

DATA INTENSIVE COMPUTING

PROJECT FINAL PHASE

House Price Forecasting with ML
Algorithms

Jonnalagadda Satvik (50560441)

Komatineni Jaswanth (50560594)

Jaladi Hima Venkata Sai Saketh Ram (50560439)

List of Contents:

1. Problem Statement
2. Overview
3. Data Acquisition
4. Data Cleaning
5. Exploratory Data Analysis
6. Algorithms & Visualization
7. Visualization Across Models
8. Why this model?
9. Steps to integrate the model into our product
10. Folder Structure
11. Instructions to deploy the product
12. Conclusion
13. Recommendations and Future Work
14. References

Problem Statement

The project's goal is to create machine learning algorithms that can accurately predict home prices. In this project, we mainly focus to prevent the problem of predicting house values, which is essential for making wise decisions in property purchases, by using historical housing data. The main contribution of this project will be advantage to promote investor risk management, increase market transparency.

Background

Predicting home prices is very challenging in real estate and common people. Real estate agents, investors, sellers, and purchasers may take good decisions when home values are accurately forecasted. However, because real estate markets are complex and impacted by different factors, including location, economic conditions, and property features, forecasting house prices is difficult.

Importance of Contribution

This project is important because of its contributions in many ways. First, by providing accurate price predictions, it can improve real estate markets' efficiency and transparency. This ensures fair transactions, which can be advantageous to both buyers and sellers. Secondly, accurate price forecasting can help investors in reducing risk. Taking care of all these points into consideration, this project has the potential to greatly enhance

real estate industry decision-making, resulting in more efficient and stable housing markets.

Finally, this project aims to leverage machine learning algorithms to predict house prices accurately. By analyzing historical housing data, the project seeks to develop models that can forecast future house prices with high precision.

Overview

Phase 1:

In this phase, we have acquired the data set, cleaning it by taking care of missing data and outliers, and lead EDA to acquire relations into the information and connections between features. In this part, we have analyzed the data set and perform all operations to clean the data and visualize the part to know the relationships between features and go through how price distribution has been taken place.

Phase 2:

In this phase, we have used different machine learning models that could be applicable to our project i.e., HOUSE PRICE PREDICTION which is regression task. They are: Linear Regression, K Nearest Neighbors, Support Vector Regressor, XG Boost, Decision Tree Regressor, Random Forest Regressor and Gradient Boost Regression and visualization.

Phase 3:

In the final phase, we translated our findings into a tangible data product – an interface capable of predicting house prices in real-time. Users can input for different features and our system predicts the house price. This interface integrates insights from both data preprocessing and model training phases, providing a seamless and intuitive tool for real estate developers and end-users alike.

Data Acquisition

Data acquisition is the process of collecting raw data from various sources and converting it into a format that can be used for analysis.

In this context of House Price Prediction, data acquisition involves obtaining the dataset from the Kaggle repository and loading it into a jupyter notebook for further analysis.

Here we will perform many operations like importing necessary libraries, importing dataset using pandas.

Quick review of the dataset:

- Determining the shape of the dataset
- Information of the dataset
- Verifying if any null values exist
- Display the dataset

Here is the data set link:

<https://www.kaggle.com/datasets/shree1992/housedata>

- **Importing Necessary Libraries using Pandas:**

Importing Libraries

```
In [221]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

- To perform different operations on the dataset, we need to first import necessary libraries in the working environment i.e., Jupyter Notebook.
- Here we have imported some libraries like pandas which will be used for data manipulation and analysis, NumPy which is used to perform mathematical calculations, matplotlib which is used for visualization etc.

- **Importing dataset:**

Data Acquisition

```
In [222]: hp = pd.read_csv("housedata.csv")  
          print(len(hp))  
          print(hp.dtypes)
```

```
4619  
date           object  
bedrooms       float64  
bathrooms      float64  
sqft_living     int64  
sqft_lot        int64  
floors          object  
waterfront     float64  
view           float64  
condition      float64  
sqft_above     int64  
sqft_basement  int64  
yr_built       int64  
yr_renovated   int64  
street         object  
cityzip        object  
state          object  
country        object  
price          float64  
dtype: object
```

- In order to analyze a dataset, firstly we need to load the dataset into the working environment.
- Since our dataset is in .csv format. We have used function called read_csv () to load the dataset.

- **Dimension of the dataset**

```
In [223]: hp.shape
```

```
Out[223]: (4619, 18)
```

- Knowing the dataset is the primary part of data acquisition.
- To extract the number of rows and features present in the dataset, we have used function called shape ().

- **Description of the raw dataset.**

In [224]: `hp.describe()`

Out[224]:

	bedrooms	bathrooms	sqft_living	sqft_lot	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	price
count	4,614.0	4,612.0	4,619.0	4,619.0	4,611.0	4,610.0	4,610.0	4,619.0	4,619.0	4,619.0	4,619.0	4,619.0
mean	3.4	2.2	2,138.4	14,840.5	0.0	0.2	3.5	1,826.3	312.0	1,970.8	809.6	551,745.1
std	0.9	0.8	962.4	35,821.0	0.1	0.8	0.7	861.6	463.8	29.7	979.6	562,907.8
min	0.0	0.0	370.0	638.0	0.0	0.0	1.0	370.0	0.0	1,900.0	0.0	0.0
25%	3.0	1.8	1,460.0	5,000.0	0.0	0.0	3.0	1,190.0	0.0	1,951.0	0.0	322,500.0
50%	3.0	2.2	1,970.0	7,683.0	0.0	0.0	3.0	1,590.0	0.0	1,976.0	0.0	460,886.9
75%	4.0	2.5	2,620.0	11,002.5	0.0	0.0	4.0	2,300.0	610.0	1,997.0	1,999.0	654,475.0
max	9.0	8.0	13,540.0	1,074,218.0	1.0	4.0	5.0	9,410.0	4,820.0	2,014.0	2,014.0	26,590,000.0

- To know more about our data, we have used a function called `describe ()` which provides the information like count, mean, standard deviation etc.
- This will only work on numerical features present in the dataset.

- **Information of the dataset.**

In [225]: `hp.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4619 entries, 0 to 4618
Data columns (total 18 columns):
#   Column          Non-Null Count  Dtype
---  -
0   date            4619 non-null   object
1   bedrooms        4614 non-null   float64
2   bathrooms       4612 non-null   float64
3   sqft_living     4619 non-null   int64
4   sqft_lot        4619 non-null   int64
5   floors          4611 non-null   object
6   waterfront      4611 non-null   float64
7   view            4610 non-null   float64
8   condition       4610 non-null   float64
9   sqft_above      4619 non-null   int64
10  sqft_basement   4619 non-null   int64
11  yr_built        4619 non-null   int64
12  yr_renovated    4619 non-null   int64
13  street         4619 non-null   object
14  cityzip        4619 non-null   object
15  state          4619 non-null   object
16  country        4619 non-null   object
17  price          4619 non-null   float64
dtypes: float64(6), int64(6), object(6)
memory usage: 649.7+ KB
```

- `info ()` function is used to extract details like data type and to list all the features present in the dataset.
- This function gives you a clear information about the dataset.

- **Finding null values of the dataset.**

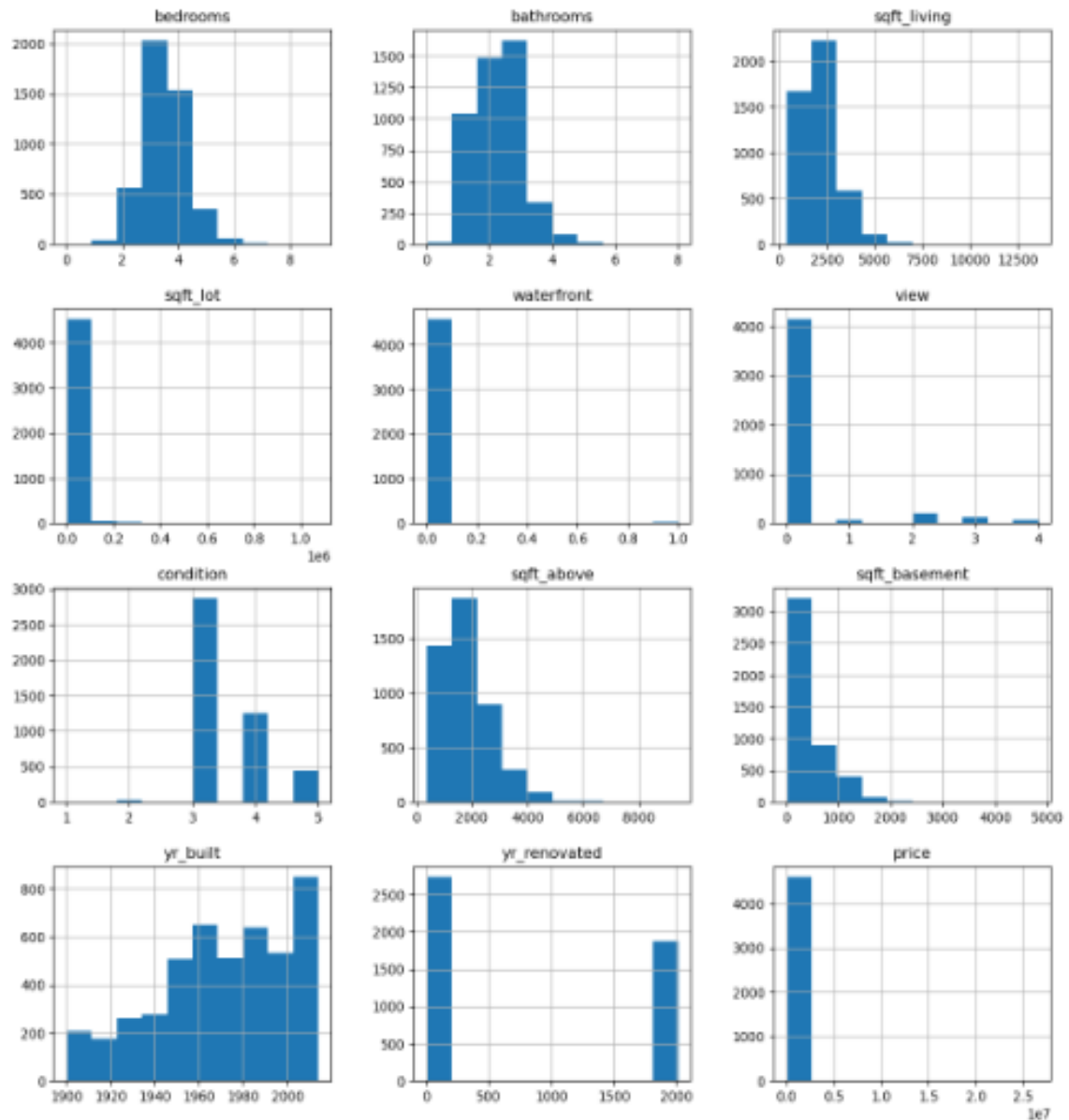
```
In [226]: hp.isnull().sum()
```

```
Out[226]: date          0  
bedrooms      5  
bathrooms     7  
sqft_living   0  
sqft_lot      0  
floors        8  
waterfront    8  
view          9  
condition     9  
sqft_above    0  
sqft_basement 0  
yr_built      0  
yr_renovated  0  
street        0  
cityzip       0  
state         0  
country       0  
price         0  
dtype: int64
```

- A function `isnull.sum ()` is used to count the number of null values present in each column or feature of the dataset.
- If we get to know that there are some null values present in the dataset, we can take care of them in Data Cleaning process.
- grid of histograms, with each histogram showing the distribution of values for a particular numerical feature in the Data Frame.
- Now, let us plot the histogram which gives a graphical representation of how features of our dataset are varying.
- This is histogram representation before Data Cleaning.

```
In [228]: hp.hist(figsize=(14,14))
```

```
Out[228]: array([[<Axes: title=[ 'center': 'bedrooms'>,<Axes: title=[ 'center': 'bathrooms'>,<Axes: title=[ 'center': 'sqft_living'>],<Axes: title=[ 'center': 'sqft_lot'>,<Axes: title=[ 'center': 'waterfront'>,<Axes: title=[ 'center': 'view'>],<Axes: title=[ 'center': 'condition'>,<Axes: title=[ 'center': 'sqft_above'>,<Axes: title=[ 'center': 'sqft_basement'>],<Axes: title=[ 'center': 'yr_built'>,<Axes: title=[ 'center': 'yr_renovated'>,<Axes: title=[ 'center': 'price'>]]], dtype=object)
```



Data Cleaning/Preparation

Data Cleaning

Data cleaning is the process of identifying and correcting errors, inconsistencies, and missing values in a dataset to improve its reliability for analysis. It involves several steps to ensure that the data is accurate, complete, and consistent.

In our context, we have observed a lot of inconsistency in data. There are many duplicate values, data type errors, null values and outliers present in the dataset.

We perform a lot of operations to get rid of such invalid data and added some new features which could be helpful for better accuracy of our project.

- **Formatting Columns:**

Formatting Columns

```
In [229]: hp.dtypes
```

```
Out[229]: date           object
bedrooms      float64
bathrooms     float64
sqft_living   int64
sqft_lot      int64
floors        object
waterfront    float64
view          float64
condition     float64
sqft_above    int64
sqft_basement int64
yr_built      int64
yr_renovated  int64
street        object
cityzip       object
state         object
country       object
price         float64
dtype: object
```

```
In [230]: #format columns
# Strip extra whitespace from strings
# Could also go further and convert everything to lowercase, collapse extra internal spaces, etc...
string_columns = hp.select_dtypes(["object"])
hp[string_columns.columns] = string_columns.apply(lambda x: x.str.strip())

hp['bathrooms'] = hp['bathrooms'].round()
```

- Formatting Column refers to process of changing the display type of a feature or an object.
- Here, in this context, we have used this formatting to clear out white spaces, hyphen present in the categorical features of the dataset

- **Renaming Columns**

Renaming Columns

```
In [231]: hp.columns = [col.upper() for col in hp.columns]
hp.columns = ["DATE", "NUM_OF_BEDROOMS", "NUM_OF_BATHROOMS", "LIVING_AREA_sqft", "LOT_AREA_sqft", "FLOORS", "WATER FRONT", "VIEW_RATING"]
```

- Renaming Column is process of changing the name of a feature or a column in a dataset.
- Here, in this context, we have performed this because to make feature more meaningful and descriptive.

- **Typecasting**

Typecasting

```
In [232]: hp.dtypes
```

```
Out[232]: DATE                object
NUM_OF_BEDROOMS             float64
NUM_OF_BATHROOMS            float64
LIVING_AREA_sqft            int64
LOT_AREA_sqft               int64
FLOORS                      object
WATER_FRONT                 float64
VIEW_RATING                 float64
CONDITION_RATING            float64
ABOVE_BASEMENT_sqft         int64
BASEMENT_sqft               int64
YEAR_BUILT                  int64
YEAR_RENOVATED              int64
STREET                     object
CITYZIP                    object
STATE                     object
COUNTRY                   object
PRICE                     float64
dtype: object
```

```
In [233]: pd.options.display.float_format = '{:,.1f}'.format
hp.PRICE = hp.PRICE.astype(float)

# Turn date string into an actual date object
hp["DATE"] = pd.to_datetime(hp["DATE"])
```

```
In [234]: hp.dtypes
```

```
Out[234]: DATE                datetime64[ns]
NUM_OF_BEDROOMS             float64
NUM_OF_BATHROOMS            float64
LIVING_AREA_sqft            int64
LOT_AREA_sqft               int64
FLOORS                      object
WATER_FRONT                 float64
VIEW_RATING                 float64
CONDITION_RATING            float64
ABOVE_BASEMENT_sqft         int64
BASEMENT_sqft               int64
YEAR_BUILT                  int64
YEAR_RENOVATED              int64
STREET                     object
CITYZIP                    object
STATE                     object
COUNTRY                   object
PRICE                     float64
dtype: object
```

- Typecasting is the process of converting a feature from one data type to another.
- Here, we have performed type casting for 'PRICE' and 'DATE' because they are initially assigned as 'OBJECT' types which is incompatible type for performing any operations on those features.

- **Dictionary Mapping**

Dictionary Mapping

```
In [235]: hp["FLOORS"].value_counts()
```

```
Out[235]: FLOORS
1         2169
2         1816
1.5       447
3         126
2.5        41
one         4
two         3
three       3
3.5         2
Name: count, dtype: int64
```

```
In [236]: word_to_num = {'one': 1, 'two': 2, 'three': 3}
hp['FLOORS'] = hp['FLOORS'].replace(word_to_num)

hp['FLOORS'] = hp['FLOORS'].astype(float)
hp.dtypes
```

```
Out[236]: DATE                datetime64[ns]
NUM_OF_BEDROOMS              float64
NUM_OF_BATHROOMS             float64
LIVING_AREA_sqft             int64
LOT_AREA_sqft                int64
FLOORS                       float64
WATER_FRONT                  float64
VIEW_RATING                   float64
CONDITION_RATING              float64
ABOVE_BASEMENT_sqft          int64
BASEMENT_sqft                 int64
YEAR_BUILT                    int64
YEAR_RENOVATED                int64
STREET                        object
CITYZIP                       object
STATE                        object
COUNTRY                       object
PRICE                         float64
dtype: object
```

```
In [237]: hp["FLOORS"].value_counts()
```

```
Out[237]: FLOORS
1.0       2173
2.0       1819
1.5       447
3.0       129
2.5        41
3.5         2
Name: count, dtype: int64
```

- Dictionary mapping is the process of assigning values using keys
- In this context, 'FLOORS' feature has unique values {one, two, three} which are categorical value but 'FLOORS' feature is a numerical feature.
- So, Here using Dictionary Mapping, I have assigned '1' for one, '2' for two and '3' for three.

- **Deleting Duplicate Rows**

Deleting Duplicate Rows

In [238]: `hp.shape`

Out[238]: (4619, 18)

In [239]: `len(hp)`

Out[239]: 4619

In [240]: `duplicate_rows = hp[hp.duplicated()]
hp.drop_duplicates(inplace=True)
duplicate_rows`

Out[240]:

	DATE	NUM OF BEDROOMS	NUM OF BATHROOMS	LIVING AREA sqft	LOT AREA sqft	FLOORS	WATER FRONT	VIEW	HEATING	CONDITION	HEATING ABOVE
4601	2014-08-04	3.0	2.0	2430	5500	2.0	0.0	0.0	0.0	3.0	
4602	2014-05-13	3.0	2.0	1240	7735	1.0	0.0	0.0	0.0	4.0	
4603	2014-06-30	4.0	2.0	2120	3380	2.0	0.0	0.0	0.0	3.0	
4604	2014-06-23	4.0	2.0	1700	20283	1.5	0.0	0.0	0.0	3.0	
4605	2014-05-23	2.0	1.0	1880	8500	1.0	0.0	0.0	0.0	4.0	
4606	2014-05-19	3.0	1.0	1080	21043	1.0	0.0	0.0	0.0	3.0	
4607	2014-06-06	4.0	2.0	1900	8800	1.0	0.0	0.0	0.0	4.0	
4608	2014-05-20	4.0	2.0	1940	8909	1.0	0.0	0.0	0.0	4.0	
4609	2014-06-03	3.0	2.0	1880	57083	2.0	0.0	0.0	0.0	4.0	
4610	2014-07-07	3.0	2.0	1720	3012	2.0	0.0	0.0	0.0	3.0	
4611	2014-05-29	3.0	2.0	1380	1018	3.0	0.0	0.0	0.0	3.0	
4612	2014-06-13	5.0	3.0	3320	23780	2.0	0.0	0.0	0.0	4.0	
4613	2014-07-02	4.0	2.0	2430	4748	1.5	0.0	0.0	0.0	3.0	
4614	2014-07-07	4.0	3.0	3990	9788	2.0	0.0	0.0	0.0	3.0	
4615	2014-07-01	3.0	1.0	1150	8000	1.5	0.0	0.0	0.0	3.0	
4616	2014-05-19	3.0	2.0	1690	1813	2.0	0.0	0.0	0.0	3.0	
4617	2014-07-08	2.0	2.0	1090	2888	2.0	0.0	0.0	0.0	3.0	
4618	2014-06-25	6.0	2.0	1900	8240	1.0	0.0	0.0	0.0	2.0	

In [241]: `hp.shape`

Out[241]: (4601, 18)

In [242]: `len(hp)`

Out[242]: 4601

- In this dataset, we have some duplicate rows so using the function `drop_duplicates ()`, we have removed those duplicate rows.

- Presence of duplicate rows will directly lead to inaccuracy of the data. Hence, we have removed these duplicates from the dataset.
- If you observe, you can find there are a total of 18 duplicate rows present in the dataset.
- The number of rows present in the dataset has been reduced from 4619 to 4601.

- Column Splitting

Column Splitting

In [243]: hp.dtypes

```
Out[243]: DATE                datetime64[ns]
NUM_OF_BEDROOMS              float64
NUM_OF_BATHROOMS             float64
LIVING_AREA_sqft             int64
LOT_AREA_sqft                 int64
FLOORS                        float64
WATER_FRONT                  float64
VIEW_RATING                   float64
CONDITION_RATING             float64
ABOVE_BASEMENT_sqft          int64
BASEMENT_sqft                 int64
YEAR_BUILT                    int64
YEAR_RENOVATED                int64
STREET                        object
CITYZIP                       object
STATE                         object
COUNTRY                       object
PRICE                         float64
dtype: object
```

In [244]: print(hp.iloc[:, -4])

```
# print(hp['CITYZIP'])

0      Shoreline 98133
1      Seattle 98119
2      Kent 98042
3      Bellevue 98008
4      Redmond 98052
...
4596   Kirkland 98033
4597   Seattle 98126
4598   Seattle 98115
4599   Sammamish 98075
4600   Federalway 98003
Name: CITYZIP, Length: 4601, dtype: object
```

```
In [245]: # Split 'cityzip' into 'city' and 'zip' columns
hp[['CITY', 'ZIP']] = hp.iloc[:, -4].str.split(' ', expand=True)

# Drop the original 'cityzip' column if needed
# hp.drop('cityzip', axis=1, inplace=True)

# Display the DataFrame with the new columns
print(hp.head())
```

```
      DATE  NUM_OF_BEDROOMS  NUM_OF_BATHROOMS  LIVING_AREA_sqft  \
0  2014-05-02              3.0                2.0             1340
1  2014-05-02              5.0                2.0             3650
2  2014-05-02              3.0                2.0             1930
3  2014-05-02              3.0                2.0             2000
4  2014-05-02              4.0                2.0             1940

      LOT_AREA_sqft  FLOORS  WATER_FRONT  VIEW_RATING  CONDITION_RATING  \
0              7912     1.5           0.0           0.0              3.0
1              9050     2.0           0.0           4.0              5.0
2             11947     1.0           0.0           0.0              4.0
3              8030     1.0           0.0           0.0              4.0
4             10500     1.0           0.0           0.0              4.0

      ABOVE_BASEMENT_sqft  BASEMENT_sqft  YEAR_BUILT  YEAR_RENOVATED  \
0              1340              0      1955      2005
1              3370              280      1921           0
2              1930              0      1966           0
3              1000             1000      1963           0
4              1140              800      1976      1992

      STREET      CITYZIP  STATE  COUNTRY  PRICE  \
0  18810 Densmore Ave N  Shoreline 98133    WA    USA    313,000.0
1    709 W Blaine St    Seattle 98119    WA    USA  2,384,000.0
2  26206-26214 143rd Ave SE    Kent 98042    WA    USA    342,000.0
3    857 170th Pl NE    Bellevue 98008    WA    USA    420,000.0
4    9105 170th Ave NE    Redmond 98052    WA    USA    550,000.0

      CITY  ZIP
0  Shoreline  98133
1    Seattle  98119
2      Kent  98042
3  Bellevue  98008
4    Redmond  98052
```

- From the Data Acquisition, you can find that this dataset has a feature called 'CITYZIP' which consists of city name and zip code combined which is inappropriate format for analysis.
- Hence, we are splitting those two columns into 'CITY' and 'ZIP' separately using some split () function.

- **Filling Null Values**

- From Data Acquisition, we have already mentioned that there is presence of null values in this dataset.
- This could be a most important data cleaning step because missing values will have direct impact on the prediction accuracy.
- we are going to fill these null values present in the dataset with the 'mode' of the feature respectively using fillna () function.

Filling Null Values

```
In [246]: hp.isnull().sum()
```

```
Out[246]: DATE                0
NUM_OF_BEDROOMS             5
NUM_OF_BATHROOMS            7
LIVING_AREA_sqft            0
LOT_AREA_sqft               0
FLOORS                      8
WATER_FRONT                 8
VIEW_RATING                 9
CONDITION_RATING            9
ABOVE_BASEMENT_sqft         0
BASEMENT_sqft               0
YEAR_BUILT                  0
YEAR_RENOVATED              0
STREET                      0
CITYZIP                     0
STATE                      0
COUNTRY                     0
PRICE                      0
CITY                       0
ZIP                        0
dtype: int64
```

```
In [247]: # Fill null values in 'NUM_OF_BEDROOMS' with the mode
mode_NUM_OF_BEDROOMS = hp["NUM_OF_BEDROOMS"].mode()[0] # Calculates the mode
hp["NUM_OF_BEDROOMS"] = hp["NUM_OF_BEDROOMS"].fillna(mode_NUM_OF_BEDROOMS)

# Fill null values in 'NUM_OF_BATHROOMS' with the mode
mode_NUM_OF_BATHROOMS = hp["NUM_OF_BATHROOMS"].mode()[0] # Calculate the mode
hp["NUM_OF_BATHROOMS"] = hp["NUM_OF_BATHROOMS"].fillna(mode_NUM_OF_BATHROOMS)

# Fill null values in 'NUM_OF_BEDROOMS' with the mode
mode_FLOORS = hp["FLOORS"].mode()[0] # Calculates the mode
hp["FLOORS"] = hp["FLOORS"].fillna(mode_FLOORS)

# Fill null values in 'NUM_OF_BEDROOMS' with the mode
mode_WATERFRONT = hp["WATER_FRONT"].mode()[0] # Calculates the mode
hp["WATER_FRONT"] = hp["WATER_FRONT"].fillna(mode_WATERFRONT)

# Fill null values in 'NUM_OF_BEDROOMS' with the mode
mode_VIEW_RATING = hp["VIEW_RATING"].mode()[0] # Calculates the mode
hp["VIEW_RATING"] = hp["VIEW_RATING"].fillna(mode_VIEW_RATING)

# Fill null values in 'NUM_OF_BEDROOMS' with the mode
mode_CONDITION_RATING = hp["CONDITION_RATING"].mode()[0] # Calculates the mode
hp["CONDITION_RATING"] = hp["CONDITION_RATING"].fillna(mode_NUM_OF_BEDROOMS)
```

```
In [248]: hp.isnull().sum()
```

```
Out[248]: DATE                0
NUM_OF_BEDROOMS             0
NUM_OF_BATHROOMS            0
LIVING_AREA_sqft            0
LOT_AREA_sqft               0
FLOORS                      0
WATER_FRONT                 0
VIEW_RATING                 0
CONDITION_RATING            0
ABOVE_BASEMENT_sqft         0
BASEMENT_sqft               0
YEAR_BUILT                  0
YEAR_RENOVATED              0
STREET                      0
CITYZIP                     0
STATE                      0
COUNTRY                     0
PRICE                      0
CITY                       0
ZIP                        0
dtype: int64
```

• Feature Engineering

Feature Engineering

```
In [249]: # Add a new feature for the total square feet
hp["TOTAL_HOUSE_AREA_sqft"] = hp["ABOVE_BASEMENT_sqft"] + hp["BASEMENT_sqft"]
hp.describe()
```

Out[249]:

	DATE	NUM_OF_BEDROOMS	NUM_OF_BATHROOMS	LIVING_AREA_sqft	LOT_AREA_sqft	FLOORS	WATER FRONT	VIEW_RATING	CONDITION_RA
count	4601	4,601.0	4,601.0	4,601.0	4,601.0	4,601.0	4,601.0	4,601.0	4,601.0
mean	2014-06-07 03:24:41.156270336	3.4	2.1	2,139.3	14,855.6	1.5	0.0	0.2	0.2
min	2014-05-02 00:00:00	0.0	0.0	370.0	638.0	1.0	0.0	0.0	0.0
25%	2014-05-21 00:00:00	3.0	2.0	1,460.0	5,001.0	1.0	0.0	0.0	0.0
50%	2014-06-09 00:00:00	3.0	2.0	1,980.0	7,683.0	1.5	0.0	0.0	0.0
75%	2014-06-24 00:00:00	4.0	2.0	2,620.0	11,005.0	2.0	0.0	0.0	0.0
max	2014-07-10 00:00:00	9.0	8.0	13,540.0	1,074,218.0	3.5	1.0	4.0	4.0
std	NaN	0.9	0.8	963.1	35,881.1	0.5	0.1	0.8	0.8

- Feature engineering is the process of creating or modifying the existing features which could be helpful for future analysis.
- Here, we have created a new feature 'TOTAL_HOUSE_AREA_sqft' which is sum of basement and Above Basement Area.
- This feature is created because this could be important feature for prediction of house price in machine learning models coming in future work.

• Invalid Column Correction

Invalid Column Correction

In our dataset, we have two columns namely YEAR_RENOVATED AND YEAR_BUILT which has some wrong data where built year is greater than renovated year. How can one can renovate without building a house.

```
In [250]: invalid_rows = hp[(hp['YEAR_BUILT'] > hp['YEAR_RENOVATED']) & (hp['YEAR_RENOVATED'] != 0)]
invalid_rows
```

```
Out[250]:
```

	DATE	NUM_OF_BEDROOMS	NUM_OF_BATHROOMS	LIVING_AREA_sqft	LOT_AREA_sqft	FLOORS	WATER FRONT	VIEW_RATING	CONDITION_RATING	ABOVE
31	2014-05-02	4.0	2.0	2880	3345	2.0	0.0	0.0	3.0	
60	2014-05-02	4.0	2.0	3310	6500	2.0	0.0	0.0	3.0	
61	2014-05-02	3.0	2.0	2880	5539	2.0	0.0	0.0	3.0	
64	2014-05-03	4.0	3.0	2920	6805	2.0	0.0	0.0	3.0	
68	2014-05-04	2.0	2.0	2880	7392	1.0	0.0	0.0	3.0	
...
4535	2014-06-23	3.0	2.0	1270	2358	2.0	0.0	0.0	3.0	
4542	2014-07-08	3.0	3.0	2830	3496	2.0	0.0	0.0	3.0	
4555	2014-08-19	4.0	2.0	2850	4850	2.0	0.0	0.0	3.0	
4564	2014-08-25	3.0	2.0	1230	8250	1.0	0.0	0.0	3.0	
4572	2014-07-01	4.0	2.0	3350	48748	2.0	0.0	0.0	3.0	

195 rows × 21 columns

```
In [251]: len(invalid_rows)
```

```
Out[251]: 195
```

```
In [252]: hp.shape
```

```
Out[252]: (4601, 21)
```

```
In [253]: hp.drop(invalid_rows.index, inplace=True)
```

```
hp
```

```
Out[253]:
```

	DATE	NUM_OF_BEDROOMS	NUM_OF_BATHROOMS	LIVING_AREA_sqft	LOT_AREA_sqft	FLOORS	WATER FRONT	VIEW_RATING	CONDITION_RATING	ABOVE
0	2014-05-02	3.0	2.0	1340	7912	1.5	0.0	0.0	3.0	
1	2014-05-02	5.0	2.0	3650	9050	2.0	0.0	4.0	5.0	
2	2014-05-02	3.0	2.0	1930	11947	1.0	0.0	0.0	4.0	
3	2014-05-02	3.0	2.0	2000	8030	1.0	0.0	0.0	4.0	
4	2014-05-02	4.0	2.0	1940	10500	1.0	0.0	0.0	4.0	
...
4596	2014-07-09	4.0	1.0	1600	5500	1.5	0.0	0.0	4.0	
4597	2014-06-17	4.0	3.0	3070	7201	2.0	0.0	0.0	3.0	
4598	2014-06-20	4.0	1.0	2080	5500	1.0	0.0	0.0	3.0	
4599	2014-06-25	3.0	2.0	1790	8144	2.0	0.0	0.0	3.0	
4600	2014-07-09	3.0	2.0	1770	29004	1.0	0.0	0.0	3.0	

4406 rows × 21 columns

- Invalid column correction is the process for identifying the issues and correcting it with column names so that they will not cause problems while analyzing.
- Here, we have two columns namely 'YEAR_BUILT' which refers to year when the house was built and 'YEAR_RENOVATED' which refers to when the house has been renovated.
- In the above snippet, you can find there are some rows where year_renovated is lesser than year built which is not logical.
- Hence, we have performed this cleaning step so that this will not cause problem while analyzing.

- Invalid Data Cleaning

Invalid Data Cleaning

In [254]: `len(hp)`

Out[254]: 4406

In [255]: `hp.shape`

Out[255]: (4406, 21)

In [256]: *# Based on the above summary data (and prior EDA!!!), we should get rid of some missing/invalid data*

```
hp = hp[hp["NUM_OF_BEDROOMS"] != ""]
hp = hp[hp["NUM_OF_BATHROOMS"] != ""]
hp = hp[hp["LIVING_AREA_sqft"] > 0]
hp = hp[hp["LOT_AREA_sqft"] > 0]
hp = hp[hp["FLOORS"] > 0]
hp = hp[hp["TOTAL_HOUSE_AREA_sqft"] > 0]
hp = hp[hp["VIEW_RATING"] >= 0]
hp = hp[hp["CONDITION_RATING"] >= 0]
hp = hp[hp["YEAR_BUILT"] > 1900]
hp = hp[hp["PRICE"] > 10000]
hp.describe()
```

Out[256]:

	DATE	NUM_OF_BEDROOMS	NUM_OF_BATHROOMS	LIVING_AREA_sqft	LOT_AREA_sqft	FLOORS	WATER FRONT	VIEW_RATING	CONDITION_RA
count	4336	4,336.0	4,336.0	4,336.0	4,336.0	4,336.0	4,336.0	4,336.0	4,336.0
mean	2014-06-07 04:46:56.236162304	3.4	2.1	2,116.4	15,151.6	1.5	0.0	0.2	
min	2014-05-02 00:00:00	0.0	0.0	370.0	638.0	1.0	0.0	0.0	
25%	2014-05-21 00:00:00	3.0	2.0	1,457.5	5,080.0	1.0	0.0	0.0	
50%	2014-06-09 00:00:00	3.0	2.0	1,960.0	7,750.5	1.5	0.0	0.0	
75%	2014-06-24 00:00:00	4.0	2.0	2,580.0	11,200.0	2.0	0.0	0.0	
max	2014-07-10 00:00:00	9.0	8.0	13,540.0	1,074,218.0	3.5	1.0	4.0	
std	NaN	0.9	0.8	951.0	36,647.2	0.5	0.1	0.8	

- Invalid Data Cleaning is another cleaning step where we will extract or get rid of some missing/invalid data present in the dataset by applying some conditions to each feature present which meets the minimum criteria.

• Dropping Columns

Dropping Columns

In [257]: `hp.info()`

```
<class 'pandas.core.frame.DataFrame'>
Index: 4336 entries, 0 to 4600
Data columns (total 21 columns):
#   Column                Non-Null Count  Dtype
---  -
0   DATE                   4336 non-null   datetime64[ns]
1   NUM_OF_BEDROOMS        4336 non-null   float64
2   NUM_OF_BATHROOMS       4336 non-null   float64
3   LIVING_AREA_sqft       4336 non-null   int64
4   LOT_AREA_sqft          4336 non-null   int64
5   FLOORS                 4336 non-null   float64
6   WATER_FRONT            4336 non-null   float64
7   VIEW_RATING            4336 non-null   float64
8   CONDITION_RATING       4336 non-null   float64
9   ABOVE_BASEMENT_sqft   4336 non-null   int64
10  BASEMENT_sqft          4336 non-null   int64
11  YEAR_BUILT             4336 non-null   int64
12  YEAR_RENOVATED         4336 non-null   int64
13  STREET                 4336 non-null   object
14  CITYZIP                4336 non-null   object
15  STATE                  4336 non-null   object
16  COUNTRY                4336 non-null   object
17  PRICE                  4336 non-null   float64
18  CITY                   4336 non-null   object
19  ZIP                    4336 non-null   object
20  TOTAL_HOUSE_AREA_sqft  4336 non-null   int64
dtypes: datetime64[ns](1), float64(7), int64(7), object(6)
memory usage: 745.2+ KB
```

In [258]: `hp = hp.drop(columns = ['DATE','COUNTRY','STREET','STATE'])`

In [259]: `print(hp.iloc[:, -5])`

```
0      Shoreline 98133
1      Seattle 98119
2      Kent 98042
3      Bellevue 98008
4      Redmond 98052
...
4596  Kirkland 98033
4597  Seattle 98126
4598  Seattle 98115
4599  Sammamish 98075
4600  Federalway 98003
Name: CITYZIP, Length: 4336, dtype: object
```

In [260]: `# Assuming 'index_to_drop' is the index of the column you want to drop`
`hp.drop(hp.columns[-5], axis=1, inplace=True)`

In [261]: `hp.info()`

```
<class 'pandas.core.frame.DataFrame'>
Index: 4336 entries, 0 to 4600
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   NUM_OF_BEDROOMS        4336 non-null   float64
1   NUM_OF_BATHROOMS       4336 non-null   float64
2   LIVING_AREA_sqft       4336 non-null   int64
3   LOT_AREA_sqft          4336 non-null   int64
4   FLOORS                 4336 non-null   float64
5   WATER_FRONT            4336 non-null   float64
6   VIEW_RATING            4336 non-null   float64
7   CONDITION_RATING       4336 non-null   float64
8   ABOVE_BASEMENT_sqft   4336 non-null   int64
9   BASEMENT_sqft          4336 non-null   int64
10  YEAR_BUILT             4336 non-null   int64
11  YEAR_RENOVATED         4336 non-null   int64
12  PRICE                  4336 non-null   float64
13  CITY                   4336 non-null   object
14  ZIP                    4336 non-null   object
15  TOTAL_HOUSE_AREA_sqft  4336 non-null   int64
dtypes: float64(7), int64(7), object(2)
memory usage: 575.9+ KB
```

- Dropping column is the process to remove an unwanted feature or column present in the dataset.

- We can remove some features which have less impact on predicting house price.
- In this dataset, we have removed country, state features which are same for all the rows and there would be no effect of their presence or absence in the dataset on prediction accuracy.

- **Removing Outliers**

- Removing Outlier is final important step in Data Cleaning.
- Presence of outliers will have direct impact on accuracy of the project.
- So, we will remove the outliers present in the dataset.

12. Removing Outliers

```
In [445]: sns.distplot(hp['LOT_AREA_sqft'])
```

```
C:\Users\Satvik Jonnalagadda\AppData\Local\Temp\ipykernel_10684\2312374234.py:1: UserWarning:
```

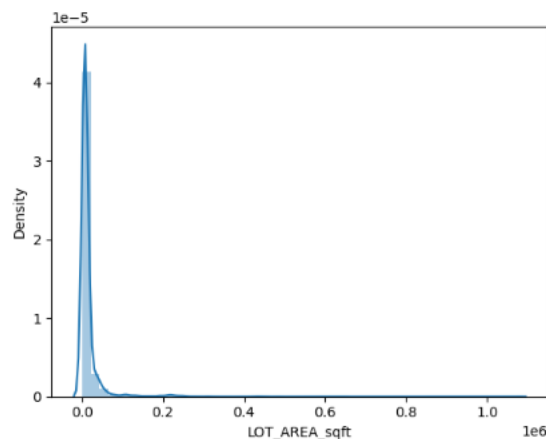
```
'distplot' is a deprecated function and will be removed in seaborn v0.14.0.
```

```
Please adapt your code to use either 'displot' (a figure-level function with similar flexibility) or 'histplot' (an axes-level function for histograms).
```

```
For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

```
sns.distplot(hp['LOT_AREA_sqft'])
```

```
Out[445]: <Axes: xlabel='LOT_AREA_sqft', ylabel='Density'>
```



- Here, you can see we have plotted for two features namely 'PRICE' and 'LOT_AREA_sqft'
- This represents their variation across the data present in the dataset.

```
In [446]: sns.distplot(hp['PRICE'])
```

```
C:\Users\Satvik Jonnalagadda\AppData\Local\Temp\ipykernel_10684\1874929932.py:1: UserWarning:
```

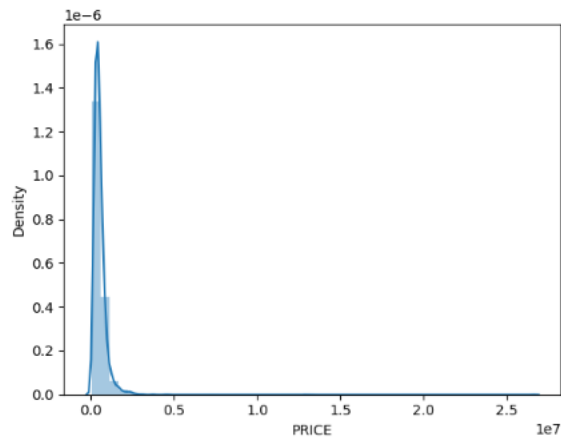
```
'distplot' is a deprecated function and will be removed in seaborn v0.14.0.
```

```
Please adapt your code to use either 'displot' (a figure-level function with similar flexibility) or 'histplot' (an axes-level function for histograms).
```

```
For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

```
sns.distplot(hp['PRICE'])
```

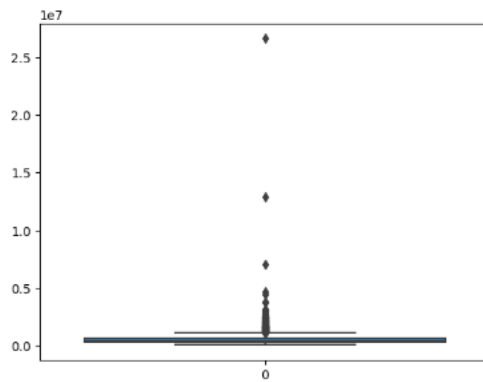
```
Out[446]: <Axes: xlabel='PRICE', ylabel='Density'>
```



- Here, you can actually see outliers present in the two features clearly.

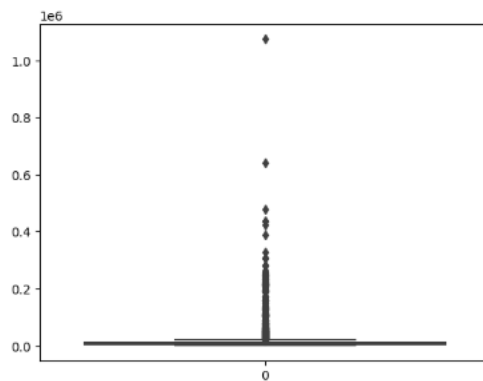
```
In [447]: #to see the outliers clearly  
sns.boxplot(hp['PRICE'])
```

Out[447]: <Axes: >



```
In [448]: #to see the outliers clearly  
sns.boxplot(hp['LOT_AREA_sqft'])
```

Out[448]: <Axes: >



```

In [449]: q1=hp['PRICE'].quantile(0.25)
          q3=hp['PRICE'].quantile(0.75)
          iqr=q3-q1
          upper_limit=q3+(1.5*iqr)
          lower_limit=q1-(1.5*iqr)
          upper_limit,lower_limit
          hp.loc[(hp['PRICE']>upper_limit) | (hp['PRICE']<lower_limit) ]

```

Out[449]:

	NUM_OF_BEDROOMS	NUM_OF_BATHROOMS	LIVING_AREA_sqft	LOT_AREA_sqft	FLOORS	WATER FRONT	VIEW_RATING	CONDITION_RATING	ABOVE_BASE
1	5.0	2.0	3650	9050	2.0	0.0	4.0	5.0	
11	4.0	2.0	2920	4000	1.5	0.0	0.0	5.0	
14	5.0	3.0	2910	9480	1.5	0.0	0.0	3.0	
99	5.0	4.0	4010	8510	2.0	0.0	1.0	5.0	
122	7.0	8.0	13540	307752	3.0	0.0	4.0	3.0	
...	
4511	5.0	6.0	7050	42840	1.0	0.0	2.0	4.0	
4539	5.0	4.0	5850	22885	2.0	0.0	2.0	4.0	
4570	4.0	4.0	6640	53330	2.0	0.0	0.0	3.0	
4581	6.0	3.0	4710	11000	2.0	0.0	3.0	3.0	
4588	3.0	3.0	2500	5568	2.0	0.0	0.0	5.0	

231 rows × 10 columns

```

In [450]: hp=hp.loc[(hp['PRICE']<upper_limit) & (hp['PRICE']>lower_limit)]

```

```

In [451]: q1=hp['LOT_AREA_sqft'].quantile(0.25)
          q3=hp['LOT_AREA_sqft'].quantile(0.75)
          iqr=q3-q1
          upper_limit=q3+(1.5*iqr)
          lower_limit=q1-(1.5*iqr)
          upper_limit,lower_limit
          hp.loc[(hp['LOT_AREA_sqft']>upper_limit) | (hp['LOT_AREA_sqft']<lower_limit) ]

```

Out[451]:

	NUM_OF_BEDROOMS	NUM_OF_BATHROOMS	LIVING_AREA_sqft	LOT_AREA_sqft	FLOORS	WATER FRONT	VIEW_RATING	CONDITION_RATING	ABOVE_BASE
7	4.0	2.0	2710	35868	2.0	0.0	0.0	3.0	
8	3.0	2.0	2430	88426	1.0	0.0	0.0	4.0	
28	5.0	2.0	2820	67518	2.0	0.0	0.0	3.0	
34	4.0	2.0	3630	42884	1.5	0.0	0.0	3.0	
35	3.0	2.0	3240	33151	2.0	0.0	2.0	3.0	
...	
4562	3.0	2.0	3320	478288	1.5	0.0	3.0	4.0	
4566	3.0	2.0	1940	219527	1.0	0.0	0.0	3.0	
4569	1.0	1.0	930	29258	1.0	0.0	0.0	3.0	
4589	3.0	2.0	1440	28516	1.0	0.0	0.0	4.0	
4600	3.0	2.0	1770	29004	1.0	0.0	0.0	3.0	

487 rows × 10 columns

```

In [452]: hp=hp.loc[(hp['LOT_AREA_sqft']<upper_limit) & (hp['LOT_AREA_sqft']>lower_limit)]

```

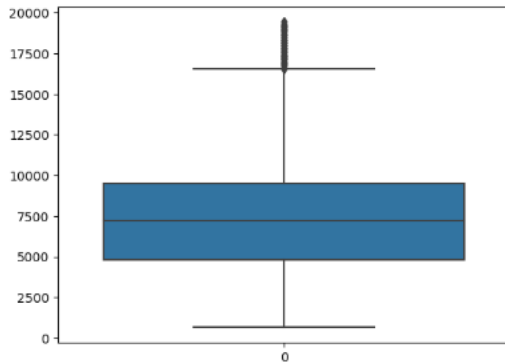
- In the above snippet, we have some operations to remove outliers present by calculating quartile ranges and limiting them to upper and lower boundary correspondingly.

Box Plot after Removing Outliers:

Box Plot after removing outliers

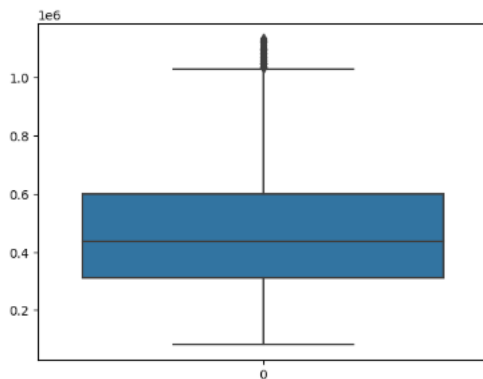
```
In [456]: sns.boxplot(hp['LOT_AREA_sqFe'])
```

Out[456]: <Axes: >



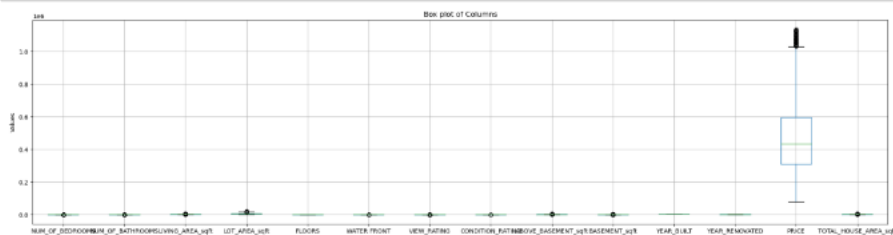
```
In [457]: sns.boxplot(hp['PRICE'])
```

Out[457]: <Axes: >



- This is box plot representation after removing outliers in those two columns.

```
In [455]: plt.figure(figsize=(25, 6))  
hp.boxplot()  
plt.ylabel('Values')  
plt.title('Box plot of Columns')  
plt.show()
```



Exploratory Data Analysis

EDA, or Exploratory Data Analysis, is a process to analyze data sets to summarize their main characteristics, using visual methods. It helps understand what the data can tell us beyond the formal modeling or performing hypothesis. EDA is primarily telling the relationship between features and their interdependency.

In this context, we have performed many operations to obtain relationship between features.

Firstly, we analyze the dataset and get its basic details like shape, number of rows and features present, data types and checking if there are any null values present After the cleaning has performed.

Secondly, we will extract correlation matrix which clearly describes the interdependency of the features.

Based on the correlation matrix, using different plots like bar plot, line plot, pie chart, count plot etc. we are going to show the graphical representation that states relationship between features and how they are varying across the dataset.

- **Dimension of the dataset after Cleaning**

1. Dimensions of the dataset

```
In [458]: hp.shape
```

```
Out[458]: (3618, 16)
```

- Here, we are extracting the number of row and columns present in the dataset after cleaning process.
- You can see that there are a total of 3618 rows and 16 features or columns are present in the dataset.

- **Extracting first 5 rows of the dataset**

2. First 5 rows of the dataset

```
In [459]: hp.head(5)
```

```
Out[459]:
```

	NUM_OF_BEDROOMS	NUM_OF_BATHROOMS	LIVING_AREA_sqft	LOT_AREA_sqft	FLOORS	WATER FRONT	VIEW_RATING	CONDITION_RATING	ABOVE_BASEMEN
0	3.0	2.0	1340	7912	1.5	0.0	0.0	3.0	
2	3.0	2.0	1930	11947	1.0	0.0	0.0	4.0	
3	3.0	2.0	2000	8030	1.0	0.0	0.0	4.0	
4	4.0	2.0	1940	10500	1.0	0.0	0.0	4.0	
5	2.0	1.0	880	6380	1.0	0.0	0.0	3.0	

- Here, we have displayed first five rows present in the dataset after cleaning.

- **Checking datatypes of features after cleaning**

3. Datatypes of each Column

```
In [460]: hp.dtypes
```

```
Out[460]: NUM_OF_BEDROOMS      float64
NUM_OF_BATHROOMS      float64
LIVING_AREA_sqft      int64
LOT_AREA_sqft         int64
FLOORS                float64
WATER_FRONT           float64
VIEW_RATING           float64
CONDITION_RATING      float64
ABOVE_BASEMENT_sqft   int64
BASEMENT_sqft         int64
YEAR_BUILT             int64
YEAR_RENOVATED         int64
PRICE                 float64
CITY                   object
ZIP                   object
TOTAL_HOUSE_AREA_sqft int64
dtype: object
```

- Here, we can see different data types allocated to different features based on their data.

- **Checking for null values if any exist after cleaning**

Check for Missing Values

```
In [269]: print(hp.isnull().sum())
```

```
NUM_OF_BEDROOMS      0
NUM_OF_BATHROOMS      0
LIVING_AREA_sqft      0
LOT_AREA_sqft         0
FLOORS                0
WATER_FRONT           0
VIEW_RATING           0
CONDITION_RATING      0
ABOVE_BASEMENT_sqft   0
BASEMENT_sqft         0
YEAR_BUILT             0
YEAR_RENOVATED         0
PRICE                 0
CITY                   0
ZIP                   0
TOTAL_HOUSE_AREA_sqft 0
dtype: int64
```

- After Cleaning has performed, we are checking if any null values exist in dataset.

- **Summary of the dataset after cleaning**

5. Summary Statistics

```
In [462]: summary_stats = hp.describe()
print(summary_stats)
```

	NUM_OF_BEDROOMS	NUM_OF_BATHROOMS	LIVING_AREA_sqft	LOT_AREA_sqft	\
count	3,618.0	3,618.0	3,618.0	3,618.0	
mean	3.3	2.0	1,933.2	7,368.9	
std	0.9	0.7	735.2	3,643.6	
min	0.0	0.0	370.0	638.0	
25%	3.0	2.0	1,390.0	4,800.0	
50%	3.0	2.0	1,850.0	7,201.0	
75%	4.0	2.0	2,370.0	9,502.2	
max	9.0	6.0	5,960.0	19,439.0	

	FLOORS	WATER_FRONT	VIEW_RATING	CONDITION_RATING	\
count	3,618.0	3,618.0	3,618.0	3,618.0	
mean	1.5	0.0	0.2	3.5	
std	0.5	0.0	0.6	0.7	
min	1.0	0.0	0.0	1.0	
25%	1.0	0.0	0.0	3.0	
50%	1.0	0.0	0.0	3.0	
75%	2.0	0.0	0.0	4.0	
max	3.5	1.0	4.0	5.0	

	ABOVE_BASEMENT_sqft	BASEMENT_sqft	YEAR_BUILT	YEAR_RENOVATED	\
count	3,618.0	3,618.0	3,618.0	3,618.0	
mean	1,650.0	283.2	1,968.8	757.2	
std	689.3	409.5	29.7	969.6	
min	370.0	0.0	1,901.0	0.0	
25%	1,140.0	0.0	1,948.0	0.0	
50%	1,460.0	0.0	1,970.0	0.0	
75%	2,020.0	570.0	1,995.0	1,998.0	
max	5,190.0	2,150.0	2,014.0	2,014.0	

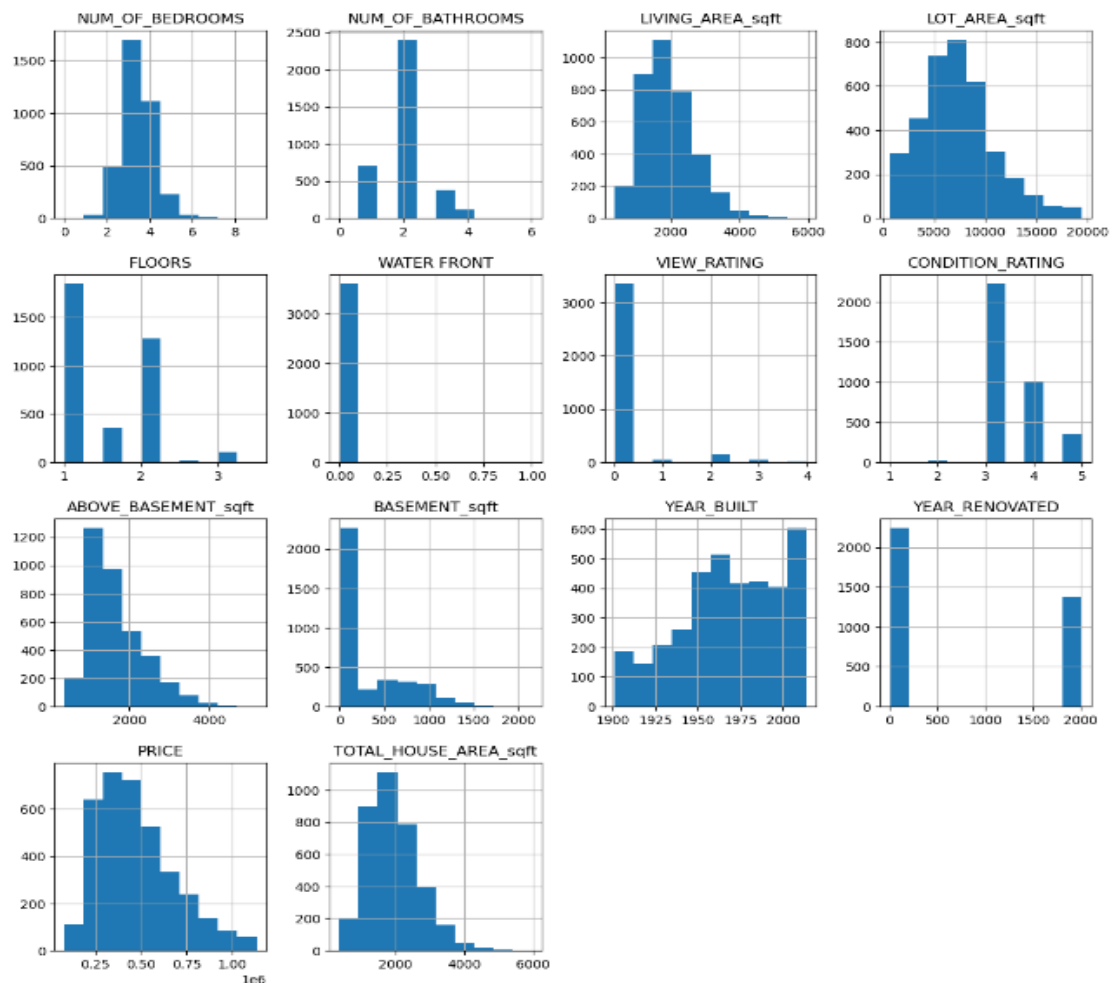
	PRICE	TOTAL_HOUSE_AREA_sqft
count	3,618.0	3,618.0
mean	473,434.7	1,933.2
std	209,236.3	735.2
min	80,000.0	370.0
25%	310,000.0	1,390.0
50%	435,750.0	1,850.0
75%	598,750.0	2,370.0
max	1,135,000.0	5,960.0

- This is summary of the dataset after performing cleaning.
- You can find all the list of features present and their brief info of mean, count etc.

- **Histogram representation after cleaning dataset**

```
In [463]: hp.hist(figsize=(14,14))
```

```
Out[463]: array([[<Axes: title={'center': 'NUM_OF_BEDROOMS'}>,<Axes: title={'center': 'NUM_OF_BATHROOMS'}>,<Axes: title={'center': 'LIVING_AREA_sqft'}>,<Axes: title={'center': 'LOT_AREA_sqft'}>],<Axes: title={'center': 'FLOORS'}>,<Axes: title={'center': 'WATER_FRONT'}>,<Axes: title={'center': 'VIEW_RATING'}>,<Axes: title={'center': 'CONDITION_RATING'}>],<Axes: title={'center': 'ABOVE_BASEMENT_sqft'}>,<Axes: title={'center': 'BASEMENT_sqft'}>,<Axes: title={'center': 'YEAR_BUILT'}>,<Axes: title={'center': 'YEAR_RENOVATED'}>],<Axes: title={'center': 'PRICE'}>,<Axes: title={'center': 'TOTAL_HOUSE_AREA_sqft'}>,<Axes: >,<Axes: >]], dtype=object)
```



- This is the histogram representation of all the features present in the dataset after cleaning.
- Here you can find how their values are distributed across the dataset.

- Here, In the above Histogram, I can see that LOT_AREA_sqft and PRICE has most similarity in their graph.
- LIVING_AREA and ABOVE_BASEMENT has similarity in the graph.
- WATER_FRONT and VIEW_RATING are other similar group

• Correlation Analysis

correlation matrix helps identify potential relationships between different features, which can be useful for further analysis and modeling.

```
In [464]: hp[["NUM_OF_BEDROOMS", "NUM_OF_BATHROOMS", "LIVING_AREA_sqft", "LOT_AREA_sqft", "FLOORS", "WATER_FRONT", "VIEW_RATING", "CONDITION_RAT
```

Out[464]:

	NUM_OF_BEDROOMS	NUM_OF_BATHROOMS	LIVING_AREA_sqft	LOT_AREA_sqft	FLOORS	WATER_FRONT	VIEW_RATING	CONDITION_RA
NUM_OF_BEDROOMS	1.0	0.5	0.6	0.2	0.1	-0.0	0.1	
NUM_OF_BATHROOMS	0.5	1.0	0.6	0.1	0.3	0.0	0.2	
LIVING_AREA_sqft	0.6	0.6	1.0	0.3	0.3	0.0	0.2	
LOT_AREA_sqft	0.2	0.1	0.3	1.0	-0.3	0.1	0.1	
FLOORS	0.1	0.3	0.3	-0.3	1.0	0.0	-0.0	
WATER_FRONT	-0.0	0.0	0.0	0.1	0.0	1.0	0.3	
VIEW_RATING	0.1	0.2	0.2	0.1	-0.0	0.3	1.0	
CONDITION_RATING	0.0	-0.1	-0.1	0.1	-0.3	0.0	0.0	
ABOVE_BASEMENT_sqft	0.5	0.5	0.8	0.2	0.5	0.0	0.1	
BASEMENT_sqft	0.3	0.3	0.4	0.1	-0.3	0.0	0.2	
YEAR_BUILT	0.1	0.4	0.3	0.0	0.5	-0.0	-0.1	
YEAR_RENOVATED	-0.1	-0.2	-0.2	0.1	-0.3	-0.0	0.0	
PRICE	0.3	0.4	0.6	0.0	0.3	0.0	0.2	
TOTAL_HOUSE_AREA_sqft	0.6	0.6	1.0	0.3	0.3	0.0	0.2	

From the above correlation analysis, we are gonna extract some important features

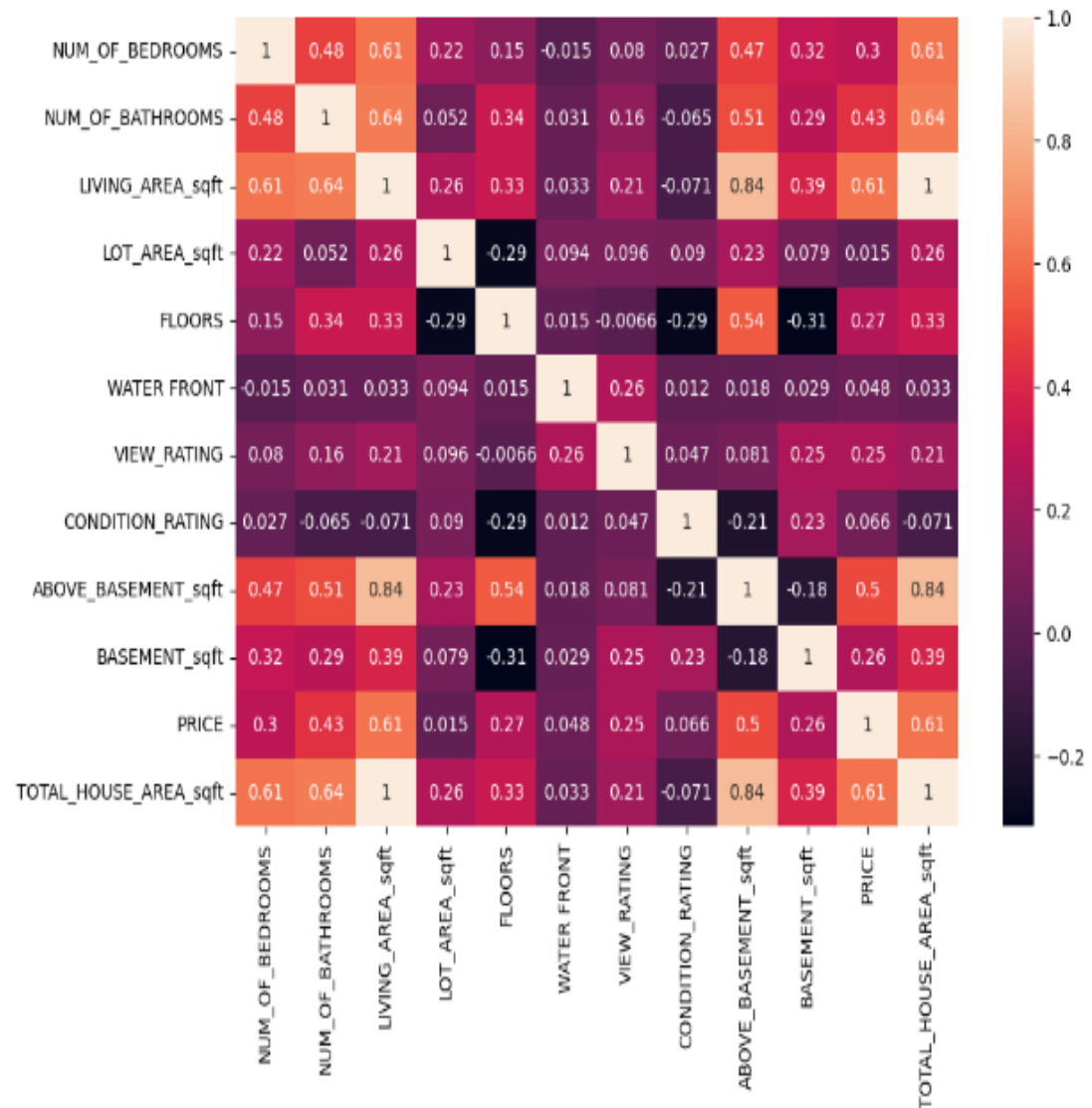
```
In [465]: # Select features we are interested in
df = hp[["NUM_OF_BEDROOMS", "NUM_OF_BATHROOMS", "LIVING_AREA_sqft", "LOT_AREA_sqft", "FLOORS", "ABOVE_BASEMENT_sqft", "BASEMENT_sqf
```

- Correlation Analysis helps to identify the potential relationship between different features, which can be useful for further analysis and modeling.

- Here I can see that WATER_FRONT does not have any effect on any of the features.
- LIVING_AREA and ABOVE_BASEMENT has 0.8 which are most dependent features
- PRICE is mostly dependent on TOTAL_HOUSE_AREA_sqft and TOTAL_HOUSE_AREA_sqft has dependency on ABOVE_BASEMENT also.

- **Clustered Heat Map for Correlation Matrix**

```
In [475]: plt.figure(figsize=(10,7))
sns.heatmap(hp.select_dtypes(include="number").drop(['YEAR_BUILT', 'YEAR_RENOVATED'], axis=1).corr(),annot=True);
```



LIVING_AREA_sqft has the biggest correlation rate with the PRICE, it is also correlating strongly with NUM_OF_BATHROOMS and ABOVE_BASEMENT_sqft.

The biggest correlation is between LIVING_AREA_sqft and ABOVE_BASEMENT_sqft(0.84)

- This Clustered Heat Map of Correlation matrix also used to describe how features are interdependent on each other.

- You can see that LIVING_AREA and ABOVE_BASEMENT has highest value.
- Here we have dropped two columns namely Year Renovated and Year Built which cannot be used for finding correlation.

- **Count Plot**

```
In [480]: count=hp["CITY"].value_counts()  
count
```

```
Out[480]: CITY  
Seattle      1340  
Renton       237  
Bellevue     211  
Redmond      166  
Kirkland     159  
Kent         145  
Issaquah     140  
Sammamish    130  
Auburn       127  
FederalWay   118  
Shoreline    104  
MapleValley  68  
Burien       64  
Woodinville  63  
Kenmore      54  
Snoqualmie   54  
MercerIsland 50  
DesMoines    50  
LakeForestPark 35  
Newcastle    32  
Covington    32  
NorthBend    31  
Duvall       26  
Bothell      26  
SeaTac       25  
Enumclaw     23  
Tukwila      22  
Vashon       16  
NormandyPark 15  
Carnation    12  
ClydeHill    7  
BlackDiamond 6  
Pacific      6  
Algona       4  
FallCity     4  
Skykomish    3  
Medina       3  
Milton       2  
YarrowPoint  2  
Ravensdale   2  
BeauxArtsVillage 1  
Preston      1  
SnoqualmiePass 1  
InglewoodFinnHill 1  
Name: count, dtype: int64
```

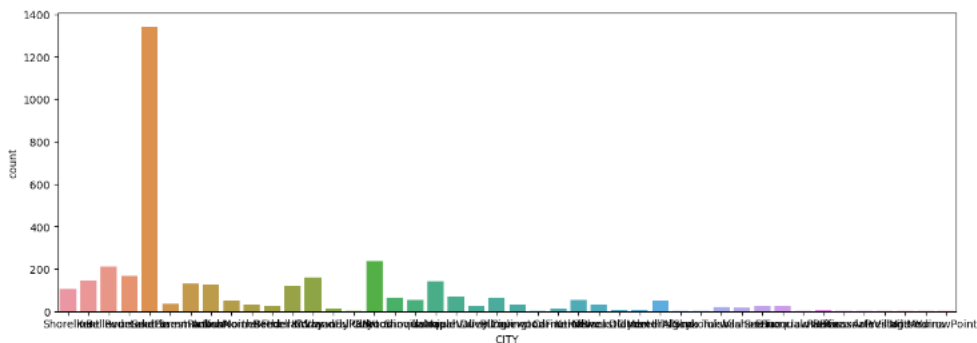
```
In [481]: len(count)
```

```
Out[481]: 44
```

- Count plot is used to find sum of the values present in a feature in a graphical representation
- In the above snippet you can find we have taken city feature to find how many houses present in different cities across the dataset.
- From this count plot we can say that Seattle has many houses when compares to any other city.
- There are total of 44 different cities present in the dataset

```
In [495]: plt.figure(figsize=(15,5))
sns.countplot(x="CITY",data=hp)

Out[495]: <Axes: xlabel='CITY', ylabel='count'>
```



• Additional Results from Correlation Analysis

- In the above Correlation Analysis, we have already mentioned some results we have extracted from the dataset.
- When we have checked correlation matrix, we have found that Condition Rating has no dependency on the Price.

- But if you check with below code snippet, you will find how price is varying for different unique values of condition rating which shows a clear inter dependency.

From Correlation Analysis let's get some results

```
In [279]: pd.crosstab(index = hp.CONDITION_RATING , values = hp.PRICE , columns = "PRICE",aggfunc = "mean")
```

```
Out[279]:
```

	col_0	PRICE
CONDITION_RATING		
1.0	366,400.0	
2.0	334,837.4	
3.0	551,325.0	
4.0	539,736.1	
5.0	647,182.4	

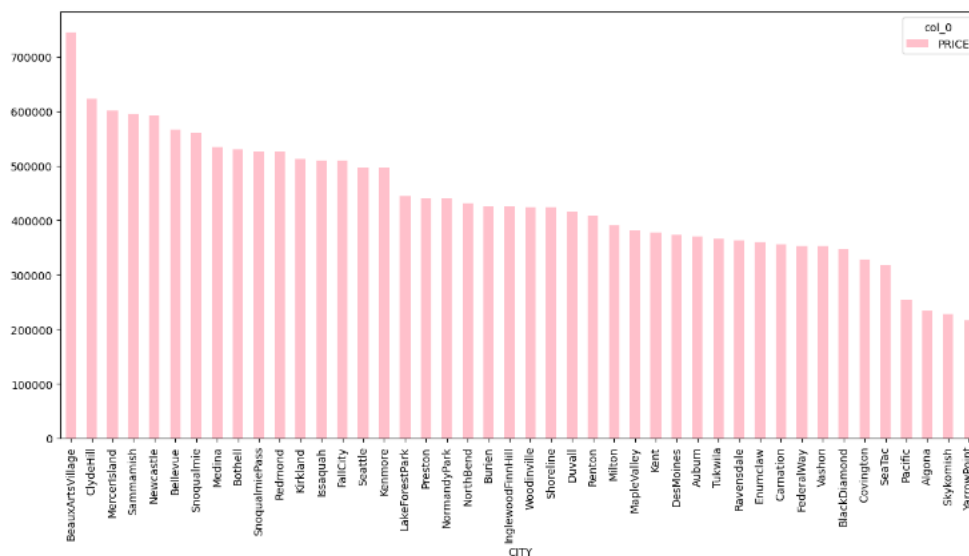
Result

From the above correlation matrix, you can find that there is no relation between Condition and Price but when you observe the above result you can find that price is increasing as the rate of house condition is increasing.

• Bar Plot

```
In [488]: hp.PRICE , aggfunc = "mean").sort_values(ascending = False , by = "PRICE").plot(kind = "bar" , figsize = (15,7) , color = "pink")
```

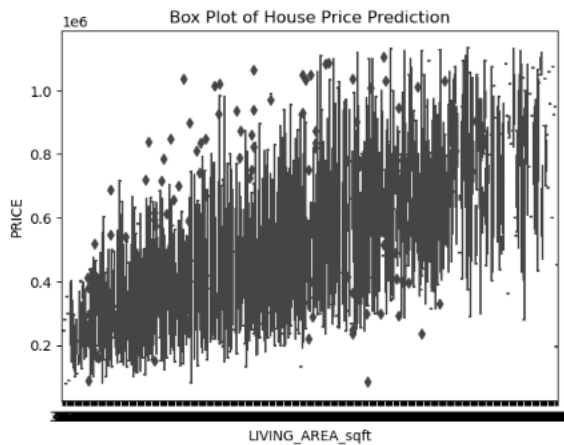
```
Out[488]: <Axes: xlabel='CITY'>
```



- This is the bar plot representation that describes how price of the houses is varying across different cities in the dataset.
 - You can see that Beaux Art Village has highest price among all other cities present in the dataset.
- **Box Plot Representation between features**

10. BOX PLOT b/w LIVING_AREA_sqft and PRICE

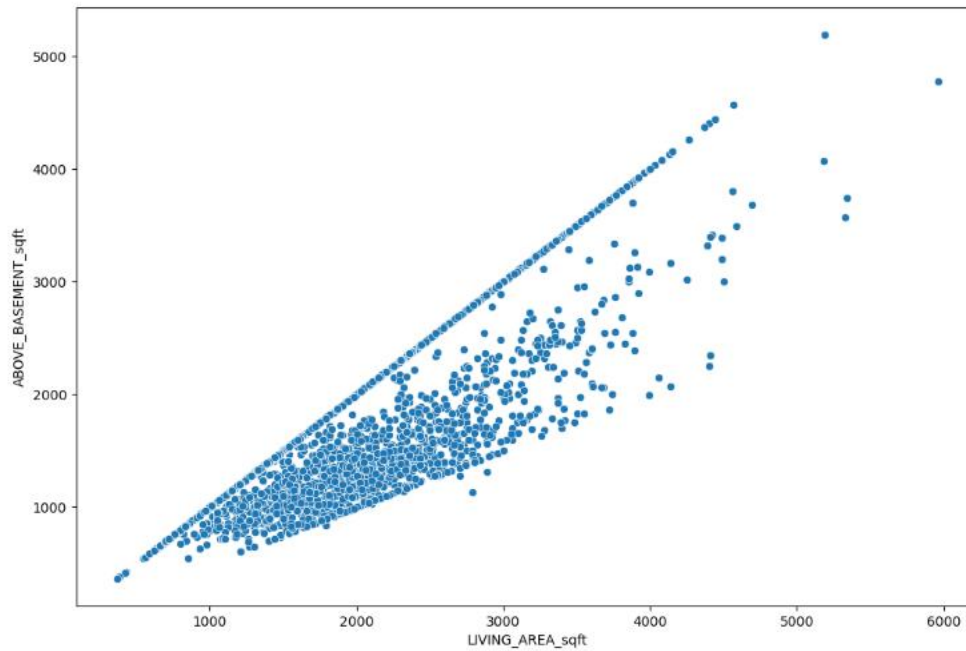
```
In [489]: import seaborn as sns  
  
# Plot box plot of a numerical feature  
sns.boxplot(x=hp['LIVING_AREA_sqft'], y=hp['PRICE'])  
plt.xlabel('LIVING_AREA_sqft')  
plt.ylabel('PRICE')  
plt.title('Box Plot of House Price Prediction')  
plt.show()
```



- This is the box plot representation that describes the interdependency between PRICE and LIVING_AREA.
- The Above Box Plot shows those both have robust dependency.
- We have chosen these two features because they have high correlation value.

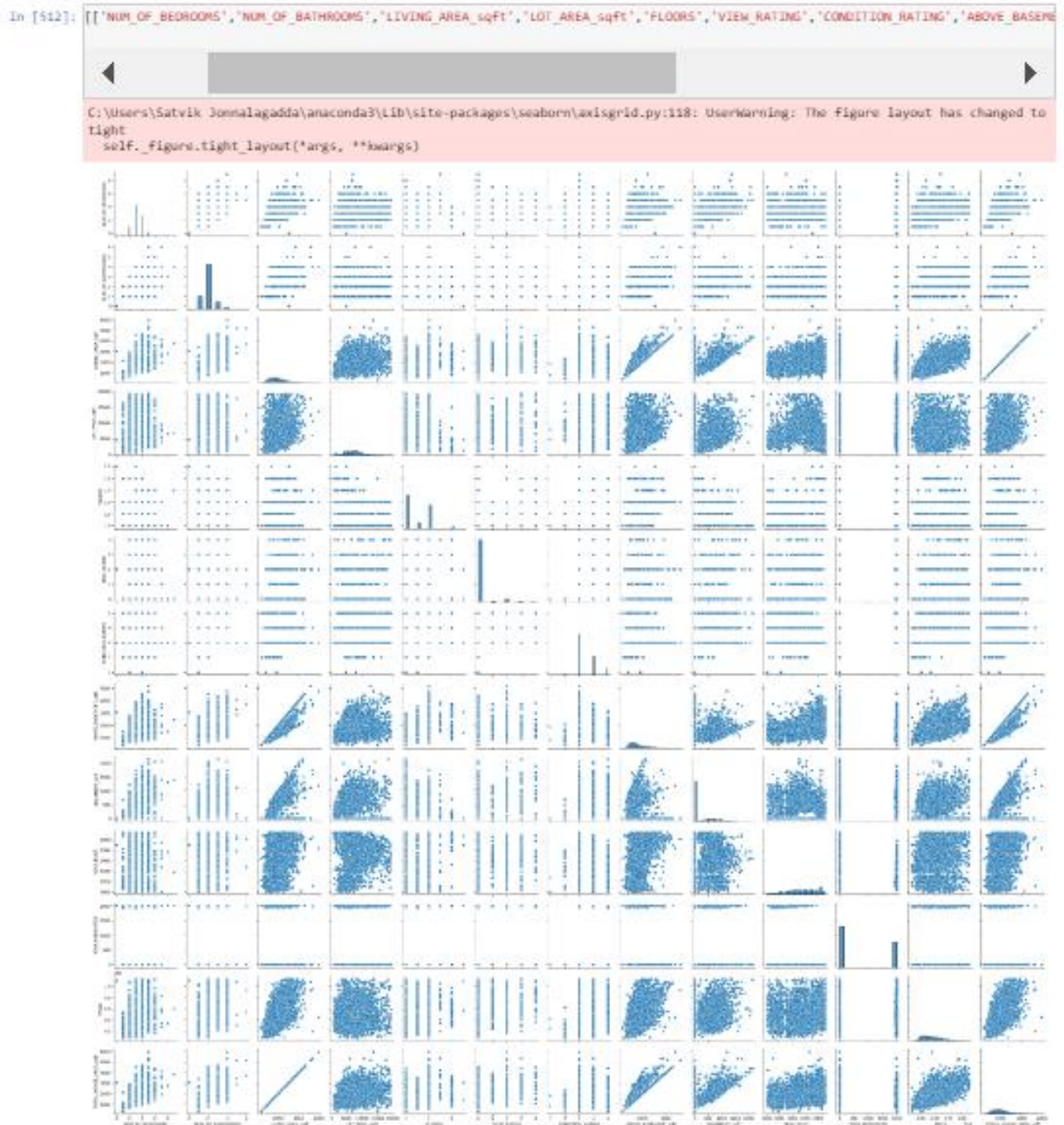
- **Scatter Plot Representation between features**

```
In [490]: plt.figure(figsize=(12,8))  
sns.scatterplot(x='LIVING_AREA_sqft',y='ABOVE_BASEMENT_sqft',data=df)  
Out[490]: <Axes: xlabel='LIVING_AREA_sqft', ylabel='ABOVE_BASEMENT_sqft'>
```



- This is the box plot representation that describes the interdependency between ABOVE BASEMENT AREA and LIVING_AREA.
- The Above Box Plot shows those both have robust dependency.
- We have chosen these two features because they have high correlation value.

- **Pair Plot**

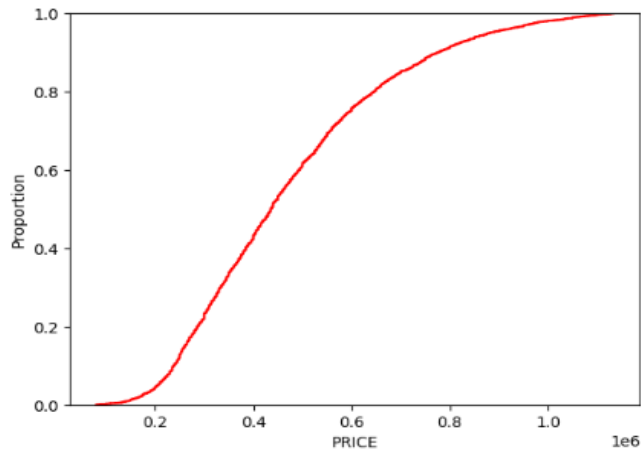


- This is the pair plot representation that describe relationship between every two attributes present in the dataset.

- **Understanding the cumulative distribution function for price**

```
In [505]: sns.ecdfplot(df['PRICE'],stat='proportion',color='red')
```

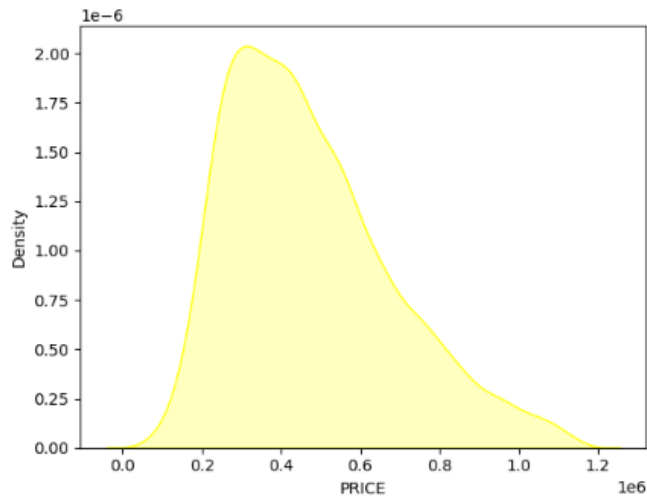
```
Out[505]: <Axes: xlabel='PRICE', ylabel='Proportion'>
```



- This is cumulative distribution function graph that represents how price is distributed across the data.
 - A curve-shaped CDF shows that prices are distributed continuously across a range of values, rather than being concentrated at specific points or intervals.
- **Analyzing probability density function for a numerical variable i.e., price using kde plots.**

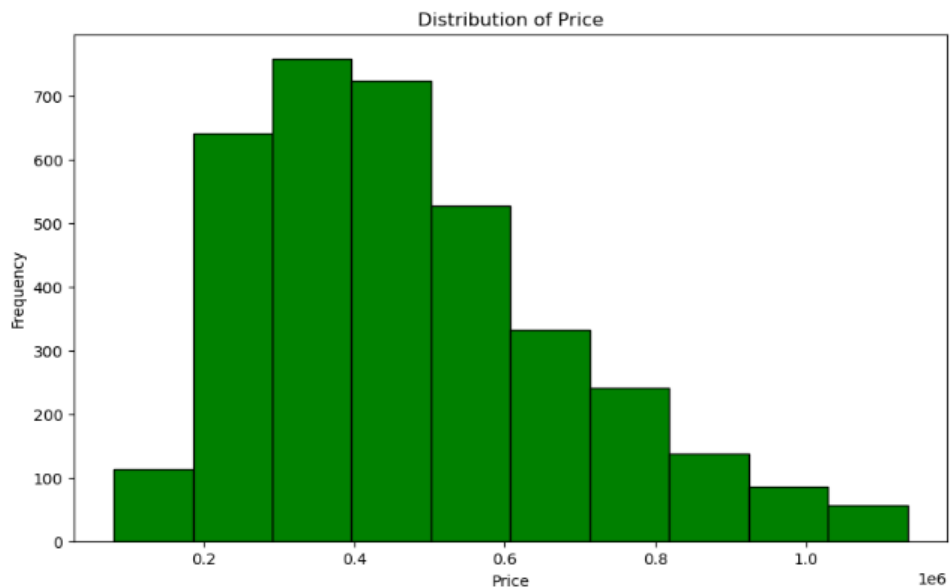
```
In [507]: sns.kdeplot(hp['PRICE'],fill=True,color='yellow')
```

```
Out[507]: <Axes: xlabel='PRICE', ylabel='Density'>
```



- Visualize how the price is distributed

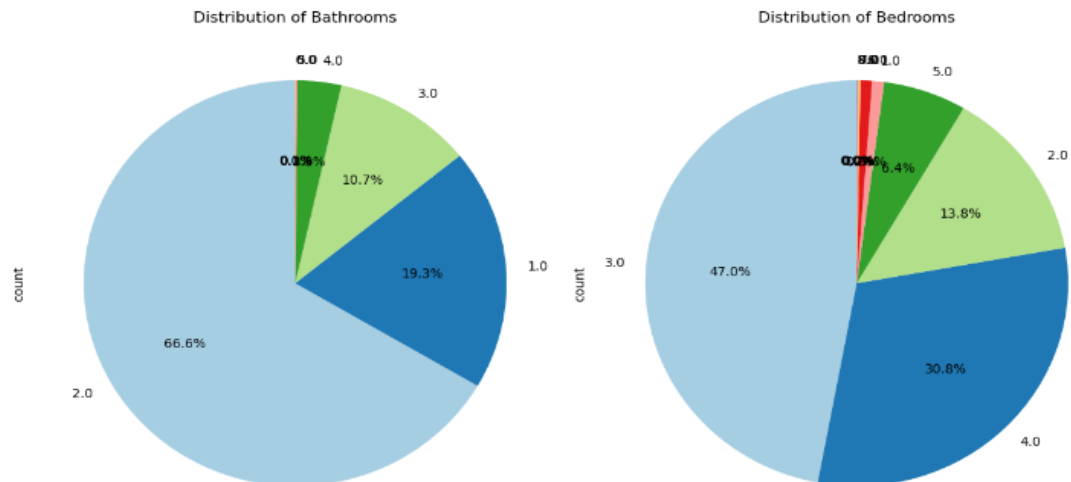
```
In [509]: plt.figure(figsize=(10, 6))  
plt.hist(hp['PRICE'], bins=10, color='green', edgecolor='black')  
plt.title('Distribution of Price')  
plt.xlabel('Price')  
plt.ylabel('Frequency')  
plt.show()
```



- In this Bar Chart, you can observe how price is distributed across the dataset.

- **Pie Chart to explore two categories' Bathrooms and Bedrooms**

```
In [510]: #Percentage distribution of source and destination
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
hp['NUM_OF_BATHROOMS'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=plt.cm.Paired.colors)
plt.title('Distribution of Bathrooms')
# Create a pie chart for Category2
plt.subplot(1, 2, 2)
hp['NUM_OF_BEDROOMS'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=plt.cm.Paired.colors)
plt.title('Distribution of Bedrooms')
plt.tight_layout()
plt.show()
```

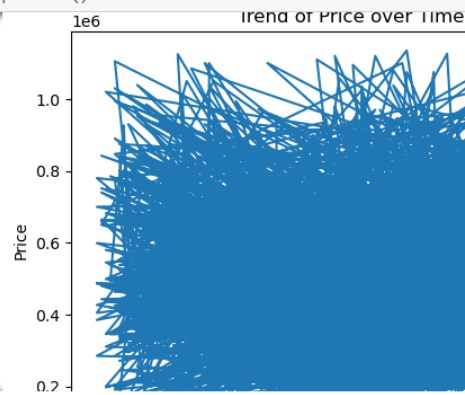


- In this PIE chart, you can find that we have taken two features namely NUM_OF_BATHROOMS and NUM_OF_BEDROOMS which have their distribution, respectively.
- From this, we can analyze that Most houses have 2 bathrooms with 68% and 3 bedrooms with 47%.

- **Plot between the Year Built and Price Variation over time**

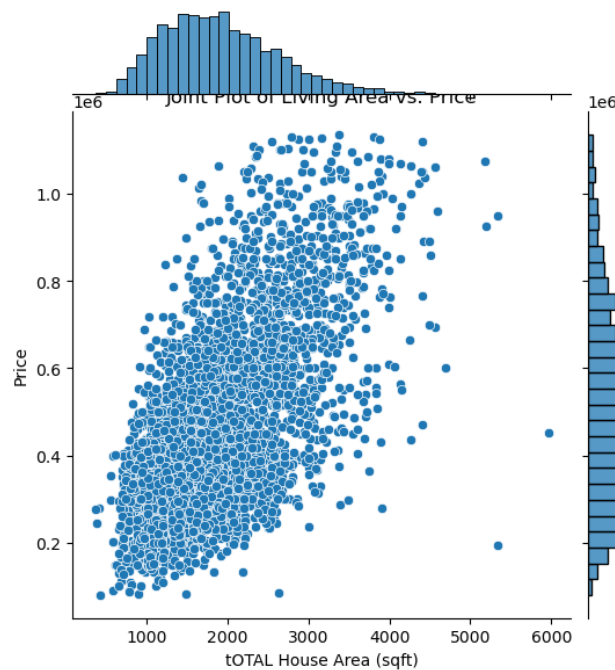
- The plot represents the relationship between price and year built.
- The above graph represents that price has been uneven overtime.

```
In [520]: plt.plot(hp['YEAR_BUILT'], hp['PRICE'])
plt.xlabel('Year Built')
plt.ylabel('Price')
plt.title('Trend of Price over Time')
plt.show()
```



- **Joint Plot Between TOTAL_HOUSE_AREA_sqft and PRICE**

```
In [517]: sns.jointplot(x='TOTAL_HOUSE_AREA_sqft', y='PRICE', data=hp, kind='scatter')
plt.xlabel('TOTAL House Area (sqft)')
plt.ylabel('Price')
plt.title('Joint Plot of Living Area vs. Price')
plt.show()
```



Algorithms

Linear Regression:

1. Linear Regression

```
In [351]: from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1688, shuffle=True, stratify=None)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model_1 = LinearRegression()
model_1.fit(X_train_scaled, y_train)

y_pred = model_1.predict(X_test_scaled)

r2_lr = r2_score(y_test, y_pred)
mse_lr = mean_squared_error(y_test, y_pred)

print("R Squared Error:", r2_lr)
print("Mean Squared Error:", mse_lr)

R Squared Error: 0.6280344569935457
Mean Squared Error: 16817657403.796228
```

- ✓ Linear Regression is a machine learning algorithm which is used to predict a continuous response variable using one or more independent variables.
- ✓ In our context, Predicting House Prices, which are continuous and like have a linear relationship with features like NUM_OF_BEDROOMS, CITY, etc.
- ✓ Hence, we have chosen Linear Regression for this regression task.

Code Explanation:

- ✓ In the code part, you can see that we have first scaled our dataset using 'StandardScaler'.

- ✓ We have standardized the features to have a mean of zero and standard deviation of one.
- ✓ In the code, you can see that we have mentioned `random_state = 1688`.
- ✓ We have achieved this state which could be best fitting value by model tuning the parameters.
- ✓ To achieve this, we have used common techniques like GridSearch and Cross-Validation.

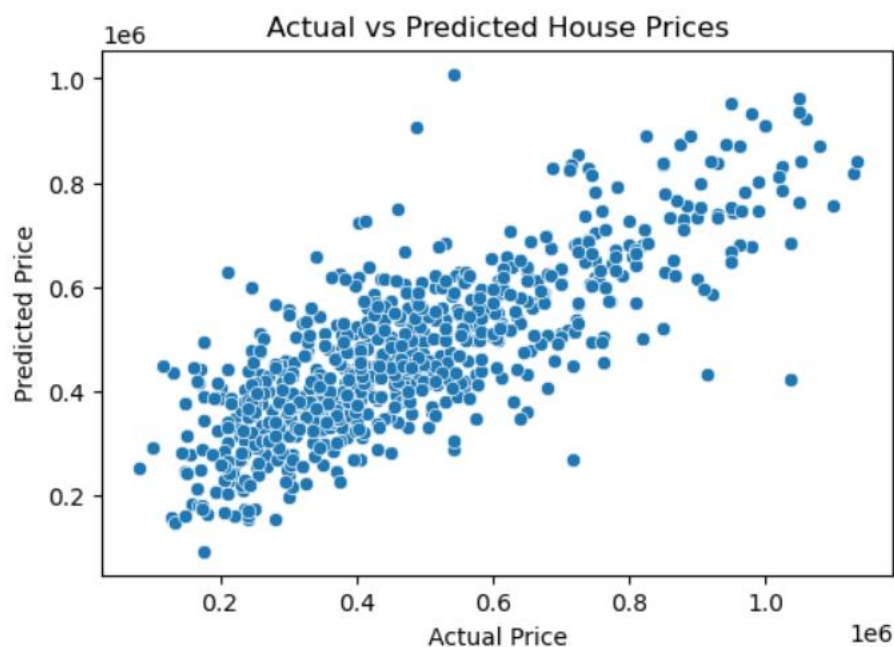
Metrics & Evaluation:

In this model, the metrics used to evaluate the Linear Regression model are R-squared and Mean Squared Error (MSE).

- ✓ The R- Squared value is 0.628, which says that there is approximately 62.8% of the variance in the house price that can be explained by the model.
- ✓ The MSE is 16817657403.796228 which is very high this also states that the model predictions are quite far off from the actual values on average.
- ✓ These results are less because of complexity of the dataset and house prices which are varying in millions.
- ✓ We have training and testing error as 0.545 and 0.628, respectively.

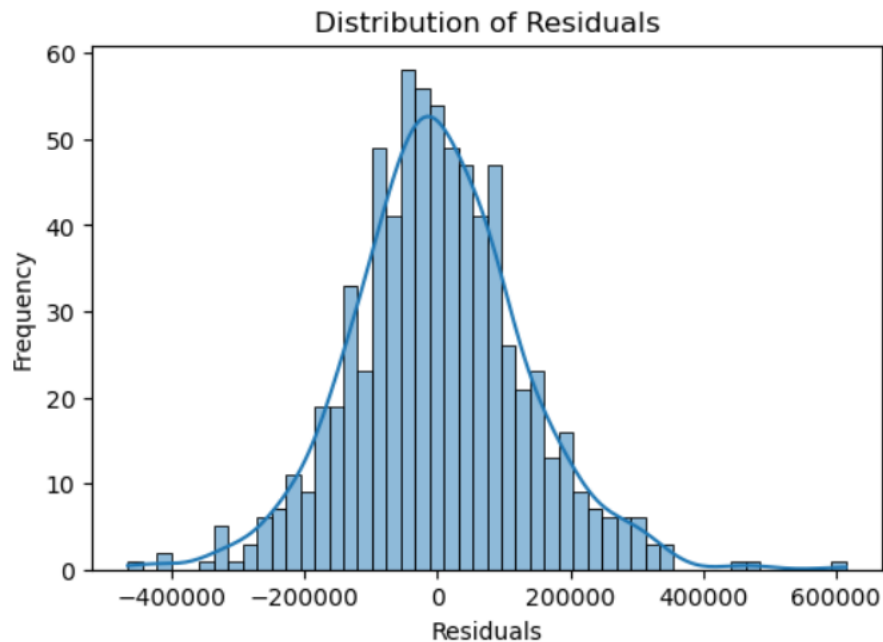
Visualization:

- ✓ The scatter plot of actual vs predicted house prices shows a positive correlation, which is good remark, but there is still a lot of variances because of higher valued house prices.
- ✓ From the plot, we can say that the model could be underfitting and not obtaining all the complexities of the data.



- ✓ The histogram of residuals which is the difference between the actual and predicted values indicates a normal distribution, which is good sign that model is unbiased.
- ✓ In the graph below, you can see that there is long tail on both sides indicating that some predictions are quite far.

- ✓ This indicates that there are nonlinear relationships that the model is not obtaining.



K Nearest Neighbor:

2. K Nearest Neighbors

```
In [356]: from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=766)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model_2 = KNeighborsRegressor(n_neighbors=9)

model_2.fit(X_train_scaled, y_train)

y_pred = model_2.predict(X_test_scaled)
mse_knn = mean_squared_error(y_test, y_pred)

r2_knn = r2_score(y_test, y_pred)
print(f"R Squared Error: {r2_knn}")
print("Mean Squared Error:", mse_knn)

R Squared Error: 0.4548642382975875
Mean Squared Error: 23716319555.336937
```

- ✓ K Nearest Neighbors (KNN) is a simple, non-parametric and instance-based learning algo used for classification and regression tasks.
- ✓ KNN makes its predictions using a similarity measure, in cases where relationship between points may necessarily not linear. KNN can capture complex relationships without assuming a specific form for the output model.
- ✓ In our context, it takes the median value of the 'k' nearest neighbors to predict continuous value.
- ✓ Hence, we have used this KNN for this regression task.

Code Explanation:

- ✓ In the code part, you can see that we have first scaled our dataset using 'StandardScalar'.
- ✓ We have standardized the features to have a mean of zero and standard deviation of one.
- ✓ We have also chosen n-neighbors = 9 which indicates there could be 9 nearest points in the feature space when algorithm makes a prediction.
- ✓ In the code, you can see that we have mentioned random_state = 766.

- ✓ We have achieved this state and n-neighbors which could be best fitting value by model tuning the parameters.
- ✓ We also checked with n-neighbors ranging from 1 to 100 and random_state ranging from 1 to 10000 and ensured the above values are best fit to the model.
- ✓ To achieve this, we have used common techniques like GridSearch and Cross-Validation.

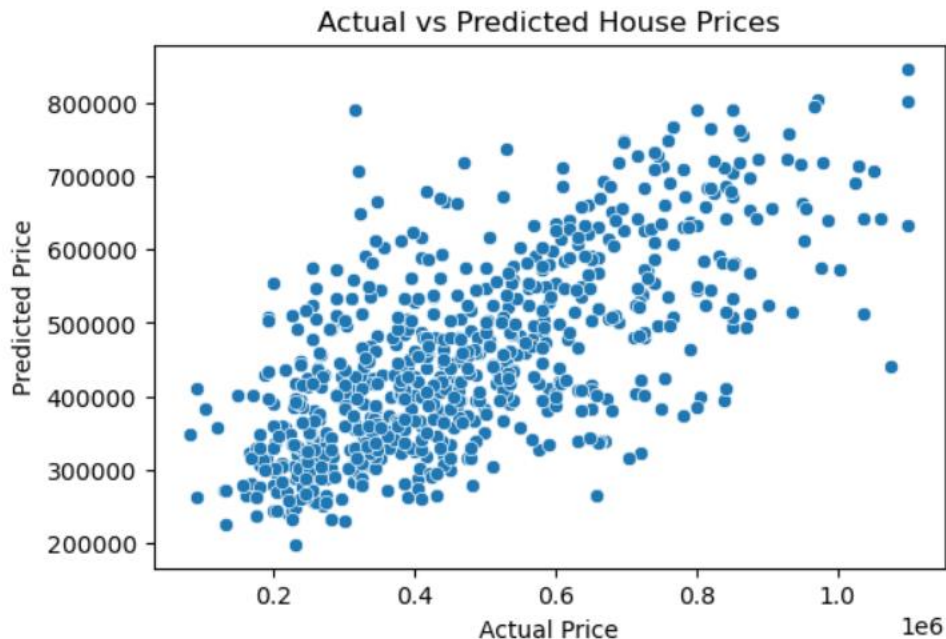
Metrics & Evaluation:

In this model, the metrics used to evaluate the KNN model are R-squared and Mean Squared Error (MSE).

- ✓ The R- Squared value is 0.454, which says that there is less than half of the variance in the data that can be explained by the model.
- ✓ The MSE is 23716319555.336937 which is very high and less accurate on average.
- ✓ These results are less because of complexity of the dataset and house prices which are varying in millions.
- ✓ We have training and testing error as 0.485 and 0.454, respectively.

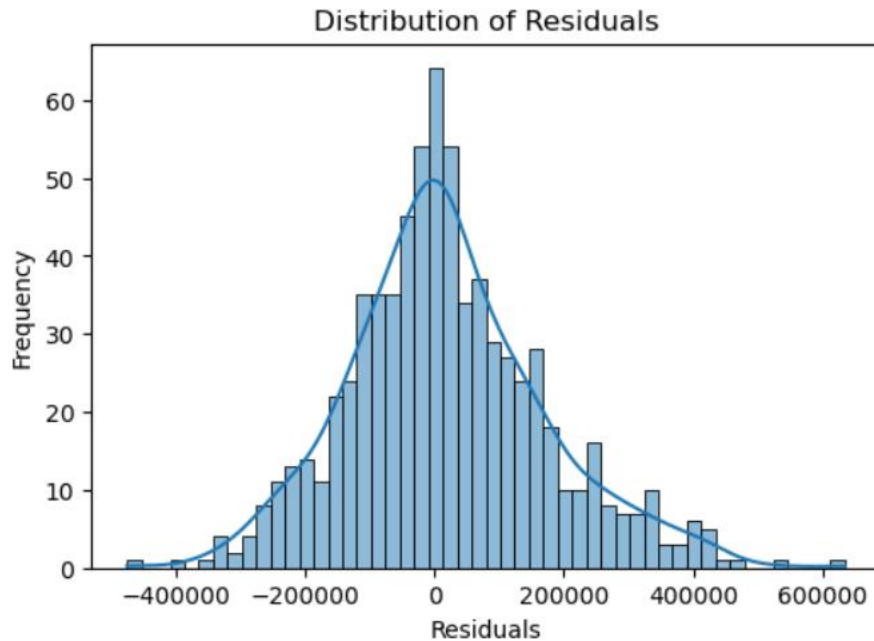
Visualization:

- ✓ The scatter plot of actual vs predicted house prices shows a weaker positive correlation, but there is still a lot of variances because of higher valued house prices.
- ✓ From the plot, we can say that the model could not be able to capture the price is not up to the mark and not obtaining all the complexities of the data.



- ✓ The histogram of residuals which is the difference between the actual and predicted values indicates a normal distribution, but still there could be presence of bias.
- ✓ In the graph below, you can see that there is long tail on both sides indicating that some predictions are quite far.

- ✓ This indicates that there are nonlinear relationships between features and the target variable that the model is not obtaining.



Support Vector Regressor:

3. Support Vector Regressor

```
In [361]: from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=12)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model_3 = SVR(kernel='linear')

model_3.fit(X_train_scaled, y_train)

y_pred = model_3.predict(X_test_scaled)
mse_svm = mean_squared_error(y_test, y_pred)

r2_svm = r2_score(y_test, y_pred)
print(f"R Squared Error: {r2_svm}")
print("Mean Squared Error:", mse_svm)

R Squared Error: 0.031146513199410086
Mean Squared Error: 37846328236.62354
```

- ✓ SVM is another important supervised algo which is used for both regression and classification tasks.
- ✓ In the context, we have chosen this model because of its effectiveness in handling both linear and non- linear datasets due to its kernel functions.
- ✓ This can also work for higher dimensional space capturing complex relationships.
- ✓ Hence, we have used SVM model in our project.

Code Explanation:

- ✓ In the code part, you can see that we have first scaled our dataset using 'StandardScaler'.
- ✓ We have standardized the features to have a mean of zero and standard deviation of one.
- ✓ We have also done hyper parameter tuning in which we have used some techniques like GridSearch and Cross Validation to get best fitting kernel and random_state.
- ✓ We also checked with kernel states like 'poly', 'sigmoid', 'linear', 'rbf' and random_state ranging from 1 to 10000 and ensured the above values are best fit to the model.

- ✓ We found that kernel = 'linear' and random_state = 12 is giving at least a positive value remaining all are indicating negative values. Hence, we have chosen this as best value
- ✓ To achieve this, we have used common techniques like GridSearch and Cross-Validation.

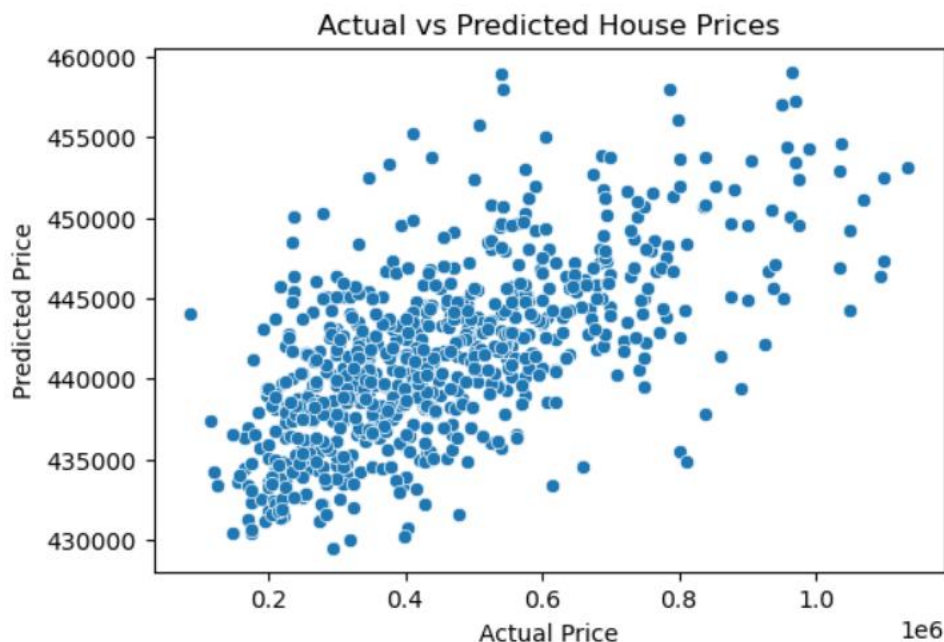
Metrics & Evaluation:

In this model, the metrics used to evaluate the SVR model are R-squared and Mean Squared Error (MSE).

- ✓ The R- Squared value is 0.031, which says that there is high variance in the data that can be explained by the model.
- ✓ The MSE is 37846328236.6235 which is very high and very less accurate on average.
- ✓ By this we can say that SVR not be suitable model for predicting the house prices project.
- ✓ These results are less because of complexity of the dataset and house prices which are varying in millions.
- ✓ We have training and testing error as 0.006 and 0.003, respectively.

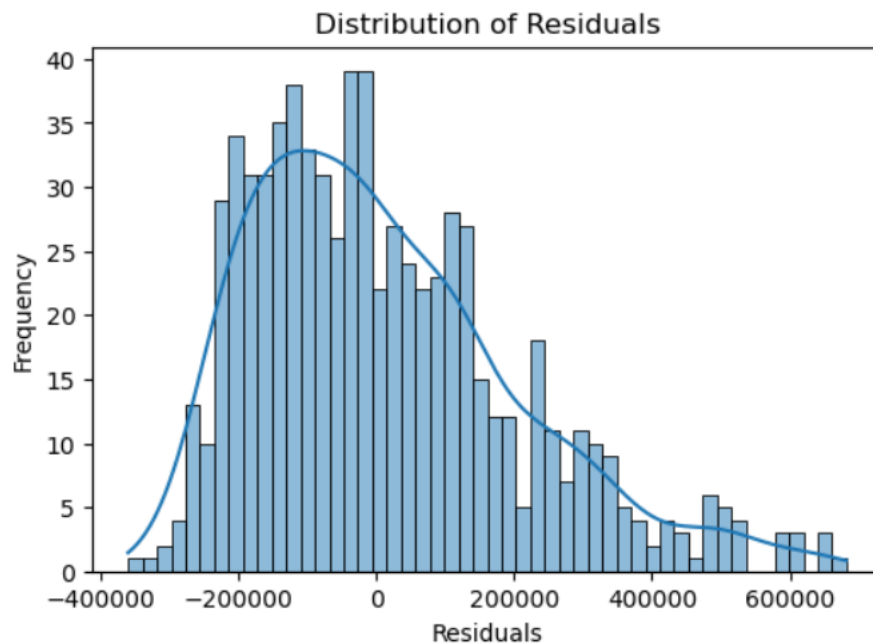
Visualization:

- ✓ The scatter plot of actual vs predicted house prices shows a weaker correlation, but there is still a lot of variances because of higher valued house prices.
- ✓ From the plot, we can say that the SVR model is not suitable to predict house prices.
- ✓ Real State applications are often hard to predict.
- ✓ Here you can see in the graph that there is a lot of spread presents.



- ✓ The histogram of residuals which is the difference between the actual and predicted values indicates a probability distribution, but still there could be presence of bias.

- ✓ In the graph below, you can see that there is long tail on both sides indicating that some predictions are quite far.
- ✓ This indicates that there are nonlinear relationships between features and the target variable that the model is not obtaining.
- ✓ To implement this project, we might need to design most effective other models rather than this.



XGBOOST:

4. XG BOOST

```
In [366]: import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from xgboost import XGBRegressor

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model_4 = XGBRegressor(objective='reg:squarederror', random_state=42)

model_4.fit(X_train, y_train)

y_pred = model_4.predict(X_test)

r2_xgb = r2_score(y_test, y_pred)
print(f"R Squared Error: {r2_xgb}")

mse_xgb = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse_xgb}")

R Squared Error: 0.5235419755270061
Mean Squared Error: 20020951047.312397
```

- ✓ XGBoost stands for Extreme Gradient Boosting. It has ability to capture complex nonlinear relationships in the data.
- ✓ In our context, Housing Price Prediction, where numerous features can interact in complex, non-linear ways to influence the house price.
- ✓ Hence, we have chosen this model for our regression task.

Code Explanation:

- ✓ We have done hyper parameter tuning in which we have used some techniques like GridSearch and Cross Validation to get best fitting max_depth, eta, lambda, alpha and random_state.
- ✓ We have checked random_state ranging from 1 to 10000 and ensured the above values are best fit to the model.

- ✓ We found that our model is look like overfitting. We found that objective = 'reg: squarederror' and random_state = 42 is giving as best compared with remaining all are indicating negative values.

Metrics & Evaluation:

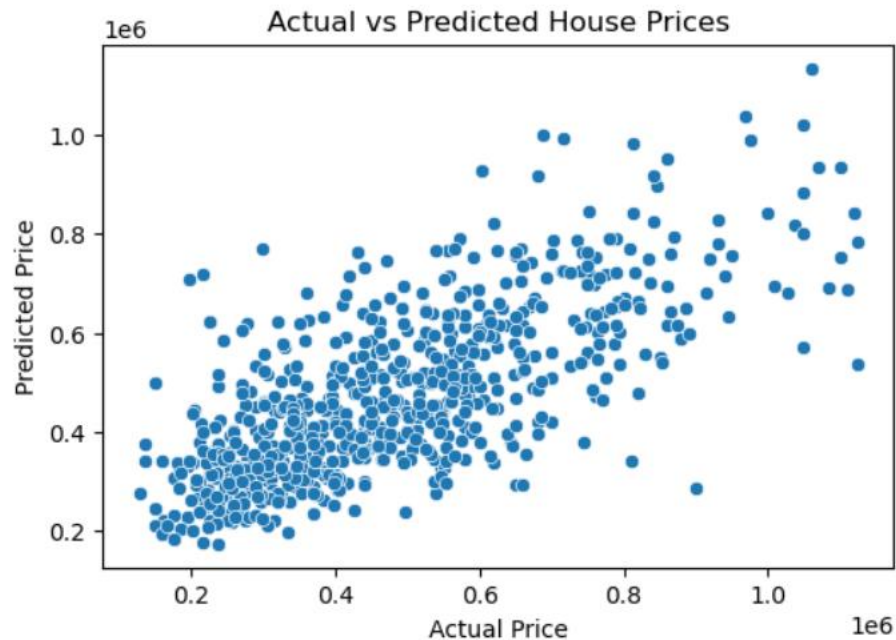
In this model, the metrics used to evaluate the XGB model are R-squared and Mean Squared Error (MSE).

- ✓ The R- Squared value is 0.523, which says that there is 52% variance in the data explained by the model.
- ✓ The MSE is 20020951047.312397 which is moderate and very accurate on average.
- ✓ Based on these results, we can say that our model is overfitting because we have high r squared value for training and less for testing.
- ✓ These results are little less because of complexity of the dataset and house prices which are varying in millions.
- ✓ We have training and testing error as 0.933 and 0.52, respectively.

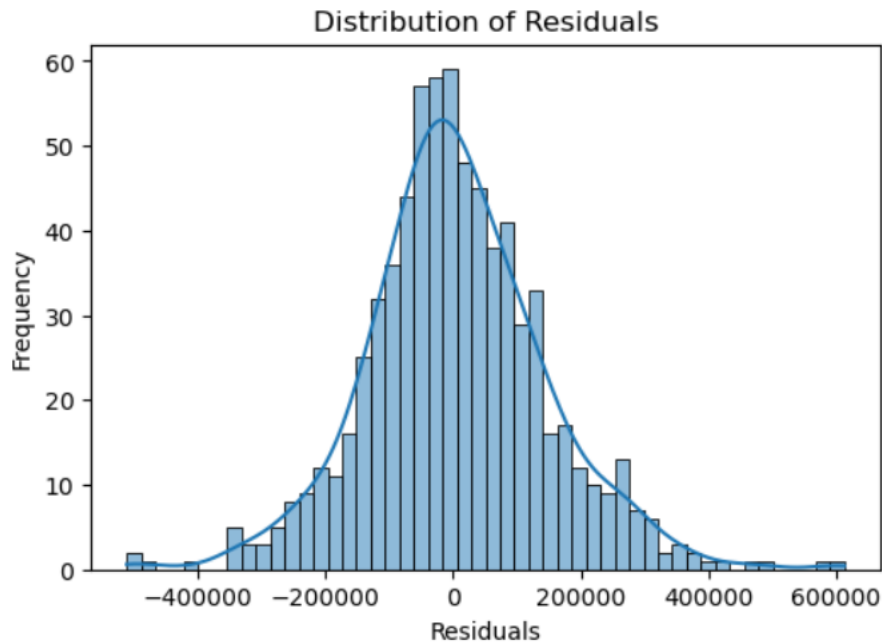
Visualization:

- ✓ The scatter plot of actual vs predicted house prices shows points figured along the diagonal line showing predictions are accurate.

- ✓ From the plot, we can say that effectiveness of the model improved understanding underlying patterns.
- ✓ Here you can see in the graph that there is a lot of spread presents.



- ✓ The histogram of residuals which is the difference between the actual and predicted values indicates a normal distribution having unbiased predictions.
- ✓ In the graph below, you can see that there is long tail which says that model is underpredicting some segments of the data.



Decision Tree Regressor:

5. Decision Tree Regressor

```
In [371]: from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
import numpy as np
import pandas as pd

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=535)
#535
params = {'max_depth': 5, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 15}

model_5 = DecisionTreeRegressor(**params, random_state=42)

model_5.fit(X_train, y_train)

y_pred = model_5.predict(X_test)
mse_dt = mean_squared_error(y_test, y_pred)

r2_dt = r2_score(y_test, y_pred)
print("R-squared Error:", r2_dt)
print("Mean Squared Error:", mse_dt)
```

R-squared Error: 0.5307791835610987
Mean Squared Error: 22622965592.472996

- ✓ Decision Tree is nonparametric model and versatile machine learning algorithm which is used for both classification and regression tasks.

- ✓ In this context, we have used this algo because of its ability to represent both linear and nonlinear relationships. They can learn complex datasets by capturing interactions effectively.
- ✓ Since Price can be influenced by different features present in the dataset. This could be better algo to use.
- ✓ Hence, we have used Decision Tree for this regression task.

Code Explanation:

- ✓ We have done hyper parameter tuning in which we have used some techniques like GridSearch and Cross Validation to get best fitting max_depth, max_features, min_samples_leaf, min_samples_split and random_state.
- ✓ We have checked random_state ranging from 1 to 10000 and ensured the above values are best fit to the model.
- ✓ We found that our model is looks like accurate. We found that 'max_depth': 5, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 15 and random_state = 535 is giving as best compared with remaining all are indicating negative values or less values.

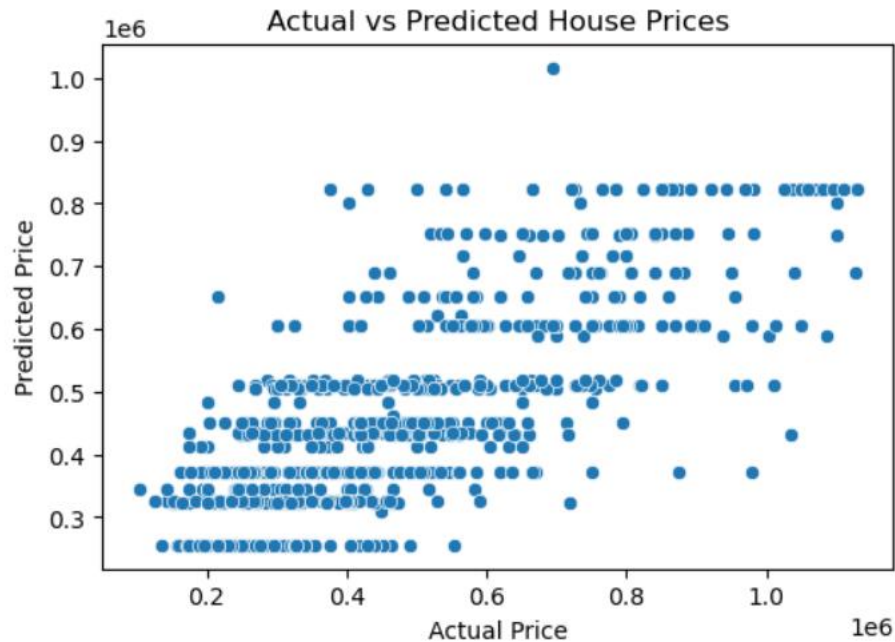
Metrics & Evaluation:

In this model, the metrics used to evaluate the Decision Tree model are R-squared and Mean Squared Error (MSE).

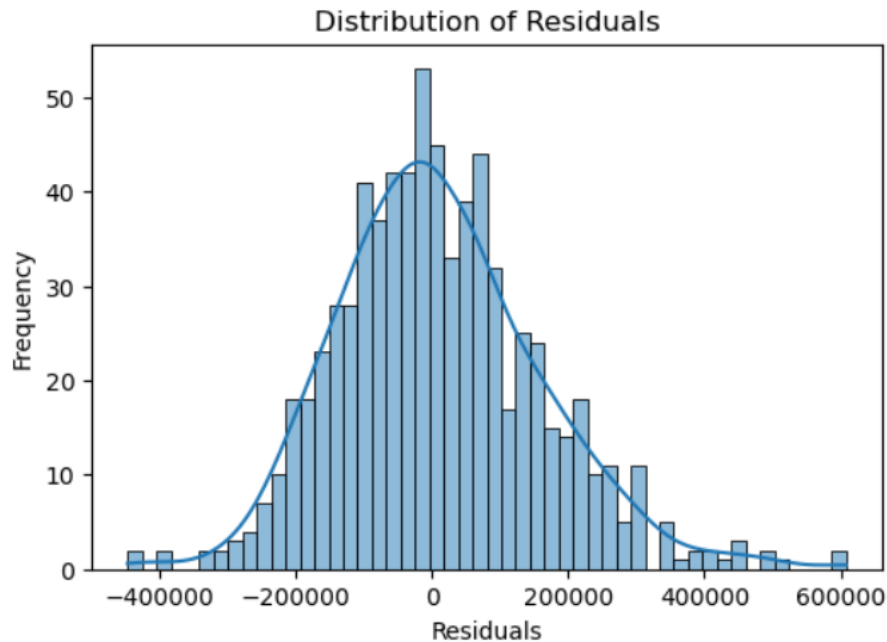
- ✓ The R- Squared value is 0.53, which says that there is 53% variance in the data explained by the model.
- ✓ The MSE is 22622965592.472996 which is moderate and very accurate on average.
- ✓ Based on these results, we can say that our model is unbiased as training and testing scores are almost same.
- ✓ These results are little less because of complexity of the dataset and house prices which are varying in millions.
- ✓ We have training and testing error as 0.483 and 0.53, respectively.

Visualization:

- ✓ The scatter plot of actual vs predicted house prices shows points figured along the diagonal line showing predictions are closely to actual values.
- ✓ From the plot, we can say that effectiveness of the model improved understanding underlying patterns, indicating variance in prediction accuracy.
- ✓ Here you can see in the graph that there is a lot of spread but there are some missing relationships in the data.



- ✓ The histogram of residuals which is the difference between the actual and predicted values indicates a normal distribution having unbiased predictions but can see a presence of skewness.
- ✓ In the graph below, you can see that there is long tail which says that model is overestimating or underestimating the house prices in certain cases.



Random Forest Regressor:

6. Random Forest Regressor

```
In [376]: import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model_6 = RandomForestRegressor(n_estimators=150, max_depth=20, min_samples_split=10, min_samples_leaf=1, random_state=42)

model_6.fit(X_train, y_train)

y_pred = model_6.predict(X_test)

r2_rf = r2_score(y_test, y_pred)
mse_rf = mean_squared_error(y_test, y_pred)

print("R-squared Error:", r2_rf)
print("Mean Squared Error:", mse_rf)

R-squared Error: 0.5290088716360211
Mean Squared Error: 19791229951.733418
```

- ✓ Random Forest is ensemble machine learning method which operates by constructing multiple decision trees and average them during prediction.
- ✓ This is also used for both regression and classification tasks.

- ✓ In our context, it can provide good score for each feature and uses most important features for prediction.
- ✓ Hence, we have used Random Forest Model for this regression task.

Code Explanation:

- ✓ We have done hyper parameter tuning in which we have used some techniques like GridSearch and Cross Validation to get best fitting max_depth, max_features, min_samples_leaf, min_samples_split, n_estimators and random_state.
- ✓ We have checked random_state ranging from 1 to 10000 and ensured the above values are best fit to the model.
- ✓ We found that our model is looks like overfitting. We found that 'max_depth': 20, 'n_estimators': 150, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 10 and random_state = 42 is giving as best compared with remaining all are indicating negative values or less values.

Metrics & Evaluation:

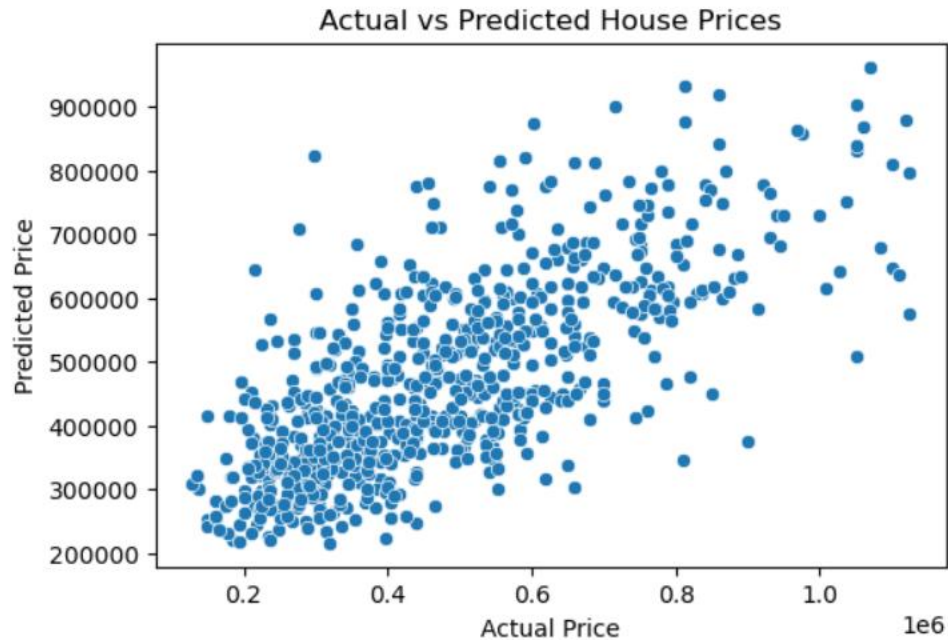
In this model, the metrics used to evaluate the Random Forest model are R-squared and Mean Squared Error (MSE).

- ✓ The R- Squared value is 0.53, which says that there is 53% variance in the data explained by the model.

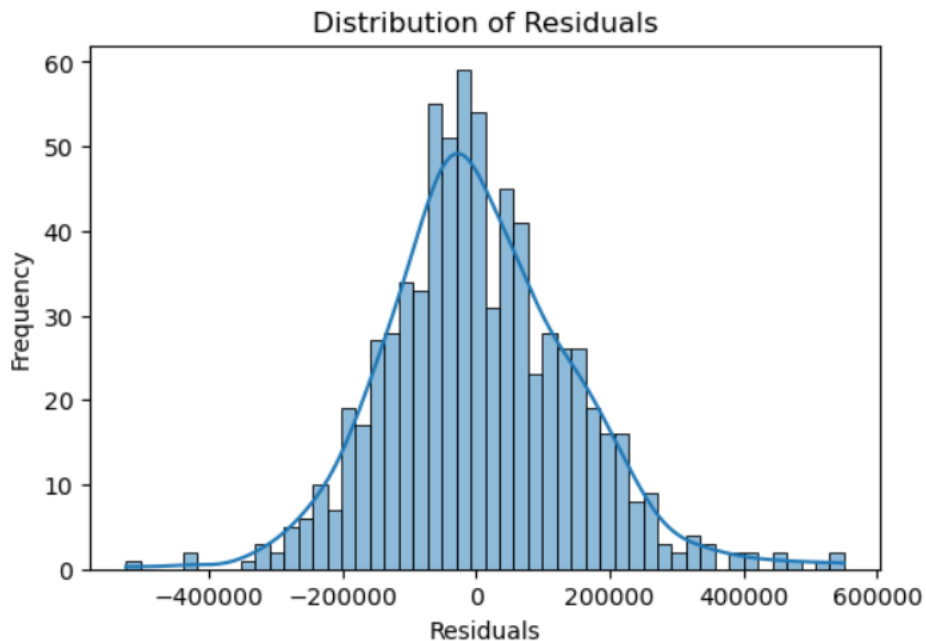
- ✓ The MSE is 19791229951.733418 which is moderate and very accurate on average.
- ✓ Based on these results, we can say that our model is little overfitting as training scores is more than testing scores.
- ✓ These results are little less because of complexity of the dataset and house prices which are varying in millions.
- ✓ We have training and testing error as 0.857 and 0.53, respectively.

Visualization:

- ✓ The scatter plot of actual vs predicted house prices shows points figured along the diagonal line showing predictions are accurate. From the plot, we can say that effectiveness of the model improved understanding underlying patterns.
- ✓ Here you can see in the graph that there is a lot of spread suggesting variability in the model's predictions across the price range



- ✓ The histogram of residuals which is the difference between the actual and predicted values indicates a normal distribution having unbiased predictions but can see a presence of skewness.
- ✓ In the graph below, you can see that there is central peak but also shows a tail, indicating presence of larger errors in some results.



Gradient Boosting Algorithm:

7.Gradient Boosting Regression

```
In [381]: from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model_7 = GradientBoostingRegressor(learning_rate=0.01, max_depth=3, n_estimators=150)
model_7.fit(X_train, y_train)

y_pred = model_7.predict(X_test)

mse_gbr = mean_squared_error(y_test, y_pred)
r2_gbr = r2_score(y_test, y_pred)

print("R Squared Error:", r2_gbr)
print("Mean Squared Error:", mse_gbr)
```

```
R Squared Error: 0.4068694774098146
Mean Squared Error: 24923574685.471603
```

- ✓ Gradient Boosting Regressor is an advanced machine learning predictive modeling algo which can be used for regression tasks due to its robustness and accuracy.

- ✓ It allows for optimization of arbitrary differential loss functions, including nonlinear problems.
- ✓ In our context, it can model some patterns in data by building trees one at a time and rectify errors one by one. Finally uses the designed model for prediction.
- ✓ Hence, we have used this GBR for this regression task.

Code Explanation:

- ✓ We have done hyper parameter tuning in which we have used some techniques like GridSearch and Cross Validation to get best fitting learning rate, max_depth, n_estimators and random_state.
- ✓ We have checked random_state ranging from 1 to 10000 and ensured the above values are best fit to the model.
- ✓ We found that our model is looks like overfitting. We found that 'max_depth': 3, 'n_estimators': 150 and random_state = 42, learning_rate = 0.01 is giving as best compared with remaining all are indicating negative values or less values.

Metrics & Evaluation:

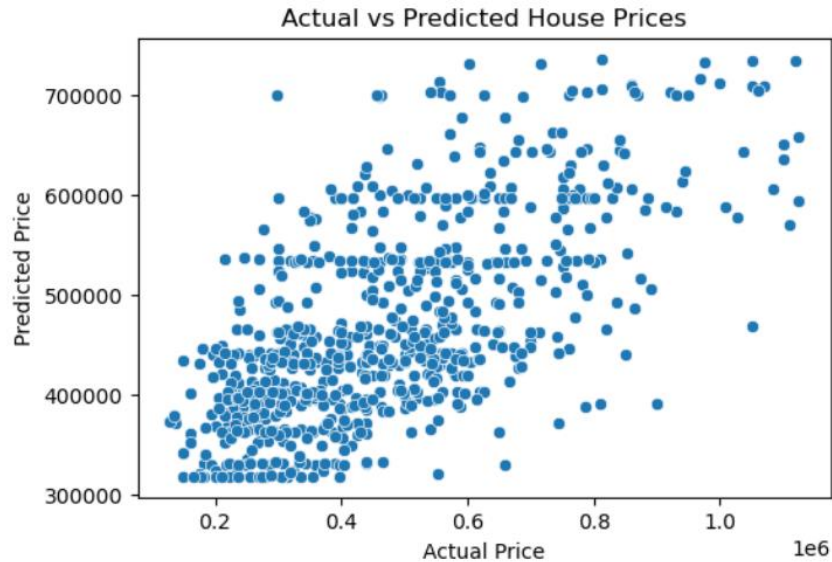
In this model, the metrics used to evaluate the GBR model are R-squared and Mean Squared Error (MSE).

- ✓ The R- Squared value is 0.40, which says that there is 40% variance in the data explained by the model.

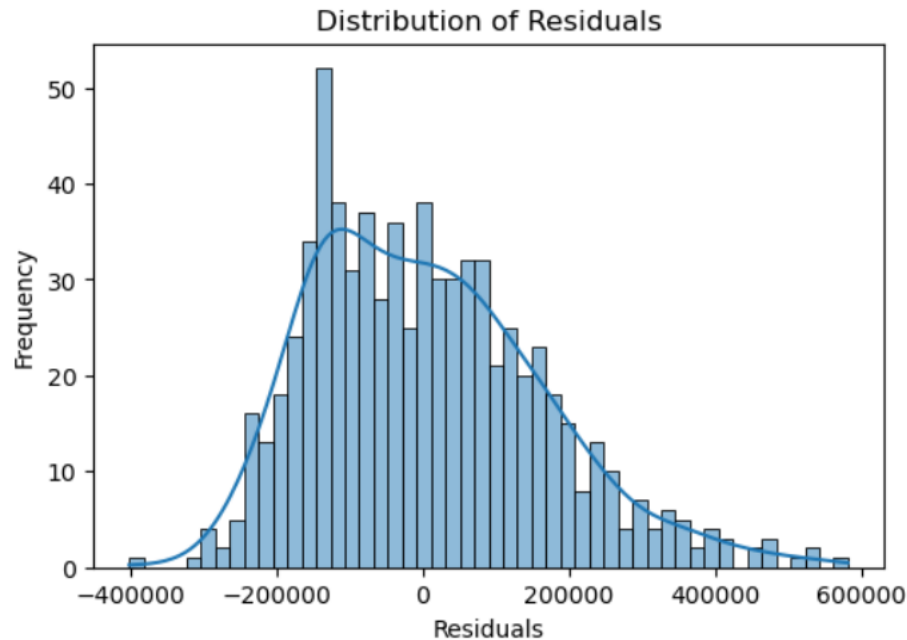
- ✓ The MSE is 24923574685.471603 which is very high and less accurate on average.
- ✓ Based on these results, we can say that our model is not fit for this house price prediction project.
- ✓ These results are little less because of complexity of the dataset and house prices which are varying in millions.
- ✓ We have training and testing error as 0.466 and 0.406, respectively.

Visualization:

- ✓ The scatter plot of actual vs predicted house prices shows a weaker positive correlation, but there is still a lot of variances because of higher valued house prices.
- ✓ From the plot, we can say that the model could not be able to capture the price is not up to the mark and there is lot of dispersion of points away from the line of perfect prediction.



- ✓ The histogram of residuals which is the difference between the actual and predicted values indicates a normal distribution, but still there seems to be skewed with several outliers indicates the model's predictions are biased for certain segments of the house prices.
- ✓ In the graph below, you can see that there is long tail on right sides indicating that some predictions are quite far.



K Means:

K Means

```
In [523]: import pandas as pd
from sklearn.cluster import KMeans
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns

city_data = hp['CITY'].str.get_dummies()
kmeans = KMeans(n_clusters=10, random_state=0).fit(city_data)
hp['city_cluster'] = kmeans.labels_

model = LinearRegression()
model.fit(X, y)

y_pred = model.predict(X)

r2_kmeans = r2_score(y, y_pred)
mse_kmeans = mean_squared_error(y, y_pred)

print("R-squared:", r2_kmeans)
print("Mean Squared Error:", mse_kmeans)
```

C:\Users\Satvik Jonnalagadda\anaconda3\Lib\site-packages\sklearn\cluster_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
super()._check_params_vs_input(X, default_n_init=10)

R-squared: 0.5674559405881774
Mean Squared Error: 18931466806.563053

- ✓ K Means is an unsupervised clustering algorithm which could be used for regression tasks by means of some changes made in the model.
- ✓ K Means finds patterns and structure in data that isn't explicitly labeled and partition data into 'k' clusters.
- ✓ In our context, we will use it to cluster groupings through 'CITY' as feature. Later based on the groupings done based on analysis of underlying patterns and data.
- ✓ Hence, we use this model for this regression task.

Code Explanation:

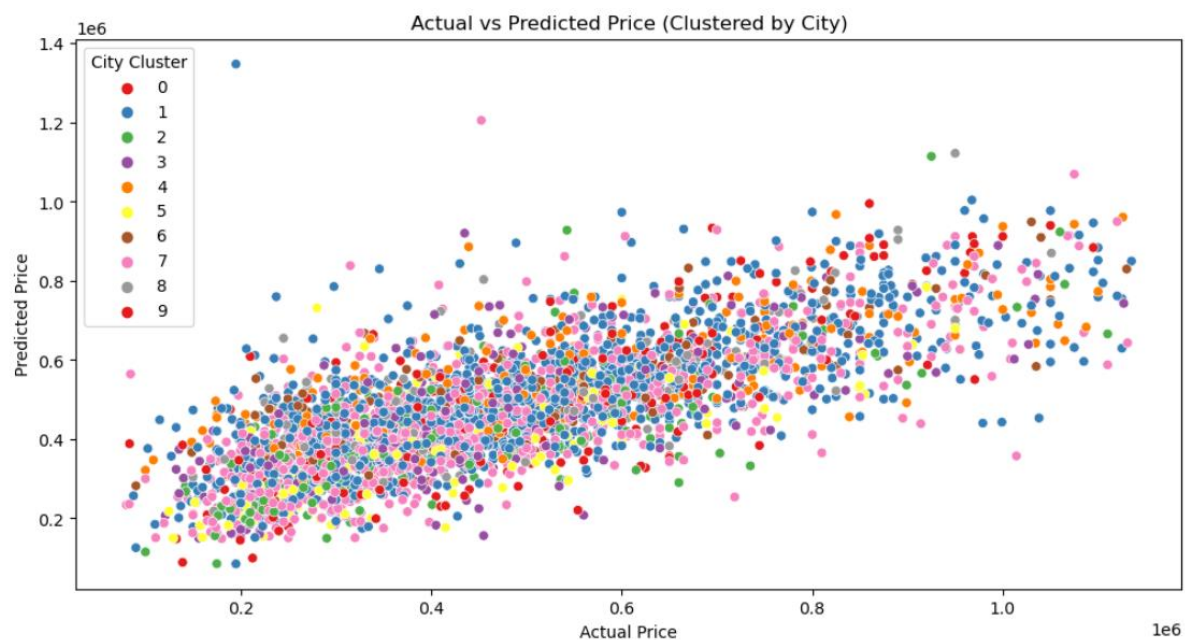
- ✓ We have also chosen n-clusters = 10 and random_state = 0 which indicates there could be 10 clustered data points in the feature space when algorithm makes a prediction.
- ✓ We have achieved this state and n-clusters which could be best fitting value by model tuning the parameters.
- ✓ We also checked with n-clusters ranging from 1 to 100 and random_state ranging from 1 to 10000 and ensured the above values are best fit to the model.
- ✓ To achieve this, we have used common techniques like GridSearch and Cross-Validation.

Metrics & Evaluation:

In this model, the metrics used to evaluate the K Means model are R-squared and Mean Squared Error (MSE).

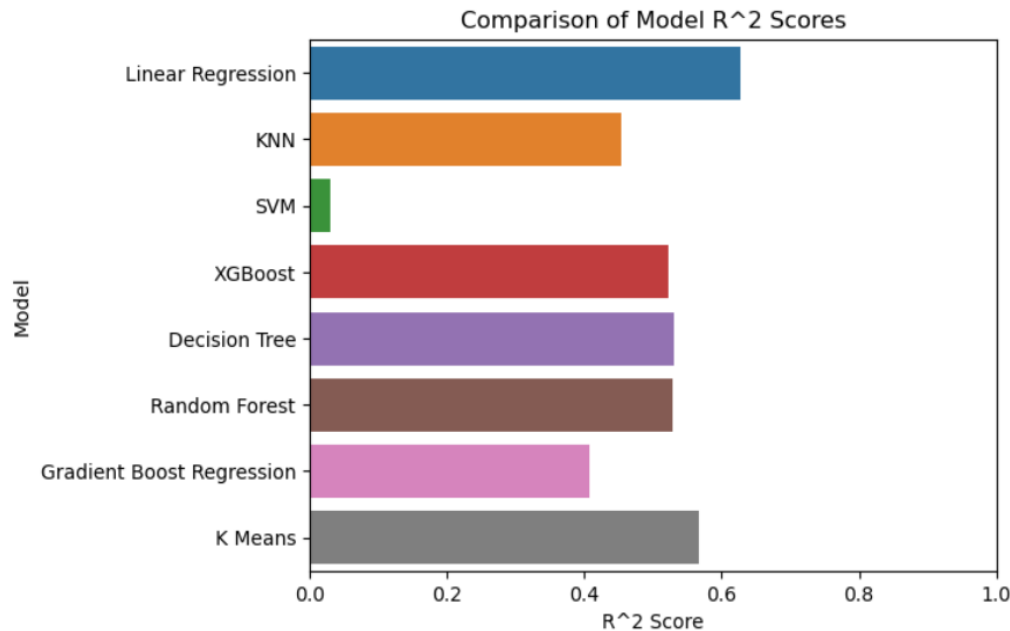
- ✓ The R- Squared value is 0.57, which says that there is 57% variance in the data explained by the model.
- ✓ The MSE is 18931466806.563053 which is moderate and good accurate on average.
- ✓ Based on these results, we can say that our model is moderately accurate for this house price prediction project.
- ✓ These results are little less because of complexity of the dataset and house prices which are varying in millions.

Visualization:

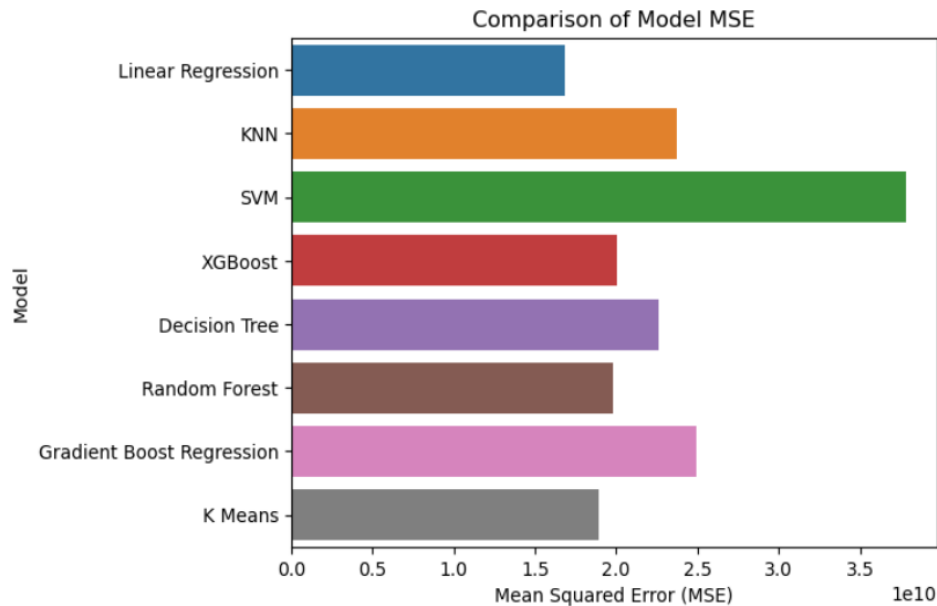


This graph is scatter plot that compares actual vs predicted house prices with the data points colored according to the city clusters determined by K Means Clustering Model.

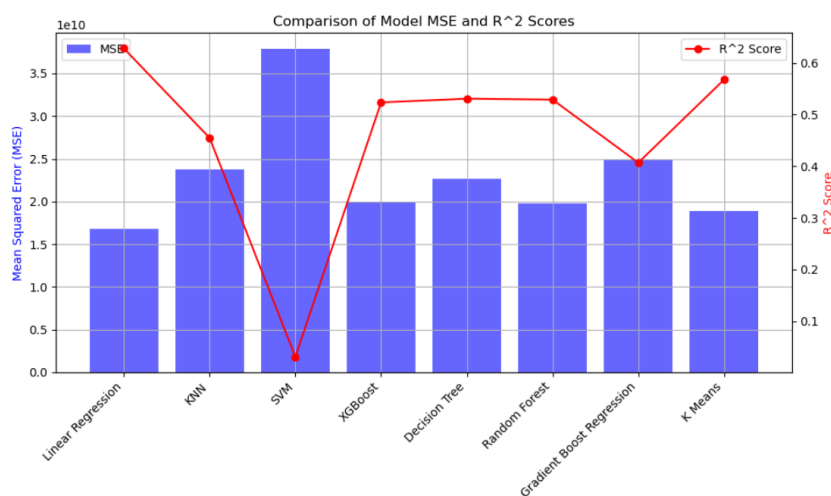
Visualization Across Models:



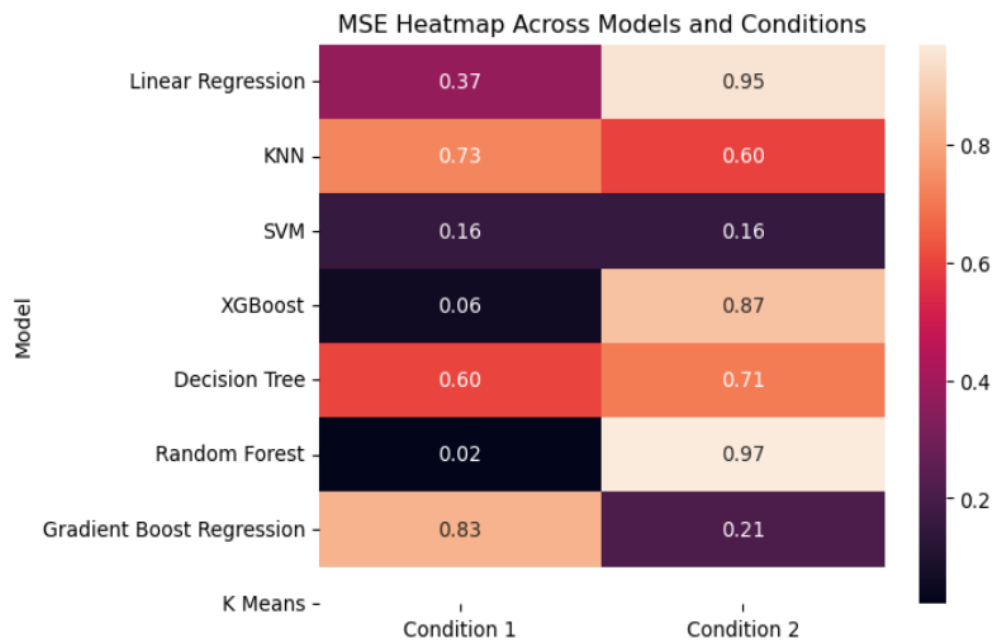
- ✓ The above bar chart compares the R-squared scores of different machine learning models used for predicting house prices.
- ✓ R squared is a statistical measure that represents the proportion of the variance for the dependent variable shown by independent variables in a regression model.
- ✓ Here, in the above graph you can see that 'Linear Regression' has highest R-squared value and SVR has least R-Squared value.
- ✓ From this we can clearly say that SVM failed to predict the price and Linear Regression is very accurate among all the models.



- ✓ The above bar chart compares the Mean Squared Error of different machine learning models used for predicting prices.
- ✓ MSE is a measure of the quality of an estimator. It is always non-negative, values close to zero are better.
- ✓ Here, in the above graph you can see that 'Linear Regression' has lowest MSE value and SVR has highest MSE value.



- ✓ The above figure represents a dual axis bar and line chart, where bar graph represents the MSE of different Machine learning models and line graph represents the R-Squared Scores for the same models.



Why this model?

In Phase 3 of our project, we have used Linear Regression Model to enhance the functionality of our website by predicting the price.

Linear Regression Model:

- ✓ This model clearly handling the data of our project and accurately predicting the house prices.
- ✓ Predicting house prices, which are continuous and like have a linear relationship can be best executed using this model.

In our phase 2 of our project, we have implemented 6 different machine learning models, among them this has been the best model that has less error in prediction.

- ✓ We initially wanted to use the basic linear regression as it is default selection, but we got more error.

```
name, price, length, year, type, location
In [90]: from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42, shuffle=True, stratify=None)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model_1 = LinearRegression()
model_1.fit(X_train_scaled, y_train)

y_pred = model_1.predict(X_test_scaled)

r2_lr = r2_score(y_test, y_pred)
mse_lr = mean_squared_error(y_test, y_pred)

print("R Squared Error:", r2_lr)
print("Mean Squared Error:", mse_lr)

R Squared Error: 0.44578449478221915
Mean Squared Error: 23288350557.00825
```

- ✓ So, we have decided to be tuning the parameters, we have used standard scalar first to standardize features to have mean of '0' and standard deviation of '1'.
- ✓ For tuning we have used GridSearch and Cross-Validation techniques and achieved a less error compared to basic model.
- ✓ We have used this model for further predicting house prices in our deployment.

```
In [80]: from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1688, shuffle=True, stratify=None)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model_1 = LinearRegression()
model_1.fit(X_train_scaled, y_train)

y_pred = model_1.predict(X_test_scaled)

r2_lr = r2_score(y_test, y_pred)
mse_lr = mean_squared_error(y_test, y_pred)

print("R Squared Error:", r2_lr)
print("Mean Squared Error:", mse_lr)
```

```
R Squared Error: 0.6280344569935457
Mean Squared Error: 16817657403.796228
```

Steps used to Integrate Linear Regression model into our product:

- ✓ **Model Loading:** We have loaded the pre-trained linear regression model, along with other necessary objects like scaler_objects and all models, using joblib library. This make sure that our model predicts as soon as values are entered in website.
- ✓ **Prediction:** we have used Linear Regression model in the predict function to predict the house price based on feature values. This prediction was made by analyzing the various features extracted from flask generated URL i.e., <https://127.0.0.1:5000>.
- ✓ We have also integrated Cleaning & EDA in our website so that user can explicitly see how dataset is cleaned and data analysis by him/herself.
- ✓ We have also shown how the features are related one another through 10 steps clearly by providing their images through coding part.
- ✓ **User Interface Integration:** we have used HTML CSS for front end development and using flask we have connected our back end.
- ✓ We have used java script for other implementations like event listeners like button clicking, rating etc.
- ✓ Finally, our linear regression model providing real time predictions on housing prices, which will be displayed in the same web page.

Folder Structure

phase3/

- |— *app.py*
- |— *clean_dataset.py*
- |— *cleaned_housing_dataset.csv*
- |— *demo video.mp4*
- |— *eda_step_1.py through eda_step_10.py*
- |— *linear_regression_model_2.pkl*
- |— *static/*
 - | |— *various eda_result*.png*
 - | |— *House Image.jpg*
 - | |— *index.css*
 - | |— *prediction.css*
 - | |— *script.js*
- |— *templates/*
 - | |— *cleaning_eda.html*
 - | |— *index.html*
 - | |— *prediction.html*
- |— *uploads/*
 - | |— *housedata.csv*

Instructions to deploy the product:

Web Application Code:

```
from flask import Flask, request, jsonify, render_template, redirect, url_for
```

```
import numpy as np
```

```
import joblib
```

```
import subprocess
```

```
import os
```

```
from nbconvert.preprocessors import ExecutePreprocessor
```

```
import nbformat
```

```
import webbrowser
```

```
app = Flask(__name__, template_folder='templates', static_folder='static')
```

```
# Load the trained model
```

```
model = joblib.load('linear_regression_model_2.pkl')
```

```
@app.route('/')
```

```
def index():
```

```
    return render_template('index.html', css_file='index.css')
```

```
@app.route('/prediction.html')
```

```
def prediction():
```

```
    return render_template('prediction.html', css_file='prediction.css')
```

```
@app.route('/cleaning_eda.html')
```

```
def clean_eda():
```

```
return render_template('cleaning_eda.html')
```

```
@app.route('/predict', methods=['POST'])
```

```
def predict_price():
```

```
    # Get the input data from the request
```

```
    input_data = request.get_json()
```

```
    print(input_data)
```

```
    try:
```

```
        # Calculate YEAR_BUILT
```

```
        year_built = 2024 - int(input_data['AGE_OF_HOUSE'])
```

```
        input_data['YEAR_BUILT'] = year_built
```

```
        # Calculate TOTAL_HOUSE_AREA_sqft
```

```
        total_house_area = int(input_data['BASEMENT_sqft']) +
```

```
int(input_data['ABOVE_BASEMENT_sqft'])
```

```
        input_data['TOTAL_HOUSE_AREA_sqft'] = total_house_area
```

```
        # Remove AGE_OF_HOUSE as it is no longer needed
```

```
        del input_data['AGE_OF_HOUSE']
```

```
        del input_data['CITY']
```

```
    print(input_data)
```

```
    attributes_order =
```

```
['NUM_OF_BEDROOMS', 'NUM_OF_BATHROOMS', 'LIVING_AREA_sqft',
```

```
'LOT_AREA_sqft', 'FLOORS', 'WATER_FRONT',
```

```
'ABOVE_BASEMENT_sqft', 'BASEMENT_sqft', 'VIEW_RATING',
```

```
'CONDITION_RATING', 'ZIP', 'YEAR_BUILT',  
'TOTAL_HOUSE_AREA_sqft']
```

```
# Create an array of values in the specified order  
input_values = [input_data[attr] if isinstance(input_data[attr],  
int) else int(input_data[attr]) for attr in attributes_order]
```

```
print(input_values)
```

```
# Convert input values to integers  
input_values = [int(value) for value in input_data.values()]  
print(input_values)
```

```
# Reshape input values  
X = np.array([input_values]).reshape(1, -1)  
print(X)
```

```
# Make a prediction using the model  
predicted_price = model.predict(X)[0]
```

```
print(predicted_price)
```

```
predicted_price = abs(predicted_price*0.0001)
```

```
predicted_price = round(predicted_price, 2)
```

```
# Return the predicted price as a JSON response  
return jsonify({'price': predicted_price})  
except (KeyError, ValueError):  
    return jsonify({'error': 'Invalid data format'}), 400
```



```
@app.route('/clean_dataset', methods=['POST'])
def clean_dataset():
    if 'file' not in request.files:
        return jsonify({'success': False, 'message': 'No file part'})

    file = request.files['file']

    if file.filename == '':
        return jsonify({'success': False, 'message': 'No selected file'})

    if file:
        # Save the uploaded file
        file_path = os.path.join('uploads', file.filename)
        file.save(file_path)

        # Execute the clean_dataset.py script with the file path as
        argument
        subprocess.call(['python', 'clean_dataset.py', file_path])

        return jsonify({'success': True, 'message': 'Dataset cleaned
        successfully!'})
    else:
        return jsonify({'success': False, 'message': 'Error cleaning
        dataset'})

@app.route('/perform_eda', methods=['POST'])
def perform_eda():
    data = request.json
    eda_step = data.get('edaStep')
```

```
print(f"Received EDA step: {eda_step}")
if eda_step == 'Step 1':
    try:
        # Call the eda_step_1.py script
        subprocess.call(['python', 'eda_step_1.py'])
        image_path = 'static/eda_result1.png' # Adjust the path
    accordingly
        return jsonify({'success': True, 'message': 'EDA Step 1 completed
successfully!', 'image_path': image_path})
    except Exception as e:
        return jsonify({'success': False, 'message': f'Error performing
EDA Step 1: {str(e)}'})
elif eda_step == 'Step 2':
    try:
        # Call the eda_step_2.py script
        subprocess.call(['python', 'eda_step_2.py'])
        image_path = 'static/eda_result2.png' # Adjust the path
    accordingly
        return jsonify({'success': True, 'message': 'EDA Step 2 completed
successfully!', 'image_path': image_path})
    except Exception as e:
        return jsonify({'success': False, 'message': f'Error performing
EDA Step 2: {str(e)}'})
elif eda_step == 'Step 3':
    try:
        # Call the eda_step_3.py script
        subprocess.call(['python', 'eda_step_3.py'])
        image_path = 'static/eda_result3.png' # Adjust the path
    accordingly
```

```
        return jsonify({'success': True, 'message': 'EDA Step 3 completed
successfully!', 'image_path': image_path})
    except Exception as e:
        return jsonify({'success': False, 'message': f'Error performing
EDA Step 3: {str(e)}'})
    elif eda_step == 'Step 4':
        try:
            # Call the eda_step_4.py script
            subprocess.call(['python', 'eda_step_4.py'])
            image_path = 'static/eda_result4.png' # Adjust the path
accordingly
            return jsonify({'success': True, 'message': 'EDA Step 4 completed
successfully!', 'image_path': image_path})
        except Exception as e:
            return jsonify({'success': False, 'message': f'Error performing
EDA Step 4: {str(e)}'})
    elif eda_step == 'Step 5':
        try:
            # Call the eda_step_5.py script
            subprocess.call(['python', 'eda_step_5.py'])
            image_path = 'static/eda_result5.png' # Adjust the path
accordingly
            return jsonify({'success': True, 'message': 'EDA Step 5 completed
successfully!', 'image_path': image_path})
        except Exception as e:
            return jsonify({'success': False, 'message': f'Error performing
EDA Step 5: {str(e)}'})
    elif eda_step == 'Step 6':
        try:
            # Call the eda_step_6.py script
```

```

        subprocess.call(['python', 'eda_step_6.py'])
        image_path = 'static/eda_result6.png' # Adjust the path
accordingly
        return jsonify({'success': True, 'message': 'EDA Step 6 completed
successfully!', 'image_path': image_path})
    except Exception as e:
        return jsonify({'success': False, 'message': f'Error performing
EDA Step 6: {str(e)}'})
    elif eda_step == 'Step 7':
        try:
            # Call the eda_step_7.py script
            subprocess.call(['python', 'eda_step_7.py'])
            image_path = 'static/eda_result7.png' # Adjust the path
accordingly
            return jsonify({'success': True, 'message': 'EDA Step 7 completed
successfully!', 'image_path': image_path})
        except Exception as e:
            return jsonify({'success': False, 'message': f'Error performing
EDA Step 7: {str(e)}'})
    elif eda_step == 'Step 8':
        try:
            # Call the eda_step_8.py script
            subprocess.call(['python', 'eda_step_8.py'])
            image_path = 'static/eda_result8.png' # Adjust the path
accordingly
            return jsonify({'success': True, 'message': 'EDA Step 8
completed successfully!', 'image_path': image_path})
        except Exception as e:
            return jsonify({'success': False, 'message': f'Error performing
EDA Step 8: {str(e)}'})

```

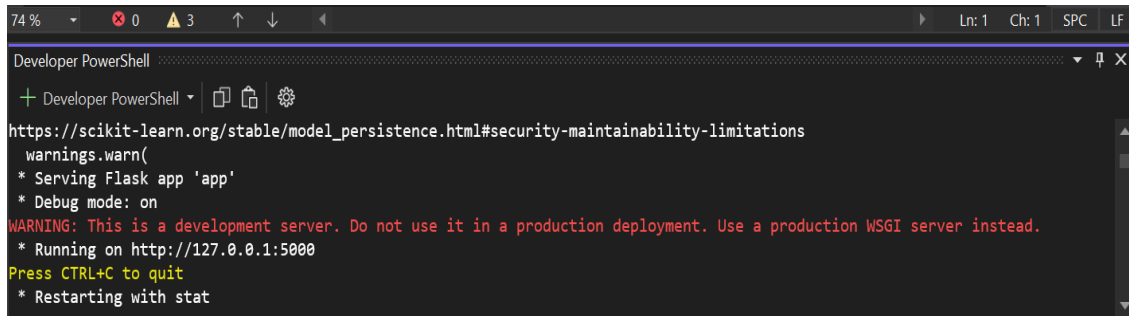
```

elif eda_step == 'Step 9':
    try:
        # Call the eda_step_9.py script
        subprocess.call(['python', 'eda_step_9.py'])
        image_path = 'static/eda_result9.png' # Adjust the path
        accordingly
        return jsonify({'success': True, 'message': 'EDA Step 9
completed successfully!', 'image_path': image_path})
    except Exception as e:
        return jsonify({'success': False, 'message': f'Error performing
EDA Step 9: {str(e)}'})
elif eda_step == 'Step 10':
    try:
        # Call the eda_step_10.py script
        subprocess.call(['python', 'eda_step_10.py'])
        image_path = 'static/eda_result10.png' # Adjust the path
        accordingly
        return jsonify({'success': True, 'message': 'EDA Step 10
completed successfully!', 'image_path': image_path})
    except Exception as e:
        return jsonify({'success': False, 'message': f'Error performing
EDA Step 10: {str(e)}'})
    else:
        return jsonify({'success': False, 'message': 'Invalid EDA step
selected!'})

if __name__ == '__main__':
    app.run(debug=True)

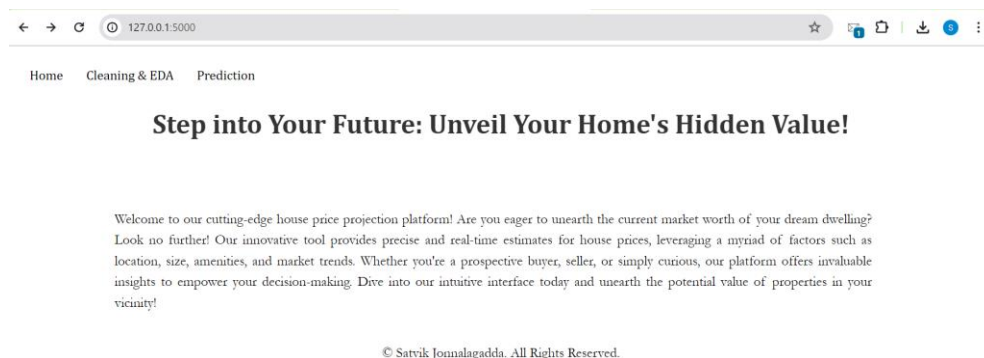
```

When we run ‘app.py’ in VSCode, you will get the below output that shows the URL to navigate to reach designated webpage.



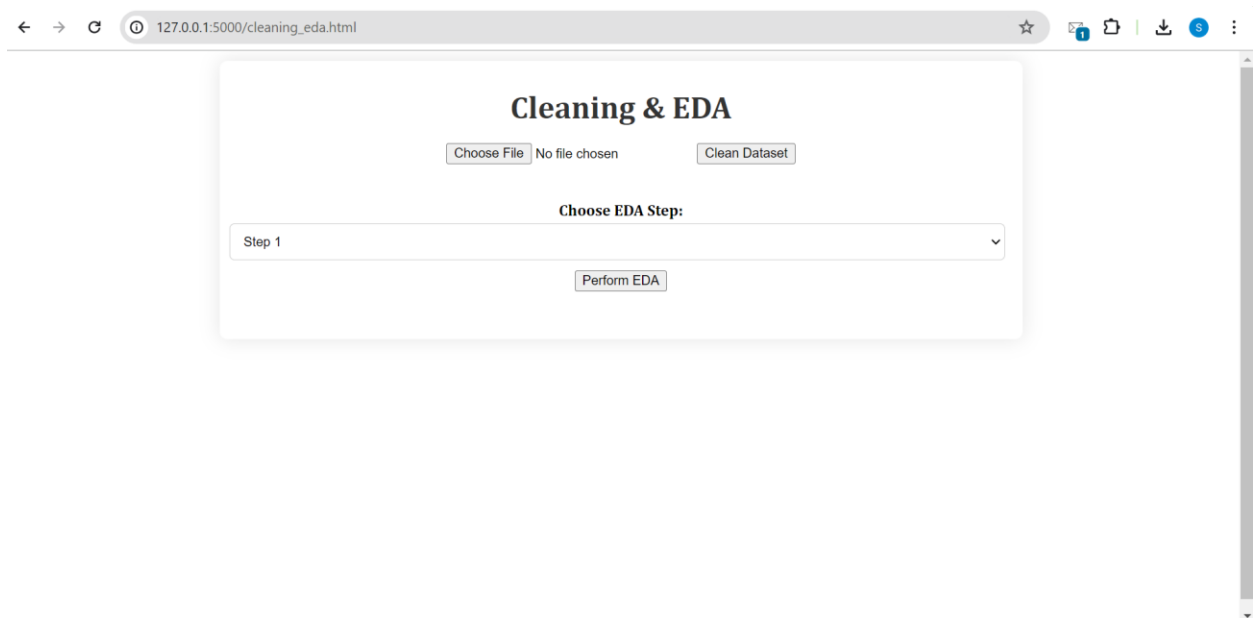
```
74 % 0 3 ↑ ↓ Ln: 1 Ch: 1 SPC LF
Developer PowerShell
+ Developer PowerShell
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
warnings.warn(
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
```

- ✓ When you go to the ‘https://127.0.0.1:5000’ in your favorite browser, this page will be shown.
- ✓ This is main page of our house price prediction project; you can see there are two buttons available at the left upper corner of the page which completely describes different phases of the project.

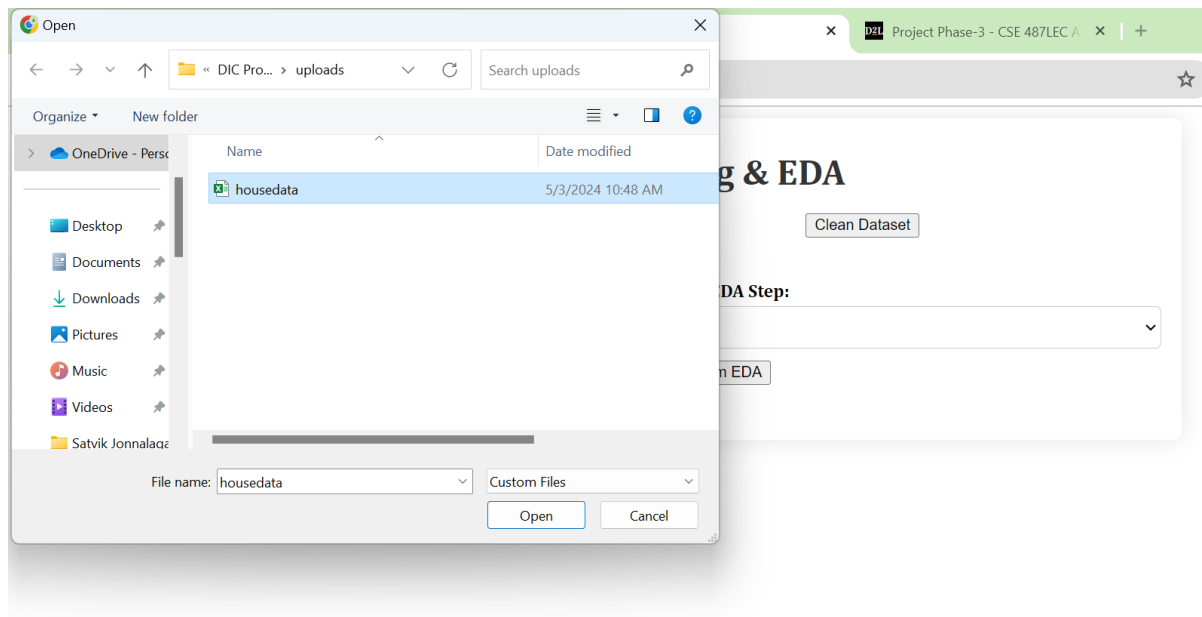


- ✓ In our phase 1, we have done data cleaning, data preprocessing and Data Analysis of our project.

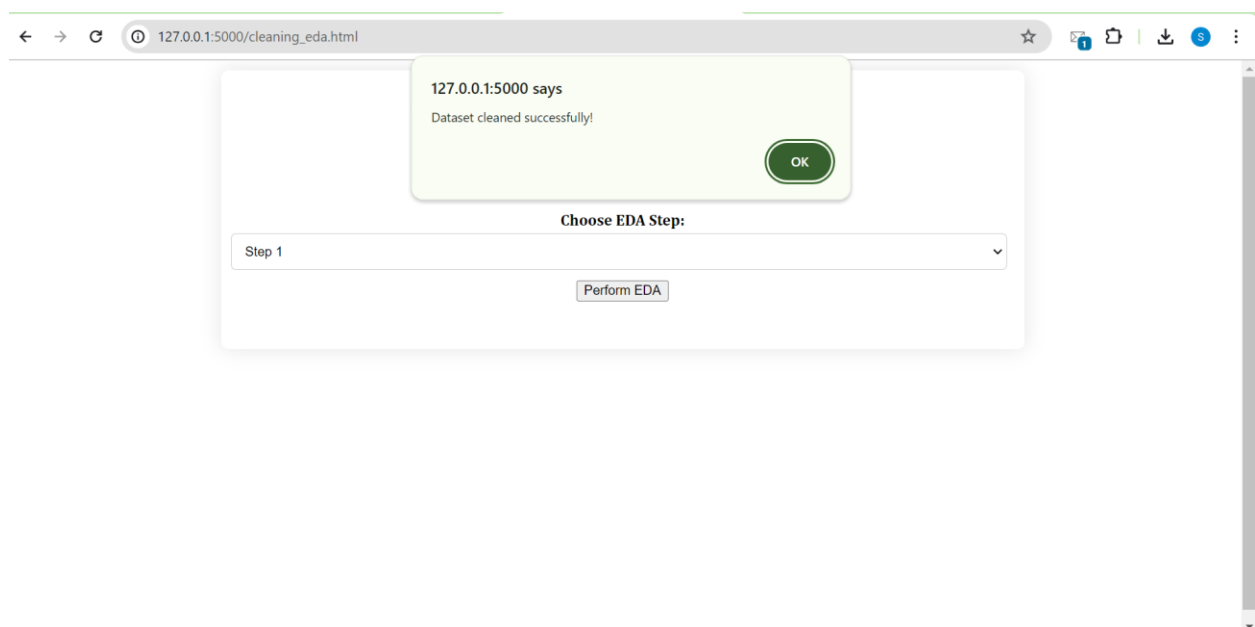
- ✓ When you click Cleaning& EDA button in main page, you will be navigated to this page shown below.



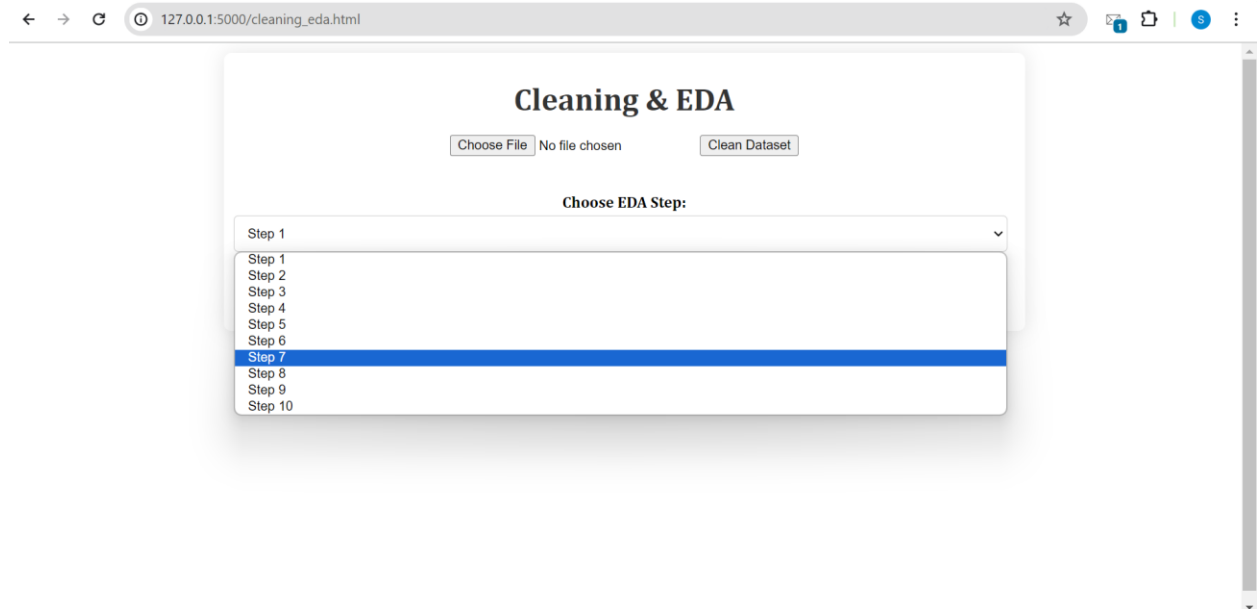
- ✓ Here, you can choose and upload a uncleaned dataset from 'uploads' folder in 'src' named 'housedata' and click 'Clean Dataset'.
- ✓ Below, you can see that I have selected 'housedata' dataset which is uncleaned from my local computer.



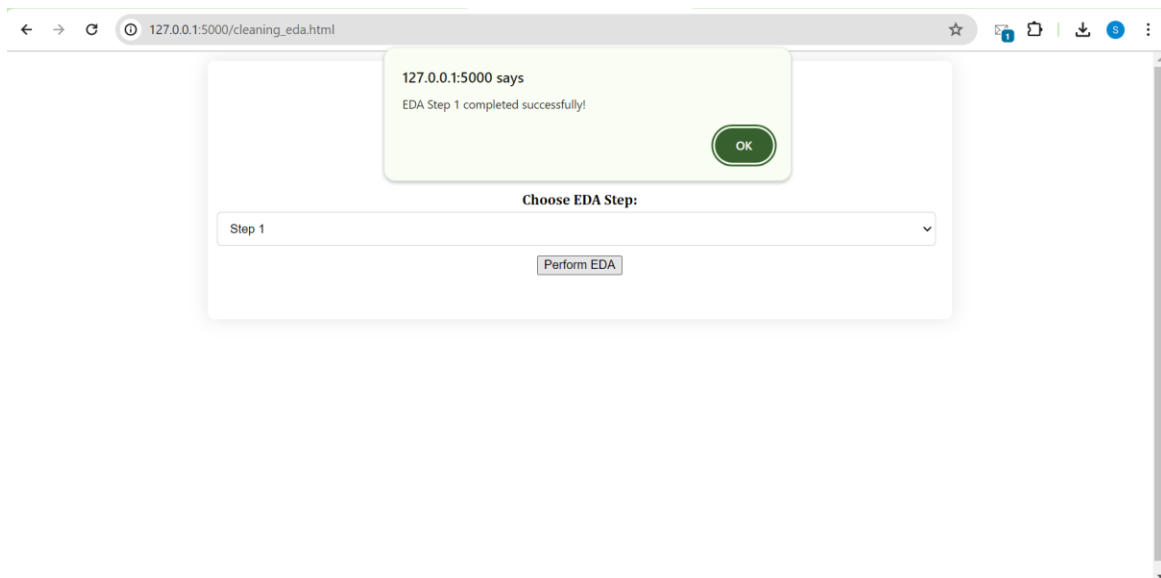
✓ Now, the dataset you have provided is cleaned successfully as you can see the alert message has been created in the below figure.



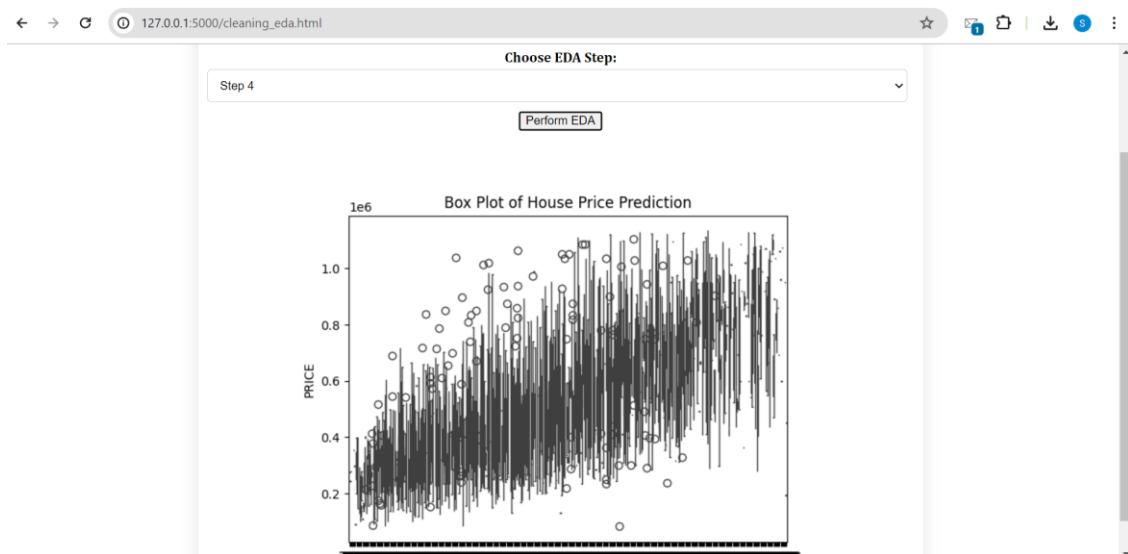
- ✓ Now you can do some data analysis on the cleaned dataset that has been generated.
- ✓ Here, I have performed 10 different EDA steps, you can choose any of them to get to know relationship between any attributes or how data is organized.



- ✓ I have chosen Step 1 for analysis on the cleaned dataset, when you click the button 'Perform EDA', the corresponding step gets called.
- ✓ After successful execution of the process, the alert box will be generated automatically shown below.

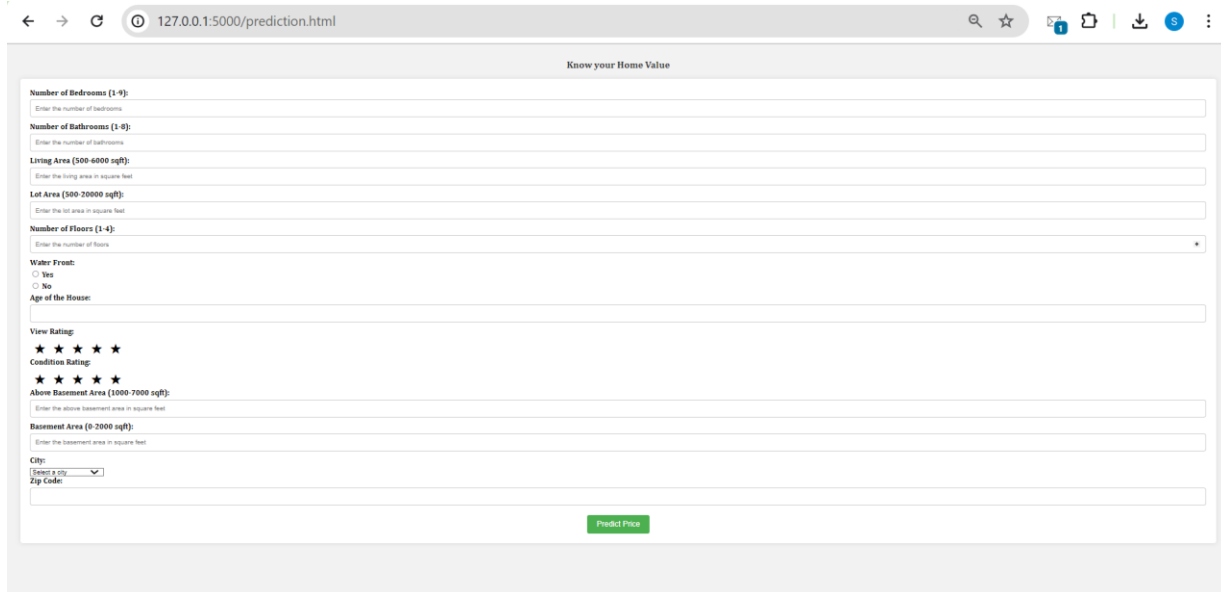


- ✓ Here, you can clearly see the box plot representation that describes the variation house price over time.



- ✓ Similarly, you can try all other EDA steps to know relationship between the attributes.
- ✓ This is how we cleaned, preprocessed, and analyzed our project in Phase 1.

- ✓ You can now get back to main page and click on ‘Prediction’ button, which will navigate you to the below page.



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/prediction.html'. The page title is 'Know your Home Value'. The form contains the following fields and options:

- Number of Bedrooms (1-9):** Input field.
- Number of Bathrooms (1-9):** Input field.
- Living Area (500-6000 sqft):** Input field.
- Lot Area (500-20000 sqft):** Input field.
- Number of Floors (1-4):** Input field.
- Water Front:** Radio buttons for 'Yes' and 'No'.
- Age of the House:** Input field.
- View Rating:** Five stars (all filled).
- Condition Rating:** Five stars (all filled).
- Above Basement Area (1000-7000 sqft):** Input field.
- Basement Area (0-2000 sqft):** Input field.
- City:** Dropdown menu with 'Upper City' selected.
- Zip Code:** Input field.

A green 'Predict Price' button is located at the bottom right of the form.

- ✓ Here, you can see that there is a form containing different attributes, you need to enter the values based on your interest but with constraints provided.
- ✓ In the below figure, you can see that I have given some values to each attribute.
- ✓ When you give all the values upon your interest and click ‘Predict’ button.
- ✓ The model in the backend will process the data and gives the predicted price as shown in the figure.
- ✓ You can try with different inputs every time and check how the price is changing while changing the values of different attributes like changing ‘city’, choosing ‘waterfront’ etc.

- ✓ By this, we have achieved our main motive of our project i.e., predicting the price which could help real estate developers etc. in real life situations.

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/prediction.html'. The page title is 'Know your Home Value'. The form contains the following fields and options:

- Number of Bedrooms (1-9): Input field with value 5.
- Number of Bathrooms (1-8): Input field with value 4.
- Living Area (500-6000 sqft): Input field with value 4000.
- Lot Area (500-20000 sqft): Input field with value 2500.
- Number of Floors (1-4): Input field with value 2.
- Water Front: Radio buttons for 'Yes' (selected) and 'No'.
- Age of the House: Input field with value 50.
- View Rating: Five star rating system with 4 stars selected.
- Condition Rating: Five star rating system with 4 stars selected.
- Above Basement Area (1000-7000 sqft): Input field with value 5000.
- Basement Area (0-2000 sqft): Input field with value 1000.
- City: Dropdown menu with 'Austin' selected.
- Zip Code: Input field with value 98106.

A green 'Predict Price' button is located at the bottom right of the form. Below the form, the text 'Predicted Price: 226122.13' is displayed.

- ✓ Let me brief you what are the different attributes present in our project.
 1. Number of Bedrooms: Must be between 1-9
 2. Number of Bathrooms: Must be between 1-9
 3. Living Area: Must be between 500-6000 square feet.
 4. Lot Area: Must be between 500-20000 square feet.
 5. Floors: Must be between 1-5
 6. Waterfront: Whether you need it or not.
 7. View Rating: which describes the overall look of the house.
 8. Condition Rating: which describes the overall condition of the house.

9. Above Basement Area: the area above the ground must be between 1000-7000 square feet.
10. Basement Area: the area below the basement must be between 0-2000 square feet.
11. City: You can select city of your own interest.
12. ZIP code: you need not enter zip code because we have already set that automatic. When you select a city, the zip code gets updated automatically.

Conclusion:

Our project represents a significant advancement in the field of real estate, offering a solution to predicting house prices. By integrating insights from data preprocessing, model training, and interface development, we have created a powerful tool capable of predicting house prices in real-time. Our model can analyze and user interactions in real-time and predicts accurately.

Recommendations and Future Directions:

- ✓ **Enhanced Data Collection:** You can get much more data that has more features like crime rate, proximity to schools, parks, and indicators like unemployment rates etc.
- ✓ **Model Complexity:** Given the non-linear and complex nature of house pricing, consider using more advanced machine learning techniques that can capture such complexities better than traditional models. Deep learning models could be explored to potentially yield better predictive performance.
- ✓ **Geospatial Analysis:** Add geospatial data to improve model predictions. Spatial data analysis can help in understanding the geographical patterns affecting house prices.
- ✓ **Risk Analysis Features:** create features that look at investment risks, such as the probability of price depreciation, to provide a more useful tool for investors.
- ✓ **Data Quality Improvement:** Regularly update and clean your dataset to maintain high data quality. This includes removing duplicates, handling missing values intelligently, and verifying data accuracy and relevance over time.

Future Work

- ✓ **Model Stacking:** Implement more models to improve predictions more accurately. This can help in blending different individual predictions to enhance the final output.
- ✓ **Automated Alerts:** Create a website such a way that can send alerts based on user's interest like price dropping or increase places, which could serve as valuable insights for investors or regular buyers.
- ✓ **Market Sentiment Analysis:** Get data from different social media platforms and articles to gauge market sentiments that could impact on housing prices.

References

<https://www.kaggle.com/datasets/shree1992/housedata>

<https://matplotlib.org/>

<https://seaborn.pydata.org/tutorial/introduction.html>

<https://realpython.com/python-data-cleaning-numpy-pandas/>

<https://docs.kanaries.net/articles/exploratory-data-analysis-python-pandas>

<https://machinelearningmastery.com/xgboost-for-regression/>

https://scikitlearn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regression.html

[https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html)

[learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html)

[https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html)

[learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html)

[https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

[learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>

<https://flask.palletsprojects.com/en/3.0.x/>

<https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3>