     sns.set(font_scale=1.25)
     hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
     plt.show()
```



```python
[73] #finding most correlated Features
     most_corr = pd.DataFrame(cols)
     most_corr.columns = ['Most Correlated Features']
     most_corr
```

	Most Correlated Features
0	price
1	total_sqft
2	price_per_sqft
3	bath
4	Bedroom

```
[74] # getting rid of unnecessary columns
     df = house_df.drop(columns = ['Latitude', 'Longitude', 'price_per_sqft', 'size'])
     df.head()
```

	location	total_sqft	bath	price	Bedroom
0	1st Block Jayanagar, Bangalore	2850.0	4.0	428.0	4
1	1st Block Jayanagar, Bangalore	1630.0	3.0	194.0	3
2	1st Block Jayanagar, Bangalore	1875.0	2.0	235.0	3
3	1st Block Jayanagar, Bangalore	1200.0	2.0	130.0	3
4	1st Block Jayanagar, Bangalore	1235.0	2.0	148.0	2

One Hot Encoding of Location feature

```
[75] dummies = pd.get_dummies(df['location'])
     dummies.head()
```

	1st Block Jayanagar, Bangalore	1st Phase JP Nagar, Bangalore	2nd Phase Judicial Layout, Bangalore	2nd Stage Nagarbhavi, Bangalore	5th Block Hbr Layout, Bangalore	5th Phase JP Nagar, Bangalore	6th Phase JP Nagar, Bangalore	7th Phase JP Nagar, Bangalore	8th Phase JP Nagar, Bangalore	9th Phase JP Nagar, Bangalore	AECS Layout, Bangalore	Abbigere, Bangalore	Aks Na Banga
0	1	0	0	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	0	0	
4	1	0	0	0	0	0	0	0	0	0	0	0	

5 rows × 241 columns

```
[76] # To avoid dummy trap variable trap, lets drop any one of the column i.e lets drop the 'other' column
     df = pd.concat([df, dummies.drop('other',axis='columns')],axis='columns')
     df.head()
```

	location	total_sqft	bath	price	Bedroom	1st Block Jayanagar, Bangalore	1st Phase JP Nagar, Bangalore	2nd Phase Judicial Layout, Bangalore	2nd Stage Nagarbhavi, Bangalore	5th Block Hbr Layout, Bangalore	5th Phase JP Nagar, Bangalore	6th Phase JP Nagar, Bangalore	7th Phase JP Nagar, Bangalore	8th Phase JP Nagar, Bangalore	9th Phase JP Nagar, Bangalore	AECS Layout, Bangalore	Abbigere, Bangalore	Akshaya Nagar, Bangalore	Am
0	1st Block Jayanagar, Bangalore	2850.0	4.0	428.0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	
1	1st Block Jayanagar, Bangalore	1630.0	3.0	194.0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	
2	1st Block Jayanagar, Bangalore	1875.0	2.0	235.0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	
3	1st Block Jayanagar, Bangalore	1200.0	2.0	130.0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	
4	1st Block Jayanagar, Bangalore	1235.0	2.0	148.0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	

5 rows × 245 columns

```
[77] df = df.drop('location',axis='columns')
     df.head(2)
```

	total_sqft	bath	price	Bedroom	1st Block Jayanagar, Bangalore	1st Phase JP Nagar, Bangalore	2nd Phase Judicial Layout, Bangalore	2nd Stage Nagarbhavi, Bangalore	5th Block Hbr Layout, Bangalore	5th Phase JP Nagar, Bangalore	6th Phase JP Nagar, Bangalore	7th Phase JP Nagar, Bangalore	8th Phase JP Nagar, Bangalore	9th Phase JP Nagar, Bangalore	AECS Layout, Bangalore	Abbigere, Bangalore	Akshaya Nagar, Bangalore	Ambalipura, Bangalore
0	2850.0	4.0	428.0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1630.0	3.0	194.0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0

2 rows × 244 columns

```
[78] target = df['price']
     features =  df.drop(columns = ['price'], axis = 'columns')
```

```
[79] features.head()
```

	total_sqft	bath	Bedroom	1st Block Jayanagar, Bangalore	1st Phase JP Nagar, Bangalore	2nd Phase Judicial Layout, Bangalore	2nd Stage Nagarbhavi, Bangalore	5th Block Hbr Layout, Bangalore	5th Phase JP Nagar, Bangalore	6th Phase JP Nagar, Bangalore	7th Phase JP Nagar, Bangalore	8th Phase JP Nagar, Bangalore	9th Phase JP Nagar, Bangalore	AECS Layout, Bangalore	Abbigere, Bangalore	Akshaya Nagar, Bangalore	Ambalipura, Bangalore	Ambedl Naga Bangal
0	2850.0	4.0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1630.0	3.0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	1875.0	2.0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	1200.0	2.0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	1235.0	2.0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	

5 rows × 243 columns

```
[80] features.shape
```

```
(7239, 243)
```

```
[81] target.head()
```

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

```
[82] target.shape
```

```
(7239,)
```

Train Test Split

```
[83] # Split data into train and test set

     X_train, X_test, y_train, y_test = train_test_split(features, target, test_size = 0.2, random_state = 10)

     # View number of training and testing data
     print('Training Dependent variable contains:', len(y_train), 'rows')
     print('Training Independent variable contains:', len(X_train), 'rows')
     print('Testing Dependent variable contains:', len(y_test), 'rows')
     print('Testing Independent variable contains:', len(X_test), 'rows')
```

```
Training Dependent variable contains: 5791 rows
Training Independent variable contains: 5791 rows
Testing Dependent variable contains: 1448 rows
Testing Independent variable contains: 1448 rows
```

Linear Regression

```
[84] # Linear Regression Model

     lr_model = LinearRegression()
     lr_model.fit(X_train, y_train)
     lr_score = lr_model.score(X_test, y_test)
     print(' Linear Regression R^2 for test is: ', lr_score)
```

```
 Linear Regression R^2 for test is:  0.8714168721388086
```

The r-squared value is the measure of how close the data are to the fitted regression line. It takes a value between 0 and 1, 1 meaning that all of the variance in the target is explained by the data. In general, a higher r-squared value means a better fit.

```
[85] #The RMSE measures the distance between our predicted values and actual values.
     lr_predictions = lr_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, lr_predictions))
```

```
RMSE is:  700.3654852218536
```

```
[86] # Cross validation
     cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

     lr_cross_val = cross_val_score(LinearRegression(), features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (lr_cross_val.mean()*100.0, lr_cross_val.std()*100.0))
```

```
Accuracy(Cross Validation Score): 83.389006% (2.854%)
```

Let's predict house prices for some locations

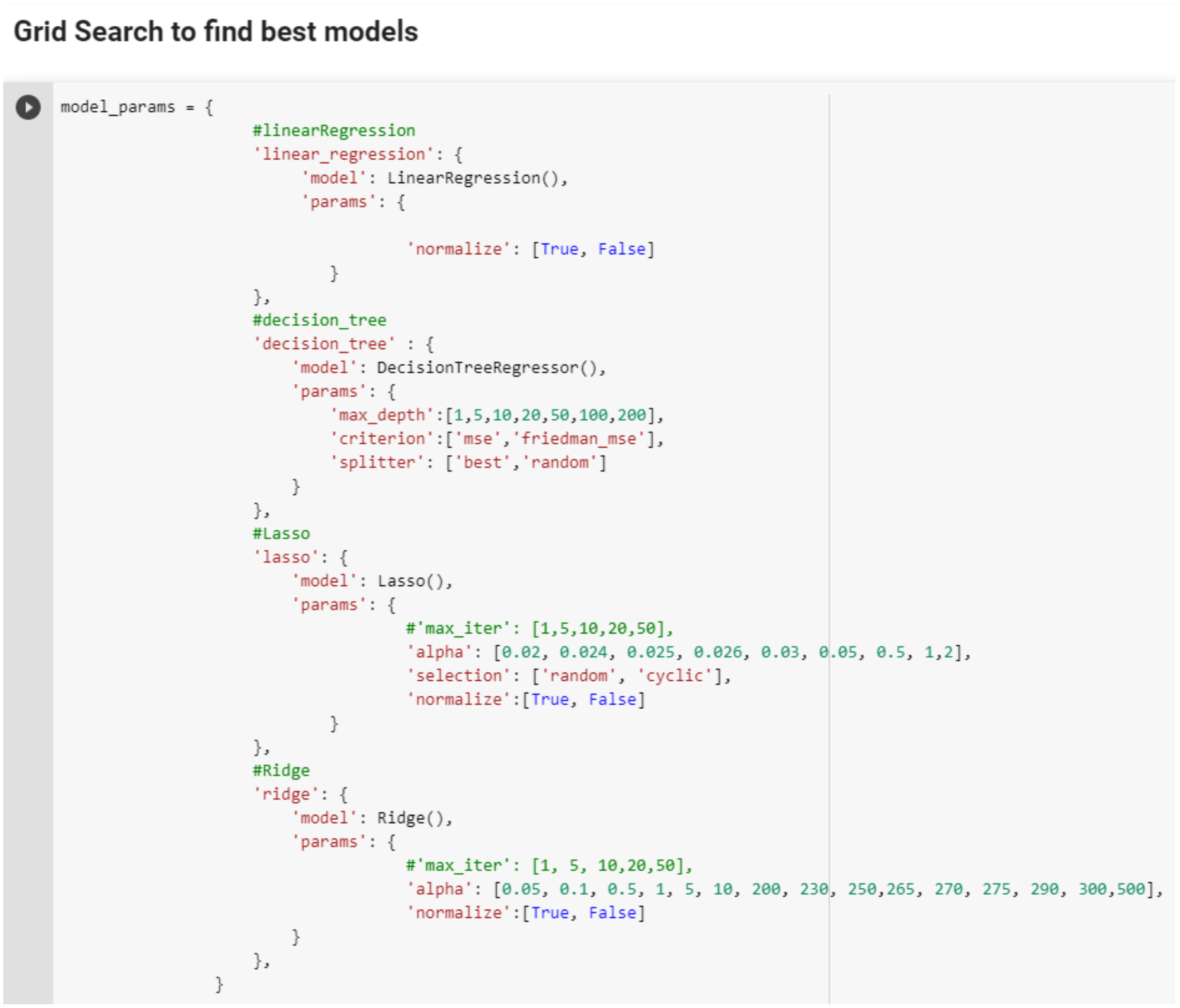
```
[87] def predict_price(location,total_sqft,bath,Bedroom):
         loc_index = np.where(features.columns==location)[0][0]

         x = np.zeros(len(features.columns))
         x[0] = total_sqft
         x[1] = bath
         x[2] = Bedroom
         if loc_index >= 0:
             x[loc_index] = 1

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

```
[88] # Predicting in Whitefield, Bangalore with sqft size as 1280, bedroom size as 2 and bathroom size as 2.
     predict_price('Whitefield, Bangalore',1280, 2, 2)
```

```
74.57376538044042
```

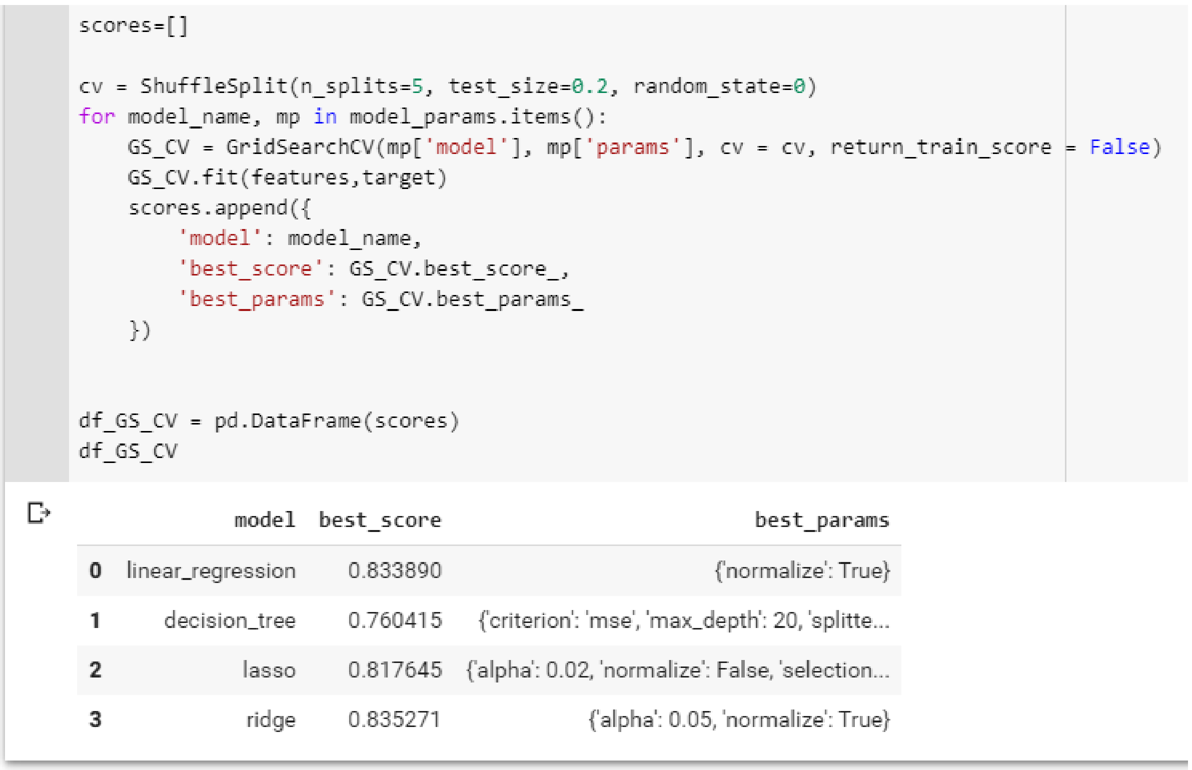
```
[89] # Predicting in 1st Phase JP Nagar, Bangalore with sqft size as 1394, bedroom size as 2 and bathroom size as 2.
     predict_price('1st Phase JP Nagar, Bangalore',1394, 2, 2)
```

```
117.11819533124137
```



Grid Search to find best models

```
model_params = {
                    #linearRegression
                    'linear_regression': {
                         'model': LinearRegression(),
                         'params': {

                                    'normalize': [True, False]
                            }
                    },
                    #decision_tree
                    'decision_tree' : {
                        'model': DecisionTreeRegressor(),
                        'params': {
                            'max_depth':[1,5,10,20,50,100,200],
                            'criterion':['mse','friedman_mse'],
                            'splitter': ['best','random']
                        }
                    },
                    #Lasso
                    'lasso': {
                        'model': Lasso(),
                        'params': {
                                    #'max_iter': [1,5,10,20,50],
                                    'alpha': [0.02, 0.024, 0.025, 0.026, 0.03, 0.05, 0.5, 1,2],
                                    'selection': ['random', 'cyclic'],
                                    'normalize':[True, False]
                            }
                    },
                    #Ridge
                    'ridge': {
                        'model': Ridge(),
                        'params': {
                                    #'max_iter': [1, 5, 10,20,50],
                                    'alpha': [0.05, 0.1, 0.5, 1, 5, 10, 200, 230, 250,265, 270, 275, 290, 300,500],
                                    'normalize':[True, False]
                        }
                    },
                }
```

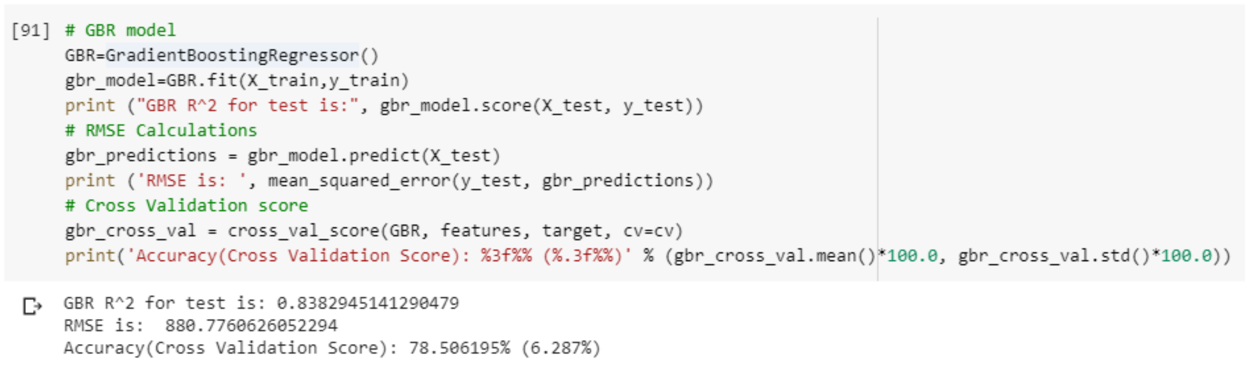


```
scores=[]

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
for model_name, mp in model_params.items():
    GS_CV = GridSearchCV(mp['model'], mp['params'], cv = cv, return_train_score = False)
    GS_CV.fit(features,target)
    scores.append({
        'model': model_name,
        'best_score': GS_CV.best_score_,
        'best_params': GS_CV.best_params_
    })


df_GS_CV = pd.DataFrame(scores)
df_GS_CV
```

	model	best_score	best_params
0	linear_regression	0.833890	{'normalize': True}
1	decision_tree	0.760415	{'criterion': 'mse', 'max_depth': 20, 'splitte...
2	lasso	0.817645	{'alpha': 0.02, 'normalize': False, 'selection...
3	ridge	0.835271	{'alpha': 0.05, 'normalize': True}

Grid Search return Ridge regression as the best model.

Gradient Boosting Regressor



```
[91] # GBR model
     GBR=GradientBoostingRegressor()
     gbr_model=GBR.fit(X_train,y_train)
     print ("GBR R^2 for test is:", gbr_model.score(X_test, y_test))
     # RMSE Calculations
     gbr_predictions = gbr_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, gbr_predictions))
     # Cross Validation score
     gbr_cross_val = cross_val_score(GBR, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (gbr_cross_val.mean()*100.0, gbr_cross_val.std()*100.0))
```

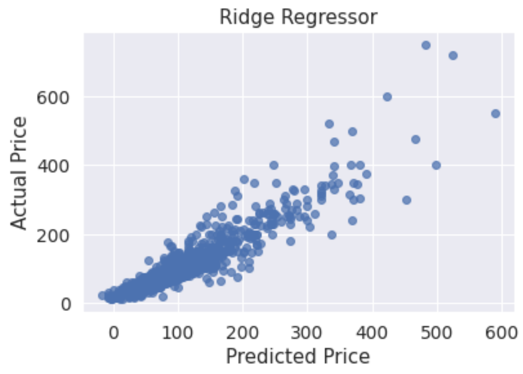
```
GBR R^2 for test is: 0.8382945141290479
RMSE is:  880.7760626052294
Accuracy(Cross Validation Score): 78.506195% (6.287%)
```

Ridge Regression

```
[93] ridge=Ridge()
     ridge_model=ridge.fit(X_train,y_train)
     print ("Ridge R^2 for test is:", ridge_model.score(X_test, y_test))
     # RMSE Calculations
     ridge_predictions = ridge_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, ridge_predictions))
     # Cross Validation score
     ridge_cross_val = cross_val_score(ridge, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (ridge_cross_val.mean()*100.0, ridge_cross_val.std()*100.0))
```

```
Ridge R^2 for test is: 0.8681384119227891
RMSE is:  718.2225743918403
Accuracy(Cross Validation Score): 83.338816% (2.852%)
```

```
[94] # Plot of variation between predicted price and actual price
     actual_values = y_test
     plt.scatter(ridge_predictions, actual_values, alpha=.75,
                 color='b') #alpha helps to show overlapping data
     plt.xlabel('Predicted Price')
     plt.ylabel('Actual Price')
     plt.title('Ridge Regressor')
     plt.show()
```



Kernel Ridge Regressor

```
[95] kridge=KernelRidge()
     kridge_model=kridge.fit(X_train,y_train)
     print ("Kernel Ridge R^2 for test is:", kridge_model.score(X_test, y_test))
     # RMSE Calculations
     kridge_predictions = kridge_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, kridge_predictions))
     # Cross Validation score
     kridge_cross_val = cross_val_score(kridge, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (kridge_cross_val.mean()*100.0, kridge_cross_val.std()*100.0))
```

```
Kernel Ridge R^2 for test is: 0.8676392821998321
RMSE is:  720.9412299139298
Accuracy(Cross Validation Score): 83.327181% (2.938%)
```

Random Forest Regressor

```
[96] Rf=RandomForestRegressor()
     Rf_model=Rf.fit(X_train,y_train)
     print ("Random Forest R^2 for test is:", Rf_model.score(X_test, y_test))
     # RMSE Calculations
     Rf_predictions = Rf_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, Rf_predictions))
     # Cross Validation score
     Rf_cross_val = cross_val_score(Rf, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (Rf_cross_val.mean()*100.0, Rf_cross_val.std()*100.0))
```

```
Random Forest R^2 for test is: 0.799058369130084
RMSE is:  1094.4871628679264
Accuracy(Cross Validation Score): 77.246292% (5.449%)
```

Elastic Net Regressor

```
[97] Enet=ElasticNet()
     Enet_model=Enet.fit(X_train,y_train)
     print ("Elastic Net R^2 for test is:", Enet_model.score(X_test, y_test))
     # RMSE Calculations
     Enet_predictions = Enet_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, Enet_predictions))
     # Cross Validation score
     Enet_cross_val = cross_val_score(Enet, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (Enet_cross_val.mean()*100.0, Enet_cross_val.std()*100.0))
```

```
Elastic Net R^2 for test is: 0.7366826565223646
RMSE is:  1434.23466281772
Accuracy(Cross Validation Score): 70.803299% (2.230%)
```

Lasso Regressor

```
[98] LassoR=Lasso()
     LassoR_model=LassoR.fit(X_train,y_train)
     print ("Lasso R^2 for test is:", LassoR_model.score(X_test, y_test))
     # RMSE Calculations
     LassoR_predictions = LassoR_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, LassoR_predictions))
     # Cross Validation score
     LassoR_cross_val = cross_val_score(LassoR, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (LassoR_cross_val.mean()*100.0, LassoR_cross_val.std()*100.0))
```

```
Lasso R^2 for test is: 0.7406822698085469
RMSE is:  1412.4496032498678
Accuracy(Cross Validation Score): 70.762387% (2.484%)
```

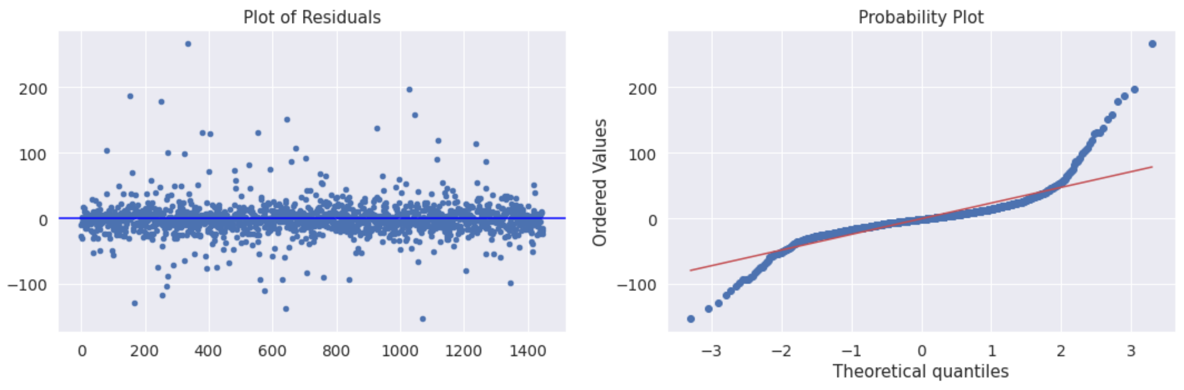
XGB Regressor

```
[99] XGB=xgb.XGBRegressor()
     XGB_model=XGB.fit(X_train,y_train)
     print ("XGB R^2 for test is:", XGB_model.score(X_test, y_test))
     # RMSE Calculations
     XGB_predictions = XGB_model.predict(X_test)
     print ('RMSE is: ', mean_squared_error(y_test, XGB_predictions))
     # Cross Validation score
     XGB_cross_val = cross_val_score(XGB, features, target, cv=cv)
     print('Accuracy(Cross Validation Score): %3f%% (%.3f%%)' % (XGB_cross_val.mean()*100.0, XGB_cross_val.std()*100.0))
```

```
[13:19:38] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGB R^2 for test is: 0.8364497053021998
RMSE is:  890.8243516043259
[13:19:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[13:19:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[13:19:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[13:19:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[13:19:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Accuracy(Cross Validation Score): 77.274904% (6.802%)
```

Residual Plot

```
[101] # calculate the residuals
      ridge_preds = pd.DataFrame(ridge_predictions)
      y_test = y_test.reset_index(drop=True)
      residuals = y_test - ridge_preds[0]

      # Plotting residual and probability graph
      plt.figure(figsize=(18,5))
      plt.subplot(1,2,1)
      plt.axhline(0, color='blue')
      plt.title('Plot of Residuals')
      plt.scatter(residuals.index, residuals, s=20)

      plt.subplot(1,2,2)
      plt.title('Probability Plot')
      stats.probplot(residuals, dist='norm', plot=plt)
      plt.show()
```


Plot of Residuals
Probability Plot
Ordered Values
Theoretical quantiles