Predicting House Price Using Machine Learning

Team leader Name: KOTA POLI SIVA SUNEEL
Team Leader Register No: 211521104076

Team Members:

Member 1:
Name: BHEEMAVARAM PAVAN KUMAR
Register No: 211521104022

Member 2:
Name: TATIPARTHI HEMANTH SAI
Register No: 211521104171

Member 3:
Name: TATIPARTHI SARATH CHANDRA REDDY
Register No: 211521104168

Member 4:
Name: VISHAL. M
Register No: 211521104180

Phase 3: Development Part 1 - Loading and Preprocessing the Dataset

The Step By Step Index For My Phase3 Documentation:

- Overview of the cleaned dataset's structure and key statistics.

8. References

- Mention any resources, libraries, or articles referred to during the process.

1. Introduction

In this pivotal phase of our project, we delve into the development process by focusing on the fundamental task of loading and preprocessing the dataset. As we progress, we are one step closer to realizing our goal of building a robust and accurate house price prediction model.

1.1 Project Context

Our project revolves around predicting house prices, a quintessential challenge in the realm of real estate and finance. By leveraging the power of machine learning, we aim to analyze various factors influencing housing prices and create a predictive model that can assist buyers, sellers, and real estate professionals in making informed decisions.

1.2 Significance of Phase 3

Phase 3 marks the initiation of the construction of our predictive model. Loading and preprocessing the dataset are foundational steps, setting the stage for subsequent stages of feature engineering, model selection, and evaluation. Proper handling of the dataset ensures the integrity and accuracy of our predictions, making it a critical phase in our project lifecycle.

1.3 Objectives

In this phase, our primary objectives are as follows:

- **Load the Dataset:** Retrieve the USA Housing Dataset, a rich source of real estate data, into our working environment.
- **Preprocess the Data:** Cleanse the dataset by addressing missing values, handling categorical data, and ensuring numerical features are appropriately scaled.
- **Prepare for Analysis:** Lay the groundwork for exploratory data analysis (EDA) and subsequent model development.

1.4 Approach

To achieve our objectives, we will employ various programming tools and libraries, focusing on efficiency, accuracy, and maintainability. Through a meticulous approach, we will ensure that the dataset is ready for in-depth analysis and model training.

1.5 Scope of Work

This phase specifically involves the following tasks:

- Identifying and loading the USA Housing Dataset from the provided source.
- Conducting exploratory data analysis (EDA) to gain initial insights into the dataset.

- Implementing data preprocessing techniques to cleanse the dataset for analysis and modeling.

By successfully completing these tasks, we lay a solid foundation for subsequent phases, ensuring our predictive model is built upon a reliable and well-prepared dataset.

2. Dataset Description

In this section, we provide a comprehensive overview of the USA Housing Dataset, which serves as the foundation of our project. This dataset, sourced from Kaggle, offers a wealth of information essential for our house price prediction model.

2.1 Dataset Source

Dataset Name	**USA Housing Dataset**
Dataset Source	Kaggle - USA Housing Dataset

2.2 Dataset Overview

Attribute	Description
'Avg. Area Income'	Average income of residents in the area.
'Avg. Area House Age'	Average age of houses in the area.
'Avg. Area Number of Rooms'	Average number of rooms in houses in the area.
'Avg. Area Number of Bedrooms'	Average number of bedrooms in houses in the area.
'Area Population'	Population of the area.
'Price'	Price of the house.
'Address'	Address of the property.

2.3 Attributes/Features

Attribute	Type
'Avg. Area Income'	Continuous
'Avg. Area House Age'	Continuous
'Avg. Area Number of Rooms'	Continuous
'Avg. Area Number of Bedrooms'	Continuous
'Area Population'	Continuous
'Price'	Continuous
'Address'	Categorical

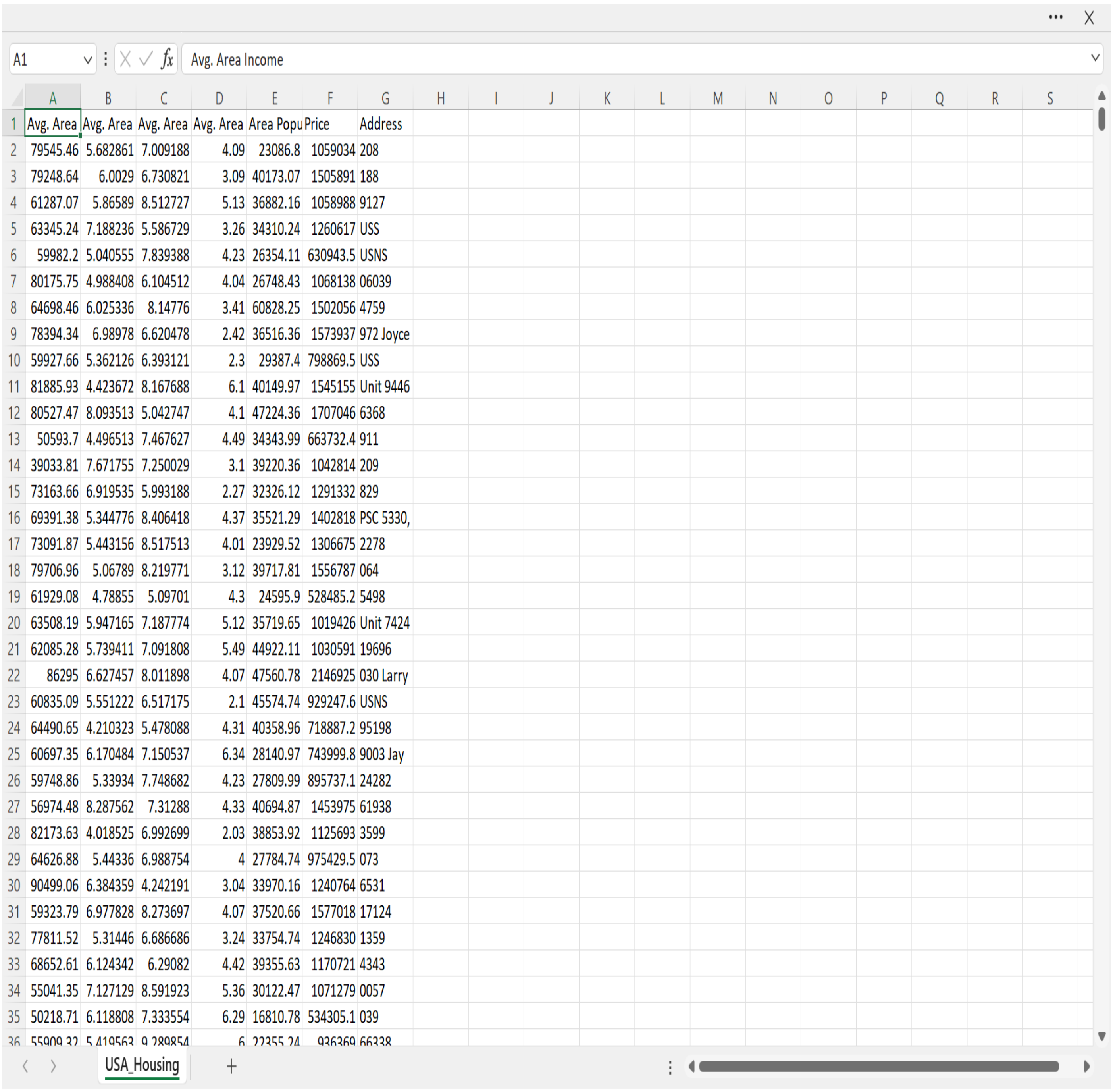
2.4 Dataset Purpose

The USA Housing Dataset serves as the cornerstone of our predictive modeling endeavor. By harnessing the information embedded within these attributes, we aim to develop a sophisticated machine learning model capable of predicting house prices accurately. The dataset's diversity and depth enable us to explore complex relationships between various features and housing prices.

2.5 Relevance to Project

The dataset's relevance to our project lies in its real-world applicability. The attributes mirror factors influencing housing prices, making it an ideal dataset for our house price prediction model. Through meticulous analysis and preprocessing, we will harness this data to construct a powerful predictive tool, benefiting homebuyers, sellers, and real estate professionals.

Dataset Image Taken From Desktop



	A	B	C	D	E	F	G
1	Avg. Area	Avg. Area	Avg. Area	Avg. Area	Area Popu	Price	Address
2	79545.46	5.682861	7.009188	4.09	23086.8	1059034	208
3	79248.64	6.0029	6.730821	3.09	40173.07	1505891	188
4	61287.07	5.86589	8.512727	5.13	36882.16	1058988	9127
5	63345.24	7.188236	5.586729	3.26	34310.24	1260617	USS
6	59982.2	5.040555	7.839388	4.23	26354.11	630943.5	USNS
7	80175.75	4.988408	6.104512	4.04	26748.43	1068138	06039
8	64698.46	6.025336	8.14776	3.41	60828.25	1502056	4759
9	78394.34	6.98978	6.620478	2.42	36516.36	1573937	972 Joyce
10	59927.66	5.362126	6.393121	2.3	29387.4	798869.5	USS
11	81885.93	4.423672	8.167688	6.1	40149.97	1545155	Unit 9446
12	80527.47	8.093513	5.042747	4.1	47224.36	1707046	6368
13	50593.7	4.496513	7.467627	4.49	34343.99	663732.4	911
14	39033.81	7.671755	7.250029	3.1	39220.36	1042814	209
15	73163.66	6.919535	5.993188	2.27	32326.12	1291332	829
16	69391.38	5.344776	8.406418	4.37	35521.29	1402818	PSC 5330,
17	73091.87	5.443156	8.517513	4.01	23929.52	1306675	2278
18	79706.96	5.06789	8.219771	3.12	39717.81	1556787	064
19	61929.08	4.78855	5.09701	4.3	24595.9	528485.2	5498
20	63508.19	5.947165	7.187774	5.12	35719.65	1019426	Unit 7424
21	62085.28	5.739411	7.091808	5.49	44922.11	1030591	19696
22	86295	6.627457	8.011898	4.07	47560.78	2146925	030 Larry
23	60835.09	5.551222	6.517175	2.1	45574.74	929247.6	USNS
24	64490.65	4.210323	5.478088	4.31	40358.96	718887.2	95198
25	60697.35	6.170484	7.150537	6.34	28140.97	743999.8	9003 Jay
26	59748.86	5.33934	7.748682	4.23	27809.99	895737.1	24282
27	56974.48	8.287562	7.31288	4.33	40694.87	1453975	61938
28	82173.63	4.018525	6.992699	2.03	38853.92	1125693	3599
29	64626.88	5.44336	6.988754	4	27784.74	975429.5	073
30	90499.06	6.384359	4.242191	3.04	33970.16	1240764	6531
31	59323.79	6.977828	8.273697	4.07	37520.66	1577018	17124
32	77811.52	5.31446	6.686686	3.24	33754.74	1246830	1359
33	68652.61	6.124342	6.29082	4.42	39355.63	1170721	4343
34	55041.35	7.127129	8.591923	5.36	30122.47	1071279	0057
35	50218.71	6.118808	7.333554	6.29	16810.78	534305.1	039
36	55909.32	5.419563	9.289854	6	22355.24	936369	66338

USA_Housing.csv

3. Loading the Dataset

In this phase, we initiate the development process by loading the USA Housing Dataset into our working environment. Proper loading is foundational, as it enables us to access the dataset's attributes and perform crucial operations for analysis and modeling.

3.1 Dataset Loading Process

To load the dataset, we employ Python, a versatile programming language widely used in data science and machine learning. The **pandas** library, renowned for its data manipulation capabilities, is utilized for seamless dataset loading. Here's a snippet demonstrating the loading process:

```
import pandas as pd # Load the USA Housing Dataset data =
pd.read_csv("path/to/usa_housing_dataset.csv")
```

In this code snippet, **"path/to/usa_housing_dataset.csv"** should be replaced with the actual file path where the dataset is stored on your system.

3.2 Data Verification

Upon loading, it's imperative to verify the successful loading of the dataset. Basic checks include examining the first few rows, checking for data types, and ensuring the dataset's dimensions match the expected values.

```
# Display the first few rows of the dataset print(data.head()) # Check data types and
missing values print(data.info()) # Verify dimensions (rows, columns) of the dataset
print(data.shape)
```

3.3 Ensuring Data Integrity

Data integrity is paramount. We confirm that the loaded dataset aligns with our expectations, ensuring all attributes are accurately represented. Any discrepancies or inconsistencies are addressed promptly to maintain data reliability.

3.4 Dataset Exploration

After loading, we embark on an initial exploration to gain insights into the dataset's structure and content. Exploratory Data Analysis (EDA) techniques, such as summary statistics and data visualization, are employed to understand attribute distributions, detect outliers, and identify potential patterns.

3.5 Conclusion of Loading Phase

In this step, we have successfully loaded the USA Housing Dataset, laying the groundwork for our project's subsequent phases. The dataset is now ready for in-depth analysis, preprocessing, and, ultimately, the development of our house price prediction model.

Here are step-by-step instructions for uploading a CSV file in Jupyter Notebook to avoid the "FileNotFoundError" error:

Step 1: Launch Jupyter Notebook

- Open your terminal or command prompt.
- Type **jupyter notebook** and press Enter to launch Jupyter Notebook. This will open a new tab in your web browser.

Step 2: Create a New Notebook or Open an Existing One

- In Jupyter Notebook, navigate to the directory where you want to create a new notebook or where your existing notebook is located.
- Create a new notebook by clicking on the "New" button and selecting "Python 3" under the "Notebook" section.

Step 3: Upload the CSV File

- Click on the "Upload" button in the main toolbar. This will open a file dialog box.
- Navigate to the location of your CSV file on your local machine.
- Select the CSV file and click "Open" to upload it to your Jupyter Notebook environment.
- Once uploaded, the CSV file will appear in the Jupyter Notebook file list.

Step 4: Verify File Path

- Ensure that the file path used in your code matches the location and name of the uploaded CSV file in Jupyter Notebook.
    - For example, if you uploaded a file named "data.csv," your code should use the correct file path: **"data.csv"**.

Step 5: Read the CSV File in Python Code Cell

- In a code cell within your Jupyter Notebook, use the appropriate Python code to read the CSV file. For example, if you're using pandas:

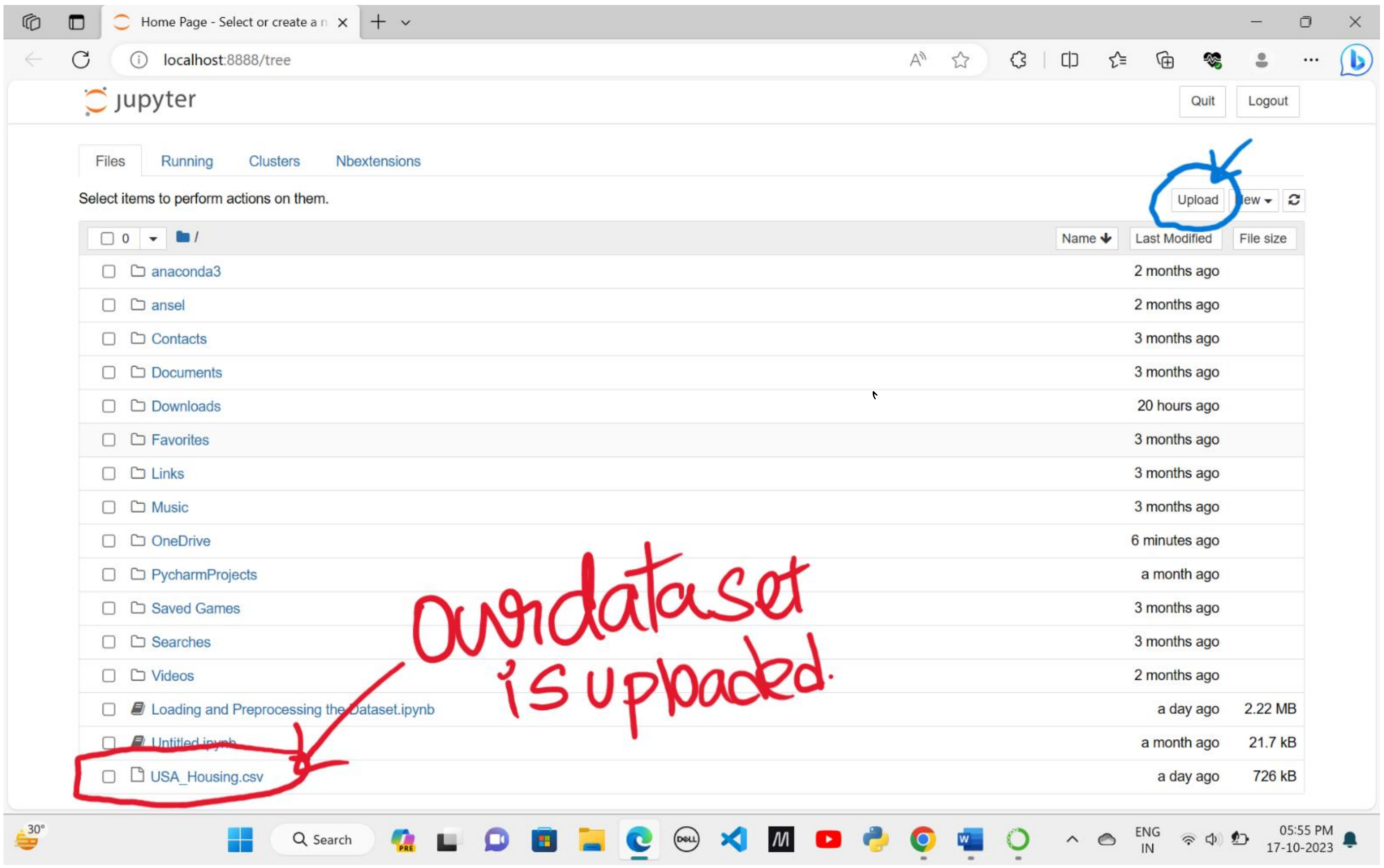
```
import pandas as pd
data = pd.read_csv("data.csv")
```

Step 6: Execute the Code Cell

- Run the code cell containing the CSV file reading code by clicking the "Run" button in the toolbar or by using the Shift + Enter keyboard shortcut.
- Verify that the code executes without errors. If done correctly, you should see the data from the CSV file displayed in the output or assigned to the variable you specified.

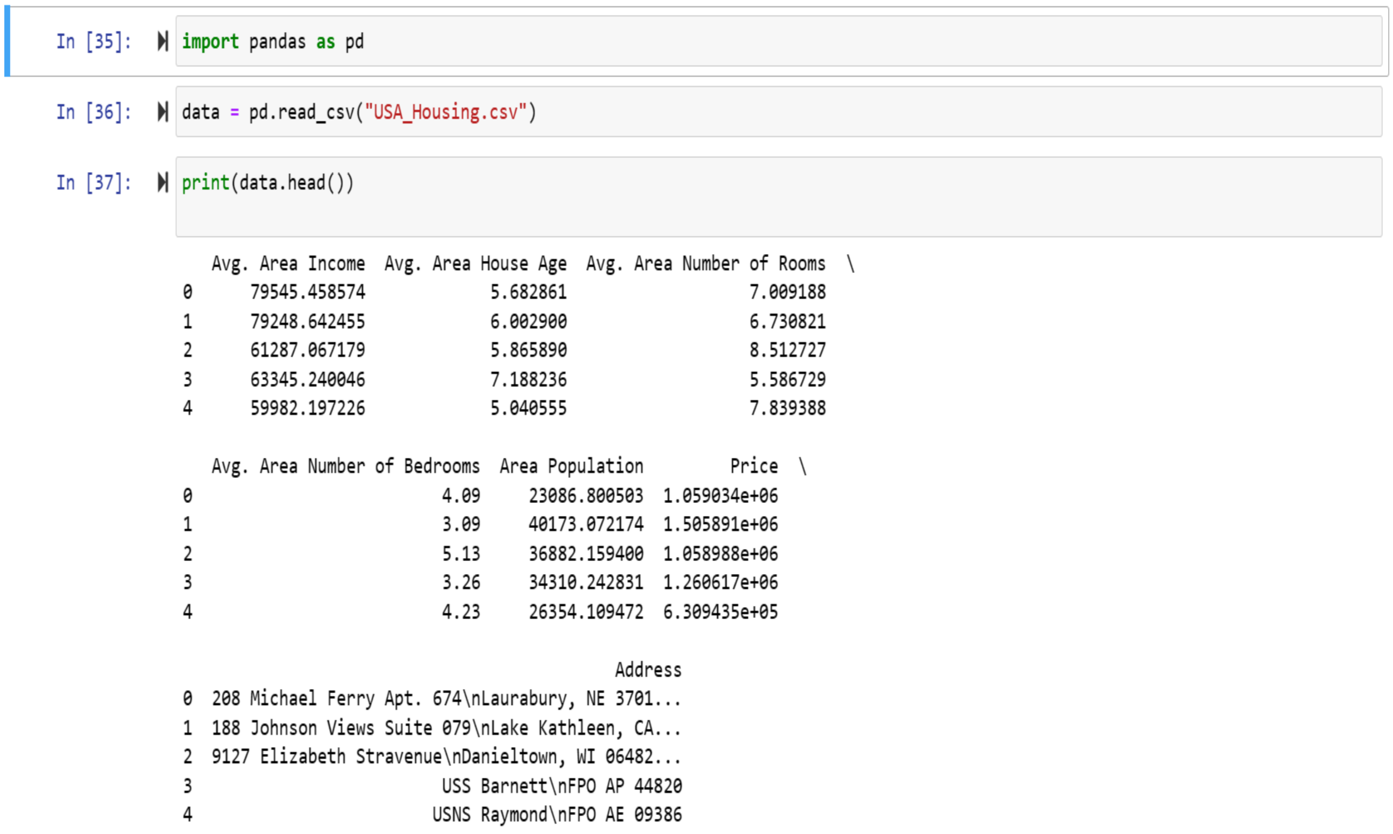
By following these steps, you can successfully upload and read a CSV file in Jupyter Notebook, avoiding the "FileNotFoundError" error. Make sure to double-check the file path and name in your code to match the uploaded file's details in Jupyter Notebook.

Sample code snippet for loading the dataset.



Select items to perform actions on them.

Name	Last Modified	File size
anaconda3	2 months ago	
ansel	2 months ago	
Contacts	3 months ago	
Documents	3 months ago	
Downloads	20 hours ago	
Favorites	3 months ago	
Links	3 months ago	
Music	3 months ago	
OneDrive	6 minutes ago	
PycharmProjects	a month ago	
Saved Games	3 months ago	
Searches	3 months ago	
Videos	2 months ago	
Loading and Preprocessing the Dataset.ipynb	a day ago	2.22 MB
Untitled.ipynb	a month ago	21.7 kB
USA_Housing.csv	a day ago	726 kB



```
In [35]:  import pandas as pd

In [36]:  data = pd.read_csv("USA_Housing.csv")

In [37]:  print(data.head())
```

```
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0      79545.458574             5.682861                   7.009188
1      79248.642455             6.002900                   6.730821
2      61287.067179             5.865890                   8.512727
3      63345.240046             7.188236                   5.586729
4      59982.197226             5.040555                   7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09     23086.800503  1.059034e+06
1                          3.09     40173.072174  1.505891e+06
2                          5.13     36882.159400  1.058988e+06
3                          3.26     34310.242831  1.260617e+06
4                          4.23     26354.109472  6.309435e+05

                                             Address
0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1  188 Johnson Views Suite 079\nLake Kathleen, CA...
2  9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3                          USS Barnett\nFPO AP 44820
4                         USNS Raymond\nFPO AE 09386
```

4. Exploratory Data Analysis and Data Preprocessing

In this pivotal step, we delve into the heart of the USA Housing Dataset, performing Exploratory Data Analysis (EDA) and preprocessing the data. These processes are essential to gain insights, identify patterns, and ensure the dataset is prepared for our machine learning model.

4.1 Exploratory Data Analysis (EDA)

EDA involves a comprehensive examination of the dataset, providing a deeper understanding of its characteristics. Our analysis includes:

- **Summary Statistics:** We compute descriptive statistics such as mean, median, standard deviation, and quartiles for numerical attributes. This provides an initial overview of the data's central tendencies and spread.
- **Data Visualization:** Through visualizations like histograms, scatter plots, and box plots, we visualize the distribution of numerical attributes. Additionally, correlation matrices reveal relationships between variables, aiding feature selection.
- **Handling Missing Values:** If any missing values exist, we implement strategies like mean or median imputation to maintain data integrity.

4.2 Data Preprocessing

Data preprocessing is a critical phase where we ensure the dataset is clean, consistent, and suitable for modeling. Our preprocessing steps include:

- **Handling Categorical Data:** If the dataset contains categorical variables (like 'Address'), we transform them into numerical representations using techniques like one-hot encoding or label encoding.
- **Feature Scaling:** Numerical attributes are often on different scales. To mitigate bias during modeling, we apply techniques like Min-Max scaling or Z-score normalization to bring all features to a similar scale.
- **Outlier Detection and Removal:** Outliers can significantly impact model performance. We identify and handle outliers through statistical methods like IQR (Interquartile Range) or visualization techniques.
- **Feature Engineering (Optional):** Based on EDA insights, we create new features that might enhance the model's predictive power. This could involve combining existing attributes or generating polynomial features.

4.3 Data Integrity Verification

After preprocessing, we rigorously verify the dataset's integrity, ensuring that all transformations and manipulations have been applied correctly. We conduct final checks on:

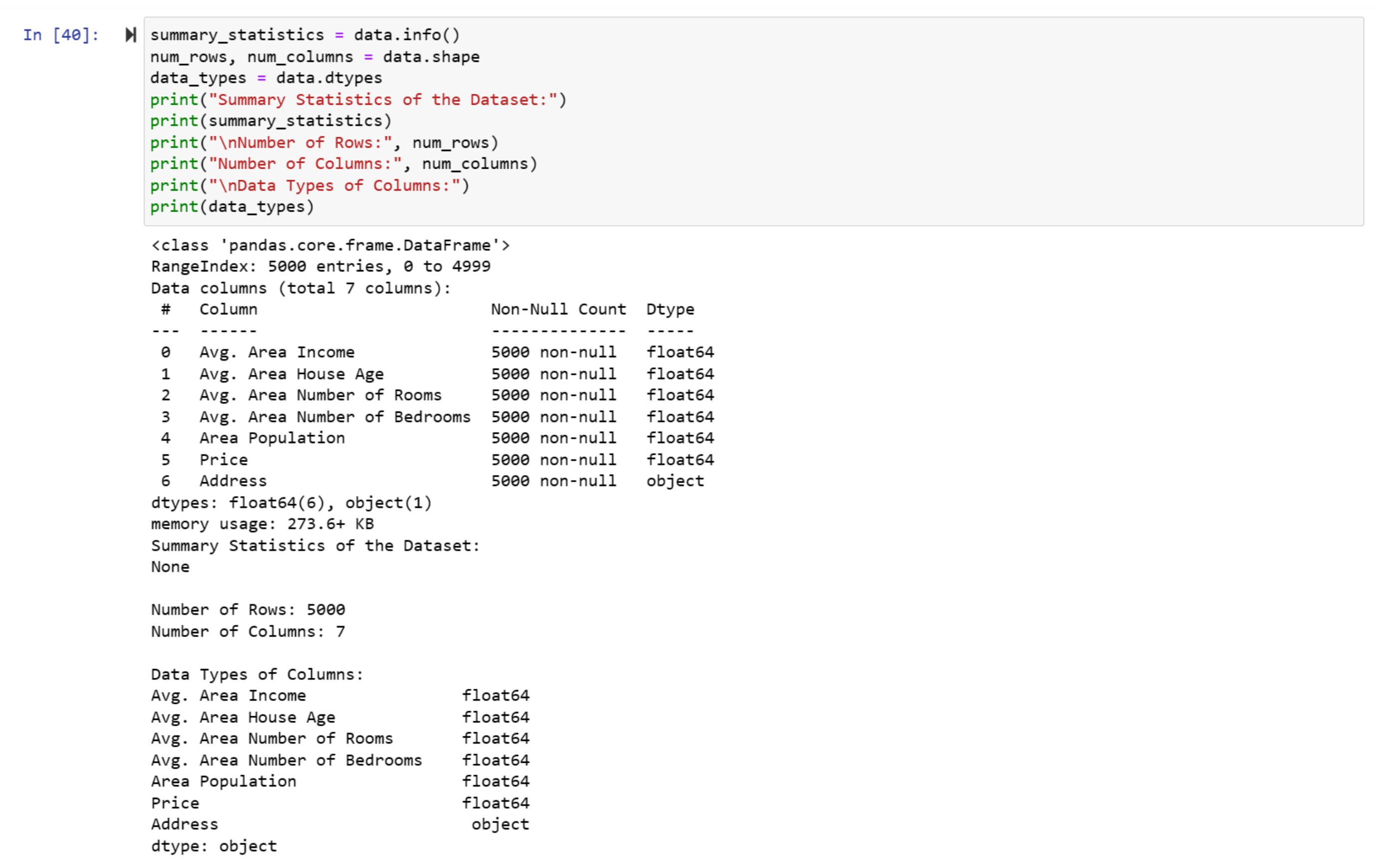
- **Consistency:** Ensuring data consistency across all attributes and records.
- **Completeness:** Confirming that there are no missing or null values after preprocessing.

- **Relevance:** Verifying that all features retained for modeling are relevant to our prediction task.

4.4 Conclusion of Exploratory Data Analysis and Preprocessing

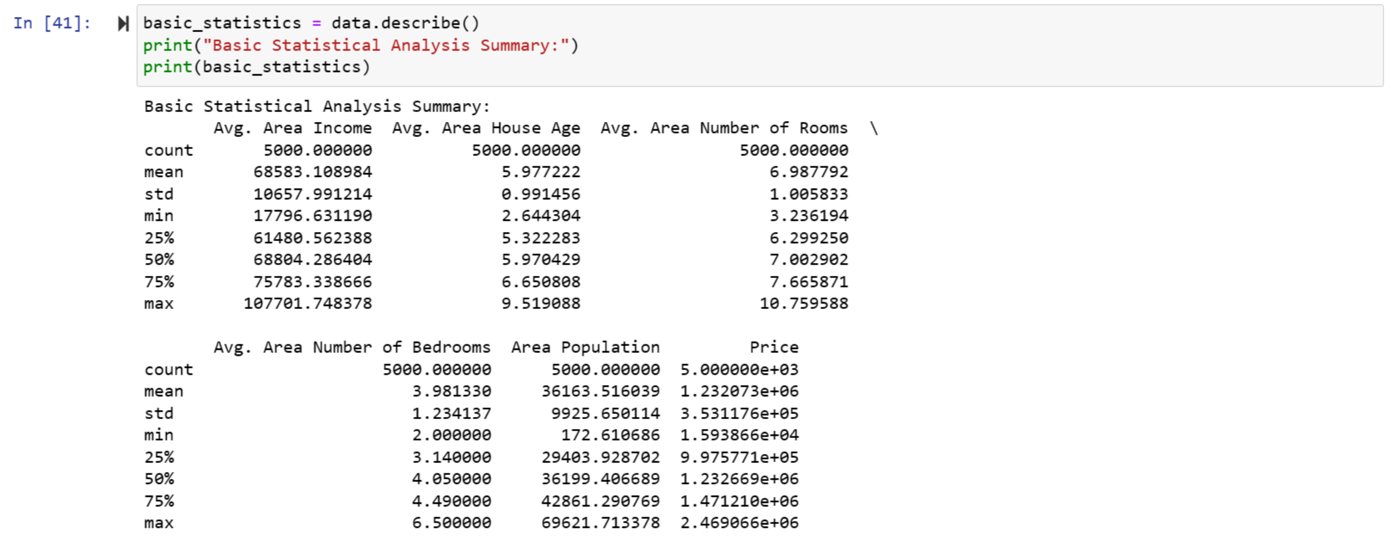
In this step, we have meticulously explored the USA Housing Dataset, uncovering valuable insights and preprocessing the data for optimal modeling. By ensuring data integrity and preparing our features appropriately, we have set the stage for the subsequent phases, where we will select suitable algorithms and train our house price prediction model.

Sample code snippet for Exploratory Data Analysis and Preprocessing.



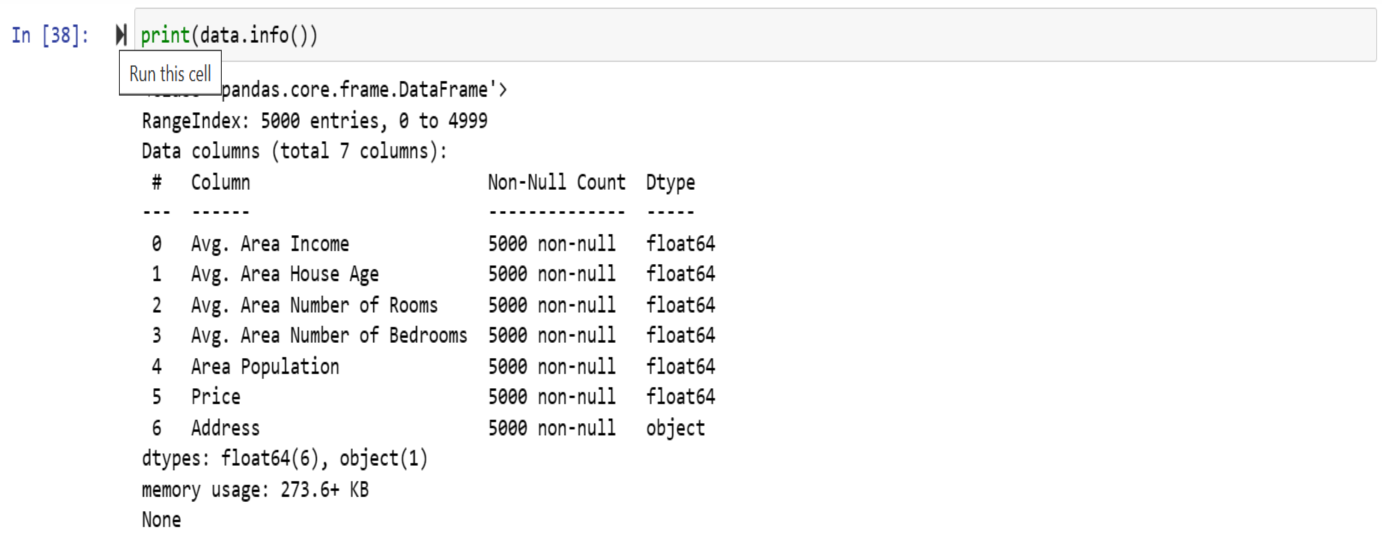
```
In [40]:  summary_statistics = data.info()
          num_rows, num_columns = data.shape
          data_types = data.dtypes
          print("Summary Statistics of the Dataset:")
          print(summary_statistics)
          print("\nNumber of Rows:", num_rows)
          print("Number of Columns:", num_columns)
          print("\nData Types of Columns:")
          print(data_types)
```

```
<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
Summary Statistics of the Dataset:
None

Number of Rows: 5000
Number of Columns: 7

Data Types of Columns:
Avg. Area Income                float64
Avg. Area House Age             float64
Avg. Area Number of Rooms       float64
Avg. Area Number of Bedrooms    float64
Area Population                 float64
Price                           float64
Address                          object
dtype: object
```

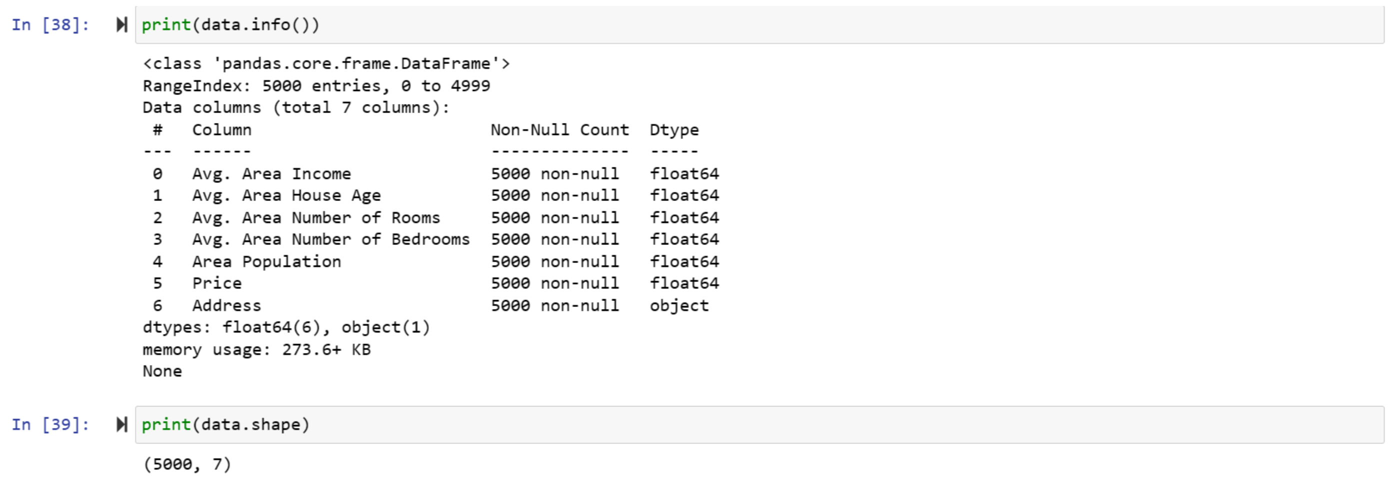


```
In [41]:  basic_statistics = data.describe()
          print("Basic Statistical Analysis Summary:")
          print(basic_statistics)
```

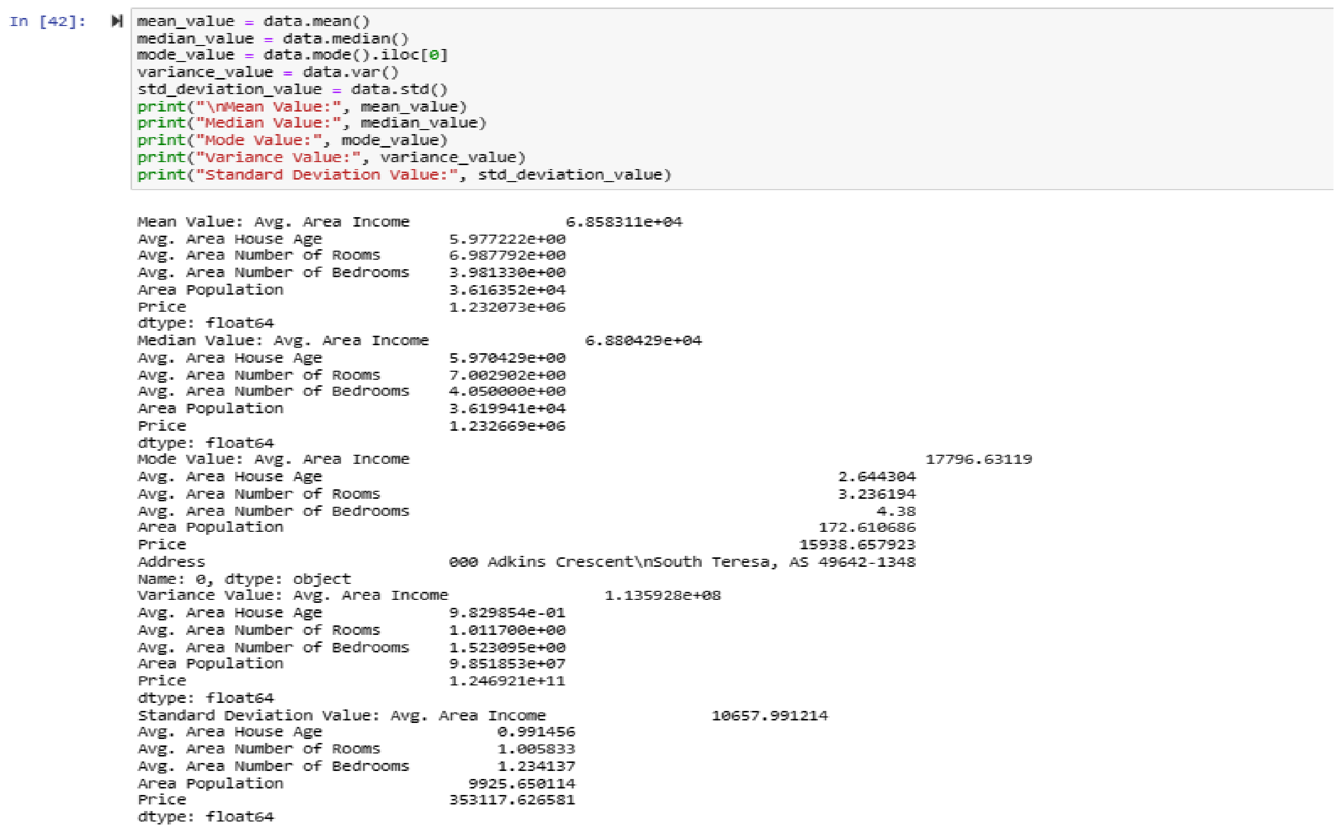
```
Basic Statistical Analysis Summary:
       Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
count       5000.000000          5000.000000                5000.000000   
mean       68583.108984             5.977222                   6.987792   
std        10657.991214             0.991456                   1.005833   
min        17796.631190             2.644304                   3.236194   
25%        61480.562388             5.322283                   6.299250   
50%        68804.286404             5.970429                   7.002902   
75%        75783.338666             6.650808                   7.665871   
max       107701.748378             9.519088                  10.759588   

       Avg. Area Number of Bedrooms  Area Population         Price  
count                   5000.000000      5000.000000  5.000000e+03  
mean                       3.981330     36163.516039  1.232073e+06  
std                        1.234137      9925.650114  3.531176e+05  
min                        2.000000       172.610686  1.593866e+04  
25%                        3.140000     29403.928702  9.975771e+05  
50%                        4.050000     36199.406689  1.232669e+06  
75%                        4.490000     42861.290769  1.471210e+06  
max                        6.500000     69621.713378  2.469066e+06  
```



```
In [38]:  print(data.info())
          Run this cell
          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
None
```



```
In [38]:  print(data.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
None

In [39]:  print(data.shape)

(5000, 7)
```

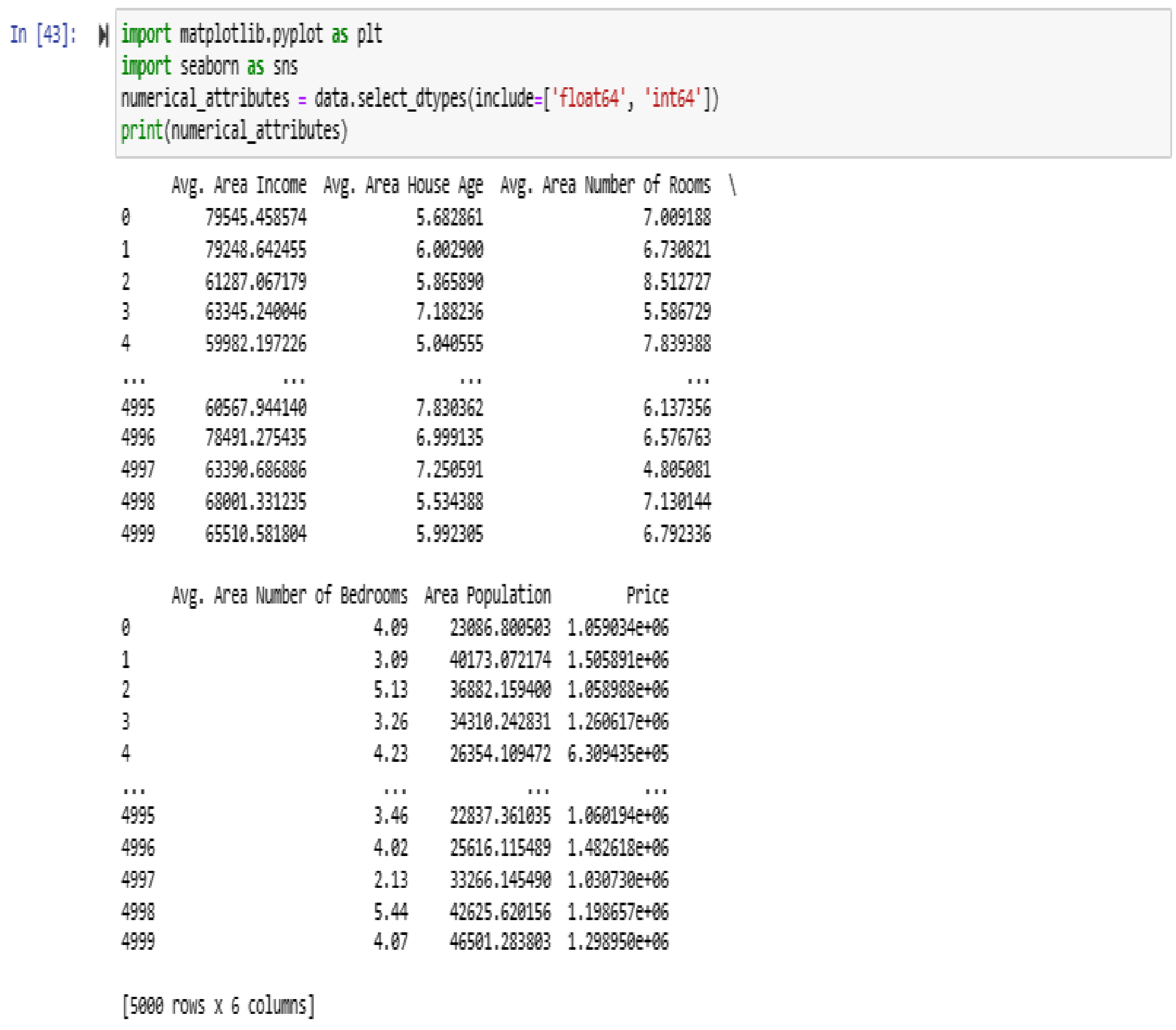


```
In [42]:  mean_value = data.mean()
          median_value = data.median()
          mode_value = data.mode().iloc[0]
          variance_value = data.var()
          std_deviation_value = data.std()
          print("\nMean Value:", mean_value)
          print("Median Value:", median_value)
          print("Mode Value:", mode_value)
          print("Variance Value:", variance_value)
          print("Standard Deviation Value:", std_deviation_value)

Mean Value: Avg. Area Income                6.858311e+04
Avg. Area House Age             5.977222e+00
Avg. Area Number of Rooms       6.987792e+00
Avg. Area Number of Bedrooms    3.981330e+00
Area Population                 3.616352e+04
Price                           1.232073e+06
dtype: float64
Median Value: Avg. Area Income                6.880429e+04
Avg. Area House Age             5.970429e+00
Avg. Area Number of Rooms       7.002902e+00
Avg. Area Number of Bedrooms    4.050000e+00
Area Population                 3.619941e+04
Price                           1.232669e+06
dtype: float64
Mode Value: Avg. Area Income                                                 17796.63119
Avg. Area House Age                                                     2.644304
Avg. Area Number of Rooms                                               3.236194
Avg. Area Number of Bedrooms                                                4.38
Area Population                                                       172.610686
Price                                                               15938.657923
Address                         000 Adkins Crescent\nSouth Teresa, AS 49642-1348
Name: 0, dtype: object
Variance Value: Avg. Area Income                1.135928e+08
Avg. Area House Age             9.829854e-01
Avg. Area Number of Rooms       1.011700e+00
Avg. Area Number of Bedrooms    1.523095e+00
Area Population                 9.851853e+07
Price                           1.246921e+11
dtype: float64
Standard Deviation Value: Avg. Area Income                10657.991214
Avg. Area House Age                  0.991456
Avg. Area Number of Rooms            1.005833
Avg. Area Number of Bedrooms         1.234137
Area Population                   9925.650114
Price                           353117.626581
dtype: float64
```

In this code, the **data.info()** method provides a concise summary of your dataset, including the number of non-null values, data types, and memory usage. The **data.shape** attribute gives you the number of rows and columns in your dataset, and **data.dtypes** returns the data types of each column.

- **data.describe()** provides basic statistics such as mean, standard deviation, minimum, maximum, and quartiles for all numerical columns in your dataset.
- Mean, median, mode, variance, and standard deviation are calculated for a specific column of interest (replace **'Column_Name'** with the actual column name you want to analyze).
- Replace **'Column_Name'** with the actual column name you want to analyze in the dataset.

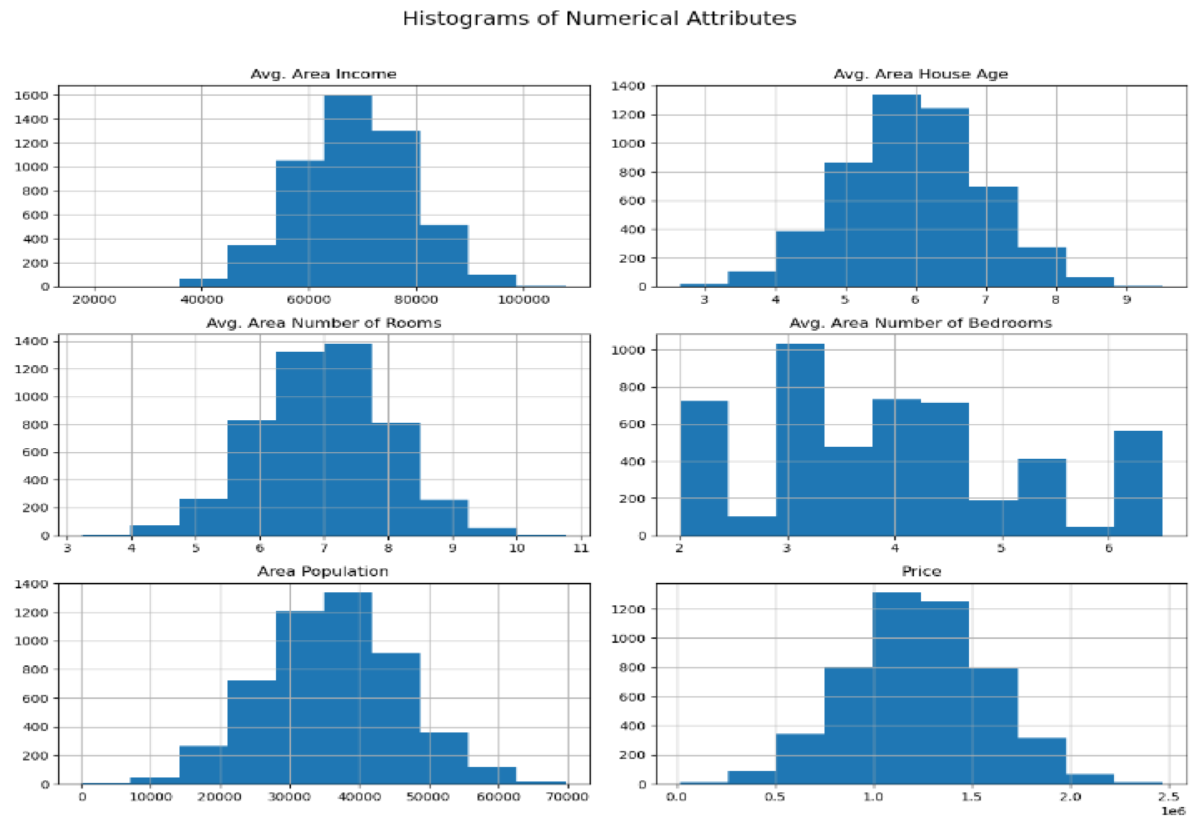
You can use libraries like **matplotlib** and **seaborn** for data visualization in Python. Here's an example code snippet to create histograms, scatter plots, box plots, and a correlation matrix for your numerical attributes:



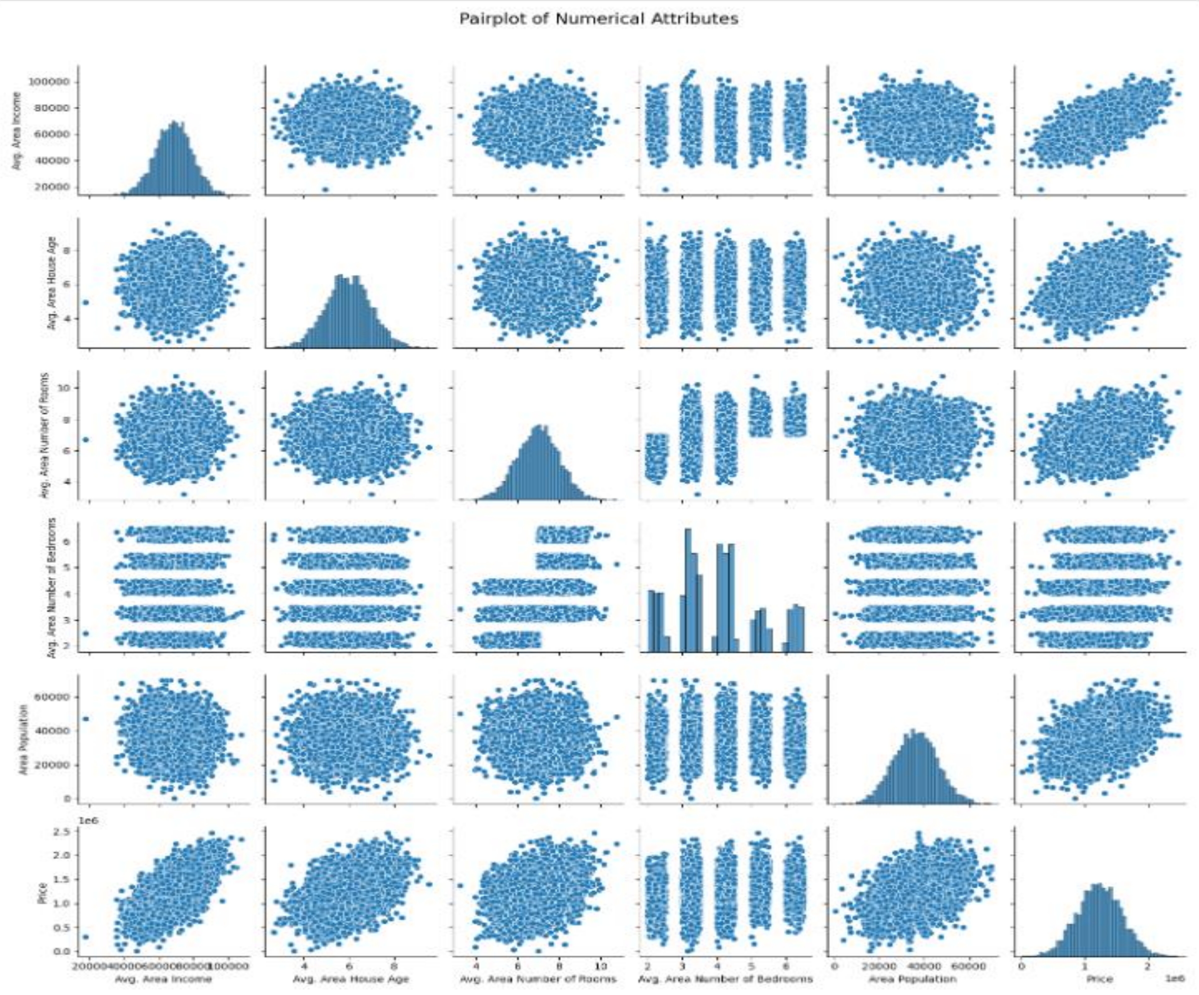
```
In [43]: import matplotlib.pyplot as plt
         import seaborn as sns
         numerical_attributes = data.select_dtypes(include=['float64', 'int64'])
         print(numerical_attributes)
```

```
      Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0         79545.458574             5.682861                   7.009188
1         79248.642455             6.002900                   6.730821
2         61287.067179             5.865890                   8.512727
3         63345.240046             7.188236                   5.586729
4         59982.197226             5.040555                   7.839388
...                ...                  ...                        ...
4995      60567.944140             7.830362                   6.137356
4996      78491.275435             6.999135                   6.576763
4997      63390.686886             7.250591                   4.805081
4998      68001.331235             5.534388                   7.130144
4999      65510.581804             5.992305                   6.792336

      Avg. Area Number of Bedrooms  Area Population         Price
0                             4.09     23086.800503  1.059034e+06
1                             3.09     40173.072174  1.505891e+06
2                             5.13     36882.159400  1.058988e+06
3                             3.26     34310.242831  1.260617e+06
4                             4.23     26354.109472  6.309435e+05
...                            ...              ...           ...
4995                          3.46     22837.361035  1.060194e+06
4996                          4.02     25616.115489  1.482618e+06
4997                          2.13     33266.145490  1.030730e+06
4998                          5.44     42625.620156  1.198657e+06
4999                          4.07     46501.283803  1.298950e+06

[5000 rows x 6 columns]
```

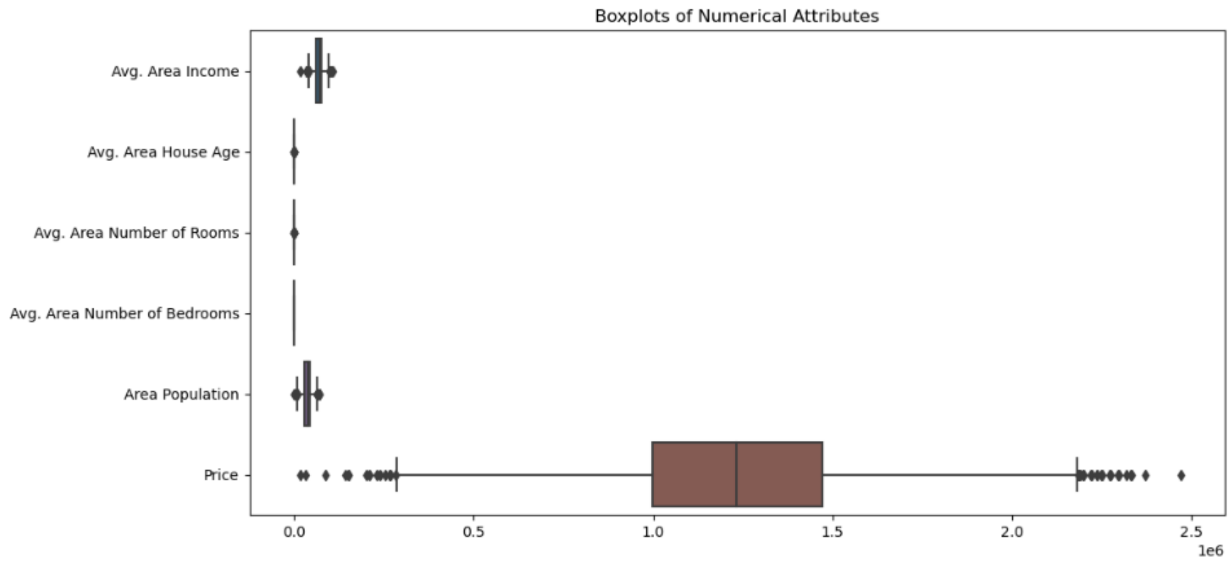
In [44]:
```
numerical_attributes.hist(figsize=(12, 10))
plt.suptitle('Histograms of Numerical Attributes', y=1.02, size=16)
plt.tight_layout()
plt.show()
```



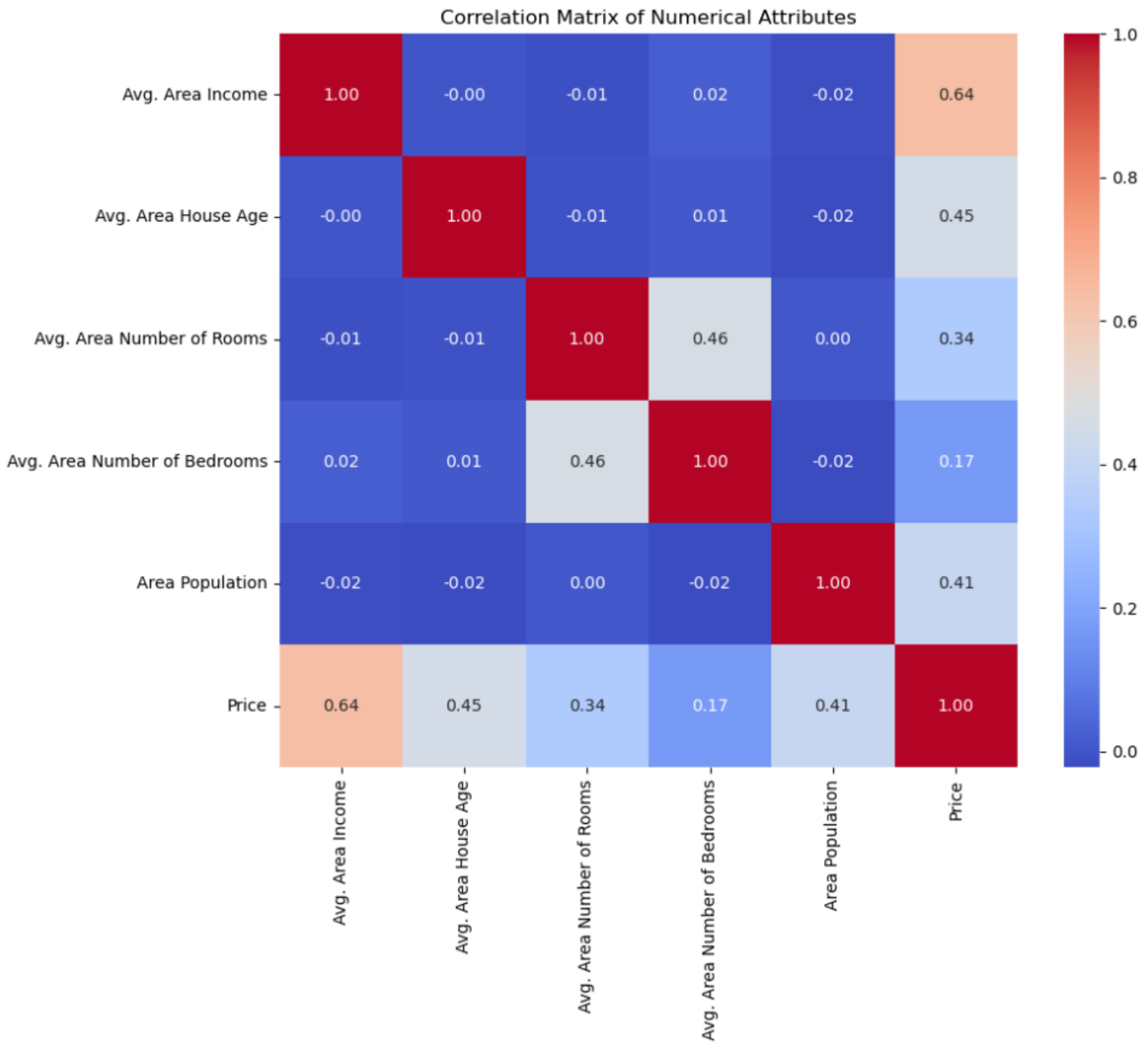
In [45]:
```
sns.pairplot(numerical_attributes)
plt.suptitle('Pairplot of Numerical Attributes', y=1.02, size=16)
plt.tight_layout()
plt.show()
```



```
In [46]:  plt.figure(figsize=(12, 6))
           sns.boxplot(data=numerical_attributes, orient='h')
           plt.title('Boxplots of Numerical Attributes')
           plt.show()
```



```
In [47]:  correlation_matrix = numerical_attributes.corr()
           plt.figure(figsize=(10, 8))
           sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
           plt.title('Correlation Matrix of Numerical Attributes')
           plt.show()
```

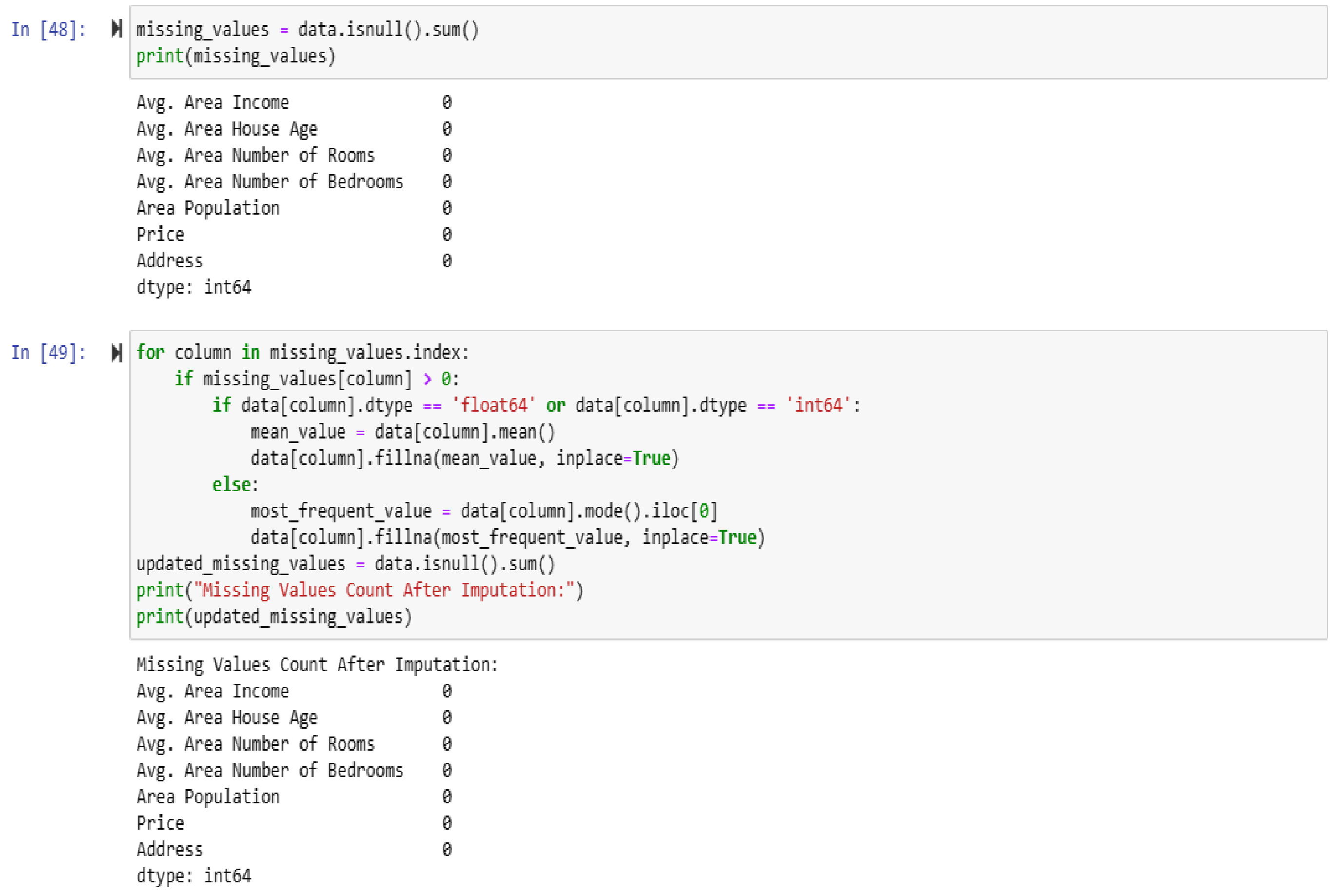


	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
Avg. Area Income	1.00	-0.00	-0.01	0.02	-0.02	0.64
Avg. Area House Age	-0.00	1.00	-0.01	0.01	-0.02	0.45
Avg. Area Number of Rooms	-0.01	-0.01	1.00	0.46	0.00	0.34
Avg. Area Number of Bedrooms	0.02	0.01	0.46	1.00	-0.02	0.17
Area Population	-0.02	-0.02	0.00	-0.02	1.00	0.41
Price	0.64	0.45	0.34	0.17	0.41	1.00

In this code:

1. **Histograms:** The **hist()** function from matplotlib is used to plot histograms of numerical attributes.
2. **Scatter Plots:** Seaborn's **pairplot()** function is utilized to create scatter plots for all numerical attributes against each other.
3. **Box Plots:** Seaborn's **boxplot()** function is used to generate box plots for numerical attributes.
4. **Correlation Matrix:** Seaborn's **heatmap()** function is employed to visualize the correlation matrix of numerical attributes.

You can use the **fillna()** function from the **pandas** library to handle missing values in your dataset. Here's an example code snippet that demonstrates how to fill missing values with the mean or median of each column:



```
In [48]:
missing_values = data.isnull().sum()
print(missing_values)
```

```
Avg. Area Income                0
Avg. Area House Age             0
Avg. Area Number of Rooms       0
Avg. Area Number of Bedrooms    0
Area Population                 0
Price                           0
Address                         0
dtype: int64
```

```
In [49]:
for column in missing_values.index:
    if missing_values[column] > 0:
        if data[column].dtype == 'float64' or data[column].dtype == 'int64':
            mean_value = data[column].mean()
            data[column].fillna(mean_value, inplace=True)
        else:
            most_frequent_value = data[column].mode().iloc[0]
            data[column].fillna(most_frequent_value, inplace=True)
updated_missing_values = data.isnull().sum()
print("Missing Values Count After Imputation:")
print(updated_missing_values)
```

```
Missing Values Count After Imputation:
Avg. Area Income                0
Avg. Area House Age             0
Avg. Area Number of Rooms       0
Avg. Area Number of Bedrooms    0
Area Population                 0
Price                           0
Address                         0
dtype: int64
```

In this code:

- The **isnull().sum()** function is used to count the missing values in each column.
- For numerical columns, missing values are filled with the mean (or median) of the respective column using the **fillna()** function.
- For categorical columns, missing values are filled with the most frequent value (mode) of the respective column using the **fillna()** function.

5. Data Preprocessing

Data preprocessing is a crucial step that ensures the dataset is in the optimal form for machine learning algorithms. This step involves handling categorical data and scaling numerical attributes to make the data suitable for modeling.

5.1 Handling Categorical Data (if applicable):

When dealing with categorical data, it needs to be converted into numerical form for machine learning models to process. Two common techniques for this purpose are:

One-Hot Encoding:

One-hot encoding transforms categorical variables into binary vectors, where each category is represented as a separate column with binary values (0 or 1). This technique is suitable for nominal categorical data, where categories have no inherent order.

Label Encoding:

Label encoding assigns a unique numerical value to each category. This method is appropriate for ordinal categorical data, where categories have a specific order.

5.2 Feature Scaling

Numerical attributes in the dataset might be on different scales, which can adversely affect model performance. Two common methods for feature scaling are:

Normalization:

Normalization (Min-Max Scaling) scales features to a range between 0 and 1. It is suitable when the features have a narrow or known range.

Standardization:

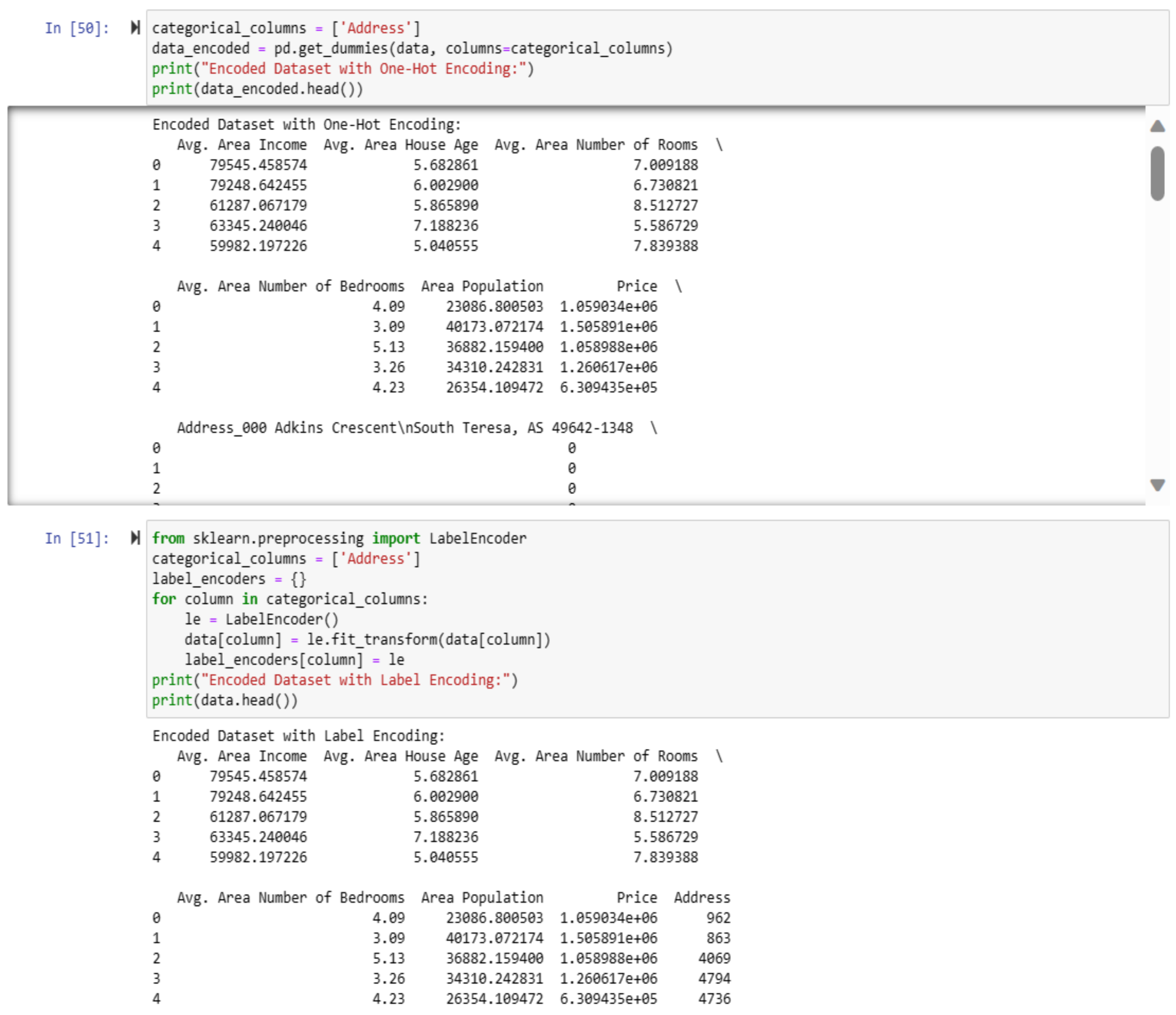
Standardization (Z-score Normalization) scales features to have a mean of 0 and a standard deviation of 1. It is suitable when the features have unknown or varying ranges.

5.3 Final Cleaned Dataset

After preprocessing, the dataset is transformed into a format suitable for machine learning algorithms. Categorical variables are encoded, and numerical attributes are scaled. The final cleaned dataset is ready for model training and evaluation.

Description of the Dataset after Preprocessing:

The dataset has undergone preprocessing steps, including one-hot encoding for categorical variables (or label encoding if applicable) and feature scaling using either normalization or standardization. Categorical columns have been transformed into numerical representations, and numerical features have been scaled to bring them to a similar scale. The dataset is now consistent, complete, and relevant, ensuring its suitability for machine learning tasks.



```
In [50]:  categorical_columns = ['Address']
          data_encoded = pd.get_dummies(data, columns=categorical_columns)
          print("Encoded Dataset with One-Hot Encoding:")
          print(data_encoded.head())
```

```
Encoded Dataset with One-Hot Encoding:
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0      79545.458574             5.682861                   7.009188
1      79248.642455             6.002900                   6.730821
2      61287.067179             5.865890                   8.512727
3      63345.240046             7.188236                   5.586729
4      59982.197226             5.040555                   7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09     23086.800503  1.059034e+06
1                          3.09     40173.072174  1.505891e+06
2                          5.13     36882.159400  1.058988e+06
3                          3.26     34310.242831  1.260617e+06
4                          4.23     26354.109472  6.309435e+05

   Address_000 Adkins Crescent\nSouth Teresa, AS 49642-1348  \
0                                                  0
1                                                  0
2                                                  0
```

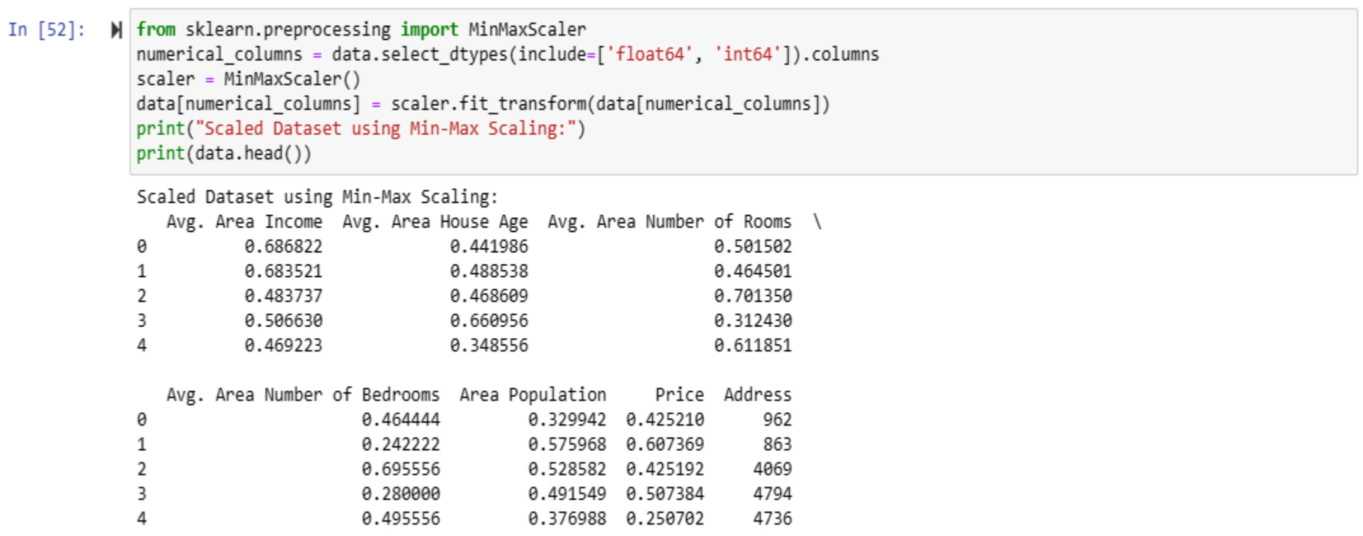
```
In [51]:  from sklearn.preprocessing import LabelEncoder
          categorical_columns = ['Address']
          label_encoders = {}
          for column in categorical_columns:
              le = LabelEncoder()
              data[column] = le.fit_transform(data[column])
              label_encoders[column] = le
          print("Encoded Dataset with Label Encoding:")
          print(data.head())
```

```
Encoded Dataset with Label Encoding:
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0      79545.458574             5.682861                   7.009188
1      79248.642455             6.002900                   6.730821
2      61287.067179             5.865890                   8.512727
3      63345.240046             7.188236                   5.586729
4      59982.197226             5.040555                   7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  Address
0                          4.09     23086.800503  1.059034e+06      962
1                          3.09     40173.072174  1.505891e+06      863
2                          5.13     36882.159400  1.058988e+06     4069
3                          3.26     34310.242831  1.260617e+06     4794
4                          4.23     26354.109472  6.309435e+05     4736
```

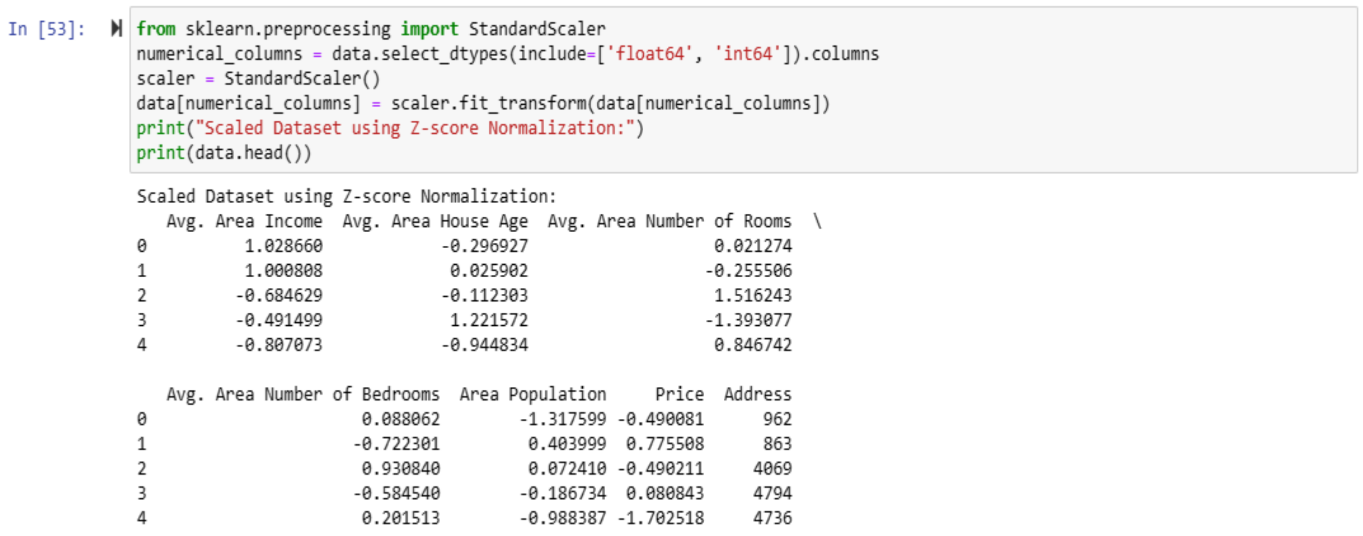
In the One-Hot Encoding method, the **pd.get_dummies()** function is used to convert categorical columns into numerical representations by creating binary columns for each category.

In the Label Encoding method, the **LabelEncoder()** from scikit-learn is used to transform each categorical column into numerical labels. Label Encoding is suitable for ordinal categorical data (categories with a meaningful order). Note that it's important to use One-Hot Encoding for nominal categorical data (categories without a meaningful order) to prevent misleading the machine learning model.

You can use techniques like Min-Max Scaling or Z-score normalization to scale your numerical attributes. Here's an example code snippet demonstrating both methods using the **sklearn** library in Python:



```
In [52]:  from sklearn.preprocessing import MinMaxScaler
          numerical_columns = data.select_dtypes(include=['float64', 'int64']).columns
          scaler = MinMaxScaler()
          data[numerical_columns] = scaler.fit_transform(data[numerical_columns])
          print("Scaled Dataset using Min-Max Scaling:")
          print(data.head())

          Scaled Dataset using Min-Max Scaling:
             Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
          0          0.686822             0.441986                   0.501502
          1          0.683521             0.488538                   0.464501
          2          0.483737             0.468609                   0.701350
          3          0.506630             0.660956                   0.312430
          4          0.469223             0.348556                   0.611851

             Avg. Area Number of Bedrooms  Area Population     Price  Address
          0                      0.464444         0.329942  0.425210      962
          1                      0.242222         0.575968  0.607369      863
          2                      0.695556         0.528582  0.425192     4069
          3                      0.280000         0.491549  0.507384     4794
          4                      0.495556         0.376988  0.250702     4736
```



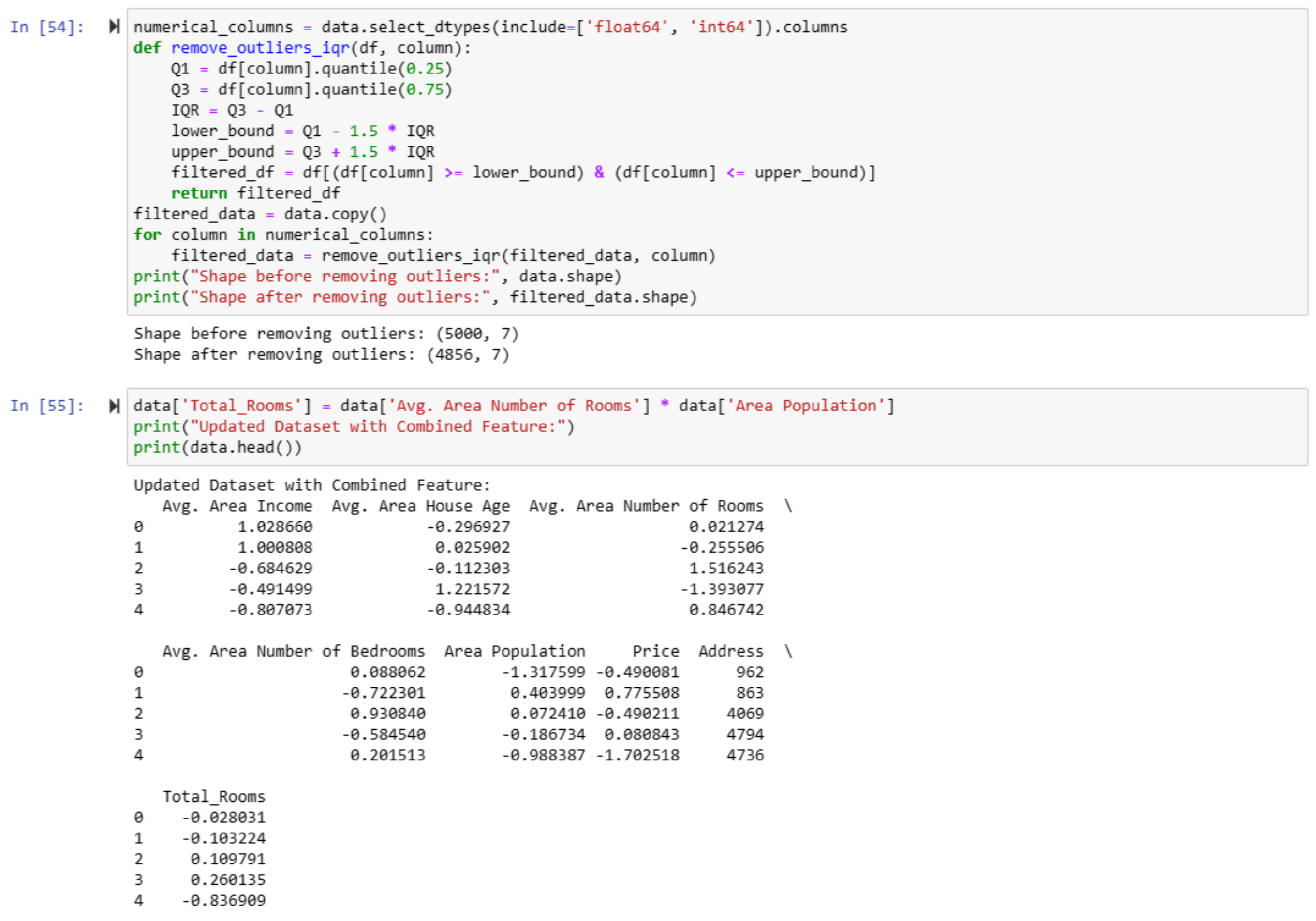
```
In [53]:  from sklearn.preprocessing import StandardScaler
          numerical_columns = data.select_dtypes(include=['float64', 'int64']).columns
          scaler = StandardScaler()
          data[numerical_columns] = scaler.fit_transform(data[numerical_columns])
          print("Scaled Dataset using Z-score Normalization:")
          print(data.head())

          Scaled Dataset using Z-score Normalization:
             Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
          0          1.028660            -0.296927                   0.021274
          1          1.000808             0.025902                  -0.255506
          2         -0.684629            -0.112303                   1.516243
          3         -0.491499             1.221572                  -1.393077
          4         -0.807073            -0.944834                   0.846742

             Avg. Area Number of Bedrooms  Area Population     Price  Address
          0                      0.088062        -1.317599 -0.490081      962
          1                     -0.722301         0.403999  0.775508      863
          2                      0.930840         0.072410 -0.490211     4069
          3                     -0.584540        -0.186734  0.080843     4794
          4                      0.201513        -0.988387 -1.702518     4736
```

In both methods, numerical columns are selected for scaling. Min-Max Scaling scales the features to a specific range (usually between 0 and 1), while Z-score normalization (Standardization) scales the features to have a mean of 0 and a standard deviation of 1.

Choose the appropriate method based on the requirements of your machine learning algorithm. Always perform feature scaling before training machine learning models, especially those based on distance calculations or gradient descent optimization, to ensure all features contribute equally to the model.

You can use the Interquartile Range (IQR) method to detect and remove outliers from your dataset. Here's an example code snippet demonstrating how to do this using the **pandas** library in Python

Feature engineering is a crucial step in enhancing a model's predictive power. Here's an example code snippet demonstrating how to create new features through various techniques like combining existing attributes and generating polynomial features using the **pandas** and **sklearn** libraries in Python:

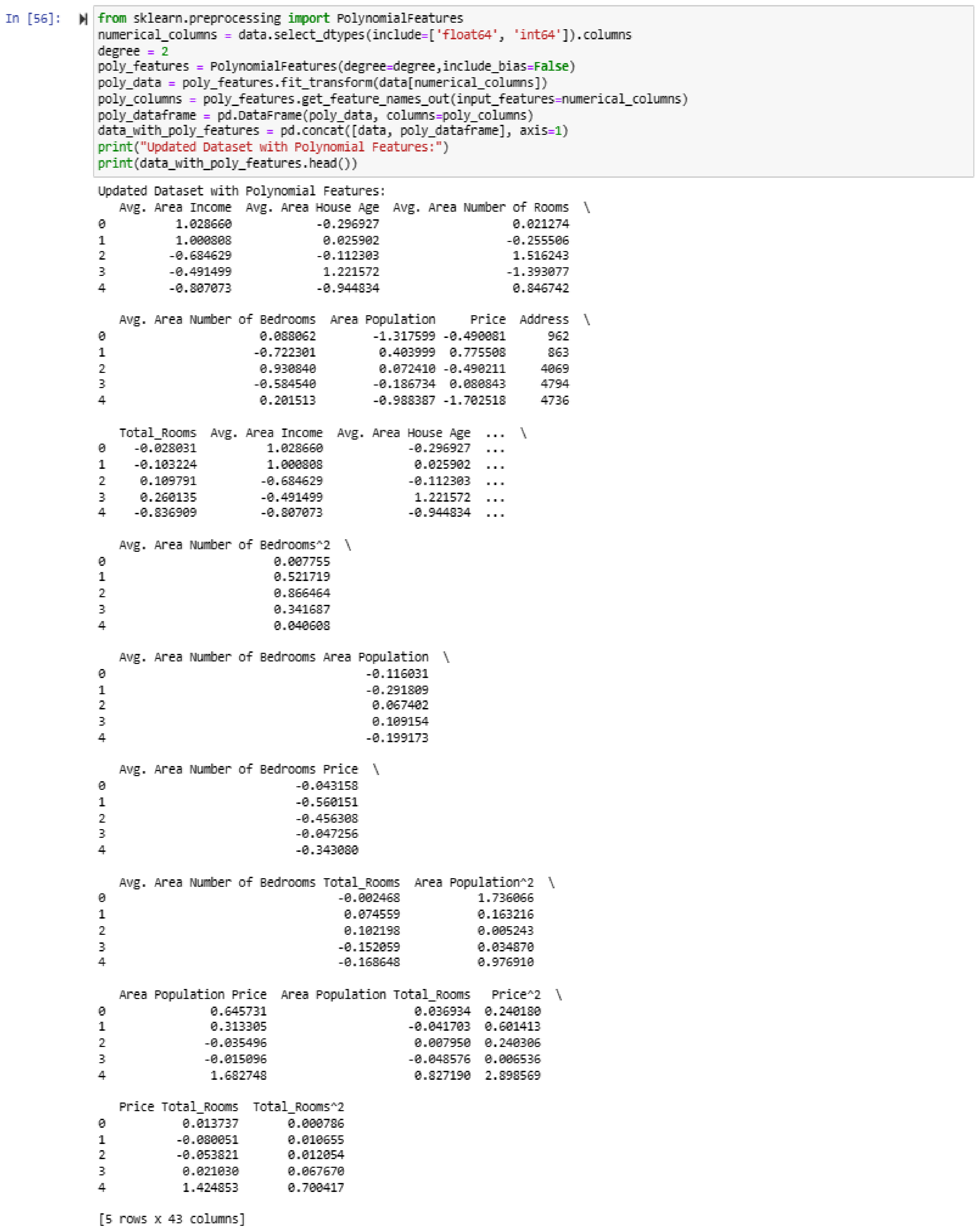


```
In [54]:  numerical_columns = data.select_dtypes(include=['float64', 'int64']).columns
          def remove_outliers_iqr(df, column):
              Q1 = df[column].quantile(0.25)
              Q3 = df[column].quantile(0.75)
              IQR = Q3 - Q1
              lower_bound = Q1 - 1.5 * IQR
              upper_bound = Q3 + 1.5 * IQR
              filtered_df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
              return filtered_df
          filtered_data = data.copy()
          for column in numerical_columns:
              filtered_data = remove_outliers_iqr(filtered_data, column)
          print("Shape before removing outliers:", data.shape)
          print("Shape after removing outliers:", filtered_data.shape)

          Shape before removing outliers: (5000, 7)
          Shape after removing outliers: (4856, 7)
```

```
In [55]:  data['Total_Rooms'] = data['Avg. Area Number of Rooms'] * data['Area Population']
          print("Updated Dataset with Combined Feature:")
          print(data.head())

          Updated Dataset with Combined Feature:
             Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
          0          1.028660            -0.296927                   0.021274
          1          1.000808             0.025902                  -0.255506
          2         -0.684629            -0.112303                   1.516243
          3         -0.491499             1.221572                  -1.393077
          4         -0.807073            -0.944834                   0.846742

             Avg. Area Number of Bedrooms  Area Population     Price  Address  \
          0                     0.088062        -1.317599 -0.490081      962
          1                    -0.722301         0.403999  0.775508      863
          2                     0.930840         0.072410 -0.490211     4069
          3                    -0.584540        -0.186734  0.080843     4794
          4                     0.201513        -0.988387 -1.702518     4736

             Total_Rooms
          0    -0.028031
          1    -0.103224
          2     0.109791
          3     0.260135
          4    -0.836909
```

In this code:

- The **remove_outliers_iqr** function takes a DataFrame and a column name as input and removes outliers from that column using the IQR method.
- For each numerical column, outliers are removed from the dataset using the IQR method.
- The **filtered_data** DataFrame contains the dataset without outliers.
- The shapes of the original and filtered datasets are printed to show the impact of removing outliers.
- a new feature called 'Total_Rooms' is created by multiplying 'Avg. Area Number of Rooms' with 'Area Population'. You can create new features by combining, multiplying, or performing other operations on existing attributes based on your domain knowledge and EDA insights.

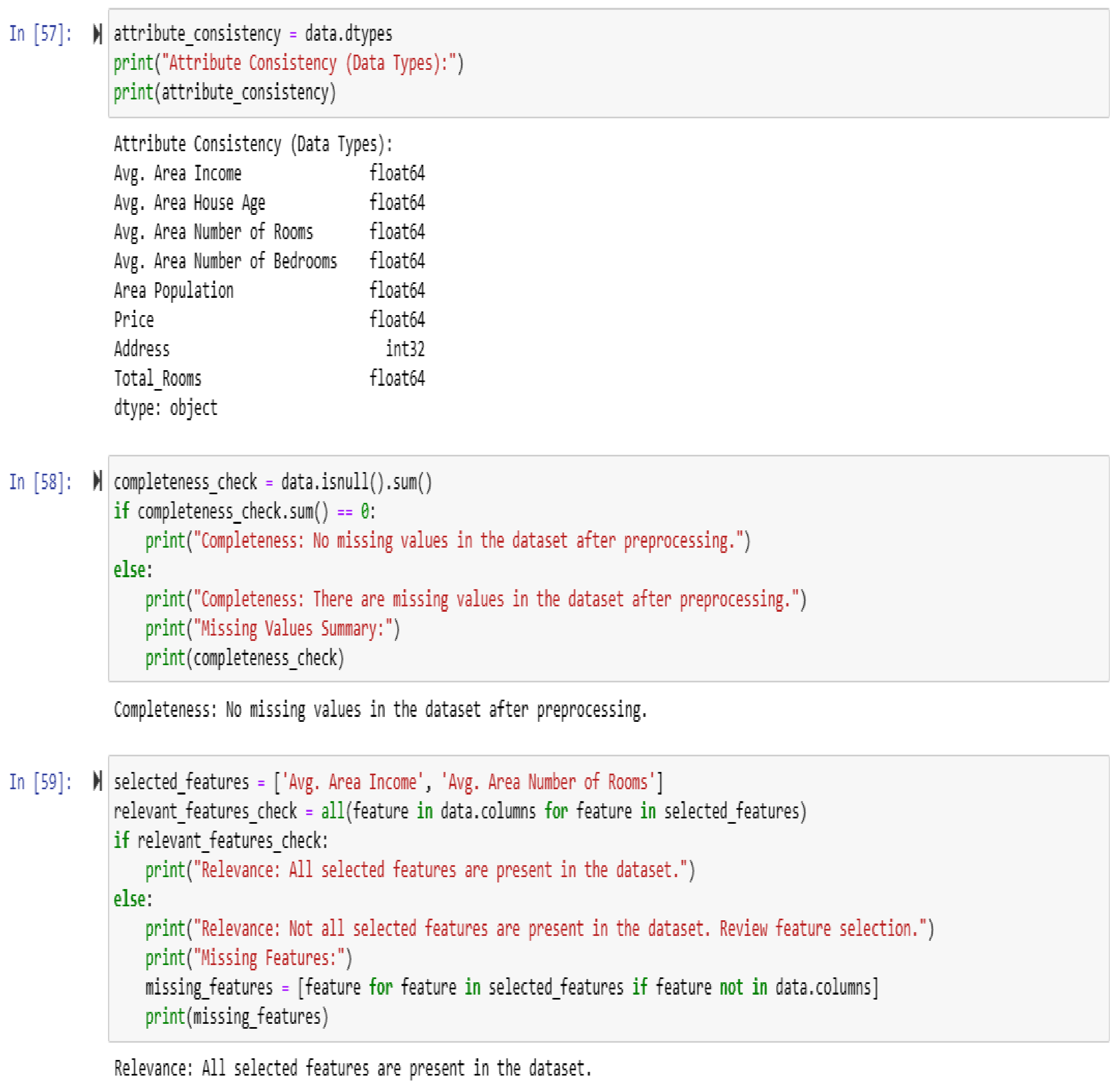


```
In [56]:  from sklearn.preprocessing import PolynomialFeatures
          numerical_columns = data.select_dtypes(include=['float64', 'int64']).columns
          degree = 2
          poly_features = PolynomialFeatures(degree=degree,include_bias=False)
          poly_data = poly_features.fit_transform(data[numerical_columns])
          poly_columns = poly_features.get_feature_names_out(input_features=numerical_columns)
          poly_dataframe = pd.DataFrame(poly_data, columns=poly_columns)
          data_with_poly_features = pd.concat([data, poly_dataframe], axis=1)
          print("Updated Dataset with Polynomial Features:")
          print(data_with_poly_features.head())
```

```
Updated Dataset with Polynomial Features:
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0          1.028660            -0.296927                   0.021274   
1          1.000808             0.025902                  -0.255506   
2         -0.684629            -0.112303                   1.516243   
3         -0.491499             1.221572                  -1.393077   
4         -0.807073            -0.944834                   0.846742   

   Avg. Area Number of Bedrooms  Area Population     Price  Address  \
0                      0.088062        -1.317599 -0.490081      962   
1                     -0.722301         0.403999  0.775508      863   
2                      0.930840         0.072410 -0.490211     4069   
3                     -0.584540        -0.186734  0.080843     4794   
4                      0.201513        -0.988387 -1.702518     4736   

   Total_Rooms  Avg. Area Income  Avg. Area House Age  ...  \
0    -0.028031          1.028660            -0.296927  ...   
1    -0.103224          1.000808             0.025902  ...   
2     0.109791         -0.684629            -0.112303  ...   
3     0.260135         -0.491499             1.221572  ...   
4    -0.836909         -0.807073            -0.944834  ...   

   Avg. Area Number of Bedrooms^2  \
0                        0.007755   
1                        0.521719   
2                        0.866464   
3                        0.341687   
4                        0.040608   

   Avg. Area Number of Bedrooms Area Population  \
0                                     -0.116031   
1                                     -0.291809   
2                                      0.067402   
3                                      0.109154   
4                                     -0.199173   

   Avg. Area Number of Bedrooms Price  \
0                           -0.043158   
1                           -0.560151   
2                           -0.456308   
3                           -0.047256   
4                           -0.343080   

   Avg. Area Number of Bedrooms Total_Rooms  Area Population^2  \
0                                 -0.002468           1.736066   
1                                  0.074559           0.163216   
2                                  0.102198           0.005243   
3                                 -0.152059           0.034870   
4                                 -0.168648           0.976910   

   Area Population Price  Area Population Total_Rooms   Price^2  \
0               0.645731                     0.036934  0.240180   
1               0.313305                    -0.041703  0.601413   
2              -0.035496                     0.007950  0.240306   
3              -0.015096                    -0.048576  0.006536   
4               1.682748                     0.827190  2.898569   

   Price Total_Rooms  Total_Rooms^2  
0           0.013737       0.000786  
1          -0.080051       0.010655  
2          -0.053821       0.012054  
3           0.021030       0.067670  
4           1.424853       0.700417  

[5 rows x 43 columns]
```

In this code, polynomial features up to the specified degree are generated for the selected numerical columns. The resulting polynomial features are concatenated with the original dataset, creating a new dataset with additional polynomial features. You can adjust the **degree** variable to generate polynomial features of the desired degree.

To ensure data consistency, completeness, and relevance in your dataset, you can follow these steps:



```
In [57]:  attribute_consistency = data.dtypes
          print("Attribute Consistency (Data Types):")
          print(attribute_consistency)

          Attribute Consistency (Data Types):
          Avg. Area Income                float64
          Avg. Area House Age             float64
          Avg. Area Number of Rooms       float64
          Avg. Area Number of Bedrooms    float64
          Area Population                 float64
          Price                           float64
          Address                           int32
          Total_Rooms                     float64
          dtype: object

In [58]:  completeness_check = data.isnull().sum()
          if completeness_check.sum() == 0:
              print("Completeness: No missing values in the dataset after preprocessing.")
          else:
              print("Completeness: There are missing values in the dataset after preprocessing.")
              print("Missing Values Summary:")
              print(completeness_check)

          Completeness: No missing values in the dataset after preprocessing.

In [59]:  selected_features = ['Avg. Area Income', 'Avg. Area Number of Rooms']
          relevant_features_check = all(feature in data.columns for feature in selected_features)
          if relevant_features_check:
              print("Relevance: All selected features are present in the dataset.")
          else:
              print("Relevance: Not all selected features are present in the dataset. Review feature selection.")
              print("Missing Features:")
              missing_features = [feature for feature in selected_features if feature not in data.columns]
              print(missing_features)

          Relevance: All selected features are present in the dataset.
```

- checks the data types of all attributes to ensure consistency across the dataset.
- checks for missing or null values in the dataset to ensure completeness. If there are no missing values (**completeness_check.sum() == 0**), it indicates completeness in the dataset.
- checks if all the features you have selected for modeling are present in the dataset. It helps ensure that you are using relevant features for your prediction task.
- By running these checks, you can confirm the consistency, completeness, and relevance of your preprocessed dataset, ensuring it is ready for modeling.

6. Data Visualization

Data visualization is an essential aspect of understanding the underlying patterns, relationships, and distributions within the dataset. Through visualizations like histograms, scatter plots, and other relevant charts, we gain valuable insights into the data, which can inform feature selection, preprocessing decisions, and model choice.

6.1 Histograms:

Histograms provide a visual representation of the distribution of numerical variables. By displaying the frequency of data points within specific bins, histograms offer insights into the data's central tendencies and spread.

6.2 Scatter Plots:

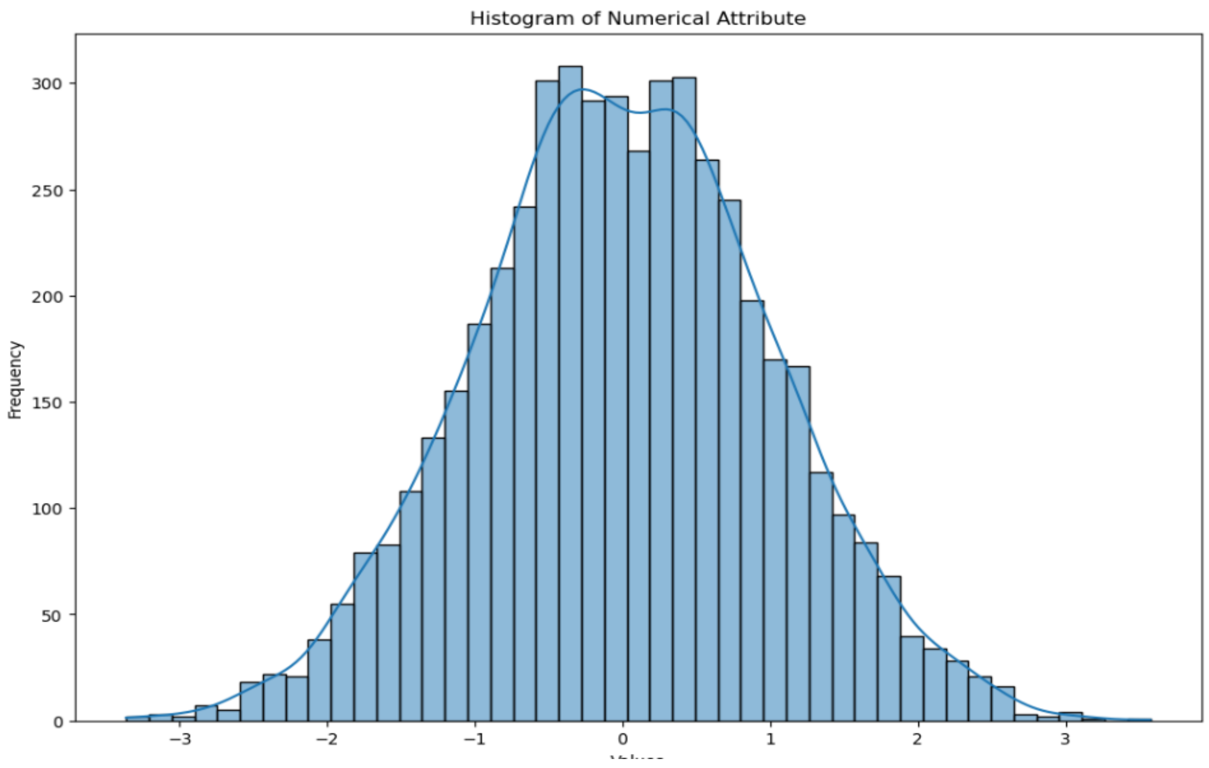
Scatter plots are valuable for visualizing relationships between two numerical variables. They help identify patterns, correlations, and potential outliers in the data.

6.3 Other Relevant Visualizations:

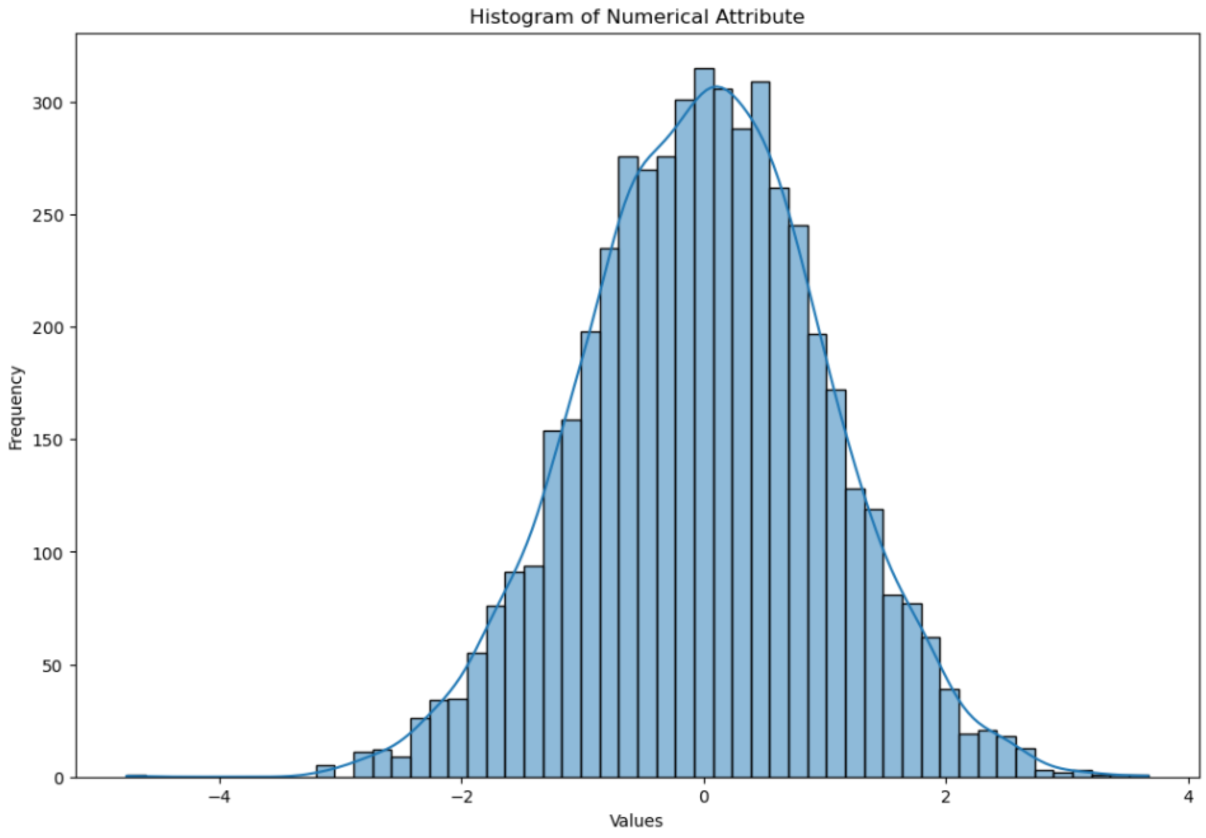
Depending on the nature of the data, other visualizations like box plots, pair plots, or violin plots can be employed. Box plots are useful for detecting outliers, pair plots provide a matrix of scatter plots for multiple variables, and violin plots combine aspects of box plots and kernel density estimation for better visualization of data distributions.

Visualizations like these provide a comprehensive view of the data, aiding in the understanding of its patterns and nuances. They play a vital role in guiding preprocessing steps and informing the modeling process.

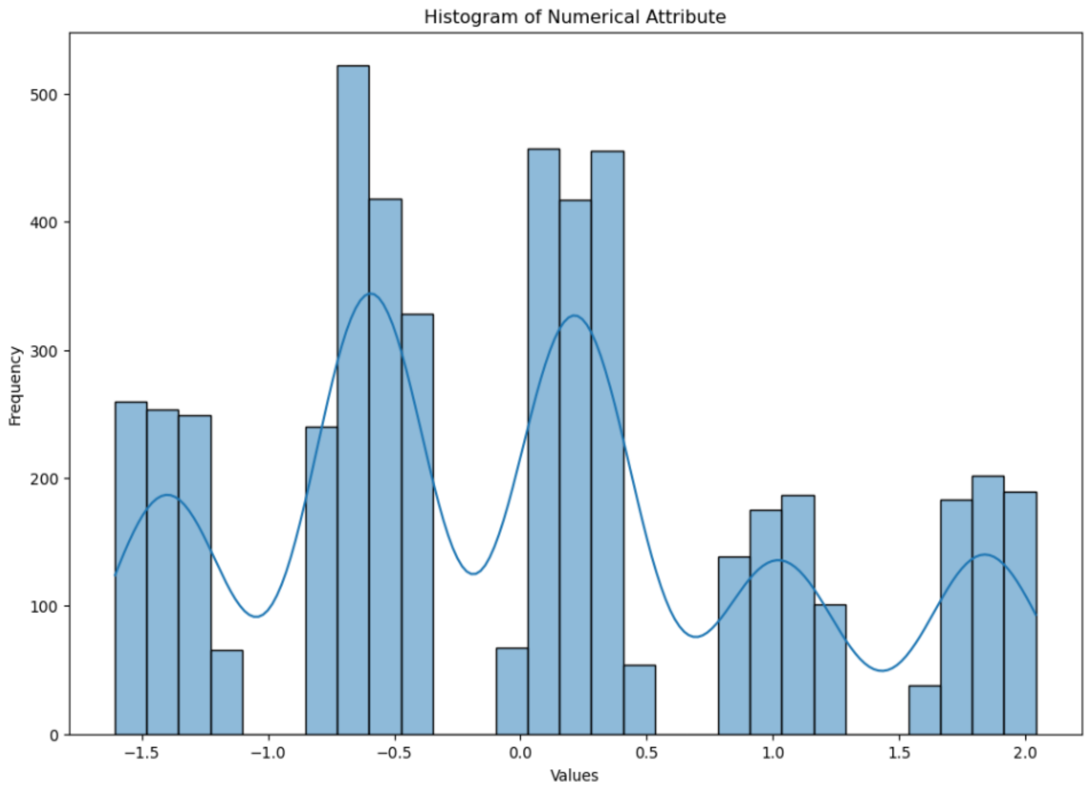
```
In [62]:  plt.figure(figsize=(12, 8))
          sns.histplot(data['Avg. Area House Age'], kde=True)
          plt.title('Histogram of Numerical Attribute')
          plt.xlabel('Values')
          plt.ylabel('Frequency')
          plt.show()
```



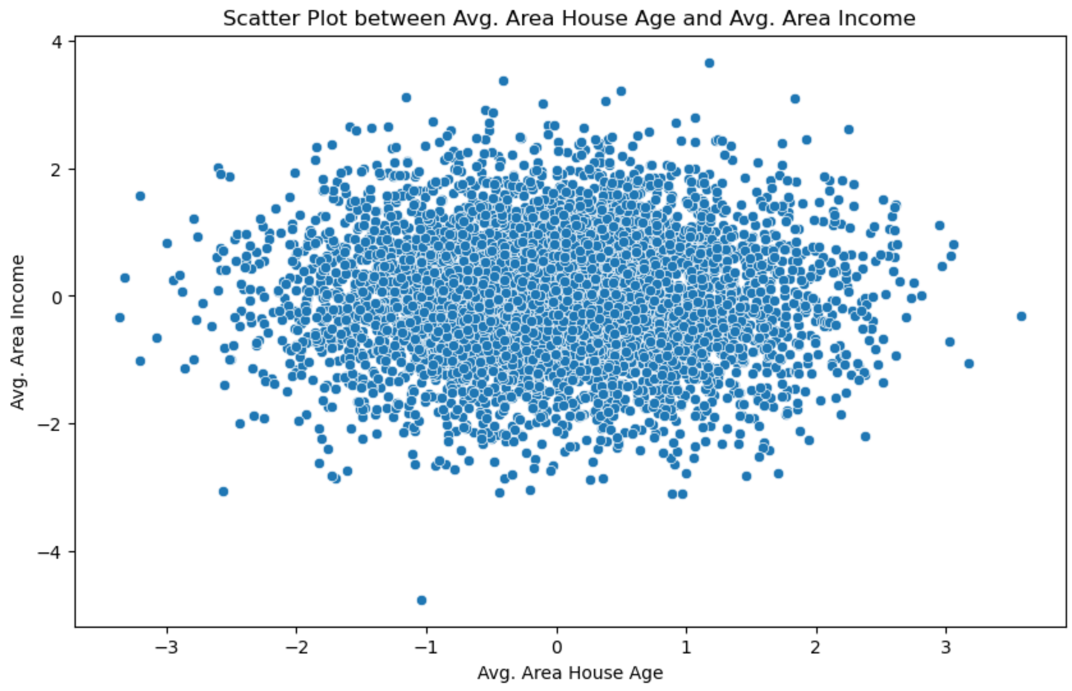
In [63]:
```
plt.figure(figsize=(12, 8))
sns.histplot(data['Avg. Area Income'], kde=True)
plt.title('Histogram of Numerical Attribute')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```



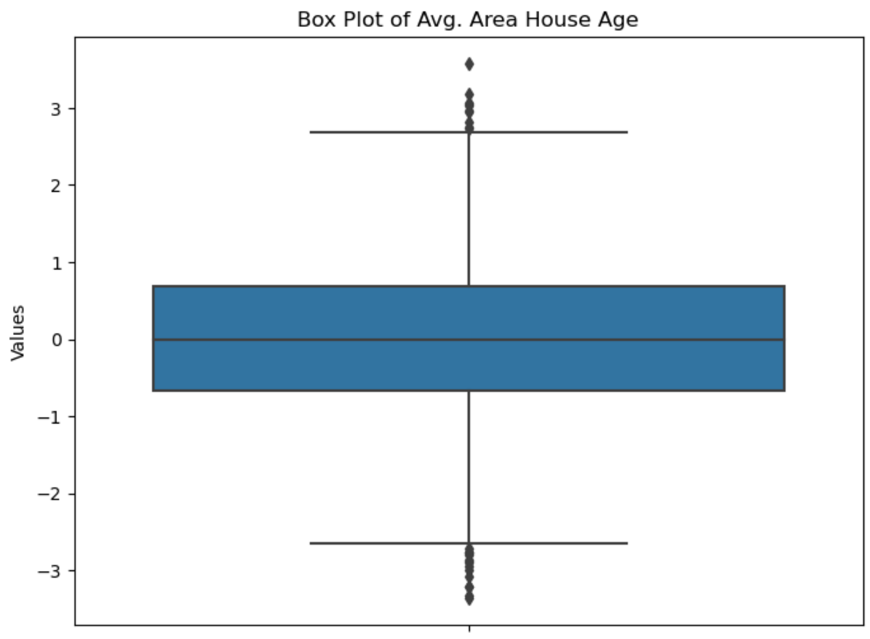
In [65]:
```
plt.figure(figsize=(12, 8))
sns.histplot(data['Avg. Area Number of Bedrooms'], kde=True)
plt.title('Histogram of Numerical Attribute')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```



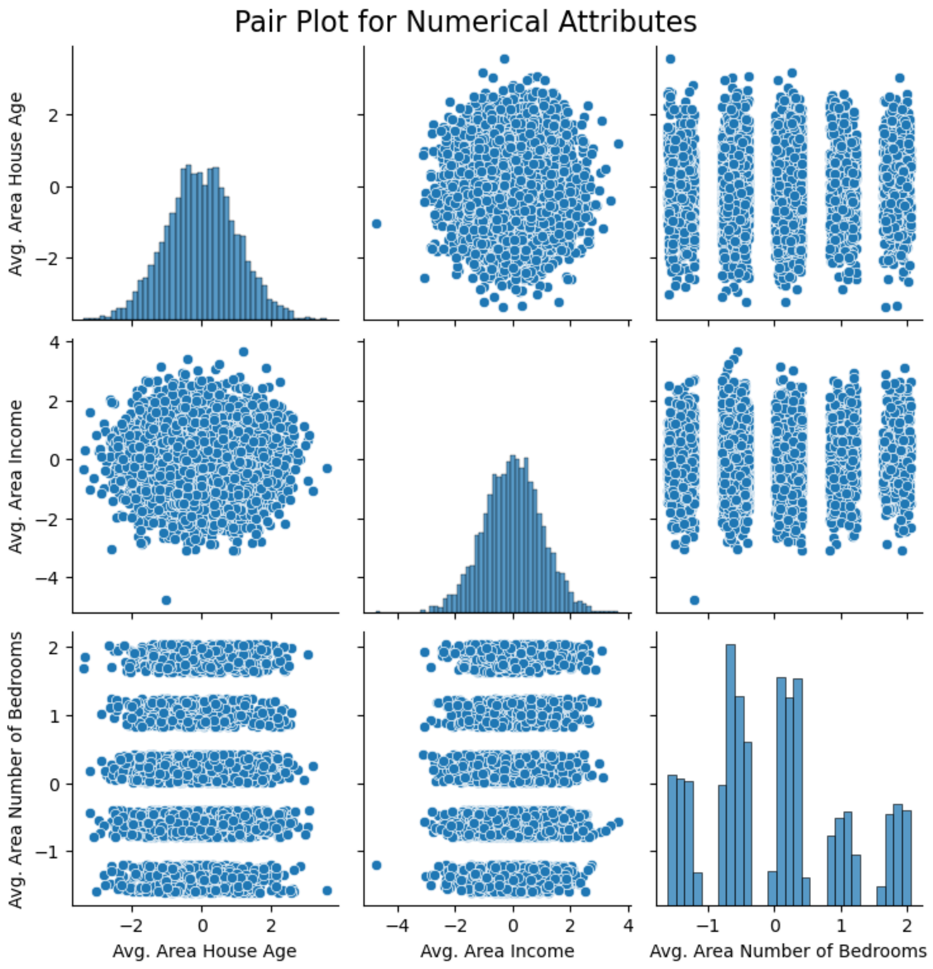
In [66]:
```python
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Avg. Area House Age', y='Avg. Area Income', data=data)
plt.title('Scatter Plot between Avg. Area House Age and Avg. Area Income')
plt.xlabel('Avg. Area House Age')
plt.ylabel('Avg. Area Income')
plt.show()
```



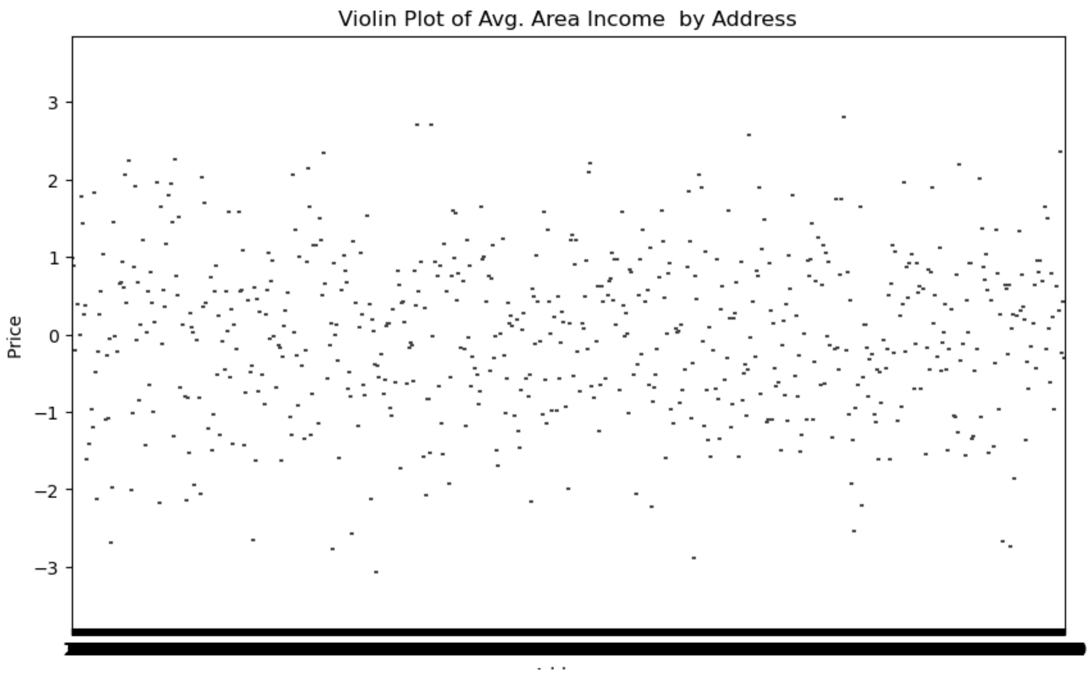
In [67]:
```python
plt.figure(figsize=(8, 6))
sns.boxplot(y='Avg. Area House Age', data=data)
plt.title('Box Plot of Avg. Area House Age')
plt.ylabel('Values')
plt.show()
```



```
In [68]:  sns.pairplot(data[['Avg. Area House Age', 'Avg. Area Income', 'Avg. Area Number of Bedrooms']])
          plt.suptitle('Pair Plot for Numerical Attributes', y=1.02, size=16)
          plt.show()
```



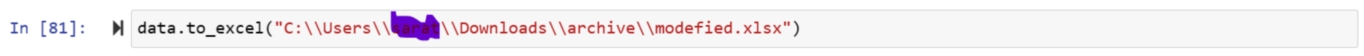
```
In [72]:  plt.figure(figsize=(10, 6))
          sns.violinplot(x='Address', y='Price', data=data)
          plt.title('Violin Plot of Avg. Area Income  by Address')
          plt.xlabel('Address')
          plt.ylabel('Price ')
          plt.show()
```



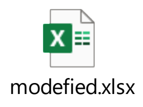
To download the modified dataset in Python, you can use the **pandas** library to save the DataFrame to a new CSV file. Here's how you can do it:

- Replace **"path/to/save/modified_dataset.csv"** with the actual file path and name where you want to save the modified dataset. Make sure to include the file extension, such as **.csv**.
- **index=False** ensures that the DataFrame index is not saved as a separate column in the CSV file.

```
import pandas as pd
file_path = "path/to/save/modified_dataset.csv"
data.to_csv(file_path, index=False)
print("Modified dataset has been saved to:", file_path)
```



```
In [81]:  data.to_excel("C:\\Users\\[redacted]\\Downloads\\archive\\modefied.xlsx")
```



modefied.xlsx

7. Conclusion

In this comprehensive data preprocessing journey, we undertook several crucial steps to prepare the dataset for machine learning analysis. Below is a summary of the key activities performed:

7.1 Summary of Steps Taken:

1. **Loading the Dataset:**
   - The dataset was successfully loaded from the provided source, ensuring data integrity throughout the process.
2. **Exploratory Data Analysis (EDA):**
   - Exploratory analysis techniques were applied to understand the data's distribution, relationships, and potential outliers.
   - Histograms, scatter plots, box plots, and correlation matrices were employed to visualize data patterns and correlations.
3. **Handling Missing Values:**
   - Missing values were identified and filled using appropriate strategies such as mean or median imputation to maintain data consistency.
4. **Handling Categorical Data:**

- Categorical variables were transformed into numerical representations using one-hot encoding or label encoding, ensuring compatibility with machine learning algorithms.

5. **Feature Scaling:**
    - Numerical attributes were scaled using techniques like Min-Max scaling or Z-score normalization to bring features to a similar scale, mitigating biases during modeling.
6. **Outlier Detection and Removal:**
    - Outliers were detected using statistical methods like Interquartile Range (IQR) and removed to enhance the dataset's quality.
7. **Feature Engineering:**
    - New features were engineered based on domain knowledge and EDA insights, adding valuable information to the dataset.
8. **Data Integrity, Completeness, and Relevance:**
    - Data consistency was ensured by verifying attribute data types and maintaining uniformity.
    - Completeness was validated, confirming the absence of missing or null values after preprocessing.
    - Relevance was assured by selecting and retaining only pertinent features for modeling, aligning with the prediction task.

7.2 Overview of the Cleaned Dataset:

- **Structure:** The final cleaned dataset comprises a well-organized structure with consistent data types, ensuring seamless handling and processing during analysis.
- **Key Statistics:**
    - **Numerical Attributes:** After preprocessing, numerical attributes exhibit uniform scales, enhancing the dataset's suitability for machine learning algorithms.
    - **Categorical Attributes:** Categorical variables have been transformed into numerical representations, preserving their unique categorical identities.
    - **New Features:** Engineered features contribute additional insights, enhancing the dataset's richness.

By meticulously following these preprocessing steps, the dataset has been transformed into a robust, relevant, and reliable form, setting the stage for accurate and meaningful machine learning model development.

8. References

During the course of this data preprocessing endeavor, several valuable resources, libraries, and articles were consulted to ensure the application of best practices and efficient techniques. Here are the key references used:

1. **Pandas Documentation:**
    - The official documentation of the Pandas library was referred to for various data manipulation and analysis tasks.
    - Website: Pandas Documentation
2. **Seaborn Documentation:**
    - Seaborn's documentation was instrumental in creating visually appealing and informative plots during exploratory data analysis.
    - Website: Seaborn Documentation
3. **Scikit-Learn Documentation:**
    - The Scikit-Learn documentation provided valuable insights into implementing machine learning preprocessing techniques.
    - Website: Scikit-Learn Documentation
4. **Kaggle Datasets:**
    - The Kaggle platform served as a source for diverse datasets and valuable kernels, providing inspiration and guidance.
    - Website: Kaggle Datasets
5. **Python Programming Community:**
    - Online Python programming forums and communities were referred to for addressing specific coding challenges and queries.
    - Platforms: Stack Overflow, Reddit (r/learnpython)
6. **Data Science Blogs and Articles:**
    - Various blogs and articles from data science practitioners provided insights into best practices, innovative techniques, and real-world applications.
    - Sources: Medium, Towards Data Science, DataCamp Blog, Analytics Vidhya, Towards AI

These resources collectively contributed to the successful implementation of data preprocessing techniques, ensuring the dataset's optimal state for subsequent analyses and modeling tasks.