Comparative Analysis of Machine Learning Techniques for Housing Price/Rent Prediction

Saheli Dutta
x21246513student@ncirl.ie
National College of Ireland
Masters in Data Analytics

***Abstract*—This paper presents a comparative analysis of machine learning techniques for predicting housing prices or rents using three different datasets. The techniques evaluated include decision tree, random forest, and XGBoost, implemented using both SKlearn and xgboost libraries. The study aims to address two research questions: 1) Do different implementations of identically-named machine learning techniques perform exactly the same, and if not, which implementation is better than the others for each dataset? 2) How do the datasets differ from each other in terms of characteristics that impact the effectiveness of machine learning techniques? The results show that the performance of the techniques varies depending on the dataset and evaluation measures used and the best-performing implementation of each technique almost remains for each dataset with respect to their library. The findings also indicate that the datasets differ significantly in terms of their characteristics, such as the number of features, size, and distribution of data. To enable a fair comparison, feature engineering was performed on each dataset. Overall, this study provides insights into the effectiveness of different machine learning techniques for housing price/rent prediction and highlights the importance of considering dataset characteristics in machine learning modeling.**

***Index Terms*—Home Price, House Rent, Prediction, EDA, Python, Decision Tree, Random Forest, XGBoost, and Machine Learning.**

I. Introduction

The prediction of housing prices and rents is an important problem in the real estate industry, with significant implications for buyers, sellers, and renters alike. In recent years, machine learning techniques have emerged as a powerful tool for addressing this problem, offering the potential to model complex relationships between various features of a property and its market value. However, selecting the appropriate machine learning method for a given dataset can be challenging, and the performance of different techniques may vary depending on the specific characteristics of the data.

In this study, a comparative analysis of several machine learning techniques for predicting housing prices or rents using three different datasets has been conducted. Specifically, performance comparison of the decision tree, random forest, and XGBoost methods implemented using both SKlearn and xgboost libraries.

To address research questions, exploratory data analysis (EDA) is first performed on each dataset to gain insights into their distributions, correlations, and other relevant properties. The data is then preprocessed and feature engineering is performed where necessary. Each machine learning technique is trained and evaluated on each dataset using appropriate evaluation measures. Finally, the results across techniques and datasets are compared to identify the best-performing method for each dataset and to shed light on the factors that influence the performance of machine learning techniques in this domain.

Overall, the findings of this study will contribute to the understanding of the effectiveness of different machine learning techniques for predicting housing prices and rents, and highlight the importance of considering dataset characteristics when selecting and applying these techniques.

II. Related Work

The real estate industry has undergone significant changes with the advent of technology and the rise of PropTech. This digital transformation has led to new functions such as transparency, data analytics, and machine learning in the industry. In India, there are many websites collecting property data, but the prices for the same apartment can vary. This project uses machine learning to predict house prices, focusing on the Bangalore city suburban housing data. Linear regression and decision tree models are used to evaluate house prices, along with a multi-dimensional object model. The aim is to solve the problems of relapse where the target variable is the value and the independent variable is the region. Classified websites can use the proposed algorithm to predict the values of new properties listed by taking variable inputs. This work is related to the use of machine learning algorithms for real estate price prediction and is in line with the current trends in PropTech. The project is edited by Palak Furia, and Anand Khandare [1].

Similarly, In their article "Price forecasting for real estate using machine learning: A case study on Riyadh city," Ali Louati, Rahma Lahyani, Abdulaziz Aldaej, Abdullah Aldumaykhi, and Saad Otai examine the potential of machine learning algorithms for predicting real estate prices. They utilize decision tree, random forest, and linear regression algorithms to build models and collect data from 5946 lands in the northern area of Riyadh, KSA. The performance of the models is evaluated using mean absolute error, mean squared error and median squared error. The authors find that the random forest-based model outperforms the other models [2].

In conclusion, the use of machine learning algorithms in real estate price prediction is becoming more prevalent with the rise of PropTech. The studies presented in this related work highlight the potential of machine learning in accurately

forecasting real estate prices. The proposed algorithms in these studies help me to understand the concept. These works contribute to the current trend in the real estate industry toward data-driven decision-making.

III. Methodology

The purpose of the methodology section is to provide a clear description of how the research was conducted and the steps taken to achieve the results. This section outlines the approach used to predict house prices using machine learning algorithms. The data sources used, KDD approach followed, and the justification for the choice of machine learning algorithms are described. The methodology section is essential to ensure that the research conducted is transparent and can be replicated.

A. Knowledge Discovery in Databases (KDD)

Knowledge Discovery in Databases is a data mining process that involves a number of steps to extract useful insights from large and complex datasets. The KDD approach involves the below steps -[3]

- **Data cleaning** involves the removal of noisy and irrelevant data from the dataset. In this study, Exploratory Data analysis(EDA) has been carried out to identify missing values, noisy data, and data discrepancies for all three datasets. Based on that, datasets were cleaned and preprocessed.
- **Data integration** involves combining heterogeneous data from multiple sources into a common source such as a data warehouse. In this study, data is from a single source so there is no need for integration.
- **Data selection** involves deciding which data is relevant to the analysis and retrieving it from the dataset. In the case of the housing datasets, relevant data includes various features such as the number of bedrooms, bathrooms, square footage, and location.
- **Data transformation** involves transforming data into an appropriate form required by the mining procedure. In the study, log transformation was performed on the Bengaluru house price dataset and Box-Cox transformation on both the house rent dataset and the Boston housing dataset. The goal of log and Box-Cox transformation is to improve the normality of the data and reduce the impact of outliers.
  After performing data transformation, the data was standardized using Standard Scaler to ensure that all the features have the same scale.
  In addition to standardization, one hot encoding was also used for the location feature in the Bengaluru house price dataset and Area Type, City, Furnishing Status, and Tenant Preferred features in house rent dataset. One hot encoding is used to convert categorical data into numerical data so that it can be used in machine learning algorithms. This step is necessary because most machine learning algorithms can only process numerical data.
- **Data mining** involves applying clever techniques to extract potentially useful patterns from the dataset. In this study, various machine learning algorithms were applied to the Bengaluru house price, house rent, and Boston housing datasets. Specifically, the Scikit-learn and XGBoost libraries were used to implement decision tree, random forest, and XGBoost algorithms. These algorithms were chosen based on their ability to handle regression problems, which is the focus of this study. The results of the data mining process will be presented and discussed in the subsequent sections of this report.
- **Pattern evaluation** involves identifying strictly increasing patterns that represent knowledge based on given measures. In this study, scatterplots, bar charts, box plots and etc have been used to identify the underlying relationship between each variable. This visualization makes the data more understandable.
- **knowledge representation** involves utilizing visualization tools to represent data mining results. This report includes details on the accuracy of the model such as R^2, MAE, RMSE, and MSE values as well as the significant features affecting house prices. Knowledge obtained from this analysis is represented through the report and presented as insightful information for stakeholders in the real estate industry.

Overall, KDD is an iterative process that can be enhanced by refining the mining process, integrating new data, and transforming data in order to get different and more appropriate results.

B. Data Sets and Characteristics

- **Bengaluru house price dataset** [4] - The Bengaluru House Price dataset contains 13123 rows and 9 columns of data on houses in Bangalore, India **(Fig 1)**.

COLUMN NAME	MEANING
AREA_TYPE	This column describes the type of area where the property is located
AVAILABILITY	This column indicates when the property can be possessed or when it will be ready for possession.
LOCATION	This column specifies the location of the property in Bengaluru, India
SIZE	This column indicates the number of bedrooms or BHK (Bedrooms, Hall, and Kitchen) of the property.
SOCIETY	This column specifies the society to which the property belongs.
TOTAL_SQFT	This column indicates the total size of the property in square feet.
BATH	This column specifies the number of bathrooms in the property.
BALCONY	This column indicates the number of balconies in the property.
PRICE	This column indicates the value of the property in lakhs (Indian Rupee - ₹).

Fig. 1: Bengaluru House Price Dataset

1) The dataset contains one dependent variable, ”price,” and eight independent variables, including ”area_type,”

"availability," "location," "size," "society," "total_sqft," "bath," and "balcony."

2) The dataset has 13123 records, where each record represents a single house in Bangalore.

3) The variables in the Bengaluru House Price dataset have been classified based on their data types. The data types include categorical for variables such as area_type and availability, nominal for variables like location and society, ordinal for the size variable, continuous numerical for variables such as total_sqft and price, and discrete numerical for variables like bath and balcony.

4) For the variable "bath," the minimum value is 1, the maximum is 40, the mean is 2.69, the median is 2, and the quartiles are 2, 2, and 3. These statistics suggest that most houses have 2-3 bathrooms, although some houses have as few as 1 or as many as 40 **(Fig 2)**.
For the variable "balcony," the minimum value is 0, the

	count	mean	std	min	25%	50%	75%	max
bath	13051.0	2.690905	1.338585	1.0	2.0	2.0	3.0	40.0
balcony	12526.0	1.583666	0.817563	0.0	1.0	2.0	2.0	3.0
price	13123.0	112.491768	146.423654	8.0	50.0	72.0	120.0	2912.0

Fig. 2: Bengaluru House Price Dataset Description

maximum is 3, the mean is 1.58, the median is 2, and the quartiles are 1, 2, and 2. These statistics indicate that most houses have 1-2 balconies, but some have none, and a few have three.
For the variable "price," the minimum value is 8, the maximum is 2912, the mean is 112.49, the median is 72, and the quartiles are 50, 72, and 120. These statistics suggest that house prices in Bangalore vary widely, with some houses selling for as little as 8 lakhs and others for as much as 2912 lakhs. The median price of 72 lakhs suggests that half of the houses in the dataset are priced below this value, while the other half are priced above it.
The output of the describe() method in Python only shows statistics for numeric columns, therefore the non-numeric columns (i.e., area_type, availability, location, size, total_sqft, and society) are not included in the output **(Fig 3)**. To see the frequency distribution of values in

```
array(['Super built-up  Area', 'Plot  Area', 'Built-up  Area',
       'Carpet  Area'], dtype=object)
```

Fig. 3: Area_Type Unique Values

the categorical and ordinal columns, the value_counts() method is used **(Fig 4)**. Additionally, the info() method is used to obtain information on the data types of all columns and the count of non-null values **(Fig 5)**.

5) After analyzing the Bengaluru House Price dataset, it is found that there are missing values in some columns.

```
Super built-up  Area    8654
Built-up  Area          2388
Plot  Area              1995
Carpet  Area              86
```

Fig. 4: Count of Unique Area_Type

```
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   area_type     13123 non-null  object
 1   availability  13123 non-null  object
 2   location      13122 non-null  object
 3   size          13107 non-null  object
 4   society       7704 non-null   object
 5   total_sqft    13123 non-null  object
 6   bath          13051 non-null  float64
 7   balcony       12526 non-null  float64
 8   price         13123 non-null  float64
```

Fig. 5: Dataset Info

Specifically, the "location" column has one missing value, the "size" column has 16 missing values, the "society" column has 5419 missing values, the "bath" column has 72 missing values, and the "balcony" column has 597 missing values **(Fig 6)**. In the data

```
area_type          0
availability       0
location           1
size              16
society         5419
total_sqft         0
bath              72
balcony          597
price              0
```

Fig. 6: Checking Null Values

cleaning process, some less important features are dropped from the dataset to simplify the analysis. The 'area_type', 'society', and 'availability' columns

Fig. 7: Correlation Check

are identified as less important and removed from the dataset using the drop() method. This is done to reduce

the dimensionality of the data and focus on the more relevant features.
The 'balcony' feature is found to have a low correlation coefficient of 0.12221 with the target variable Price, further supporting its removal from the dataset**(Fig 7)**. The resulting dataset contained only the 'location', 'size', 'total_sqft', 'bath', and 'price' columns.

After dropping the less important features from the dataset, null values are checked again. As a result, the 'location', 'size', 'bath' columns still have missing values. The 'total_sqft' and 'price' columns do not have any missing values. Specifically, there is 1 missing value in the 'location' column, 16 missing values in the 'size' column, and 72 missing values in the 'bath' column. The dropna() function is used to drop the rows with missing values **(Fig 8)**. In addition to checking for

```
location      0
size          0
total_sqft    0
bath          0
price         0
```

Fig. 8: Features with not null values

null values, the dataset was also checked for duplicate records using the duplicated() function. It was found that there were 608 duplicate records in the dataset, which were then dropped using the drop_duplicates() method. This step was taken to ensure that each observation in the dataset was unique and prevent any bias or inconsistencies in the analysis.

To further analyze the 'size' column in the dataset, the unique values of the column were checked using the unique() function. It was found that the values in the 'size' column contained the number of bedrooms and BHK (Hall-Kitchen) in the property **(Fig 9)**. To

	location	size	total_sqft	bath	price	Bhk
0	Electronic City Phase II	2 BHK	1056	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600	5.0	120.00	4
2	Uttarahalli	3 BHK	1440	2.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521	3.0	95.00	3
4	Kothanur	2 BHK	1200	2.0	51.00	2

Fig. 9: Bhk Feature

extract the BHK value, a new column 'Bhk' was created using the apply() function with a lambda function. The 'Bhk' column contains only the integer value of the number of BHK in the property. This new column can be used as a relevant feature for the analysis.

In addition, the 'total_sqft' column is converted to a numerical format using a custom function 'convert_totalsqft_to_number'. This function converts values like '2100 - 2850' to their mean value, i.e., 2475.0 **(Fig 10)**. The 'price_per_sqft' column is also

	location	size	total_sqft	bath	price	Bhk	price_per_sqft
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699.810606
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	4615.384615
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305.555556
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245.890861
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250.000000

Fig. 10: Price Per Sqft Feature

added to make the data more sensible. This column was calculated by dividing the price of the property by its total square footage.
After analyzing the data, it is observed that there are several locations in the dataset that have a very low count of properties. To avoid overfitting, these locations are grouped together under a new category called 'other'. This is done by identifying the locations whose count is less than or equal to 10 using the 'Location_Count' variable, and then replacing these locations with 'other' using the 'apply' function. By performing these data cleaning steps, the dataset was made more suitable for analysis and modeling.

Some data points have a suspiciously low square footage per bedroom. For example, a 2 BHK apartment with only 200 sqft would be considered an outlier as the minimum threshold for 2 BHK is 600 sqft (i.e., 300 sqft per bedroom). To remove such outliers, the minimum threshold of 300 sqft per bedroom is set and removed data points that did not meet this threshold. The threshold of 300 sqft per bedroom is a commonly accepted standard in the real estate industry and is based on the assumption that each bedroom should have a minimum amount of living space to be functional.

After removing outliers based on price per sqft using mean and one standard deviation, another round of outlier removal is performed based on the number of bhk for each location **(Fig 11)**.
This is done using a custom function 'bhk_outliers_removal', which calculates the mean, standard deviation, and count of prices per square foot for each bedroom type (bhk) in each location. It then compares the price per sqft of a bhk to the mean price per sqft of the next lower bhk type in the same location and removes any data points that fall below this threshold **(Fig 12)**.
In addition to removing outliers based on square footage per bedroom, the dataset is also cleaned to exclude data points where the number of bathrooms is

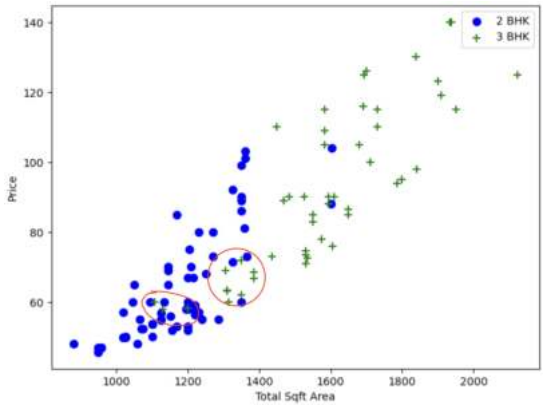

Fig. 11: Outliers Detected

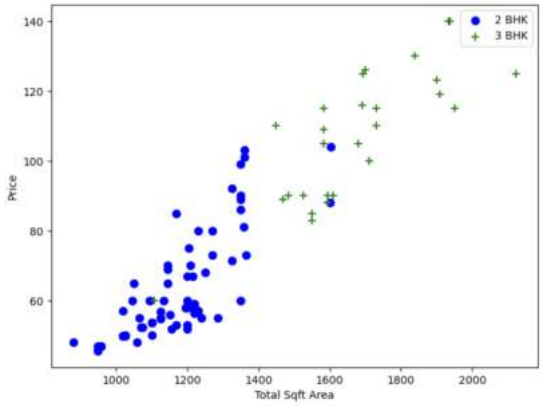

Fig. 12: Outliers Removed

greater than the number of bedrooms plus 2. This is a logical constraint in real estate as it is uncommon to have more bathrooms than bedrooms in a residential property. By dropping such data points, the dataset was further cleaned and prepared for analysis.

One-hot encoding is a technique used to convert categorical variables into a format that can be easily interpreted and used by machine learning algorithms. In this study, one-hot encoding is used to convert the 'location' feature of the housing data into numerical data that can be utilized by the machine learning model **(Fig 13)**. The get_dummies() function is used

	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar	9th Phase JP Nagar
0	1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0

Fig. 13: One Hot Encoding of Location Feature

to create binary variables for each unique category in the 'location' feature. Now a data frame with all the relevant numerical features is created which is needed to train our machine learning model. This transformation helps the model to interpret and analyze the data effectively and accurately.
The 'location' and 'size' feature is then dropped as it no longer holds any useful information.

The final Dataframe looks like below **(Fig 14)**.

	total_sqft	bath	price	Bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar
0	7.955074	4.0	6.059123	4	1	0	0	0	0	0
1	7.396335	3.0	5.267858	3	1	0	0	0	0	0
2	7.536364	2.0	5.459586	3	1	0	0	0	0	0
3	7.090077	2.0	4.867534	3	1	0	0	0	0	0
4	7.118826	2.0	4.997212	2	1	0	0	0	0	0

5 rows × 243 columns

Fig. 14: Final Dataframe

6) The 'total_sqft' and 'Price' variables are continuous numerical variables representing the total area of the property in square feet and the price of the house.
To address the non-normality issue, a log transformation is applied, which compresses the values and reduces the variability in the data. This helps to improve the model's performance, particularly for regression models, which assume a linear relationship between the independent and dependent variables **(Fig 15)**.

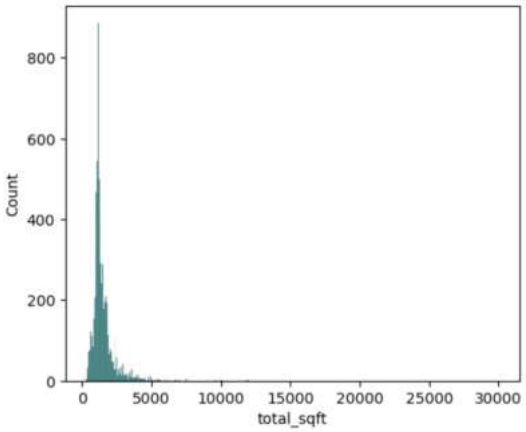

Fig. 15: Skewed Feature

This transformation helps to scale down the effect of extreme values and makes the variable more interpretable **(Fig 16)**.
On the other hand, the 'bath' and 'bhk' variables are not log-transformed because they are discrete variables and have a limited number of possible values. Therefore, a log transformation is not appropriate for these variables, and they were left as is.

7) The input features for the Bengaluru house price dataset are defined as all variables except the target variable

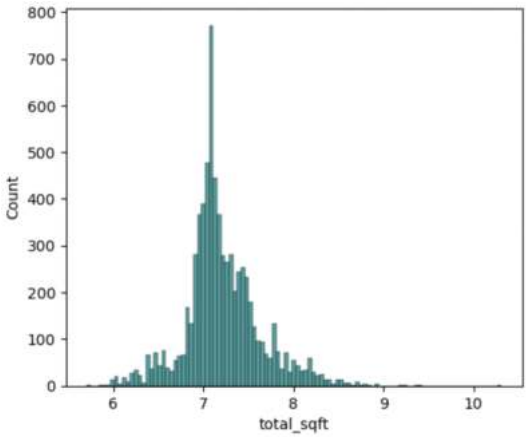

Fig. 16: Log Transformed Feature

'price'. The target variable 'price' is defined separately. The dataset is split into training and testing sets using a 70/30 split, where 70% of the data is used for training and the remaining 30% is used for testing. The training set contains 5043 records and the Testing set contains 2162 records.

To scale the numerical variables in the dataset, the StandardScaler() function is applied to the 'total_sqft', 'Bhk', and 'bath' columns of both the training and testing sets. This was done to ensure that all variables were on the same scale, which can help to improve the performance of the machine learning model.

- **House Rent dataset** [5] - The House Rent dataset contains 4632 rows and 12 columns of data on houses in India (**Fig 17**).

Column Name	Column Meaning
BHK	Number of bedrooms, hall, and kitchen
Rent	Rent of the house/apartment/flat
Size	Size of the house/apartment/flat in square feet
Floor	Floor number of the house/apartment/flat and total number of floors
Area Type	Type of area measurement (super area, carpet area, build area)
Area Locality	Locality of the house/apartment/flat
City	City where the house/apartment/flat is located
Furnishing Status	Furnishing status of the house/apartment/flat (furnished, semi-furnished, unfurnished)
Tenant Preferred	Type of tenant preferred by the owner/agent
Bathroom	Number of bathrooms
Point of Contact	Contact person for more information regarding the house/apartment/flat

Fig. 17: House Rent Dataset

1) The House Rent Dataset contains one dependent variable, "Rent," and ten independent variables, including "BHK," "Size," "Floor," "Area Type," "Area Locality," "City," "Furnishing Status," "Tenant Preferred," "Bathroom," and "Point of Contact."
2) The dataset has 4632 records, and each record represents a single house all over India.
3) The dataset includes a numerical variable, 'BHK,' 'Rent' and 'Size'. The 'Floor', and 'Point of Contact' columns are textual variables, while the 'Area Type,' 'Area Locality,' 'City,' 'Tenant Preferred' and 'Furnishing Status' columns are nominal categorical variables. while the 'Bathroom' column is a numerical variable.
4) For the variable "BHK," the dataset contains 4632 observations with a minimum value of 1 and a maximum value of 6. The mean value is indicating that, on average, each house/apartment/flat has just over two bedrooms, halls, and kitchens. The standard deviation is 0.83, suggesting that there is moderate variability in the number of BHKs across the dataset. The median value is 2, and the quartiles are 2, 2, and 3, indicating that most houses have two or three BHKs (**Fig 18**). The describe()

	count	mean	std	min	25%	50%	75%	max
BHK	4632.0	2.083981	0.830785	1.0	2.0	2.0	3.0	6.0
Rent	4632.0	35446.010363	78976.106746	1200.0	10000.0	16000.0	34000.0	3500000.0
Size	4632.0	964.262953	631.631187	20.0	550.0	850.0	1200.0	8000.0
Bathroom	4632.0	1.964810	0.884286	1.0	1.0	2.0	2.0	10.0

Fig. 18: House Rent Dataset Description

method in Python only provides statistical information for numerical columns in a dataset. Therefore, non-numerical columns, such as 'Posted On', 'Floor', 'Area Type', 'Area Locality', 'City', 'Furnishing Status', 'Tenant Preferred', and 'Point of Contact', are not included in the output of the describe() method.

The value counts() method is used to observe the frequency distribution of values in categorical and ordinal

```
array(['Bandel', 'Phool Bagan, Kankurgachi', 'Salt Lake City Sector 2',
       ..., 'Murad Nagar, Lal Darwaza, Falaknuma Road',
       'Kakatiya Hills Madhapur', 'Manikonda, Hyderabad'], dtype=object)
```

Fig. 19: unique Values of Area Locality Feature

columns (**Fig 19**). On the other hand, the info() method provides information about the data types of all columns and the count of non-null values (**Fig 20**).

```
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Posted On          4632 non-null   object
 1   BHK                4632 non-null   int64
 2   Rent               4632 non-null   int64
 3   Size               4632 non-null   int64
 4   Floor              4632 non-null   object
 5   Area Type          4632 non-null   object
 6   Area Locality      4632 non-null   object
 7   City               4632 non-null   object
 8   Furnishing Status  4632 non-null   object
 9   Tenant Preferred   4632 non-null   object
 10  Bathroom           4632 non-null   int64
 11  Point of Contact   4632 non-null   object
```

Fig. 20: Dataset Info

5) The Box-Cox transformation is a technique used to transform skewed data into normally distributed data. In this particular dataset, the 'Rent' and 'Size' columns are both left-skewed, which can negatively impact the accuracy of predictive models. By applying

the Box-Cox transformation to these columns, the distribution of the data is improved and increases the accuracy of models that use these variables. Also, extreme values are removed **(Fig 21)**. The resulting

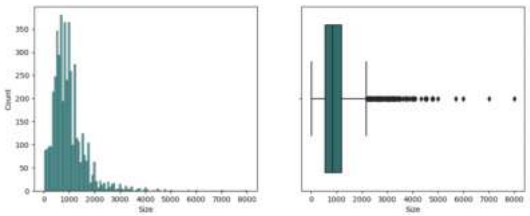

Fig. 21: Skewed Feature

transformed columns are named 'boxcox_size' and 'boxcox_rent', respectively (**Fig 22**). The 'BHK' and

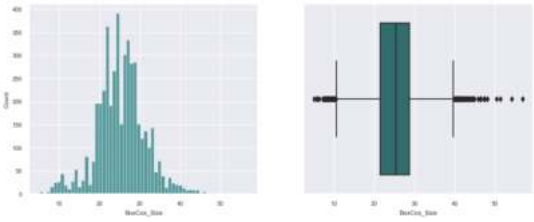

Fig. 22: Tranformed Feature

'Bathroom' columns, which represent discrete values, were not transformed using Box-Cox. This is because Box-Cox transformation is only suitable for continuous data, and attempting to transform discrete data could lead to erroneous results.

The correlation matrix heatmap provides a visual representation of the correlation between different variables in the dataset (**Fig 23**).

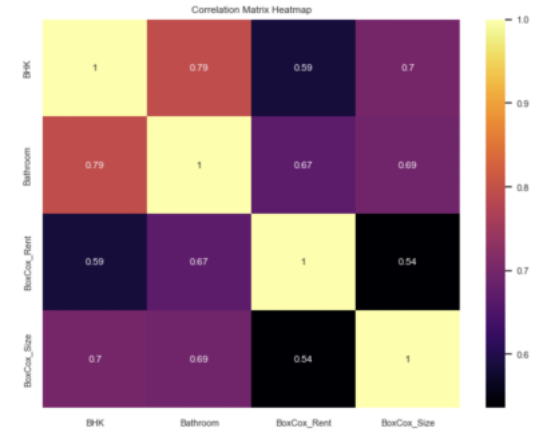

Fig. 23: Correlation Check

6) The 'Floor' column in the dataset contains information about the floor level and the total number of floors in the building. To make this data more usable for analysis, feature engineering techniques are applied to split this column into two separate columns, 'Floor Level' and 'Total Floors'. Also converted the values for 'Ground', 'Lower Basement', and 'Upper Basement' to 0, -1, and -2 respectively, and converted the remaining values to integers (**Fig 24**). The 'Posted On' column

	Posted On	BHK	Area Type	City	Furnishing Status	Tenant Preferred	Bathroom	BoxCox_Rent	BoxCox_Size	Floor Level	Total Floors
4627	2022-05-18	2	Carpet Area	Hyderabad	Semi-Furnished	Bachelors/Family	2	2.854800	26.927612	3	5.0
4628	2022-05-15	3	Super Area	Hyderabad	Semi-Furnished	Bachelors/Family	3	2.878052	34.692764	1	4.0
4629	2022-07-10	3	Carpet Area	Hyderabad	Semi-Furnished	Bachelors/Family	3	2.883793	33.054371	3	5.0
4630	2022-07-06	3	Carpet Area	Hyderabad	Semi-Furnished	Family	2	2.860819	31.251305	23	34.0
4631	2022-05-04	2	Carpet Area	Hyderabad	Unfurnished	Bachelors	2	2.854800	26.927612	4	5.0

Fig. 24: Feature Engineering - Floor

contains information about the date the post is made. To make this information more useful, the year, month, and day are extracted as separate columns using the dt. year, dt. month, and dt. day functions, respectively **(Fig 25)**. Overall, these feature engineering processes have

	BHK	Area Type	City	Furnishing Status	Tenant Preferred	Bathroom
0	2	Super Area	Kolkata	Unfurnished	Bachelors/Family	2
1	2	Super Area	Kolkata	Semi-Furnished	Bachelors/Family	1
2	2	Super Area	Kolkata	Semi-Furnished	Bachelors/Family	1
3	2	Super Area	Kolkata	Unfurnished	Bachelors/Family	1
4	2	Carpet Area	Kolkata	Unfurnished	Bachelors	1

	BoxCox_Rent	BoxCox_Size	Floor Level	Total Floors	year	month	day
0	2.837715	27.892154	0	2.0	2022	5	18
1	2.865584	24.785524	1	3.0	2022	5	13
2	2.859620	26.927612	1	3.0	2022	5	16
3	2.837715	24.785524	1	2.0	2022	7	4
4	2.824099	25.351850	1	2.0	2022	5	9

Fig. 25: Feature Engineering - Posted On

a significant impact on the accuracy and usefulness of predictive models.

Upon inspection of the dataset, it is discovered that there are 10 duplicate values and 4 null values present. Given that the number of records in the dataset is significantly larger than the number of duplicates and null values, it is deemed appropriate to remove them from the dataset. By removing these values, the integrity of the data is preserved and any potential issues with data analysis or model building that could arise from these values are eliminated.

In this dataset, the 'Area Type', 'City', 'Furnishing Status', and 'Tenant Preferred' columns are one hot encoded using the pandas get_dummies() method. This creates a binary matrix where each category becomes its own column, with a value of 1 indicating the presence of that category and a value of 0 indicating the absence of that category (**Fig 26**). The resulting

City_Delhi	City_Hyderabad	City_Kolkata	City_Mumbai	Furnishing Status_Furnished	Furnishing Status_Semi-Furnished	Furnishing Status_Unfurnished	Tenant Preferred_Bachelors	Tenant Preferred_Bachelors/Family	Pre
0	0	1	0	0	0	1	0	1	
0	0	1	0	0	1	0	0	1	
0	0	1	0	0	1	0	0	1	
0	0	1	0	0	0	1	0	1	
0	0	1	0	0	0	1	1	0	

Fig. 26: One Hot Encoded Columns

one-hot encoded columns use as input for machine learning models, which require numeric data. One hot encoding is necessary in this case because machine

learning algorithms cannot directly process categorical data.

The final dataset looks like below **(Fig 27)**.

	BHK	Bathroom	BoxCox_Rent	BoxCox_Size	Floor Level	Total Floors	year	month	day	Type	Area Built Area	...	City_Delhi	City_Hyderabad	City_Kolkata
0	2	2	2.837715	27.892154	0	2.0	2022	5	18	0	...	0	0	1
1	2	1	2.865584	24.785524	1	3.0	2022	5	13	0	...	0	0	1
2	2	1	2.859620	26.927612	1	3.0	2022	5	16	0	...	0	0	1
3	2	1	2.837715	24.785524	1	2.0	2022	7	4	0	...	0	0	1
4	2	1	2.824099	25.351850	1	2.0	2022	5	9	0	...	0	0	1

5 rows × 24 columns

Fig. 27: Final Dataset

7) The input features for the house rent dataset include all variables except the target variable, "boxcox_rent", which is defined separately. To train and test the model, the dataset is split into a 70/30 ratio for the training and testing sets, respectively. The training set contains 3,231 records, and the testing set contains 1,386 records.
   To ensure that all variables are on the same scale, the "StandardScaler()" function is applied to the "BHK", "Bathroom", "BoxCox_Size", "Floor Level", and "Total Floors" columns of both the training and testing sets. This step can potentially improve the performance of the machine learning model.

- **Boston Housing dataset** - The Boston Housing dataset is loaded, and the data was converted into a Pandas DataFrame. It contains 506 rows and 14 columns (**Fig 28**).

Column Name	Meaning
CRIM	per capita crime rate by town
ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	proportion of non-retail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX	nitric oxides concentration (parts per 10 million)
RM	average number of rooms per dwelling
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centers
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per $10,000
PTRATIO	pupil-teacher ratio by town
B	1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
LSTAT	% lower status of the population
target	median value of owner-occupied homes in $1000s

Fig. 28: Boston Housing Dataset

1) The Boston Housing Dataset contains one dependent variable, "target(MEDV)" and thirteen independent variables, including "CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", and "LSTAT".
2) The dataset has 506 records and each record presents a single house in Boston.
3) The Boston dataset includes a combination of data types, including binary (CHAS), nominal categorical (RAD), numerical (CRIM, ZN, INDUS, NOX, RM, AGE, DIS, TAX, PTRATIO, B, LSTAT), and textual (none). These variables provide information on factors that may impact the median value of owner-occupied homes, which is the target variable in this dataset (**Fig 29**).

```
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   CRIM      506 non-null    float64
 1   ZN        506 non-null    float64
 2   INDUS     506 non-null    float64
 3   CHAS      506 non-null    float64
 4   NOX       506 non-null    float64
 5   RM        506 non-null    float64
 6   AGE       506 non-null    float64
 7   DIS       506 non-null    float64
 8   RAD       506 non-null    float64
 9   TAX       506 non-null    float64
 10  PTRATIO   506 non-null    float64
 11  B         506 non-null    float64
 12  LSTAT     506 non-null    float64
 13  target    506 non-null    float64
```

Fig. 29: Boston Housing Dataset Info

4) The 'CRIM' variable represents the per capita crime rate by town. The minimum value is 0.00632 and the maximum is 88.9762. The median value is 0.25651, indicating that the majority of towns have a relatively low crime rate. The 25th percentile is 0.082045 and the 75th percentile is 3.677083. These values indicate that there is a wide range of crime rates across the towns in this dataset, with some areas having very low rates and others having extremely high rates (**Fig 30**). The 'RM' variable represents the average number of rooms per dwelling. The minimum value is 3.561 and the maximum is 8.78. The median value is 6.2085, indicating that the majority of dwellings have around 6 rooms. The 25th percentile is 5.8855 and the 75th percentile is 6.6235, suggesting that there is a relatively narrow range of the number of rooms in dwellings in the Boston area.

	count	mean	std	min	25%	50%	75%	max
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.9762
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.0000
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.7400
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.0000
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.8710
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.7800
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.0000
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.1265
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.0000
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.0000
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.0000
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.9000
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.9700
target	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.0000

Fig. 30: Boston Housing Dataset Description

5) The Box-Cox transformation is a statistical technique used to transform non-normal data into a normal distribution. In this dataset, the Box-Cox transformation

is applied to a set of skewed features in the Boston dataset, including 'CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS', 'TAX', 'PTRATIO', 'B', and 'LSTAT'. By transforming these features into a normal distribution (**Fig 31**). It reduces the impact of outliers

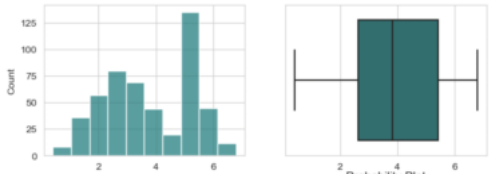

Fig. 31: Box Cox Transformed Feature

and improves the accuracy of statistical models that are sensitive to normality assumptions, such as linear regression, Decision Trees. The Box-Cox transformation reduces the variability of the data, leading to better model performance. After applying the Box-Cox transformation to the selected features in the Boston dataset, the outliers in the data are reduced (**Fig 31 & 32**).

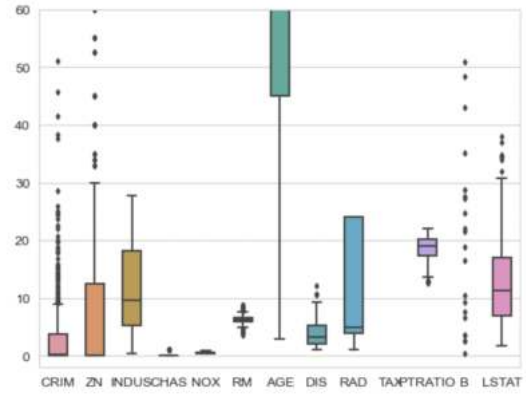

Fig. 32: Outliers Present

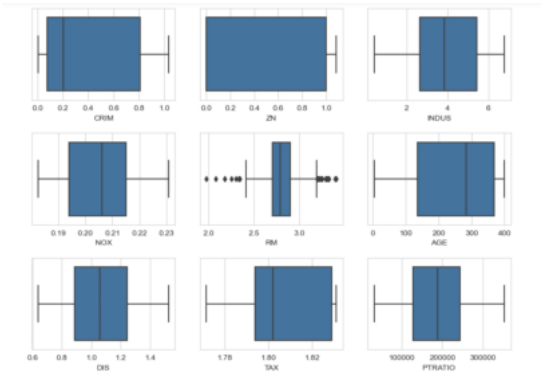

Fig. 33: Outliers Removed

6) The heatmap of the correlation matrix provides a graphical illustration of the interrelation between various variables in the given dataset (**Fig 33**). Where the highly correlated feature is RM with a value of 0.67 and the weakly correlated feature is AGE with a value of 0.38.

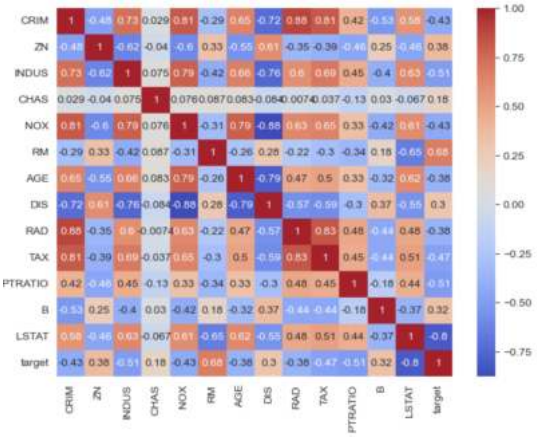

Fig. 34: Correlation Matrix

7) The Boston dataset contains several input features, excluding the target variable. The dataset is split into a 70/30 ratio for the training and testing sets, respectively, with 354 records in the training set and 152 records in the testing set.
To ensure all input features are on the same scale, the StandardScaler() function is applied to the selected columns, including 'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', and 'LSTAT'. This scaling process is performed on both the training and testing sets, and it has the potential to improve the performance of the machine learning model.

C. Visualization

- **Bengaluru House Price Dataset Visual**

1) A grid density plot is created to analyze the distribution of the number of bathrooms (bath) in different types of housing areas. The plot shows that the majority of houses have 1-5 bathrooms, regardless of the area type.

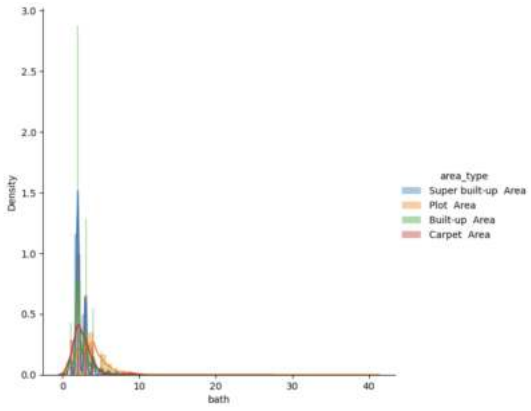

Fig. 35: Grid Density Plot to Analyze the Distribution of The Number of Bathrooms in Different Area_Type

However, the plot also shows that houses in Built_Up areas tend to have a higher number of bathrooms compared to other area types (**Fig 35**).
The plot provides insights into the distribution of the number of bathrooms in different area types and can be useful for identifying patterns in the data.

2) A stacked bar chart is created to visualize the distribution of area types based on the number of balconies in properties. The x-axis represents the number of balconies, while the y-axis shows the count of properties. The chart is color-coded to represent the different area types, namely Built-up Area, Carpet Area, Plot Area, and Super built-up Area (**Fig 36**). From the chart, it is observed that

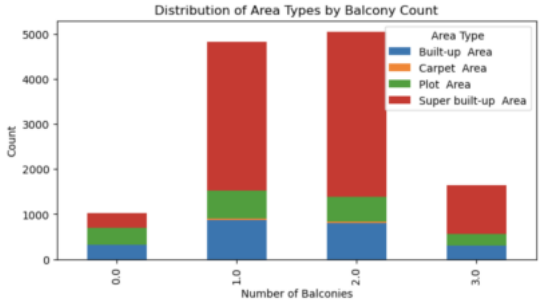

Fig. 36: Distribution of Area Types by Balcony Count

the majority of properties have two balconies, followed by one balcony. The Super built-up Area is the most common area type among properties with one, two, and three balconies, while properties with zero balcony are equally distributed among super built-up, built-up, and plot areas.

- **House Rent Dataset Visual**

1) The bar plot displays the relationship between the Rent and Tenant city based on the Furnishing status. The

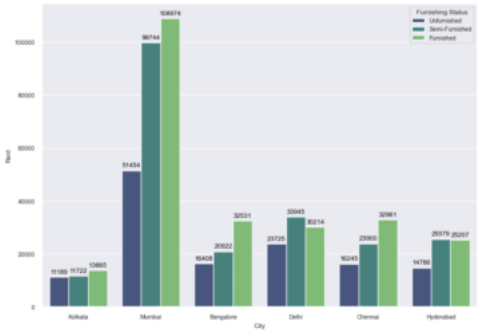

Fig. 37: Relationship between Rent and Tenant City based on Furnishing Status

x-axis represents the City, and the y-axis represents the Rent. The hue indicates the Furnishing status of the apartment, which can be Unfurnished, Semi-Furnished, or Furnished. The graph shows that the Rent is higher for Furnished apartments compared to Semi-Furnished and Unfurnished apartments in all cities except Delhi and Hyderabad (**Fig 37**).
Additionally, the Rent is higher in Mumbai than in other cities for all types of Furnishing Status. The height of each bar represents the Rent value for the corresponding Furnishing status and city. The annotations on top of the bars represent the Rent value.
Overall, the graph provides a visual representation of the Rent trend in different cities based on the Furnishing status of the apartment.

2) A countplot is created to explore the distribution of the numerical variables BHK (bedroom, hall, and kitchen) and Bathroom in the dataset. The plot shows that the majority of the houses have 2 BHK and 1 or 2 bathrooms (**Fig 37**). The count of the BHK variable indicates

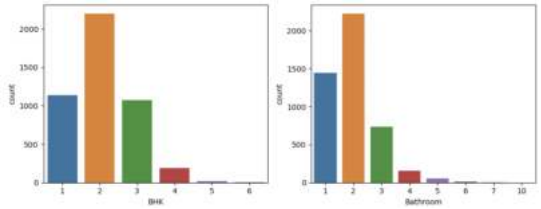

Fig. 38: Distribution of BHK and Bathroom Variables in the Dataset

that the dataset is skewed towards 2 BHK apartments, while the count of the Bathroom variable shows that the majority of apartments have either 1 or 2 bathrooms.

3) The pie chart displays the distribution of Area Localities in the dataset, where all of the properties are located in India (**Fing 39**). The chart provides a visual repre-

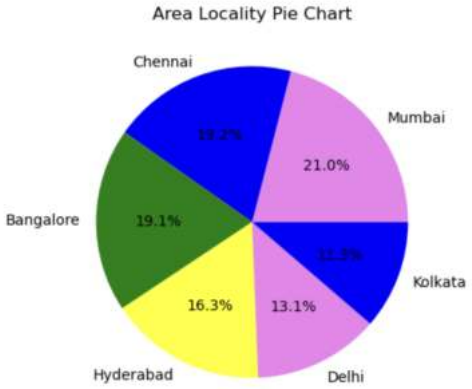

Fig. 39: Area Locality Distribution in the Dataset

sentation of the proportion of properties in each state (Mumbai has the majority of rental assets followed by Chennai and Bangalore), making it easier to understand the distribution of properties in the dataset.

- **Boston Housing Dataset Visual**

1) A scatter plot is generated to visualize the relationship between the average number of rooms per dwelling (RM) and the median value of owner-occupied homes in $1000s (target) in the dataset (**Fig 40**). This observation

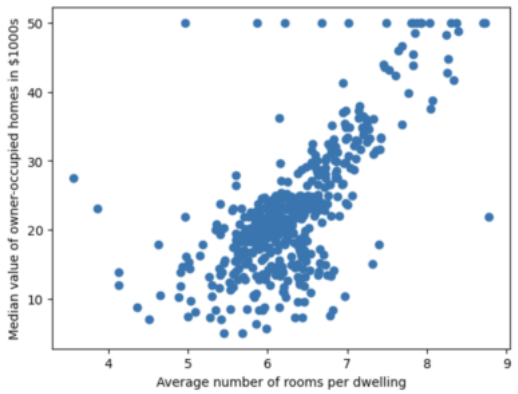

Fig. 40: Relationship between Average Number of Rooms and Median Value of Owner-Occupied Homes

is consistent with the intuition that larger houses are generally more expensive than smaller ones. The plot shows a positive linear correlation between the two variables, indicating that as the average number of rooms per home increases, the median value of owner-occupied homes also tends to increase.

2) Histograms are generated to understand the distribution of each variable in the Boston dataset (**Fig 41**). Two

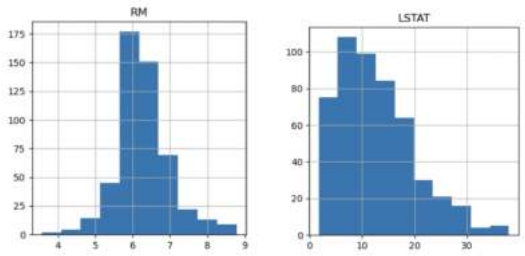

Fig. 41: Distribution of Important Features in Boston Housing Dataset: Average Number of Rooms per Dwelling (RM) and Percentage of Lower Status of the Population (LSTAT)

important features in the Boston housing dataset, namely the RM and LSAT. The RM feature appears to be normally distributed, with the majority of the data falling between 5 to 7 rooms per dwelling.
On the other hand, the LSTAT feature shows a skewed right distribution, with the majority of the data clustered around smaller values and a long tail towards higher values. These visualizations provide valuable insights into the underlying distribution of the features and are used to inform further analysis and modeling.

D. Machine Learning Algorithms

- **Decision Tree** is a widely used tool in data mining and machine learning that is used to visually represent decisions and decision-making processes. This algorithm can be used for both classification and regression, where regression trees are used to predict continuous values such as the price of a house. In general, decision tree algorithms are known as CART, which stands for Classification and Regression Trees [6].

  The reason for choosing decision tree is that it is an algorithm that captures complex relationships between features and the target variable, which is useful in making accurate predictions. One disadvantage of decision trees is that they are prone to overfitting, which leads to poor generalization performance on new data. Finally, decision trees are relatively fast and efficient,

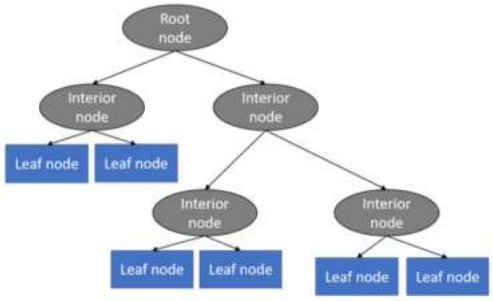

Fig. 42: Decision Tree. Adapted from Ahmad Abdulal and Nawar Aghi.House Price Prediction Report. Retrieved May 01, 2023, from https://researchportal.hkr.se/en/studentTheses/house-price-prediction-5

.

  making it possible to train and evaluate models on large datasets in a reasonable amount of time. For these reasons, the decision tree algorithm is a popular choice for machine learning problems involving regression tasks **(Fig 42)**.

- **Random Forest** is an ensemble learning technique that uses multiple decision trees to solve classification and regression problems. It leverages bagging or bootstrap aggregating to train a forest of decision trees, which predict the outcome by taking the average or mean of the outputs from various trees. With an increasing number of trees, the precision of the prediction increases, eradicating the limitations of a decision tree algorithm. Random forest reduces overfitting and generates predictions with high precision, without requiring many configurations in packages [7].
  The reason for choosing Random Forest is that it is a popular choice for solving regression problems due to its ability to capture complex relationships and reduce overfitting (**Fig 43**). It is an ensemble method

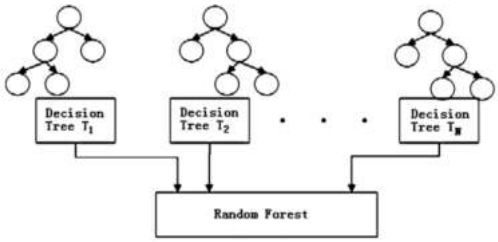

Fig. 43: Random Forest. Adapted from Ahmad Abdulal and Nawar Aghi.House Price Prediction Report. Retrieved May 01, 2023, from https://researchportal.hkr.se/en/studentTheses/house-price-prediction-5.

that combines multiple decision trees, providing higher accuracy than a single decision tree. The random forest handle missing values and outliers and requires little feature engineering.

- **XGBoost** is a popular implementation of gradient boosting that offers regularization to prevent overfitting and can handle sparse data sets using the weighted quantile sketch algorithm. It also has a block structure for parallel learning, uses cache awareness, and offers out-of-core computing capabilities. XGBoost does not require parameter tuning and is designed for speed and performance on large datasets [8].
  The reason for choosing XGBoost is that it is specifically designed to handle large datasets, which makes it a great choice for predicting home rent or price. It offers regularization, which helps control overfitting (**Fig 44**). However, one disadvantage of XGBoost is that it can

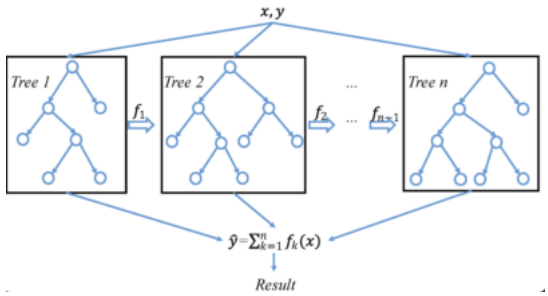

Fig. 44: XGBoost. Adapted from Researchgate. Retrieved May 02, 2023, from https://www.researchgate.net/figure/A-general-architecture-of-XGBoost_fig3_335483097.

be computationally expensive and slow on very large datasets.

E. Machine Learning Library

- **Scikit-learn** is a Python library that provides a variety of supervised and unsupervised learning algorithms, built upon familiar technologies such as NumPy, pandas, and Matplotlib. It includes tools for regression, classification, clustering, model selection, and preprocessing [9].

- **XGBoost** is a machine-learning library that implements gradient boosting for classification, regression, and clustering problems. It utilizes the concept of weak learners that work together to produce a strong learner. XGBoost sequentially adds models to correct errors made by previous models, resulting in an optimized solution. This process is known as ensembling [10].

F. Evaluation Metrics

- To evaluate the prediction accuracy of the trained model for home rent/price prediction problem, performance metrics such as R-Squared (R2), Root Mean Square Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Error (MAE) will be utilized. R2 will indicate whether the model is overfitted or not, while RMSE, MSE, and MAE will measure the percentage of error between the actual and predicted values of the house prices.

G. Experimental Design

The train-test split is performed using the train_test_split function from sklearn.model_selection module. A random state of 42 is used to ensure that the same split is obtained every time the code is run, thus making the results reproducible.

The experimental design used to compare the performance of different models is as follows:

- **Decision Tree Regression**: A decision tree model is trained using the DecisionTreeRegressor class from sklearn.tree module. The hyperparameters for the model are set to 'criterion': 'mse', 'splitter': 'best'. Cross-validation is performed using the cross_val_score function from sklearn.model_selection module with ShuffleSplit as the cross-validation strategy. Cross-validation was utilized to evaluate the Decision Tree model's performance on multiple data subsets and prevent overfitting.

- **XGBoost Decision Tree Regression**: A XGBoost decision tree model is trained using the XGBRegressor class from the xgboost module. The hyperparameters for the model are set to 'booster': 'gbtree', 'objective': 'reg:squarederror', 'random_state': 42. Booster hyperparameter set to 'gbtree', which will make XGBoost train a decision tree instead of a gradient boosting model.
  Cross-validation is performed using the cross_val_score function with ShuffleSplit as the cross-validation strategy. Cross-validation was utilized to evaluate the XGBoost Decision Tree model's performance on multiple data subsets and prevent overfitting. The XGBoost library is chosen for this study due to its popularity in regression

tasks owing to its flexibility, scalability, and efficiency.

- **XGBoost Random Forest Regression**: XGBoost random forest model is trained to predict the price of a house. The XGBoost library, a popular gradient-boosting framework known for its high performance and scalability, is used to implement the model.
  To optimize the XGBoost model's performance, hyperparameter tuning is conducted by varying the number of estimators, maximum depth, and learning rate. The purpose of the hyperparameter tuning was to find the optimal values that would maximize the R-squared score.

  To further evaluate the XGBoost random forest model's performance, K-fold cross-validation is used, where K is set to 5. The ShuffleSplit cross-validator is used to randomly split the data into training and test sets for each iteration. The purpose of using cross-validation was to assess the model's performance on different subsets of the data and avoid overfitting.
- **XGBoost Regression from XGBoost Library**: XGBoost library is utilized to predict the price of a house using machine learning. The hyperparameters for the XGBoost model are set by specifying the objective as 'reg:squarederror' and setting the random_state to 42. The hyperparameters are tuned to obtain the optimal values that maximize the R-squared score.
  The ShuffleSplit cross-validation iterator with five splits and a test size of 0.3 is used to train and evaluate the XGBoost model. The R-squared scoring metric is used to compute the cross-validation scores.
  The hyperparameter tuning process helped to optimize the model's performance and increase its accuracy. The k-fold cross-validation technique is used to evaluate the model's performance and avoid overfitting. The XGBoost library is selected as it is a popular choice for regression tasks due to its efficiency, scalability, and flexibility.
- **Randomized Search CV**: A randomized search is performed to find the best hyperparameters for two models: RandomForestRegressor and XGBRegressor. The models are evaluated using the mean cross-validation score. Hyperparameter tuning is performed for the XGBoost decision tree and the RandomForestRegressor using a randomized search CV.
  For instance, for the random_forest model, the hyperparameters to be tuned are n_estimators (number of trees in the forest), max_features (maximum number of features to consider when splitting a node), max_depth (maximum depth of the tree), min_samples_split (minimum number of samples required to split an internal node), and min_samples_leaf (minimum number of samples required to be at a leaf node).
  Similarly, for the xgboost model, the hyperparameters to be tuned are learning_rate (step size shrinkage used in each boosting iteration), max_depth (maximum depth of the tree), n_estimators (number of boosted trees to fit), and objective (the loss function to be minimized).

  The RandomizedSearchCV class from the sklearn.model_selection module is used with a ShuffleSplit strategy with 5 splits, and the number of iterations is set to 5. The hyperparameters and their respective values are specified in a dictionary format.

IV. Results and Analysis

A. *Bengaluru House Price Dataset Analysis*

- The Decision Tree model from the SKlearn library achieves a relatively high R-squared score of 0.91 on the training data, indicating a good fit to the data (**Fig 45**). On the test data, the model achieved an R-squared score

```
Deciosion Tree R-squared score on training data: 0.9098087476703975
```

Fig. 45: Decision Tree R-square Score on Training Data

of 0.76, indicating that the model has some overfitting issues but still has good predictive performance (**Fig 46**). The Mean Squared Error (MSE) on the test data

```
Decision Tree R-squared score on test data: 0.7620414174211332
Decision Tree MSE on test data: 0.10
Decision Tree RMSE on test data: 0.32
Decision Tree MAE on test data: 0.22
```

Fig. 46: R-square, RMSE, MSE, and MSE Score on Test Data

is 0.10, indicating that the model's predictions are on average within 0.32 of the actual target values, which is relatively small. The Mean Absolute Error (MAE) is 0.22, indicating that the average absolute difference between the predicted and actual values is relatively small.

The Decision Tree model is evaluated using cross-validation with a ShuffleSplit strategy to assess its generalization performance and potential overfitting issues. The model achieved a mean R-squared score of 0.78 across all cross-validation folds, which is comparable to the R-squared score achieved on the test data (**Fig 47**). The

```
Decision Tree Cross-Validation Scores: [0.77032549 0.76034103 0.7932489  0.79664548 0.7823523 ]
Mean R-squared: 0.7805826413888834
```

Fig. 47: Decision Tree K-Fold Score

fact that the model achieved a similar R-squared score on the test data and in cross-validation suggests that it has good generalization performance and is not overfitting the training data.

The scatter plot reveals that the model has a good predictive performance, with the majority of the data points closely clustered around the regression line (**Fig 48**).

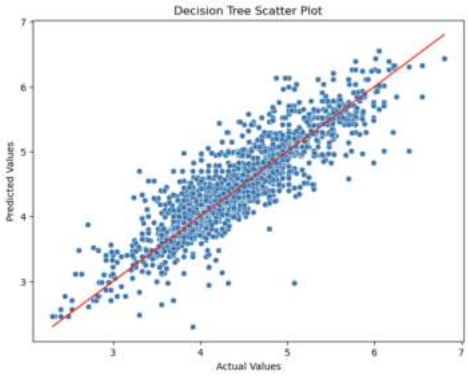

Fig. 48: Decision Tree Scatter Plot from Scikit-learn

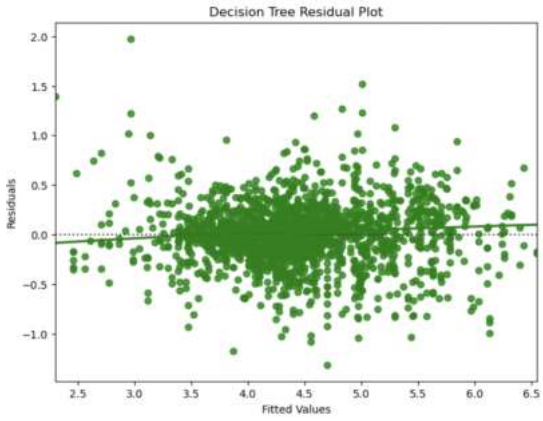

Fig. 49: Decision Tree Residual Plot from Scikit-learn

The residual plot shows that the majority of the data points are closely clustered around the horizontal line at zero, indicating that the model is making accurate predictions.

However, there are some outliers with high residuals that suggest that the model is not able to capture all of the complexity of the data (**Fig 49**). However, the Decision Trees model is prone to overfitting according to the result, particularly may not generalize well to new data.

- The XGBoost decision tree model achieved an R-squared score of 0.8926 on the test set, indicating that the model explains around 89% of the variance in the target variable. The model's mean squared error (MSE) and root mean squared error (RMSE) on the test set were 0.05 and 0.21, respectively, which are relatively low values. The mean absolute error (MAE) of the model on the test set was 0.16, indicating that the model's predictions were typically off by around INR 16 lakhs (**Fig 50**). The cross-validation scores showed consistent

```
XGBoost Decision Tree R-squared score on test data: 0.8926010046566053
XGBoost Decision Tree MSE on test data: 0.05
XGBoost Decision Tree RMSE on test data: 0.21
XGBoost Decision Tree MAE on test data: 0.16
```

Fig. 50: R-square, RMSE, MSE, and MSE Score on Test Data

performance across different folds, with a mean R-squared score of 0.886. This indicates that the model is reliable and generalizes well to new data (**Fig 51**). The scatter plot shows a positive correlation between

```
XGBoost Decision Tree Cross-Validation Scores: [0.8858715  0.8750951  0.8865578  0.8846079  0.89743973]
Mean R-squared: 0.8859144060444463
```

Fig. 51: Cross Validation Score

the predicted and actual values, indicating that the model's predictions align well with the regression line (**Fig 52**). The residual plot indicates that the residuals

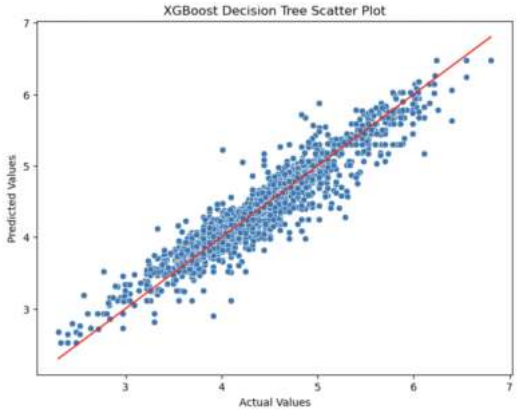

Fig. 52: Scatter Plot of XGBoost Decision Tree

are randomly scattered around zero, indicating that the model's errors are normally distributed (**Fig 53**). Overall,

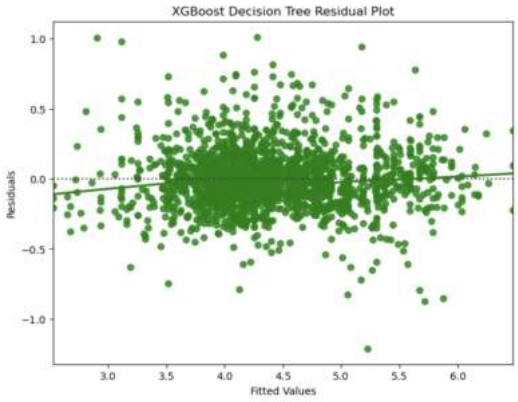

Fig. 53: Residual Plot of XGBoost Decision Tree

the XGBoost decision tree model appears to be a strong candidate for predicting housing prices, given its high accuracy and robust performance on the test set.

Additionally, the XGBoost decision tree model outperformed the decision tree from the Sklearn library in terms of accuracy and robustness, making it a better candidate for predicting housing prices.

- The results of XGBoost and Random Forest from the scikit-learn library indicate that the XGBoost model outperforms the Random Forest model in predicting housing prices, achieving a mean cross-validation score of 0.87 compared to 0.72 for the Random Forest (**Fig 54**). Also, The XGBoost model has a slightly higher R-squared value than the Random Forest, indicating that it is a better fit for the data.

  The cross-validation technique is used to estimate

```
random_forest Mean Cross-Validation Score: 0.72
random_forest Cross-Validation Scores: [0.53120126 0.79649308 0.80050501 0.72579752 0.75414097]
xgboost Mean Cross-Validation Score: 0.87
xgboost Cross-Validation Scores: [0.8697647  0.85974775 0.88222853 0.87623886 0.86270797]
```

	model	best_score	best_params	mse	rmse	mae
0	random_forest	0.800505	{'n_estimators': 50, 'min_samples_split': 10, ...	0.072184	0.268670	0.204649
1	xgboost	0.882229	{'objective': 'reg:squarederror', 'n_estimator...	0.038303	0.195713	0.146061

Fig. 54: Error Metrics and R-Square Score

the performance of the models on unseen data. The mean cross-validation score provides an estimate of the expected performance of the model on new data.

The XGBoost model also has lower values of mean squared error (MSE), root means squared error (RMSE), and mean absolute error (MAE) than the Random Forest. The RandomizedSearchCV algorithm is used to optimize the hyperparameters of the models, and the best parameters are used for training the final models.

The scatter plot shows the predicted values on the y-axis and the true values on the x-axis. A perfect model would have all the points lying on a straight line with a slope of 1. The scatter plot for both models shows a reasonably linear relationship between the true and predicted values, indicating that the models are doing a decent job of predicting housing prices (**Fig 55**). The residual plot for both

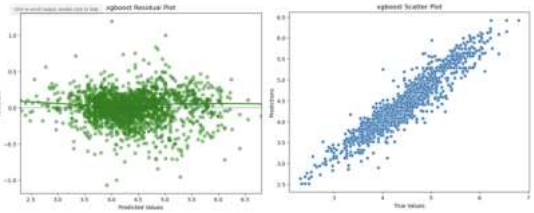

Fig. 55: Scatter and Residual Plot of XGBoost Model

models shows that the residuals are reasonably randomly scattered around the x-axis, indicating that the models are doing a good job of predicting housing prices (**Fig 56**). However, in the Random Forest model, the residual plot shows a slightly larger spread of residuals, with some values concentrated around the edges. Additionally, the x-axis for the Random Forest residual plot starts at 0.25, indicating that some of the predictions may be biased towards higher values.

The strengths of the XGBoost model are its ability to handle missing values, and complex interactions between the features. The Random Forest model, on the

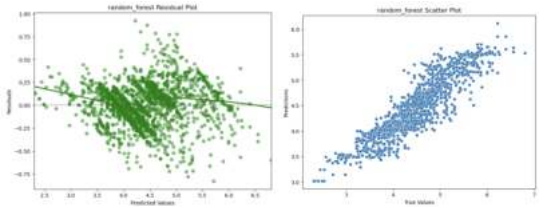

Fig. 56: Scatter and Residual Plot of Random Forest Model

other hand, is less prone to overfitting and can handle high-dimensional datasets.

Overall, both models demonstrate strong performance, with XGBoost outperforming Random Forest in terms of mean cross-validation score and error metrics.

- XGBoost library is chosen for its ability to handle complex and high-dimensional datasets while providing high accuracy. Again, XGBoost outperformed Random Forest, achieving a higher R-squared score of 0.8926 on the test data compared to 0.8194 for Random Forest (**Fig 57**). The mean cross-validation R-squared score of

```
XGBoost R-squared score on test data: 0.8926010046566053
XGBoost MSE on test data: 0.05
XGBoost RMSE on test data: 0.21
XGBoost MAE on test data: 0.16
XGBoost Random Forest R-squared score on test data: 0.8193966281638057
XGBoost Random Forest MSE on test data: 0.08
XGBoost Random Forest RMSE on test data: 0.28
XGBoost Random Forest MAE on test data: 0.21
```

Fig. 57: R-Square and Error Metrics

XGBoost is 0.8859, indicating that the model generalizes well to unseen data. Both models have low MSE, RMSE, and MAE values on the test data, indicating that they are able to predict housing prices with high accuracy (**Fig 58**). Cross-validation is performed for both models

```
XGBoost Cross-Validation Scores: [0.8858715  0.8750951  0.8865578  0.8846079  0.89743973]
Mean R-squared: 0.8859144060444463
XGBoost Random Forest Cross-Validation Scores: [0.80810227 0.80208256 0.8181131  0.80805847 0.82762849]
Mean R-squared: 0.8127969781543835
```

Fig. 58: Cross Validation Score

to ensure that the models are not overfitting the training data. The mean R-squared score from cross-validation is high for both models, indicating that they were not overfitting.

The scatter plots showed that the predicted values are close to the actual values for both XGBoost and Random Forest (**Fig 59**). The residual plots showed that the residuals are normally distributed around zero, indicating that the models have captured the underlying patterns in the data (**Fig 60**).

Overall, the results suggest that XGBoost is a powerful and accurate model for predicting housing prices while using the XGBoost library.

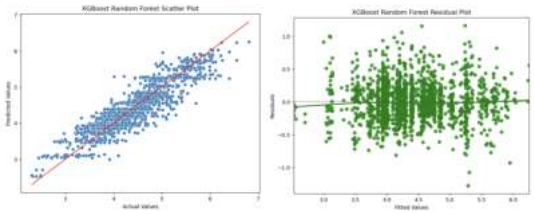

Fig. 59: Scatter and Residual Plot of Random Forest Model from XGBoost Library

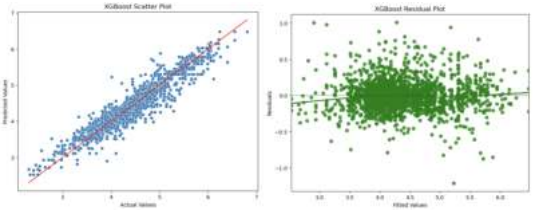

Fig. 60: Scatter and Residual Plot of XGBoost Model from XGBoost Library

B. House Rent Dataset Analysis

- The Scikit-learn library is used to implement the models due to its ease of use and robustness. The study evaluates the performance of three models, namely Decision Tree, Random Forest, and XGBoost to predict house prices using the second dataset.

  The Decision Tree model had an R-squared score of 0.55, indicating poor performance, with a mean cross-validation score of 0.57. The MSE, RMSE, and MAE on the test data were 0.00044, 0.02096, and 0.01553, respectively (**Fig 61**). Although both the Random Forest

```
Decision Tree R-squared score on test data: 0.5513641980995945
Decision Tree MSE on test data: 0.00044
Decision Tree RMSE on test data: 0.02096
Decision Tree MAE on test data: 0.01553
```

Fig. 61: R-Square and Error Metrics of Decision Tree

and XGBoost models performed well with a mean cross-validation score of 0.78. The Random Forest model has the lowest mean squared error (MSE) and means absolute error (MAE), indicating better predictive accuracy than the other models. (**Fig 62**) The scatter

```
random_forest Mean Cross-Validation Score: 0.78
random_forest Cross-Validation Scores: [0.7768445  0.78136092 0.77744348 0.77561405 0.77542818]
xgboost Mean Cross-Validation Score: 0.78
xgboost Cross-Validation Scores: [0.78357974 0.78626459 0.77902392 0.77460149 0.78678184]
```

	model	best_score	best_params	mse	rmse	mae
0	random_forest	0.781361	{'n_estimators': 150, 'min_samples_split': 2, ...	0.000029	0.005431	0.003988
1	xgboost	0.786782	{'objective': 'reg:squarederror', 'n_estimator...	0.000170	0.013036	0.009719

Fig. 62: R-Square and Error Metrics of Random Forest and XGBoost

plots show how well the predicted values match the true values, and the residual plots show the distribution of the errors (**Fig 63**). The scatter plots for the Random Forest and XGBoost models show a more linear relationship between the predicted and true values, while the scatter

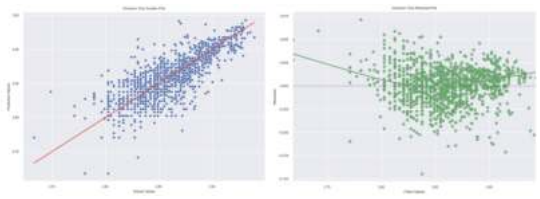

Fig. 63: Scatter and Residual Plot of Decision Tree Model

plot for the Decision Tree model seems to have a wider spread. (**Fig 64**).

The residual plots for the Decision Tree, Random

Fig. 64: Scatter and Residual Plot of Random Forest Model

Forest, and XGBoost models indicate that the models are capturing some of the patterns in the data. While the Decision Tree model, Random Forest, and XGBoost models seem to show homoscedasticity (**Fig 65**).

Overall, the Random Forest and XGBoost models

Fig. 65: Scatter and Residual Plot of XGBoost Model

performed well in predicting the rent of houses in India, with the Random Forest model having the same mean cross-validation score and R-Square score as XGBoost but the lowest error metrics comparing the XGBoost model. Hence, Random Forest is the best-suited model for the second dataset while using the Scikit-learn library.

- The XGBoost library is used to develop three models, namely XGBoost Decision Tree, XGBoost Random Forest, and XGBoost. All three models are evaluated using various metrics such as R-squared score, mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE) on the test data.

  The XGBoost Decision Tree model achieved the highest R-squared score of 0.776 and the lowest MSE and RMSE values same as the XGBoost model. However, the XGBoost Random Forest model also performed well with R-squared scores of 0.766 (**Fig 66**). The models

```
XGBoost Decision Tree R-squared score on test data: 0.7760200114986203
XGBoost Decision Tree MSE on test data: 0.00022
XGBoost Decision Tree RMSE on test data: 0.01481
XGBoost Decision Tree MAE on test data: 0.01110
XGBoost Random Forest R-squared score on test data: 0.7663358475200389
XGBoost Random Forest MSE on test data: 0.00023
XGBoost Random Forest RMSE on test data: 0.01513
XGBoost Random Forest MAE on test data: 0.01143
```

Fig. 66: R-Square and Error Metrics for Decision Tree and Random Forest Model

are also evaluated using cross-validation scores, which are consistent with the performance of the test data. The XGBoost Decision Tree and XGBoost models have the highest mean R-squared score of 0.7596, while the XGBoost Random Forest has a slightly lower mean R-squared score of 0.7587 (**Fig 67**). The scatter and

```
XGBoost Decision Tree Cross-Validation Scores: [0.76032994 0.75111058 0.7514529  0.76188652 0.77363239]
Mean R-squared: 0.7596824668311274
XGBoost Random Forest Cross-Validation Scores: [0.74237678 0.74699288 0.7537331  0.75745993 0.77911596]
Mean R-squared: 0.7587357291644775
```

Fig. 67: Cross Validation Score of Decision Tree and Random Forest Model

residual plots are also used to evaluate the performance of the models. The scatter plots show that the models are able to predict house rent prices with a reasonable degree of accuracy, as the predicted values are generally close to the actual values (**Fig 68**).

The residual plots show that the errors are normally

Fig. 68: Scatter and Residual Plot of Random Forest Model

distributed around zero and do not show any patterns, indicating that the residuals are randomly distributed(Homoscedasticity). Hence, this proves that the models are working well (**Fig 69**).

Overall, the XGBoost library performed well in

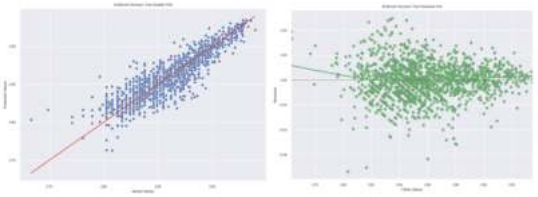

Fig. 69: Scatter and Residual Plot of Decision Tree Model

predicting house rent prices, with the XGBoost Decision Tree model achieving the highest performance.

C. Boston Housing Dataset Analysis

- In this study, three regression models (Decision Tree, Random Forest, and XGBoost) are implemented using the scikit-learn library to predict Boston housing prices.

The XGBoost model yields the best performance with a mean cross-validation score of 0.83 and the lowest mean squared error (0.88), followed by the random forest model with a mean cross-validation score of 0.77 and a mean squared error of 1.49 (**Fig 70**).

The Decision Tree model had a lower mean cross-

```
random_forest Mean Cross-Validation Score: 0.77
random_forest Cross-Validation Scores: [0.77523226 0.79831367 0.76795667 0.74157862 0.74760264]
xgboost Mean Cross-Validation Score: 0.83
xgboost Cross-Validation Scores: [0.82694524 0.82141411 0.82529979 0.82639922 0.83869846]
```

	model	best_score	best_params	mse	rmse	mae
0	random_forest	0.798314	{'n_estimators': 150, 'min_samples_split': 2, ...	1.494267	1.222402	0.794124
1	xgboost	0.838698	{'objective': 'reg:squarederror', 'n_estimator...	0.883734	0.940071	0.720922

Fig. 70: R-Square and Error Metrics of Random Forest and XGBoost Model

validation score of 0.73 and a mean squared error of 11.65, However, the Decision Tree model tends to overfit easily due to its high variance (**Fig 71**).

```
Decision Tree R-squared score on test data: 0.8436172098250776
Decision Tree MSE on test data: 11.65
Decision Tree RMSE on test data: 3.41
Decision Tree MAE on test data: 2.55
Decision Tree Cross-Validation Scores: [0.60495567 0.73714999 0.80768346 0.72988534 0.7863969 ]
Mean R-squared: 0.7332142746572263
```

Fig. 71: R-Square and Error Metrics of Decision Tree Model

The scatter plot and residual plot are useful for evaluating the model's performance. The scatter plot for the Decision Tree model indicates not so good fit (**Fig 72**), as the predicted values are a bit further than the actual values. The residual plot shows that the residuals are randomly distributed, which is desirable for a regression model **(Fig 73)**.

The scatter and residual plots for the best models

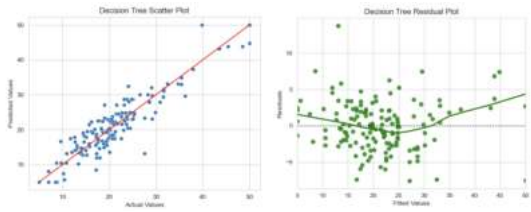

Fig. 72: Scatter and Residual Plot of Decision Tree Model

generated using RandomizedSearchCV indicate that the models have good fits and random residuals (**Fig 74**).

The scatter plots and residual plots of Random Forest and XGBoost models indicate that the predicted values are closely aligned with the actual values, as shown by the regression line, and the residuals are normally distributed, indicating that the models are well-fit and

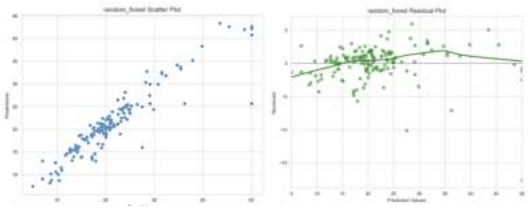

Fig. 73: Scatter and Residual Plot of Random Forest Model

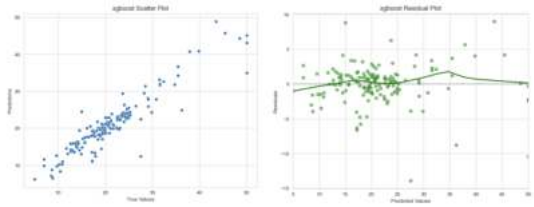

Fig. 74: Scatter and Residual Plot of XGBoost Model

accurate.
Overall, the XGBoost model performs the best while using the Scikit-learn library.

- In this study, the XGBoost library is utilized to build three regression models, namely the XGBoost Decision Tree, XGBoost Random Forest, and XGBoost.

The XGBoost Decision Tree and XGBoost Random Forest models produced R-squared scores of 0.871 and 0.876, respectively, while the XGBoost model generated an R-squared score of 0.871 on the test data (**Fig 75**).
The MSE, RMSE, and MAE metrics also indicate that

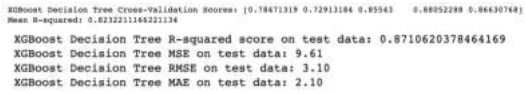

Fig. 75: R-Square and Error Metrics of Decision Tree Model

the models perform well in predicting the target variable, with the XGBoost Random Forest model having the lowest MSE of 9.22 and RMSE of 3.04 (**Fig 76**).
The cross-validation scores suggest that the models are

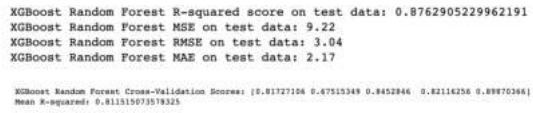

Fig. 76: R-Square and Error Metrics of Random Forest Model

stable and not overfitting, with the XGBoost Decision Tree and XGBoost models having a mean R-squared of 0.823 and the XGBoost Random Forest having a slightly lower mean R-squared of 0.811 (**Fig 77**).
Overall, The XGBoost Random Forest model has the highest R-squared score of 0.876 and the lowest MSE and RMSE metrics of 9.22 and 3.04, respectively.

Fig. 77: R-Square and Error Metrics of XGBoost Model

However, the cross-validation scores for this model suggest that it may be slightly less stable than the other two models, with a mean R-squared of 0.811 (**Fig 78**).
On the other hand, the XGBoost Decision Tree and

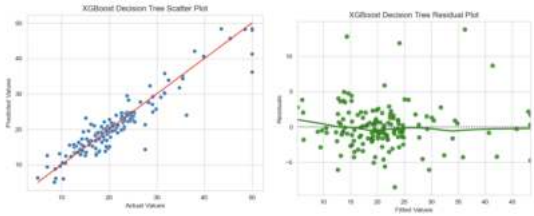

Fig. 78: Scatter and Residual Plot of Decision Tree Model using XGBoost Library

XGBoost models have a mean R-squared of 0.823 in cross-validation, which indicates they are more stable than the XGBoost Random Forest model (**Fig 79**).
The scatter and residual plots are created to evaluate

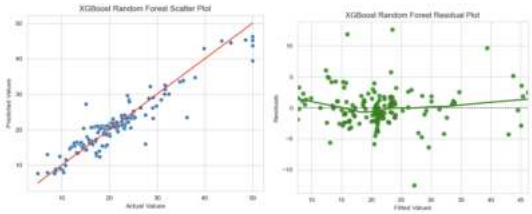

Fig. 79: Scatter and Residual Plot of Random Forest Model using XGBoost Library

the performance of the models visually. The scatter plots indicate that the predicted values are highly correlated with the actual values, with the majority of the data points falling near the regression line (**Fig 80**).
The residual plots indicate that the models are unbiased,

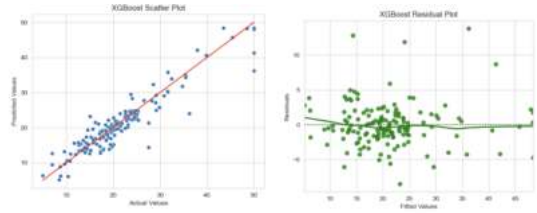

Fig. 80: Scatter and Residual Plot of XGBoost Model using XGBoost Library

as the residuals are evenly distributed around the zero line, and there is no apparent pattern.
However, the XGBoost Decision Tree and XGBoost models are better suited in situations where stability and efficiency are a priority.

V. Conclusion

This study analyzes three different datasets to predict the rent or price of houses using decision tree, random forest, and XGBoost algorithms from Sklearn and XGBoost libraries.

The results show that the XGBoost and XGBoost Decision Tree algorithm performs better overall compared to the Random Forest algorithm, achieving higher R-squared scores and lower errors (MSE, RMSE, and MAE).

Both XGBoost and XGBoost decision tree algorithms work similarly, Although the XGBoost decision tree uses a single decision tree instead of an ensemble of trees, which reduces the training time and memory usage and XGBoost uses decision trees as base estimators in gradient boosting.

The decision tree algorithm in XGBoost is a simpler and faster version of the full XGBoost algorithm. Additionally, the decision tree algorithm provides better interpretability as it creates a single tree that can be easily visualized and analyzed.

Regarding the libraries, the results are consistent with the XGBoost library performing better overall on all datasets compared to scikit-learn. Although the scikit-learn random forest algorithm performs well on the House Rent dataset, it doesn't generalize well on other datasets.

On the other hand, XGBoost's decision tree and random forest algorithms perform well on all datasets, indicating the robustness and generalization of the XGBoost library.

In conclusion, this study suggests that XGBoost's decision tree algorithm has demonstrated superior performance and generalization capabilities in predicting house prices and rents, making it a recommended choice for accurate and reliable predictions. Incorporating the XGBoost library enhances the model's accuracy and generalization, leading to improved performance in various datasets.

VI. Future Work

The limitations of this study include the limited number of datasets used for analysis, which may not be representative of all housing markets. Additionally, the study only considers decision tree, random forest, and XGBoost algorithms from Sklearn and XGBoost libraries, neglecting other machine learning algorithms that could potentially perform better.

Future research could expand on this study by using more datasets from diverse housing markets, and including other machine learning algorithms to compare and evaluate their performance. Another avenue for future research is to explore the impact of different hyperparameters on the performance of the XGBoost algorithm and compare it with other algorithms. Additionally, incorporating external factors such as economic indicators and demographic data could further improve the accuracy and robustness of the models.

References

[1] Palak Furia, Anand Khandare, "Real Estate Price Prediction Using Machine Learning Algorithms", [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119792437.ch2/

[2] Ali Louati, Saad Otai, Abdulaziz Aldaej, "Price forecasting for real estate using machine learning: A case study on Riyadh city", [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.6748/

[3] "KDD Process in Data Mining" [Online]. Available: https://www.geeksforgeeks.org/kdd-process-in-data-mining//

[4] "Bengaluru House price data" [Kaggle]. Available: https://www.kaggle.com/datasets/amitabhajoy/bengaluru-house-price-data/

[5] "House Rent Dataset" [Kaggle]. Available: https://www.kaggle.com/datasets/iamsouravbanerjee/house-rent-prediction-dataset?select=House$_Rent_Dataset.csv/$

[6] Prashant Gupta, "Decision Trees in Machine Learning" [Online]. Available: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052/

[7] "Introduction to Random Forest in Machine Learning" [Online]. Available: https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/

[8] "What is XGBoost? An Introduction to XGBoost Algorithm in Machine Learning " [Online]. Available: https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article/

[9] "What is Scikit-Learn? " [Online]. Available: https://www.codecademy.com/article/scikit-learn/

[10] "Machine Learning with XGBoost and Scikit-learn " [Online]. Available: https://www.section.io/engineering-education/machine-learning-with-xgboost-and-scikit-learn/: :text=XGBoost