HOUSE PRICE PREDICTION ANALYSIS USING MACHINE LEARNING

PHASE 3: DEVELOPMENT PART 1

Start building the house price prediction model by loading and preprocessing the dataset.

Project Title: House Price Predictor

INTRODUCTION:

The housing market is an important and complex sector that impacts people's lives in many ways. For many individuals and families, buying a house is one of the biggest investments they will make in their lifetime. Therefore, it is essential to accurately predict the prices of houses so that buyers and sellers can make informed decisions. This project aims to use machine learning techniques to predict house prices based on various features such as location, square footage, number of bedrooms and bathrooms, and other relevant factors **.**

COLAB LINK:
https://colab.research.google.com/drive/1bE0cFUY6tFo317FFmzpjncu8Mxi1TkC-?usp=sharing

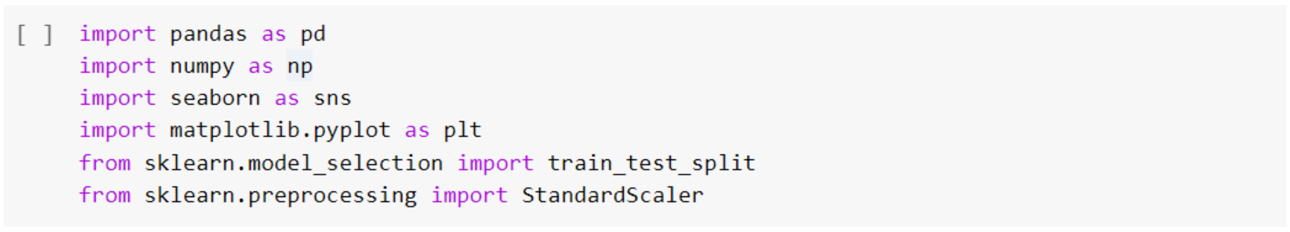
PROGRAM:

Importing dataset:

Loading data is a crucial step in any data analysis or machine learning task. It involves bringing external datasets into your programming environment so that you can manipulate, analyze, and draw insights from the data.

Importing Libraries:

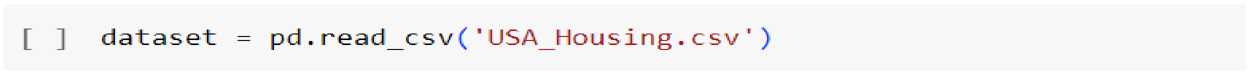
- import pandas as pd: For handling your dataset.
- import numpy as np: Useful for numerical operations.
- from sklearn.model_selection import train_test_split: Split your data into training and testing sets.
- import seaborn as sns: for data visualization in python.
- import matplotlib.pyplot as plt: another powerful library for creating visualizations.



```
[ ]  import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
```

Load Your Dataset:

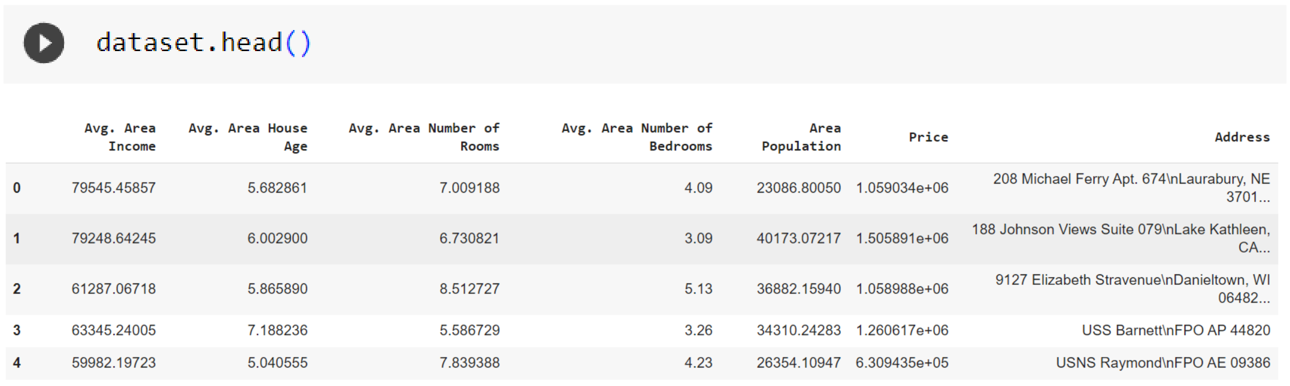
Use pd.read_csv() if your dataset is in a CSV file.



```
[ ]  dataset = pd.read_csv('USA_Housing.csv')
```

Explore Your Data:

Check out the first few rows using dataset.head().



```
dataset.head()
```

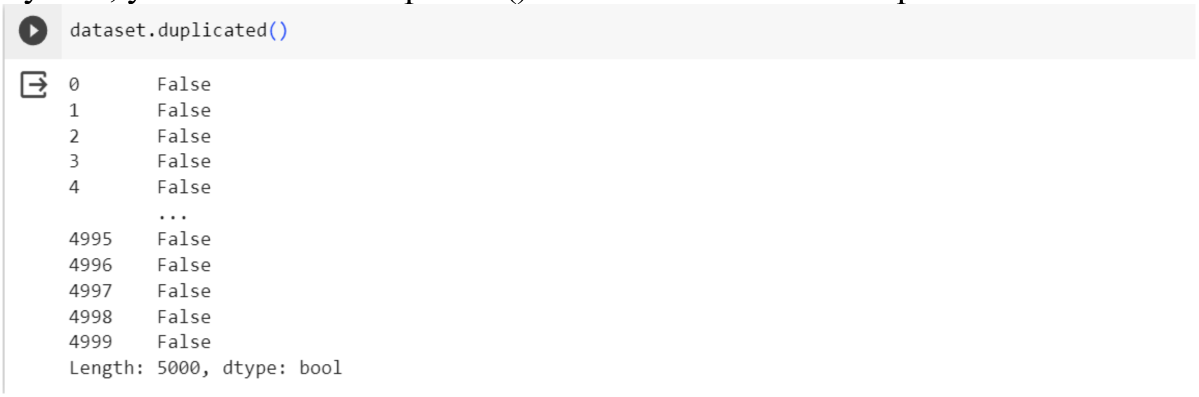
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386

Data cleaning techniques:

Data cleaning is the process of identifying and correcting errors, inconsistencies, and inaccuracies in datasets. It's a crucial step in the data analysis pipeline, as the quality of your analysis depends heavily on the quality of your data.

- Handling missing values
- Dealing with duplicates
- Handling Outliers
- Data type Conversion

If we want to identify and handle duplicate rows in a Pandas,DataFrame in Python, you can use the duplicate() function to check for duplicates.

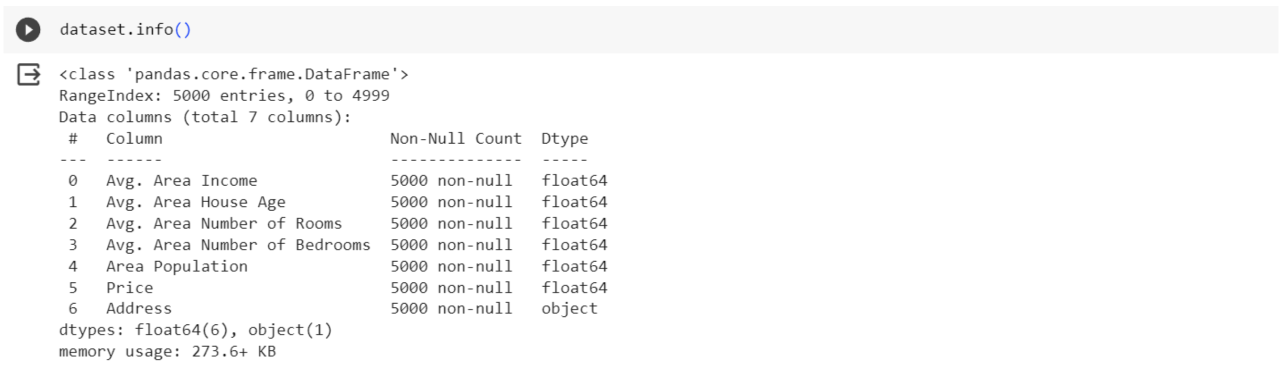


```
dataset.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
4995    False
4996    False
4997    False
4998    False
4999    False
Length: 5000, dtype: bool
```

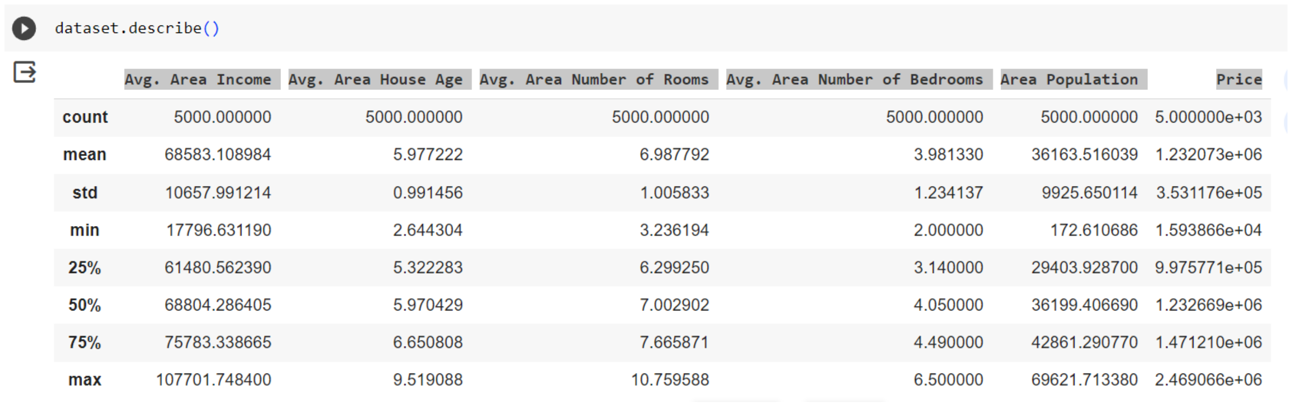
Use dataset.info() to get an overview of data types and missing values.

dataset.describe() gives you statistical summaries



```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```



```
dataset.describe()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562390	5.322283	6.299250	3.140000	29403.928700	9.975771e+05
50%	68804.286405	5.970429	7.002902	4.050000	36199.406690	1.232669e+06
75%	75783.338665	6.650808	7.665871	4.490000	42861.290770	1.471210e+06
max	107701.748400	9.519088	10.759588	6.500000	69621.713380	2.469066e+06

If we want to split a DataFrame into two separate DataFrames based on whether a column is of type "object" or not, we can use a for loop to iterate through the columns and check their data types.Then we can split the Dataframe into numerical column and categorical columns.

```python
# Categorical columns
cat_col = [col for col in dataset.columns if dataset[col].dtype == 'object']
print('Categorical columns :',cat_col)
# Numerical columns
num_col = [col for col in dataset.columns if dataset[col].dtype != 'object']
print('Numerical columns :',num_col)
```

```
Categorical columns : ['Address']
Numerical columns : ['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population', 'Price']
```

The nunique() function in pandas is used to count the number of unique values in a dataframe.

```python
[ ] dataset[cat_col].nunique()
```

```
Address     5000
dtype: int64
```

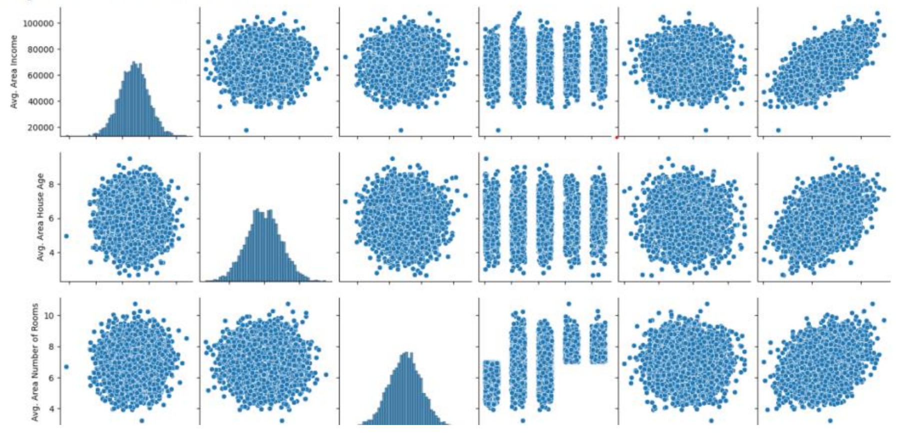
Check for missing values and decide on a strategy for handling them.

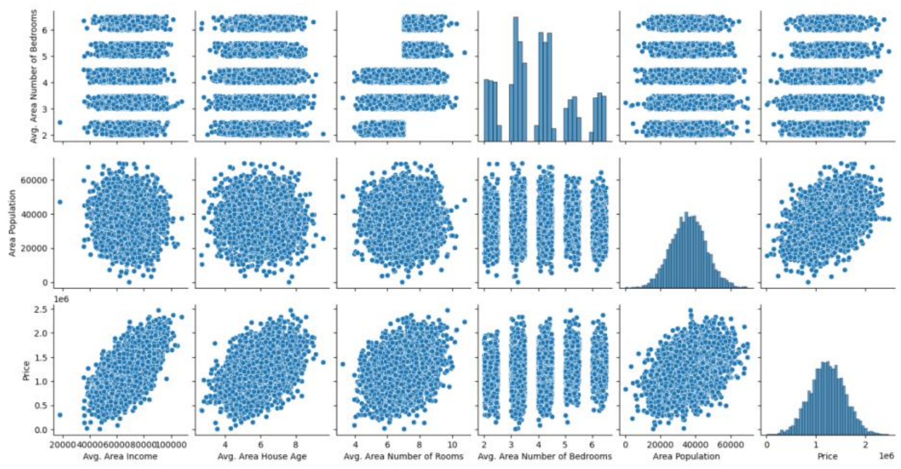
```python
round((dataset.isnull().sum()/dataset.shape[0])*100,2)
```

```
Avg. Area Income                0.0
Avg. Area House Age             0.0
Avg. Area Number of Rooms       0.0
Avg. Area Number of Bedrooms    0.0
Area Population                 0.0
Price                           0.0
Address                         0.0
dtype: float64
```

```python
plt.figure(figsize=(12,8))
sns.pairplot(dataset)
```

```
<seaborn.axisgrid.PairGrid at 0x79e7fe5cf100>
<Figure size 1200x800 with 0 Axes>
```



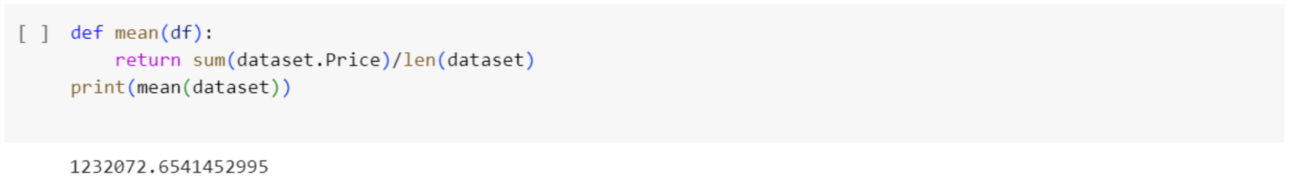


Data Analysis:

Data analysis is the process of inspecting, cleaning, transforming, and modeling data to uncover useful information, draw conclusions, and support decision-making.

- Data visualization
- Exploratory Data Analysis

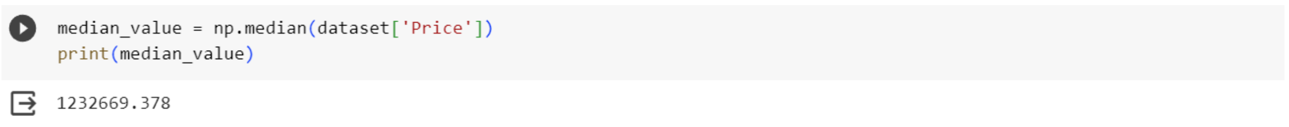
The mean is a measure of central tendency that represents the average value of a set of numbers. In the context of a dataset, it is often used to describe the central or typical value of a numerical variable.However, it's essential to be cautious with the mean, especially if the dataset contains outliers, as they can significantly impact its value.



```
[ ] def mean(df):
        return sum(dataset.Price)/len(dataset)
    print(mean(dataset))

    1232072.6541452995
```

The median is a measure of central tendency in a dataset, representing the middle value when the data is sorted in ascending or descending order. Unlike the mean (average), which is sensitive to extreme values, the median is resistant to outliers.



```
median_value = np.median(dataset['Price'])
print(median_value)

1232669.378
```

In the context of a dataset, the "mode" refers to the value that appears most frequently in a particular column or variable. It's a measure of central tendency, similar to mean (average) and median. It's particularly handy when dealing with categorical variables or when you want to identify the most common value in a dataset.

```python
import statistics
mode_result = statistics.mode(dataset['Price'])

print(f'Mode: {mode_result}')
```

```
Mode: 1059033.558
```

Standard deviation is a measure of the amount of variation or dispersion in a set of values. In the context of a dataset, it provides a quantifiable measure of how much individual data points differ from the mean (average) of the dataset.The formula for calculating the standard deviation involves taking the square root of the variance. While it provides valuable information about the spread of data, it should be used in conjunction with other descriptive statistics for a comprehensive analysis of a dataset.

```python
std_deviation = np.std(dataset['Price'])
print(f"Standard Deviation: {std_deviation}")
```

```
Standard Deviation: 353082.3130552725
```

Percentiles are a statistical concept used to describe the relative standing of a particular value within a dataset. The nth percentile of a dataset is the value below which n percent of the data falls. It's a way to understand the distribution of values and identify specific points in the data.

Median (50th percentile):

The value below which 50% of the data falls.

It divides the dataset into two equal halves.

Quartiles:

Q1 (25th percentile): The value below which 25% of the data falls.

Q2 (50th percentile, median): The value below which 50% of the data falls.

Q3 (75th percentile): The value below which 75% of the data falls.

Percentile Ranks:

The nth percentile is the value below which n% of the data falls.

For example, the 90th percentile is the value below which 90% of the data falls.

```python
percentiles = np.percentile(dataset['Price'], [25, 50, 75])
print(f"25th Percentile (Q1): {percentiles[0]}")
print(f"50th Percentile (Q2 or Median): {percentiles[1]}")
print(f"75th Percentile (Q3): {percentiles[2]}")
```

```
25th Percentile (Q1): 997577.135075
50th Percentile (Q2 or Median): 1232669.378
75th Percentile (Q3): 1471210.2045
```

A scatter plot is a type of data visualization that displays individual data points on a two-dimensional graph. Each point on the graph represents the values of two variables. Scatter plots are useful for visually identifying relationships or patterns between the two variables.

Key components of a scatter plot:

Data Points: Each point represents a pair of values from the two variables.

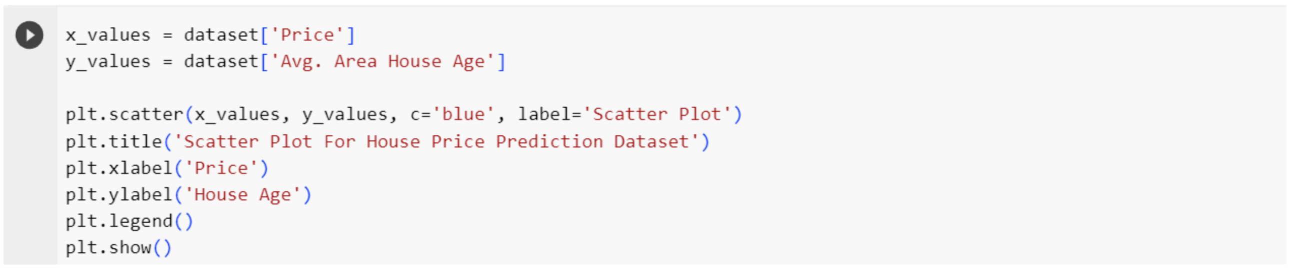
X-axis and Y-axis: The horizontal and vertical axes represent the values of the two variables.

Labels: Axes are labeled to indicate which variable they represent.

Title: Describes the purpose or context of the scatter plot.

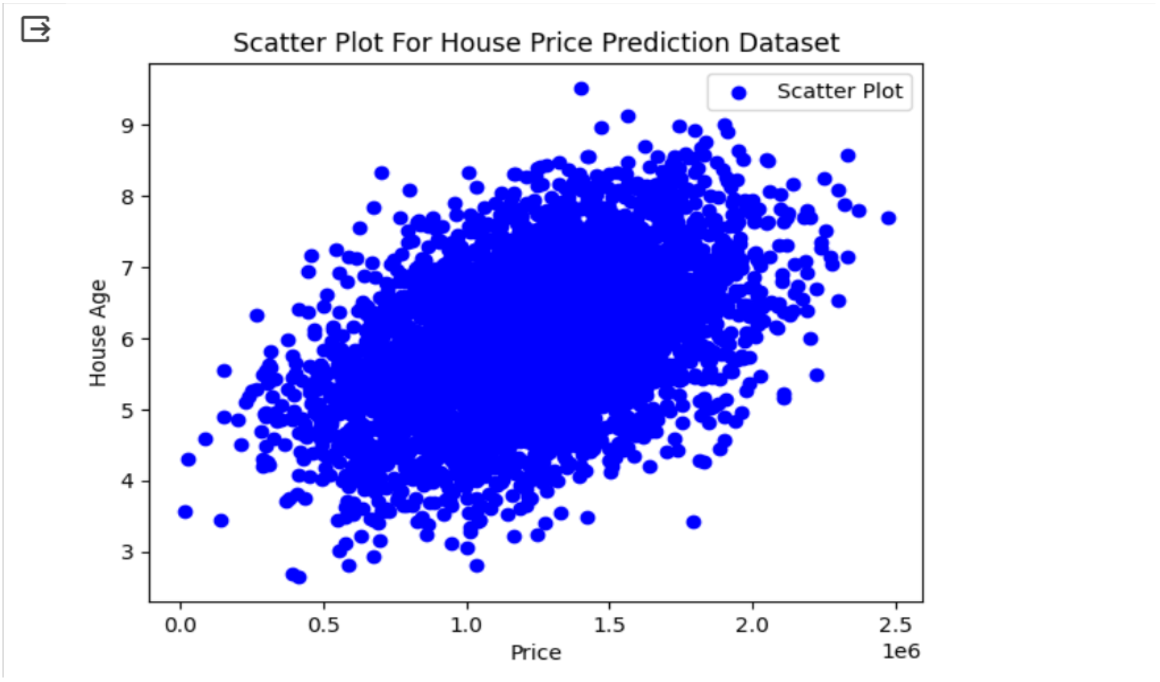
Legend: If multiple datasets are plotted, a legend helps identify them.

Scatter plots are especially useful when you want to explore the relationship between two continuous variables, identify trends, clusters, or outliers.



```
x_values = dataset['Price']
y_values = dataset['Avg. Area House Age']

plt.scatter(x_values, y_values, c='blue', label='Scatter Plot')
plt.title('Scatter Plot For House Price Prediction Dataset')
plt.xlabel('Price')
plt.ylabel('House Age')
plt.legend()
plt.show()
```



A histogram is a graphical representation of the distribution of a dataset. It displays the frequency of data points in different bins or ranges. In Python, you can create a histogram using the matplotlib library.

- data is an array of random values drawn from a normal distribution using NumPy.
- plt.hist() creates the histogram. The bins parameter determines the number of bins or ranges for the data, and alpha controls the transparency of the bars.
- Labels and a title are added for clarity.
- Finally, plt.show() displays the histogram.

Key components of a histogram:

Bins: Intervals along the x-axis representing ranges of values.

Frequency: Height of the bars indicates the number of data points in each bin.

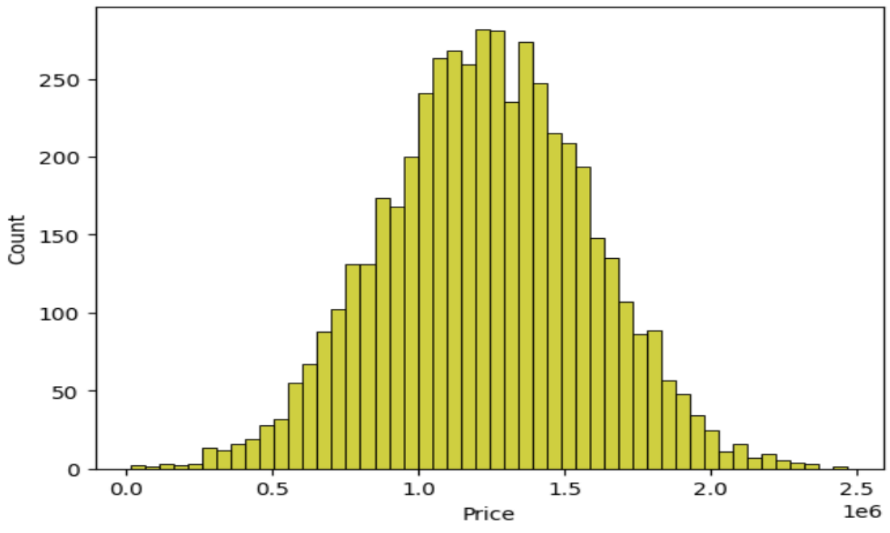
X-axis: Represents the values or ranges of the variable.

Y-axis: Represents the frequency or count of data points.

Histograms are useful for visualizing the distribution of a dataset, including information about central tendency, spread, and potential outliers.

```
sns.histplot(dataset, x='Price', bins=50, color='y')
```

```
<Axes: xlabel='Price', ylabel='Count'>
```



Data Preprocessing techniques:

Data preprocessing is a crucial step in the data analysis and machine learning pipeline. It involves cleaning and transforming raw data into a format suitable for analysis or model training. The goal is to enhance the quality of the data, address issues like missing values and outliers, and prepare it for effective exploration and modeling.

Label encoding is a technique to convert categorical data into numerical form, which is required for many machine learning algorithms. In Python, you can use the LabelEncoder class from the sklearn.preprocessing module to perform label encoding.

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
for column in dataset.columns:
    dataset[column] = labelencoder.fit_transform(dataset[column])
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   int64
 1   Avg. Area House Age           5000 non-null   int64
 2   Avg. Area Number of Rooms     5000 non-null   int64
 3   Avg. Area Number of Bedrooms  5000 non-null   int64
 4   Area Population               5000 non-null   int64
 5   Price                         5000 non-null   int64
 6   Address                       5000 non-null   int64
dtypes: int64(7)
memory usage: 273.6 KB
```

CONCLUSION:

These steps provide a simplified overview of concepts like data cleaning, data preprocessing, data analysis for the house price prediction. The specific implementation details and choice of algorithms may vary depending on the dataset and the goals of the project.