

REPORT ON HOUSE PRICE PREDICTION



S. NO.	Topic	Page No.
1	<p>Introduction</p> <ul style="list-style-type: none"> Machine learning in the Real Estate Main Objectives of House Price Prediction 	<p>5</p> <p>6</p>
2	<p>Workflow</p> <ul style="list-style-type: none"> Machine Learning Life Cycle <p>Fig-1: Workflow for House Price Prediction</p> <ul style="list-style-type: none"> About Dataset <p>Fig-2: First five rows of the data set</p>	<p>6</p> <p>7</p>
3	<p>Exploratory Data Analysis</p> <ul style="list-style-type: none"> Data Analysis Unique values analysis of the categorical columns <p>Fig-3: Code snapt for checking the unique values of categorical columns</p> <p>Fig-4: Unique values of size column</p> <p>Fig-5: Unique values of area_type column</p> <p>Fig-6: Unique values of location column</p> <p>Fig-7: Unique values of availability column</p> <p>Fig-8: Unique values of society column</p> <p>Fig-9: Unique values of balcony column</p> <p>Fig-10: Unique values of bath column</p> <p>Fig-11: Skewness of the data</p> <ul style="list-style-type: none"> Statistical Analysis Univariate Analysis <p>Fig-13: UA of area_type</p> <p>Fig-14: UA of location</p> <p>Fig-15: UA of size</p> <p>Fig-16: UA of society</p> <p>Fig-17: UA of bath</p> <p>Fig-18: UA of balcony</p>	<p>8</p> <p>9</p> <p>10</p> <p>11</p> <p>12</p> <p>13</p> <p>14</p>

	Fig-19: UA of total_sqft Fig-20: UA of price(in lakhs) • Bivariate Analysis Fig-21: BA of area_type Fig-22: BA of location(1) Fig-23: BA of location(2) Fig-24: BA of size Fig-25: BA of bath Fig-26: BA of balcony Fig-27: BA of total_sqft	15 16 17 18 19
4	Date Cleaning and Preprocessing • Missing Value Fig-28: Code snapt of Checking the missing values Fig-29: Code snapt of handling the missing data • Variable Transformation Fig-30: Code snapt of variable transformation • Treating the duplicate values Fig-31: Code snapt of handling duplicate date • Outlier detection and handling Fig-32: Code snapt of outliers handling	20 21 22 23 24
5	Modeling • Cluster analysis Fig-33: Code snapt of data scaling Fig-34: Elbow plot using Silhouette analysis for 100 clusters Fig-35: Elbow plot using Sum of Squared Distance for 100 clusters Fig-36: Code snapt of k mean clustering using Sklearn module Fig-37: Cluster creation with k=2 Fig-38: Code snapt of k means clustering using own function Fig-39: Cluster creation with k=60 Fig-40: Cluster creation using dendrogram • Machine Learning Algorithm Fig-41: Code snapt of separation of independent and dependent variable Fig-42: Heatmap of the independent variables Fig-43: VIF between the independent variables	24 25 26 27 28 29 30 31

	Fig-44: Code snapt of showing the final features Fig-45: Pairplot of the final features Fig-46: Code snapt of data splitting • Used Algorithms	32 33
6	Result and Performance • Performance Metrics • Result Analysis for Individual Models Fig-47: Regression plot of Simple Linear Regression Fig-48: Regression plot of Lasso Regression Fig-49: Regression plot of Ridge Regression Fig-50: Regression plot of KNN Fig-51: Regression plot of SVR with RBF Fig-52: Regression plot of Simple Decision Tree Fig-53: Regression plot of Logistic Regression Fig-54: Regression plot of Gradient Boosting Fig-55: Regression plot of Bagging Fig-56: Regression plot of Random Forest Fig-57: Regression plot of AdaBoost Fig-58: Regression plot of XGBoost Fig-59: Code snapt of assigning parameter grid of Gradient Boosting by Grid Search Fig-60: Tuned parameters of Gradient boosting using Grid Search Fig-61: Regression plot of Gradient Boosting by Grid Search Fig-62: Code snapt of assigning parameter grid of Gradient Boosting Tuning by Graph Fig-63: Tuned parameters of Gradient boosting Tuning using Graph Fig-64: Regression plot of Gradient Boosting Tuning by Graph Fig-65: Code snapt of assigning parameter grid of Gradient Boosting by Random Search Fig-66: Tuned parameters of Gradient boosting using Random Search Fig-67: Regression plot of Gradient Boosting by Random Search • Performance Comparison Fig-68: Models performance	36 38 39 40 41 42 43 44 45 46 47 48
7	Final Finding and Recommendation • Findings	48

	• Recommendation	49
8	Conclusion	50

1. Introduction

Real estate is anything continuously connected to or constructed on land, whether it be created naturally or artificially. Real estate includes the land as well as any individual's human constructions, including homes and other structures. An improvement is any addition to or alteration to the land that raises or lowers the property's value. Real property consists of the land, its improvements, as well as the underlying ownership and usage rights. Real estate can be divided into five primary categories: residential, commercial, industrial, raw land, and special use.

The most important challenges by real estate agents are:

1. The fact that the real estate business has no control over regional market circumstances is one issue that looks unavoidable. However, they have power over how they deal with their clientele in this circumstance.
2. As a real estate agent, they get to fully feel the emotions that come with selling a property, which can be a difficult time for many people. When it's time to move on, clients may feel anxious, uncertain, excited, stressed, sad, and a variety of other feelings.
3. One of the biggest challenges for real estate agents is setting the asking price for a home, especially when the customer is insistent about it. Inform their clients about the terrible effects that an overpriced property may have and the logic behind their figures.
4. On the other hand, they may also run against customers who are unrealistic home buyers. They demand everything for free. They want to enter with an offer that is much less than the house is worth. Realistic buyers may also hold irrational beliefs about the state of a house.

Machine learning in the Real Estate:

AI and ML are essential in the real estate sector. During recent years, AI has had an impact on all facets of the real estate industry. The following scenarios will help us comprehend how artificial intelligence will impact standard real estate transactions. AI in real estate can help businesses determine the ideal moment to buy or sell a property as well as project future sale or rental pricing. In order to determine a reasonable pricing range, it can also use a regression technique that takes into account property characteristics like size, age, the number of rooms, and house furnishings. People who intend to purchase a home will likely benefit from predictions of house prices since they will be able to better organize their finances if they are aware of the price range in the future. Additionally, projections of house prices help real estate investors understand the trajectory of local housing costs.

Main Objectives of House Price Prediction

1. Create a framework to identify the essential features influencing the cost of a house using Exploratory Data Analysis
2. Create machine learning models to understand the aspects affecting the cluster model for houses and estimate house prices based on the attributes.

2. Workflow

Machine Learning Life Cycle

Data science projects go through a cyclical process called the machine learning life cycle. It outlines each step a company needs to take to benefit from artificial intelligence (AI) and machine learning in order to provide useful economic value. The machine learning life cycle consists of five main processes, each of which is equally significant and follows a predetermined sequence. Define the project's goals, acquire and explore data, model data, interpret and communicate, implement, document, and maintain data are all included in this list.

The machine learning life cycle is attempted to be maintained in this project. The project's primary workflow is depicted in the diagram below:

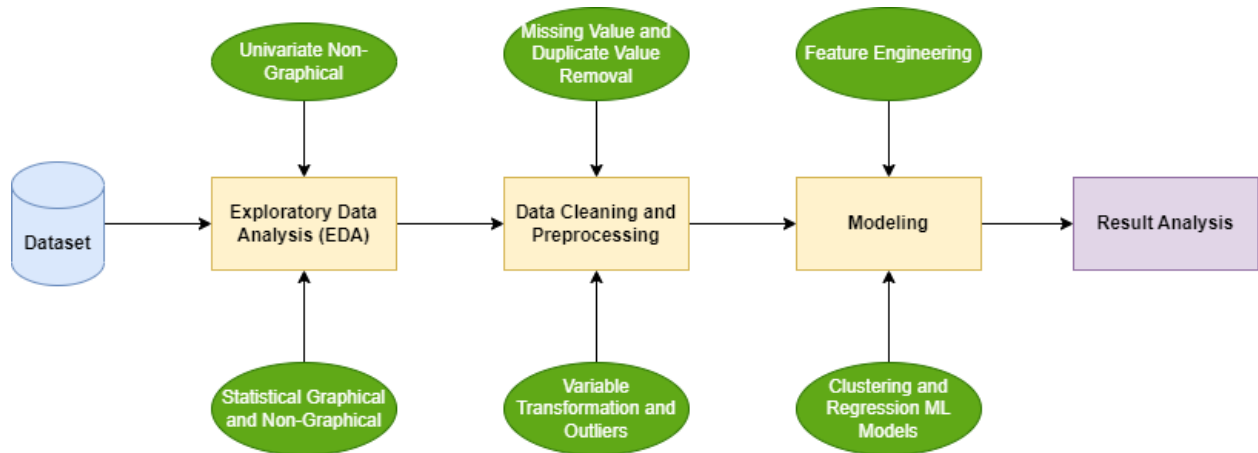


Fig-1: Workflow for House Price Prediction

All the steps in the workflow are discussed in the following sections.

About Dataset

The data is collected from the kaggle which is Bangalore house price data. It consists of 8 independent features and 1 target feature which is the price of the house. The 8 independent features are: area type that means in which type of area the house belongs to, location of house, availability that means when the house will available., size (total rooms), society of the house, total bathrooms, total balconies and total sqft of the house. There are 13320 rows and 9 columns in the dataset. The dataset appears as follows:

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00

Fig-2: First five rows of the data set

3. Exploratory Data Analysis

Data analysis utilizing visual methods is called exploratory data analysis (EDA). With the aid of statistical summaries and graphical representations, it is used to identify trends, patterns, or to verify hypotheses.

Data Analysis

Unique values analysis of the categorical columns

```
def check_cat_cols(data, col):  
    print(f"Unique values of the {col} column: {df[col].unique()}\n")  
    print(f"Total unique values of the {col} column: {len(df[col].unique())}")
```

Fig-3: Code snap for checking the unique values of categorical columns

Size column

```
1 check_cat_cols(df, 'size')  
  
Unique values of the size column: ['2 BHK' '4 Bedroom' '3 BHK' '4 BHK' '6 Bedroom' '3 Bedroom' '1 BHK'  
'1 RK' '1 Bedroom' '8 Bedroom' '2 Bedroom' '7 Bedroom' '5 BHK' '7 BHK'  
'6 BHK' '5 Bedroom' '11 BHK' '9 BHK' nan '9 Bedroom' '27 BHK'  
'10 Bedroom' '11 Bedroom' '10 BHK' '19 BHK' '16 BHK' '43 Bedroom'  
'14 BHK' '8 BHK' '12 Bedroom' '13 BHK' '18 Bedroom']  
  
Total unique values of the size column: 32
```

Fig-4: Unique values of size column

Area_type column

```
1 check_cat_cols(df, 'area_type')  
  
Unique values of the area_type column: ['Super built-up Area' 'Plot Area' 'Built-up Area' 'Carpet Area']  
  
Total unique values of the area_type column: 4
```

Fig-5: Unique values of area_type column

Location column

```
1 check_cat_cols(df, 'location')

Unique values of the location column: ['Electronic City Phase II' 'Chikka Tirupathi' 'Uttarahalli' ...
'12th cross srinivas nagar banshankari 3rd stage' 'Havanur extension'
'Abshot Layout']

Total unique values of the location column: 1306
```

Fig-6: Unique values of location column

Availability column

```
1 check_cat_cols(df, 'availability')

Unique values of the availability column: ['19-Dec' 'Ready To Move' '18-May' '18-Feb' '18-Nov' '20-Dec' '17-Oct'
'21-Dec' '19-Sep' '20-Sep' '18-Mar' '20-Feb' '18-Apr' '20-Aug' '18-Oct'
'19-Mar' '17-Sep' '18-Dec' '17-Aug' '19-Apr' '18-Jun' '22-Dec' '22-Jan'
'18-Aug' '19-Jan' '17-Jul' '18-Jul' '21-Jun' '20-May' '19-Aug' '18-Sep'
'17-May' '17-Jun' '21-May' '18-Jan' '20-Mar' '17-Dec' '16-Mar' '19-Jun'
'22-Jun' '19-Jul' '21-Feb' 'Immediate Possession' '19-May' '17-Nov'
'20-Oct' '20-Jun' '19-Feb' '21-Oct' '21-Jan' '17-Mar' '17-Apr' '22-May'
'19-Oct' '21-Jul' '21-Nov' '21-Mar' '16-Dec' '22-Mar' '20-Jan' '21-Sep'
'21-Aug' '14-Nov' '19-Nov' '15-Nov' '16-Jul' '15-Jun' '17-Feb' '20-Nov'
'20-Jul' '16-Sep' '15-Oct' '15-Dec' '16-Oct' '22-Nov' '15-Aug' '17-Jan'
'16-Nov' '20-Apr' '16-Jan' '14-Jul']

Total unique values of the availability column: 81
```

Fig-7: Unique values of availability column

- Availability column is not necessary to predict the house as the dataset is not dynamic so prices are fixed.

Society column

```
1 check_cat_cols(df, 'society')

Unique values of the society column: ['Coomee ' 'Theanmp' nan ... 'SJovest' 'ThhtsV ' 'RSntsAp']

Total unique values of the society column: 2689
```

Fig-8: Unique values of society column

Balcony column

```
1 check_cat_cols(df, 'balcony')

Unique values of the balcony column: [ 1.  3. nan  2.  0.]

Total unique values of the balcony column: 5
```

Fig-9: Unique values of balcony column

Bath column

```
1 check_cat_cols(df, 'bath')

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

Total unique values of the bath column: 20
```

Fig-10: Unique values of bath column

Skewness of the data

```
Checking the skewness of the data.
Mean of the columns:
area_type      2.264367
location      673.001895
size          2.820808
society       1911.195295
total_sqft    1572.066238
bath          2.708083
balcony       1.603884
price (in lakhs) 114.712707
dtype: float64
Median of the columns:
area_type      3.0
location      654.0
size          3.0
society       2232.0
total_sqft    1288.5
bath          2.0
balcony       2.0
price (in lakhs) 73.0
dtype: float64
```

Fig-11: Skewness of the data

Observation:

area_type: Mean < Median, so Left skewed
location: Mean slightly > Median, so slightly Right skewed
size: Mean < Median slightly, so slightly Left skewed
society: Mean < Median, so Left skewed
total_sqft: Mean > Median, so Right skewed
bath: Mean slightly > Median, so slightly Right skewed
balcony: Mean < Median slightly, so slightly Left skewed
price (in lakhs): Mean > Median, so Right skewed

Statistical Analysis:

For statistical analysis, I used univariate and bivariate analysis. In a univariate analysis, every variable in a data set is examined independently. It examines both the distribution of values and their central tendency. It describes the manner in which the variable has responded. Each individual variable is described. Data are summarized and described using descriptive statistics. One type of statistical analysis where two variables are observed is known as bivariate analysis. Here we have a dependent variable and an independent variable. Typically, X and Y are used to represent these variables. Therefore, in this analysis, we examine how much the two variables changed from one another.

Univariate Analysis (UA)

UA of area type column

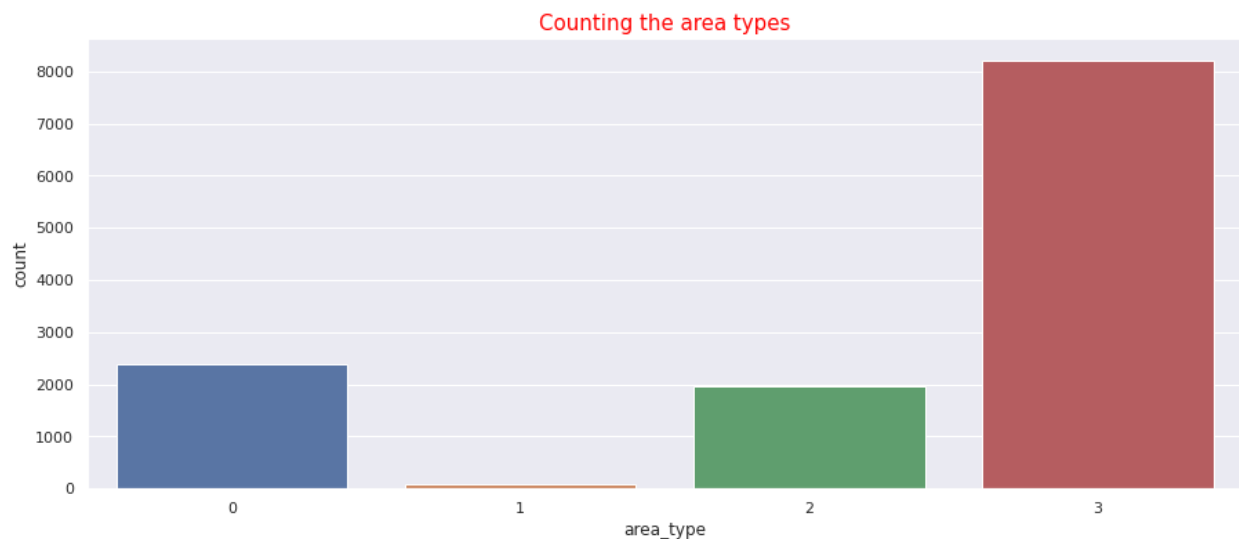


Fig-13: UA of area_type

Observation:

- Here we can see that area type 1 is much lower than the others. We may consider it as outliers but we can confirm it from the outlier analysis.

Analysis of location:

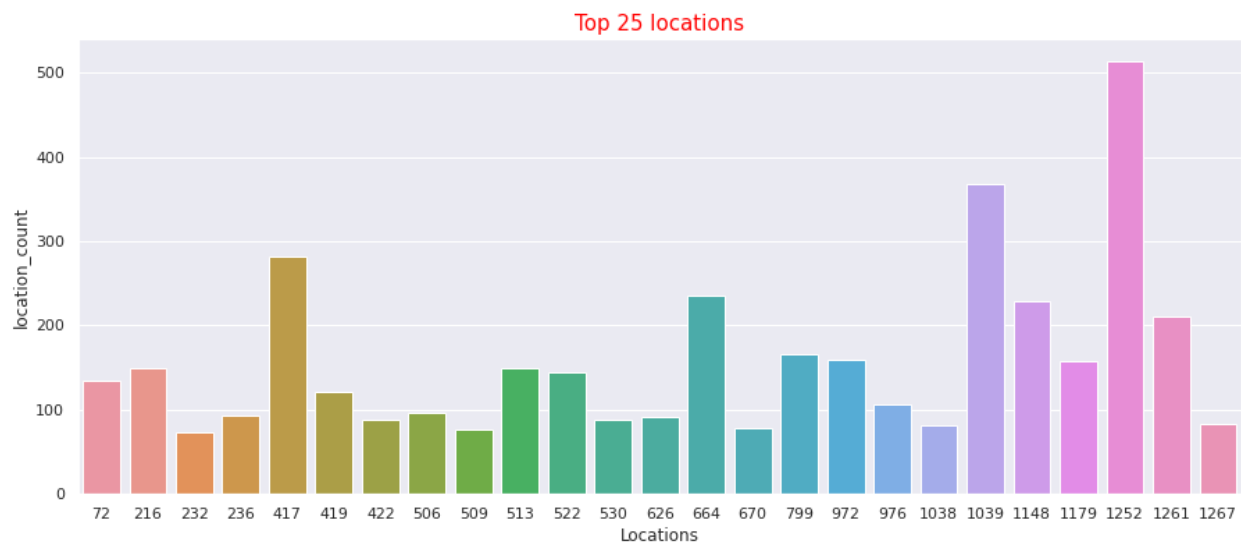


Fig-14: UA of location

Observation:

- Maximum appearance of any location is around 500 that means location is a big factor for housing prices.
- Any location which appears less than or equal to 5 we can treat them as outliers.

Analysis of size:

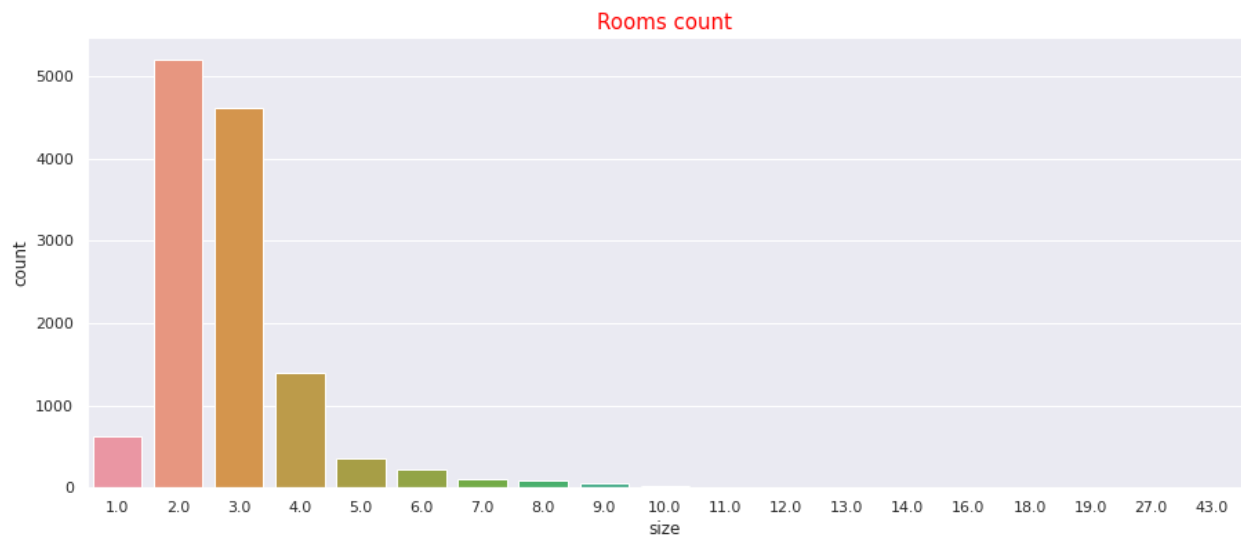


Fig-15: UA of size

Observation:

- We can observe that most homes have two or three rooms.
- Since rooms with more than 6 are extremely uncommon, we will remove the data for rooms with 7 or more.

Analysis of Society column

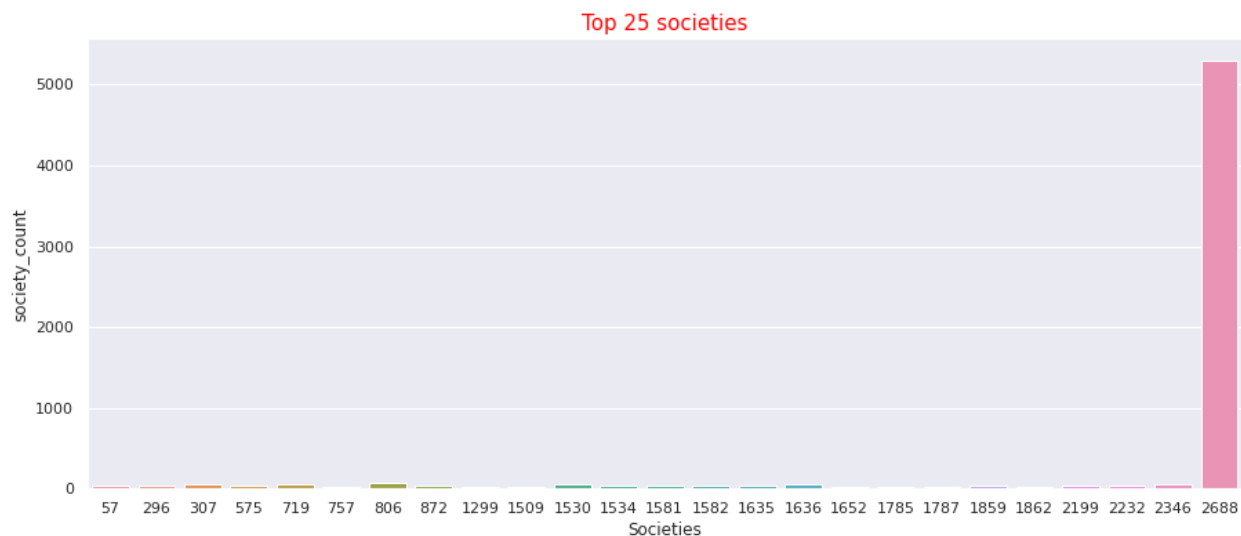


Fig-16: UA of society

Observation:

- Maximum number of societies are coming from 2688 which was encoded for the null values. So we can remove the column as it is not necessary to build the model.

Analysis of bath:

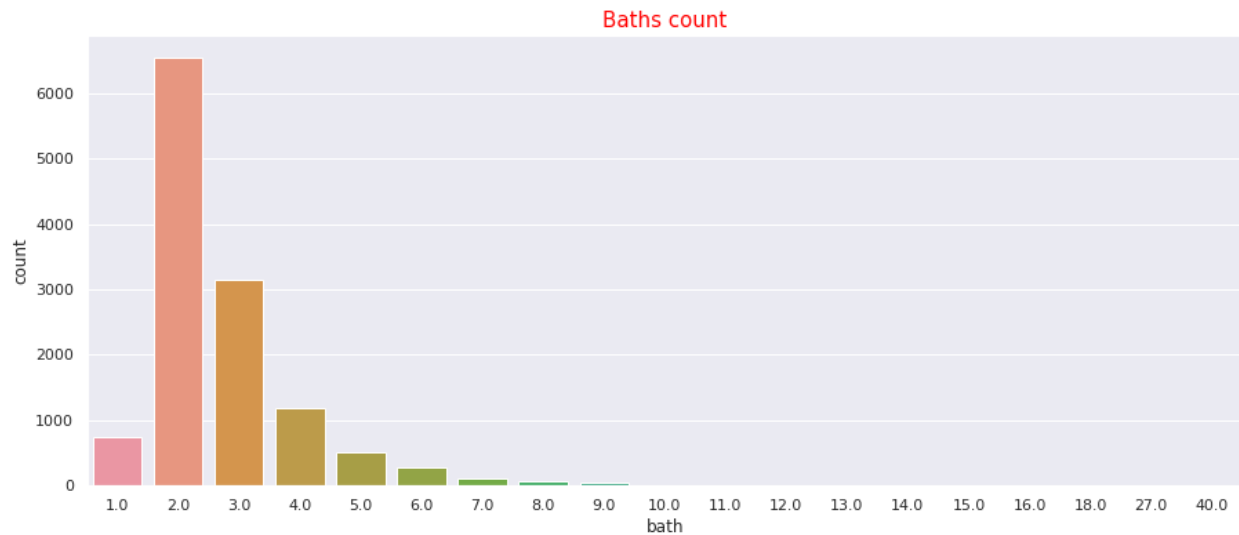


Fig-17: UA of bath

Observation:

- The graph gives the impression that most homes have two or three bathrooms upto four.
- Considering up to 5 bathrooms in relation to a 6 room house (4 baths for 4 bedrooms and one servant bath).

Analysis of balcony:

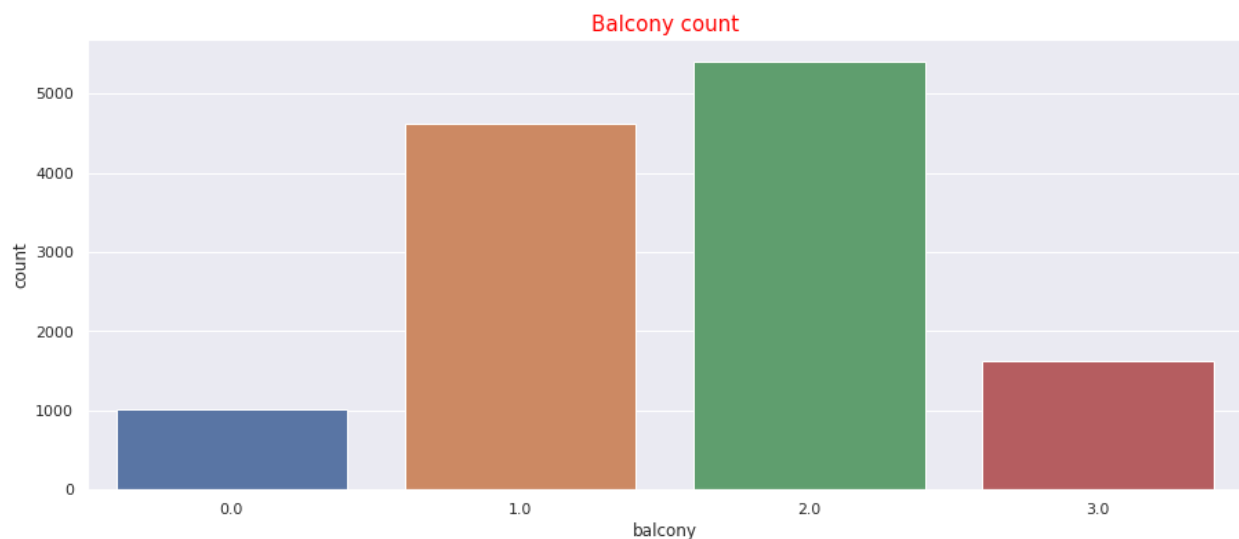


Fig-18: UA of balcony

Observation:

- Most houses have one or two balconies
- A home without a balcony is strange. The data will be deleted with 0 balconies.

We have taken into consideration up to four bedrooms, thus it is entirely probable that any bedroom will be modest and lack a balcony.

Analysis of total_sqft:

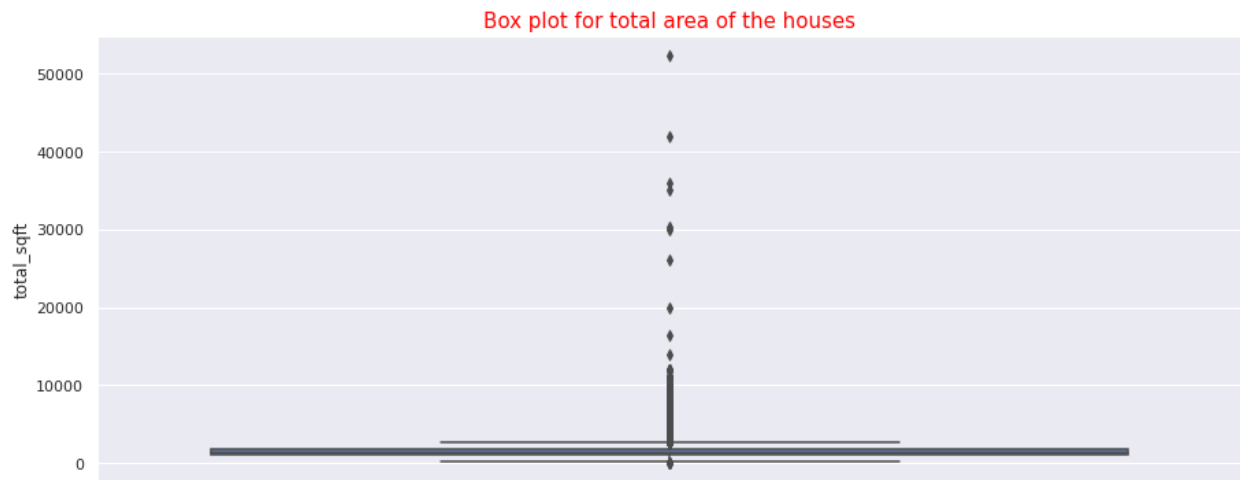


Fig-19: UA of total_sqft

Observation:

- The box plot makes it evident that there are outliers. A house with more than 5000 square feet is extremely rare. We'll evaluate it using an outlier analysis.

Analysis of price(in lakhs)



Fig-20: UA of price(in lakhs)

Observation:

- The box plot makes it evident that there are outliers. We'll evaluate it using an outlier analysis.

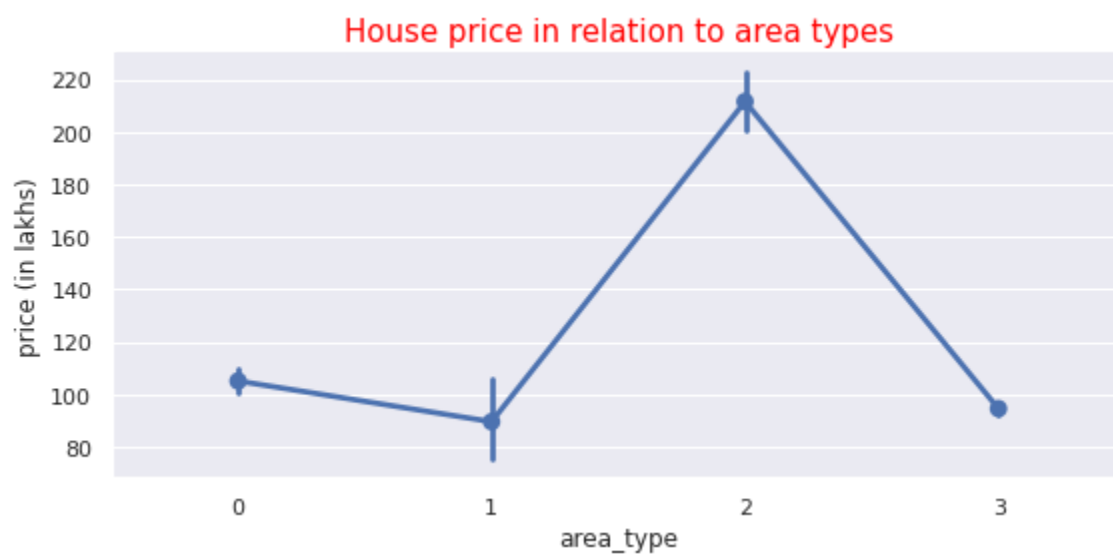
Bivariate Analysis (BA)**BA of area_type**

Fig-21: BA of area_type

Observation:

It is evident that the median is the same for both area types 0 and 1. It declines for type 1 and rises for type 2. Both the mean and the median are high for area type 2. As a result, area type 2 can be described as expensive.

BA of Location

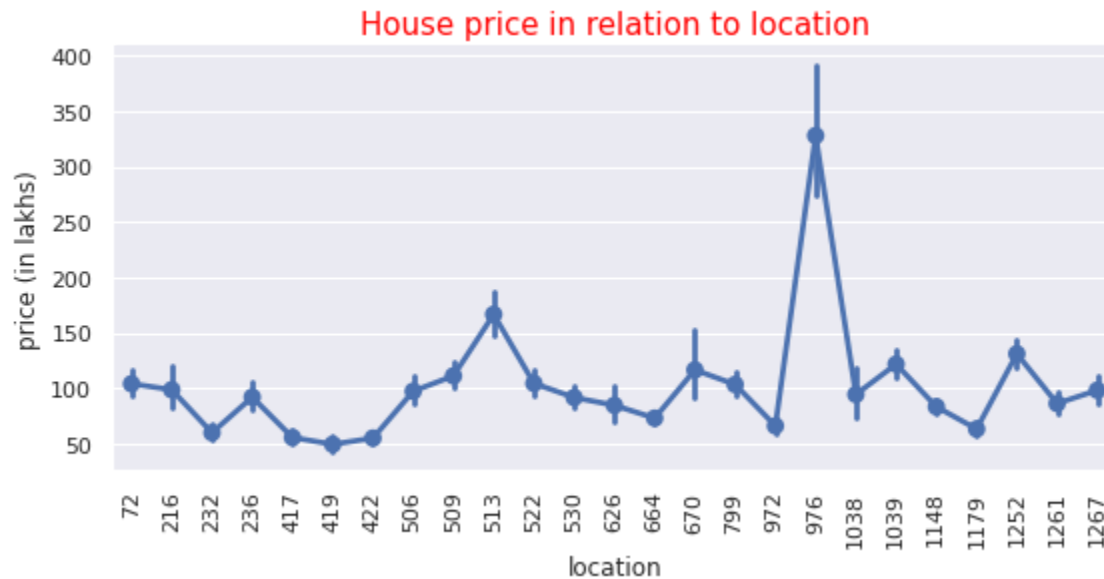


Fig-22: BA of location(1)

Observation:

- We find that maximum mean and median is for the location 351 yet it appears only one time in the sample. So consider the most appeared sites. The graph shows that the location 976 is where the extreme price is coming from.

```
1 print(f"Mean, median price of the location 976: \n{df_loc_group[df_loc_group.index==976]}")
```

	mean	median	size
location			
976	327.693084	250.0	107

Fig-23: BA of location(2)

BA of size

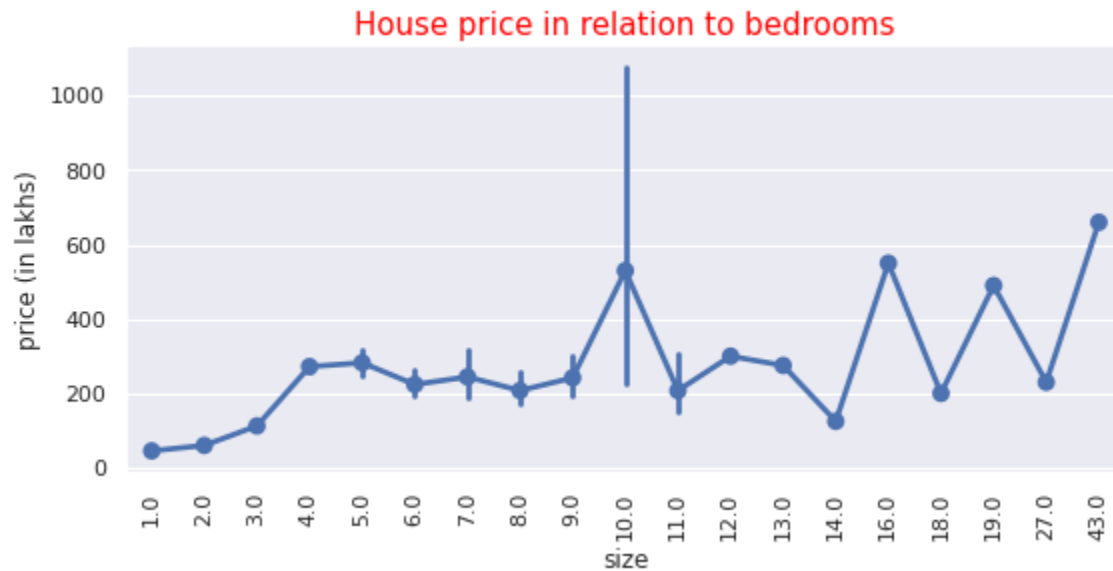


Fig-24: BA of size

Observation:

- Although a house with 10 rooms is quite uncommon, the graph depicting room size with 10 has the median. Additionally, the graph shows that more than 7/8 of the rooms have very erratic attitudes.

BA of bath

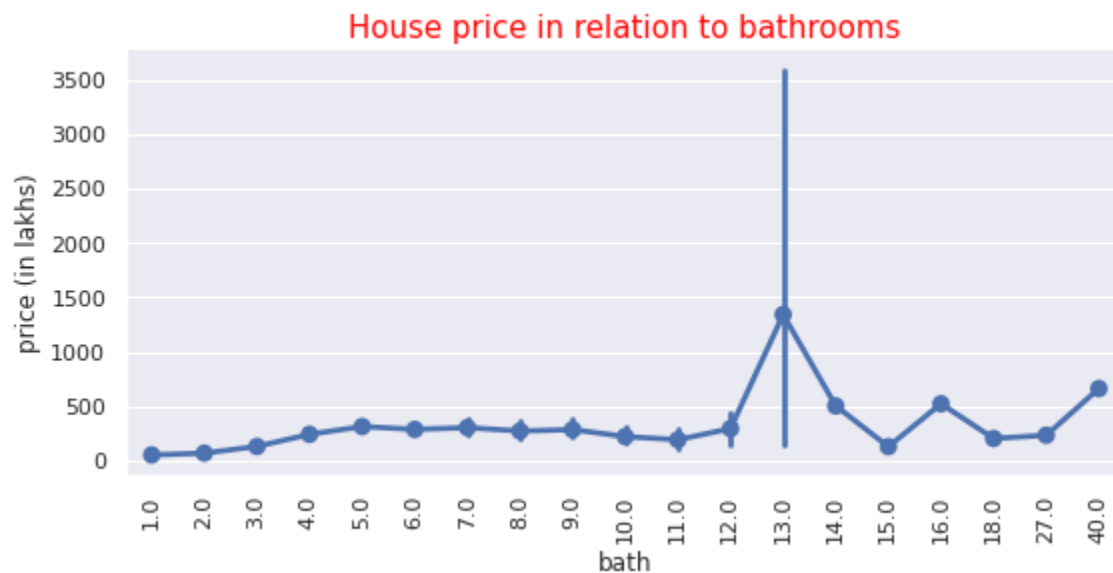


Fig-25: BA of bath

Observation:

- Bathrooms with 13 has very extreme value

BA of balcony

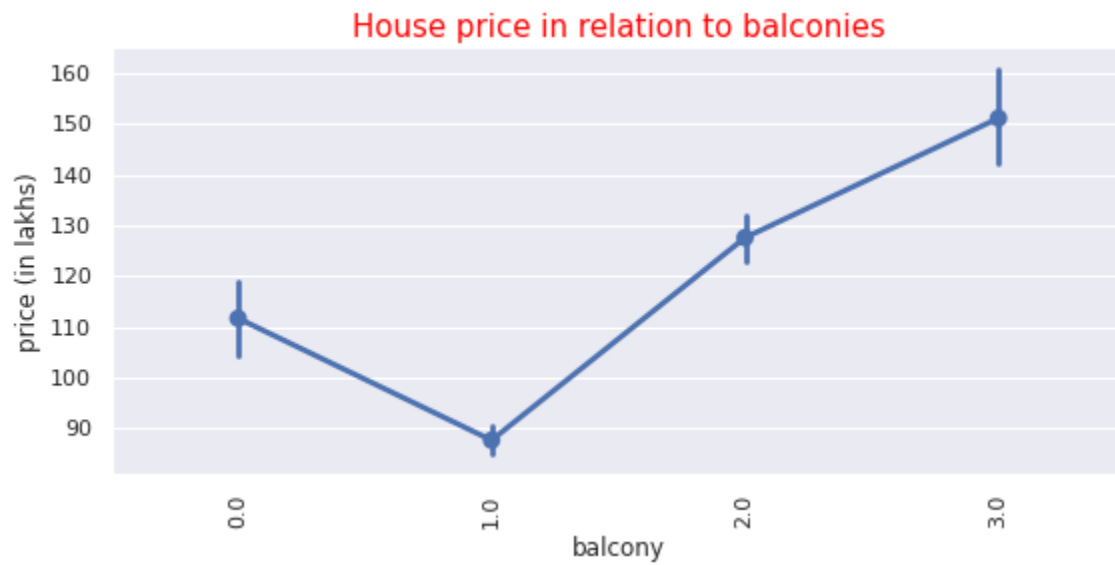


Fig-26: BA of balcony

Observation:

- Balcony has positive correlation with the price which is logical

BA of total_sqft

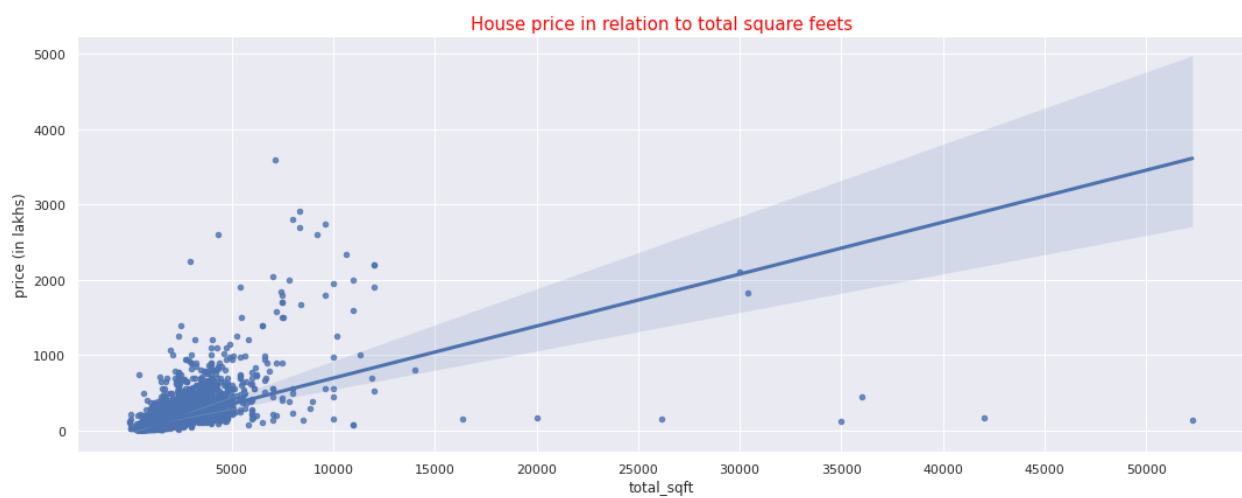


Fig-27: BA of total_sqft

Observation:

- It can be seen, the price of housing increases as the square footage increases
- Most of the house prices lie within 1 crore and 5 thousands square feet and it seems there are some outliers in the total square feet.

4. Data Cleaning & Pre-processing

The process of correcting missing information and deleting inaccurate or unnecessary material from a data set is known as data cleaning. The most crucial preprocessing stage is period cleaning because it guarantees that our data is prepared for our downstream requirements. Data transformation will start the process of transforming the data into the appropriate format(s) you'll need for analysis and other downstream operations. Data cleaning has already started the process of modifying our data.

The approach used for identifying and treating missing values, variable transformation, duplicate values and outlier treatment:-

Missing value:

If there are missing values in the dataset, many machine learning techniques fall short. Algorithms like Naive Bayes and K-nearest, however, support data with missing values. If the missing values are not handled properly, we risk creating a biased machine learning model that produces false results.

```

1 print(f"Checking the null values: \n{df.isnull().sum()}")

Checking the null values:
area_type      0
location       0
size          16
society        0
total_sqft     46
bath           73
balcony       609
price (in lakhs)  0
dtype: int64

```

Fig-28: Code snap of Checking the missing values

As we can see, there are a lot of values missing from that column balcony. So, our next step is to figure out how to deal with a lot of missing values. One strategy is to delete the column if we decide not to use it for further investigation. In that case, we would use a method called fillna, which is a predefined function in the Pandas.

```

1 def filling_null_mode(data):
2     data=data.fillna(data.mode()[0])
3     return data

1 df[['size']]=df[['size']].apply(filling_null_mode)
2 df[['bath']]=df[['bath']].apply(filling_null_mode)
3 df[['balcony']]=df[['balcony']].apply(filling_null_mode)
4

1 df[['total_sqft']]=df[['total_sqft']].fillna(df[['total_sqft']].mean()[0])
2 print(f"Checking the null values: \n{df.isnull().sum()}")

Checking the null values:
area_type      0
location       0
size           0
society        0
total_sqft     0
bath           0
balcony        0
price (in lakhs)  0
dtype: int64

```

Fig-29: Code snap of handling the missing data

Using the mean for a numerical variable and the mode for a categorical variable, respectively, we filled in the missing values. We have a method for identifying and dealing with missing values that works like this.

Variable Transformation:

Using variable transformation, we can improve the way the data behaves in your model. For this we clean the size column values which give information about the total number of rooms in the house. It was string and we used regular expressions to clean this. Then we transformed the column into float. We also transformed the numerical object data types to float also. We label encoded the categorical variables.

```
df['size']=df['size'].apply(lambda x: re.sub(r'BHK', ' ', str(x)))
df['size']=df['size'].apply(lambda x: re.sub(r'Bedroom', ' ', str(x)))
df['size']=df['size'].apply(lambda x: re.sub(r'RK', ' ', str(x)))
```

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

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

df[['area_type', 'location', 'society']]=df[['area_type', 'location', 'society']].apply(le.fit_transform)
df.head()
```

Fig-30: Code snapt of variable transformation

Treating the duplicate values:

Any record that unintentionally shares data with another record in a database is considered to have duplicate data. When identical items are not all in the same set, the split between the train, validation, and test sets might be ruined by duplicate entries. This may result in inaccurate performance predictions that let the model down when it comes to actual results. We have to drop the duplicate data. There were 652 duplicated data.

```

1 print(f"Data shape before dropping the duplicate rows: {df.shape}\n")
2 df.drop_duplicates(inplace=True, ignore_index=True)
3 duplicate=df[df.duplicated()]
4 df.reset_index(drop=True, inplace=True)
5 print(f"Duplicate Rows: \n\n {duplicate}\n\n and data shape: {df.shape}")

Data shape before dropping the duplicate rows: (13320, 8)

Duplicate Rows:

Empty DataFrame
Columns: [area_type, location, size, society, total_sqft, bath, balcony, price (in lakhs)]
Index: []

and data shape: (12668, 8)

```

Fig-31: Code snap of handling duplicate data

Outlier detection and handling:

A single data point that differs significantly from other points in the dataset is known as an outlier. It is a dataset anomaly that could have been brought about by a variety of mistakes in the collection, processing, or manipulation of data. Because outliers can distort general data trends, outlier detection techniques are crucial in statistics. In any area where data is used to inform choices, outliers will be taken into account. Outliers present a genuine risk to a company that is using data to obtain insight.

Following the execution of univariate and bivariate analyses, we can gain insight into the outliers present in our dataset. We deleted those outliers manually from the dataset. After removing those outliers the rows of the dataset decreased to 10857 from 12668. Here another method which is standard deviation method also used to detect the outliers. The cut-off for detecting outliers as greater than 3 standard deviations from the mean can be determined by first calculating the mean and standard deviation of a particular sample.


```

1 def outlier_handling(data, col):
2     mean, std=data[col].mean(), data[col].std()
3     print(f'Mean of {col}:-> {mean} and standard deviation of {col}:-> {std}\n')
4     cut_off=std*3
5     lower_cut_off, upper_cut_off=mean-cut_off, mean+cut_off
6     lower_cut_off, upper_cut_off
7     print(f"Lower bound and upper bound are:-> {lower_cut_off, upper_cut_off}\n")
8     outliers=[x for x in data[col] if x<lower_cut_off or x>upper_cut_off]
9     print(f"Number of outliers in the {col}: {len(outliers)}\n")
10    print(f"Shape before removing outliers from the {col} column:-> {data.shape}\n")
11    data=data.drop(data[(data[col] > upper_cut_off) | (data[col] < lower_cut_off)].index)
12    data.reset_index(drop=True, inplace=True)
13    print(f"Shape after removing outliers from the {col} column:-> {data.shape}")
14    return data

```

Fig-32: Code snap of outliers handling

After removing those outliers from every column, data decreased to 10507 from 10857.

5. Modeling

After successful data preprocessing, model building comes into action. Creating methods for data collection, comprehending and paying attention to what is significant in the data to address the questions you are posing, and creating a statistical, mathematical, or simulation model to acquire understanding and make predictions are all part of the model-building process. Algorithms for cluster analysis and regression models of machine learning are applied to modeling.

Cluster analysis

It is an unsupervised learning approach, therefore prior to running the model, you are unsure of how many clusters are present in the data. Contrary to many other statistical techniques, cluster analysis is frequently employed when no assumptions are made regarding the probable relationships among the data. Although it tells us where relationships and patterns in the data are, it doesn't explain what they might be or what they mean.

For cluster analysis, I used k means clustering and hierarchical clustering using dendrogram. First I normalized the data so that all the data are in similar scales. I converted the data between 0 to 1.

```
1 print(f"Before scaling: \n{df_cluster.head()}\n")
2
3 df_cluster=df_cluster/df_cluster.max()
4
5 print(f"After scaling: \n{df_cluster.head()}")
```

Fig-33: Code snapt of data scaling

Then I used an elbow plot to find the possible number of clusters. I visualized using an elbow plot. I used the silhouette analysis and sum of squared distance to search for the optimal number of clusters. I created the plot for 7, 10, 20, 40 and 100 clusters, respectively.

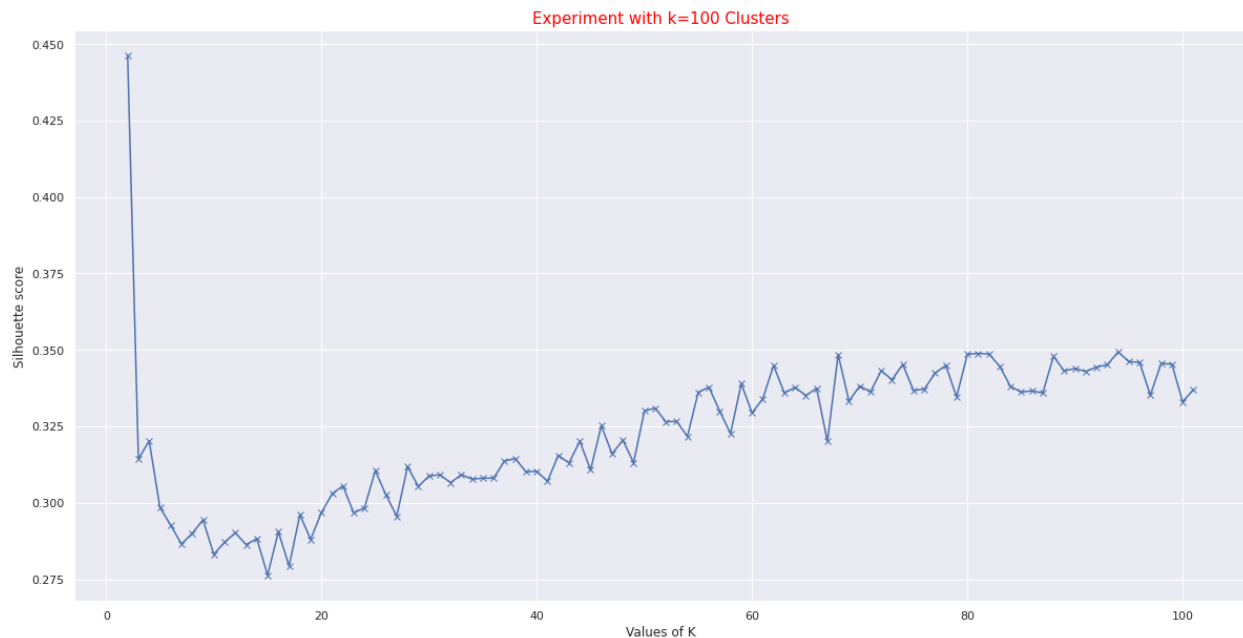


Fig-34: Elbow plot using Silhouette analysis for 100 clusters

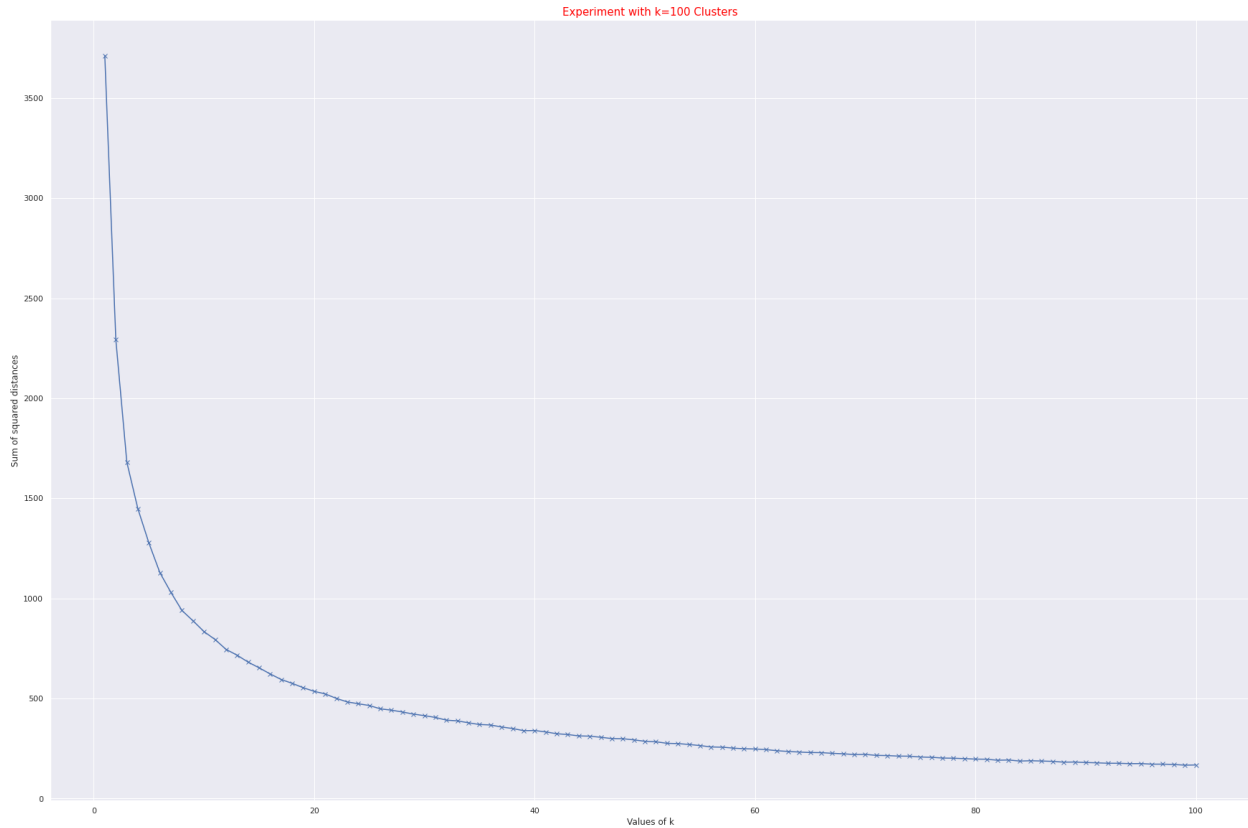


Fig-35: Elbow plot using Sum of Squared Distance for 100 clusters

I looked for up to 100 clusters. In the Silhouette approach, we can observe that the average score decreased up to 19 clusters but continued to increase beyond that. However, it only offers the highest score for 2 clusters, indicating that the clustering setup is adequate.

It is exceedingly challenging to determine the ideal number of clusters in SSD. It produces a large number of clusters. By doing experiments for 100 clusters, we can roughly estimate that the ssd graph is greatly flattened in 60 clusters. However, if we conduct experiments with up to 1000 clusters, we will find that the approximation is incorrect. As a result, we can conclude that the SSD provides a less accurate estimate than the Silhouette score.

```

1 def k_means(dataframe, k):
2     pca=PCA(n_components=2, random_state=0)
3     data=pca.fit_transform(dataframe)
4     kmeans=KMeans(n_clusters=k)
5     kmeans=kmeans.fit(data)
6
7     dataframe['Clusters']=kmeans.labels_
8     plt.figure(figsize=(25, 12.5))
9     sns.scatterplot(data[:, 0], data[:, 1], hue=dataframe['Clusters'])
10    plt.xticks(fontsize=15)
11    plt.yticks(fontsize=15)
12    plt.legend(bbox_to_anchor=(1.05, 0.6), fancybox=True, shadow=True, prop={'size': 20}, facecolor='white', edgecolor='black', title='Cluster:
13    plt.title(f"Clusters k={k}", fontsize=25, color='red')
14    plt.tight_layout()
15    plt.show()

```

Fig-36: Code snap of k mean clustering using Sklearn module

For cluster creation, I implemented the k means algorithm by implementing my own function in addition to using the k means method from the Sklearn package to create clusters. For hierarchical clustering, I also employed a dendrogram. For dimensionality reduction I used PCA.



Fig-37: Cluster creation with k=2

```

1 def my_kmeans(data,k, epochs):
2     """This function in for creating the kmeans algorithm from scratch.
3
4     Args:
5         data: Dataframe
6         k: No. of clusters
7         epochs: no. of iterations
8     Returns:
9         All the cluster points
10    """
11    idx=np.random.choice(len(data), k, replace=False)
12    centroids=data[idx, :]
13    distances=cdist(data, centroids , 'euclidean')
14    labels=np.array([np.argmin(i) for i in distances])
15    for _ in range(epochs):
16        centroids=[]
17        for idx in range(k):
18            cent_update=data[labels==idx].mean(axis=0)
19            centroids.append(cent_update)
20        centroids=np.vstack(centroids)
21        distances=cdist(data, centroids , 'euclidean')
22        labels=np.array([np.argmin(i) for i in distances])
23
24    return labels
25
26
27 def perform_my_kmeans(data, k):
28     """This function is for performing the implemented my_kmeans function.
29
30     Arg:
31         The dataframe
32     Returns:
33         Plots all the clusters and gets the unique cluster labels
34    """
35    pca=PCA(n_components=2, random_state=0)
36    data=pca.fit_transform(data)
37    labels=my_kmeans(data, k, 500)
38    unique_labels=np.unique(labels)
39    plt.figure(figsize=(25, 12.5))
40    for i in unique_labels:
41        plt.scatter(data[labels == i , 0] , data[labels == i , 1] , label = i, alpha=0.4)
42    plt.xticks(fontsize=15)
43    plt.yticks(fontsize=15)
44    plt.legend(bbox_to_anchor=(1.05, 0.6), fancybox=True, shadow=True, prop={'size': 20}, facecolor='white', edgecolor='black', title='Clusters')
45    plt.title(f"Clusters k={k}", fontsize=25, color='red')
46    plt.tight_layout()
47    plt.show()

```

Fig-38: Code snapt of k means clustering using own function

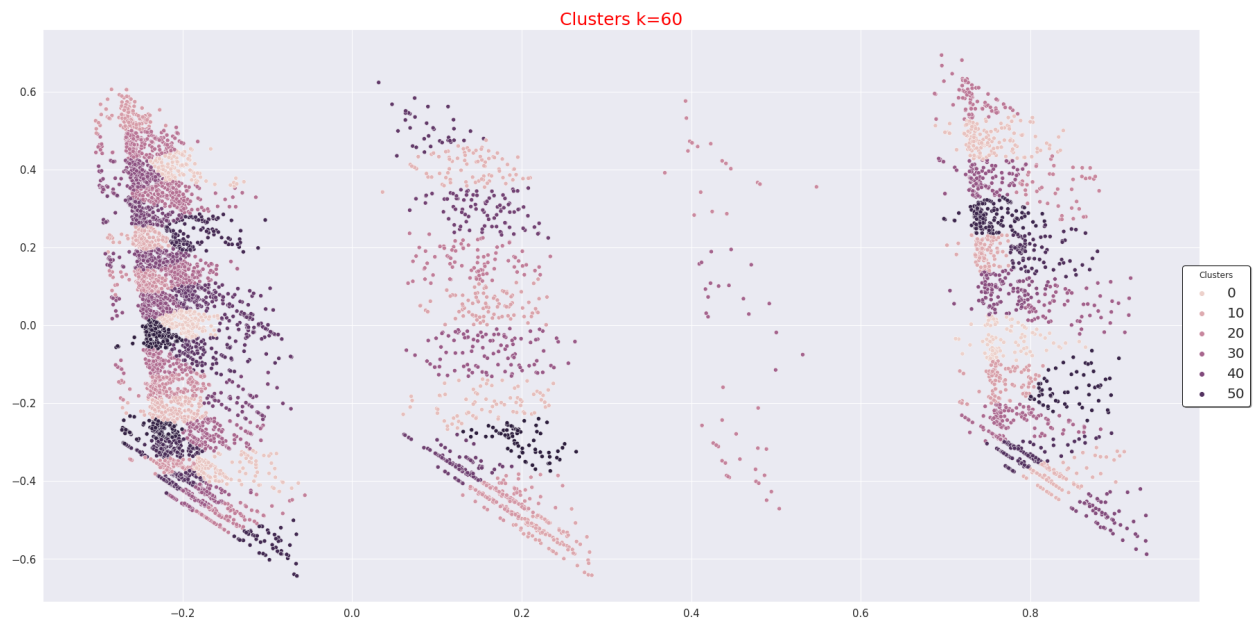


Fig-39: Cluster creation with k=60

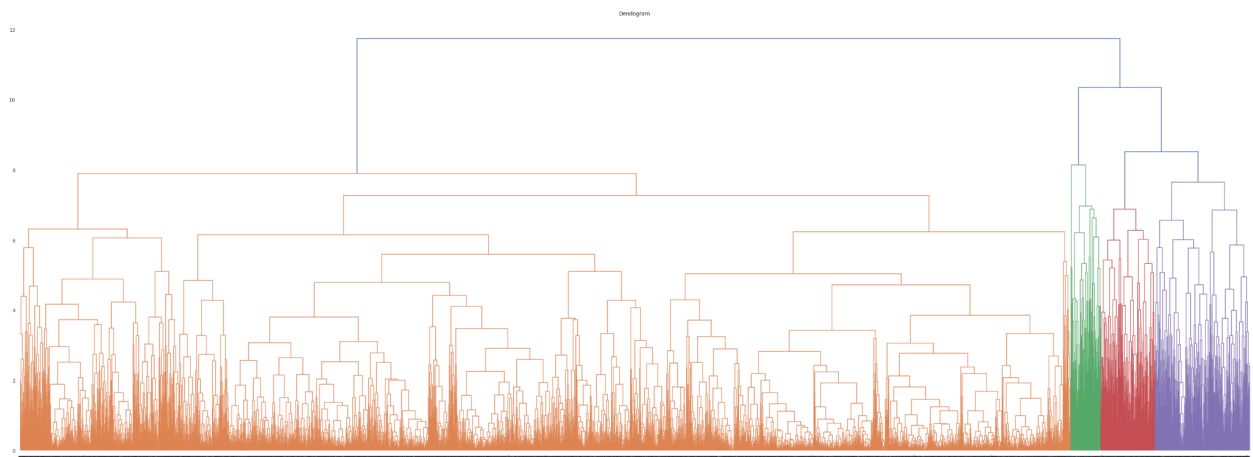


Fig-40: Cluster creation using dendrogram

The above dendrogram does not provide us useful information about the clusters. We can see that the Silhouette analysis determined that two clusters are the ideal number.

Machine Learning Algorithm

Artificial intelligence, which is widely defined as a machine's ability to mimic intelligent human behavior, includes the subfield of machine learning. Artificial intelligence (AI) systems are used to carry out complicated tasks in a manner akin to how people solve issues. It uses little to no human intervention and operates by examining data and recognizing patterns. We're now utilizing one such tool to forecast house prices. We need to first create a model and train it using historical data before making any predictions.

First, I separate the independent(property attributes) and dependent features(prices).

```
1 X=df_ml.drop('price (in lakhs)', axis=1)
2 y=df_ml['price (in lakhs)']
```

Fig-41: Code snapt of separation of independent and dependent variable

Then I checked the multicollinearity of the data. It is a technique for calculating the correlation between the independent variables. I checked the multicollinearity using two different approaches: pearson correlation coefficient and variance inflation factors.

For the Pearson correlation coefficient, I used a heatmap.

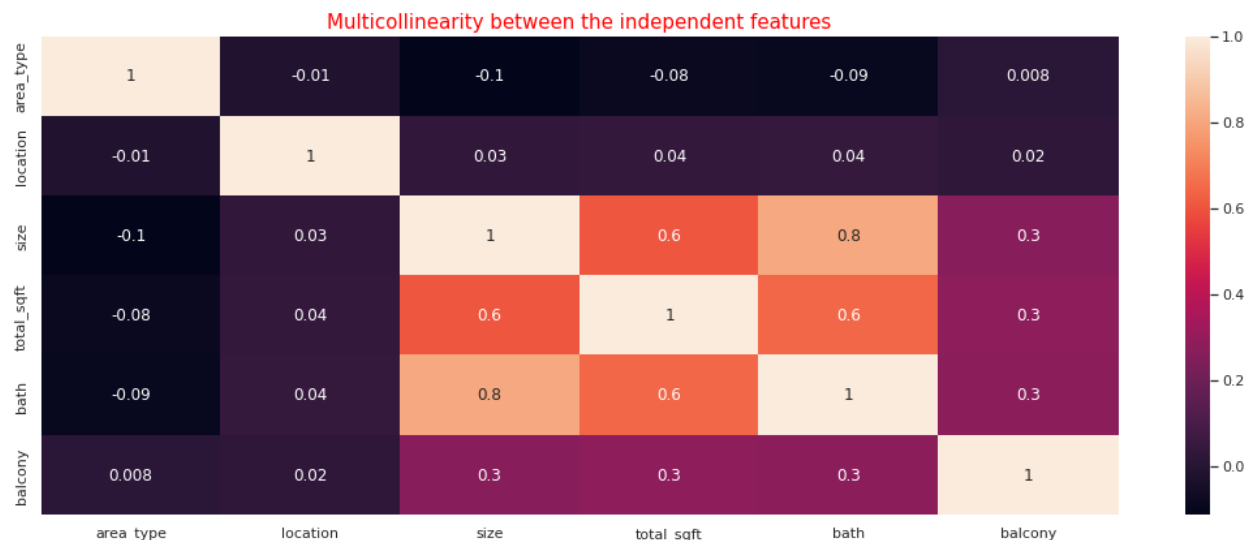


Fig-42: Heatmap of the independent variables

The heatmap shows that size and bath column are highly correlated with each other (0.80).

For VIF, I used a barplot to show the correlation between the features.

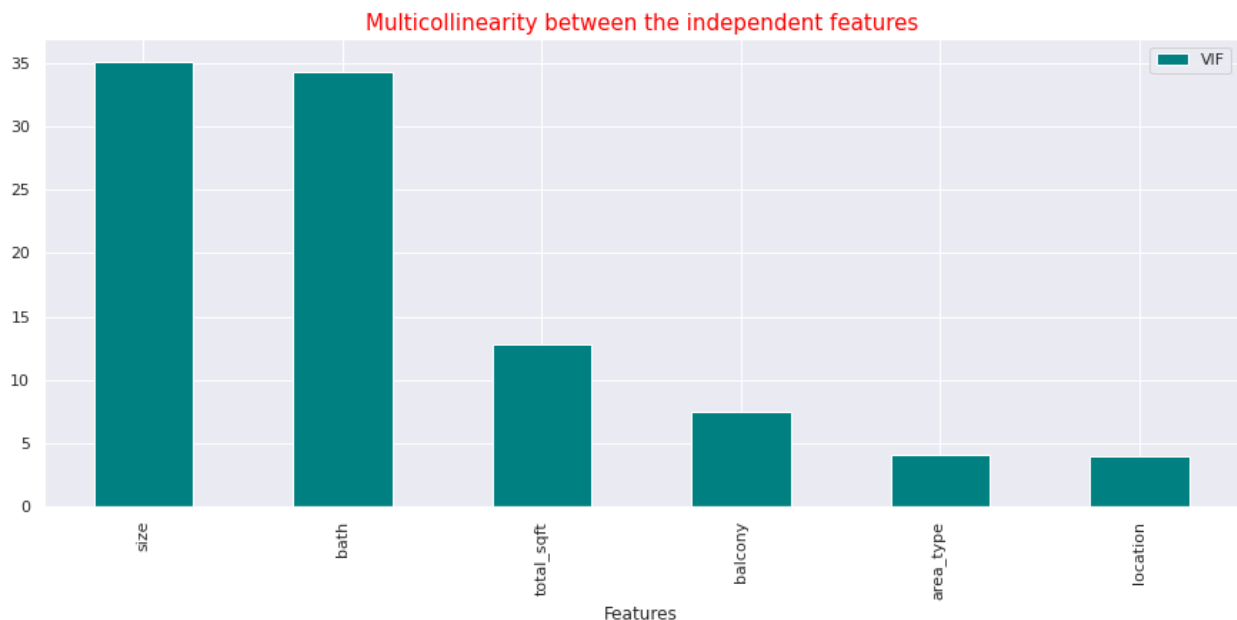


Fig-43: VIF between the independent variables

The plot in the above graph demonstrates the strong correlation between size and bath. As a result, since they are causing multicollinearity, we can eliminate any of them. The bath column can be removed because the size column, which indicates the number of rooms in the house, is more crucial for predicting the price of a home.

Thus we found our final features to predict the price of a house.

```
1 print(f"Final independent columns or the features for training: \n{X.columns}")  
  
Final independent columns or the features for training:  
Index(['area_type', 'location', 'size', 'total_sqft', 'balcony'], dtype='object')
```

Fig-44: Code snap of showing the final features

The pairplot shows that the data are cleaned and It has no outliers present in the data. The data is ready for building the machine learning models. For feature scaling I used the same method that I have used for the cluster analysis.

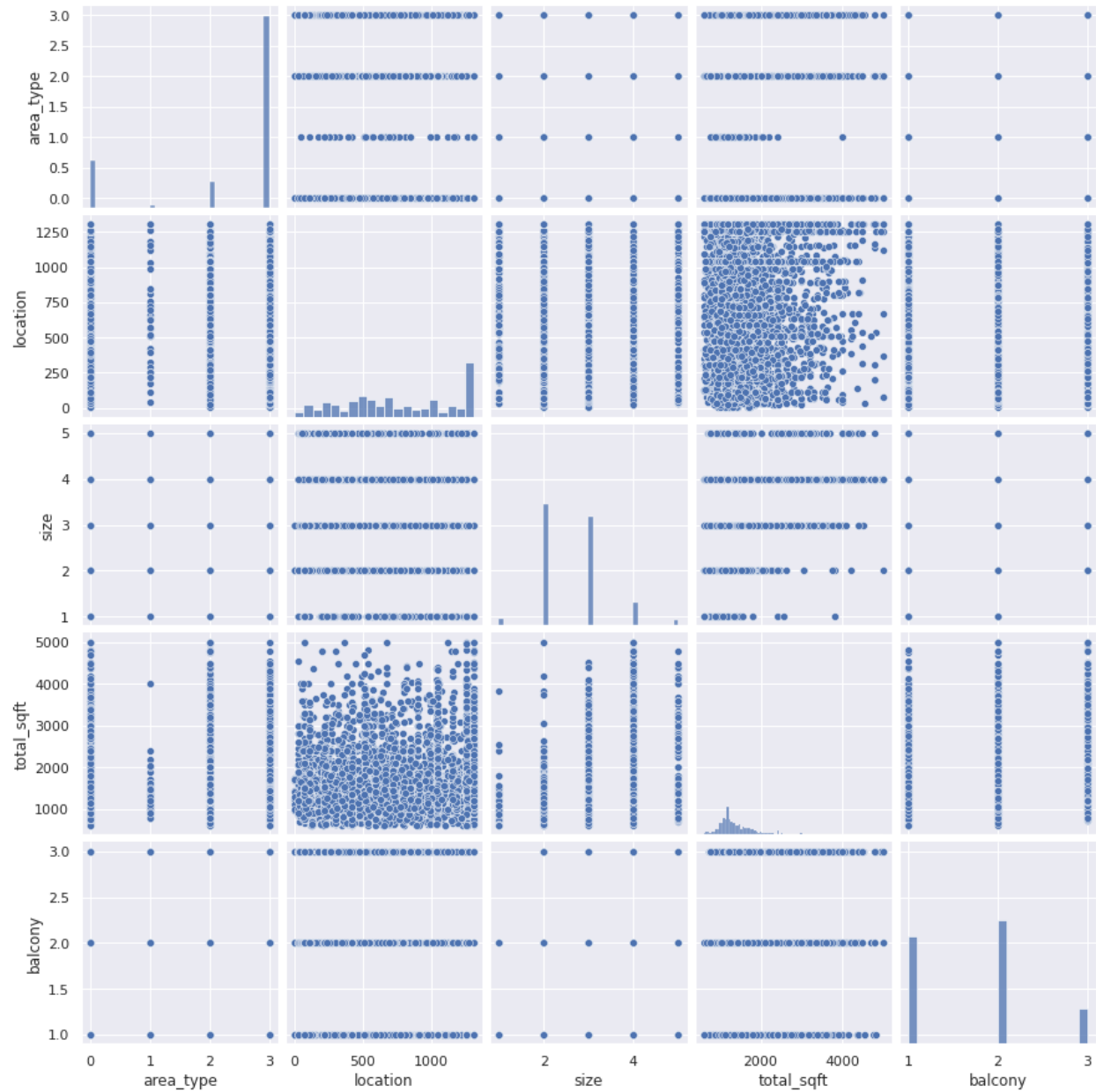


Fig-45: Pairplot of the final features

Then the data needs to be split into 3 sets:

1. Training set: A minimum of 60–75% of the total data we have should be in the training set, which is the portion of the dataset that the model will use to train itself. My training used 75% of the data.

2. Validation set: This collection is used to validate the performance of our model for various sets of hyperparameters. The remaining set can be divided into a validation set and a test set after the train set has been removed.
3. Test set: The test set is used to assess the model's performance on the unobserved data that it will use to make future predictions. Understanding the degree of error between actual and anticipated numbers is useful.

```
1 X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.25)
2 X_train, X_val, y_train, y_val=train_test_split(X_train, y_train, test_size=0.25)]

1 print(f'Training data size: {X_train.shape[0]}\n validation data size: {X_val.shape[0]}\n testing data size: {X_test.shape[0]}')
```

Training data size: 5906
validation data size: 1969
testing data size: 2626

Fig-46: Code snapt of data splitting

I built **15 regression algorithms** and using the validation set we can determine which model to keep for making final predictions. The complete list of built and evaluated algorithms are: **Simple Linear Regression, Lasso, Ridge, KNN, SVR with RBF, Simple Decision Tree, Logistic Regression, Gradient Boosting, Bangging, Random Forest, ADA Boost, XGBoost, Gradient Boosting by Grid Search, Gradient Boosting Tuning using Graph, Gradient Boosting using Randomized Search.**

Used Algorithms

Simple Linear Regression

The first model that I built is the simple linear regression. A regression model called simple linear regression uses a straight line to calculate the association between one independent variable and one dependent variable.

Lasso:

Then I used lasso regression. Shrinkage is used in the linear regression method known as lasso regression. When data values shrink toward a middle value, such as the mean, this is called shrinkage. Simple, sparse models are encouraged by the lasso approach (i.e. models with fewer parameters).

Ridge:

Any data that shows multicollinearity can be analyzed using the model tuning technique known as ridge regression. This technique carries out L2 regularization. Predicted values differ much from real values when the problem of multicollinearity arises, least-squares are unbiased, and variances are significant.

KNN:

One of the simplest machine learning algorithms, based on the supervised learning method, is K-Nearest Neighbour. The K-NN algorithm makes the assumption that the new case and the existing cases are comparable, and it places the new instance in the category that is most like the existing categories.

SVR with RBF Kernel:

An approach for supervised learning called support vector regression is used to forecast continuous values. The SVMs and Support Vector Regression both operate on the same theory. Finding the optimum fit line is the fundamental tenet of SVR. The hyperplane with the most points is the best-fitting line in SVR.

The radial basis function kernel, or RBF kernel, is a well-liked kernel function in machine learning that is utilized in a variety of kernelized learning techniques. It is frequently employed in support vector machine classification in particular.

Simple Decision Tree

A non-parametric supervised learning technique for classification and regression is called a decision tree (DT). The objective is to learn straightforward decision rules derived from the data features in order to build a model that predicts the value of a target variable. A piecewise constant approximation of a tree can be thought of.

Logistic Regression

A statistical analysis method called logistic regression uses previous observations from a data set to predict a binary outcome, such as yes or no. By examining the correlation between one or more already present independent variables, a logistic regression model forecasts a dependent data variable. A logistic regression model makes mathematical predictions about $P(Y=1)$ as a function of X . One of the most basic

machine learning algorithms, it may be applied to a number of categorization issues, including spam identification, diabetes prediction, cancer diagnosis, etc.

Gradient Boosting

One kind of machine learning boosting is gradient boosting. It is predicated on the hunch that when prior models are coupled with the best possible upcoming model, the overall prediction error is minimized. Setting the desired results for this subsequent model in order to reduce mistakes is the important concept.

Bagging

An ensemble learning technique called bagging, often referred to as Bootstrap aggregating, aids in enhancing the efficiency and precision of machine learning algorithms. It lowers the variance of a prediction model and is used to handle bias-variance trade-offs. Bagging, specifically decision tree methods, is used for both regression and classification models to prevent overfitting of the data.

Random Forest

A large number of decision trees are built during the training phase of the random forests or random decision forests ensemble learning approach, which is used for classification, regression, and other tasks. The class that the majority of the trees chose is the output of the random forest for classification problems. With more explanatory factors in a dataset, random forest has a higher true and false positive rate.

AdaBoost

AdaBoost, also known as Adaptive Boosting, is a machine learning method used in an ensemble setting. Decision trees with one level, or Decision trees with only one split, are the most popular algorithm used with AdaBoost. Another name for these trees is Decision Stumps. The abbreviation "AdaBoost" stands for "Adaptive Boosting," which is a very well-liked boosting approach that turns several "poor classifiers" into one "strong classifier."

XGBoost

Extreme Gradient Boosting (XGBoost) is a distributed, scalable gradient-boosted decision tree (GBDT) machine learning framework. The top machine learning library for regression, classification, and ranking issues, it offers parallel tree boosting.

Comprehending the machine learning ideas and techniques that supervised machine learning, decision trees, ensemble learning, and gradient boosting are built upon is essential to understanding XGBoost.

Grid Search CV

The tool used for hyperparameter optimization is called GridSearch. As previously mentioned, machine learning in practice entails comparing various models to one another in an effort to identify the most effective model. Grid Search assesses the performance for each possible combination of the hyperparameters and their values, chooses the combination with the best performance, and takes that combination as its starting point. With so many hyperparameters involved, processing becomes time-consuming and expensive.

Randomized Search CV

It tries a variety of random combinations of values (we have to define the number of iterations). It excels at testing a wide range of values and typically comes up with excellent combinations quickly, but the drawback is that it cannot guarantee to provide the best parameter combination because not all parameter values are tested (recommended for large datasets or a high number of parameters to tune).

6. Result and Performance

Performance Metrics

For evaluation of the models. R^2 score, mean squared error (mse), root mean squared error (rmse), mean absolute error (mae), mean absolute percentage error (mape) and adjusted R^2 score are used. The main performance metrics that are used in this problem are train, validation and test score.

R² score

When assessing the effectiveness of a machine learning model based on regression, the R² score is a crucial indicator. It is also referred to as the coefficient of determination and is pronounced as R squared. It operates by calculating the variation in the predictions that the dataset can explain. In the world of finance, an R-Squared value above 0.7 is typically seen as indicating a high level of correlation, whereas one below 0.4 indicates a low level of correlation.

Mean Squared Error(MSE)

How closely a regression line resembles a set of data points is determined by the Mean Squared Error. It is a risk function that corresponds to the squared error loss's expected value. The average, more particularly the mean, of errors squared from data related to a function is used to determine mean square error. For MSE, there is no ideal value. In other words, the lower the value, the better, and 0 denotes a perfect model.

Root Mean Squared Error(RMSE)

One of the methods most frequently used to assess the accuracy of forecasts is root mean square error, also known as root mean square deviation. It illustrates the Euclidean distance between measured true values and forecasts. Better fit is shown by lower RMSE values. A decent indicator of how effectively the model predicts the reaction is the RMSE.

Mean Absolute Error(MAE)

Without adjusting for their tendency, MAE calculates the average magnitude of the mistakes in a set of forecasts. It represents the weighted average of the individual deviations between the actual observation and the forecast over the test sample. The model is more accurate the closer MAE is to 0. There isn't a set standard for what a good score is because MAE is returned on the same scale as the target you are projecting for.

Mean Absolute Percentage Error(MAPE)

The mean absolute percentage error (MAPE) assesses how accurately a company's forecasting process performed. It shows, on average, how accurate the anticipated

quantities were in relation to the actual amounts by averaging the absolute percentage errors of each entry in a dataset. A MAPE of less than 5% is regarded as a sign that the forecast is within acceptable bounds of accuracy. If the MAPE is greater than 10% but less than 25%, it shows low accuracy that is still acceptable, and if it is greater than 25%, it indicates very low precision that is so low that the forecast's accuracy is unacceptably low.

Adjusted R^2 score

A variant of R-squared that has been changed to account for the number of predictors in the model is known as adjusted R-squared. When the additional term enhances the model more than would be predicted by chance, the adjusted R-squared rises. When a predictor enhances the model by less than anticipated, it falls. Other disciplines may have substantially higher requirements for a respectable R-Squared reading, such as 0.9 or higher. In the world of finance, an R-Squared value above 0.7 is typically seen as indicating a high level of correlation, whereas one below 0.4 indicates a low level of correlation.

Result Analysis for Individual Models

Simple Linear Regression

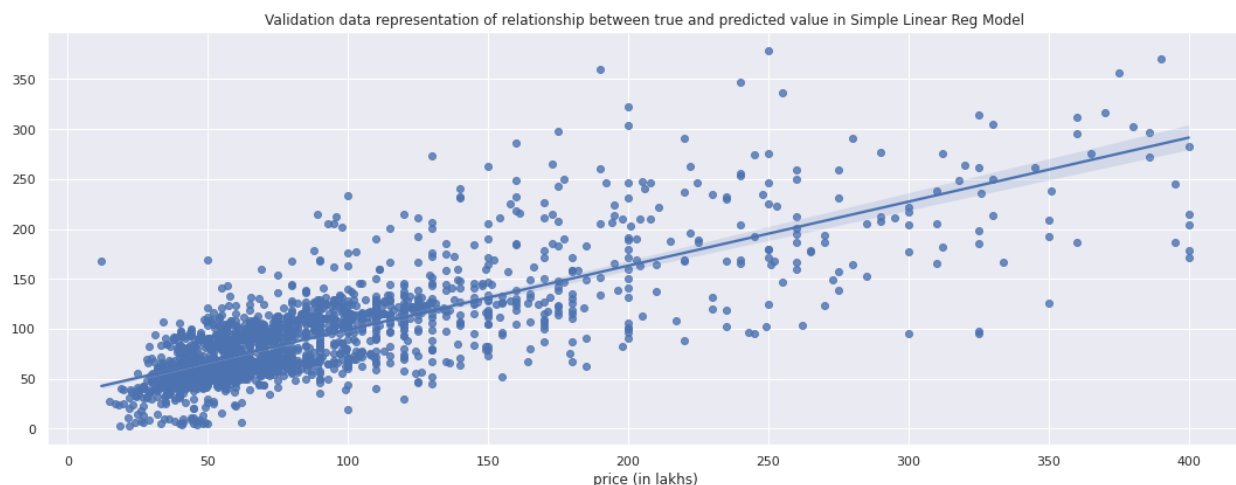


Fig-47: Regression plot of Simple Linear Regression

Simple linear regression given 58% accuracy in training data, and 61% & 57% for validation and test data, respectively. This is not a satisfying result. We have to move on the other linear regressions like lasso and ridge.

Lasso

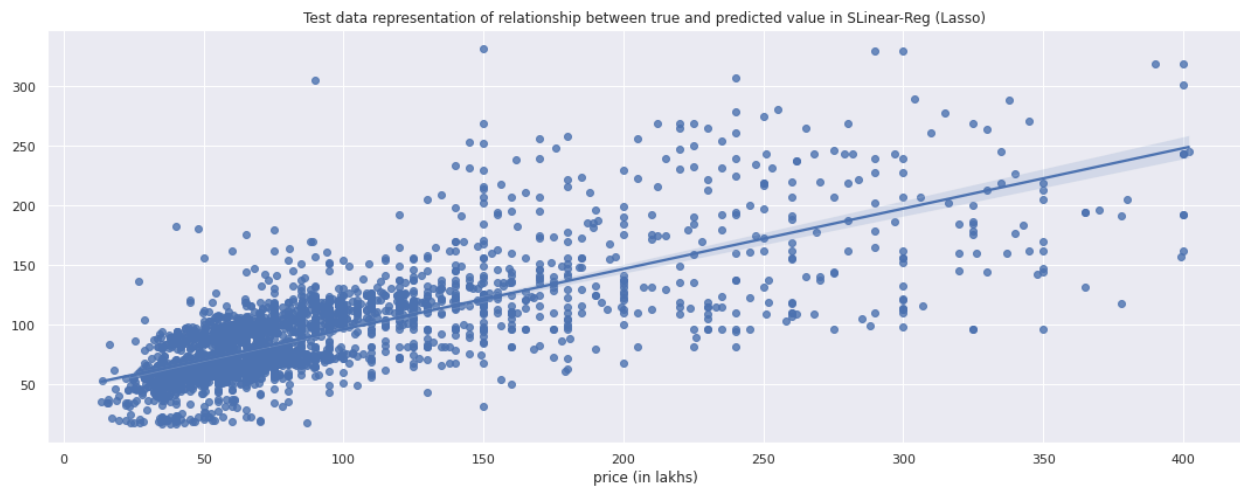


Fig-48: Regression plot of Lasso Regression

Lasso regression given 56% training accuracy, 60% validation accuracy and 56% test accuracy. But the model gives slightly worse performance than a simple linear regression model.

Ridge

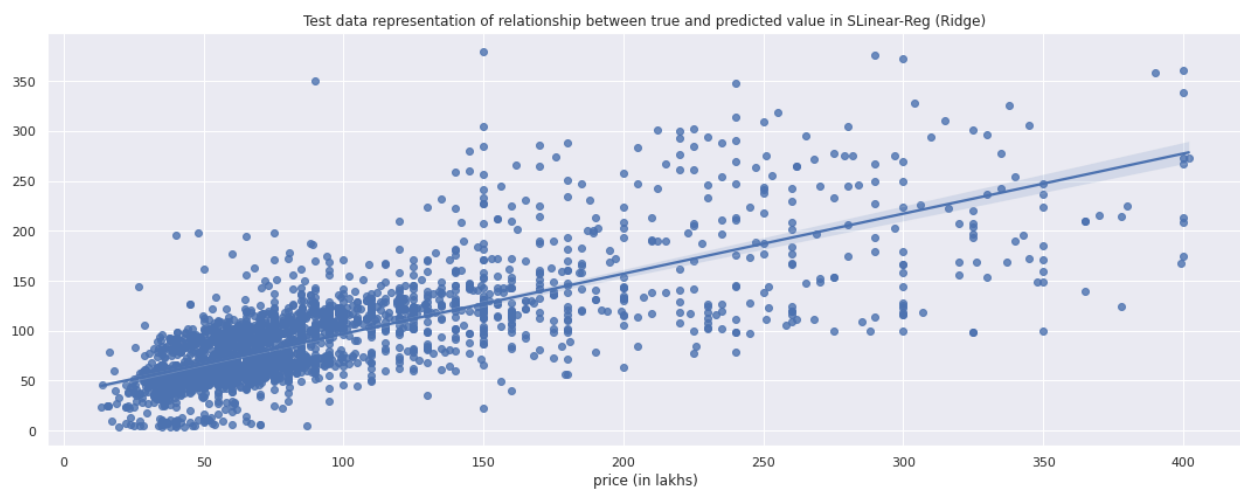


Fig-49: Regression plot of Ridge Regression

It gives 58%, 61% and 57% accuracy for train, validation and test set, respectively. The model gives almost the same performance as simple linear regression. So now we have to move on to more advanced models.

KNN

KNN model given 98% training accuracy but 68% and 66% accuracy for validation and test set, respectively.

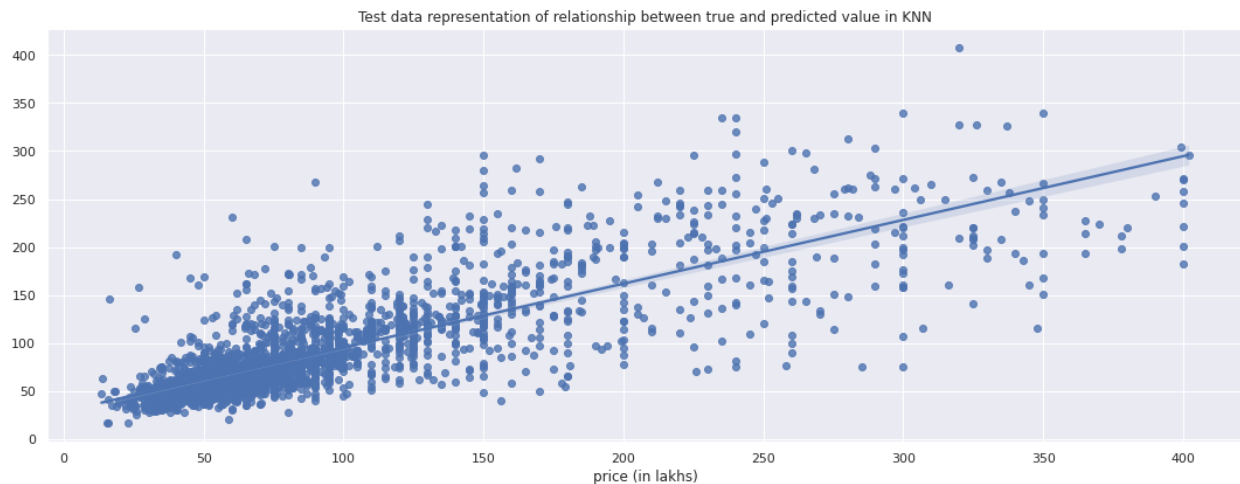


Fig-50: Regression plot of KNN

Although the KNN model's accuracy is better to that of simple linear and ridge and lasso regression, it performs poorly on test and validation sets of data, indicating that it overfits the data since it generalizes well on train data.

SVR with RBF Kernel

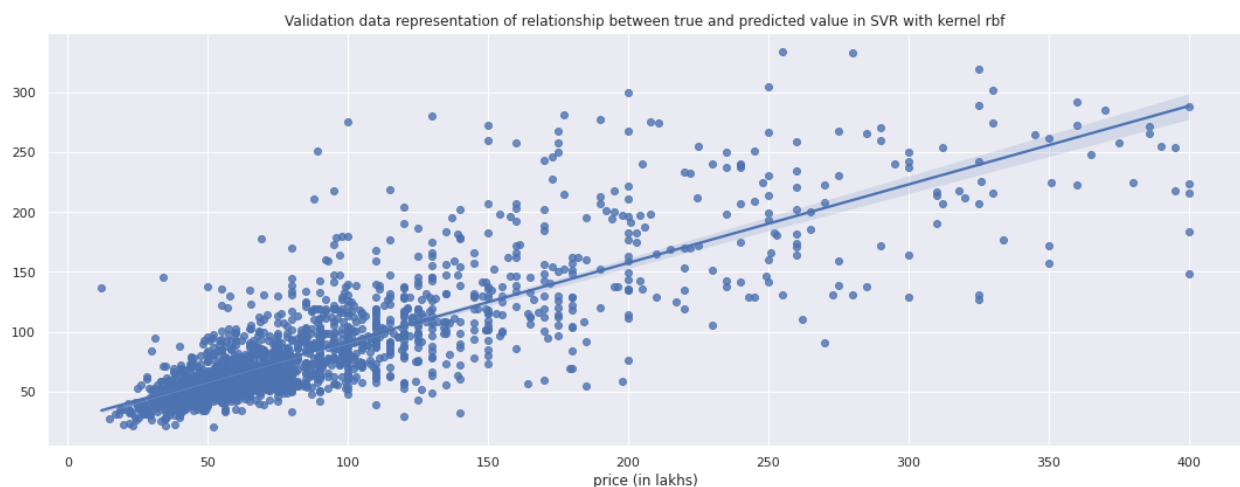


Fig-51: Regression plot of SVR with RBF

The SVR model given 64% train, 67% validation and 62% test accuracy. It performs better than the previous models and does not overfit. But the accuracy is still low. So now we've to go on the tree based models.

Simple Decision Tree

It gave 98% training accuracy but 49% validation accuracy and 47% test accuracy. Simple decision tree models overfit and fail completely compared to other models. It has a less than 50% accuracy rate.

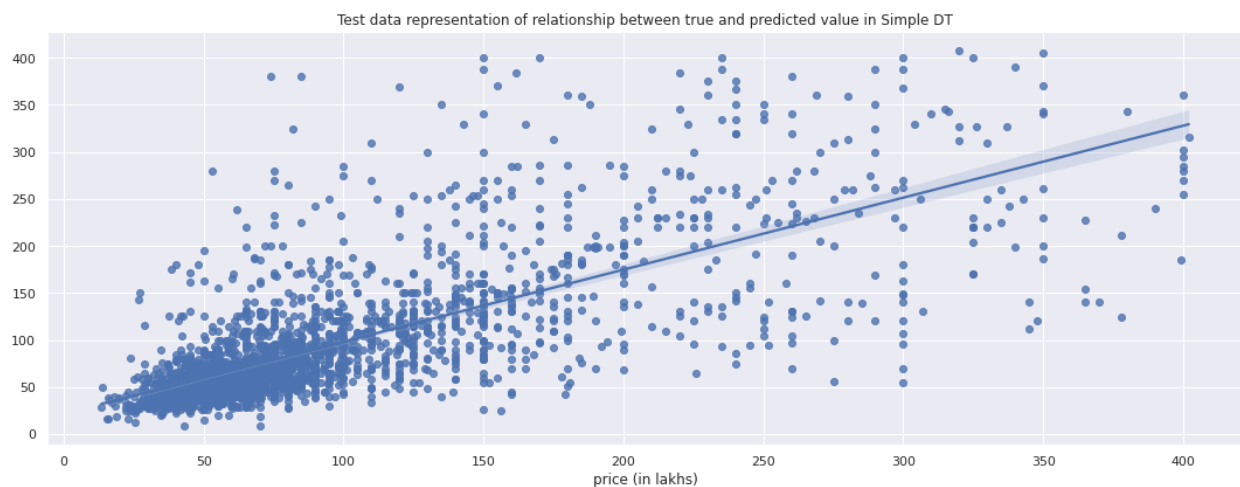


Fig-52: Regression plot of Simple Decision Tree

Logistic Regression

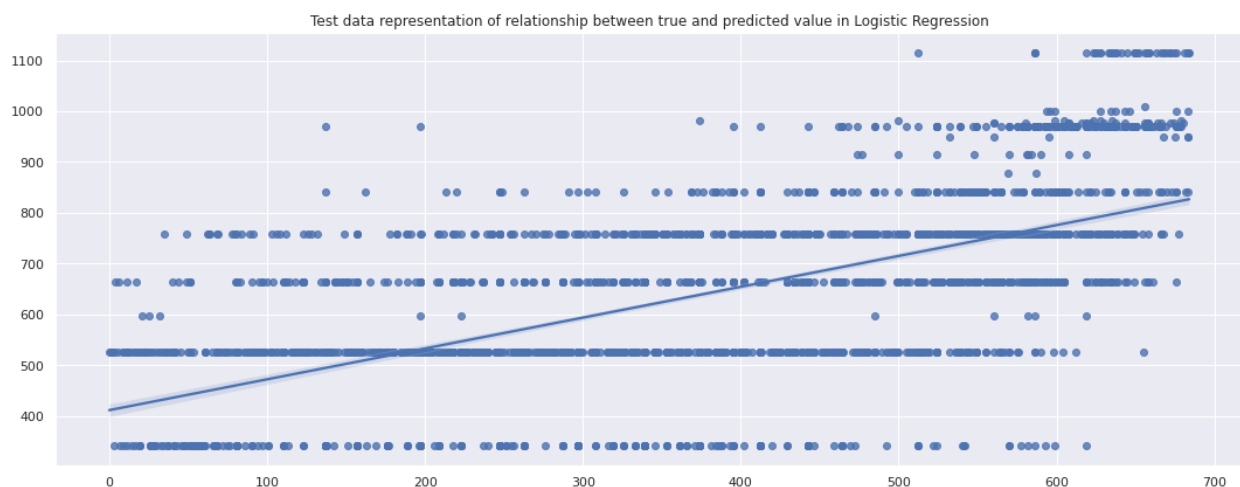


Fig-53: Regression plot of Logistic Regression

Logistic Regression given very poor performance with only 33% training accuracy. A classification problem is appropriate for logistic regression. However, since this is a

regression problem, its performance is very very poor, as expected. Now move on to the ensemble techniques.

Gradient Boosting

It gives good performance of 73% training accuracy with 72% validation accuracy and 69% test accuracy. Since gradient boosting is a tree-based ensemble technique, as expected, it outperforms all other models in terms of performance.

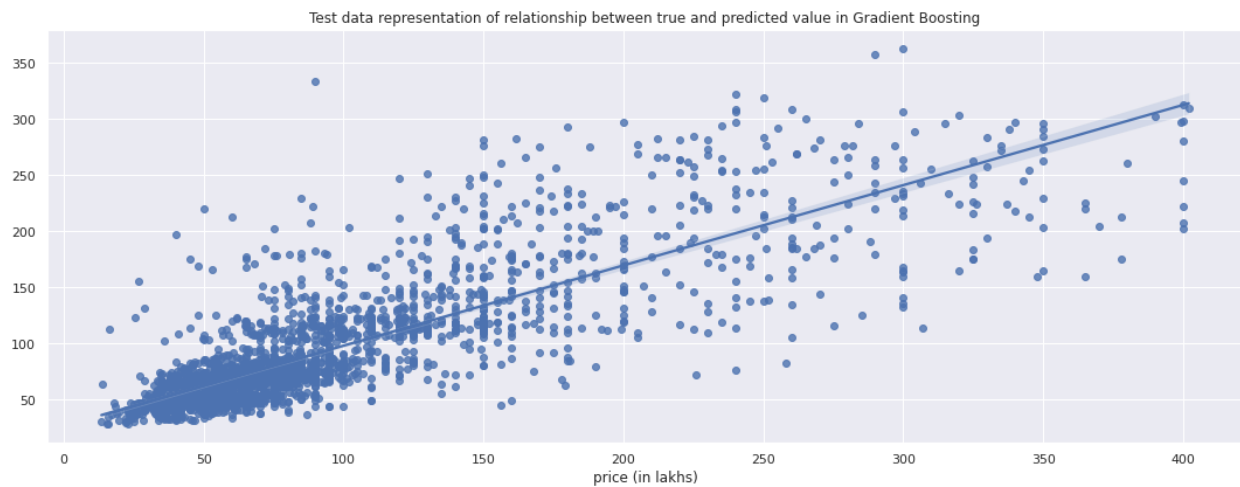


Fig-54: Regression plot of Gradient Boosting

Bagging

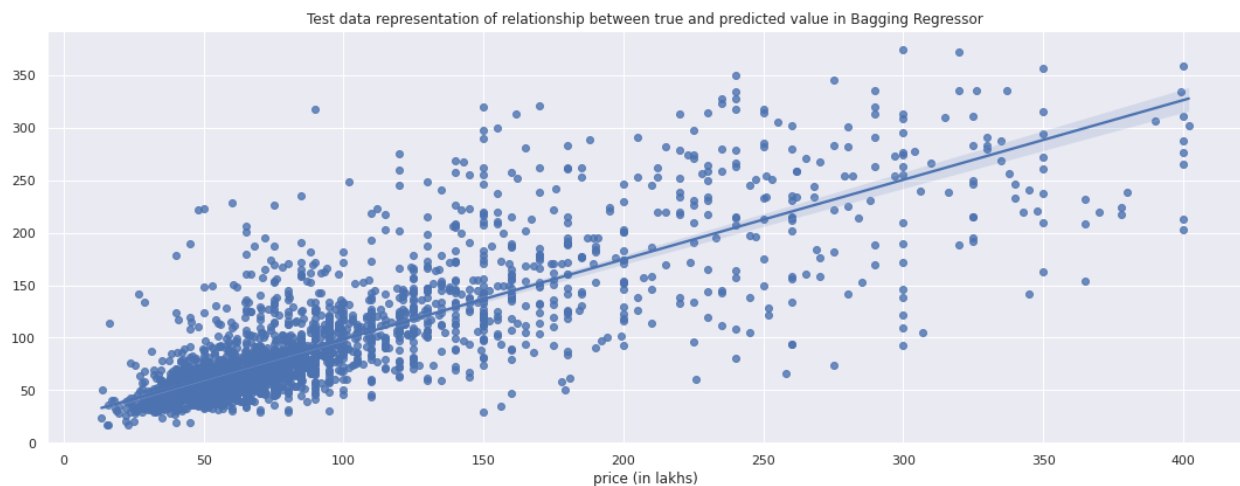


Fig-55: Regression plot of Bagging

Bagging overfits. This can be as a result of the decision tree estimator being used by default. A simple decision tree model also overfits the data. It has 93% training

accuracy, 69% validation accuracy and 66% test accuracy. But apart from this, hyperparameter tuning can solve this issue.

Random Forest

Random Forest given 98% training accuracy but 50% validation and 46% test accuracy. Additionally, random forest overfit the data. Similar to bagging, it also uses a decision tree as an estimator. But in addition to this, the models' hyperparameters need to be tuned.

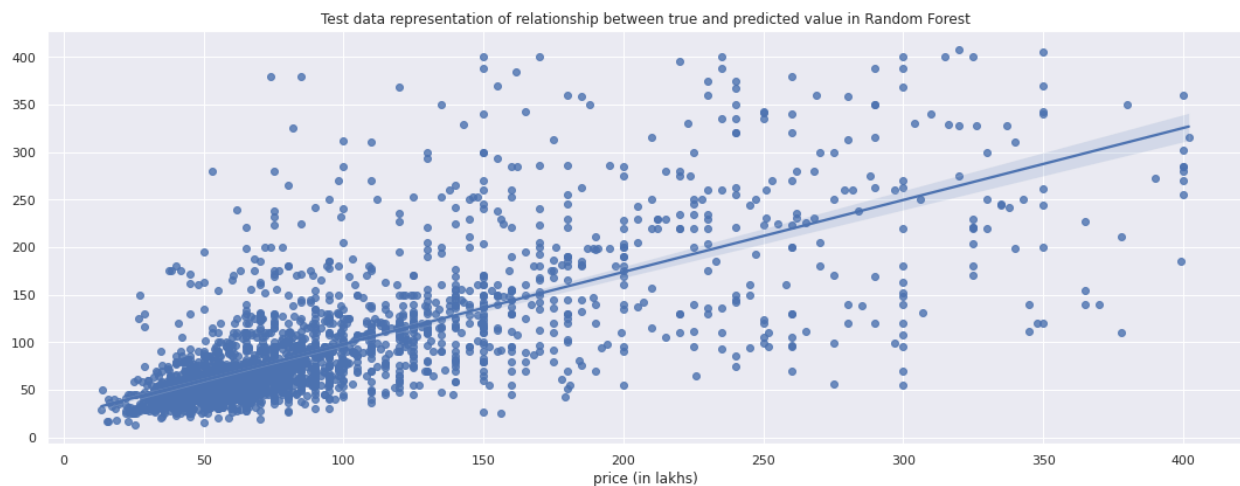


Fig-56: Regression plot of Random Forest

AdaBoost

Adaptive Gradient Boosting given 95%, 68% and 65%, train, validation and test accuracy, respectively. Adaboost also gives poor performance for the overfitting problem. Hyperparameter tuning is necessary.

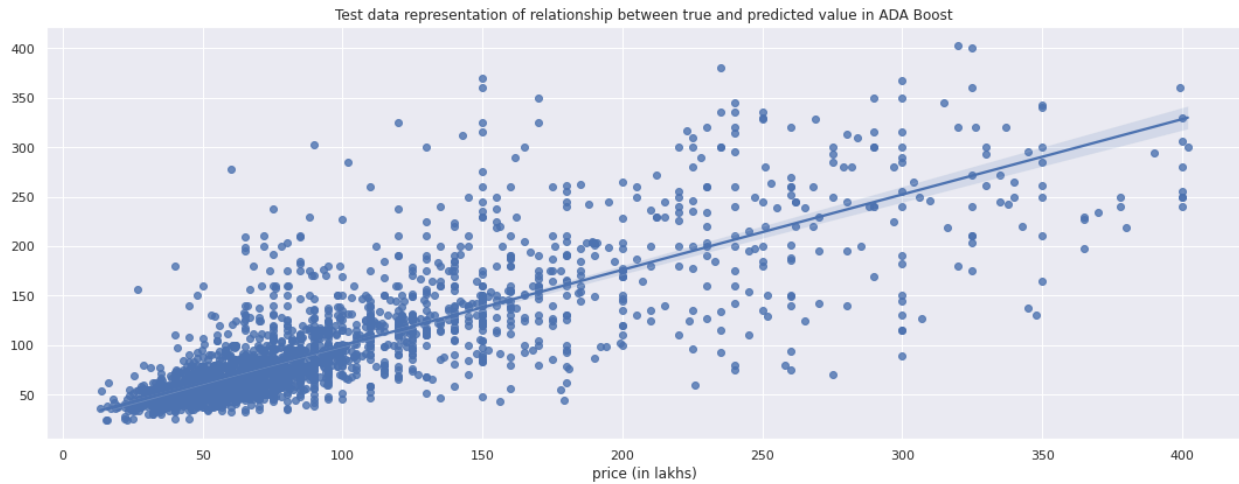


Fig-57: Regression plot of AdaBoost

XGBoost

Extreme Gradient Boosting model given 73% train, 72% validation and 69% test accuracy that means works well, although it is less accurate in terms of the test data. Hyperparameter optimization can resolve the problem. So we have to go on the hyperparameter tuning.

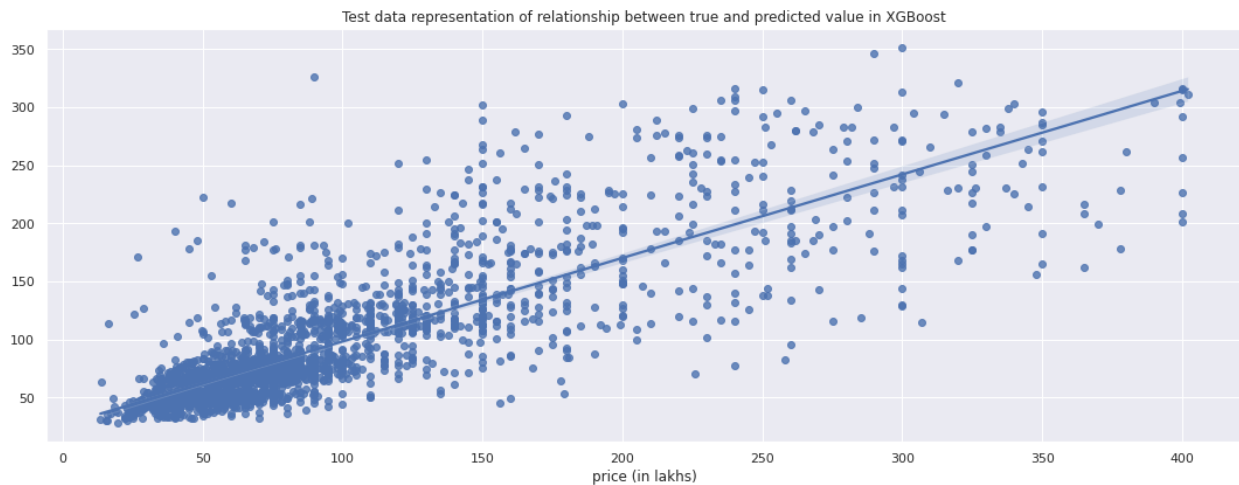


Fig-58: Regression plot of XGBoost

Gradient Boosting using Grid Search

Here the hyperparameters of the gradient boosting algorithms are tuned. So we have to use a parameter set.

```
1 params_grid={
2     'learning_rate': [0.1, 0.01, 1.0],
3     'max_depth': [5, 8, 10],
4     'min_samples_leaf': [5, 10],
5     'min_samples_split': [30, 50],
6     'n_estimators': [50, 150, 200],
7 }
8
```

Fig-59: Code snapt of assigning parameter grid of Gradient Boosting by Grid Search

The 10-fold cross validation is also used for the tuning. The tuned parameters are as follows:

```
Best parameters:
{'learning_rate': 0.1, 'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_split': 30, 'n_estimators': 200}
Best score: 0.7307745217061269
```

Fig-60: Tuned parameters of Gradient boosting using Grid Search

After building the model using these parameters, the performance is improved with 83% training accuracy, 75% validation accuracy and 71% test accuracy.

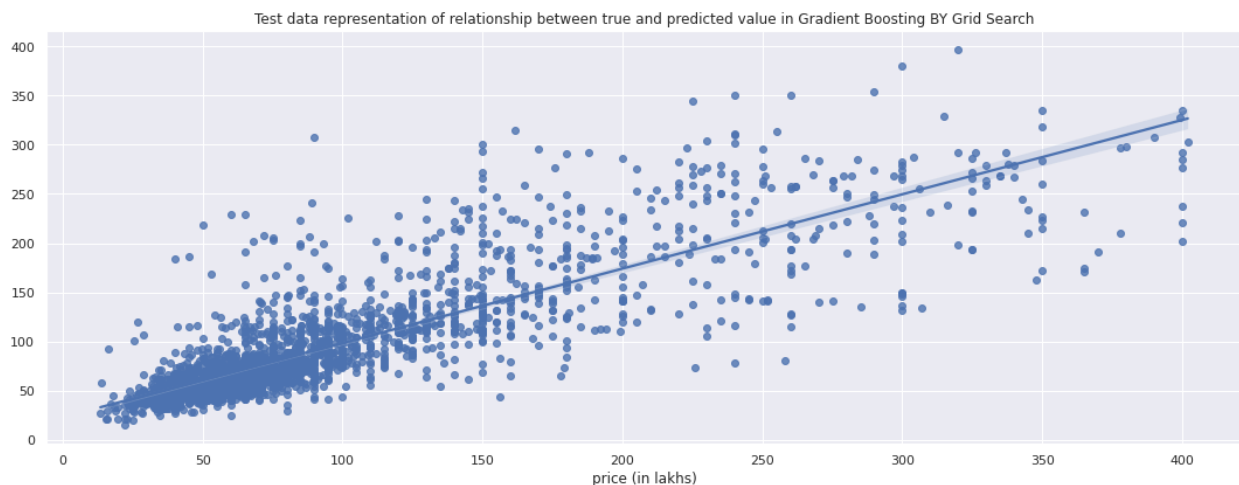


Fig-61: Regression plot of Gradient Boosting by Grid Search

Gradient Boosting Tuning with Graph

Here I updated the parameter set.

```
1 params_grid={
2     'learning_rate': [0.1, 0.01],
3     'max_depth': [3, 8],
4     'min_samples_leaf': [3, 6],
5     'min_samples_split': [20, 40],
6     'n_estimators': [200, 300],
7 }
8
```

Fig-62: Code snap of assigning parameter grid of Gradient Boosting Tuning by Graph

I also used 10-fold cross validation. The best parameters set is:

```
Fitting 10 folds for each of 32 candidates, totaling 320 fits
Best parameters:
{'learning_rate': 0.1, 'max_depth': 8, 'min_samples_leaf': 3, 'min_samples_split': 40, 'n_estimators': 300}
Best score: 0.733343586779629
```

Fig-63: Tuned parameters of Gradient boosting Tuning using Graph

But this time the training accuracy is improved to 90% but the validation and test accuracy are the same that means the model slightly overfits the data. So now move on to the randomized search.

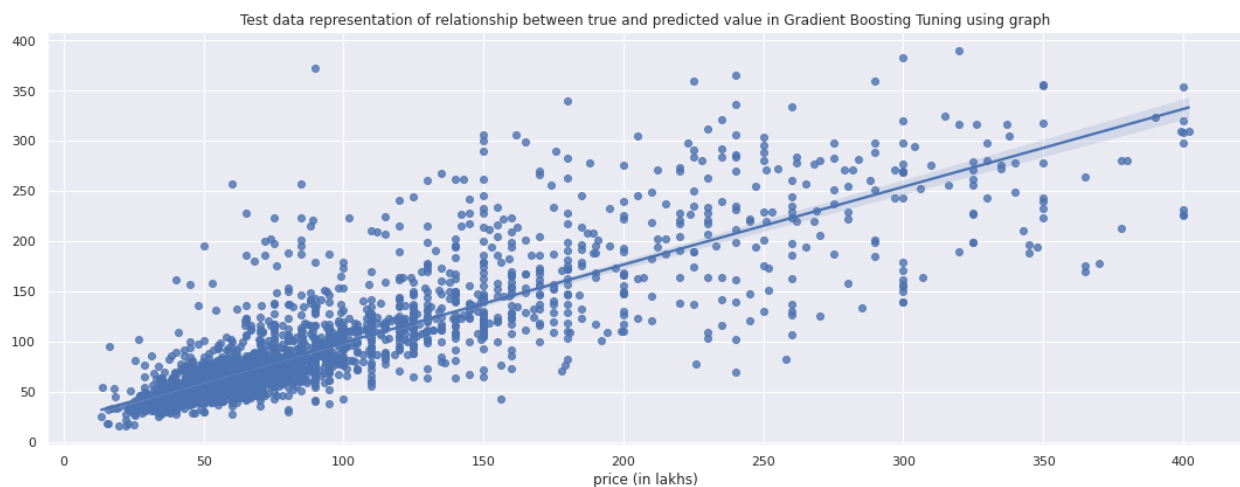


Fig-64: Regression plot of Gradient Boosting Tuning by Graph

Gradient Boosting using Randomized Search

The parameter set that is used as follows:

```
1 params_grid={
2     'learning_rate': [0.1, 0.01],
3     'max_depth': [3, 5],
4     'min_samples_leaf': [5, 10],
5     'min_samples_split': [30, 40],
6     'n_estimators': [200, 300],
7 }
8
```

Fig-65: Code snapt of assigning parameter grid of Gradient Boosting by Random Search

This time I also used 10-fold cross validation. Now the best parameters are updated to as follows:

```
Fitting 10 folds for each of 10 candidates; totalling 100 fits
Best parameters:
{'n_estimators': 300, 'min_samples_split': 30, 'min_samples_leaf': 5, 'max_depth': 5, 'learning_rate': 0.1}
Best score: 0.7322854192742101
```

Fig-66: Tuned parameters of Gradient boosting using Random Search

After applying 10-fold cross validation to tune the gradient boosted trees' hyperparameter, the results are more accurate. We can see that randomized search cv tuning produces the greatest results compared to other models. It also seems to generalize the data quite a bit without being overfitted much.

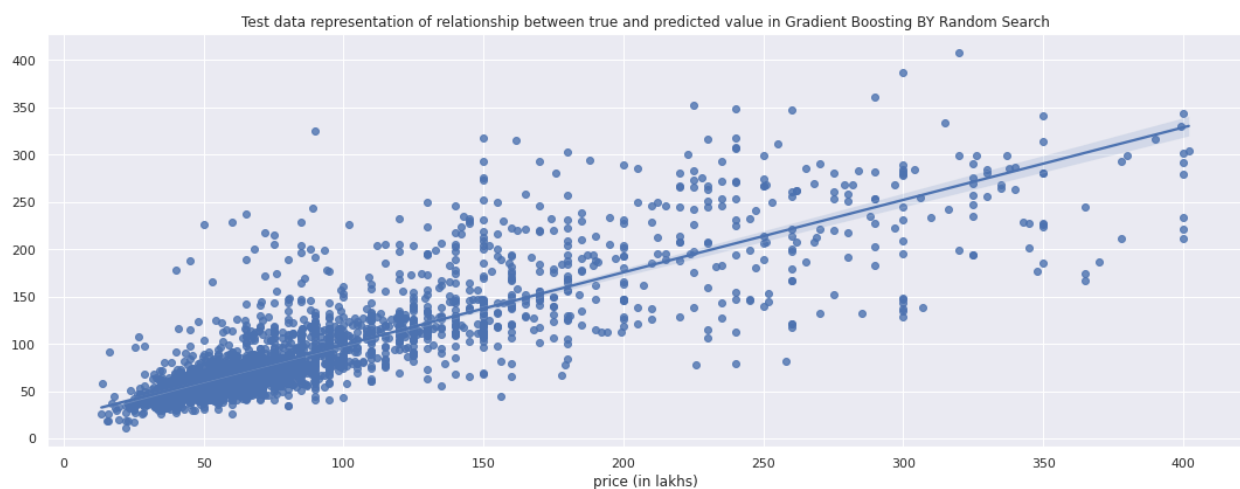
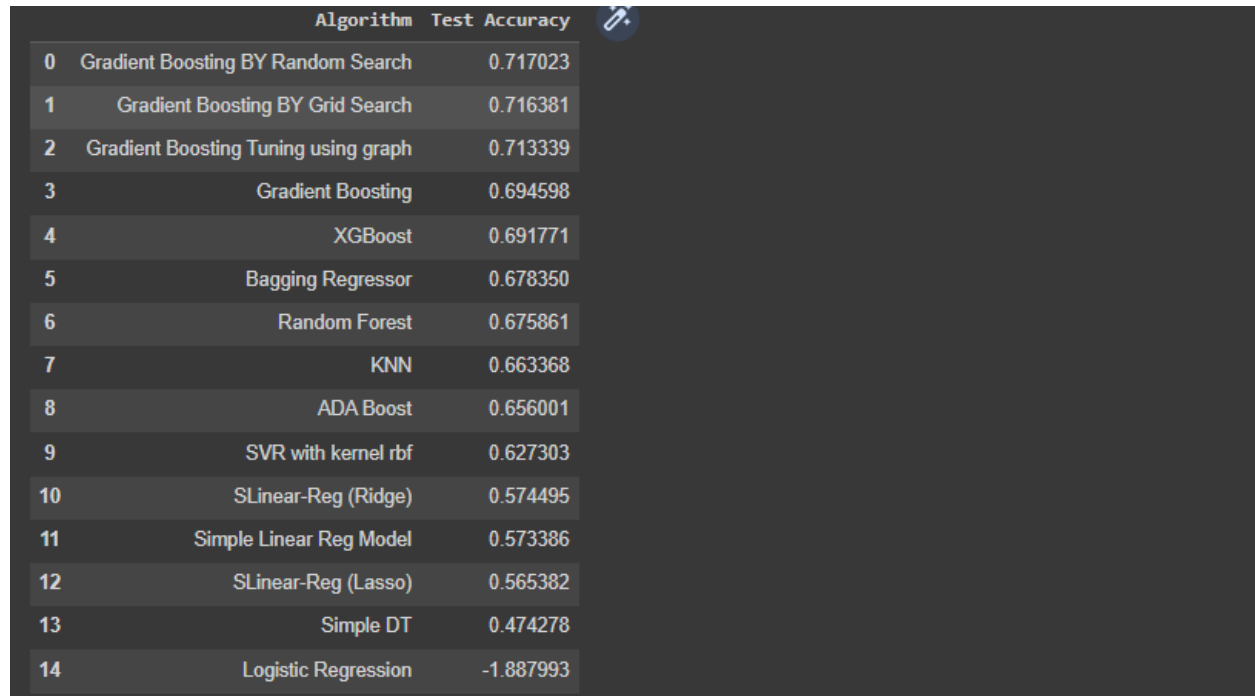


Fig-67: Regression plot of Gradient Boosting by Random Search

Performance Comparison

Now I'll compare the performance of all models by testing accuracy.

A screenshot of a Jupyter Notebook interface showing a table of model performance. The table has two columns: 'Algorithm' and 'Test Accuracy'. It lists 15 different models, with Gradient Boosting BY Random Search having the highest accuracy (0.717023) and Logistic Regression having the lowest (-1.887993).

	Algorithm	Test Accuracy
0	Gradient Boosting BY Random Search	0.717023
1	Gradient Boosting BY Grid Search	0.716381
2	Gradient Boosting Tuning using graph	0.713339
3	Gradient Boosting	0.694598
4	XGBoost	0.691771
5	Bagging Regressor	0.678350
6	Random Forest	0.675861
7	KNN	0.663368
8	ADA Boost	0.656001
9	SVR with kernel rbf	0.627303
10	SLinear-Reg (Ridge)	0.574495
11	Simple Linear Reg Model	0.573386
12	SLinear-Reg (Lasso)	0.565382
13	Simple DT	0.474278
14	Logistic Regression	-1.887993

Fig-68: Models performance

The table shows that, among the other models, Gradient Boosting by Random Search produces the best outcomes. Additionally, the performance of the ensemble models was greatly enhanced by hyperparameter tuning. Since that is an inappropriate approach for the regression problem, it is not unexpected that the performance of the logistic regression is the lowest it has ever been in the train, validation, and test data.

7. Final Findings and Recommendation

Findings

I'll analyze the final findings of the modeling for clustering and regression machine learning models, respectively.

Clustering

1. With the help of elbow plots, silhouette analysis and the sum of squared distance method are employed.
2. The silhouette approach generates two clusters, but the ssd method generates thirty to sixty or more clusters. However, the silhouette method is more efficient.
3. It was very hard to find the optimal clusters. The clusters are almost similar in kind.

Machine Learning

1. Data is cleaned and there is no multicollinearity in the data.
2. Simple ml models like linear regression, KNN, SVR, decision tree performed poor.
3. Ensemble methods are performed well.
4. Gradient boosting algorithm is performed well by performing the hyperparameter tuning using randomized search cv.
5. 10-fold cross validation is used to tune the hyperparameters.

Recommendations:

Now I'll make some recommendations with respect to the clustering and the regression models, respectively for improving the performance.

Clustering

1. More clustering methods like DBSCAN, TSNE etc can be used to increase the performance
2. Clustering performance is enhanced by employing either RICA or SFT to apply unsupervised feature learning to input data.
3. Using Agglomerative Hierarchical clustering can improve the performance..

Regression Machine Learning Models

1. Results can be improved by using neural networks or more hyperparameter tuning on both simple machine learning and the neural networks.
2. To model how two or more independent factors combined affect the target variable, add interaction terms.
3. To model the nonlinear relationship between an independent variable and the target variable, include polynomial terms.

8. Conclusion

- The rigorous method of investigation used during statistical analysis that enhances the data.
- Clustering is done carefully using better methods of the analysis.
- No multicollinearity present in the final features.
- Ensemble methods have performed well during the prediction and Gradient Boosting model performed better from the other models..
- 10-fold cross validation is used to validate the data using tuning.
- Randomized Search CV performed slightly better than the Grid Search CV during hyperparameter tuning.