SYMBIOSIS
INSTITUTE OF GEOINFORMATICS

PROJECT REPORT SUBMITTED TO
Symbiosis Institute of Geoinformatics

FOR PARTIAL FULFILLMENT OF THE M. Sc. DEGREE

By
Dilpreet Kaur
PRN-22070243017

M.Sc. (Data Science and Spatial Analytics)

BATCH-2022-24

MACHINE LEARNING AND PYTHON MINI PROJECT ON

TOPIC: HOUSING PRICE PREDICTION

Symbiosis International (Deemed University)
5th Floor, Atur Centre
Gokhale Cross Road
Model Colony
Pune – 411016
Maharashtra
India

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr. Vidya Patkar and Mr. Sahil Shah for their invaluable guidance, encouragement, and support throughout the project. This project would not have been possible without their expertise and unwavering support.

I would like to extend my gratitude to Symbiosis Institute of Geoinformatics for their invaluable contributions, support, and cooperation. I would also like to acknowledge my teachers for their support and assistance during the course of this project.

Lastly, I would like to extend my sincere thanks to all my friends for their cooperation and support. I take complete responsibility for any errors or omissions in this report

DECLARATION

I, Dilpreet Kaur (PRN: 22070243017), student of M.Sc. Data Science and Spatial Analytics 2022-2024 certify that this project is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this project has not previously been submitted for assessment in any academic capacity, and that I have not copied in part or whole or otherwise plagiarised the work of other persons. I confirm that I have identified and declared all possible conflicts that I may have.

Dilpreet Kaur

PRN:22070243017

Date: 31 Jan 2023

TABLE OF CONTENTS

S.no.	Index	Page no.
1	ABSTRACT	5
2	INTRODUCTION	6
3	REPORT	7
4	MODEL DESIGN	16
5	MODEL WORKING	18
6	CONCLUSION	27
7	REFERENCES	27

ABSTRACT

This report presents the findings of a project that aimed to build a **Linear Regression** model to predict the prices of houses based on various independent variables such as the area of the house, number of bedrooms, number of bathrooms, and Per square feet. The study utilized a dataset of housing prices from a Delhi region and evaluated the accuracy of the model using mean squared error, root mean squared error, and r-squared value. We also used **Random Forest** and **Decision Tree** to check for the best accuracy of the models. Additionally, the report includes a detailed analysis of the relationships between the independent and dependent variables using various data visualization techniques such as scatter plots, box plots, and pair plots

The results of the study suggest that the **Random Forest Regressor** model can be used to make informed decisions regarding the pricing of houses in the region based on the available data as it obtained the highest accuracy. However, it is important to note that the model has its limitations and may not be accurate in predicting the prices of houses in different markets or with unique features. Where Linear Regression gave moderately good accuracy, the accuracy of Decision Tree was low. Overall, the study provides valuable insights into the relationship between housing prices and various independent variables and can serve as a useful starting point for further research in this area.

We also worked on the **POSTGRESQL** database to understand the working of python environments along with databases. **Psycopg2** library was used to draw a connection between the python and the database as it acts a driver between the both.

INTRODUCTION

This report presents the findings of a project that aimed to develop a linear regression model for predicting housing prices based on various independent variables. The study utilized a dataset of housing prices of Delhi region obtained from a PostgreSQL database that contained information on the area of the house, number of bedrooms, number of bathrooms, status, transaction and per square feet. The project involved using **psycopg2**, a popular Python library, to connect to the **PostgreSQL** database and extract the necessary data for analysis.

The goal of the project was to build a predictive model that could provide valuable insights into the relationship between the housing prices and the independent variables available in the dataset. The study utilized various data analysis techniques and machine learning algorithms to extract meaningful insights from the data and to develop a model that accurately predicts housing prices in the region.

However, we also looked into algorithms other than Linear Regression i.e., Random Forest and Decision Tree to get the best accuracy result. We compared the accuracy for each of the algorithm and obtained the best fit line to make our analysis. **Random Forest** had the **highest accuracy**.

The report provides an in-depth analysis of the dataset and the methods used to build the model, along with the results and conclusions drawn from the analysis. The report also includes data visualizations that provide a better understanding of the relationships between the independent and dependent variables.

Overall, the project provides valuable insights into the housing market and can be useful for policymakers, real estate agents, and homeowners who are interested in the factors that determine the prices of houses in a particular region.

REPORT

The purpose of this project is to predict the housing prices in Delhi region of India. The housing market is an important economic indicator and having accurate information on housing prices can be useful for buyers, sellers, real estate agents, and investors. In this project, we are using a housing price dataset to build a Linear Regression model that can predict the housing prices for different areas of Delhi. Next, we will use Random Forest and Decision Tree to compare the accuracy.

DATA COLLECTION

The data for this project was collected from the Kaggle website and is a housing price prediction dataset for Delhi, India. The dataset includes various features and attributes of houses in Delhi, such as the area, BHK, number of bathrooms, price, status of the house, parking etc.

The dataset includes 1259 observations and 11 variables for different localities, where 6 variables are numeric and the rest are categorical, which provides a comprehensive representation of the housing market in Delhi. By using the data from this dataset, we aim to gain a better understanding of the relationships between the features of a house and its price, and to build a linear regression model that can accurately predict the housing prices in Delhi.

Link: - https://www.kaggle.com/datasets/neelkamal692/delhi-house-price-prediction

DATA EXTRACTION

The data collected from Kaggle website was stored into the PostgreSQL database using **psycopg2 library** of python which helps to draw a connection between the python and PostgreSQL database. It was then extracted with the help of pscopg2 query and then stored into a dataframe for further processes.

```
## Creating table
cursor= connection.cursor()
cursor.execute("DROP TABLE IF EXISTS Housing_Price")
cursor.execute("create table Housing_Price(Area float, BHK float,Bathroom float,Furnishing text,Locality text, Parking float, P


## Entering data from csv
cursor.execute("COPY Housing_Price FROM 'C:/Users/Public/Documents/Housing prices Delhi.csv' DELIMITER ',' CSV HEADER;")

cursor.execute("SELECT Area, BHK, Bathroom, Furnishing,Locality, Parking, Price, Status, Transaction, Type, Per_Sqft  FROM Hous

# Fetch the data from the cursor into a list of tuples
db_data = cursor.fetchall()

# Create a Pandas DataFrame from the list of tuples
housing_price= pd.DataFrame(db_data, columns=["Area", "BHK", "Bathroom","Furnishing","Locality","Parking","Price","Status","Tra


connection.commit()
```

DATA EXPLORATION

The first step we did was exploring the data. We loaded the data which was stored in a data frame and stored it into another variable and explored various features.

```
In [3]: #load dataset for train
        data = housing_price
        data
```

Out[3]:

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Price	Status	Transaction	Type	Per_Sqft
0	800.0	3.0	2.0	Semi-Furnished	Rohini Sector 25	1.0	6500000.0	Ready_to_move	New_Property	Builder_Floor	NaN
1	750.0	2.0	2.0	Semi-Furnished	J R Designers Floors, Rohini Sector 24	1.0	5000000.0	Ready_to_move	New_Property	Apartment	6667.0
2	950.0	2.0	2.0	Furnished	Citizen Apartment, Rohini Sector 13	1.0	15500000.0	Ready_to_move	Resale	Apartment	6667.0
3	600.0	2.0	2.0	Semi-Furnished	Rohini Sector 24	1.0	4200000.0	Ready_to_move	Resale	Builder_Floor	6667.0

We looked through the top and last rows of the data for better understanding.

```
In [4]: data.tail()
```

Out[4]:

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Price	Status	Transaction	Type	Per_Sqft
1254	4118.0	4	5.0	Unfurnished	Chittaranjan Park	3.0	55000000	Ready_to_move	New_Property	Builder_Floor	12916.0
1255	1050.0	3	2.0	Semi-Furnished	Chittaranjan Park	3.0	12500000	Ready_to_move	Resale	Builder_Floor	12916.0
1256	875.0	3	3.0	Semi-Furnished	Chittaranjan Park	3.0	17500000	Ready_to_move	New_Property	Builder_Floor	12916.0
1257	990.0	2	2.0	Unfurnished	Chittaranjan Park Block A	1.0	11500000	Ready_to_move	Resale	Builder_Floor	12916.0
1258	11050.0	3	3.0	Unfurnished	Chittaranjan Park	1.0	18500000	Ready_to_move	New_Property	Builder_Floor	12916.0

Next, we listed the columns which were present in the data.

```
In [5]: data.columns
Out[5]: Index(['Area', 'BHK', 'Bathroom', 'Furnishing', 'Locality', 'Parking', 'Price',
               'Status', 'Transaction', 'Type', 'Per_Sqft'],
              dtype='object')
```

The data consisted of 1259 rows and 11 columns.

```
In [6]: data.shape
Out[6]: (1259, 11)
```

The basic information about the data can be seen that it consists of 3 data types i.e., float, integer and object, total number of entries, the null count in each column and the memory usage.

```
In [7]: data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Area         1259 non-null   float64
 1   BHK          1259 non-null   int64
 2   Bathroom     1257 non-null   float64
 3   Furnishing   1254 non-null   object
 4   Locality     1259 non-null   object
 5   Parking      1226 non-null   float64
 6   Price        1259 non-null   int64
 7   Status       1259 non-null   object
 8   Transaction  1259 non-null   object
 9   Type         1254 non-null   object
 10  Per_Sqft     1018 non-null   float64
dtypes: float64(4), int64(2), object(5)
memory usage: 108.3+ KB
```

Checking of the null values plays an important role in data exploration as the models are highly affected by null values. As we can see there are quite a few null values in 5 columns as presented.

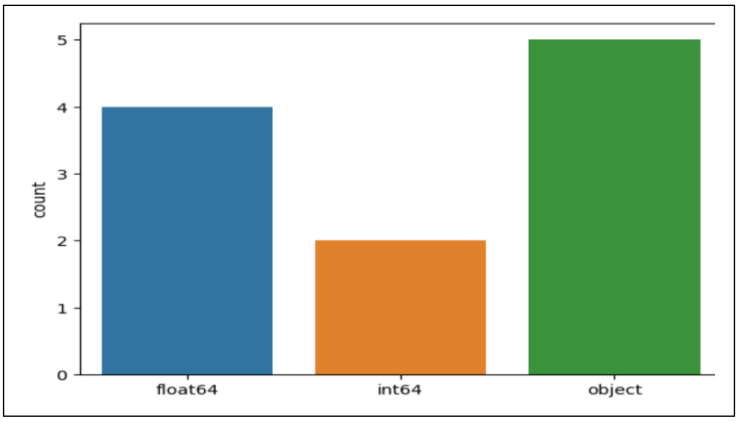
```
In [13]: ## Checking for null values

         data.isnull().sum()

Out[13]: Area            0
         BHK             0
         Bathroom        2
         Furnishing      5
         Locality        0
         Parking        33
         Price           0
         Status          0
         Transaction     0
         Type            5
         Per_Sqft      241
         dtype: int64
```

```
In [10]: data.dtypes

Out[10]: Area           float64
         BHK              int64
         Bathroom        float64
         Furnishing       object
         Locality         object
         Parking         float64
         Price            int64
         Status          object
         Transaction     object
         Type            object
         Per_Sqft        float64
         dtype: object
```

There are 3 data types present in the dataset which are displayed as follows.

They can be visualised as followed. We can see that we have majority of object data type and the float and then integer.

```
In [11]: sns.countplot(data.dtypes.map(str))
         plt.show()
```



Next, we did the values counts for the data set which gave us the frequency of each unique value in the data set in descending order.

```
In [12]: data.value_counts()

Out[12]: Area     BHK  Bathroom  Furnishing      Locality                                 Parking  Price     Status         Transaction
         Type           Per_Sqft
         750.0    2    2.0       Semi-Furnished  J R Designers Floors, Rohini Sector 24    1.0      5000000   Ready_to_move  New_Property
         Apartment      6667.0       4
         1000.0   3    2.0       Unfurnished     Virat Residency, Dwarka Mor              1.0      4620000   Ready_to_move  New_Property
         Apartment      4620.0       3
         540.0    2    2.0       Semi-Furnished  Adarsh Homes, Dwarka Mor                 1.0      3000000   Ready_to_move  New_Property
         Apartment      5000.0       3
         950.0    3    2.0       Furnished       Uttam Nagar Floors, Uttam Nagar          1.0      4370000   Ready_to_move  New_Property
         Apartment      4600.0       3
         1775.0   3    3.0       Semi-Furnished  The Amaryllis, Karol Bagh                1.0      25500000  Almost_ready   New_Property
         Apartment      14366.0      3
```

DATA PREPROCESSING

The data collected was subjected to several pre-processing steps to ensure its quality and suitability for analysis. The first step was to check for **missing** or incorrect **values** and handle them in an appropriate manner.

```
In [9]: data.isnull().sum()

Out[9]: Area              0
        BHK               0
        Bathroom          2
        Furnishing        5
        Locality          0
        Parking          33
        Price             0
        Status            0
        Transaction       0
        Type              5
        Per_Sqft        241
        dtype: int64
```

The missing values in the dataset for the **numeric** variable were filled with **the mean of the column** and the **categorical** variables were filled with **none values**.

```
In [13]: ## Replacing numeric null values with mean of the districuted data

         data.fillna(data.mean(),inplace=True)

         ## Replacing object type data with none

         data['Type'].fillna('None', inplace=True)
         data['Furnishing'].fillna('None', inplace=True)
```

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

Out[15]: Area            0
         BHK             0
         Bathroom        0
         Furnishing      0
         Locality        0
         Parking         0
         Price           0
         Status          0
         Transaction     0
         Type            0
         Per_Sqft        0
         dtype: int64
```

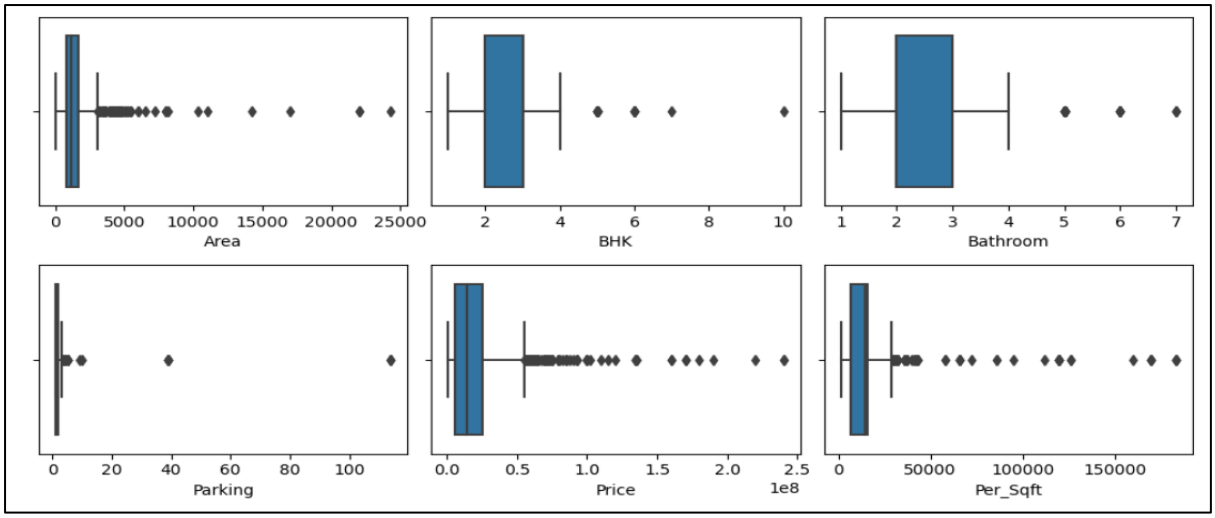
Though describing the data comes under data exploration, since our data consisted of null values which could have been affected the statistical description of our data hence first, we treated our missing values then described the **statistical summary** of the data as given below.

```
In [16]: data.describe()
```

Out[16]:

	Area	BHK	Bathroom	Parking	Price	Per_Sqft
count	1259.000000	1259.000000	1259.000000	1259.000000	1.259000e+03	1259.000000
mean	1466.452724	2.796664	2.556086	1.935563	2.130670e+07	15690.136542
std	1568.055040	0.954425	1.041391	6.196306	2.560115e+07	19002.775429
min	28.000000	1.000000	1.000000	1.000000	1.000000e+06	1259.000000
25%	800.000000	2.000000	2.000000	1.000000	5.700000e+06	6714.000000
50%	1200.000000	3.000000	2.000000	1.000000	1.420000e+07	14722.000000
75%	1700.000000	3.000000	3.000000	2.000000	2.550000e+07	15690.136542
max	24300.000000	10.000000	7.000000	114.000000	2.400000e+08	183333.000000

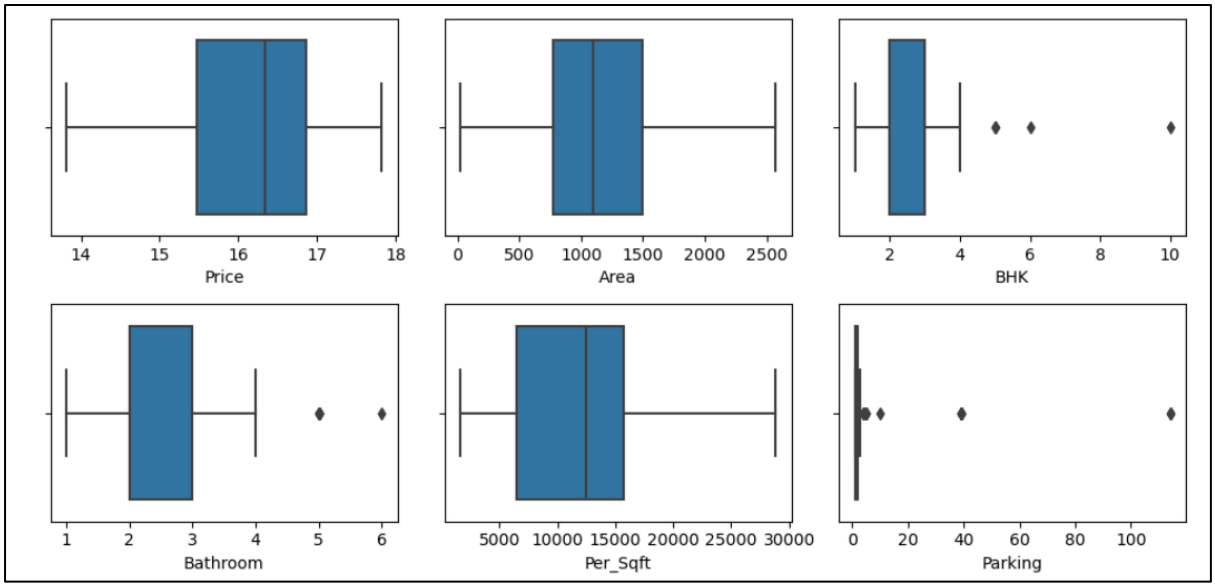
In addition to handling missing values, we also checked for **outliers** in the dataset. Outliers can have a significant impact on the results of the analysis, so it is important to identify and deal with them appropriately. In this project, outliers were detected with the help of boxplot from seaborn library. They were visualised with the help of a **sns.boxplot**.



As we can see attributes such as area, price and per_sqft have significant amounts of outliers hence using the **interquartile range** method they were transformed to bring them within the range of the other data points. Below displayed is for price, the rest were performed with the same method

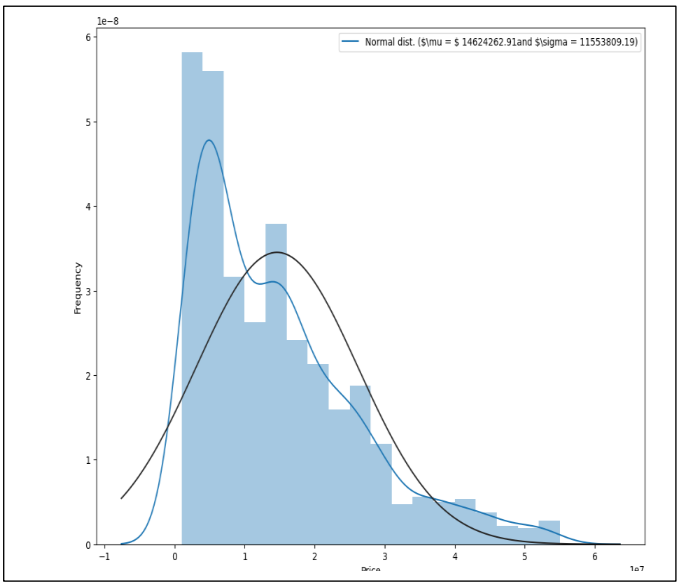
```
In [45]: # outlier treatment for price

         plt.boxplot(data.Price)
         Q1 = data.Price.quantile(0.25)
         Q3 = data.Price.quantile(0.75)
         IQR = Q3 - Q1
         data = data[(data.Price >= Q1 - 1.5*IQR) & (data.Price <= Q3+1.5*IQR)]
```

The final output was thus plotted with help of boxplot.

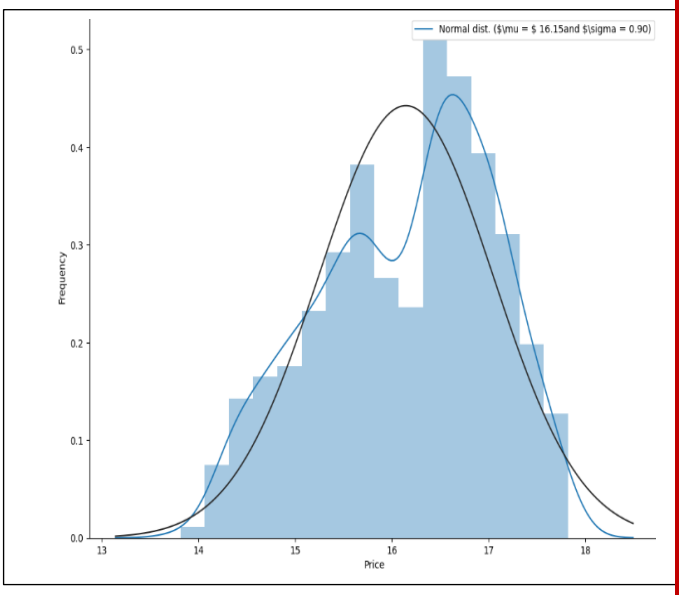


We created a **histogram visualization of the distribution** of the "Price" column in the data frame to check if the "Price" column is normally distributed. A normally distributed dataset is important for linear regression models as many of their underlying assumptions are based on a normal distribution. Thus, data was transformed using a log function of Numpy library.

The Seaborn library's sns.distplot function was used to create a histogram of the data, which is overlaid with a fitted normal distribution. The parameters of the normal distribution are calculated using the stats.norm.fit function.

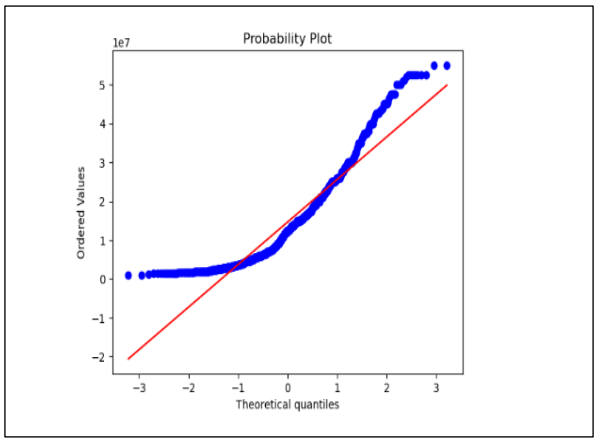


BEFORE TRANSFORMATION

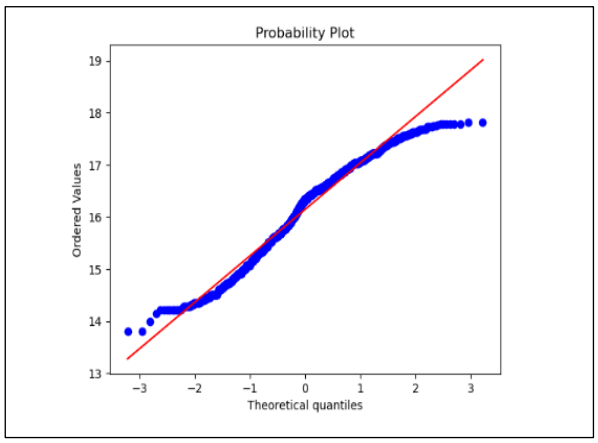


AFTER TRANSFORMATION

The **probability scatter plot** is also used with the help of stats.probplot function that shows the observed values of the data against their theoretical quantiles if they follow a normal distribution. A line is drawn to show the agreement between the observed values and the theoretical quantiles.



BEFORE TRANSFORMATION



AFTER TRANSFORMATION

Finally, we also converted the categorical variables into numerical variables, as linear regression models can only work with numerical data. This was achieved using technique **label encoding**, depending on the nature of the data. We separated the categorical columns and then transformed them using label encoder.

```
In [24]: ## Converting into categorical form for application of model

         data_cat = data.select_dtypes(include = 'object').columns   #seperating categorical variables

         data.drop(['Locality'], axis=1)


         ##converting to categoriccal form


         for c in data_cat:
             lbl = LabelEncoder()
             lbl.fit(list(data[c].values))
             data[c]=lbl.transform(list(data[c].values))
```

We can see the transformed data as follows:

```
]: data.head()
```

]:

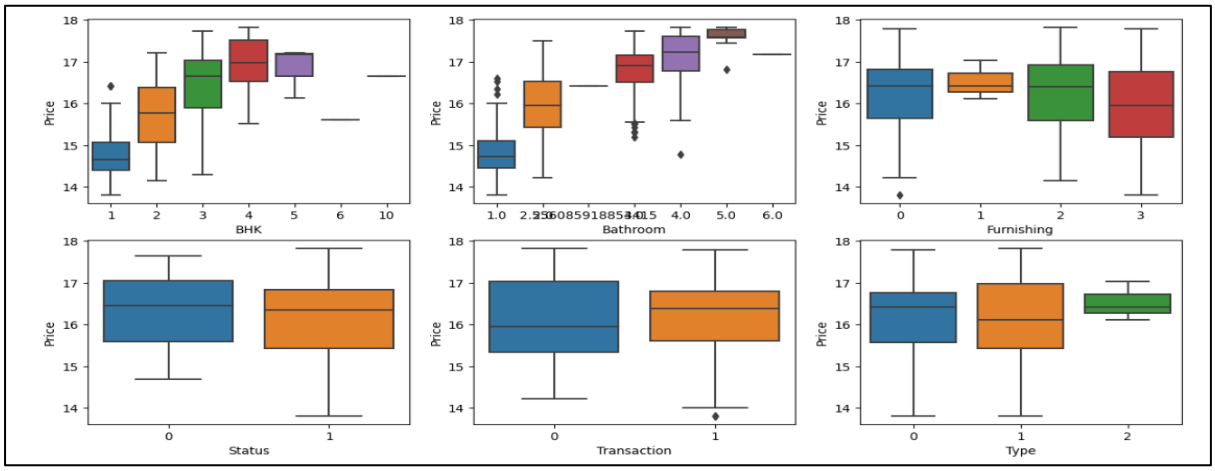
	Area	BHK	Bathroom	Furnishing	Locality	Parking	Price	Status	Transaction	Type	Per_Sqft
0	800.0	3	2.0	2	260	1.0	15.687313	1	0	1	15690.136542
1	750.0	2	2.0	2	126	1.0	15.424949	1	0	0	6667.000000
2	950.0	2	2.0	0	44	1.0	16.556351	1	1	0	6667.000000
3	600.0	2	2.0	2	258	1.0	15.250595	1	1	1	6667.000000
4	650.0	2	2.0	2	259	1.0	15.640060	1	0	1	6667.000000

By following these pre-processing steps, we were able to ensure that the data was of high quality and ready for analysis. This is an important step in the data analysis process and helps to ensure that the results of the analysis are accurate and reliable.

DATA ANALYSIS

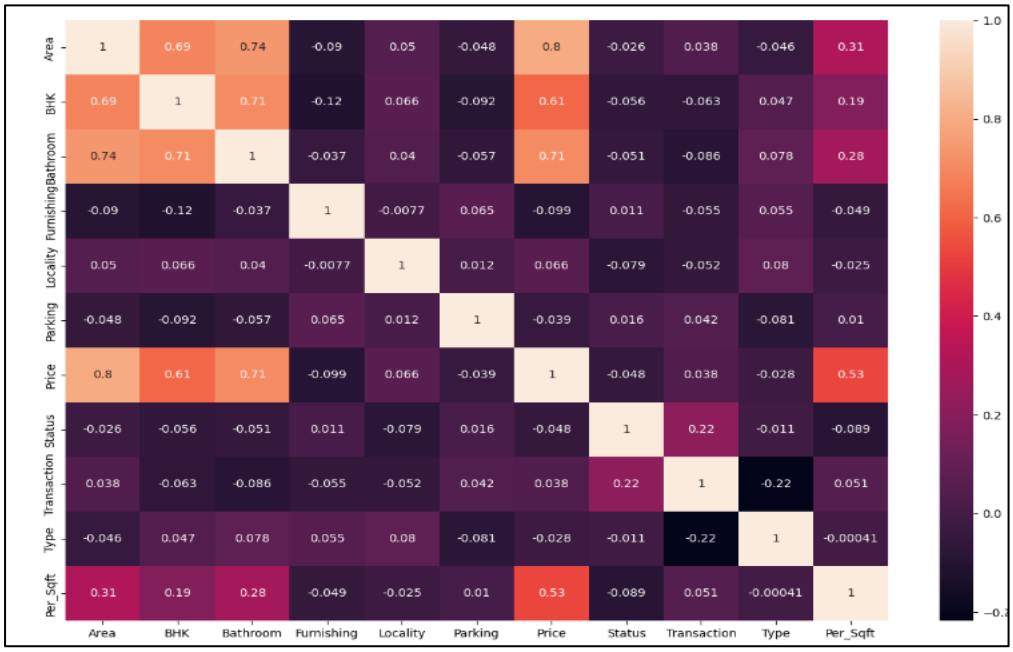
It plays a very important role as we are dependent on this part to gain the useful insights from out data. Below described is the methodology used for the data analysis.

Using boxplot from seaborn library comparison of each attribute with price was made. We can see how the house price is affected by the changes in the features of the dataset.



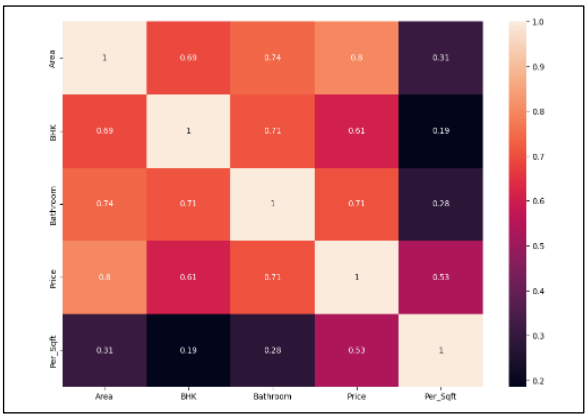
2 Bhk house prices are more as compared to the other. Similarly, we can see how the prices are increasing with the number of bathrooms. Furnished houses along with ready to move availability have high prices. New property along with builder-floor type are costlier as compared to others.

Correlation between the features is necessary to depict the strength of the linear relation between pair of variables. Hence, we plotted a correlation map between the variables with the help of a heat map.



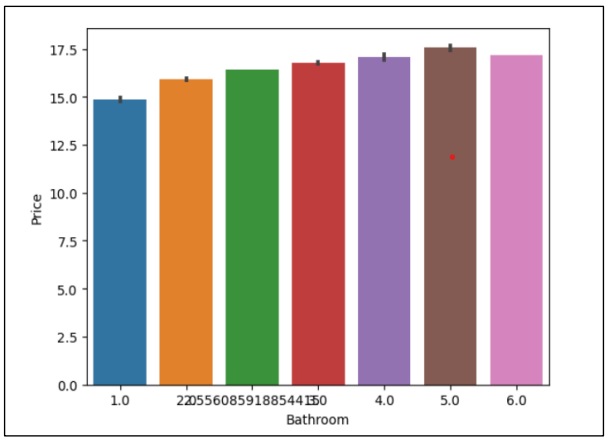
We can see that all the features are not very well related

Thus, we plot a heatmap for the top related features with the price which highly affect our price parameter.

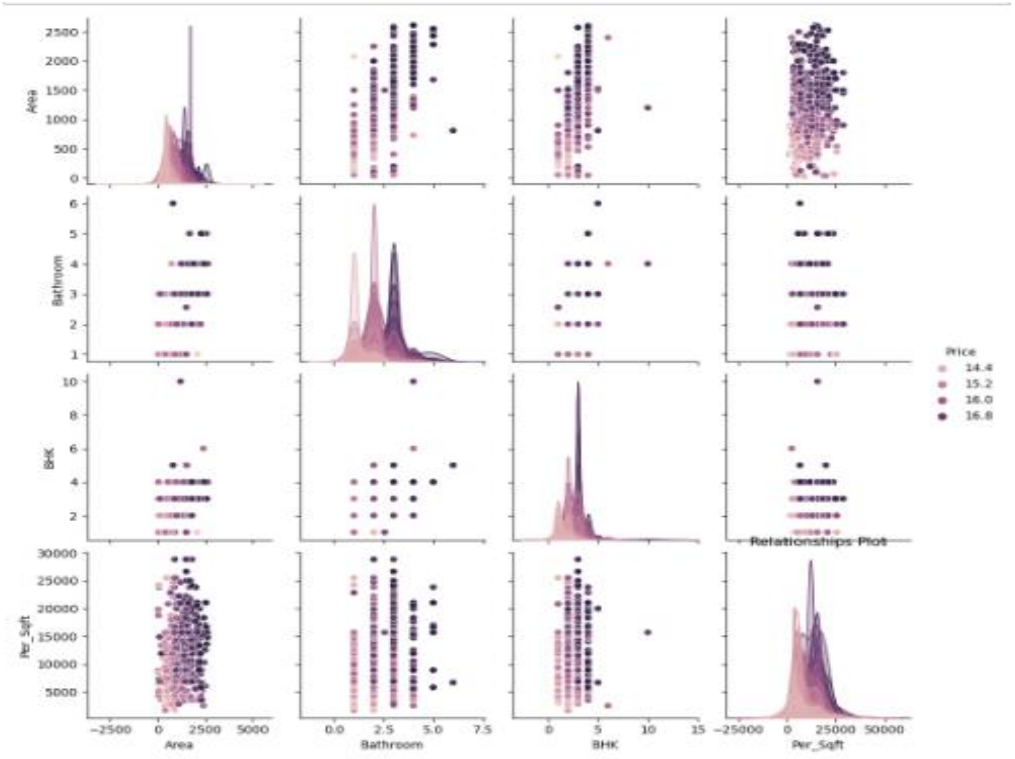


We can see a significant relation between price and number of bathrooms. Then we tried to analyse and visualize the most related feature with the target variable with the help of a bar plot

```
In [30]: print('Find the most important features relative to the traget')
         corr = data.corr()
         corr.sort_values(['Price'], ascending = False , inplace = True)
         corr.Price

         Find the most important features relative to the traget

Out[30]: Price          1.000000
         Area           0.798598
         Bathroom       0.705724
         BHK            0.610344
         Per_Sqft       0.532196
         Locality       0.066220
         Transaction    0.038361
         Type          -0.027555
         Parking       -0.039305
         Status        -0.048348
         Furnishing    -0.098926
         Name: Price, dtype: float64
```



Lastly, we created a pair plot that shows the scatter plots of the target variable (Price) with the top independent variables (Bathroom, Area, BHK, Transaction, Per_sqft), as well as the scatter plots of the independent variables with each other. The points are coloured by the value of the target variable, so we can see how the values of the target variable are distributed across the different values of the independent variables.



MACHINE LEARNING MODEL DESIGN

LINEAR REGRESSION

Linear regression is a type of statistical analysis that allows you to model the relationship between a dependent variable (often denoted as "Y") and one or more independent variables (often denoted as "X"). The goal of linear regression is to find the best fit line that explains the relationship between the variables.

Assuming we have a set of data points (x1, y1), (x2, y2), ..., (xn, yn), where x represents the independent variable and y represents the dependent variable, we want to find the line of best fit that represents the relationship between x and y.

1. Collect your data: Gather a set of data that includes two variables, X (the independent variable) and Y (the dependent variable).
2. Prepare your data: Clean and structure your data.
3. Choose your model: Decide on a linear equation that represents the relationship between X and Y. The equation will have the form: Y = b0 + b1*X, where b0 is the intercept and b1 is the slope of the line.
4. Fit your model: The goal is to find the values of b0 and b1 that minimize the sum of the squared errors between the predicted values and the actual values. The squared error for a single data point (xi, yi) is given by:
   (yi - (b0 + b1*xi))^2
5. The total squared error for all data points is given by:
   E = Σ(yi - (b0 + b1*xi))^2
6. To find the values of b0 and b1 that minimize E, we take the partial derivatives of E with respect to b0 and b1, set them equal to zero, and solve for b0 and b1. This gives us the following equations:
   b1 = Σ(xi - x̄)(yi - ȳ) / Σ(xi - x̄)^2
   b0 = ȳ - b1*x̄
   where x̄ and ȳ are the mean values of x and y, respectively.
7. Test your model: Evaluate your model's performance by using it to make predictions on new data and comparing the predicted values to the actual values.
8. Use your model: Once you're satisfied with your model's performance, use it to make predictions and draw insights about the relationship between X and Y.

DECISION TREE

A decision tree model is a type of supervised machine learning model used for both classification and regression tasks. It works by creating a tree-like model of decisions and their possible consequences or outcomes. The tree is constructed by splitting the data into smaller and smaller subsets based on the most significant differentiating factors or features, and each split forms a new decision node in the tree. The final nodes of the tree are the leaves, which represent the output or prediction of the model.

The decision tree algorithm is the process of constructing the decision tree model. The algorithm recursively partitions the data into subsets based on the values of the input features until it arrives at the best possible split, which maximizes the information gain or reduces the entropy. The information gain is a measure of the reduction in uncertainty or randomness achieved by the split.

The algorithm can use different splitting criteria, such as Gini index, entropy, or information gain ratio. They are easy to interpret and visualize, can handle both numerical and categorical data, and can capture non-linear relationships between the input features and the output variable. However, decision trees can be prone to overfitting, where the model is too complex and fits the training data too closely, leading to poor generalization performance on new data. Therefore, it is important to tune the parameters of the decision tree model and use techniques such as pruning, ensembling, or regularization to prevent overfitting.

The working algorithm of a decision tree can be summarized in the following steps:

(1) The algorithm starts with the entire dataset and calculates the impurity or entropy of the target variable. Impurity is a measure of the amount of uncertainty or randomness in the data. For classification tasks, impurity can be measured using metrics such as Gini index, entropy, or classification error. For regression tasks, impurity can be measured using metrics such as mean squared error or mean absolute error.
(2) The algorithm then evaluates each feature in the dataset and calculates the impurity or information gain that would result from splitting the data based on that feature. Information gain is the difference between the impurity of the parent node and the weighted average of the impurity of the child nodes. The feature with the highest information gain is selected as the best feature for splitting.
(3) The algorithm creates a decision node based on the selected feature and splits the data into subsets based on the possible values of that feature. For categorical features, each value corresponds to a separate branch in the decision tree. For continuous features, the algorithm can use binary splits based on a threshold value.
(4) Steps 1-3 are then repeated recursively for each subset of the data, starting with the subsets that resulted from the first split. The algorithm stops when a stopping criterion is met, such as reaching a maximum depth, reaching a minimum number of samples per leaf node, or when all the samples in a subset belong to the same class or have the same target value.
(5) Finally, the algorithm assigns a class or value to each leaf node based on the majority class or the mean value of the samples in that node. The decision tree model can then be used to make predictions on new data by traversing the tree from the root node to the appropriate leaf node based on the values of the input features.

The resulting decision tree can be visualized as a tree structure, where each internal node represents a decision based on a feature, and each leaf node represents a final decision or prediction. The tree can be used to interpret the relationships between the input features and the target variable and can be used to identify the most important features for the task.

RANDOM FOREST

A Random Forest model is an ensemble learning method that combines multiple decision trees to create a more accurate and robust model for classification or regression tasks. Random Forest is a powerful machine learning algorithm that can be used for various tasks such as classification, regression, and feature selection.

In a Random Forest model, the algorithm randomly selects a subset of features and data points from the training set, and uses this subset to build multiple decision trees. The decision trees are constructed using standard decision tree algorithms such as CART or ID3. Each decision tree is built independently and votes on the final prediction.

When making a prediction for a new data point, each decision tree in the forest independently predicts the outcome, and the forest's final prediction is the majority vote (for classification) or the average (for regression) of the individual decision trees' predictions.

Random Forest models are known for their robustness, accuracy, and generalization performance, and they are less prone to overfitting than a single decision tree. They can also handle missing values and outliers in the data, and are capable of handling large datasets with a large number of features.

The algorithm works as follows:

(1) The algorithm starts by selecting a random subset of the features from the dataset.
(2) A decision tree is built on this subset of the features. The decision tree is built using the standard decision tree algorithm, such as CART (Classification and Regression Trees) or ID3 (Iterative Dichotomiser 3).
(3) Steps 1 and 2 are repeated multiple times, each time selecting a different random subset of the features and building a decision tree on that subset.
(4) Each decision tree is used to make predictions on new data, and the final prediction is the average (for regression problems) or majority vote (for classification problems) of the predictions of all the decision trees.
(5) The performance of the random forest model can be evaluated using metrics such as accuracy, precision, recall, F1-score, or mean squared error (MSE).
(6) The algorithm can be further optimized by tuning hyperparameters such as the number of decision trees, the maximum depth of each tree, the minimum number of samples required to split a node, and the size of the feature subset.

MODEL PRE-PROCESSING

We are creating two variables: x and y, which represent the independent variables and the dependent variable (Price) respectively. The x variable contains the top features from the data set and 'Price' column, is the dependent variable that we want to predict which is stored in y variable.

Also, we checked the shape for our variables as it is necessary for them to be of the same for the model to process correctly. Since the y variable had different dimension so we reshaped it with help of numpy library.

```
In [38]: ## Assigning varibles to the data set for the model
         x=data[['Area','BHK','Bathroom','Per_Sqft']]
         y=np.array(data['Price']).reshape(-1,1)

In [39]: x.shape,y.shape
Out[39]: ((1036, 4), (1036, 1))
```

Next, we split the data into training and test dataset with the help pf train_test_split.

```
#splitting the data in train and test

x_train, x_test, y_train, y_test = train_test_split(x,y, train_size=0.7, random_state=42)
```

We used 70% of our data for model training and rest 30% for testing our models working and accuracy. The random number is generator used to shuffle the data or perform other operations. The random seed is used to ensure that the randomness is reproducible, so that running the same code multiple times with the same random seed will produce the same results.

(1) LINEAR REGRESSION

Linear regression is a commonly used machine learning technique for predicting a continuous dependent variable in our case which is the price, based on one or more independent variables which are area, bathroom, bhk and per square feet.

We created an instance of the LinearRegression class from the scikit-learn library. This class implements the linear regression algorithm for fitting a linear model to a set of data.

The lr_reg variable now holds a linear regression model object that can be used for training and prediction. In order to fit the model to the data, we called the fit method of the model object, passing in the x_train and y_train variables as arguments.

```
In [42]: lr_reg=LinearRegression()
         lr_reg.fit(x_train,y_train)
         lr_y_pred=lr_reg.predict(x_test)
```

The lr_reg.fit(x_train, y_train) was used to fit the linear regression model to the training data. The fit method is called on the model object and takes two arguments: x_train and y_train.

x_train is the training data for the independent variables (features), and y_train is the training data for the dependent variable (Price). The fit method trains the linear regression model by learning the relationship between the independent variables in x_train and the dependent variable in y_train.

After fitting the model, it can be used to make predictions on new data by calling the predict method. It is important to note that the performance of the model will only be as good as the quality of the training data, and that the model may not perform well on unseen data if it has overfit to the training data. To prevent overfitting, it is common to split the data into a training set and a validation set, and to evaluate the performance of the model on the validation set to ensure it generalizes well to new data.

Next, we interpreted the intercept and the coefficient of the model as they play important role in result and predictions. Though intercept value may not have any practical meaning, as it represents the predicted value of the target variable when all feature values are zero. So we concentrate on the coefficients of the model.

The values of lr_reg.coef_ in a linear regression model represent the change in the target variable for a one-unit change in each corresponding feature, holding all other features constant. It is 1x4 array (or a row vector), with each element representing the coefficient for each feature and their impact on the target variable.

```
In [43]: print(lr_reg.intercept_,lr_reg.coef_)

         [13.92771881] [[8.80388974e-04 5.81813327e-02 2.23043380e-01 4.57437067e-05]]
```

Next, we calculate three metrics for evaluating the model's performance: mean squared error (MSE), root mean squared error (RMSE), and R-squared.

The mean squared error (MSE) is a measure of the average squared difference between the predicted values and the true values. The function mean_squared_error from the sklearn.metrics module is used to calculate the MSE between the true target values y_test and the predicted target values y_pred.

The root mean squared error (RMSE) is the square root of the MSE, and gives the error in the same units as the target variable. The sqrt function from the math module is used to calculate the square root of the MSE.

R-squared is a measure of how well the model fits the data, with a value of 1 indicating a perfect fit and a value of 0 indicating that the model does not explain any of the variability in the data. The function r2_score from the sklearn.metrics module is used to calculate the R-squared between the true target values y_test and the predicted target values y_pred.

Finally, the values of the three metrics are printed to the console to allow for evaluation of the model's performance.

```
In [44]: ## Evaluating the performance


mse=mean_squared_error(y_test,lr_y_pred)
rmse=sqrt(mse)
r_squared = r2_score(y_test, lr_y_pred)

print('Mean_Squared_Error:',mse)
print('Root_Mean_Sqaured_Erro:',rmse)
print('r_sqaure_value:',r_squared)
```

```
Mean_Squared_Error: 0.18687048489722932
Root_Mean_Sqaured_Erro: 0.4322851893105168
r_sqaure_value: 0.765324675301181
```

In the output we get, indicates that the model has a relatively low error rate and is a good fit for the data.

The R-squared value, which is 0.7653 in our case, represents the proportion of the variance in the target variable that is explained by the independent variables in the model. In our case, the R-squared value of 0.768 indicates that the model explains a moderate amount of the variance in the target variable.

Next, we calculate and print the accuracy of the linear regression model on both the training set and the test set.

The lr_reg.score method is used to calculate the accuracy of the model, which is defined as the coefficient of determination (R-squared) for the model.

The lr_reg.score method takes two arguments: the independent variables (features) and the dependent variable (target). In the first line of code, x_train and y_train are passed as arguments to calculate the accuracy of the model on the training set, and in the second line of code, x_test and y_test are passed as arguments to calculate the accuracy of the model on the test set.

The accuracy of the model on the training set is expressed as a percentage by multiplying the result by 100, and the accuracy of the model on the test set is similarly expressed as a percentage. The results are then printed to the console using string formatting.

```
In [45]: print(f"Train Accuracy : {lr_reg.score(x_train, y_train)*100}")
         print(f"Test Accuracy : {lr_reg.score(x_test, y_test)*100}")

         Train Accuracy : 72.0180977718929
         Test Accuracy : 76.5324675301181
```

And then we calculate the accuracy of the linear regression model on the test set and displays it as an HTML formatted string.

The lr_reg.score method is passed the test set independent variables (x_test) and dependent variable (y_test) as arguments. The result is multiplied by 100 to express it as a percentage and rounded to 2 decimal places using the round function.

Finally, the formatted string is displayed using the HTML function from the IPython.core.display module. This allows the result to be displayed as HTML within a Jupyter Notebook, making it more visually appealing.

```
In [46]: from IPython.core.display import display, HTML
         lr_accuracy=round((lr_reg.score(x_test,y_test)*100),2)
         HTML (f'Linear Regression Model Accuracy : <b>{lr_accuracy}%</b>')

Out[46]: Linear Regression Model Accuracy : 76.53%
```

Lastly, we used our model to predict the price of house if a new dataset is given.

Before working with new dataset we used the original dataset to understand the variations in the actual and predicted value and to evaluate the accuracy.

Original dataset for prediction: -

```
In [47]: data.head()

Out[47]:
```

	Area	BHK	Bathroom	Furnishing	Parking	Price	Status	Transaction	Type	Per_Sqft
0	800.0	3.0	2.0	2	1.0	15.687313	1	0	1	15690.136542

We can see that the predicted price is almost in the range of the actual price hence the results suggest that the linear regression model has a reasonable accuracy for predicting the target variable based on the independent variables in the dataset.

```
In [49]: ## Predicting Price for a old Data to check for accuracy

         old_data = [[800,3,2,15690.136542]]
         old_data = np.array(old_data)
         predicted_price = lr_reg.predict(old_data)
         print(f"The predicted price for the property is: {predicted_price} ")

         The predicted price for the property is: [[15.97038575]]
```

Now we will predict the house price on a new dataset.

```
In [50]: ## Prediction Price for New Data

         new_data = [[900,3,3,11700]]
         new_data = np.array(new_data)
         predicted_price = lr_reg.predict(new_data)
         print(f"The predicted price for the new property is: {predicted_price}")

         The predicted price for the new property is: [[16.09894439]]
```

(2) DECISION TREE ALGORITHM

Decision trees are a popular and powerful tool for predicting outcomes based on a set of input variables. In predicting housing prices, where a decision tree can be trained on a dataset of housing prices and corresponding features, such as number of bedrooms, square footage, location, bathroom, and other characteristics.

We performed the same steps in the decision tree algorithm as we did for linear regression. We created an instance for the Decision Tree Regressor. Next, we trained the dependent and independent features with the help of fit method. Then, we predicted the target variable I.e., x_test with help of predict method.

```
In [51]: from sklearn.tree import DecisionTreeRegressor

         dt_reg = DecisionTreeRegressor()
         dt_reg.fit(x_train, y_train)
         dt_y_pred = dt_reg.predict(x_test)
```

Next, we evaluated the metrics for the predicted values obtained from the Decision Tree Regressor model. As we can see from the output that the model has performed moderately well as low values of MSE and RMSE can be seen. While the R-squared value of 0.7448873186411435 indicates that the model explains about 74.49% of the variability in the target variable.

```
In [52]: mse=mean_squared_error(y_test,dt_y_pred)
         rmse=sqrt(mse)
         r_squared = r2_score(y_test,dt_y_pred)

         print('Mean_Squared_Error:',mse)
         print('Root_Mean_Sqaured_Erro:',rmse)
         print('r_sqaure_value:',r_squared)

         Mean_Squared_Error: 0.20314462345005868
         Root_Mean_Sqaured_Erro: 0.4507156791704263
         r_sqaure_value: 0.7448873186411435
```

Next, we calculated the accuracy of train set and test set.

```
In [53]: print(f"Train Accuracy : {dt_reg.score(x_train, y_train)*100}")
         print(f"Test Accuracy : {dt_reg.score(x_test, y_test)*100}")

         Train Accuracy : 98.07567506467674
         Test Accuracy : 74.48873186411436
```

And lastly, we printed the accuracy of model with the help of test set to understand how the model is performing.

```
In [54]: from IPython.core.display import display, HTML
         dt_accuracy=round((dt_reg.score(x_test,y_test)*100),2)
         HTML (f'Decision Tree Model Accuracy : <b>{dt_accuracy}%</b>')

Out[54]: Decision Tree Model Accuracy : 74.49%
```

(3) RANDOM FOREST REGRESSOR

Random Forest is a powerful machine learning algorithm that can be used for predicting housing prices. In fact, it is often used as an alternative to decision trees because it has several advantages, such as better accuracy, reduced overfitting, and more robustness to noisy and unbalanced datasets.

The train-test splitted data was passed into Random Forest Regressor to check the accuracy of the algorithm.

We created instance for the Random Forest Regressor and stored it in rf_reg variable. We then passed our train set to the fit method for the model to train the model on the basis of the relationship between x and y.

```
In [55]: from sklearn.ensemble import RandomForestRegressor

         rf_reg = RandomForestRegressor()
         rf_reg.fit(x_train, y_train)
         rf_y_pred = rf_reg.predict(x_test)
```

Next, we calculated the performance of the model.

```
In [56]: mse=mean_squared_error(y_test,rf_y_pred)
         rmse=sqrt(mse)
         r_squared = r2_score(y_test,rf_y_pred)

         print('Mean_Squared_Error:',mse)
         print('Root_Mean_Sqaured_Erro:',rmse)
         print('r_sqaure_value:',r_squared)

         Mean_Squared_Error: 0.1515998031970481
         Root_Mean_Sqaured_Erro: 0.38935819395133847
         r_sqaure_value: 0.8096182334031509
```

We printed the train and test accuracy with the help of score method.

```
In [57]: print(f"Train Accuracy : {rf_reg.score(x_train, y_train)*100}")
         print(f"Test Accuracy : {rf_reg.score(x_test, y_test)*100}")

         Train Accuracy : 95.58753087675817
         Test Accuracy : 80.96182334031509
```

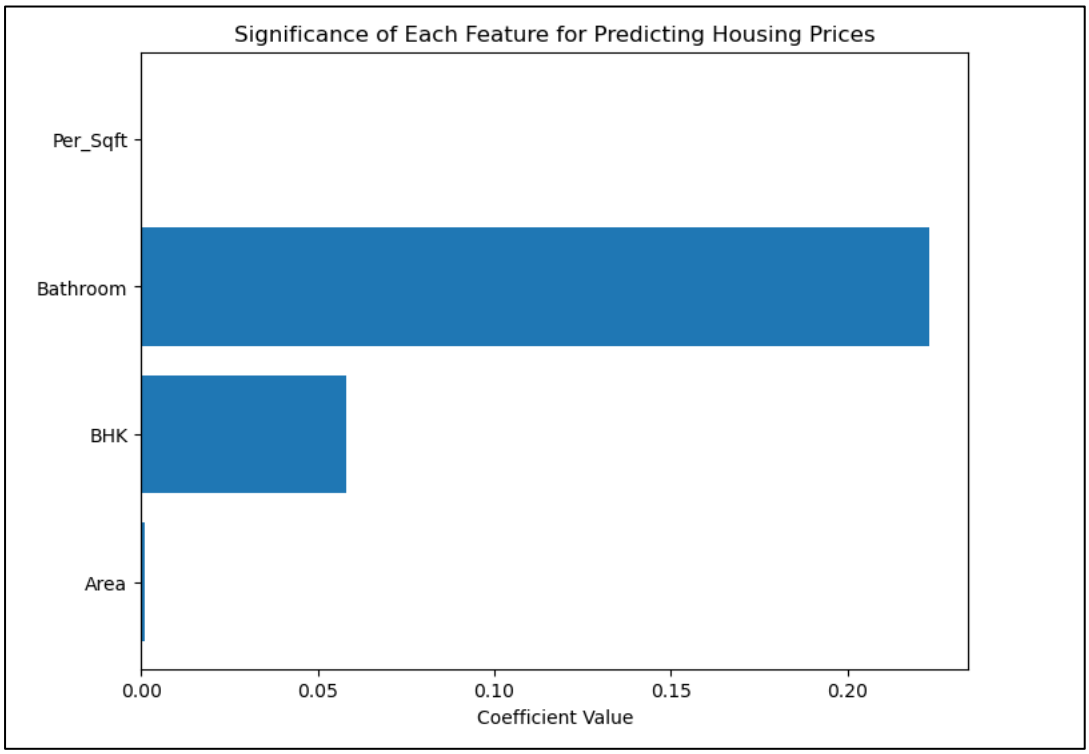
Finally, the accuracy was displayed.

```
In [58]: from IPython.core.display import display, HTML
         rf_accuracy=round((rf_reg.score(x_test,y_test)*100),2)
         HTML (f'Random Forest Model Accuracy : <b>{rf_accuracy}%</b>')
```

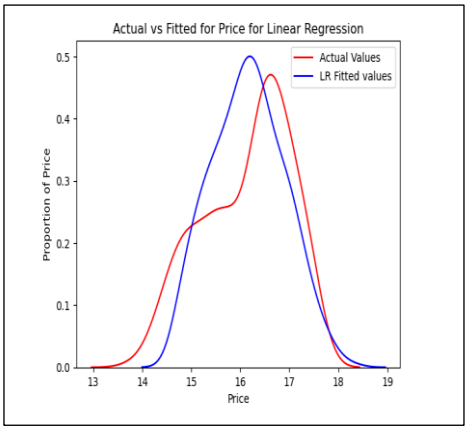
Out[58]: Random Forest Model Accuracy : **80.96%**

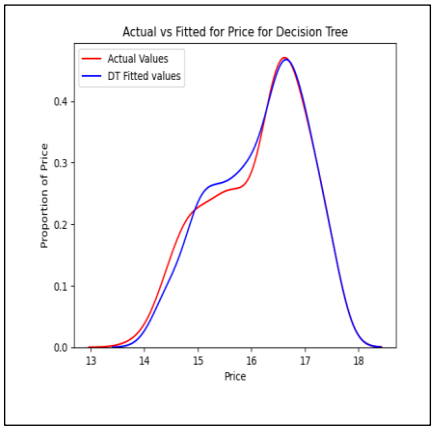
DATA VISUALISATION

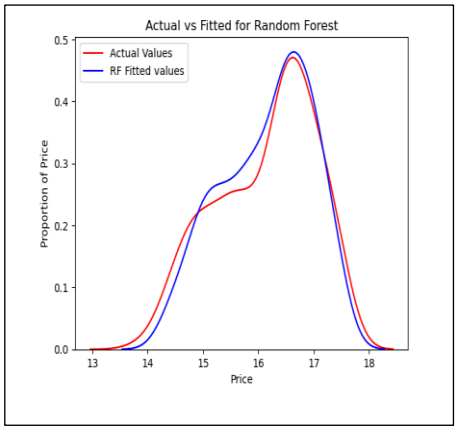
We tried to understand the significance of each attribute on the price on the basis of their coefficient. Number of bathrooms have the most significance in the prediction of houses.



We displayed a distribution plot to understand the variation between the actual and the predicted price for the algorithms. Given the accuracy of our model as 76% we can see some variations in the visualisation for Linear Regression model. While there is least amount of variation in Random Forest Regressor and Decision Tree Regressor. The red line displays the actual values where as the blue line displays the predicted values.

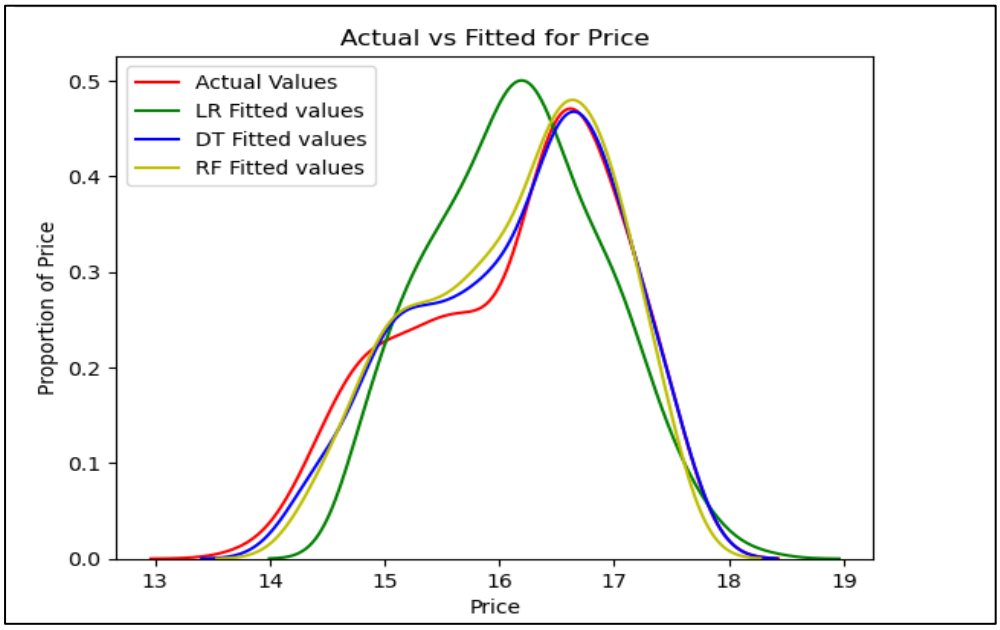




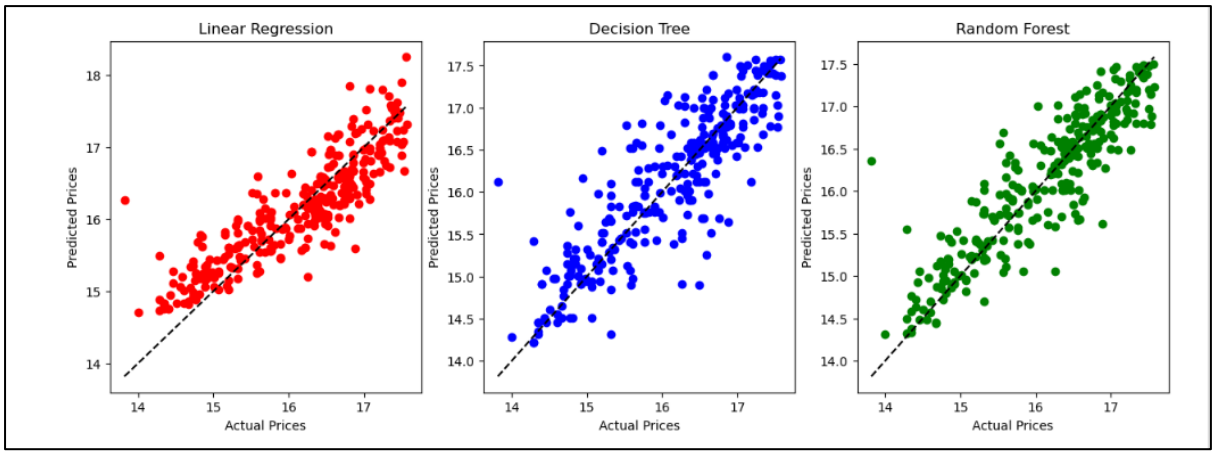


There may be other factors that are playing a role in determining the price of a property, and the model was not capture all of these relationships. Further analysis and refinement of the model may be necessary to improve its accuracy.

Next, we tried to compare the predicted values of each algorithm with the actual value with help of distribution plot. Here we can clearly understand how each model is performing in comparison to the other and what are the variations between them.



Next, we plotted a scatter plot that compared the predicted and actual prices of the test data, where the diagonal line represented perfect correlation between the predicted and actual values. With the help of this plot, we can visually inspect how well the model fits the data, and also, we can understand patterns and observe a few outliers.

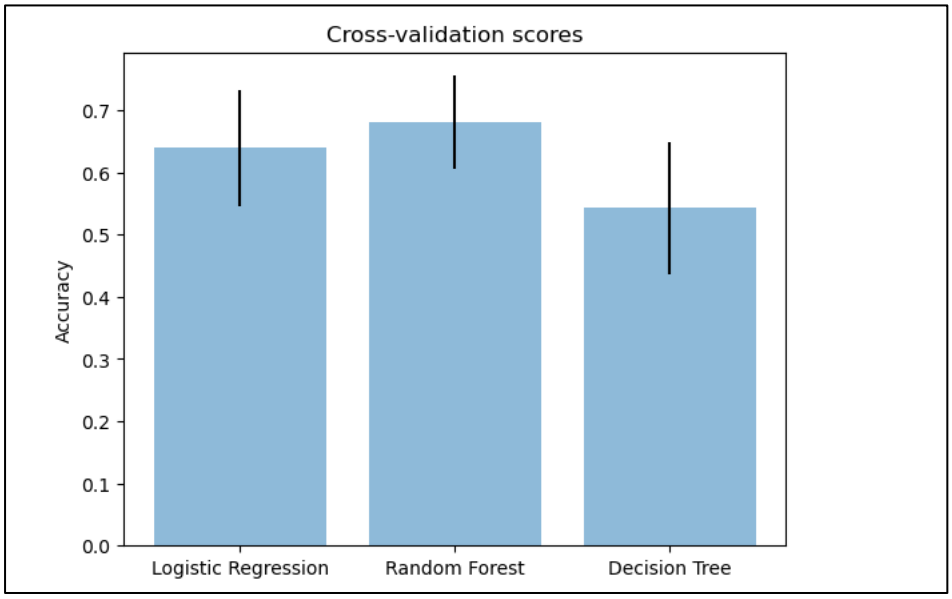


RESULT

We performed cross-validation on three different regression models, namely Logistic Regression, Random Forest, and Decision Tree, and printing the mean and standard deviation of their accuracy scores to compare the performance of each.

```
In [64]: from sklearn.model_selection import cross_val_score
         # Perform cross-validation
         lr_scores = cross_val_score(lr_reg, x, y, cv=5)
         rf_scores = cross_val_score(rf_reg, x, y, cv=5)
         dt_scores = cross_val_score(dt_reg, x, y, cv=5)

         # Print the mean and standard deviation of the scores
         print("Logistic Regression Accuracy: %0.2f (+/- %0.2f)" % (lr_scores.mean(), lr_scores.std() * 2))
         print("Random Forest Accuracy: %0.2f (+/- %0.2f)" % (rf_scores.mean(), rf_scores.std() * 2))
         print("Decision Tree Accuracy: %0.2f (+/- %0.2f)" % (dt_scores.mean(), dt_scores.std() * 2))

         Logistic Regression Accuracy: 0.64 (+/- 0.19)
         Random Forest Accuracy: 0.68 (+/- 0.15)
         Decision Tree Accuracy: 0.54 (+/- 0.21)
```

Comparing the accuracy scores of different models can help us choose the best model for our specific task. We can see that the Random Forest model has the highest accuracy score, followed by Logistic Regression, and then Decision Tree. This suggests that the Random Forest model is likely the best model for the given dataset.



Also, when comparing the accuracy for all the models we can see the results. he Random Forest model has the highest accuracy score (80.96%), followed by the Linear Regression model (76.53%), and then the Decision Tree model (74.49%). Therefore, the Random Forest model may be the best choice for this particular dataset and problem.

```
In [72]: ## Accuracy of all Models

         print("Linear Regression Accuracy : ", lr_accuracy)
         print("Decision Tree Accuracy :", dt_accuracy)
         print("Random Forest Accuracy :",rf_accuracy)

         Linear Regression Accuracy :  76.53
         Decision Tree Accuracy : 74.49
         Random Forest Accuracy : 80.96
```

CONCLUSION

In the conclusion of the report for this model, we can state that a linear regression model, Decision Tree Regressor and Random Forest Regressor was built to predict the prices of houses based on various independent variables such as the area of the house, BHK, number of bathrooms, area and per square feet etc. The model was evaluated using mean squared error, root mean squared error, and r-squared value, which indicated that the model had an accuracy of each model. Random Forest regressor had the best accuracy, as it reduces the variance of the model by using an averaging effect of multiple trees, which helps to minimize overfitting. Random Forest is an ensemble learning method that combines multiple decision trees to create a more robust and accurate model. Each tree in the forest is built independently, and the final prediction is made by aggregating the predictions of all the individual trees.

Further analysis of the data using visualizations such as scatter plots, distribution plots, and pair plots provided insights into the relationships between the independent and dependent variables. The results of the analysis can be used to make informed decisions regarding the pricing of houses in different locations of Delhi region and with varying features.

Overall, the model can be considered as a good starting point for predicting house prices based on the available data. However, it is important to note that the model has its limitations and may not be accurate in predicting the prices of houses in different markets or with unique features.

REFERENCES

- https://www.kaggle.com/datasets/neelkamal692/delhi-house-price-prediction
- https://towardsdatascience.com/create-a-model-to-predict-house-prices-using-python-d34fe8fad88f