LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

Level 6 – CC6057NI Applied Machine Learning

Assessment Type

Milestone 1

Semester

2024/25 Autumn

Student Name: Bibek Poudel

London Met ID: 22067316

College ID: np01ai4a220032@islingtoncollege.edu.np

Assignment Due Date: Monday, December 23, 2024

Assignment Submission Date: Monday, December 23, 2024

Submitted To: Mahotsav Bhattarai

Word Count:

I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Table of Contents

Table of Figures

1. Introduction

1.1. Overview

This coursework showcases the implementation of classical machine learning algorithms to predict the price of different houses from **Kathmandu, Nepal** is performed through this coursework. This prediction is performed using the two different models for comparative study in house prediction systems using different machine learning algorithms. The two different algorithms used in this coursework are Linear Regression and Extreme Gradient Boost. These two different algorithms will be used for training the model on the researched dataset with various data preprocessing and feature engineering process along with required visualization. This coursework will also implement the basic hyper parameter tuning along with optimization for the enhanced model performance and robustness on the unseen dataset.

1.2. Data Collection, Preprocessing & Feature Engineering

Data collection is the initial step for any machine learning and data analysis project, the data collected might contain various noise, null values, outliers which are dealt by data preprocessing step. The data collection is the important step for development of prediction models. For this coursework, data collection involves the gathering of the relevant data for the prediction system with the features like price of the house, different facilities in house along with the location of the house (Jain, Sep, 2024). Data collection is followed by data preprocessing step where the noise and outliers contained data are dealt with proper steps. Ensuring the quality of the data before model training, handling of the messy data by reducing the noise, inconsistencies which can lead to inaccurate model prediction. Concepts like outlier removal, smoothing out the data, normalization of data, transformation and integration are also used on the dataset during the data preprocessing step to ensure the quality of the dataset (Novogroder, April,2024).

Feature Engineering is the step where the performance of the machine learning models is improved by the process of selection, modification and creation of new features from the dataset occur. The dataset used in this project will also requires the feature engineering steps to encode the categorical features such as type of apartment, scaling and normalizing the numerical values and detecting the outliers. This will allow the machine learning model trained on the dataset to have the robust accuracy and performance (Fluhler, Feb, 2024).

1.3. Classical Machine Learning Algorithms

This coursework will utilize two different algorithms for the prediction process along with the comparative study of the performance of both algorithms in house prediction dataset from Kathmandu, Nepal.

1.3.1. Linear Regression

Linear Regression is the most fundamental and starting point for most of the machine learning algorithms. The concepts such as finding the best fit line, gradient descent, minimizing the loss are introduced in this algorithm which allowed the creation of many machine learning and deep learning algorithms for various problems. The prediction system for this coursework will also utilizes the linear regression algorithm which will compete with Decision Tree for better performance on the pre-processed data. Linea regression is a go-to model for prediction tasks as well (WLLber, Sep, 2022).

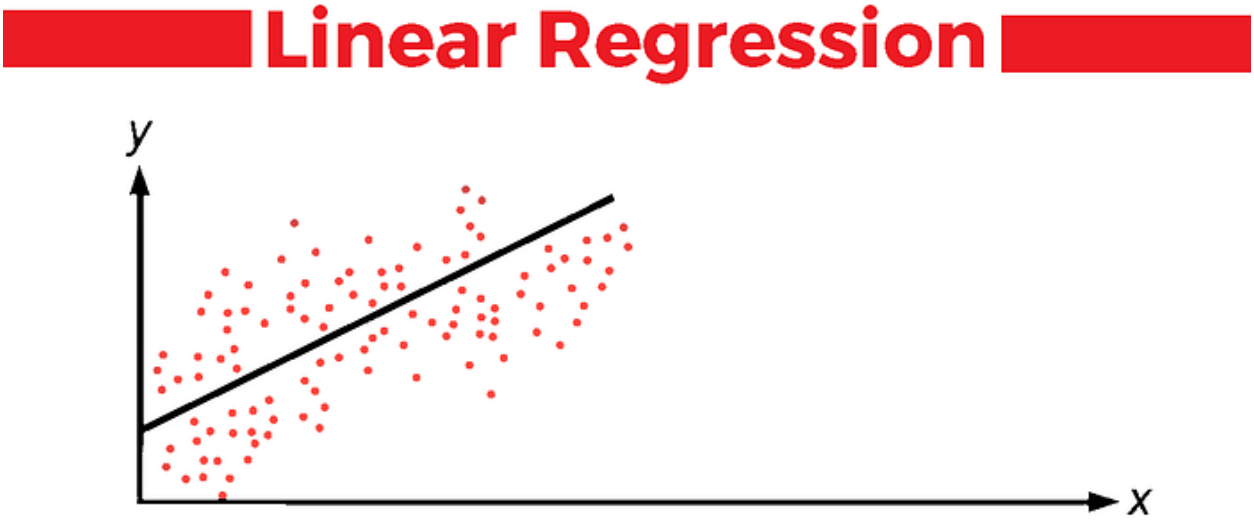


Figure 1: Linear Regression

1.3.2. Extreme Gradient Boost

XGBoost is a robust and advanced machine learning technique for regression tasks, such as house price prediction. It combines an objective function with loss functions like RMSE (Root Mean Squared Error) and MAE (Mean Average Error) and regularization (L1, L2) to ensure model simplicity and prevent overfitting. XGBoost uses ensemble learning to aggregate predictions from weak base learners, enhancing accuracy. Features like hyperparameter tuning, tree pruning, similarity scores, and second-order Taylor approximations improve its performance and adaptability. Additionally, the use of DMatrix optimizes computations, making XGBoost efficient and highly feasible for real-world regression problems (Hachcham, Sep, 2024).

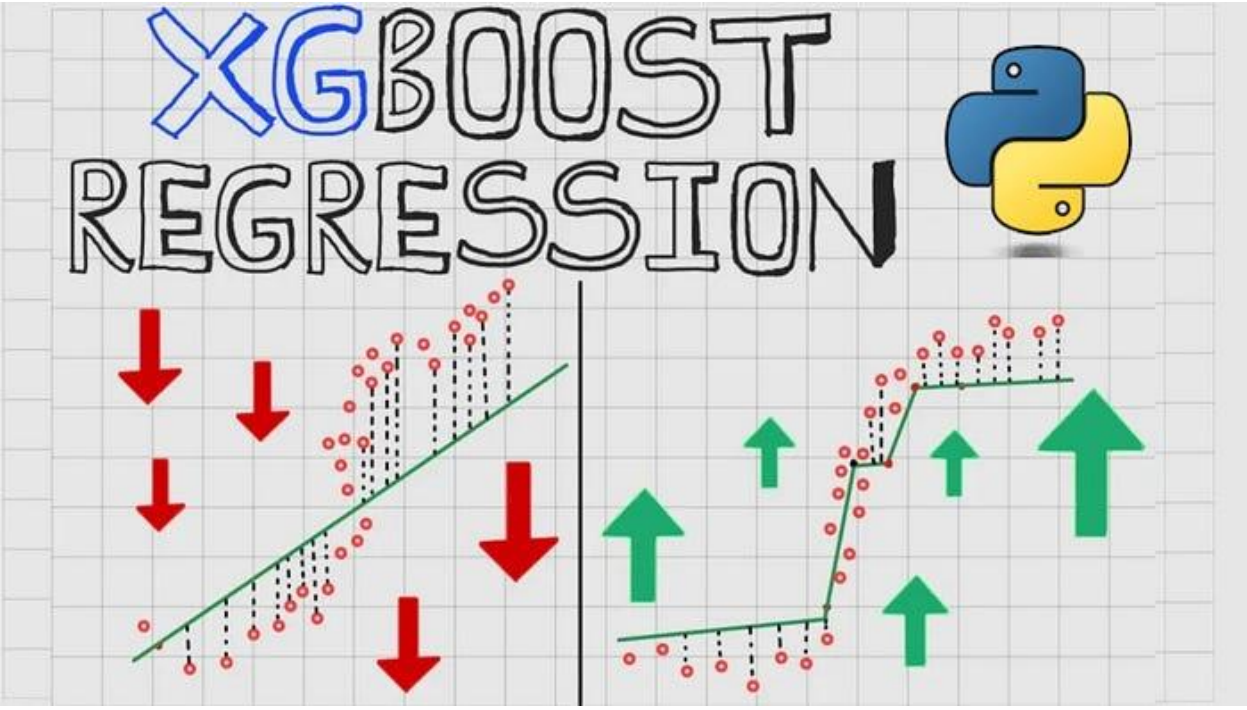


Figure 2: XGBoost for Regression

1.4. Encoding

Encoding is defined as the technique to convert the categorical data into numerical data. Machine learning algorithms are mostly based on numerical data as the input which will requires the conversion of categorical features like location, road type and so on. These features will contain the categorical values which are encoded into numerical values using various encoding techniques such as One-Hot Encoding, Label Encoding and Ordinal Encoding. Each methods have its own advantages and challenges and choosing the proper encoding technique will depends on the nature of dataset along with the machine learning model used. This coursework will utilize Label Encoding and One Hot Encoding to apply on the given dataset for the conversion of categorical column into numerical for the model training process (Lewinson, Feb 17, 2022).

2. Problem Domain

2.1. Problem Overview

The task of house price prediction using machine learning involves accurately estimating property values in Kathmandu's real estate market. By considering factors like location, size, and property condition, machine learning models can offer more reliable predictions than traditional valuation methods. This approach aims to improve the accuracy of price estimations, providing a clearer understanding of the housing market and enabling a meaningful comparison between different predictive models.

2.2. Dataset & Research

This coursework focuses on predicting property prices using a dataset of 1,624 property records. The dataset includes key attributes such as the number of bedrooms, bathrooms, parking spaces, land size, road type, and price. However, some columns, including "floors," "Aana," and "Paisa," have missing values that require data cleaning and preprocessing to ensure the accuracy of the models. Addressing these missing values is essential for building a reliable predictive model (Mahat, April, 2022).

For the prediction task, machine learning models such as linear regression and XGBoost are applied. Linear regression is used as a baseline model to understand the basic relationships between features and the target price. XGBoost, on the other hand, is a more advanced model known for its ability to handle complex interactions among features and efficiently manage missing data. The objective of this coursework is to optimize the predictive performance of these models, providing valuable insights into the real estate market and offering practical applications for decision-makers.

3. Approach and Methodology

3.1. Solution Approach

The house price prediction dataset chosen for this coursework will utilizes the machine learning algorithms discussed in earlier sections. Linear Regression and XGBoost will be used for the comparative study of the prediction models on the same dataset. These algorithms will be implemented with actual trained model for prediction using the libraries of Python programming language for data analysis and machine learning. Such libraries include, pandas, NumPy, Matplotlib and Scikit-Learn. These libraries will allow to correction preprocessing, exploration of dataset along with feature engineering, model training and model evaluation as well.

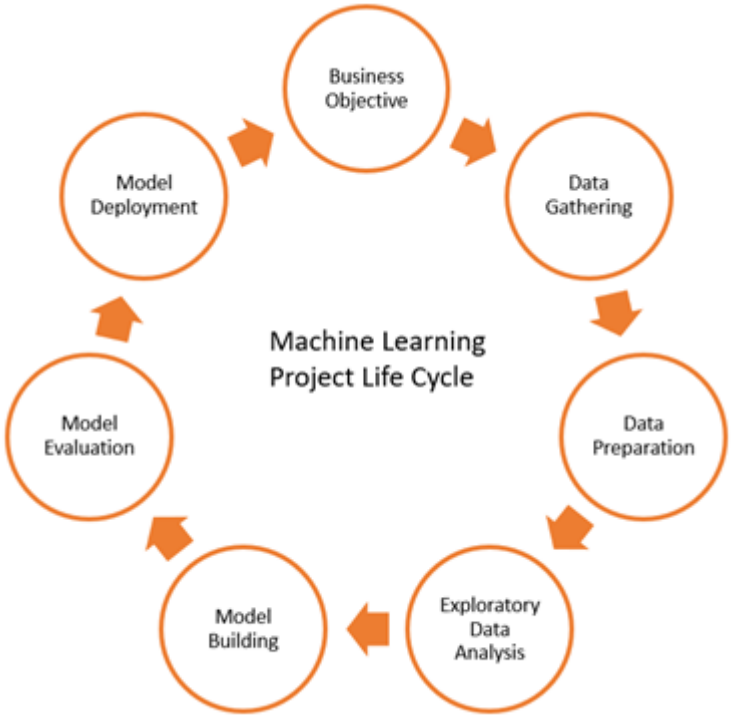


Figure 3: Machine Learning Life Cycle

Library Used

1. **Pandas**: Pandas is a Python library for data analysis, specializing in sequential and tabular data manipulation. Pandas offers features for reading diverse dataset formats, handling missing values, cleaning data, summarizing dataset statistics. Additionally, it also offers the functionalities of reshaping the datasets, performing various mathematical operations, aggregations and exporting data in various format which will be beneficial for this project of house price prediction (McLntire, 2024).

Figure 4: Pandas Python

2. **NumPy**: NumPy is an open-source Python library for scientific computing in machine learning and exploratory data analysis. It simplifies and ease the working with multidimensional arrays and offers wide range of mathematical operations for linear algebra, statistics, and data operations. Handling of larger datasets is also supported by NumPy using the vectorized operations, broadcasting and array slicing, leading to enhance machine learning models (Bourke, Nov, 2023).

Figure 5: NumPy Library

3. **Matplotlib**: Matplotlib is an open-source library for data visualization and plotting, aiding in data analysis and interpretation. It supports creating various visualizations, including line plots, scatter plots, bar charts, and histograms, with extensive customization options. In this coursework, Matplotlib will be used for feature engineering, preprocessing, and model evaluation after training on two algorithms (Bhandari, Oct, 2024).

Figure 6: Matplotlib Library

4. **Scikit-Learn**: Scikit-learn is a crucial Python library for machine learning and exploratory data analysis, offering efficient tools for classification, regression, clustering, and dimensionality reduction. It includes algorithms like linear regression, decision trees, SVMs, and PCA for unsupervised learning. Scikit-learn also provides compatibility with external libraries like XGBoost, enabling seamless integration for advanced gradient-boosting algorithms. Primarily focused on model building and training, Scikit-learn is indispensable for developing machine learning solutions (Jain, Oct, 2024).

Figure 7: Scikit-learn Python Library

3.2. Flowchart

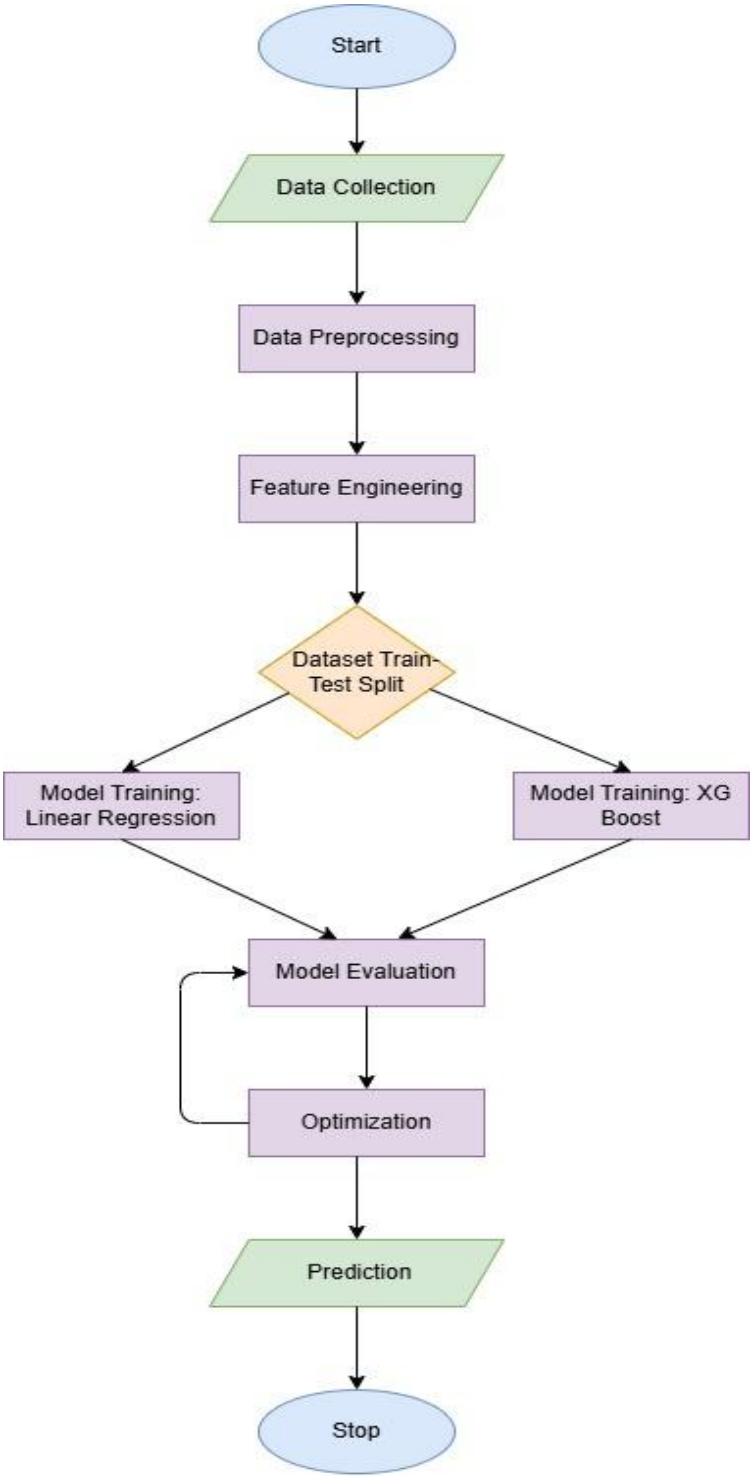


Figure 8: Flowchart

3.3. Pseudocode

Import Libraries

Load Dataset

Handle Missing Values

Encoded Categorical Variables

Drop Irrelevant Features

Create a New Feature

Plot Missing Values

Replace Missing Values for Floors

Split the Dataset

Normalize Data

Train Model

Linear Regression

Train a linear regression model on normalized data.

Calculate and print evaluation metrics: MAE, MSE, R^2, and Accuracy.

XGBoost Regressor

Train the XGBoost regressor with specified parameters.

Predict on test data and calculate R^2.

Tune XGBoost with GridSearchCV

Feature Importance

Visualization

Function Definitions

4. Results

namely Linear Regression and Extreme Gradient Boosting for Regression.

Importing Neccessary Libraries

```
[2]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.preprocessing import LabelEncoder
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
     import xgboost as xgb
     from sklearn.model_selection import GridSearchCV

     import warnings
     warnings.simplefilter(action='ignore')
```

Figure 9: Importing libraries

Loading and Exploring the dataset

```
[3]: house_df = pd.read_csv('kathmandudata.csv')
     house_df.head(15)
     house_df_1 = house_df.copy(0)
```

```
[4]: house_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1624 entries, 0 to 1623
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   bedroom         1624 non-null   int64
 1   bathroom        1624 non-null   int64
 2   floors          948 non-null    float64
 3   parking         1624 non-null   int64
 4   location        1624 non-null   object
 5   dhur            1622 non-null   object
 6   Aana            1166 non-null   float64
 7   Paisa           981 non-null    float64
 8   Daam            960 non-null    float64
 9   roadsize(feet)  1624 non-null   float64
 10  Road type       1624 non-null   object
 11  Years ago       1624 non-null   float64
 12  Area(sq ft)     1624 non-null   float64
 13  Price           1624 non-null   int64
dtypes: float64(7), int64(4), object(3)
memory usage: 177.8+ KB
```

Figure 10: Loading and exploring the dataset

Numerical Features and Categorical Features

```
[5]: def identify_features(dataset):
         numerical_features = dataset.select_dtypes(include=['number']).columns.tolist()
         categorical_features = dataset.select_dtypes(include=['object','category']).columns.tolist()

         print(f'Numerical Features: {numerical_features}')
         print(f'Categorical Features: {categorical_features}')

     identify_features(house_df)
     house_df_1.drop(columns=['location'], inplace=True)
```

```
Numerical Features: ['bedroom', 'bathroom', 'floors', 'parking', 'Aana', 'Paisa', 'Daam', 'roadsize(feet)', 'Years ago', 'Area(sq ft)', 'Price']
Categorical Features: ['location', 'dhur', 'Road type']
```

Figure 11: Identifying the numerical and categorical features

Checking unique values from Categorical Columns

```
[10]: categorical_columns = ['Road type']
for col in categorical_columns:
    print(f"Unique values in '{col}': {house_df_1[col].unique()}\n")
```

```
Unique values in 'Road type': ['Soil Stabilized' 'Gravelled' 'Blacktopped' 'Paved' 'Concrete' 'Alley']
```

The **location** column have alot of unique values which can not be practically encoded using any of the encoding techniques due to which, we create new feature named **location_category** by changing the locations like Kathmandu, Lalitpur and Bhaktapur to inside valley and other locations to outside the valley.

And **Road type** column have seven unique values in the ordinal order so Label Encoding will be used for encoding this categorical feature from the dataset.

Figure 12: Checking unique values from categorical columns

Dealing with Missing Values

```
[11]: house_df_1.isnull().sum()
```

```
[11]: bedroom             0
bathroom            0
floors            676
parking             0
dhur                2
Aana              458
Paisa             643
Daam              664
roadsize(feet)      0
Road type           0
Years ago           0
Area(sq ft)         0
Price               0
dtype: int64
```

Figure 13: Dealing with missing values

```
dtype: int64
```

```
[12]: def  plot_null_values(dataset):
          plt.figure(figsize=(12,8))
          sns.heatmap(dataset.isnull(), cbar=False, cmap='viridis', yticklabels=False)
          plt.title('Missing Values Heatmap')
          plt.xlabel('Columns')
          plt.ylabel('Rows')
          plt.show()

      plot_null_values(house_df_1)
```

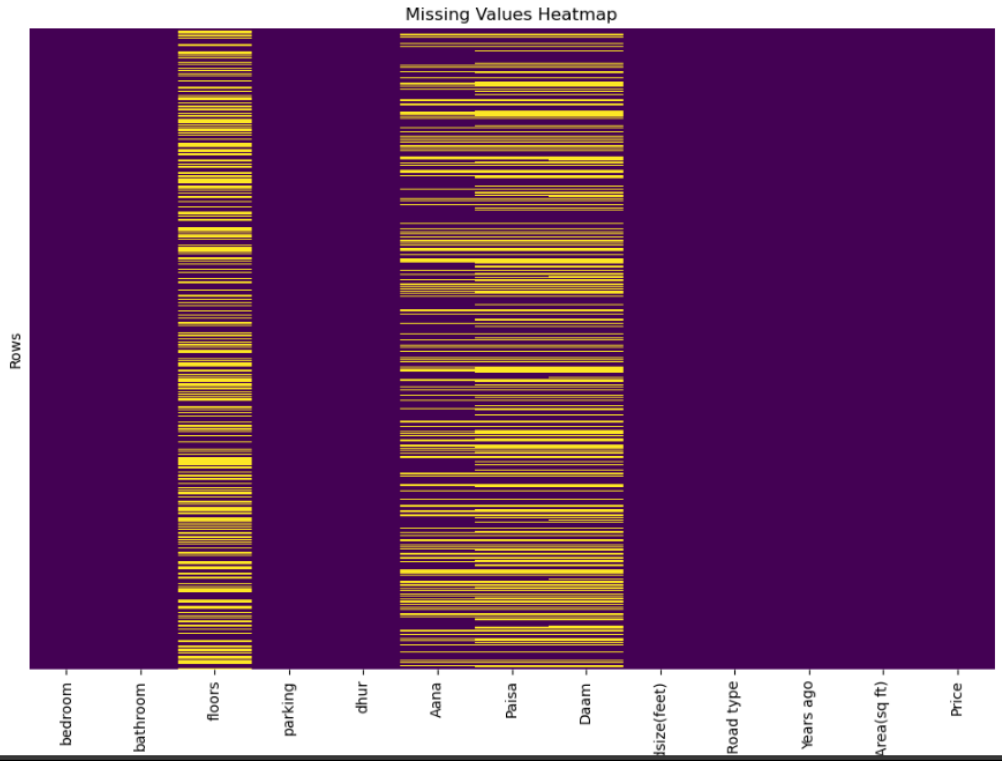


Figure 14: Visualizing missing values

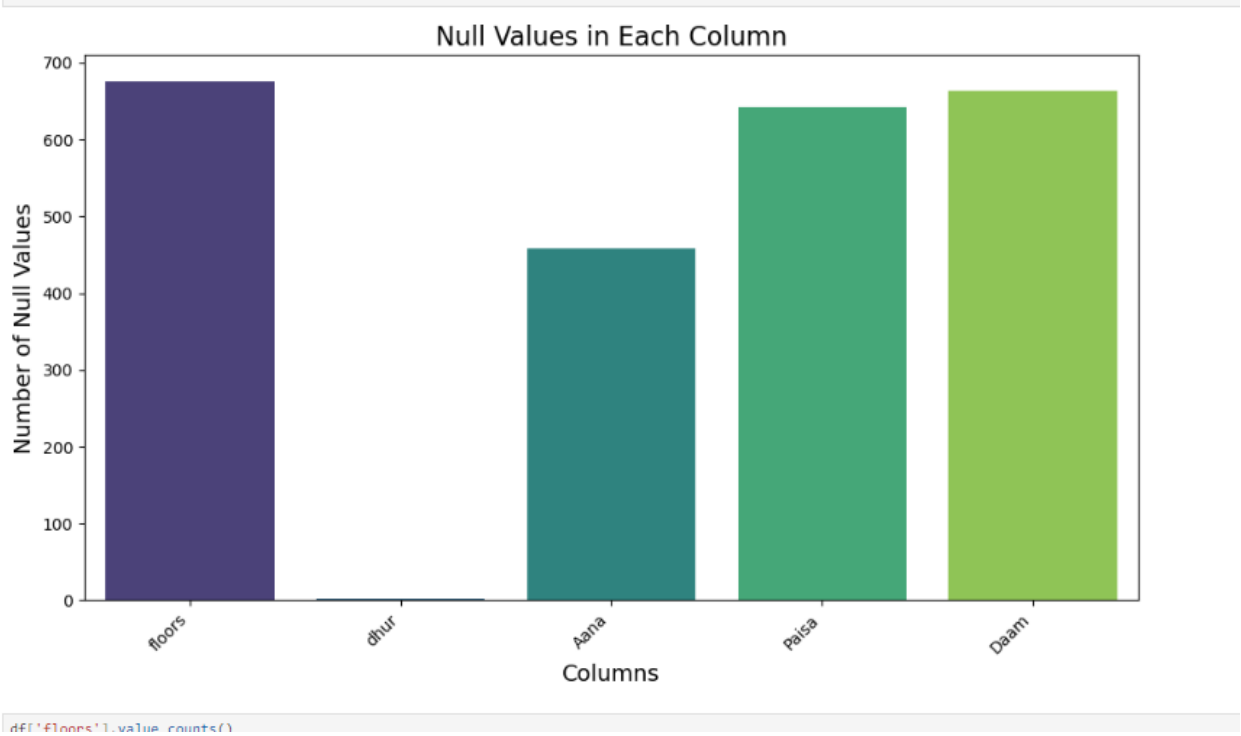


Figure 15:Null values in each column

```
[6]: array([[<Axes: title={'center': 'bedroom'}>,
            <Axes: title={'center': 'bathroom'}>,
            <Axes: title={'center': 'floors'}>],
           [<Axes: title={'center': 'parking'}>,
            <Axes: title={'center': 'Aana'}>,
            <Axes: title={'center': 'Paisa'}>],
           [<Axes: title={'center': 'Daam'}>,
            <Axes: title={'center': 'roadsize(feet)'}>,
            <Axes: title={'center': 'Years ago'}>],
           [<Axes: title={'center': 'Area(sq ft)'}>,
            <Axes: title={'center': 'Price'}>, <Axes: >]], dtype=object)
```

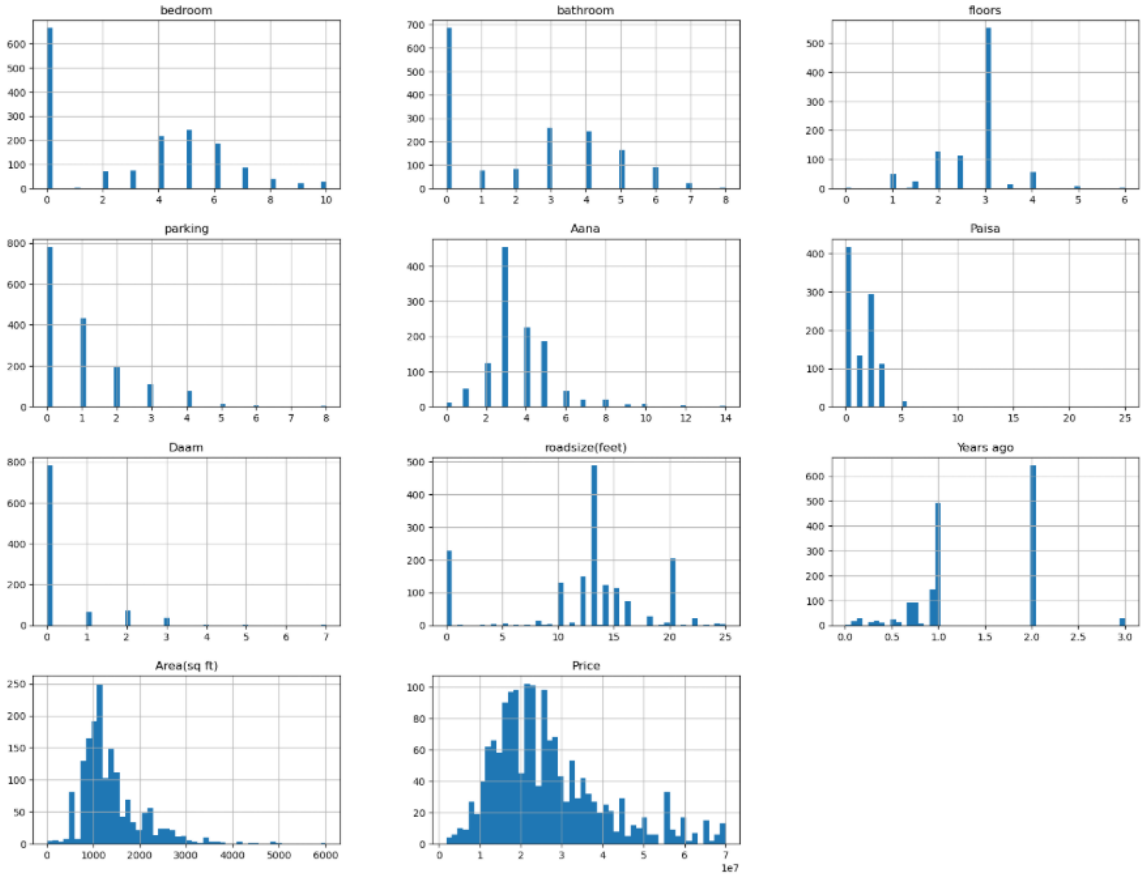


Figure 16: Distribution of the dataset

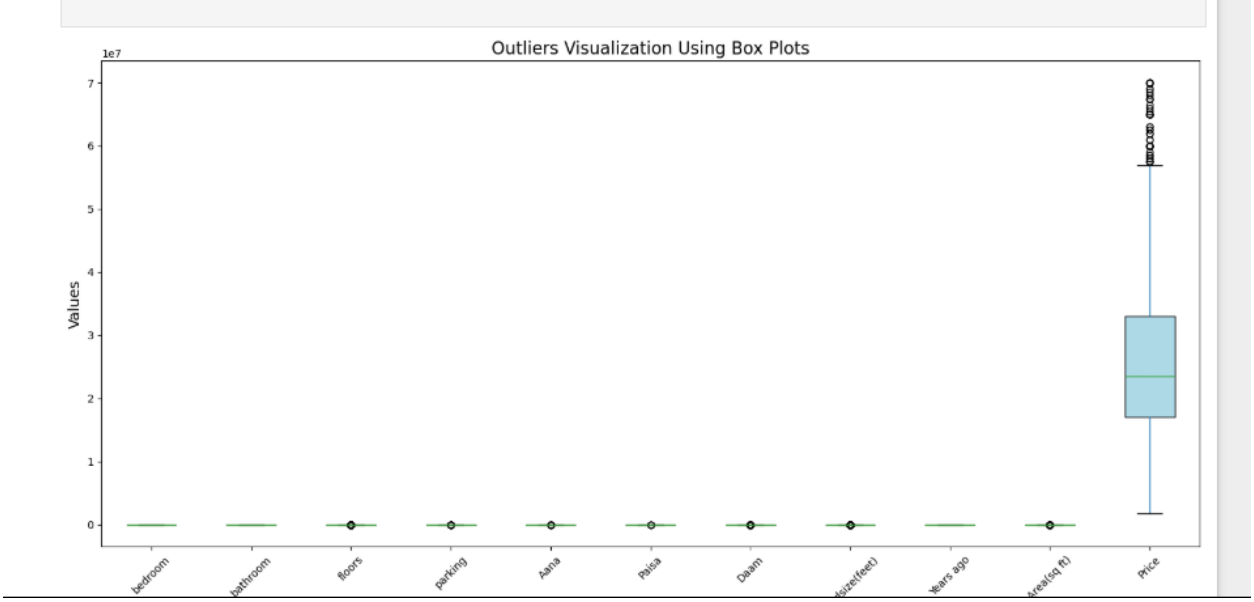


Figure 17: Outliers detection

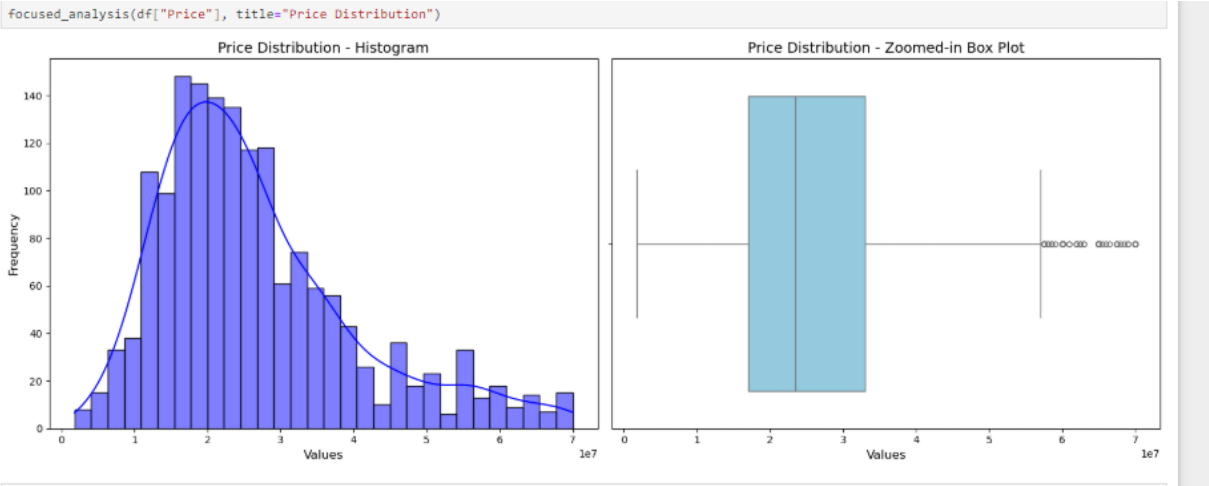
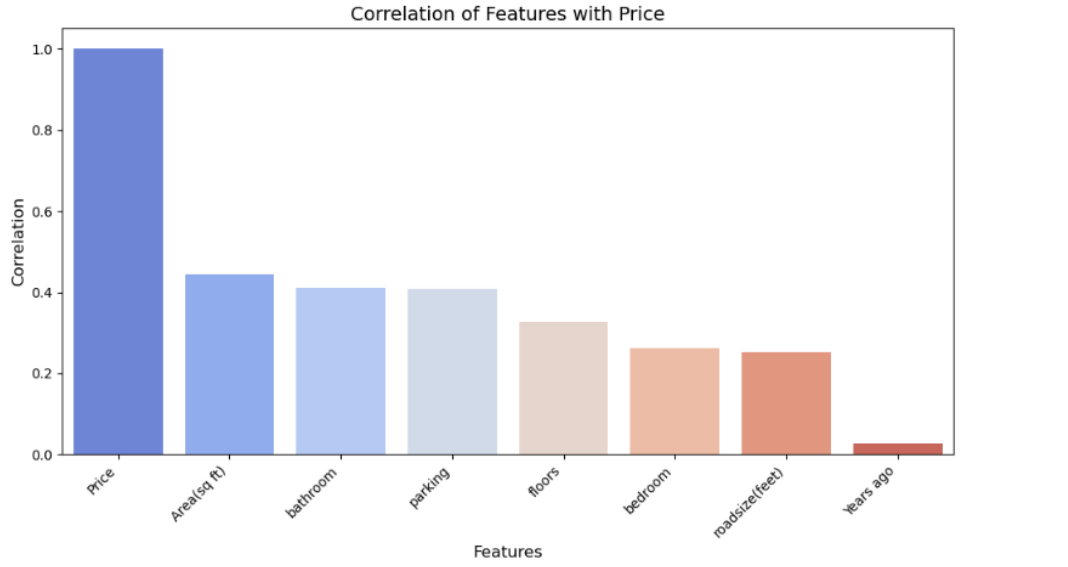


Figure 18: Price Column analysis



[20]: `df.isnull().sum()`

Figure 19: Correlation with Price

There are numerous null values in the columns such as Aana, Paisa, Daam and Floors. These values need to be dealt for preparating the clean data for machine learning procedure. We are replacing the Floor column with value 0 whereas for Aana,Paisa,Daam we are creating new column by using the metric system square meter conversion.

[13]:
```
house_df_1['dhur'] = pd.to_numeric(house_df_1['dhur'], errors='coerce')
house_df_1['Aana'] = pd.to_numeric(house_df_1['Aana'], errors='coerce')
house_df_1['Paisa'] = pd.to_numeric(house_df_1['Paisa'], errors='coerce')
house_df_1['Daam'] = pd.to_numeric(house_df_1['Daam'], errors='coerce')

def convert_to_sqm(row):
    sqm = 0

    if not pd.isna(row['dhur']):
        sqm += row['dhur'] * 16.93  # 1 Dhur = 16.93 m²
    if not pd.isna(row['Aana']):
        sqm += row['Aana'] * 31.80  # 1 Aana = 31.80 m²
    if not pd.isna(row['Paisa']):
        sqm += row['Paisa'] * 7.95  # 1 Paisa = 7.95 m²
    if not pd.isna(row['Daam']):
        sqm += row['Daam'] * 1.99  # 1 Daam = 1.99 m²

    return sqm

house_df_1['Area_in_SQM'] = house_df_1.apply(convert_to_sqm, axis=1)

print(house_df_1[['dhur', 'Aana', 'Paisa', 'Daam', 'Area_in_SQM']])
```

```
      dhur  Aana  Paisa  Daam  Area_in_SQM
0      0.0   7.0    2.0   0.0       238.50
1      0.0   3.0    0.0   0.0        95.40
2      0.0   2.0    2.0   0.0        79.50
3     12.0   NaN    NaN   NaN       203.16
4      0.0   4.0    2.0   0.0       143.10
...    ...   ...    ...   ...          ...
1619   0.0   3.0    1.0   0.0       103.35
1620   0.0   3.0    2.0   0.0       111.30
1621   0.0   5.0    0.0   0.0       159.00
1622   0.0   5.0    0.0   2.0       162.98
1623   3.0   5.0    NaN   NaN       209.79

[1624 rows x 5 columns]
```

Figure 20: Converting the feature value with metric system

[15]:
```
house_df_1.isnull().sum()
```

[15]:
```
bedroom             0
bathroom            0
floors            676
parking             0
dhur                3
Aana              458
Paisa             643
Daam              664
roadsize(feet)      0
Road type           0
Years ago           0
Area(sq ft)         0
Price               0
Area_in_SQM         0
dtype: int64
```

[16]:
```
house_df_1.drop(columns=['dhur', 'Aana', 'Paisa', 'Daam'], inplace=True)
```

Figure 21: Checking null values and dropping columns

```
dtype: int64
```

```
[19]: sns.histplot(house_df_1['floors'])
```

```
[19]: <Axes: xlabel='floors', ylabel='Count'>
```

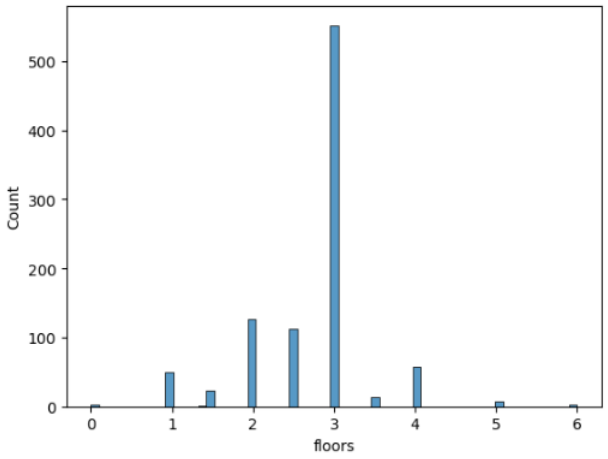


Figure 22: Visualizing the values in floors column

```
[20]: average_floors = house_df_1['floors'].mean()

      house_df_1['floors'].fillna(average_floors, inplace=True)
```

```
[19]: house_df_1.isnull().sum()
```

```
[19]: bedroom          0
      bathroom         0
      floors           0
      parking          0
      roadsize(feet)   0
      Road type        0
      Years ago        0
      Area(sq ft)      0
      Price            0
      Area_in_SQM      0
      dtype: int64
```

Figure 23: Filling missing values with average

Dealing with Categorical Features ¶

[22]:
```
house_df_1.head()
```

[22]:

	bedroom	bathroom	floors	parking	roadsize(feet)	Road type	Years ago	Area(sq ft)	Price	Area_in_SQM
0	0	0	2.750369	0	20.0	Soil Stabilized	2.0	2566.87	36000000	238.50
1	4	3	2.500000	1	13.0	Gravelled	2.0	1026.75	18000000	95.40
2	4	3	3.000000	0	13.0	Gravelled	2.0	855.62	15000000	79.50
3	7	5	2.000000	3	15.0	Blacktopped	1.0	2187.00	57000000	203.16
4	7	3	2.500000	3	13.0	Blacktopped	1.0	1540.12	24500000	143.10

There are two categorical features, **Road type** and **location_category** which needs the proper encoding for conversion into numerical features. **Road type** will be encoded in ordinal order using mapping whereas **location_category** will be encoded using binary encoding.

[23]:
```
road_type_mapping = {
    'Alley': 0,
    'Soil Stabilized': 1,
    'Gravelled': 2,
    'Blacktopped': 3,
    'Paved': 4,
    'Concrete': 5
}

house_df_1['Road type'] = house_df_1['Road type'].map(road_type_mapping)
```

Figure 24: Dealing with categorical features

[24]:
```
house_df_1.head(14)
```

[24]:

	bedroom	bathroom	floors	parking	roadsize(feet)	Road type	Years ago	Area(sq ft)	Price	Area_in_SQM
0	0	0	2.750369	0	20.0	1	2.000000	2566.87	36000000	238.50
1	4	3	2.500000	1	13.0	2	2.000000	1026.75	18000000	95.40
2	4	3	3.000000	0	13.0	2	2.000000	855.62	15000000	79.50
3	7	5	2.000000	3	15.0	3	1.000000	2187.00	57000000	203.16
4	7	3	2.500000	3	13.0	3	1.000000	1540.12	24500000	143.10
5	5	6	4.000000	4	13.0	4	2.000000	2622.50	60000000	243.65
6	0	0	2.750369	0	12.0	2	2.000000	1090.92	11500000	101.37
7	6	3	3.000000	1	10.0	2	0.833333	3533.75	16500000	328.30
8	2	4	3.000000	1	12.0	2	1.000000	1283.43	21500000	119.25
9	0	0	2.750369	0	0.0	1	0.916667	1133.70	22500000	105.34
10	2	1	1.000000	2	10.0	2	2.000000	0.00	13000000	0.00
11	0	0	2.750369	0	0.0	1	0.750000	1240.65	15000000	115.28
12	3	3	3.000000	1	13.0	2	1.000000	1391.25	16000000	129.26
13	0	0	2.750369	0	13.0	1	1.000000	1197.87	19200000	111.30

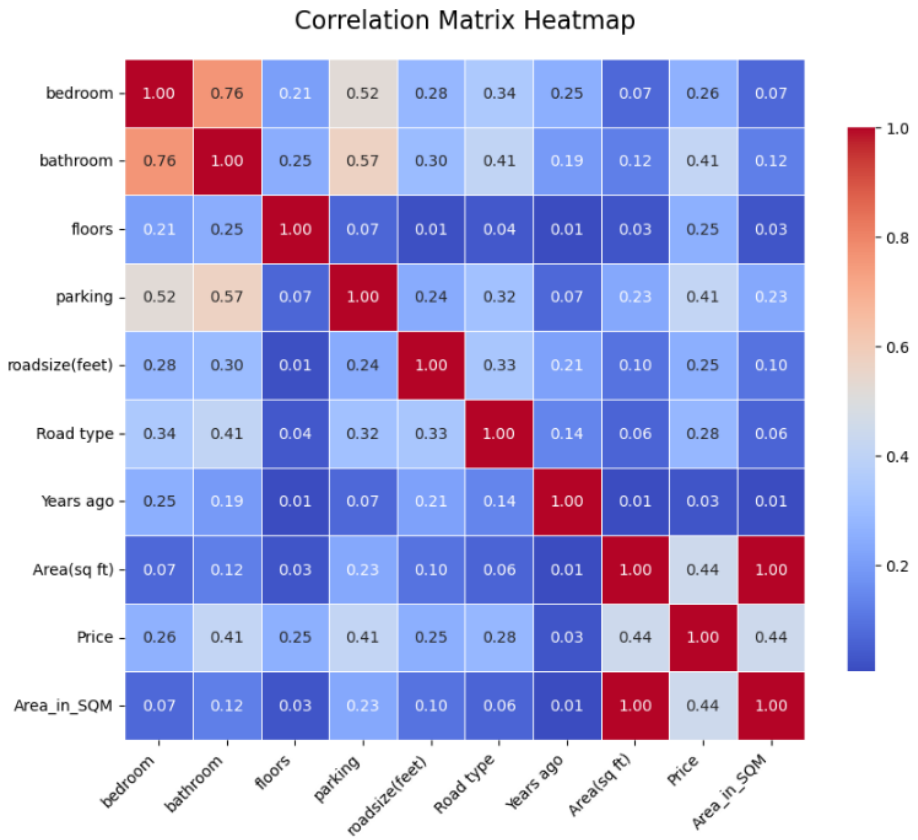
Figure 25: Displaying the dataset

```
[26]: correlation = house_df_1.corr()
      correlation
```

[26]:

	bedroom	bathroom	floors	parking	roadsize(feet)	Road type	Years ago	Area(sq ft)	Price	Area_in_SQM
bedroom	1.000000	0.761232	0.206637	0.521286	0.281097	0.344776	0.253863	0.068877	0.261856	0.068773
bathroom	0.761232	1.000000	0.248829	0.571267	0.295581	0.408704	0.192034	0.123960	0.410562	0.123867
floors	0.206637	0.248829	1.000000	0.065965	0.013649	0.036424	0.005484	0.025287	0.254519	0.025347
parking	0.521286	0.571267	0.065965	1.000000	0.242239	0.315887	0.068506	0.231585	0.409572	0.231472
roadsize(feet)	0.281097	0.295581	0.013649	0.242239	1.000000	0.332946	0.212882	0.097898	0.252386	0.097980
Road type	0.344776	0.408704	0.036424	0.315887	0.332946	1.000000	0.137964	0.063324	0.281182	0.063228
Years ago	0.253863	0.192034	0.005484	0.068506	0.212882	0.137964	1.000000	0.007298	0.027075	0.007276
Area(sq ft)	0.068877	0.123960	0.025287	0.231585	0.097898	0.063324	0.007298	1.000000	0.444660	0.999997
Price	0.261856	0.410562	0.254519	0.409572	0.252386	0.281182	0.027075	0.444660	1.000000	0.444594
Area_in_SQM	0.068773	0.123867	0.025347	0.231472	0.097980	0.063228	0.007276	0.999997	0.444594	1.000000

Figure 26: Correlations for the dataset



From above correlation heatmap, it is concluded that the **location_category_encoded** and the **Years ago** have very low correlation with respect to our target feature price .

Figure 27: Correlation heatmap

```
[28]: house_df_1.drop(['Years ago'], axis=1, inplace=True)
```

```
[29]: house_df_1.head(10)
```

[29]:

	bedroom	bathroom	floors	parking	roadsize(feet)	Road type	Area(sq ft)	Price	Area_in_SQM
0	0	0	2.750369	0	20.0	1	2566.87	36000000	238.50
1	4	3	2.500000	1	13.0	2	1026.75	18000000	95.40
2	4	3	3.000000	0	13.0	2	855.62	15000000	79.50
3	7	5	2.000000	3	15.0	3	2187.00	57000000	203.16
4	7	3	2.500000	3	13.0	3	1540.12	24500000	143.10
5	5	6	4.000000	4	13.0	4	2622.50	60000000	243.65
6	0	0	2.750369	0	12.0	2	1090.92	11500000	101.37
7	6	3	3.000000	1	10.0	2	3533.75	16500000	328.30
8	2	4	3.000000	1	12.0	2	1283.43	21500000	119.25
9	0	0	2.750369	0	0.0	1	1133.70	22500000	105.34

Figure 28: Dropping column with low correlation with target

Dealing with Outliers

Here, we visualizes various columns for detecting the outliers. Columns like Price, Areas are more prone to outliers.

```
[31]: def plot_count_and_box(data, column):
          plt.figure(figsize=(14, 6))

          plt.subplot(1, 2, 1)
          sns.countplot(data=data, x=column)
          plt.title(f'Countplot of {column}')

          plt.subplot(1, 2, 2)
          sns.boxplot(data=data, x=column)
          plt.title(f'Boxplot of {column}')

          plt.tight_layout()
          plt.show()

      plot_count_and_box(house_df_1, 'Price')
```

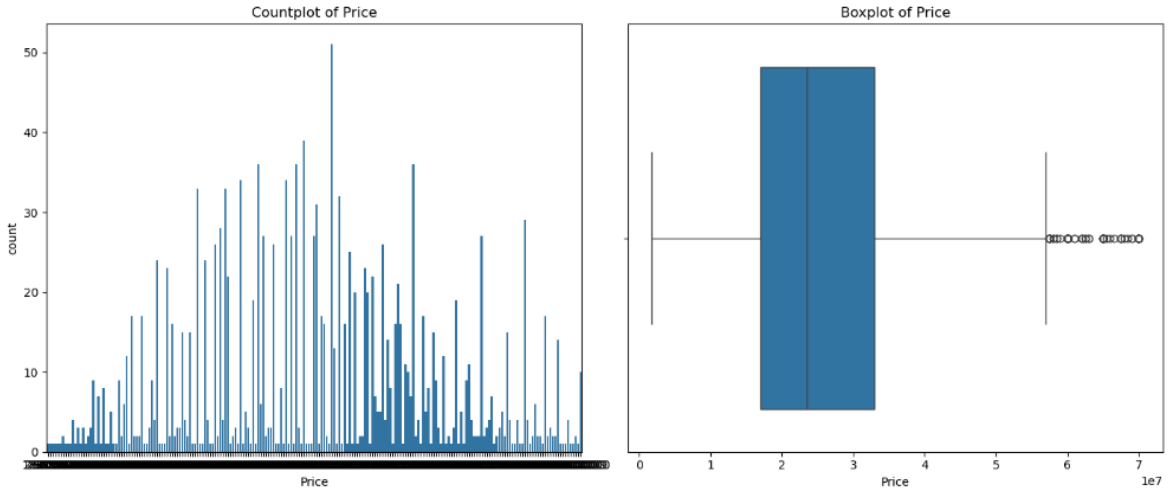


Figure 29: Dealing with outliers

```
[72]: plot_count_and_box(house_df_1, 'Area(sq ft)')
```

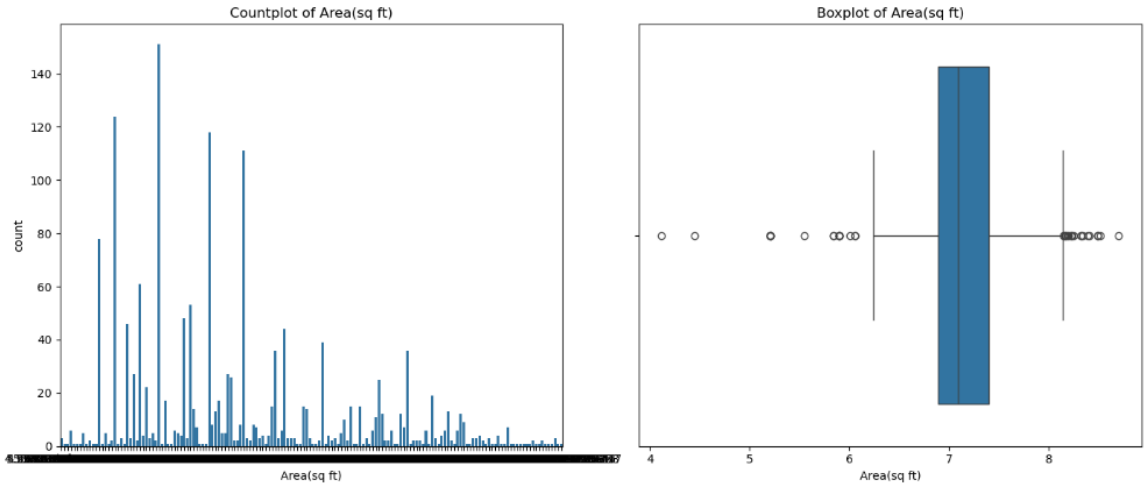


Figure 30: Outliers detection

```
[32]: plot_count_and_box(house_df_1, 'Area_in_SQM')
```

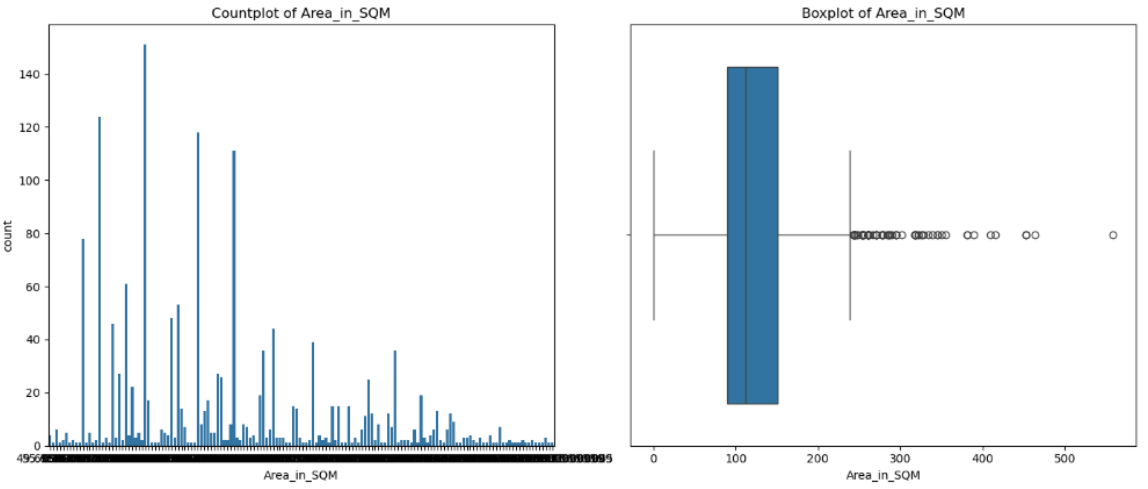


Figure 31: Outliers detection_1

```
[34]: def apply_split(dataset, test_size=0.2, random_state=42):
          X = dataset.drop(columns='Price')
          X['Area(sq ft)'].fillna(X['Area(sq ft)'].median(), inplace=True)
          X['Area_in_SQM'].fillna(X['Area_in_SQM'].median(), inplace=True)

          y = dataset['Price']

          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)
          return X_train, X_test, y_train, y_test

[35]: X_train, X_test, y_train, y_test = apply_split(house_df_1)
      X_train.isnull().sum()

[35]: bedroom          0
      bathroom         0
      floors           0
      parking          0
      roadsize(feet)   0
      Road type        0
      Area(sq ft)      0
      Area_in_SQM      0
      dtype: int64
```

Figure 32: Splitting the dataset

```
dtype: int64

[36]: X_train.head(30)
```

[36]:

	bedroom	bathroom	floors	parking	roadsize(feet)	Road type	Area(sq ft)	Area_in_SQM
429	5	3	2.750369	3	12.0	3	2053.50	190.80
1090	0	0	2.750369	0	13.0	1	1369.00	127.20
1333	9	0	3.000000	3	13.0	1	889.00	82.59
1402	5	4	2.500000	1	12.0	2	1026.75	95.40
650	5	4	2.750369	2	15.0	2	1967.93	182.85
898	5	4	4.000000	7	13.0	2	1197.87	111.30
978	5	6	3.000000	3	22.0	3	1458.00	135.44
1389	3	4	3.000000	1	20.0	3	1347.60	125.22
1535	5	5	3.000000	1	13.0	2	729.00	67.72
721	6	4	2.500000	2	13.0	2	1197.87	111.30
664	0	0	3.000000	0	13.0	1	2622.50	243.65
1562	0	0	2.000000	0	13.0	3	729.00	67.72
1210	0	0	2.750369	0	0.0	1	1283.43	119.25
701	6	3	3.000000	1	10.0	3	1625.68	151.05
855	0	0	2.750369	0	11.0	2	1369.00	127.20
722	3	4	3.000000	2	10.0	1	546.75	50.79
535	0	0	2.750369	0	13.0	1	1112.30	103.35
1517	4	0	3.000000	0	13.0	2	2440.25	226.72
678	0	0	2.750369	0	13.0	3	546.75	50.79
1170	6	6	3.000000	1	13.0	2	729.00	67.72
277	6	3	3.000000	3	12.0	1	729.00	67.72
1350	3	4	3.000000	1	0.0	1	1026.75	95.40
994	5	7	4.000000	4	15.0	4	911.25	84.65
812	8	6	4.000000	1	12.0	3	2481.31	230.55
236	0	0	2.750369	0	12.0	2	1454.55	135.15
1580	0	0	4.000000	0	13.0	1	546.75	50.79

Figure 33: X_train before scaling

```
[*]: import numpy as np

def mean_normalization_inplace(X_train, X_test):
    mean = np.mean(X_train, axis=0)

    X_train -= mean
    X_test -= mean


mean_normalization_inplace(X_train, X_test)
```

```
[38]: X_train
```

[38]:

	bedroom	bathroom	floors	parking	roadsize(feet)	Road type	Area(sq ft)	Area_in_SQM
429	1.916089	0.792148	0.003605	2.003079	-0.263765	1.047729	686.908858	63.835473
1090	-3.083911	-2.207852	0.003605	-0.996921	0.736235	-0.952271	2.408858	0.235473
1333	5.916089	-2.207852	0.253236	2.003079	0.736235	-0.952271	-477.591142	-44.374527
1402	1.916089	1.792148	-0.246764	0.003079	-0.263765	0.047729	-339.841142	-31.564527
650	1.916089	1.792148	0.003605	1.003079	2.736235	0.047729	601.338858	55.885473
...	...	...	...	...	...	...	...	...
1130	-3.083911	-2.207852	0.003605	-0.996921	2.736235	1.047729	455.908858	42.335473
1294	0.916089	1.792148	0.253236	0.003079	0.736235	0.047729	-339.841142	-31.564527
860	0.916089	0.792148	0.253236	-0.996921	-0.263765	-0.952271	3629.408858	337.225473
1459	-1.083911	-1.207852	-1.746764	0.003079	-12.263765	0.047729	-1184.341142	-110.034527
1126	2.916089	0.792148	-0.246764	4.003079	7.736235	1.047729	-168.721142	-15.664527

1299 rows × 8 columns

Figure 34: X_train after scaling

Training machine learning models

```
[41]: def train_linear_model(X_train_norm, X_test_norm, y_train, y_test):
    linear_model = LinearRegression()
    linear_model.fit(X_train_norm, y_train)
    y_pred_linear = linear_model.predict(X_test_norm)

    mae = mean_absolute_error(y_test, y_pred_linear)
    mse = mean_squared_error(y_test, y_pred_linear)
    r2 = r2_score(y_test, y_pred_linear)
    accuracy = linear_model.score(X_test_norm, y_test)

    print(f'Accuracy: {accuracy}')
    print(f'MAE: {mae}')
    print(f'MSE: {mse}')
    print(f'R²: {r2}')
```

```
[42]: train_linear_model(X_train,X_test,y_train,y_test)
```

```
Accuracy: 0.4369726314206436
MAE: 7475356.996535046
MSE: 102358431812415.8
R²: 0.4369726314206436
```

Figure 35: Training and metrics of linear model

```
[43]: def train_xgboost_regressor(X_train, X_test, y_train, y_test, parameters):
          xgboost_model = xgb.XGBRegressor(**parameters)
          xgboost_model.fit(X_train, y_train)
          y_pred_xgboost = xgboost_model.predict(X_test)

          mae = mean_absolute_error(y_test, y_pred_xgboost)
          mse = mean_squared_error(y_test,y_pred_xgboost)
          r2 = r2_score(y_test, y_pred_xgboost)

          print(f"R-2 score is: {r2}")

[46]: parameters = {
          'n_estimators': 100,
          'learning_rate': 0.1,
          'max_depth': 6,
          'subsample': 0.8
      }

[47]: train_xgboost_regressor(X_train, X_test, y_train, y_test, parameters)
      R-2 score is: 0.5552560687065125
```

Figure 36: Training xgboost and its metrics

```
[49]: def tune_xgboost_model(X_train, y_train):
          param_grid = {
              'n_estimators': [1200, 1000,900],
              'learning_rate': [0.002,0.003, 0.005, 0.1],
              'max_depth': [3, 6, 9],
              'min_child_weight': [1, 3, 5],
              'subsample': [0.7, 0.8, 0.9],
              'colsample_bytree': [0.7, 0.8, 0.9],
              'gamma': [0, 0.1, 0.2],  # Regularization parameter for controlling complexity
              'reg_alpha': [0, 0.1, 0.5],  # L1 regularization term
              'reg_lambda': [1, 2.5, 2]   # L2 regularization term
          }
          tuned_model = xgb.XGBRegressor()
          grid_search = GridSearchCV(estimator=tuned_model, param_grid=param_grid, cv=5, n_jobs=-1, scoring='r2')

          # Fit the grid search to the training data
          grid_search.fit(X_train, y_train)

          # Get the best model and parameters
          best_model = grid_search.best_estimator_
          best_params = grid_search.best_params_

          print(f'Best Hyperparameters: {best_params}')
          print(f'Best Model: {best_model}')

          return best_model, best_params

[*]: best_model, best_params = tune_xgboost_model(X_train, y_train)

[ ]: print(f'Best Parameters are:{best_params}')

[ ]: y_pred_tuned_model = best_model.predict(X_test)
     r2 = r2_score(y_test, y_pred_tuned_model)
     print(f'r2 score is: {r2}')
```

Figure 37: Finetuning hyperparameters with GridSearchCV

5. Conclusion

To sum up, the problem of house price prediction using two different machine learning algorithms provided numerous opportunities and concept to learn from. The application of exploratory data analysis, feature engineering, feature selection, scaling, training a model, fine tuning the hyperparameters are well implemented in this prediction problems for solving the issue of house and property price in Kathmandu. Machine learning algorithms find hard to learn from the complex relationship between the features for correct prediction due to which the prediction metrics of the model can seem overwhelming but these shortcomings can be improved with the help of deep learning models and neural network architecture in future works. Deep learning will quickly adapt to the data for finding the appropriate loss function for the prediction problem unlike the traditional machine learning algorithm like Linear Regression and ensemble technique like gradient boosting and XGBoost. The dataset contained irrelevant columns along with some columns that does contain the meaningful data but in very raw format. The feature like **Address** which contained numerous categorical values got dropped as this comes out of the scope for this problem statement. Those string values in the features aren't well fed with the machine learning models as well. Identification of null values, dealing with those null values, selecting features based on the correlation heatmap, detecting outliers and appropriately dealing with them without dropping them, creation of new important feature from existing feature, visualization through various plots, splitting the dataset, creating and training the machine learning models with the dataset and evaluating their performance metrics for fine-tuning procedure is done through this house price prediction system as part of the coursework for the module Applied Machine Learning (CC6057NI).

Linear Regression and XGBoost Regression were implemented for the proposed system of house price prediction along with the comparative study of these two different algorithms for a same regression task of predicting the house/property prices from Kathmandu. Linear regression was not able to correctly adapt to the dataset due to it simple nature and the dataset's complex nature. The r2 score for the linear regression model trained on the training set of data and evaluated in testing set of data is 0.4369, whereas the mean absolute error and mean square are 7475356.996535046, 102358431812415.8 respectively. On the other hand, the XGBoost regressor model without any hyperparameter have produced the r2 score of 0.5552560687065125 which is not great for the regression problem but this was further tuned by tuning the hyperparameters of XGBoost regressor using GridSearchCV. The hyperparameters included were *n_estimators, learning_rate, max_depth, min_child_weight, subsample, colsample_bytree, gamma

(regularization parameter),reg_alpha and reg_lambda*. These parameters were tuned and the training period for the model was around 3000 seconds.

6. References

Bhandari, A., Oct, 2024. *A Beginner's Guide to matplotlib for Data Visualization and Exploration in Python.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2020/02/beginner-guide-matplotlib-data-visualization-exploration-python/
[Accessed 20 Dec 2024].

Bourke, D., Nov, 2023. *NumPy 101: Intro To Numerical Data Manipulation With NumPy + Python.* [Online]
Available at: https://zerotomastery.io/blog/numpy-101-tutorial/
[Accessed 20 Dec 2024].

Fluhler, M., Feb, 2024. *Feature Engineering: The Difference Maker for ML Models.* [Online]
Available at: https://blog.dataiku.com/what-is-feature-engineering
[Accessed 22 Dec 2024].

Hachcham, A., Sep, 2024. *XGBoost: Everything You Need to Know.* [Online]
Available at: https://neptune.ai/blog/xgboost-everything-you-need-to-know
[Accessed 19 Dec 2024].

Jain, K., Oct, 2024. *Scikit-learn(sklearn) in Python – the most important Machine Learning tool I learnt last year!.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2015/01/scikit-learn-python-machine-learning-tool/
[Accessed 20 Dec 2024].

Jain, P., Sep, 2024. *What Is Data Collection: Methods, Types, Tools.* [Online]
Available at: https://www.simplilearn.com/what-is-data-collection-article
[Accessed 21 Dec 2024].

Lewinson, E., Feb 17, 2022. *Three Approaches to Encoding Time Information as Features for ML Models.* [Online]
Available at: https://developer.nvidia.com/blog/three-approaches-to-encoding-time-information-as-features-for-ml-models/
[Accessed 19 Dec 2024].

Mahat, M., April, 2022. *Kathmandu-House-Price-Prediction.* [Online]
Available at: https://gitlab.com/LordMilan/Kathmandu-House-Price-Prediction
[Accessed 21 Dec 2024].

McLntire, G., 2024. *Python Pandas Tutorial: A Complete Introduction for Beginners.* [Online]
Available at: https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/
[Accessed 20 Dec 2024].

Novogroder, I., April,2024. *Data Preprocessing in Machine Learning: Steps & Best Practices.* [Online]
Available at: https://lakefs.io/blog/data-preprocessing-in-machine-learning/
[Accessed 21 Dec 2024].

WLLber, J., Sep, 2022. *Linear Regression.* [Online]
Available at: https://mlu-explain.github.io/linear-regression/
[Accessed 19 Dec 2024].