



House Price Prediction on different Metropolitan cities of India

Shashank Satyarthi

09/12/2022

School of Mathematics,
Cardiff University

A dissertation submitted in partial fulfilment of the requirements for **MSc Data Science and Analytics**
by taught programme, supervised by **Prof. Yipeng Qin**

Acknowledgements

I would like to thank my supervisor, Professor Yipeng Qin, for his timely help and for pointing me towards the right direction throughout this project. I am grateful to Ms Joanna Emery for providing me with an opportunity to pursue this project. Also, I would like to acknowledge the advice and moral support provided by both of my professor.

Executive Summary:

Businesses want to communicate with their clients in a way that is not limited by time, the availability of staff, or language. They must anticipate their needs and satisfaction in order to keep their consumers satisfied and hence reduce client churn. Our daily lives depend on houses in many ways. When someone is thinking about buying a home, there are several factors that could influence their interest. When choosing a home, there are several factors to consider, including cost, neighbourhood, location, number of bedrooms, maintenance needs, safety precautions, etc. It was used in this study to apply various data regression models to the dataset used to evaluate housing prices. In this project, I've used a variety of regression methods to forecast the circumstances in which the customer will buy the house, which is also predicted by all the different regressions. Here, it is clear that supervised algorithms like K-Nearest Neighbours, Random Forest, Decision Tree, and XG Boost perform better. However, Decision Tree Regressor performed somewhat better than Gradient Boosting methods, whereas Linear Regression techniques and Gradient Boosting algorithms only offered poor results. My research indicates that the best kind of algorithm for the housing evaluation dataset is Random Forest and XG Boost Regressor.

Table of Contents

Acknowledgement

Executive Summary

1. Introduction:
 - 1.1.Goals of the Study:
 - 1.2. Paper Organisation:
2. Literature review/ Related work:
3. Descriptive analysis of the dataset:
 - 3.1. Missing Values:
 - 3.2. Treating duplicate values:
4. Exploratory Data Analysis:
 - 4.1. Data Visualization of Main Features:
 - 4.2. Other Binary Features of the Dataset:
 - 4.3. Correlation Analysis:
5. Data pre-processing:
 - 5.1. Label Encoding(Ordinal Encoding Technique):
 - 5.2. Feature Scaling:
 - 5.2.1. Method 1 - Standard Scalar:
 - 5.2.2. Method 2 – Min-Max Scalar:
 - 5.2.3. Method 3 - Without Feature Scaling:
 - 5.2.3.1. Outlier Treatment:
 - 5.3. Feature Selection:
 - 5.3.1. Procedure 1 - Feature Selection:
 - 5.3.2. Procedure 2 - Principal Component Analysis (PCA):
- 6.: Methodology & Implementation:
 - 6.1. Model Selection after Standard Scaling:
 - 6.2. Model Selection after Min-Max Scaling:
 - 6.3. Model Selection without Feature Scaling:
 - 6.3.1. Model Selection after Outlier Treatment:
 - 6.4.1.Model Selection after Feature Selection:
 - 6.4.2. Model Selection after PCA:
 - 6.5. Neural Network (ANN):

7. Results:

7.1. Comparison between all the Regression models & methods:

7.2. Neural Network (ANN):

7.3. Regression Model vs Neural Network (Comparison):

8. Analysis:

9. Conclusion & Future Work:

10. References:

11. Appendices

1.Introduction:

Due to the fact that I was born and raised in a large city, I have seen how the city grows and how housing costs increase as a result of local amenity availability. This was my driving force behind compiling a dataset for analysis.

Let's now explore the elements that influence pricing!

Every day, thousands of homes are sold. Every buyer has some internal concerns, such as: What is the true value of this house? Am I receiving a decent value?? In this study, a machine learning model is suggested to forecast the price of a home using information about the home (its size, location, Bedrooms, etc.). I will display the code used for each step of the creation and assessment of our model, followed by its results. This will make our work easier to reproduce. The Python programming language and a variety of Python packages will be used in this study.

1.1.Goals of the Study :

The following are the study's primary goals:

- To use data preparation and pre-processing methods to obtain clean data.
- To create machine learning models that can forecast home prices based on attributes of the home.
- In order to compare and evaluate model performance and select the best model.

1.2.Paper Organisation:

This paper is structured as follows: in the next section, section 2, we evaluate papers related to our work/a scientific journal literature review. In section 3, we cover Descriptive analysis of the dataset, which depicts six data files, a list of features, a description of all features, the shape of all features, data cleaning, missing values, the treatment of duplicate values, and feature engineering. Next, in section 4, we examine exploratory data analysis, including data visualisation of primary features, other binary aspects of datasets, and correlations between dependent and independent features. In section 5, we demonstrated data pre-processing approaches/methods such as label encoding, feature scaling using standard scalar, min-max scalar, and without scaling, and feature selection strategies such as principal component analysis and feature importance. Next, in section 6, we examine the methodology and implementation, as well as the type of machine-learning prediction that should be deployed; I also list the prediction techniques that will be used. I've also defined the algorithm for deep learning in this section. In section 7, we analyse and compare the section 6 results, describe the neural network algorithm, compare regression and neural network models, and conclude the paper with an analysis. In sections 10 and 11, I've listed my bibliography and appendices.

2.Literature Review/ Related work:

Different Regression Adopted In The Study:

In machine learning, we deploy a variety of techniques to teach computers how the relationships within the provided data work and to enable them to predict the future based on patterns or rules found in the dataset. Regression is a machine learning method that predicts the outcome as a continuous numerical value. The association between a single dependent variable (the target variable) and multiple independent ones is discovered using regression analysis, which is frequently used in banking, investing, and other fields. The most frequent regression issues are, for instance, projecting housing prices, stock market fluctuations, or employee salaries. (Regression analysis in machine learning - javatpoint).

Approximating a mapping function (f) from input variables (X) to a continuous output variable (y) is the goal of regression predictive modelling .

A real-value, such as an integer or floating point value, is a continuous output variable. These are frequently quantities like sums and sizes.

For instance, it might be projected that a house will sell for a certain amount of money, possibly in the neighbourhood of \$200,000.

- A regression problem demands for a quantity forecast.
- Real-valued or discrete input variables are both acceptable in a regression.
- A multivariate regression problem is one that has several input variables.

The diagrams below can be used to better understand regression techniques.

Linear Regression:

Linear Regression is a machine learning (ML) algorithm utilised for supervised learning. The objective of linear regression is to forecast a dependent variable (target) based on the provided independent variable (s). Therefore, this regression method establishes a linear relationship between a dependent variable and the other independent variables. Consequently, this algorithm is called Linear Regression. (Sharma, 2022).

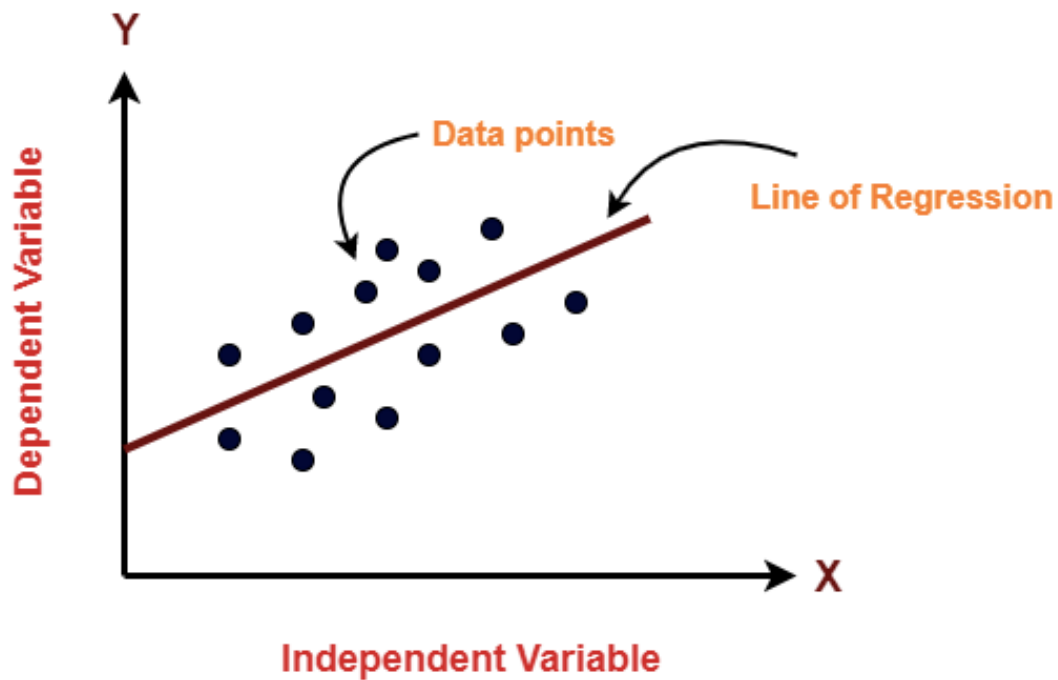


Figure 1: Linear Regression

On the X-axis of the picture above is the independent variable, and on the Y-axis is the output. The regression line provides the best model fit. And the primary purpose of this algorithm is to identify the optimal fit line. (Sharma, 2022).

Pros:

- Implementing linear regression is straightforward.
- Having less complexity than other algorithms.
- Using some dimensionality reduction approaches, regularisation techniques, and cross-validation, it is possible to avoid over-fitting in linear regression models.

Cons:

- Outliers negatively impact this algorithm.
- It oversimplifies real-world situations by assuming a linear relationship between the variables, and is therefore unsuitable for actual applications.

Decision Tree:

The decision tree models can be used to all data sets with numerical and categorical attributes. Decision trees are effective at capturing the nonlinear relationship between

characteristics and the goal variable. Due to the resemblance between decision trees and human cognitive processes, comprehending the data is quite intuitive.

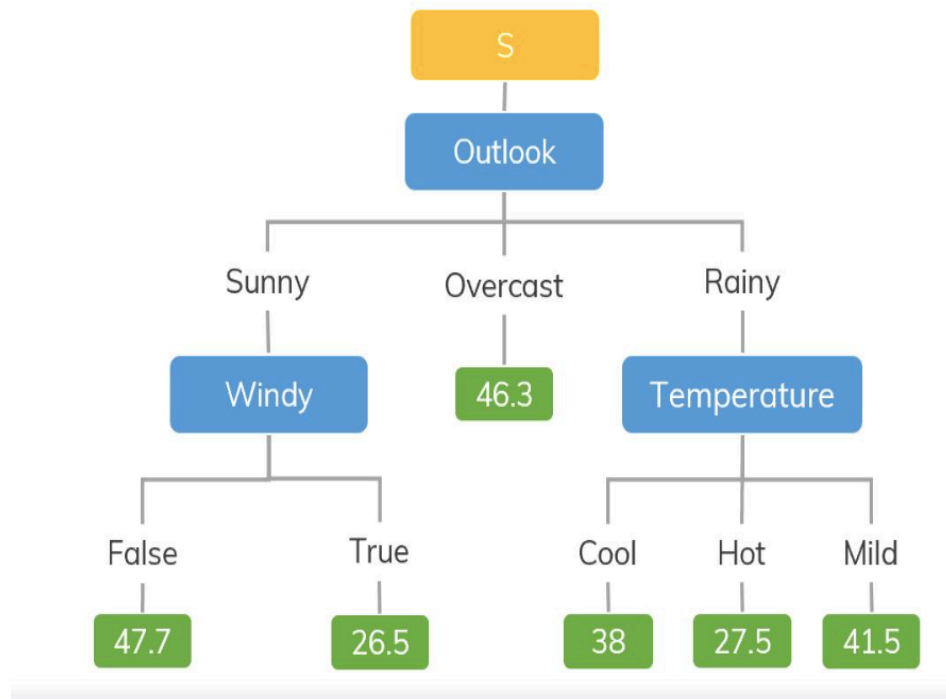


Figure 2: Decision Tree

For instance, if we are classifying the number of hours a child spends playing in specific weather, the decision tree would resemble the one pictured above. A decision tree is a tree in which each node represents a feature, each branch represents a decision, and each leaf represents a result (numerical value for regression). (Sharma, 2022).

Pros:

- Easily comprehensible and interpretable; visually intuitive.
- It can handle numerical and categorical characteristics.
- Minimal data pre-processing is required, as there is no requirement for one-hot encoding, dummy variables, etc.

Cons:

- It tends to overfit.
- A slight change in the data often results in a substantial change in the tree structure, resulting in instability.

Random Forest Regressor:

Random Forests are collections of decision trees. It's a classification and regression technique based on Supervised Learning. Numerous decision trees are applied to the supplied data. It operates by creating a variable number of decision trees during training and outputting the class that is the mode of the classes (for classification) or the mean prediction (for regression) of the individual trees. (Sharma, 2022).

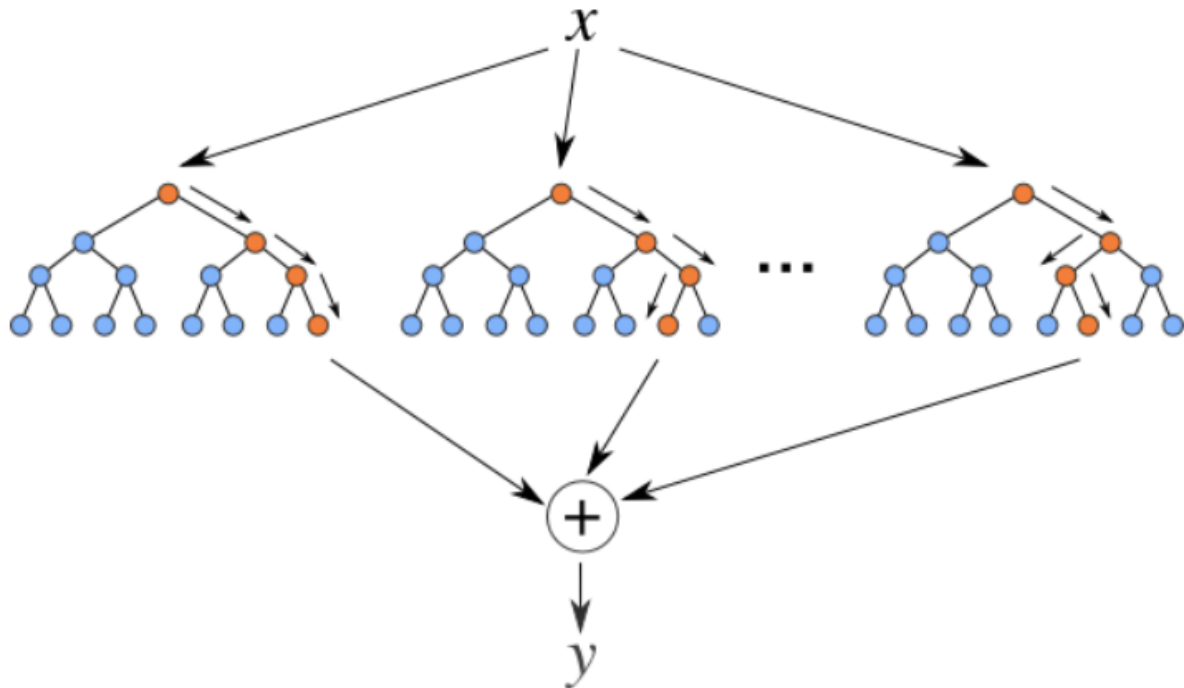


Figure 3: Random Forest

Pros:

- Able to comprehend complicated and non-linear relationships.
- Very simple to interpret and comprehend.

Cons:

- They have a tendency to overfit.
- Using larger random forest ensembles to improve performance reduces their speed and increases their memory requirements.

K-Nearest Neighbours:

K nearest neighbours is a straightforward algorithm that saves all available cases and predicts the numerical target using a measure of similarity (e.g., distance functions). KNN has been

utilised in statistical estimates and pattern recognition as a non-parametric technique since the early 1970s. Algorithm A straightforward use of KNN regression involves calculating the mean of the numerical target's K nearest neighbours. A different strategy employs an inverse distance-weighted average of the K nearest neighbours. Similar to KNN classification, KNN regression employs distance functions. (Muhajir, 2020).

Distance functions

Euclidean	$\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$
Manhattan	$\sum_{i=1}^k x_i - y_i $
Minkowski	$\left(\sum_{i=1}^k (x_i - y_i)^q \right)^{1/q}$

Figure 4: K Nearest Neighbour

Pros:

- No Training Period
- Easy Implementation

Cons:

- Does not work well with large dataset
- Does not work well with high dimensionality
- Sensitive to noisy and missing data
- Feature Scaling - Data in every dimension ought to be scaled (normalised and standardised) appropriately.

Boosting Techniques:

Boosting is a form of ensemble learning that combines a bunch of weak learners into a single strong learner in order to reduce the number of training errors. When utilising boosting, a random sample of data is taken, then it is fitted with a model, and then the model is trained progressively. This means that each successive model strives to address the problems of its predecessor. After each cycle, the weak prediction rules generated by each individual classifier are combined to generate a single, robust rule. (Lateef, 2022).

Types of Boosting:

Some popular types of boosting methods include:

Gradient boosting:

Building on the work of Leo Breiman, Jerome H. Friedman developed gradient boosting, which works by adding predictors to an ensemble sequentially, with each predictor correcting for the errors of its predecessor. Unlike AdaBoost, gradient boosting trains on the residual errors of the preceding prediction, as opposed to modifying the weights of data points. Gradient boosting is its name because it combines the gradient descent approach with the boosting method.

Extreme gradient boosting, or XG Boost:

Gradient boosting solution optimised for computing performance and scalability. XG Boost takes advantage of several CPU cores to provide parallel learning during training.

Pros:

- Ease of Implementation
- Reduction of bias
- Computational Efficiency

Cons:

- Overfitting
- Intense computation

Given the interest in house price prediction there are numerous publications and articles available that discuss the approaches taken and models used.

Virtual Reality for Real Estate.

This article discusses the findings of the VR4RE (Virtual Reality for Real Estate) initiative, which intends to save both sellers and buyers of real estate time and money through the use of modern technologies. VR4RE is one of Bluemind Software's creative initiatives, and it is in an advanced condition. This study also details the history of in-house technical efforts to develop 3D and VR presentation tools for real estate properties (Virtual Reality). (Deaky & Parv, 2017).

Developing Smart Commercial Real Estate.

To evaluate the possibilities of smart commercial real estate (CRE), researchers examined a Swedish commercial real estate firm that has created and implemented a technology-based self-service (TBSS) to assist renters in reducing their energy consumption. (Ekman et al., 2016).

Comparison of Ensemble Methods for Real Estate Appraisal

Four ensemble approaches, namely Bagging, Random Forest, Gradient Boosting, and Extreme Gradient Boosting, were evaluated and contrasted in terms of their efficacy in the valuation of Mumbai real estate in this research. This study used the property listings available on the real estate website 99acres as its data source. The investigation demonstrated that the Extreme Gradient Boosting (XGBoost) model outperformed the other ensemble models. The results demonstrate that ensemble models can be utilised to estimate home prices. (Kumkar et al., 2018).

Prediction of House Pricing Using Machine Learning with Python.

This paper provides an overview of how to predict the cost of a home using several regression techniques and Python modules. The proposed method took into account the more refined factors utilised in the calculation of home values and provided a more accurate forecast. In addition, it provides an overview of the numerous graphical and numerical techniques that will be necessary to forecast the price of a home. This paper describes what and how the machine learning-based house price model operates, as well as the dataset utilised by our proposed approach. (Jain et al.).

3.Descriptive analysis of the dataset:

The next picture provides an illustration of the six data files, together with a list of the features that are unique to each data file and a description of those features:-

Datasets	Feature name	Description
mumbai_df/chennai_df/kolkata_df /hyderabad_df/bangalore_df/delhi_df.csv	Price	Price of the house in INR
mumbai_df/chennai_df/kolkata_df /hyderabad_df/bangalore_df/delhi_df.csv	Area	Area of the house in sq. ft

mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	Location	Location of the Property
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	No. of Bedrooms	Number of bedrooms in that property
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	Resale	Resale: 1 New: 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	Maintenance Staff	Maintenance Staff present for the property Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	Gymnasium	Gymnasium around property Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	SwimmingPool	SwimmingPool in the property Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	LandScapedGardens	LandScapedGardens in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	JoggingTracks	JoggingTracks near property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	RainWaterHarvesting	RainWaterHarvesting near the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	IndoorGames	IndoorGames in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	ShoppingMall	ShoppingMall near the property.

		Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	Intercom	Intercom in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	SportsFacility	SportsFacility near the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	ATM	ATM near the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	ClubHouse	ClubHouse near the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	School	School near the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	24X7Security	24X7Security in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	PowerBackup	PowerBackup in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	CarParking	CarParking near the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	StaffQuarter	StaffQuarter near the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/banglore_df/delhi_df.csv	Cafeteria	Cafeteria near the property.

		Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hydrabad_df/banglore_df/delhi_df.csv	MultipurposeRoom	MultipurposeRoom near the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hydrabad_df/banglore_df/delhi_df.csv	Hospital	Hospital near the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hydrabad_df/banglore_df/delhi_df.csv	WashingMachine	WashingMachine in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hydrabad_df/banglore_df/delhi_df.csv	Gasconnection	Gasconnection in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hydrabad_df/banglore_df/delhi_df.csv	Wifi	Wifi in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hydrabad_df/banglore_df/delhi_df.csv	Children'splayarea	Children'splayarea in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hydrabad_df/banglore_df/delhi_df.csv	LiftAvailable	LiftAvailable in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hydrabad_df/banglore_df/delhi_df.csv	BED	BED in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hydrabad_df/banglore_df/delhi_df.csv	VaastuCompliant	VaastuCompliant Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hydrabad_df/banglore_df/delhi_df.csv	Microwave	Microwave in the property. Available : 1 Unavailable : 0

mumbai_df/chennai_df/kolkata_df /hyderabad_df/bangalore_df/delhi_df.csv	GolfCourse	GolfCourse near the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/bangalore_df/delhi_df.csv	TV	TV in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/bangalore_df/delhi_df.csv	DiningTable	DiningTable in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/bangalore_df/delhi_df.csv	Sofa	Sofa in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/bangalore_df/delhi_df.csv	Wardrobe	Wardrobe in the property. Available : 1 Unavailable : 0
mumbai_df/chennai_df/kolkata_df /hyderabad_df/bangalore_df/delhi_df.csv	Refrigerator	Refrigerator in the property. Available : 1 Unavailable : 0

Table 1: Description of the data files

The purpose of this project is to generate predictions of Price based on the dataframe final_df.

Table 2: Shape of the data files

Data name	Rows	Columns
Mumbai.csv	7719	40
Chennai.csv	5014	40
Kolkata.csv	6507	40
Hyderabad.csv	2518	40
Bangalore.csv	6207	40

Information of the dataset.....

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 32963 entries, 0 to 32962

Data columns (total 40 columns):

#	Column	Non-Null Count	Dtype
---	-----	-----	-----
0	Price	32963 non-null	int64
1	Area	32963 non-null	int64
2	Location	32963 non-null	object
3	No. of Bedrooms	32963 non-null	int64
4	Resale	32963 non-null	int64
5	MaintenanceStaff	32963 non-null	int64
6	Gymnasium	32963 non-null	int64
7	SwimmingPool	32963 non-null	int64
8	LandscapedGardens	32963 non-null	int64
9	JoggingTrack	32963 non-null	int64
10	RainWaterHarvesting	32963 non-null	int64
11	IndoorGames	32963 non-null	int64
12	ShoppingMall	32963 non-null	int64
13	Intercom	32963 non-null	int64
14	SportsFacility	32963 non-null	int64
15	ATM	32963 non-null	int64
16	ClubHouse	32963 non-null	int64
17	School	32963 non-null	int64
18	24X7Security	32963 non-null	int64
19	PowerBackup	32963 non-null	int64
20	CarParking	32963 non-null	int64
21	StaffQuarter	32963 non-null	int64
22	Cafeteria	32963 non-null	int64
23	MultipurposeRoom	32963 non-null	int64
24	Hospital	32963 non-null	int64
25	WashingMachine	32963 non-null	int64
26	Gasconnection	32963 non-null	int64
27	AC	32963 non-null	int64
28	Wifi	32963 non-null	int64
29	Children'splayarea	32963 non-null	int64
30	LiftAvailable	32963 non-null	int64

```

31 BED 32963 non-null int64
32 VaastuCompliant 32963 non-null int64
33 Microwave 32963 non-null int64
34 GolfCourse 32963 non-null int64
35 TV 32963 non-null int64
36 DiningTable 32963 non-null int64
37 Sofa 32963 non-null int64
38 Wardrobe 32963 non-null int64
39 Refrigerator 32963 non-null int64

```

```
dtypes: int64(39), object(1)
```

```
memory usage: 10.1+ MB
```

```
None
```

```
Statistically infomation of the dataset.....
```

	Price	Area	No. of Bedrooms	Resale \
count	3.296300e+04	32963.000000	32963.000000	32963.000000
mean	1.168672e+07	1293.362194	2.411765	0.381397
std	2.307368e+07	763.703754	0.812800	0.485737
min	2.000000e+06	200.000000	1.000000	0.000000
25%	4.071500e+06	853.000000	2.000000	0.000000
50%	6.711000e+06	1125.000000	2.000000	0.000000
75%	1.200000e+07	1500.000000	3.000000	1.000000
max	8.546000e+08	16000.000000	9.000000	1.000000

	MaintenanceStaff	Gymnasium	SwimmingPool	LandscapedGardens \
count	32963.000000	32963.000000	32963.000000	32963.000000
mean	6.296454	6.403725	6.371325	6.351151
std	4.075024	3.917995	3.966385	3.996086
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	1.000000	1.000000
50%	9.000000	9.000000	9.000000	9.000000
75%	9.000000	9.000000	9.000000	9.000000
max	9.000000	9.000000	9.000000	9.000000

	JoggingTrack	RainWaterHarvesting	...	LiftAvailable	BED	\
count	32963.000000	32963.000000	...	32963.000000	32963.000000	
mean	6.346328	6.356855	...	6.463671	6.272154	
std	4.003140	3.987722	...	3.826128	4.109373	
min	0.000000	0.000000	...	0.000000	0.000000	
25%	1.000000	1.000000	...	1.000000	0.000000	
50%	9.000000	9.000000	...	9.000000	9.000000	
75%	9.000000	9.000000	...	9.000000	9.000000	
max	9.000000	9.000000	...	9.000000	9.000000	

	VaastuCompliant	Microwave	GolfCourse	TV	\
count	32963.000000	32963.000000	32963.000000	32963.000000	
mean	6.335497	6.259169	6.250887	6.261293	
std	4.018912	4.127551	4.139083	4.124586	
min	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	0.000000	0.000000	0.000000	
50%	9.000000	9.000000	9.000000	9.000000	
75%	9.000000	9.000000	9.000000	9.000000	
max	9.000000	9.000000	9.000000	9.000000	

	DiningTable	Sofa	Wardrobe	Refrigerator
count	32963.000000	32963.000000	32963.000000	32963.000000
mean	6.260413	6.259867	6.250675	6.260019
std	4.125815	4.126577	4.139378	4.126365
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	9.000000	9.000000	9.000000	9.000000
75%	9.000000	9.000000	9.000000	9.000000
max	9.000000	9.000000	9.000000	9.000000

[8 rows x 39 columns]

Following that, we assigned classes to the characteristics shown in table2. The variable of interest is denoted in bold by the price. The pre-processing stage of data analysis begins with the classification of the variables' types, which is necessary for future encoding and scaling.

Table 2 : Features and its type

Categorical variables		Numeric variables	
Nominal	Ordinal	Interval-scaled	Ratio-scaled
Resale, MaintenanceStaff, Gymnasium, SwimmingPool, LandscapedGardens, JoggingTrack, RainWaterHarvesting, IndoorGames, ShoppingMall, Intercom, SportsFacility, ATM,	Location	No. of Bedrooms	Price, Area

ClubHouse, School,
24X7Security,
PowerBackup,
CarParking,
StaffQuarter,
Cafeteria,
MultipurposeRoom,
Hospital,
WashingMachine,
Gasconnection, AC,
Wifi,
Children'splayarea,
LiftAvailable, BED,
VaastuCompliant,
Microwave,
GolfCourse, TV,
DiningTable, Sofa,
Wardrobe,
Refrigerator.

3.1 Missing value

From the next figure, we can see the result of the missing value analysis, it is possible to draw the conclusion that we do not have any null values. However, based on the columns and the data analysis, we are able to understand that the missing values are in the form of the number 9. Because this number does not make any sense in the dataset, we will treat it as a nan value and replace it. Therefore, we will be removing the rows that contain null values row by row while simultaneously attempting to obtain a prediction for our model.

Price	0
Area	0
Location	0
No. of Bedrooms	0
Resale	0
MaintenanceStaff	0
Gymnasium	0
SwimmingPool	0
LandscapedGardens	0
JoggingTrack	0
RainWaterHarvesting	0
IndoorGames	0
ShoppingMall	0
Intercom	0
SportsFacility	0
ATM	0
ClubHouse	0
School	0
24X7Security	0
PowerBackup	0
CarParking	0
StaffQuarter	0
Cafeteria	0
MultipurposeRoom	0
Hospital	0
WashingMachine	0
Gasconnection	0
AC	0
Wifi	0
Children'splayarea	0
LiftAvailable	0
BED	0
VaastuCompliant	0
Microwave	0
GolfCourse	0
TV	0
DiningTable	0
Sofa	0
Wardrobe	0
Refrigerator	0

Price	0
Area	0
Location	0
Resale	0
MaintenanceStaff	22870
Gymnasium	22870
SwimmingPool	22870
LandscapedGardens	22870
JoggingTrack	22870
RainWaterHarvesting	22870
IndoorGames	22870
ShoppingMall	22870
Intercom	22870
SportsFacility	22870
ATM	22870
ClubHouse	22870
School	22870
24X7Security	22870
PowerBackup	22870
CarParking	22870
StaffQuarter	22870
Cafeteria	22870
MultipurposeRoom	22870
Hospital	22870
WashingMachine	22870
Gasconnection	22870
AC	22870
Wifi	22870
Children'splayarea	22870
LiftAvailable	22870
BED	22870
VaastuCompliant	22870
Microwave	22870
GolfCourse	22870
TV	22870
DiningTable	22870
Sofa	22870
Wardrobe	22870
Refrigerator	22870

Figure 5: Replacing 9 with nan

When we change the value 9 to nan, we discover that the dataset from figure 5 has a significant number of null values.

The percentage of missing values in the dataset can be seen in figure 6, and based on this information, we are able to estimate that these features will be of the least use to us when it comes to making predictions for our model because they will prevent us from achieving the results we want.

Missing Values	
MaintenanceStaff	69.380821
BED	69.380821
WashingMachine	69.380821
Gasconnection	69.380821
AC	69.380821
Wifi	69.380821
Children'splayarea	69.380821
LiftAvailable	69.380821
VaastuCompliant	69.380821
MultipurposeRoom	69.380821
Microwave	69.380821
GolfCourse	69.380821
TV	69.380821
DiningTable	69.380821
Sofa	69.380821
Wardrobe	69.380821
Hospital	69.380821
Cafeteria	69.380821
Gymnasium	69.380821
Intercom	69.380821
SwimmingPool	69.380821

SwimmingPool	69.380821
LandscapedGardens	69.380821
JoggingTrack	69.380821
RainWaterHarvesting	69.380821
IndoorGames	69.380821
ShoppingMall	69.380821
SportsFacility	69.380821
StaffQuarter	69.380821
ATM	69.380821
ClubHouse	69.380821
School	69.380821
24X7Security	69.380821
PowerBackup	69.380821
CarParking	69.380821
Refrigerator	69.380821

Figure 6: Missing Values Percentage

3.2 Treating duplicate values

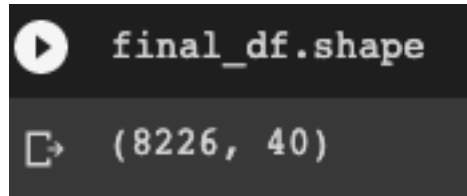
```
[ ] # Printing duplicate values in the dataset
    print(len(final_df[final_df.duplicated()]))

1867
```

Figure 7: Length of Duplicate Values

Figure 7 reveals that our dataset has 1867 values that are identical to one another; hence, we are going to remove all of the duplicated values from the dataset.

From figure 8, we can see that after all the data cleaning that was done in the dataset, we now have 8226 rows and 40 columns to work with. We can now go on to the next step.



```
final_df.shape
```

```
(8226, 40)
```

Figure 8: Shape of final_df value

4. Exploratory Data Analysis:

4.1. Data Visualization of Main Features:

Price Distribution

The distribution of prices in Indian rupees, also known as lakhs, for each of the cities included in the dataset is depicted graphically in figure 9. Because we want to make it as easy as possible for you to visualise the distribution, we have combined two graphs into a single plot. In the first place, let's have a look at the histogram, which shows us that the majority of the distribution falls between the range of 0-350 on the graph. And based on the second graph, we can see quite clearly that the boxplot has a significant number of outliers, which are values that fall outside of the upper bound range of 350. In the second half, we are going to deal with this issue.

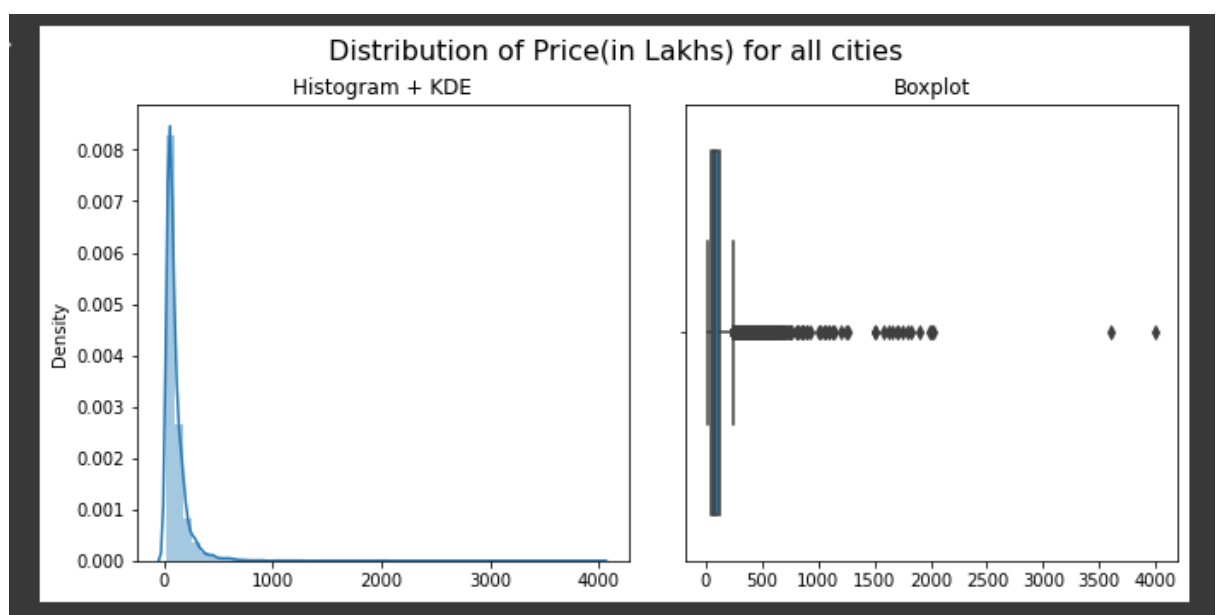


Fig 9: Price Distribution of all cities(in lakhs)

Area Distribution:

In this figure 10, we have attempted to depict the distribution of land area across all of India's major urban areas. To begin, we have a histogram as well as a KDE plot, both of which allow us to deduce that the maximum value falls somewhere between 0 and 3000. And second of all, we are able to observe that the present outlier is greater than the upper bound limit of 3000, which is something that we are going to deal with in the later part of this discussion.

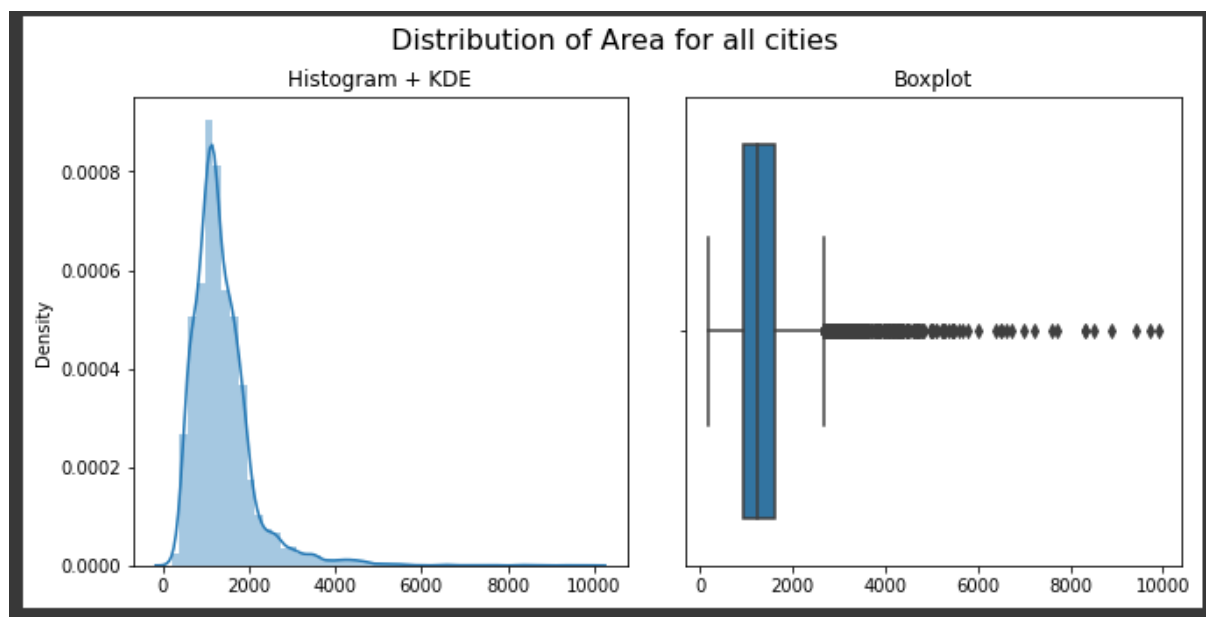


Fig 10: Area Distribution of all cities

New & Resale Properties Distribution:

From the information presented in figure 11, we are able to deduce that Dwarka Mor, Uttam Nagar has the greatest counts according to the location, followed by Kukatpally and Kondapur. This information is based on the first plot in the New properties value count

according to the locations category.

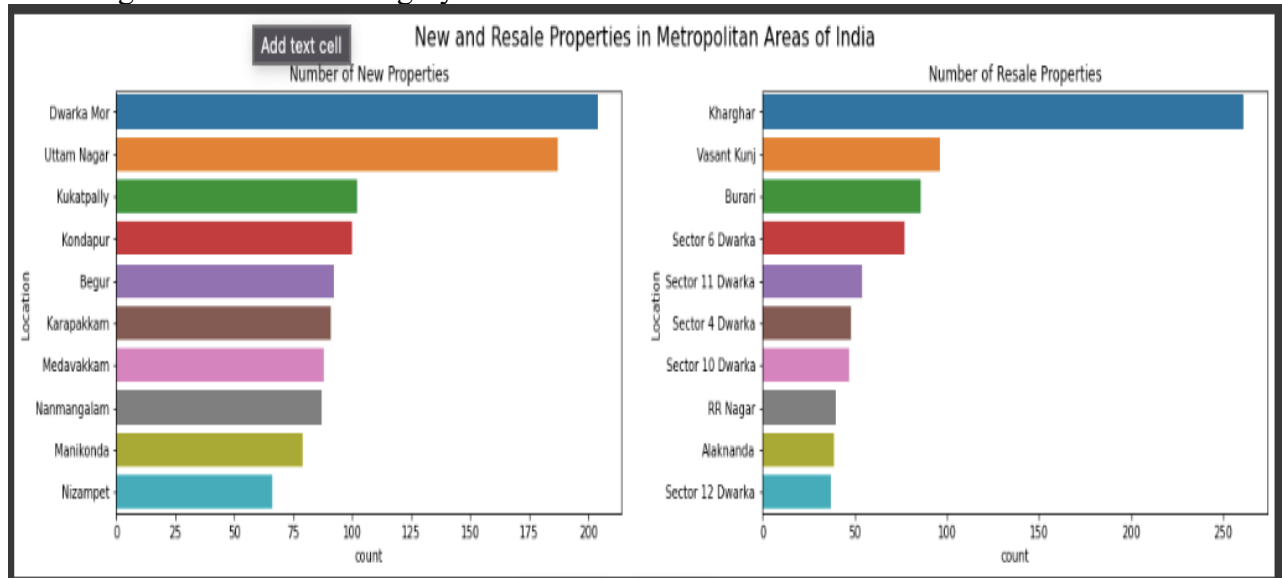


Fig 11: Distribution of New & Resale Properties

Based on the results of the second plot, in which we counted resale properties according to location, it is abundantly clear that Kharghar has the greatest number of resale properties, followed by Vasant Kunj, Burari, and Sector 6 Dwarka, each of which has approximately one-half as many resale properties as Kharghar does.

Bar plot of Resale Properties for mean Price:

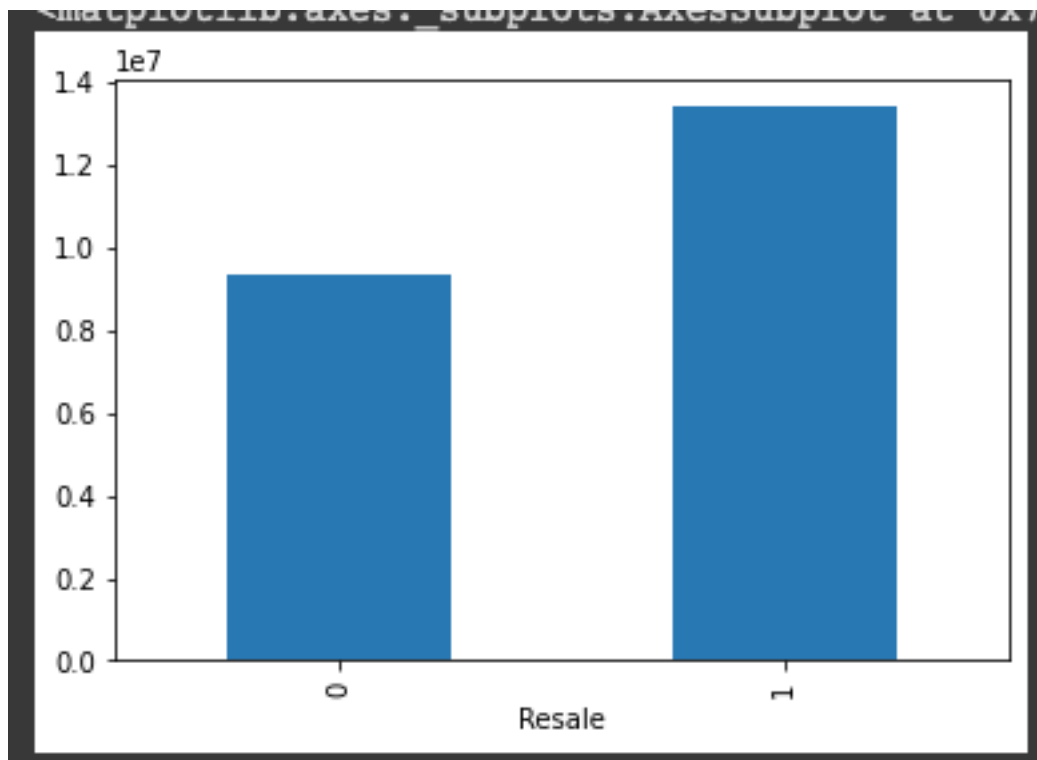


Fig 12: Mean price of Resale Properties

Creating a bar graph to show the average selling price of the item being resold. Therefore, we are able to say that the value of our property for Resale(1) is higher than the value of our property for Non-resale(0).

Top 25 Properties by location:

It is possible to infer from figure 13 that the top 25 properties in this graph are organised according to their location. Therefore, the first three cities that immediately come to mind are Kharghar, then Dwarka Mor, and finally Uttam Nagar. Because of this, we can also conclude that prices in and around these locations would rise, as a result, because these are the areas that are experiencing the highest levels of demand.

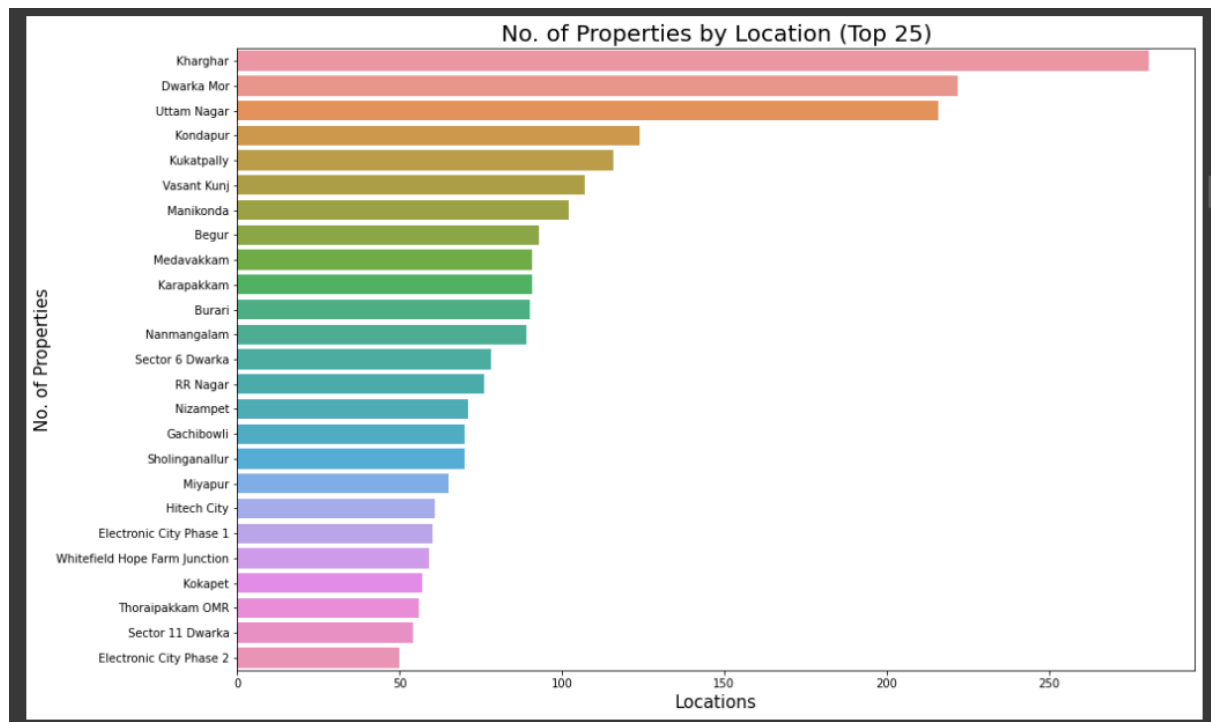


Fig 13: Top 25 Properties by Location

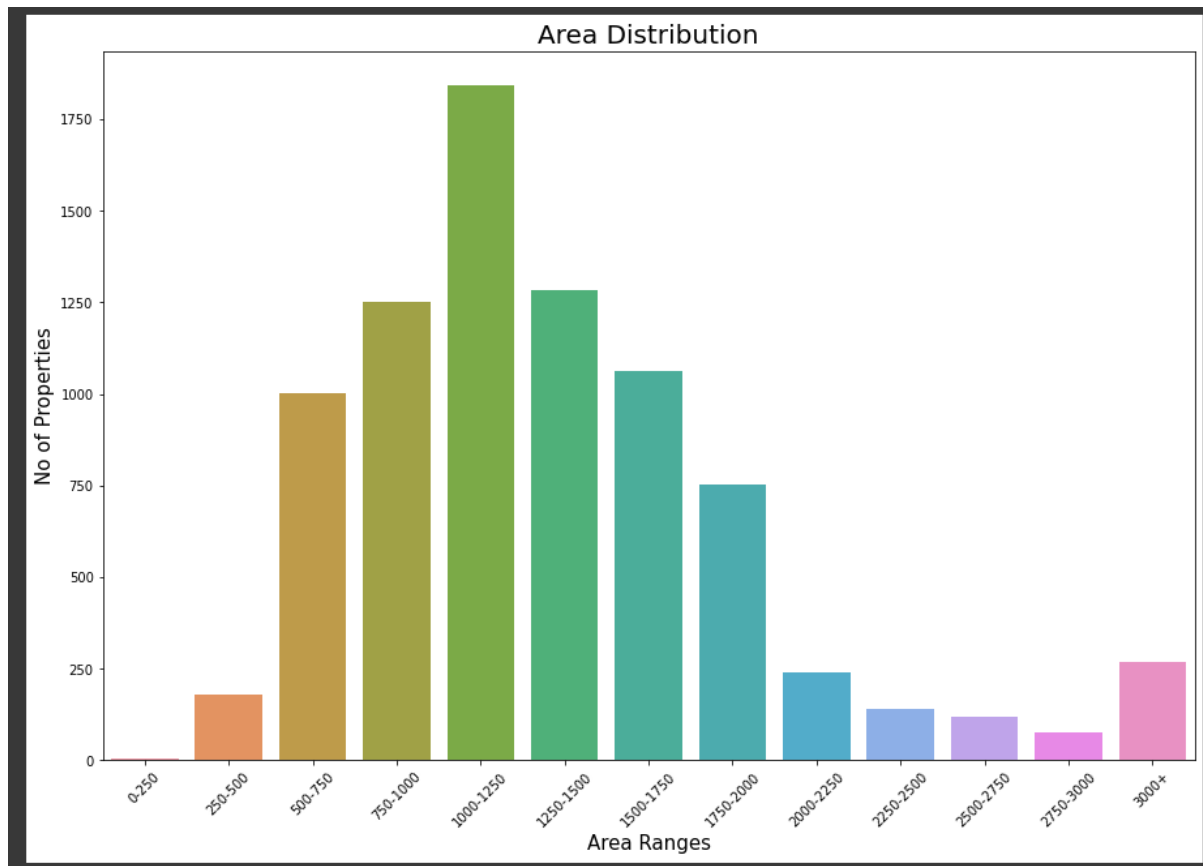


Fig 14: Area Distribution Bar Graph

The normal distribution may be seen to be followed by the area distribution graph (fig. 14). The highest concentration of homes may be found in the range from 1000 to 1250, which is followed by the 750-1000 & 1250 to 1500 range. Based on this information, we may deduce that a big group of people are interested in purchasing a home in the price range of 500 to 2000. The market demand is particularly strong in this particular region range. There is a portion of the general public that appears to have an interest in areas larger than 3000 square feet.

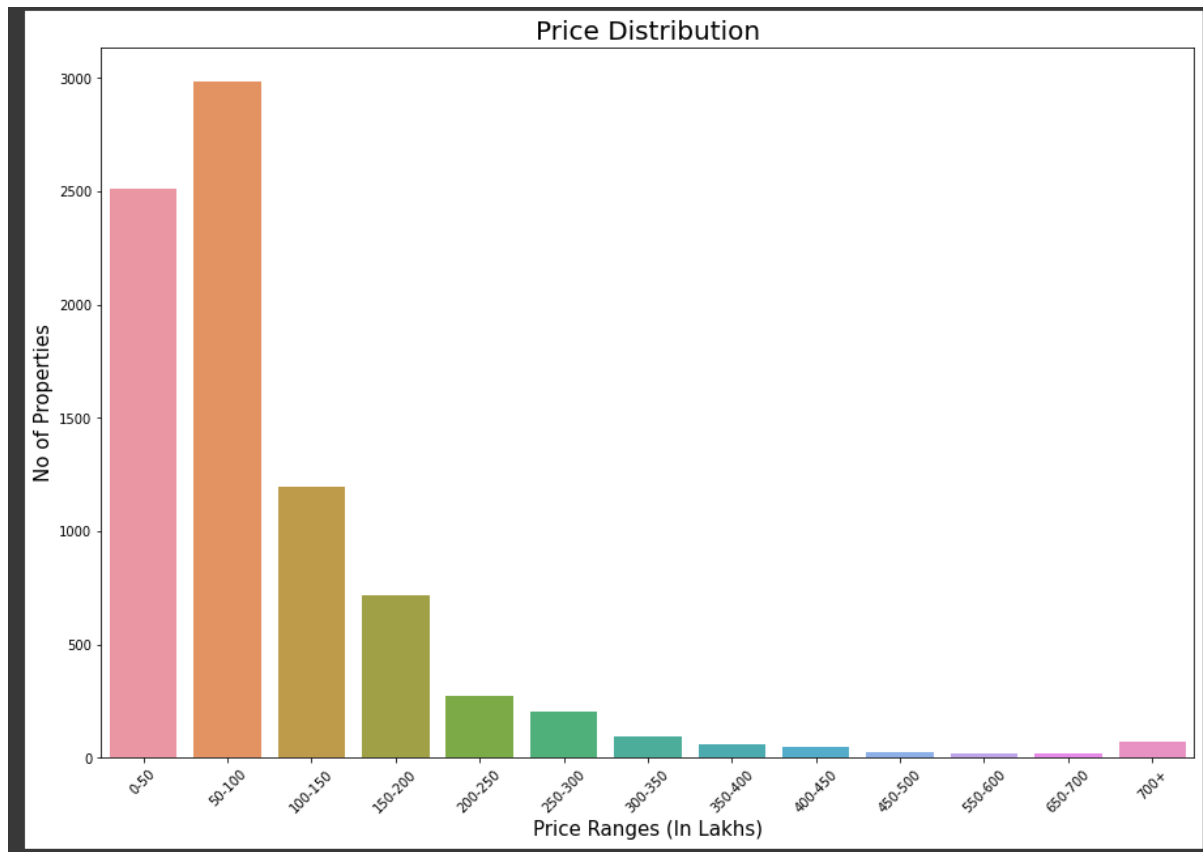


Fig 15: Price Distribution Bar Graph

The price range between 50 and 100 lakhs has the most properties, followed by the price ranges between 0 and 50 and then between 100 and 150. This information is presented in the form of a bar graph in figure 15 entitled "Price Distribution." When we look closely, we can see that the number of available properties reduces as the price goes up. If we restrict our focus to homes with prices between 0 and 100 lakhs, where the majority of people are looking to make their purchase, and then we will see a significant drop in the chart.

In fig. 16, Number of bedrooms, the majority of properties built in terms of the number of bedrooms are 2 and 3, respectively, having a count about around 3500, which is the largest, and then 1 and 4, respectively, following in that order. The public has an extremely strong demand for homes with two and three bedrooms.

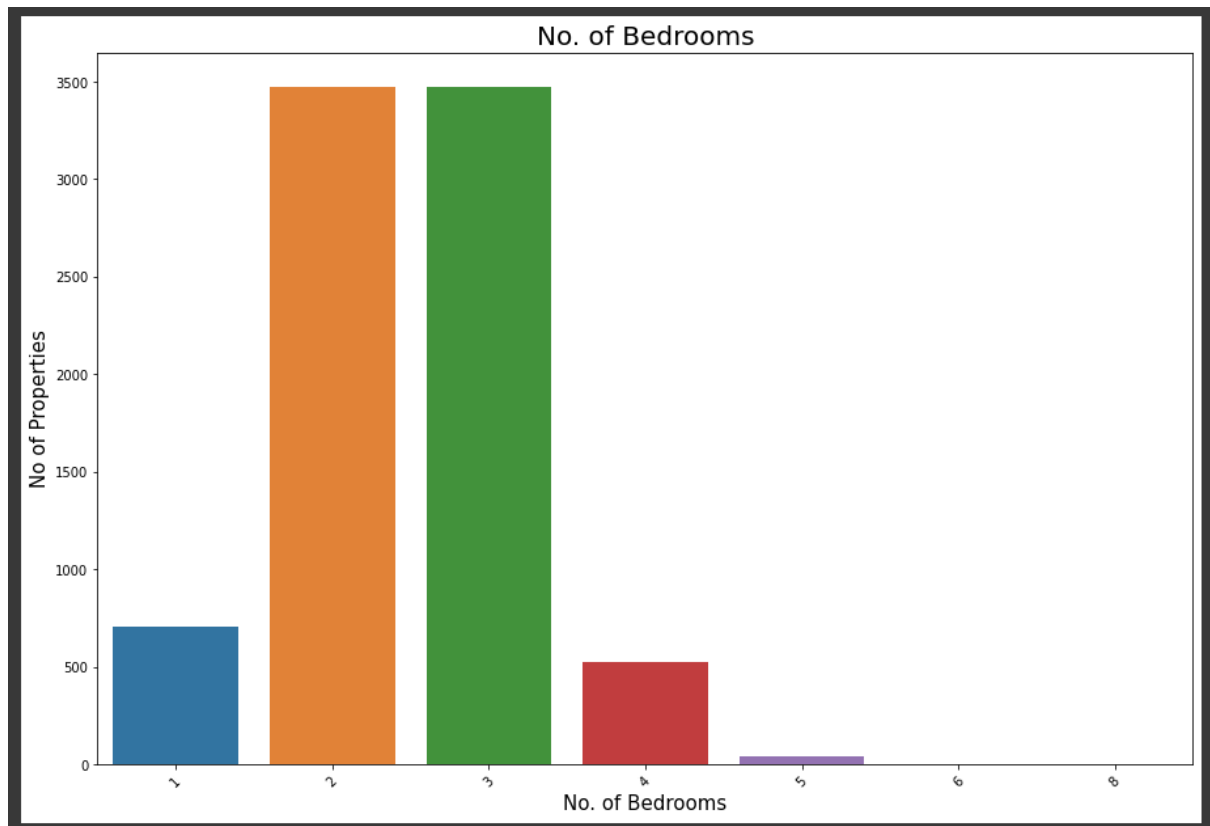
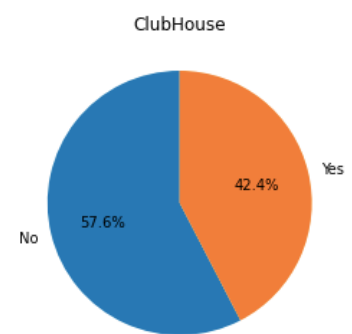
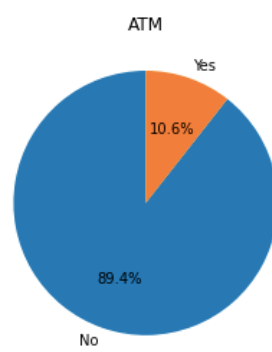
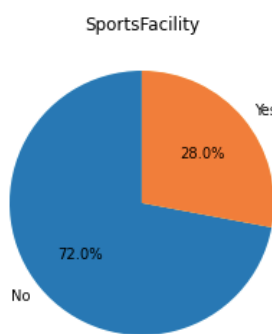
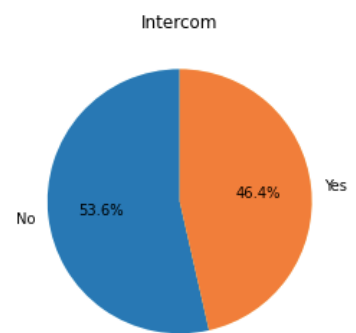
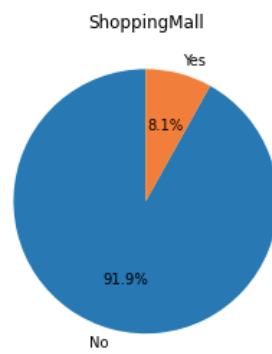
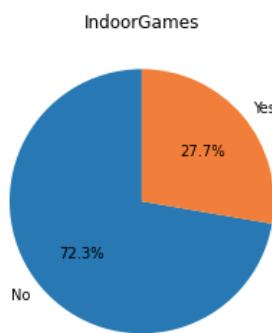
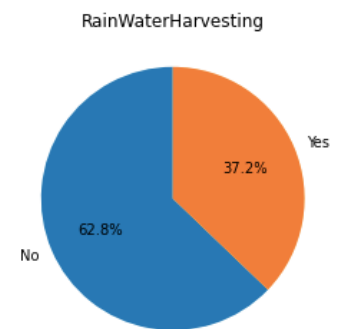
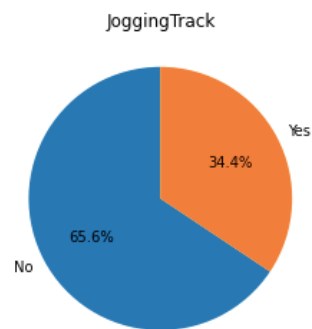
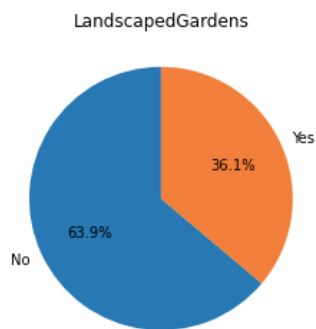
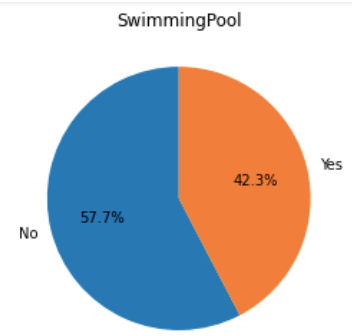
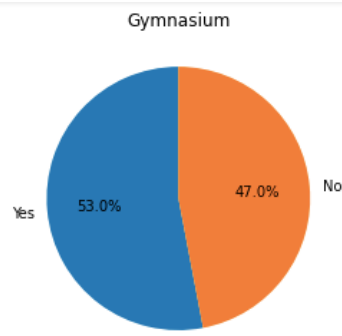
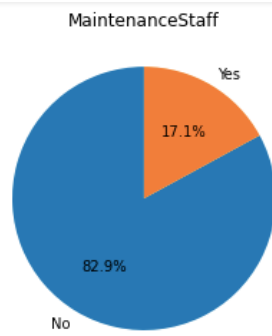
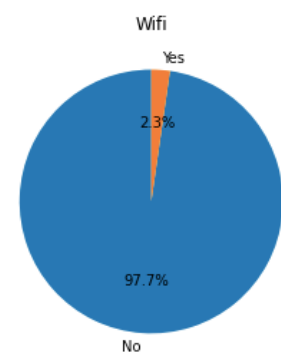
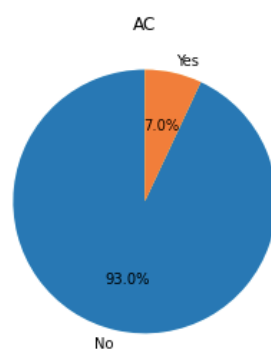
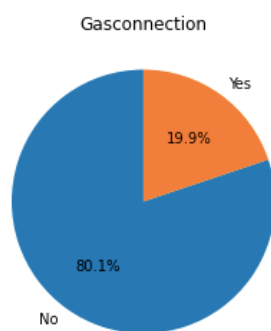
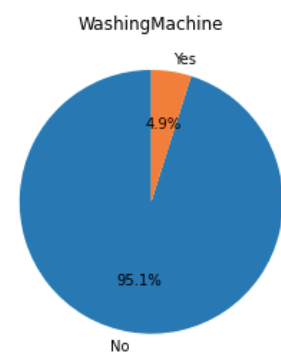
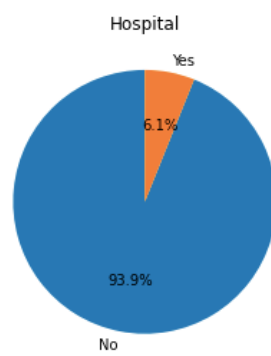
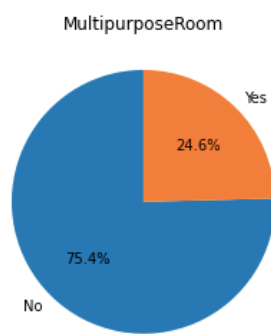
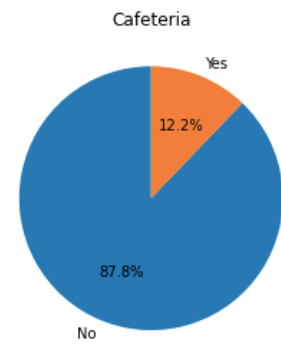
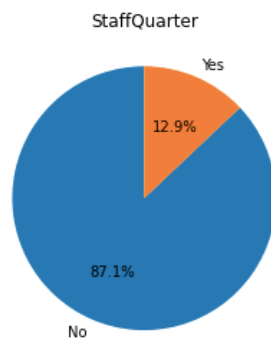
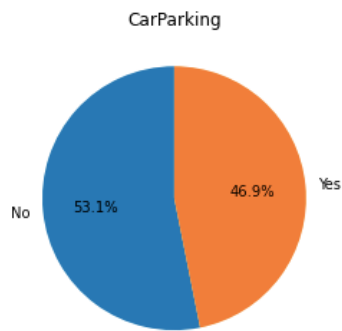
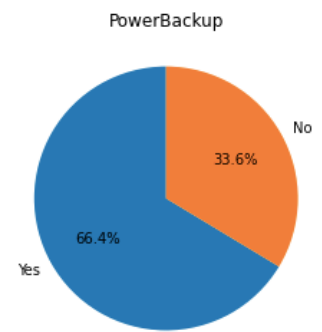
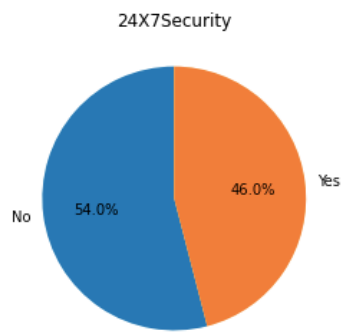
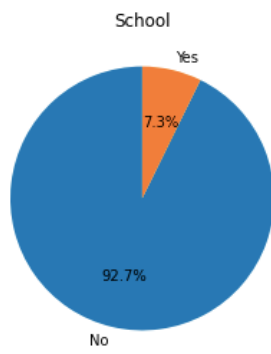


Fig 16: No. of Bedrooms Bar Graph

4.2. Other Binary Features of the Dataset:

Figure 17 clearly demonstrates that these are the binary characteristics of the dataset we are working with. The presentation takes the shape of a pie chart. These are some additional elements that can cause the price of the house to change; we will endeavour to estimate all of the features in order to arrive best optimal option.





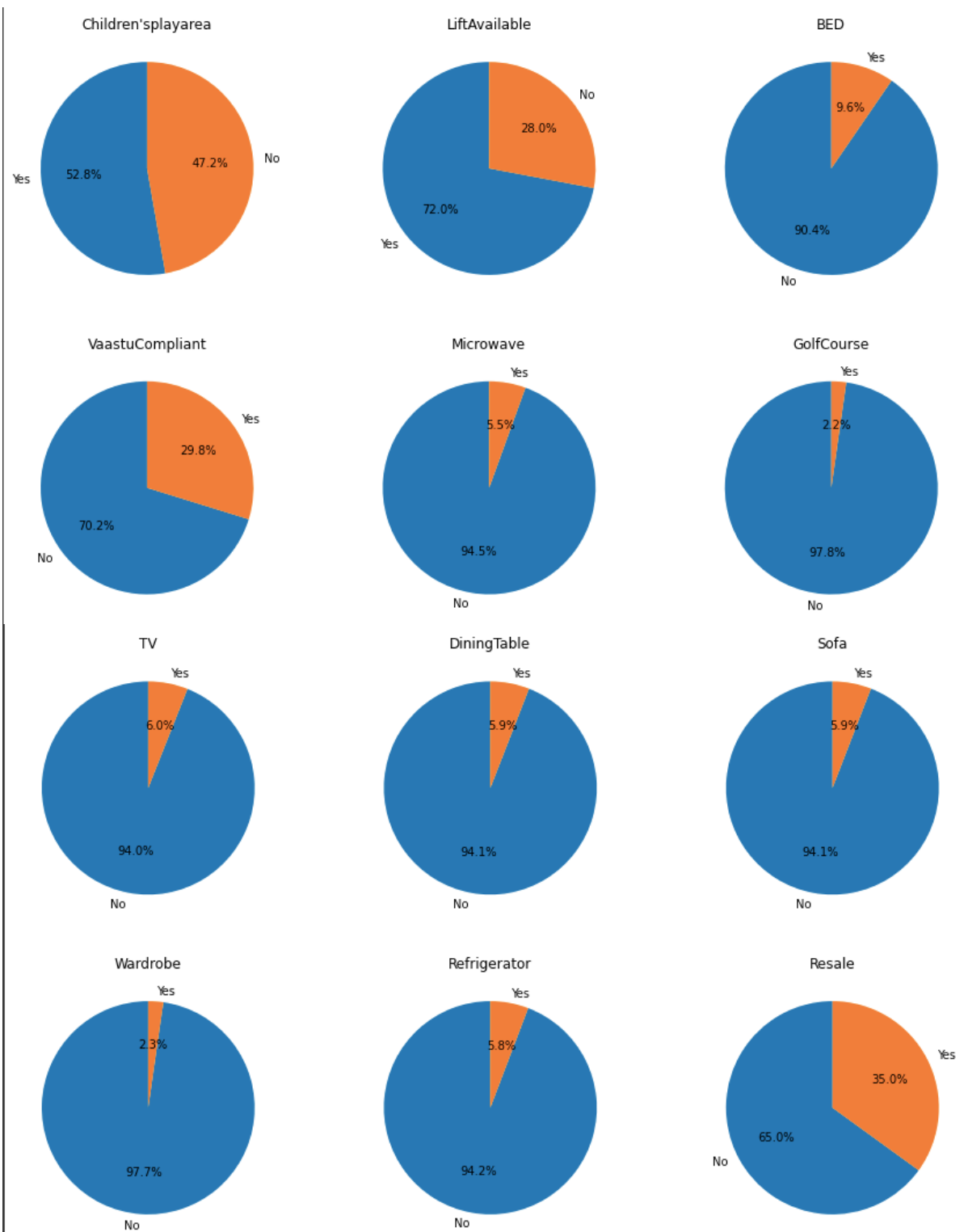


Fig 17: Other Binary Features

The shade of colour indicates the strength of the correlation between variables. Output float value between -1 to 1 where -1 -> very strong negative relation and +1 -> very strong positive relation.

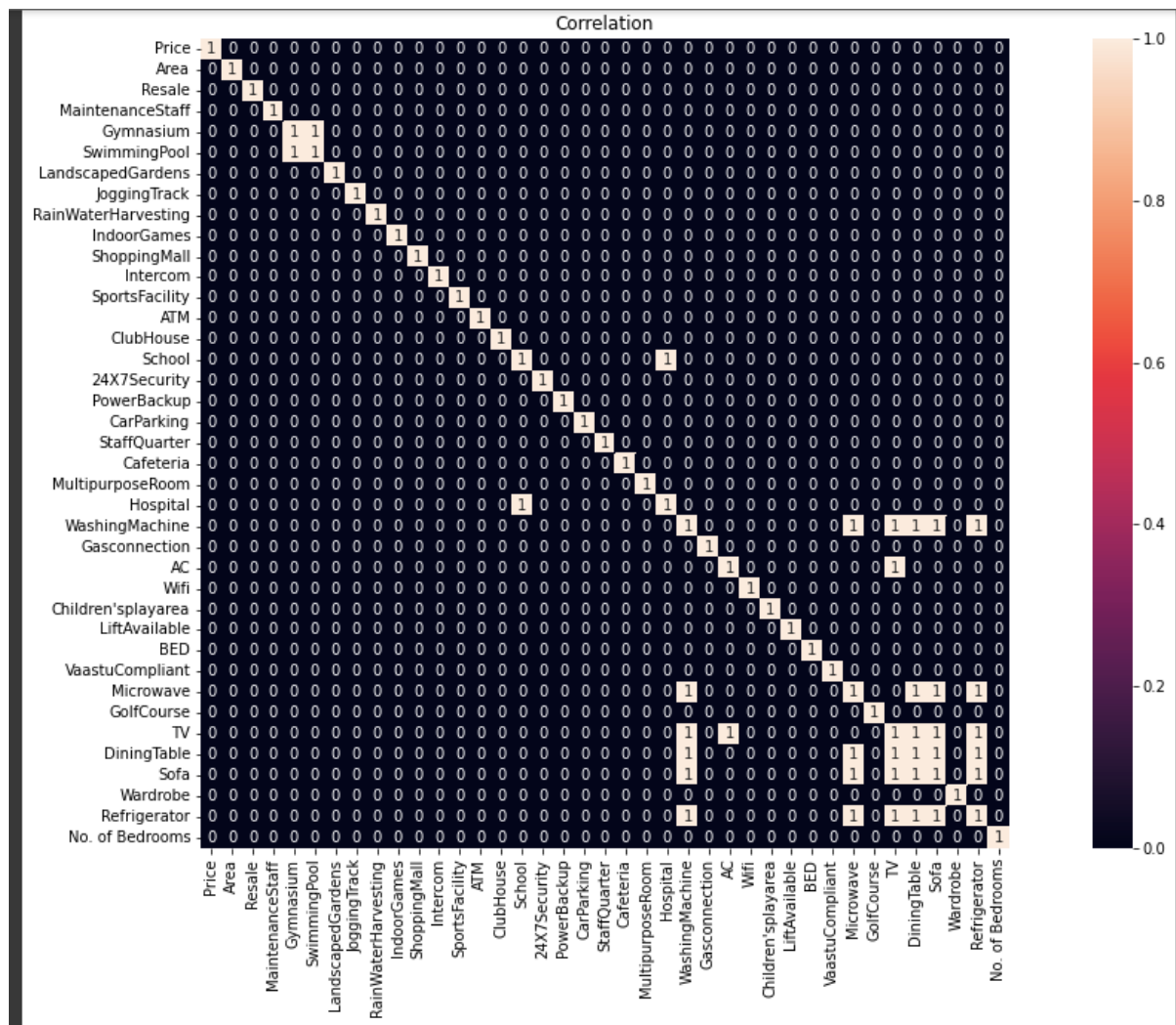


Fig 18: Correlation heatmap

In Fig. 19, the joint plot allows us to see that the price is directly proportional to the area. This means that if area increases simultaneously, the price will go up as well. The majority of Area values fall within the range of 0 to 3000, and the majority of Price values fall within the range of 0 to 1.

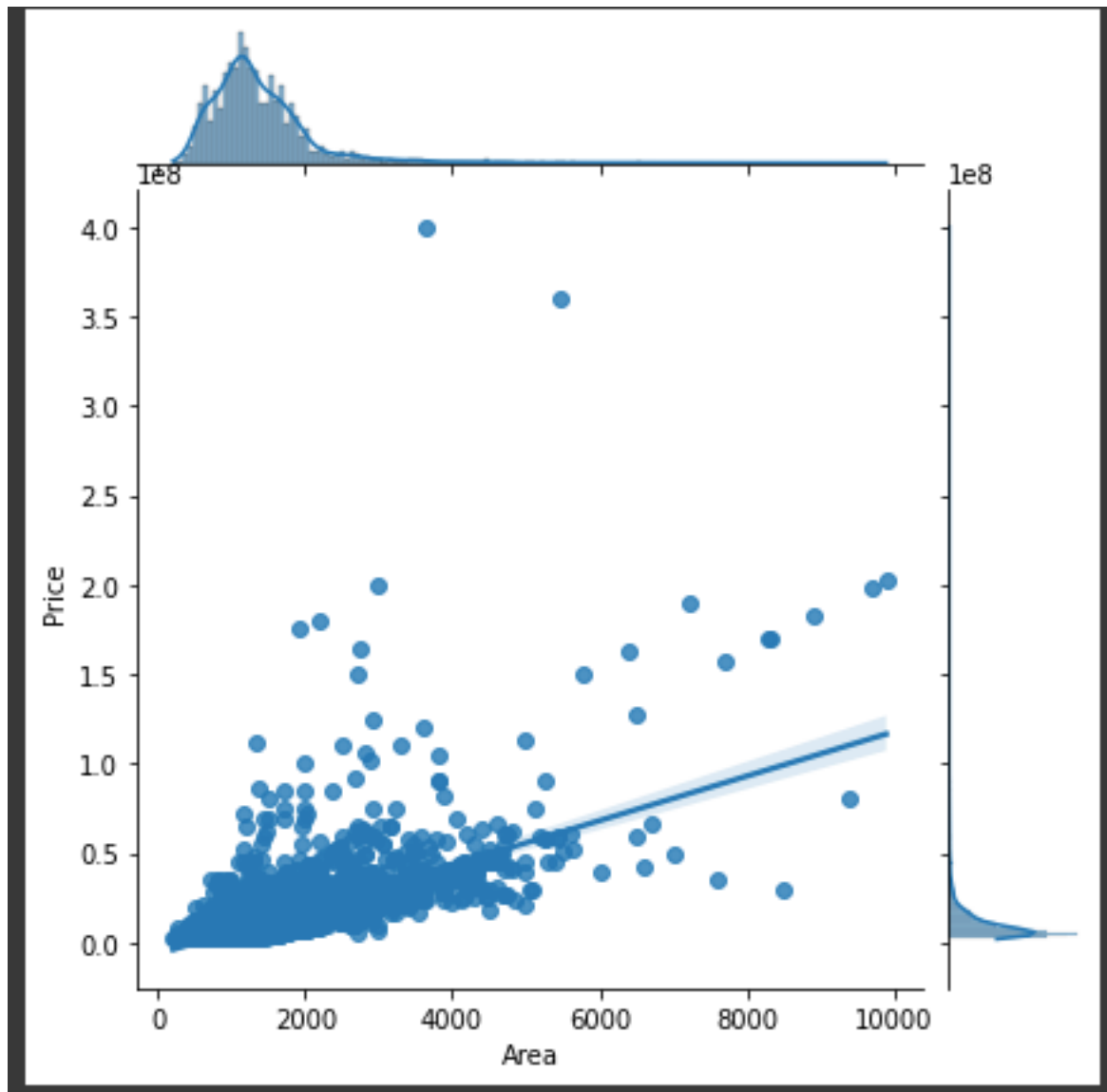


Fig 19: Joint Plot for Area & Price

As shown in Figure 20, the house with the most number of rooms—five—has the highest average price, followed by those with eight, six, and four bedrooms. It is clear from this information that a person who is interested in purchasing a home with more than three

bedrooms will have to pay out a greater amount of money.

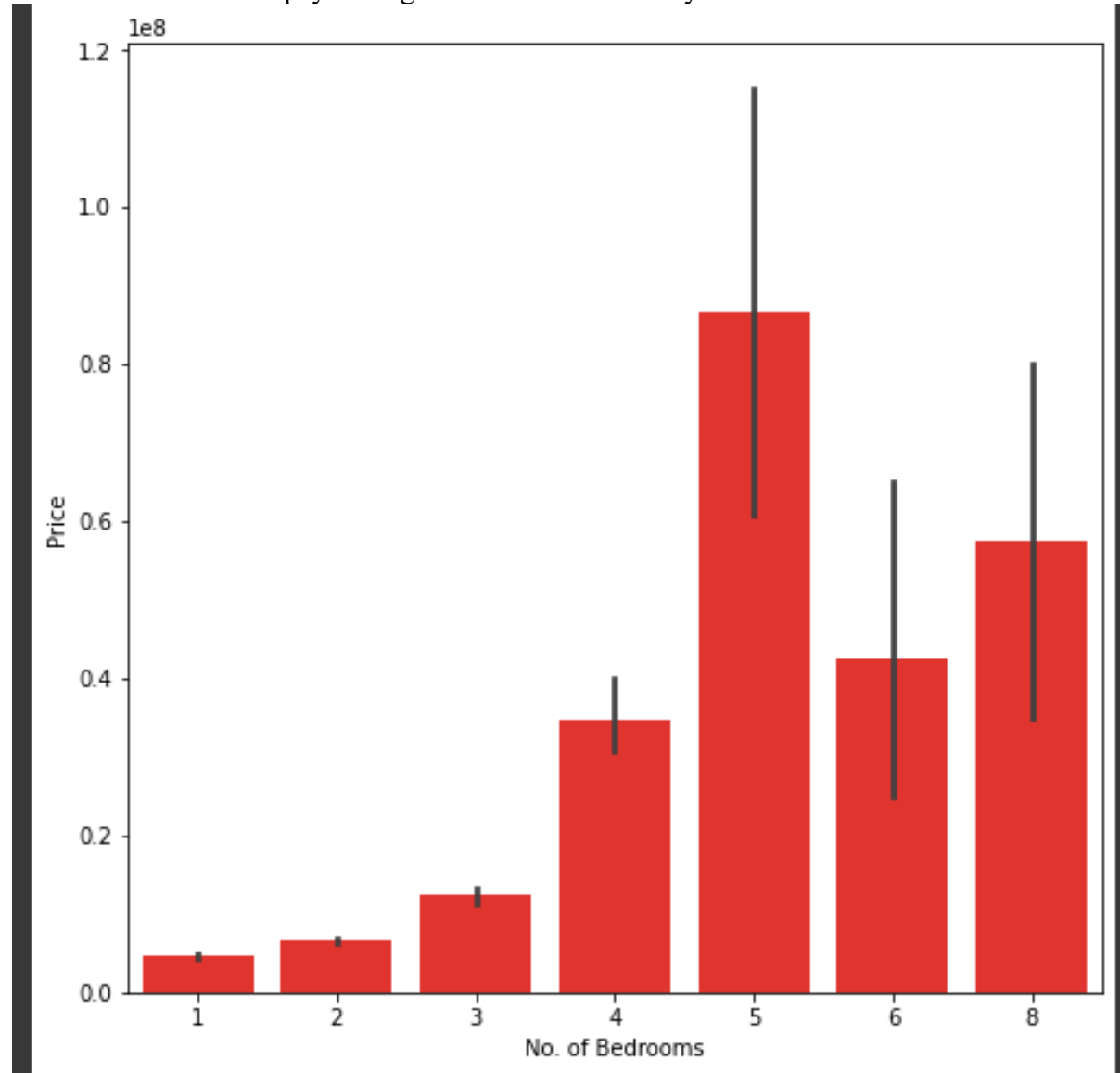


Fig 20: Comparison between Bedrooms & Price

5. Data Pre-processing:

5.1. Label Encoding(Ordinal Encoding Technique):

The categorical format of the Location feature can be seen in figure 21, which was created before Encoding. In the second stage of this process, we will change the values in the Location column to integers so that it can be used with our machine learning model. When it comes to supervised learning, this pre-processing stage for the structured dataset is quite crucial. (Team, 2022).

	Price	Area	Location	Resale	MaintenanceStaff	Gymnasium	SwimmingPool	LandscapedGardens	JoggingTrack	RainWaterHar
0	4850000	720	Kharghar	1	1.0	0.0	0.0	0.0	0.0	
1	4500000	600	Kharghar	1	1.0	1.0	1.0	0.0	1.0	
2	6700000	650	Kharghar	1	1.0	1.0	1.0	0.0	1.0	
3	4500000	650	Kharghar	1	1.0	0.0	0.0	1.0	0.0	
4	5000000	665	Kharghar	1	1.0	0.0	0.0	1.0	0.0	

5 rows x 40 columns

Fig 21: Before Encoding

After looking at figure 22, we are able to deduce that the category value in our Location column was successfully turned into an integer. This was accomplished via encoding. All of our data is now in a format that can be read by machines. On the basis of our pre-processed data, we are now prepared to carry out any model prediction.

```
# Converting the categorical value into numerical value
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
final_df['Location'] = le.fit_transform(final_df['Location'])
```

final_df.head()

	Price	Area	Location	Resale	MaintenanceStaff	Gymnasium	SwimmingPool	LandscapedGardens	JoggingTrack	RainWaterHar
0	4850000	720	325	1	1.0	0.0	0.0	0.0	0.0	
1	4500000	600	325	1	1.0	1.0	1.0	0.0	1.0	
2	6700000	650	325	1	1.0	1.0	1.0	0.0	1.0	
3	4500000	650	325	1	1.0	0.0	0.0	1.0	0.0	
4	5000000	665	325	1	1.0	0.0	0.0	1.0	0.0	

5 rows x 40 columns

+ Code + Text

final_df['Location'].tail()

```
29960    156
29961    482
29962    522
29963    538
29964    146
Name: Location, dtype: int64
```

Fig 22: After Encoding

5.2. Feature Scaling:

Now that our data has been pre-processed, it is time to standardise the independent features within the data range that has been fixed and scale our major features accordingly.

In the part of the article devoted to feature scaling, we are going to attempt to carry out procedures such as Standard scalar, Min-Max scalar, and without scaling. Based on this, we will attempt to determine which scaling strategy would be most beneficial to implement. And is able to provide the highest possible model score for our algorithms. (Hale, 2021).

5.2.1. Method 1 - Standard Scalar:

We have utilised the standard scalar technique for the purpose of scaling features, and at the moment, we are using standard scalar to characteristics such as "Area," "Location," and "No. of Bedrooms." The only continuous data we have is in these particular columns; the rest of the columns include categorical features, which don't need to be scaled because they are already in the form of 0 and 1. This is the rationale behind our decision to apply the scaling technique to those particular columns.

	Area	Location	No. of Bedrooms
0	-1.166708	-0.392359	-0.98408
1	-1.166708	-0.392359	-0.98408
2	-1.166708	-0.392359	-0.98408
3	-1.166708	-0.392359	-0.98408
4	-1.166708	-0.392359	-0.98408

Fig 23: Standard Scalar on these features

```
final_df_copy1.isnull().sum()
```

Price	0
Area	6606
Location	6606
Resale	0
MaintenanceStaff	0
Gymnasium	0
SwimmingPool	0
LandscapedGardens	0
JoggingTrack	0
RainWaterHarvesting	0
IndoorGames	0
ShoppingMall	0
Intercom	0
SportsFacility	0
ATM	0
ClubHouse	0
School	0
24X7Security	0
PowerBackup	0
CarParking	0
StaffQuarter	0
Cafeteria	0
MultipurposeRoom	0
Hospital	0

```

WashingMachine      0
Gasconnection        0
AC                  0
Wifi                 0
Children'splayarea   0
LiftAvailable        0
BED                  0
VaastuCompliant      0
Microwave            0
GolfCourse           0
TV                   0
DiningTable          0
Sofa                 0
Wardrobe             0
Refrigerator         0
No. of Bedrooms      6606
dtype: int64

```

Fig 24: Missing values after scaling

As you can see in Figure 24, we are receiving a large number of null values as a result of Feature scaling using Standard Scalar. There are times when the standard scalar translates numbers in such a range that our mean will become 0. When utilising a standard scaler, the number is divided by the standard deviation of its range; hence, when the standard deviation is close to 0, the code returns null results. Because of this, we have temporarily filled it in with 0 and will examine the result later.

5.2.2. Method 2 – Min-Max Scalar:

We used a Min-Max Scalar to adjust the values of the three key attributes of the dataset, which were "Area," "Location," and "Number of Bedrooms." in order to preserve the shape of the distribution as it was originally. The information that was originally included in the data has not been fundamentally changed. It doesn't minimise the significance of any outliers that are found. Minmax is a function that converts values in such a range so that our values are between 0 and 1. When we use the minmax scaler, we divide the number by its range's maximum and minimum by its range . So when we get this subtraction result equal 0, code will return null values.

The transformation is given by:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))  
X_scaled = X_std * (max - min) + min
```

where min, max = feature_range.

Fig 25: Min-Max Scalar Transformation

The following is a list of the outcomes that we have obtained for our features after scaling:-

	Area	Location	No. of Bedrooms
0	0.0	0.383255	0.0
1	0.0	0.383255	0.0
2	0.0	0.383255	0.0
3	0.0	0.383255	0.0
4	0.0	0.383255	0.0

Fig 26: After Scaling

Unfortunately, we were not expecting null values, but after scaling, we can see that there are a lot of null values present in it. This is what we are doing in Figure 27, where we are checking the null values after doing a min-max scaling. For the time being, we have those values set to 0 as a temporary measure. In order to prevent any manipulation of the data, and we will check the results.

```
final_df_copy1.fillna(0,inplace=True)
```

```

final_df_copy1.isnull().sum()
Price      0
Area      6606
Location   6606
Resale      0
MaintenanceStaff  0
Gymnasium  0
SwimmingPool  0
LandscapedGardens  0
JoggingTrack  0
RainWaterHarvesting  0
IndoorGames  0
ShoppingMall  0
Intercom    0
SportsFacility  0
ATM         0
ClubHouse   0
School      0
24X7Security  0
PowerBackup  0
CarParking   0
StaffQuarter  0
Cafeteria    0
MultipurposeRoom  0
Hospital     0
WashingMachine  0
Gasconnection  0
AC           0
Wifi         0
Children'splayarea  0
LiftAvailable  0
BED          0
VaastuCompliant  0
Microwave    0
GolfCourse    0
TV           0
DiningTable   0
Sofa          0
Wardrobe      0
Refrigerator  0
No. of Bedrooms  6606
dtype: int64

```

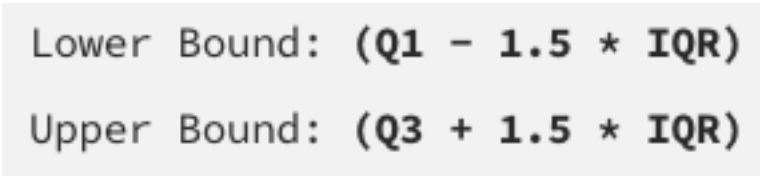
Fig 27: Checking Null values after Scaling

5.2.3. Method 3 - Without Feature Scaling:

After the data has been pre-processed, we have reached the point where we are attempting to obtain the final data in its raw form. The Standard scalar and Min-Max scalar approaches have already been attempted before we arrived at this stage. However, after conducting the scaling, we found that the dataset contained a large number of values that were absent. To get around that, we are going to try out this strategy, which involves not scaling the features and attempting to forecast the model score based on that..

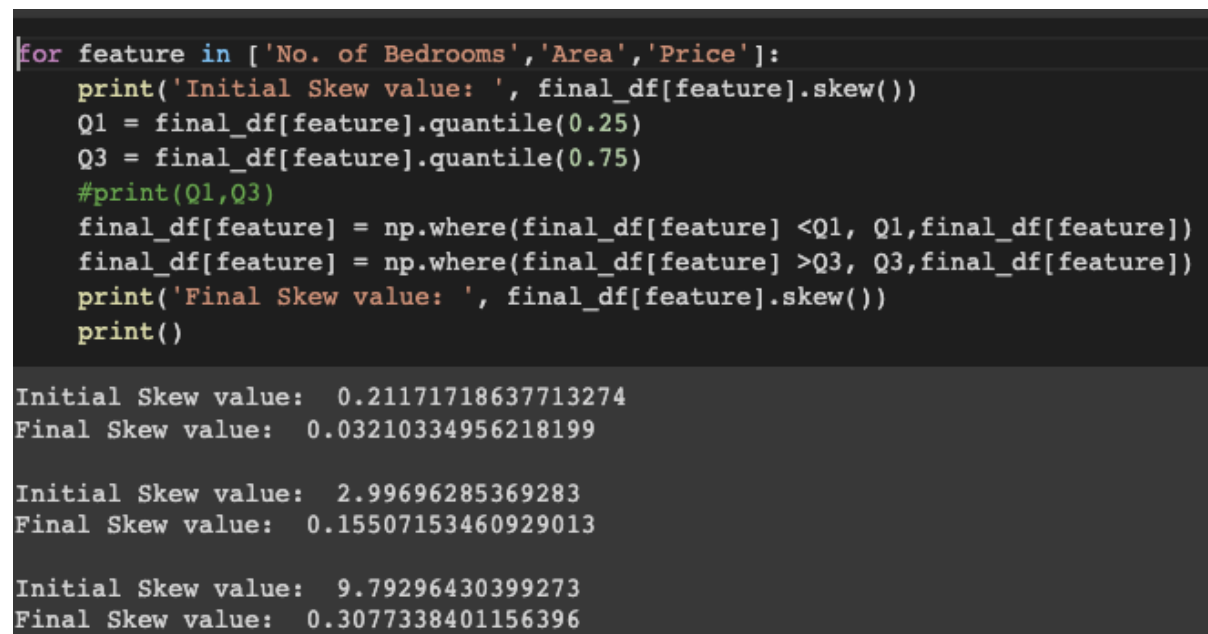
5.2.3.1. Outlier Treatment:

Outlier detection refers to the process of locating anomalous values within a given data set and then eliminating those values from further consideration. Any data point that lies outside of this range is considered to be an outlier, and appropriate action is taken in response to it as necessary. We define a new range, which we will call the decision range, so that we may detect outliers using this method. (Techopedia, 2017). The following is an example of each range:



Lower Bound: $(Q1 - 1.5 * IQR)$
Upper Bound: $(Q3 + 1.5 * IQR)$

Fig 28: Decision range



```
for feature in ['No. of Bedrooms', 'Area', 'Price']:
    print('Initial Skew value: ', final_df[feature].skew())
    Q1 = final_df[feature].quantile(0.25)
    Q3 = final_df[feature].quantile(0.75)
    #print(Q1,Q3)
    final_df[feature] = np.where(final_df[feature] < Q1, Q1, final_df[feature])
    final_df[feature] = np.where(final_df[feature] > Q3, Q3, final_df[feature])
    print('Final Skew value: ', final_df[feature].skew())
    print()

Initial Skew value:  0.21171718637713274
Final Skew value:  0.03210334956218199

Initial Skew value:  2.99696285369283
Final Skew value:  0.15507153460929013

Initial Skew value:  9.79296430399273
Final Skew value:  0.3077338401156396
```

Fig 29: Skewness of data

Consequently, the IQR technique is being utilised here as part of our Outlier Treatment. We are using three features that have outliers, and they are "Area," "Price," and "Number of Bedrooms." We are examining the initial skewness as well as the final skewness following

the implementation of the outlier treatment strategy utilising IQR. Figure 29 presents the findings for our inspection.

5.3. Feature Selection:

5.3.1. Procedure 1 - Feature Selection:

The purpose of the process known as feature selection is to carry out a procedure in which you choose, either automatically or manually, the features that contribute the most to your target variable. In a nutshell, the feature Importance score is used for performing Feature Selection.

We are able to get a clear picture of the top characteristics that are really vital for this particular stage by looking at figure 30. After all of this has been completed, you can then endeavour to determine how the model scored. After it has been utilised, how much of a difference does it make? (Prajwallandge, 2022).

In order to provide a clearer picture of how feature selection is carried out using random forest, it is helpful to understand that the characteristics that are chosen from the root node at the very top of the trees are, on average, more significant than the characteristics that are chosen from the end nodes of the trees, given that the top splits typically result in greater information gains.

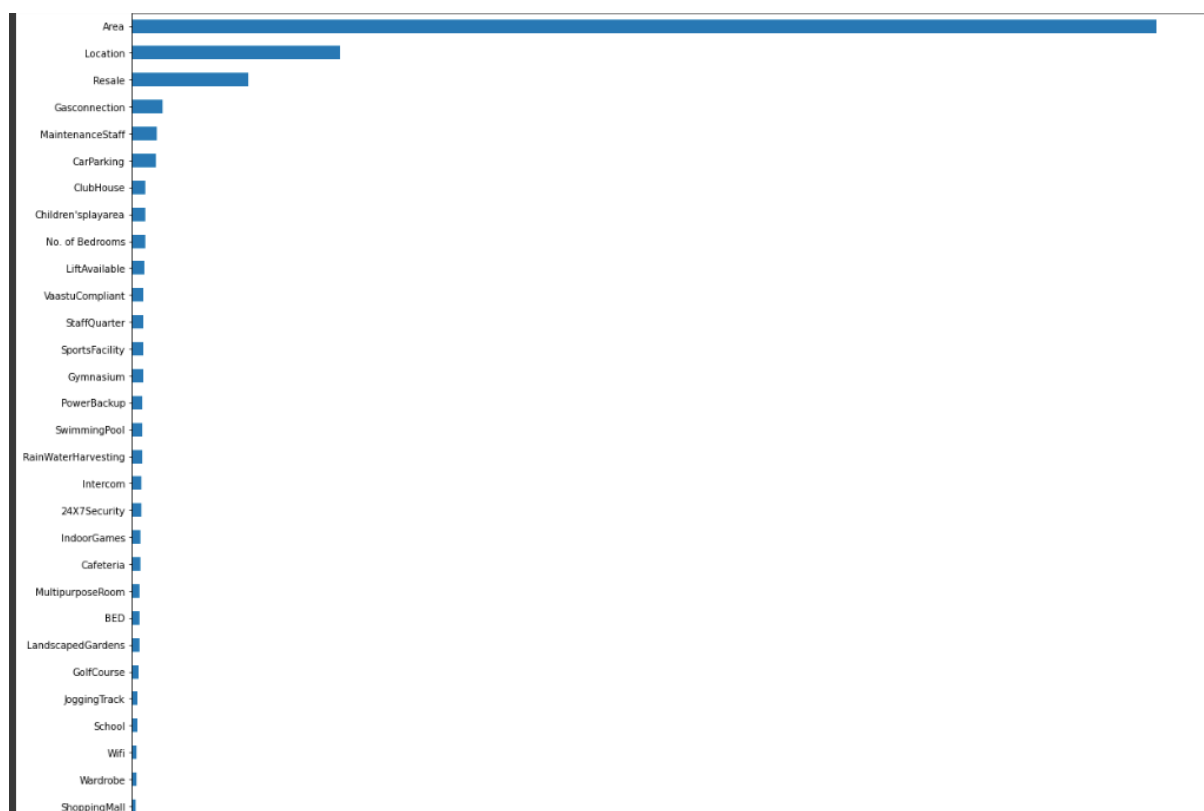


Fig 30: Linear Regression Model Score

Therefore, at this point, we are only taking into account the first three features from the dataset and attempting to determine its score. By carrying out the test on all of the models that had been doing well in the steps before this one.

5.3.2. Procedure 2 - Principal Component Analysis (PCA):

In the field of machine learning, principal component analysis (PCA) refers to an unsupervised, non-parametric statistical technique that is employed largely for dimensionality reduction.

A dataset is said to have high dimensionality if it contains a significant number of different features. The ability to generalise beyond the examples in the training set is limited when dealing with high-dimensionality, which is the primary issue that arises in the field of machine learning and is connected with the high-dimensionality problem. In 1961, Richard Bellman coined the term "Curse of Dimensionality" to describe this phenomena. According to Bellman, the problem occurs when "many algorithms that operate perfectly in low dimensions become intractable when the input is high-dimensional." (Goonewardana, 2019).

In addition, thus basically what PCA does is it minimises the dimensions (features) by developing new and enhanced features utilising existing features. This helps to make PCA more efficient.

5.3.3. Comparison of Principal Component Analysis with Feature Selection:

When we do feature selection, we choose n features from the set of existing features. When we do principal component analysis, we try to generate n new features that are based on the set of existing features.

6. Methodology & Implementation:

Train test split:

We are splitting the training & testing data from our original final dataset. To avoid getting overfitted data from the dataset. It evaluates how well machine learning algorithms perform when used to make predictions based on data that wasn't used to train the model. We are splitting training & testing size as 80% & 20% respectively as we can see in the figure.

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(final_df.drop('Price',axis=1),final_df['Price'],test_size=0.2)
```


6.1. Model Selection after Standard Scaling:

6.1.1.Linear Regression Model:

The Linear Regression model will serve as our initial option for addressing the regression issue. Following the application of this model, we have received the following score for both the training data and the testing data. In addition to this, we can investigate the Root mean squared error.

```
Rmse = 12432141.105134234
Score for training data = 0.11926266483927705
Score for testing data = 0.06440328340204271
```

Fig 31: Linear Regression Model Score

6.1.2.Random Forest Regressor:

Our second model is a Random Forest Regressor. Our previous model, which was a linear regression model, actually performed worse than our current model, which is a Random Forest Regressor. The score that our model receives on the training data set has greatly improved, but it does not perform well on the testing data.

```
Rmse = 12059991.201836523
Score for training data = 0.7762209027121242
Score for testing data= 0.11957815547045303
```

Fig 32: Random Forest Regressor Model Score

Hyper-parameter Tuning for Random Forest Regressor:

In this section from fig 33 & 34, we were attempting to make our model better by adjusting the values of its hyper-parameters. So far, for our Random forest regressor, we have experimented with both the Randomised search CV and the Grid search CV. After doing some hyper parameter adjusting, however, I came to the realisation that we were not receiving the results we wanted. After that, there is a minute reduction applied to the training score. Because there is a substantial gap between the train and test data, we can conclude that there is overfitting in this situation; hence, we will attempt to eliminate this overfitting and raise the test score through the use of hyperparameter tuning.

```
Score for training data = 0.5583031546977532
Score for testing data = 0.18601721864390974
```

Fig 33: Randomised search Cv model score

```
Score for training data = 0.5627375859140838
Score for testing data = 0.18472789281739355
```

Fig 34: Grid search Cv model score

6.1.3. Decision Tree Regressor:

We can see that our third model is a decision tree regressor by looking at fig. 35. The training score for this model is fairly high, but when we run it on testing data, we receive a negative score. This indicates that our data is becoming overfitted while we are running it on testing data.

```
Rmse = 14086043.506749557
Score for training data = 0.8522203098044902
Score for testing data = -0.2010881394880728
```

Fig 35: Decision Tree Regressor Model Score

6.1.4. XG Boost:

From figure 36, we can see that XG Boost is the very last model that we tested, but it produced results that were quite similar to those produced by Random Forest. The score is marginally better than the Random Forest Model.

```
Rmse = 11926087.80675359
Score for training data = 0.8107614347127179
Score for testing data = 0.13902045707594668
```

Fig 36: XG Boost Model Score

Conclusion for Method 1: Standard Scalar:

We have tried using models such as Linear Regression, Decision Tree, Random Forest Regressor, and XG Boost, but we have now come to the realisation that the results we have

gotten from using these models are not adequate. As a result, we are going to attempt using a different way.

6.2. Model Selection after Min-Max Scaling:

6.2.1.Linear Regression Model:

From fig 37, The Linear Regression model is the first one we try to solve our regression problem with. The following score was obtained for both the training data and the testing data after applying this model. Additionally, we may have a look at the Root mean squared error.

```
Rmse = 12405254.5366815
Score for training data= 0.12078181500683038
Score for testing data= 0.0684456738539082
```

Fig 37: Linear Regression Model Score

6.2.2.Random Forest Regressor:

From fig 38, The Random Forest Regressor is our second model, and it is really doing far better than the linear regression model that served as our prior model. The score that our model received has seen a huge increase for training data but relatively low for testing data.

```
Rmse = 11781144.626571976
Score for training data= 0.7818990308842649
Score for testing data= 0.1598210383303078
```

Fig 38: Random Forest Regressor Model Score

6.2.3.Decision Tree Regressor:

If we take a look at figure 39, we will notice that our third model is a decision tree regressor. The score that we get when we run this model on testing data is negative, despite the fact that the training score for it is rather high. This suggests that our data are becoming overfitted as we continue to run it on testing data.

```
Rmse = 14764796.38192805
Score for training data= 0.8522203098044902
Score for testing data= -0.3196286983510006
```

Fig 39: Decision Tree Regressor Model Score

6.2.4.XG Boost:

Figure 40, also known as XG Boost, is the fourth model that we have attempted, but it produces results that are almost identical to those produced by Random Forest. When compared to the Random Forest Model, this model has a little lower score for the testing data and a slightly better score for the training data.

```
Rmse = 12255164.629614087
Score for training data= 0.8245783775285821
Score for testing data= 0.09085087104220424
```

Fig 40: XG Boost Model Score

6.2.5.KNN Regressor:

From figure 41, we can see that our final model is the KNN Regressor, which is doing fairly admirably but is not performing better than the Random Forest Regressor and the XG Boost. A simple method known as K-nearest neighbours can accurately forecast the numerical target by storing all of the available cases and use a similarity measure to calculate the average numerical target of the K nearest neighbours.

```
Rmse = 12526967.681616595
Score for training data= 0.4494287646911901
Score for testing data= 0.050076259908785725
```

Fig 41: KNN Regressor Model Score

Conclusion for Method 2: Min-Max Scalar:

We have tried various models such as Linear Regression, Decision Trees, Random Forest Regressors, KNN Regressors, and XG Boost, but we have finally come to the realisation that none of these models produce results that are satisfactory for us. Because of this, we are going to try a different method that does not involve the scaling technique. And after that, we'll attempt to compare our results.

6.3. *Model Selection without Feature Scaling:*

Linear Regression Model:

The Linear Regression model is going to be the first one we try to solve our regression problem with. Following the application of this model, we have obtained the following score for both the training data and the testing data. In addition to that, we can investigate the Root

mean squared error. In this case, the results that our model produces are superior to those obtained using scaled approaches. We are computing the R2 score for train and test here.

```
Rmse = 9917019.022072533
Score for training data = 0.5169826815331625
Score for testing data = 0.40466814680969876
```

Fig 42: Linear Regression Model Score

Random Forest Regressor:

Our second model, Random Forest Regressor, really performs better than our first model, the linear regression model. Our model's score has greatly increased. From figure 43, we can conclude that our model's performance is superior to that of scaled approaches. The training score has skyrocketed and is yielding positive outcomes, but our testing data has increased slightly.

```
Rmse = 9850559.37352323
Score for training data= 0.9441837781503956
Score for testing data= 0.41262073213918316
```

Fig 43: Random Forest Regressor Model Score

Decision Tree Regressor:

Our third model is the Decision tree regressor, and we can see in figure 44 that the training data is providing a high quality score. However, when we use the testing data, we receive a very low score, which indicates that this model is providing us with less accurate results for the testing data.

```
Rmse = 11909773.490950577
Score for training data= 0.9996729095242173
Score for testing data= 0.14137440332618967
```

Fig 44: Decision Tree Regressor Model Score

XG Boost:

The XG Boost is the fourth model that we have examined and evaluated. Due to the fact that the data in this model has been overfit, we are achieving remarkable success with our training data, but we are only creating disappointing outcomes with our testing data. Let's try something different in order to improve the results of the test data we're doing.

```
Rmse = 10552329.340834478
Score for training data = 0.9900729770810718
Score for testing data= 0.32594784436467616
```

Fig 45: XG Boost Model Score

KNN Regressor:

Our last model is the KNN Regressor, which can be visualised in figure 46. This model is performing fairly admirably, outperforming the Linear Regressor model, but it is not performing as well as the Random Forest Regressor and XG Boost approach.

```
Rmse = 10770181.693149934
Score for training data = 0.6431392595604211
Score for testing data= 0.2978290025191822
```

Fig 46: KNN Regressor Model Score

Conclusion for Method 3: Without Scaling:

We have tried various models, such as Linear Regression, Decision Tree, Random Forest Regressor, KNN Regressor, and XG Boost; however, we have come to the conclusion that the results we are getting for the training data are less satisfactory; however, there is the possibility that we could try another method that would give us better results. Let's give the Outlier Treatment a shot to see if it helps improve the accuracy of our model.

6.3.1. Model Selection after Outlier Treatment:

Linear Regression Model:

After applying this model to the Linear Regression problem, we are getting the following score for both the training data and the testing data. In addition to this, we can investigate the Root mean squared error. Let's have a look at the figure in fig 47. Now that we've eliminated the outliers, our model is functioning even more effectively. There was an increase in score for both the training data and the testing data.

```
Rmse = 1920268.2512013707
Score = 0.665938533744456
Score for testing data= 0.6458458979399884
```

Fig 47: Linear Regression Model Score

XG Boost:

We are achieving one of the best model scores possible from this model following the outlier treatment stage in the XG Boost algorithm, as seen in the figure 48. After looking at a few different models, it appears that this one has the most accurate predictions.

```
Rmse = 1314982.5637677019  
Score = 0.9705319698026492  
Score for testing data= 0.8339233936124196
```

Fig 48: XG Boost Model Score

Hyperparameter Tuning using Grid Search for XG Boost:

We have therefore attempted Grid search Cv for our Random forest regressor. After extensive parameter adjustment, however, I discovered that we are not achieving the desired outcome. After that, the score decreases marginally.

```
Rmse = 1506864.4029774985  
Score = 0.9388815363028106  
Score for testing data= 0.7819195069034409
```

Fig 49: Grid Search XG Boost Model Score

Light Gradient Boost Regressor:

When we look at the figure, we can see that training and testing score is producing quite decent results in terms of boosting methods, but these results are a little lower when compared to those produced by XG Boost. However, that is a considerable increase.

```
Rmse = 1413334.0507406997  
Score = 0.8680835401279031  
Score for testing data= 0.8081516280847196
```

Fig 50: Light Gradient Boost Regressor Model Score

Decision Tree Regressor:

According to the figure 51, in Decision Tree Regressor, we have obtained a solid model score for our training data and a testing score that is fairly well.

```
Rmse = 1757345.1527588498  
Score = 0.9928812635172424  
Score for testing data= 0.7033921717535321
```

Fig 51: Decision Tree Regressor Model Score

Hyperparameter Tuning for RandomizedSearchCV & GridSearchCV:

In this section, we performed hyper-parameter tuning in an effort to improve our model. We have therefore attempted both Randomized search CV and Grid search CV for our Decision Tree regressor. However, I discovered that despite hyper parameter adjustment, we are achieving the desired outcome. But, the score is dropped for both training & testing data. As we can visualize fig 52 & 53.

```
Score for training data = 0.7536374339861895  
Score for testing data= 0.661321722309238
```

Fig 52: RandomizedSearchCV Decision Tree Regressor Model Score

```
Score for training data = 0.7543364215232766  
Score for testing data= 0.6460230865610357
```

Fig 53: GridSearchCV Decision Tree Regressor Model Score

Random Forest Regressor:

As can be seen in the figure 54, the Random Forest Regressor that we are using has given us a strong model score for both the training data and the testing data. We are able to report that this model is performing at the second best score we have seen up to this point, behind only XG Boost.

```
Rmse = 1383005.299555655  
Score = 0.9661940872452536  
Score for testing data= 0.8162970367016316
```

Fig 54: Random Forest Regressor Model Score

Hyperparameter Tuning for RandomizedSearchCV & GridSearchCV:

For our Random forest regressor, we have therefore explored both Randomised search CV and Grid search CV. Then I realised that hyper parameter adjustment was yielding the desired results. After that, the score is somewhat increased. (Brownlee, 2020).

```
Score for training data = 0.9684188810224647  
Score for testing data = 0.8190024513888835
```

Fig 55: RandomizedSearchCV Random Forest Regressor Model Score

```
Score for training data = 0.9684188810224647  
Score for testing data = 0.8190024513888835
```

Fig 56: GridSearchCV Random Forest Regressor Model Score

Conclusion after Outlier Treatment:

After the implementation of this stage of the outlier therapy, we were obtaining fantastic outcomes for our models. However, we will make an effort to produce the same score while reducing the number of columns in order to escape the "curse of dimensionality." In order to do this, we will look into methods such as feature selection and principal component analysis (PCA) in order to cut down on the number of features and increase the score that our model is capable of achieving.

6.4.1. Model Selection after Feature Selection:

6.4.1.1. Gradient Boosting Regressor:

After removing the majority of the features from our dataset, we can see that the gradient boosting regressor is still providing us with a relatively excellent score for both our training data and our testing data, as seen in the figure 57.

```
Rmse = 1729945.469480023  
Score = 0.7378638771296674  
Score for testing data= 0.7125692018919628
```

Fig 57: Gradient Boosting Regression Model Score

6.4.1.2. XG Boost Regressor:

We can conclude, based on the figure 58, that our model is still functioning extremely well and providing a good accuracy for our model, despite the fact that we have selected only the first three key features by applying feature importance.

```
Rmse = 1444500.1087617571  
Score = 0.9276340755874699  
Score for testing data= 0.7995972710757409
```

Fig 58: XG Boost Model Score

6.4.1.3. Random Forest:

After using feature importance technique, it is clear from the figure 59 that the scores that our model receives are satisfactory after selecting only few features.

```
Rmse = 1589475.8032075388  
Score for training data= 0.9398632575775988  
Score for testing data= 0.757344450625438
```

Fig 59: Random Forest Regressor Model Score

6.4.1.4. Decision Tree:

Based on Figure 60, we have obtained a very high score for training data and a slightly lower score for testing data, but this model's overall performance is not too terrible following the application of the feature importance technique.

```
Rmse = 1899168.1985219948  
Score for training data= 0.9685981885815743  
Score for testing data= 0.6535749847960374
```

Fig 60: Decision Tree Model Score

6.4.2. Model Selection after PCA:

6.4.2.1. Random Forest:

After using principal components analysis (PCA), it is clear from the figure 61 that the scores that our model receives are satisfactory.

```
Score = 0.9578449723248182
Score for testing data= 0.7435853272804633
```

Fig 61: Random Forest Model Score

6.4.2.2. Decision Tree:

Based on Figure 62, we have obtained a very high score for training data but a lower score for testing data, but overall this model is not performing too poorly.

```
Score = 0.9928812635172424
Score for testing data= 0.6016752774400254
```

Fig 62: Decision Tree Model Score

6.4.2.3. LGBM:

Figure 63 indicates that the Light Gradient Boosting Method achieves sufficient accuracy for our model. The training data accounts for 79% of the model's score, while the testing data accounts for 70%.

```
Rmse = 1746964.6897699775
Score = 0.7905859523698024
Score for testing data= 0.7068858885364779
```

Fig 63: LGBM Model Score

6.4.2.4. Gradient Boost Regressor:

After performing principal component analysis (PCA), both the training and testing data scores for the Gradient Boosting Regressor decreased somewhat. In addition, the RMSE value has increased. Consequently, we can conclude that feature importance technique is more effective than PCA for gradient boosting regressors.

```
Rmse = 1886799.5922913882
Score = 0.6858074441392678
Score for testing data= 0.6580835302427044
```

Fig 64: Gradient Boost Regressor Model Score

6.4.2.5. XG Boost:

The XG Boost model received the following rating: 93% of the model's score is determined by the training data, whereas 75% is determined by the testing data. This model has thus far been pretty satisfactory. And RMSE is in a rather advantageous position as compared to the competitors.

```
Rmse = 1597890.0199384696  
Score = 0.9306173279781661  
Score for testing data= 0.7547764123855396
```

Fig 65: XG Boost Model Score

6.4.2.6. KNN:

The model performs exceptionally well in KNN, but we have numerous models that perform considerably better. It has a high training score but a below-average testing score.

```
Rmse = 1860636.2821176418  
Score = 0.7826472580739624  
Score for testing data= 0.6675001573568452
```

Fig 66: KNN Model Score

6.5. Neural Network (ANN):

Let's get started on solving the regression problem by developing a basic neural network model. Beginning with all of the libraries and functions that are required. (Mir, 2020).

```
# Creating a Neural Network Model  
from tensorflow import keras  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense, Activation  
from tensorflow.keras.optimizers import Adam
```

Fig 67: Creating Neural Network model

Given that we have forty features, let's begin by include forty neurons, followed by four hidden layers and one output layer so that we can forecast house price.

Additionally, the ADAM optimization technique is utilised in order to optimise the loss function (Mean squared error)

```
# having 40 neuron is based on the number of available features
model = Sequential()
model.add(Dense(40,activation='relu'))
model.add(Dense(40,activation='relu'))
model.add(Dense(40,activation='relu'))
model.add(Dense(40,activation='relu'))
model.add(Dense(1))
model.compile(optimizer='Adam',loss='mse')
```

Fig 68: Adam Optimization Function

After that, we train the model for a total of 400 epochs, and each time we do so, we record the accuracy of the training as well as the validation in the history object. In order to keep track on how well the model is performing for each epoch, the model will run in both train data and test data while simultaneously calculating the loss function. This will allow for accurate tracking of the model's performance.

```
model.fit(x=X_train,y=y_train,
          validation_data=(X_test,y_test),
          batch_size=128,epochs=400)
model.summary()
```

Fig 69: Model Fitting

Loss Function:

The line graph for The following is a representation of the line graph for train and validation loss:

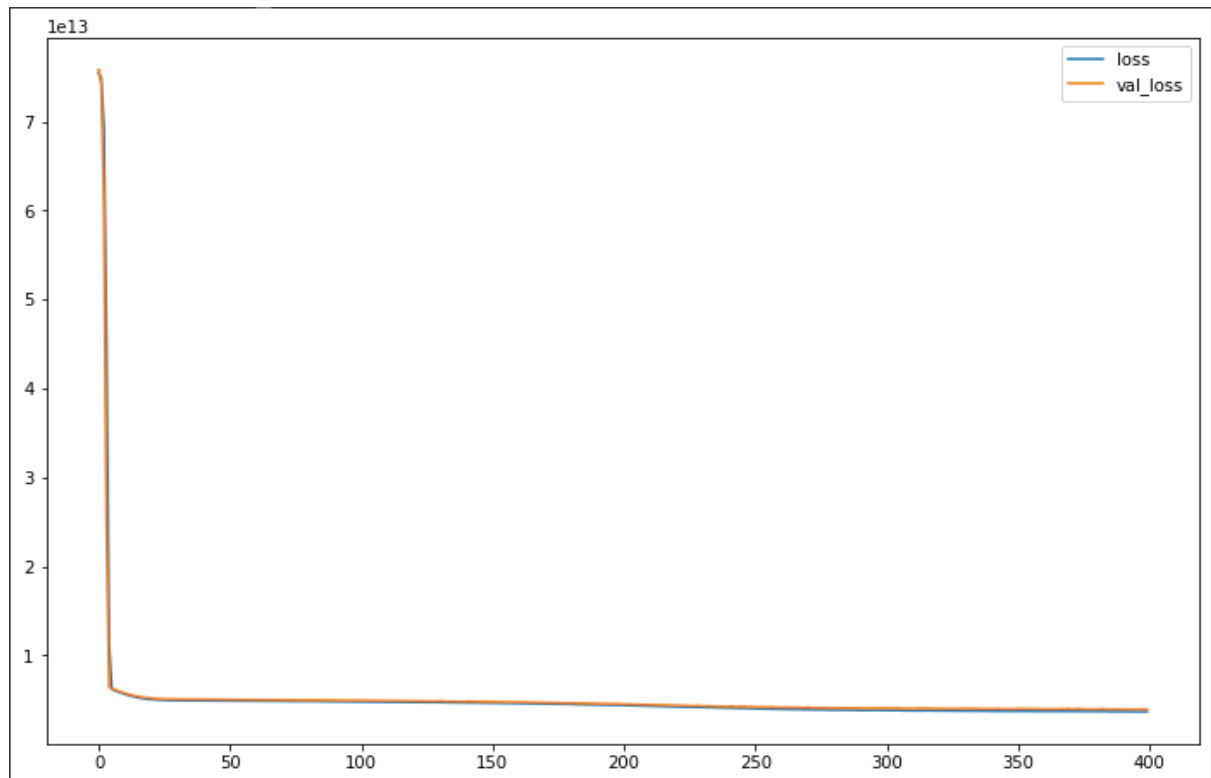


Fig 70: Loss Function

From fig 70, our model learns well in the first 50 iterations and suffers minimal loss in subsequent iterations.

Visualizing residuals:

At start & end we get very less loss and in middle our loss function is high.

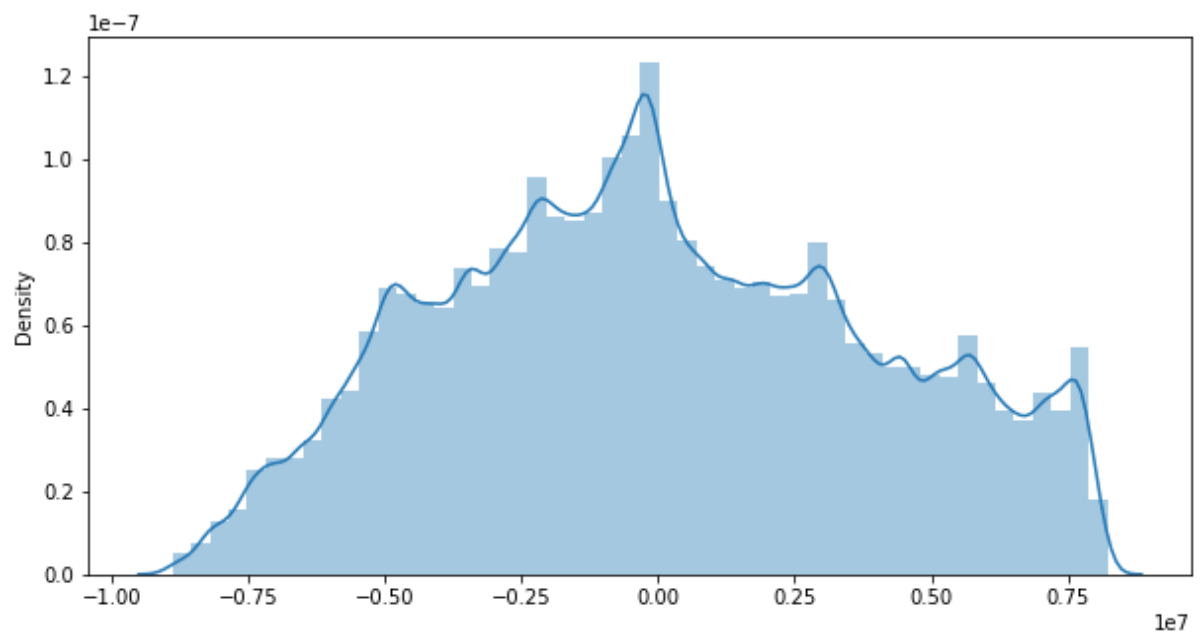


Fig 71: Residual visualization

7.Results:

7.1. Comparison between all the Regression models & methods:

	Train	Test
RF	0.785000	0.129229
DTR	0.852220	-0.050915
XGBR	0.810761	0.139020
Linear Reg	0.119263	0.064403

Fig 72: Model Score after Standard Scalar

These are the findings that we receive when we run the dataset using procedure 1, which is the standard scalar method. This is important to notice. When it comes to the training data, Random Forest, Decision Tree, and XG Boost generate relatively decent model scores; however, when it comes to the testing data, we are obtaining poor model scores. In addition, the linear regression model does not do particularly well in any of the data.

From fig 73, Using method 2, which involves applying the Min-Max scalar and checking the model score of the result. It is plain to see from the score that we have obtained terrible results for our testing data using this method as well, just like we did when we used the standard scalar method. In addition, we are unable to apply either of the scaling methods on this dataset. Now that we have this opportunity, we can also examine whether without scaling method gives us satisfactory results in later sections.

	Train	Test
RF	0.793748	0.169426
DTR	0.852220	-0.137243
XGBR	0.824578	0.090851
KNN	0.449429	0.050076
Linear Reg	0.120782	0.068446

Fig 73: Model Score after Min-Max Scalar

From fig 74, When we use method 3, which does not involve scaling, we are able to observe improvement in the model's score. Without making use of the feature scaling technique, each of our models is producing scores that are quite better. When compared against the Min-Max scalar and Standard scalar techniques, this strategy comes out on top as the more practical option to pursue.

	Train	Test
RF	0.944184	0.412621
DTR	0.999673	0.141374
XGBR	0.990073	0.325948
GBR	0.808598	0.442374
LXGB	0.824797	0.485118
KNN	0.643139	0.297829

Fig 74: Model Score without feature scaling

From here onwards we will only focusing on method 3 (which is without scaling technique). Now, after this I have done outlier treatment for the data. From fig 75, we can see our score after outlier treatment is up to the mark. And efficiency for scores has been maximized.

	Train	Test
RF	0.966260	0.815892
DTR	0.992881	0.712504
XGBR	0.969683	0.837159
GBR	0.764682	0.738295
LXGB	0.868082	0.808149
KNN	0.784994	0.673715

Fig 75: Model Score without feature scaling after outlier treatment

	Train	Test
RF	-0.090604	-0.091596
DTR	-0.514162	-0.490803
XGBR	0.928146	0.802690
GBR	0.737860	0.712566
LXGB	0.815342	0.763378
KNN	0.779887	0.670415

Fig 76: Model Score after feature selection

After that outlier treatment we were getting our highest possible result for our data. But then we tried feature selection technique to reduce the features and maximize the model score. We have just chose top 3 features and checking the model score. But we were acquiring good performance from the model. As we can see in the fig 76.

	Train	Test
RF	0.957964	0.750707
DTR	0.992881	0.592006
XGBR	0.933563	0.753619
GBR	0.685802	0.658083
LXGB	0.790582	0.706883
KNN	0.782641	0.667486

Fig 77: Model Score after PCA

From fig 77, we can see model score after PCA has been imposed (which is our procedure 2). This method also gives almost the same result like feature selection. By using this method we can optimize the computational speed & the curse of dimensionality.

7.2. Neural Network (ANN):

Evaluation on Test Data:

From fig 78, we can evaluate that our Neural network (ANN) model is giving us following scores for our testing data, only using 4 hidden layers, we are getting almost 62% accuracy for our R2 score & Var score.

```
MAE: 1520077.0786755772
MSE: 3930815624146.356
RMSE: 1982628.4634662028
VarScore: 0.6225269212267908
R2 Score: 0.6224702791314156
```

Fig 78: Evaluation on test data

7.3. Regression Model vs Neural Network (Comparison):

The comparison between Regression Model and Neural Network is as follows:-

- When compared to the performance of the ANN model, the regression model has a better overall performance.
- The R2 score and RMSE values that the regression model generates are higher than those that the ANN model generates.

8. Analysis:

- During our feature scaling part, we have tried several techniques to get good scaled data but unfortunately after performing feature scaling we were getting a lot of null values, which were affecting our models. So we chose without scaling technique to move forward which actually worked quite stupendous and maximized the outcome.
- While doing feature selection, we have just took top 3 features for our model score, but still by choosing only 3 features we were getting highest possible score for our data.
- While selecting the dataset for this project, we had a lot of datasets which were having lot of missing & misleading values. We have dealt it either by correcting the data during the code implementation or by directly removing those rows from our datasets.
- Also we can say that performing outlier treatment on data can give significant boost to our scores.
- We have used hyperparameter tuning so from that we come to the conclusion that is not helping us to increase our scores or reduce overfitting issue as well.

9. Conclusion & Future Work:

As a conclusion we have built several regression models and ANN to predict the house price given some of the house features. Each model was analysed and compared to find which one had the highest performance. In this project, we applied the data science methodology, starting with data collection, cleaning, and pre-processing, then exploring the data and developing models, assessing the results, and communicating them with visualizations. At last, for the regression model we can state that our XG Boost regressor & Random forest regressor model have worked more efficiently than the other models.

However, for future work, we can improve our price prediction of the house, we can incorporate time-related data for this dataset, in my opinion, since it can provide more precise results. We could accumulate data that has been web scraped from somewhere into our time-related functionality. Additionally, we can upload all of those machine learning algorithms to the cloud, allowing anyone to view any information from any location using the website that we will continue to develop and maintain in the future.

Due to restricted computational power, we are using fewer hidden layers in our ANN model; however, we can add more hidden layers in the future to improve the accuracy of our neural network.

10. References:

1. AWS (1978) What is Boosting, Amazon. The University. Available at: <https://aws.amazon.com/what-is/boosting/> (Accessed: November 20, 2022).
2. Babu, A. and Chandran, A.S. (2019) "International Journal of Computer Science and Mobile Applications," Literature Review on Real Estate Value Prediction Using Machine Learning, pp. 8–15. Available at: <https://doi.org/10.47760/ijcsma>.
3. Biswal, A. (2022) Principal Component Analysis in machine learning: Simplilearn, Simplilearn.com. Simplilearn. Available at: <https://www.simplilearn.com/tutorials/machine-learning-tutorial/principal-component-analysis#:~:text=The%20Principal%20Component%20Analysis%20is,plotting%20in%202D%20and%203D>. (Accessed: November 3, 2022).
4. Brownlee, J. (2020) How to use StandardScaler and MinMaxScaler transforms in Python, MachineLearningMastery.com. Available at: <https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/> (Accessed: September 30, 2022).
5. Brownlee, J. (2020) Hyperparameter optimization with random search and grid search, MachineLearningMastery.com. Available at: <https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/> (Accessed: October 30, 2022).
6. Brownlee, J. (2019) Difference between classification and regression in machine learning, MachineLearningMastery.com. Available at: <https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/> (Accessed: October 22, 2022).
7. Chaudhary, S. (2020) Why "1.5" in IQR method of outlier detection?, Medium. Towards Data Science. Available at: <https://towardsdatascience.com/why-1-5-in-iqr-method-of-outlier-detection-5d07fdc82097> (Accessed: October 20, 2022).
8. Chordia, M.P. et al. (2022) "Prediction of house price using machine learning," International Journal for Research in Applied Science and Engineering Technology, 10(2), pp. 1387–1390. Available at: <https://doi.org/10.22214/ijraset.2022.40466>.
9. Cloud Education, I.B.M. (2021) What is boosting?, IBM. Available at: <https://www.ibm.com/cloud/learn/boosting> (Accessed: November 20, 2022).
10. Deaky, B.-A. and Parv, L. (2017) "Virtual reality for real estate: Its evolution in Bluemind software," 2017 4th Experiment@International Conference (exp.at'17) [Preprint]. Available at: <https://doi.org/10.1109/expat.2017.7984408>.
11. Ekman, P., Raggio, R.D. and Thompson, S. (2016) "Developing smart commercial real estate: Technology-based self-service (TBSS) in commercial real estate facilities," 2016 IEEE International Smart Cities Conference (ISC2), pp. 1–6. Available at: <https://doi.org/10.1109/isc2.2016.7580744>.
12. Goonewardana, H. (2019) PCA: Application in machine learning, Medium. Apprentice Journal. Available at: <https://medium.com/apprentice-journal/pca-application-in-machine-learning-4827c07a61db> (Accessed: November 5, 2022).

13. Hale, J. (2021) Scale, standardize, or normalize with Scikit-Learn, Medium. Towards Data Science. Available at: <https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02> (Accessed: October 4, 2022).
14. Jain, M., Rajput, H. and Garg, N. (no date) Prediction of House pricing using Machine Learning with python | IEEE ..., Prediction of House Pricing Using Machine Learning with Python. Available at: <https://ieeexplore.ieee.org/document/9155839> (Accessed: September 15, 2022).
15. Jordan, J. (2018) Hyperparameter tuning for machine learning models., Jeremy Jordan. Jeremy Jordan. Available at: <https://www.jeremyjordan.me/hyperparameter-tuning/> (Accessed: October 31, 2022).
16. Kumkar, P. et al. (2018) "Comparison of ensemble methods for real estate appraisal," 2018 3rd International Conference on Inventive Computation Technologies (ICICT) [Preprint]. Available at: <https://doi.org/10.1109/icict43934.2018.9034449>.
17. Lateef, Z. (2022) A beginners guide to Boosting Machine Learning Algorithms, Edureka. Available at: <https://www.edureka.co/blog/boosting-machine-learning/> (Accessed: November 22, 2022).
18. Mir, M. (2020) House prices prediction using Deep Learning, Medium. Towards Data Science. Available at: <https://towardsdatascience.com/house-prices-prediction-using-deep-learning-dea265cc3154> (Accessed: November 29, 2022).
19. Muhajir, I. (2020) K-neighbors regression analysis in Python, Medium. Analytics Vidhya. Available at: <https://medium.com/analytics-vidhya/k-neighbors-regression-analysis-in-python-61532d56d8e4> (Accessed: October 25, 2022).
20. Prajwallandge (2022) Gradient boosting ,PCA, feature importance, Kaggle. Kaggle. Available at: <https://www.kaggle.com/code/prajwallandge/gradient-boosting-pca-feature-importance> (Accessed: November 6, 2022).
21. Pitrubhakta, R. (2021) Feature Importance & Feature Selection, Medium. WiCDS. Available at: <https://medium.com/wicds/feature-importance-feature-selection-acac802ba565> (Accessed: November 15, 2022).
22. Regression analysis in machine learning - javatpoint (no date) www.javatpoint.com. javatpoint. Available at: <https://www.javatpoint.com/regression-analysis-in-machine-learning> (Accessed: October 2, 2022).
23. Sayad, S. (2022) K Nearest Neighbors – Regression, KNN regression. Available at: https://www.saedsayad.com/k_nearest_neighbors_reg.htm (Accessed: October 23, 2022).
24. Singh, A. (2022) K-Nearest Neighbors Algorithm: KNN Regression Python, Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/> (Accessed: November 20, 2022).
25. Sharma, G. (2022) Regression algorithms: 5 regression algorithms you should know, Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2021/05/5-regression-algorithms-you-should-know-introductory-guide/> (Accessed: October 20, 2022).
26. Team, G.L. (2022) Label encoding in Python explained, Great Learning Blog: Free Resources what Matters to shape your Career! Available at: <https://www.mygreatlearning.com/blog/label-encoding-in-python/> (Accessed: October 2, 2022).
27. Techopedia (2017) What is outlier detection? - definition from Techopedia, Techopedia.com. Available at: <https://www.techopedia.com/definition/30351/outlier-detection#:~:text=Outlier%20detection%20is%20the%20process,average%20of%20the%20data%20set.> (Accessed: October 15, 2022).

11.Appendices:

```
from google.colab import drive
drive.mount('/content/drive')
```

Data preparation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import plotly.express as px

## ignore useless warnings
import warnings
warnings.filterwarnings(action='ignore')
```

Import the dataset

```
# Loading the train set
mumbai_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Housing
Prices in Metropolitan Areas of India/Mumbai.csv')
chennai_df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Housing Prices in Metropolitan Areas of India/Chennai.csv')
kolkata_df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Housing Prices in Metropolitan Areas of India/Kolkata.csv')
hydrabad_df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Housing Prices in Metropolitan Areas of India/Hyderabad.csv')
bangalore_df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Housing Prices in Metropolitan Areas of India/Bangalore.csv')
delhi_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Housing
Prices in Metropolitan Areas of India/Delhi.csv')
# Shapes of the dataframes
print('Shape of Mumbai data:', mumbai_df.shape)
print('Shape of Chennai data:', chennai_df.shape)
print('Shape of Kolkata data:', kolkata_df.shape)
print('Shape of Hyderabad data:', hydrabad_df.shape)
print('Shape of Bangalore data:', bangalore_df.shape)
print('Shape of Delhi data:', delhi_df.shape)
```

Combining Datasets

```
#Concating all the dataset into single dataset.
final_df =
pd.concat([mumbai_df, chennai_df, kolkata_df, hydrabad_df, bangalore_df, delh
i_df], ignore_index=True)
```

Descriptive analysis of the dataset

```
print('\nInformation of the dataset..... \n')
print(final_df.info())
print('\nStatistically infomation of the dataset.....\n',
final_df.describe())
```

```
final_df.shape  
(32963, 40)
```

Checking Missing Values

```
final_df.isnull().sum()
```

Null Data seems to be marked as '9' so we'll replace 9 with NaN

```
# Checking Null Values  
x.replace(9, np.nan, inplace=True)  
x.isna().sum()  
# Concating 'No. of Bedrooms' columns back to the dataframe.  
final_df = pd.concat([x,y], axis=1)  
# missing_values percentage  
  
from pandas.core.groupby.grouper import final  
  
missing_values = pd.DataFrame(final_df.isnull().sum() * 100 /  
len(final_df))  
missing_values.columns = ["Missing Values"]  
  
missing_values = missing_values[missing_values["Missing Values"] != 0]  
missing_values.sort_values(by = "Missing Values", axis = 0, ascending =  
False, inplace = True)  
  
missing_values  
# Dropping all the NA values row-wise.  
final_df.dropna(inplace=True)
```

Treating Duplicates value

```
# Printing duplicate values in the dataset  
print(len(final_df[final_df.duplicated()]))  
final_df = final_df.drop_duplicates()  
final_df.shape  
(8226, 40)
```

EDA

```
# Exploring the dataset  
Price =  
final_df.groupby('Price')['Price'].agg('count').sort_values(ascending=False)  
Price  
# Exploring the dataset  
Area =  
final_df.groupby('Area')['Area'].agg('count').sort_values(ascending=False)  
Area  
# Exploring the dataset
```



```
Bedrooms = final_df.groupby('No. of Bedrooms')['No. of  
Bedrooms'].agg('count').sort_values(ascending=False)  
Bedrooms
```

Data Visualization

```
def two_plot(x, title):  
    fig, ax = plt.subplots(1,2,figsize=(10,5))  
    sns.distplot(x, ax=ax[0])  
    ax[0].set(xlabel=None)  
    ax[0].set_title('Histogram + KDE')  
    sns.boxplot(x, ax=ax[1])  
    ax[1].set(xlabel=None)  
    ax[1].set_title('Boxplot')  
    fig.suptitle(title, fontsize=16)  
    plt.tight_layout(pad=3.0)  
    plt.show()
```

Main Features of the dataset

```
two_plot(final_df['Price']/100000, 'Distribution of Price(in Lakhs) for  
all cities')  
two_plot(final_df['Area'], 'Distribution of Area for all cities')  
df1 = final_df[final_df['Resale']== 0]  
df2 = final_df[final_df['Resale']== 1]  
fig, ax = plt.subplots(1,2,figsize=(18, 5))  
ax[0]=sns.countplot(y='Location', data=df1,  
order=df1.Location.value_counts().index[:10],ax=ax[0])  
ax[0].set_title('Number of New Properties')  
ax[1]=sns.countplot(y='Location', data=df2,  
order=df2.Location.value_counts().index[:10],ax=ax[1])  
ax[1].set_title('Number of Resale Properties')  
fig.suptitle("New and Resale Properties in Metropolitan Areas of  
India", fontsize=16)  
plt.tight_layout(pad=3.0)  
plt.show()  
  
# Bar graph for mean price of resale value  
final_df.groupby('Resale')['Price'].mean().plot(kind='bar')  
  
# No. of Properties by Location (Top 25)  
fig, ax = plt.subplots(figsize=(15, 10))  
ax.set_title('No. of Properties by Location (Top 25)', fontsize=20)  
sns.countplot(y='Location', data=final_df,  
order=final_df.Location.value_counts().index[:25])  
ax.set_xlabel('Locations', fontsize=15)  
ax.set_ylabel('No. of Properties', fontsize=15)  
plt.show()  
  
# Area Distribution count plot  
df3 = final_df.copy()  
df3['Area'] = pd.cut(df3['Area'], bins=[0, 250, 500, 750, 1000, 1250,  
1500, 1750, 2000, 2250, 2500, 2750, 3000, np.inf], labels=['0-250',
```

```

-250', '250-500', '500-750', '750-1000', '1000-1250', '1250-1500',
'1500-1750', '1750-2000', '2000-2250', '2250-2500', '2500-2750', '2750-
3000', '3000+'])
fig, ax = plt.subplots(figsize=(15, 10))
ax.set_title('Area Distribution', fontsize=20)
sns.countplot(x='Area', data=df3)
ax.set_ylabel('No of Properties', fontsize=15)
ax.set_xlabel('Area Ranges', fontsize=15)
plt.xticks(rotation=45)
plt.show()

# Price Distribution count plot
df4 = final_df.copy()
df4['Price'] = pd.cut(df4['Price'],
                      bins=[0, 5000000, 10000000, 15000000, 20000000,
25000000, 30000000, 35000000, 40000000, 45000000, 50000000, 55000000,
60000000, np.inf],
                      labels=['0-50', '50-100', '100-150', '150-200',
'200-250', '250-300', '300-350', '350-400', '400-450', '450-500', '550-
600', '650-700', '700+'])

fig, ax = plt.subplots(figsize=(15, 10))
ax.set_title('Price Distribution', fontsize=20)
sns.countplot(x='Price', data=df4)
ax.set_ylabel('No of Properties', fontsize=15)
ax.set_xlabel('Price Ranges (In Lakhs)', fontsize=15)
plt.xticks(rotation=45)
plt.show()

# No. of Bedrooms count plot
fig, ax = plt.subplots(figsize=(15, 10))
ax.set_title('No. of Bedrooms', fontsize=20)
sns.countplot(x='No. of Bedrooms', data=final_df)
ax.set_ylabel('No of Properties', fontsize=15)
ax.set_xlabel('No. of Bedrooms', fontsize=15)
plt.xticks(rotation=45)
plt.show()

```

Other Binary Features of the Dataset

```

def fixlabels(li):
    return ['Yes', 'No'] if li[0] == 1 else ['No', 'Yes']
fig, axes = plt.subplots(nrows=12, ncols=3, figsize=(15, 60))
for i in range(3, len(final_df.columns)-1):
    axes[(i-4)//3][(i-
4)%3].pie(final_df[final_df.columns[i]].value_counts(),
labels=fixlabels(final_df[final_df.columns[i]].value_counts().index),
autopct='%1.1f%%', startangle=90)
    axes[(i-4)//3][(i-4)%3].set_title(final_df.columns[i])
plt.show()
#Correlation

```

```

num=final_df.select_dtypes(exclude='object')
numeric_correlation=num.corr()
plt.figure(figsize=(20,10))
plt.title('Correlation')
sns.heatmap(numeric_correlation>0.75, annot=True, square=True)
plt.show()

# Joint plot for Price & Area
sns.jointplot(x='Area',y='Price',data=final_df ,kind='reg')
# Bar plot for Number of Bedrooms & Price
plt.figure(figsize=(8,8))
temp_df= final_df.groupby(['No. of
Bedrooms','Price']).size().reset_index()
sns.barplot(x=temp_df['No. of Bedrooms'], y=temp_df['Price'],
color='red')
plt.ylabel("Price")
plt.show()

```

Label Encoding

```

# Converting the categorical value into numerical value
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
final_df['Location'] = le.fit_transform(final_df['Location'])

```

Train test split

```

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test =
train_test_split(final_df_std.drop('Price',axis=1),final_df_std['Price'
],random_state=42,test_size=0.2)

```

Feature Scaling

Method 1 - Standard Scaler

```

from sklearn.preprocessing import StandardScaler
standard_scaler=StandardScaler()
final_df_std = final_df.copy()
features=pd.DataFrame(standard_scaler.fit_transform(final_df[['Area','L
ocation','No. of Bedrooms']]),columns= final_df[['Area','Location','No.
of Bedrooms']].columns)
features.head()
final_df_std[['Area','Location','No. of Bedrooms']] = features
final_df_std.isnull().sum()
# Filling na values with 0 temporarily
final_df_std.fillna(0,inplace=True)

```

Model Selection

Linear Regression

```
from sklearn.linear_model import LinearRegression
Linear_model = LinearRegression()
Linear_model.fit(X_train,y_train)
predict = Linear_model.predict(X_test)

# calculating root mean squared error
from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data = ",Linear_model.score(X_train,y_train))
print("Score for testing data = ",Linear_model.score(X_test,y_test))
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor

RFR_model=RandomForestRegressor()
RFR_model.fit(X_train,y_train)
predict = RFR_model.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data = ",RFR_model.score(X_train,y_train))

print("Score for testing data= ",RFR_model.score(X_test,y_test))
```

Randomized Search Cv for Random Forest

```
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 200, num
= 10)]
# Number of features to consider at every split
max_features = ['None', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(1, 100,10)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
```

```

        'min_samples_split': min_samples_split,
        'min_samples_leaf': min_samples_leaf,
        'criterion':['squared_error', 'absolute_error',
'poisson']}]
print(random_grid)
RFR_model=RandomForestRegressor()
rf_randomcv=RandomizedSearchCV(estimator=RFR_model,param_distributions=
random_grid,n_iter=10,cv=10,verbose=2,n_jobs=-1)
### fit the randomized model
rf_randomcv.fit(X_train,y_train)
rf_randomcv.best_params_
rf_randomcv.best_score_
rf_randomcv
best_random_grid=rf_randomcv.best_estimator_
best_random_grid
print("Score for training data = ",rf_randomcv.score(X_train,y_train))
print("Score for testing data = ",rf_randomcv.score(X_test,y_test))

```

GridSearch CV for Random Forest

```

from sklearn.model_selection import GridSearchCV

param_grid = {
    'criterion': [rf_randomcv.best_params_['criterion']],
    'max_depth': [rf_randomcv.best_params_['max_depth']],
    'max_features': [rf_randomcv.best_params_['max_features']],
    'min_samples_leaf': [rf_randomcv.best_params_['min_samples_leaf'],
rf_randomcv.best_params_['min_samples_leaf']+2,
rf_randomcv.best_params_['min_samples_leaf'] +
4],
    'min_samples_split': [rf_randomcv.best_params_['min_samples_split']
- 2,
rf_randomcv.best_params_['min_samples_split']
- 1,
rf_randomcv.best_params_['min_samples_split'],
rf_randomcv.best_params_['min_samples_split']
+1,
rf_randomcv.best_params_['min_samples_split']
+ 2],
    'n_estimators': [rf_randomcv.best_params_['n_estimators'] - 20,
rf_randomcv.best_params_['n_estimators'] - 10,
rf_randomcv.best_params_['n_estimators'],
rf_randomcv.best_params_['n_estimators'] + 10,
rf_randomcv.best_params_['n_estimators'] + 20]
}

print(param_grid)

```

```
#### Fit the grid_search to the data
RFR_model=RandomForestRegressor()
grid_search=GridSearchCV(estimator=RFR_model,param_grid=param_grid,cv=10,n_jobs=-1,verbose=2)
grid_search.fit(X_train,y_train)
grid_search.best_estimator_
best_grid=grid_search.best_estimator_
best_grid
print("Score for training data = ",best_grid.score(X_train,y_train))
print("Score for testing data = ",best_grid.score(X_test,y_test))
```

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor

DTR_model = DecisionTreeRegressor()
DTR_model.fit(X_train,y_train)
predict = DTR_model.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data = ",DTR_model.score(X_train,y_train))

print("Score for testing data = ",DTR_model.score(X_test,y_test))
```

XG Boost

```
from xgboost import XGBRegressor

XGBR = XGBRegressor(max_depth=10,verbosity=1)

XGBR.fit(X_train,y_train)
predict = XGBR.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data = ",XGBR.score(X_train,y_train))
print("Score for testing data= ",XGBR.score(X_test,y_test))
```

Method 2 by Min Max Scaler

```
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
final_df_minmax = final_df.copy()
feature=
pd.DataFrame(min_max_scaler.fit_transform(final_df[['Area','Location'],'
```

```
'Area','Location','No. of Bedrooms']]),columns=
final_df[['Area','Location','No. of Bedrooms']].columns)
final_df_minmax[['Area','Location','No. of Bedrooms']] = feature
final_df_minmax.isnull().sum()
# Filling na values with 0 temporarily
final_df_minmax.fillna(0,inplace=True)
```

Model Selection

Linear Regression

```
from sklearn.linear_model import LinearRegression
Linear_model = LinearRegression()
Linear_model.fit(X_train,y_train)
predict = Linear_model.predict(X_test)

# calculating root mean squared error
from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data= ",Linear_model.score(X_train,y_train))
print("Score for testing data= ",Linear_model.score(X_test,y_test))
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor

RFR_model=RandomForestRegressor()
RFR_model.fit(X_train,y_train)
predict = RFR_model.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data= ",RFR_model.score(X_train,y_train))
print("Score for testing data= ",RFR_model.score(X_test,y_test))
```

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor

DTR_model = DecisionTreeRegressor()
DTR_model.fit(X_train,y_train)
predict = DTR_model.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
```

```
print("Rmse = ",rmse)

print("Score for training data= ",DTR_model.score(X_train,y_train))
print("Score for testing data= ",DTR_model.score(X_test,y_test))
```

XG Boost Technique

```
from xgboost import XGBRegressor

XGBR = XGBRegressor(max_depth=10,verbosity=1)
print(XGBR)
XGBR.fit(X_train,y_train)
predict = XGBR.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data= ",XGBR.score(X_train,y_train))
print("Score for testing data= ",XGBR.score(X_test,y_test))
```

KNN Regressor

```
from sklearn.neighbors import KNeighborsRegressor

KNN = KNeighborsRegressor()
KNN.fit(X_train,y_train)
predict = KNN.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data= ",KNN.score(X_train,y_train))
print("Score for testing data= ",KNN.score(X_test,y_test))
```

Method 3 - Without Feature Scaling

Oulier Treatment

```
for feature in ['No. of Bedrooms','Area','Price']:
    print('Initial Skew value: ', final_df[feature].skew())
    Q1 = final_df[feature].quantile(0.25)
    Q3 = final_df[feature].quantile(0.75)
    #print(Q1,Q3)
    final_df[feature] = np.where(final_df[feature] <Q1,
Q1,final_df[feature])
    final_df[feature] = np.where(final_df[feature] >Q3,
Q3,final_df[feature])
    print('Final Skew value: ', final_df[feature].skew())
```



```
print()
```

Model Selection

Linear Regression

```
from sklearn.linear_model import LinearRegression
Linear_model = LinearRegression()
Linear_model.fit(X_train,y_train)
predict = Linear_model.predict(X_test)

# calculating root mean squared error
from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data= ",Linear_model.score(X_train,y_train))
print("Score for testing data= ",Linear_model.score(X_test,y_test))
```

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor

DTR_model = DecisionTreeRegressor()
DTR_model.fit(X_train,y_train)
predict = DTR_model.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data = ",DTR_model.score(X_train,y_train))
print("Score for testing data = ",DTR_model.score(X_test,y_test))
```

XG Boost

```
from xgboost import XGBRegressor

XGBR = XGBRegressor(max_depth=10,verbosity=1)

XGBR.fit(X_train,y_train)
predict = XGBR.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score = ",XGBR.score(X_train,y_train))
print("Score for testing data= ",XGBR.score(X_test,y_test))
```

Hyperparameter Tuning using Grid Search for XG Boost

```
param_grid = {"max_depth": [5,10,15],
              "n_estimators": [50,150,200],
              "learning_rate": [0.01, 0.015]}
search = GridSearchCV(XGBR, param_grid, cv=5).fit(X_train, y_train)
print("The best hyperparameters are ",search.best_params_)
XGBR = XGBRegressor(learning_rate = 0.015, max_depth = 15, n_estimators
= 200)

XGBR.fit(X_train,y_train)
predict = XGBR.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score = ",XGBR.score(X_train,y_train))

print("Score for testing data= ",XGBR.score(X_test,y_test))
```

Light Gradient Boosting Technique

```
from lightgbm import LGBMRegressor

LXGB = LGBMRegressor()

LXGB.fit(X_train,y_train)
predict = LXGB.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data= ",LXGB.score(X_train,y_train))
print("Score for testing data= ",LXGB.score(X_test,y_test))
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor

RFR_model=RandomForestRegressor()
RFR_model.fit(X_train,y_train)
predict = RFR_model.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data= ",RFR_model.score(X_train,y_train))
```

```
print("Score for testing data= ",RFR_model.score(X_test,y_test))
```

Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
GBR = GradientBoostingRegressor()
```

```
GBR.fit(X_train,y_train)
```

```
predict = GBR.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error
```

```
rmse=np.sqrt(mean_squared_error(y_test,predict))
```

```
print("Rmse = ",rmse)
```

```
print("Score for training data = ",GBR.score(X_train,y_train))
```

```
print("Score for testing data = ",GBR.score(X_test,y_test))
```

KNN

```
from sklearn.neighbors import KNeighborsRegressor
```

```
KNN = KNeighborsRegressor()
```

```
KNN.fit(X_train,y_train)
```

```
predict = KNN.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error
```

```
rmse=np.sqrt(mean_squared_error(y_test,predict))
```

```
print("Rmse = ",rmse)
```

```
print("Score for training data= ",KNN.score(X_train,y_train))
```

```
print("Score for testing data= ",KNN.score(X_test,y_test))
```

Results

```
list = []
```

```
list.append(RFR_model.score(X_train,y_train))
```

```
list.append(DTR_model.score(X_train,y_train))
```

```
list.append(XGBR.score(X_train,y_train))
```

```
list.append(GBR.score(X_train,y_train))
```

```
list.append(LXGB.score(X_train,y_train))
```

```
#list.append(Linear_model.score(X_train,y_train))
```

```
list.append(KNN.score(X_train,y_train))
```

```
list1 = []
```

```
list1.append(RFR_model.score(X_test,y_test))
```

```
list1.append(DTR_model.score(X_test,y_test))
```

```
list1.append(XGBR.score(X_test,y_test))
```

```
list1.append(GBR.score(X_test,y_test))
```

```
list1.append(LXGB.score(X_test,y_test))
#list1.append(Linear_model.score(X_test,y_test))
list1.append(KNN.score(X_test,y_test))
from operator import index
arr = []
arr1 = []
arr =
pd.DataFrame(list,index=["RF","DTR","XGBR",'GBR','LXGB','KNN'],columns=
['Train'])
arr1 =
pd.DataFrame(list1,index=["RF","DTR","XGBR",'GBR','LXGB','KNN'],columns=
['Test'])
result = pd.concat([arr,arr1],axis=1)
```

Feature Selection

Method 1 : Feature Importance

```
RFR_model.feature_importances_
feature_importance = pd.DataFrame({'importance':
RFR_model.feature_importances_}, index=
X_train.columns).sort_values('importance')
# Top features selection
feature_importance=feature_importance[feature_importance.importance>0.0
20]
feature_importance.plot(kind='barh',figsize=(15,20))
```

Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor

GBR = GradientBoostingRegressor()

GBR.fit(X_train,y_train)
predict = GBR.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score = ",GBR.score(X_train,y_train))
print("Score for testing data= ",GBR.score(X_test,y_test))
```

XG Boost

```
from xgboost import XGBRegressor

XGBR = XGBRegressor(max_depth=10,verbosity=1)

XGBR.fit(X_train,y_train)
predict = XGBR.predict(X_test)
```

```

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score = ",XGBR.score(X_train,y_train))
print("Score for testing data= ",XGBR.score(X_test,y_test))

```

Hyperparameter Tuning using Grid Search for XG Boost

```

param_grid = {"max_depth": [5,10,15],
              "n_estimators": [50,150,200],
              "learning_rate": [0.01, 0.015]}
search = GridSearchCV(XGBR, param_grid, cv=5).fit(X_train, y_train)
print("The best hyperparameters are ",search.best_params_)
XGBR = XGBRegressor(learning_rate = 0.015, max_depth = 15, n_estimators
= 200)

XGBR.fit(X_train,y_train)
predict = XGBR.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data = ",XGBR.score(X_train,y_train))
print("Score for testing data= ",XGBR.score(X_test,y_test))

```

Light Gradient Boosting Technique

```

from lightgbm import LGBMRegressor

LXGB = LGBMRegressor()

LXGB.fit(X_train,y_train)
predict = LXGB.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score = ",LXGB.score(X_train,y_train))
print("Score for testing data= ",LXGB.score(X_test,y_test))

```

Random Forest Regressor

```

from sklearn.ensemble import RandomForestRegressor

RFR_model=RandomForestRegressor()
RFR_model.fit(X_train,y_train)
predict = RFR_model.predict(X_test)

```

```

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data= ",RFR_model.score(X_train,y_train))

print("Score for testing data= ",RFR_model.score(X_test,y_test))

```

KNN

```

from sklearn.neighbors import KNeighborsRegressor

KNN = KNeighborsRegressor()
KNN.fit(X_train,y_train)
predict = KNN.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score = ",KNN.score(X_train,y_train))

print("Score for testing data= ",KNN.score(X_test,y_test))

```

Results

```

list = []

list.append(RFR_model.score(X_train,y_train))
list.append(DTR_model.score(X_train,y_train))
list.append(XGBR.score(X_train,y_train))
list.append(GBR.score(X_train,y_train))
list.append(LXGB.score(X_train,y_train))
#list.append(Linear_model.score(X_train,y_train))
list.append(KNN.score(X_train,y_train))
list1 = []

list1.append(RFR_model.score(X_test,y_test))
list1.append(DTR_model.score(X_test,y_test))
list1.append(XGBR.score(X_test,y_test))
list1.append(GBR.score(X_test,y_test))
list1.append(LXGB.score(X_test,y_test))
#list1.append(Linear_model.score(X_test,y_test))
list1.append(KNN.score(X_test,y_test))
from operator import index
arr = []
arr1 = []

```

```

arr =
pd.DataFrame(list,index=["RF","DTR","XGBR",'GBR','LXGB','KNN'],columns=
['Train'])
arr1 =
pd.DataFrame(list1,index=["RF","DTR","XGBR",'GBR','LXGB','KNN'],columns=
['Test'])
result = pd.concat([arr,arr1],axis=1)
result

```

Method 2 : PCA

```

from sklearn.ensemble import RandomForestRegressor
from sklearn.decomposition import PCA
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

x = final_df.drop('Price', axis=1)
rf=RandomForestRegressor(100)
pc_x=PCA(n_components=3)
X_pca = pc_x.fit_transform(x)
X_pca.shape

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test =
train_test_split(X_pca,final_df['Price'],random_state=42,test_size=0.2)

```

Random Forest

```

rf = rf.fit(X_train,y_train)

print("Score for training data = ",rf.score(X_train,y_train))
print("Score for testing data = ",rf.score(X_test,y_test))

```

Decision Tree

```

dt = DTR_model.fit(X_train,y_train)

print("Score for training data = ",dt.score(X_train,y_train))
print("Score for testing data= ",dt.score(X_test,y_test))

```

Light Gradient Boosting Technique

```

from lightgbm import LGBMRegressor

LXGB = LGBMRegressor()

LXGB.fit(X_train,y_train)
predict = LXGB.predict(X_test)

from sklearn.metrics import mean_squared_error

```

```
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data= ",LXGB.score(X_train,y_train))
print("Score for testing data= ",LXGB.score(X_test,y_test))
```

Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor

GBR = GradientBoostingRegressor()

GBR.fit(X_train,y_train)
predict = GBR.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data = ",GBR.score(X_train,y_train))
print("Score for testing data= ",GBR.score(X_test,y_test))
```

XG Boost

```
from xgboost import XGBRegressor

XGBR = XGBRegressor(max_depth=10,verbosity=1)

XGBR.fit(X_train,y_train)
predict = XGBR.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data = ",XGBR.score(X_train,y_train))
print("Score for testing data= ",XGBR.score(X_test,y_test))
```

KNN

```
from sklearn.neighbors import KNeighborsRegressor

KNN = KNeighborsRegressor()
KNN.fit(X_train,y_train)
predict = KNN.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse=np.sqrt(mean_squared_error(y_test,predict))
print("Rmse = ",rmse)

print("Score for training data = ",KNN.score(X_train,y_train))
```



```
print("Score for testing data= ",KNN.score(X_test,y_test))
```

Results

```
list = []

list.append(rf.score(X_train,y_train))
list.append(dt.score(X_train,y_train))
list.append(XGBR.score(X_train,y_train))
list.append(GBR.score(X_train,y_train))
list.append(LXGB.score(X_train,y_train))
#list.append(Linear_model.score(X_train,y_train))
list.append(KNN.score(X_train,y_train))
list1 = []

list1.append(rf.score(X_test,y_test))
list1.append(dt.score(X_test,y_test))
list1.append(XGBR.score(X_test,y_test))
list1.append(GBR.score(X_test,y_test))
list1.append(LXGB.score(X_test,y_test))
#list1.append(Linear_model.score(X_test,y_test))
list1.append(KNN.score(X_test,y_test))
from operator import index
arr = []
arr1 = []
arr =
pd.DataFrame(list,index=["RF","DTR","XGBR",'GBR','LXGB','KNN'],columns=
['Train'])
arr1 =
pd.DataFrame(list1,index=["RF","DTR","XGBR",'GBR','LXGB','KNN'],columns
=['Test'])
result = pd.concat([arr,arr1],axis=1)
result
```

Neural Network (ANN)

```
# Creating a Neural Network Model
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam
# having 40 neuron is based on the number of available features
model = Sequential()
model.add(Dense(40,activation='relu'))
model.add(Dense(40,activation='relu'))
model.add(Dense(40,activation='relu'))
model.add(Dense(40,activation='relu'))
model.add(Dense(1))
model.compile(optimizer='Adam',loss='mse')
model.fit(x=X_train,y=y_train,
```

```
        validation_data=(X_test,y_test),  
        batch_size=128,epochs=400)  
model.summary()  
# Loss function graph.  
loss_df = pd.DataFrame(model.history.history)  
loss_df.plot(figsize=(12,8))
```

```
y_pred = model.predict(X_test)  
from sklearn import metrics  
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))  
print('MSE:', metrics.mean_squared_error(y_test, y_pred))  
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))  
print('VarScore:', metrics.explained_variance_score(y_test, y_pred))  
print('R2 Score:', metrics.r2_score(y_test, y_pred))  
# visualizing residuals  
fig = plt.figure(figsize=(10,5))  
residuals = (y_test.values - y_pred)  
sns.distplot(residuals)
```