MACHINE LEARNING PROJECT

Project Name: House Price Prediction using Machine Learning in Python

By- Shreya Vishe

Under the guidance of: Mr. Sameer Warsolkar

Introduction

Thousands of houses are sold everyday. There are some questions every buyer asks himself like: What is the actual price that this house deserves? Am I paying a fair price? In this poject, a machine learning model is proposed to predict a house price based on data related to the house and its location. During the development and evaluation of our model, we will show the code used for each step followed by its output. This will facilitate the reproducibility of our work. In this study, Python programming language with a number of Python packages will be used.

Objective

Predict the selling price of houses based on various features using a machine learning model.

The dataset contains 7 features

Avg. Area Income: The average income of residents in a specific area.

Avg. Area House Age: The average age of houses in a particular area.

Avg. Area Number of Rooms: The average number of rooms in houses in the area.

Avg. Area Number of Bedrooms: The average number of bedrooms in houses in the area.

Area Population: The population of the area.

Price: The target variable to be predicted, likely representing the price of houses in the area.

Address: The address of the houses in the dataset.

Importing Libraries and Dataset

Importing libraries provides ready-made tools for data manipulation, analysis, and visualization, while importing datasets allows you to explore, clean, and analyze data in your programming environment. It's a fundamental step for efficient and effective data analysis and machine learning.

In [1]:
```
#Importing Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

Here we are using

Numpy: For efficient data representation, manipulation, and mathematical operations.

Pandas: To load the data.

Matplotlib: To visualize the data features.

Seaborn: To see the correlation between features using heatmap.

Warnings: To suppress all warnings generated by program code.

Reading the Data

In [2]:
```
df = pd.read_csv("USA_Housing.csv")
df
```

Out[2]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386
...	...	...	...	...	...	...	...
4995	60567.944140	7.830362	6.137356	3.46	22837.361035	1.060194e+06	USNS Williams\nFPO AP 30153-7653
4996	78491.275435	6.999135	6.576763	4.02	25616.115489	1.482618e+06	PSC 9258, Box 8489\nAPO AA 42991-3352
4997	63390.686886	7.250591	4.805081	2.13	33266.145490	1.030730e+06	4215 Tracy Garden Suite 076\nJoshualand, VA 01...
4998	68001.331235	5.534388	7.130144	5.44	42625.620156	1.198657e+06	USS Wallace\nFPO AE 73316
4999	65510.581804	5.992305	6.792336	4.07	46501.283803	1.298950e+06	37778 George Ridges Apt. 509\nEast Holly, NV 2...

5000 rows × 7 columns

In [3]:
```
#To get the first five rows from the dataset

df.head()
```

Out[3]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

In [4]:
```
#To get the last five rows from the dataset

df.tail()
```

Out[4]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
4995	60567.944140	7.830362	6.137356	3.46	22837.361035	1.060194e+06	USNS Williams\nFPO AP 30153-7653
4996	78491.275435	6.999135	6.576763	4.02	25616.115489	1.482618e+06	PSC 9258, Box 8489\nAPO AA 42991-3352
4997	63390.686886	7.250591	4.805081	2.13	33266.145490	1.030730e+06	4215 Tracy Garden Suite 076\nJoshualand, VA 01...
4998	68001.331235	5.534388	7.130144	5.44	42625.620156	1.198657e+06	USS Wallace\nFPO AE 73316
4999	65510.581804	5.992305	6.792336	4.07	46501.283803	1.298950e+06	37778 George Ridges Apt. 509\nEast Holly, NV 2...

In [5]:
```
#Data Preprocessing (To categorize the features depending on their datatype (int, float, object) ar

categorical_cols = list(df.select_dtypes(include=['object']).columns)
print("Categorical variables:", len(categorical_cols))

int_cols = list(df.select_dtypes(include=['int64']).columns)
print("Integer variables:", len(int_cols))

fl_cols = list(df.select_dtypes(include = ['float']).columns)
print("Float variables:",len(fl_cols))
```

```
Categorical variables: 1
Integer variables: 0
Float variables: 6
```

Here, we can see the dataset is having 1 categorical variable, no integer, and 6 float.

EDA - Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in data analysis that involves exploring and visualizing a dataset to understand its main characteristics, patterns, and trends. EDA helps in making informed decisions about data cleaning, feature engineering, and the direction of further analysis.

In [6]:
```
#To get the information about the data

df.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
```

Here, we can see there are 7 attributes in total out of which 64 are float and 1 object.

In [7]:
```
#Handeling missing values

df.isnull().sum()
```

Out[7]:
```
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
```

As we can see, there are no null values present in this dataset.

In [8]:
```
#To obtain the shape of a dataframe

df.shape
```

Out[8]:
```
(5000, 7)
```

There are 7 columns and 5000 rows in this dataset.

In [9]:
```
#To display the statistical information of the data

df.describe()
```

Out[9]:

	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%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

In [10]:
```
# To get the information about the data

df.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
```

Here, we can see there are 7 attributes in total out of which 64 are float and 1 object.

In [11]:
```
#Handeling missing values

df.isnull().sum()
```

Out[11]:
```
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
```

As we can see here, there are no missing values in this dataset.

In [12]:
```
#To get the information about the data

df.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
```

Here, we can see there are 7 attributes in total out of which 64 are float and 1 object.

In [13]:
```
#To compute the pairwise correlation of columns in a DataFrame

df.corr()
```

Out[13]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
Avg. Area Income	1.000000	-0.002007	-0.011032	0.019788	-0.016234	0.639734
Avg. Area House Age	-0.002007	1.000000	-0.009428	0.006149	-0.018743	0.452543
Avg. Area Number of Rooms	-0.011032	-0.009428	1.000000	0.462695	0.002040	0.335664
Avg. Area Number of Bedrooms	0.019788	0.006149	0.462695	1.000000	-0.022168	0.171071
Area Population	-0.016234	-0.018743	0.002040	-0.022168	1.000000	0.408556
Price	0.639734	0.452543	0.335664	0.171071	0.408556	1.000000

Lable Encoding

A method in machine learning to convert categorical data into numerical form by assigning a unique integer to each category. It's useful for preprocessing data before feeding it into algorithms that require numerical input.

In [14]:
```
#Importing Lable Encoder

from sklearn.preprocessing import LabelEncoder, MinMaxScaler
```

Here we are using

Sklearn:Python library for machine learning, offering a versatile set of tools and consistent interfaces for building and evaluating machine learning models.

Sklearn.preprocessing: A module in scikit-learn that offers tools for preparing data before using it in machine learning models. It includes functions for scaling, encoding categorical variables, handling missing data, and more.

Lable Encoder: To normalize labels.

Min Max Scaler:A normalization technique used in machine learning to scale and transform numerical features within a specific range, usually between 0 and 1.

In [15]:
```
#Selecting the columns in the DataFrame with data type object

catdata=df.select_dtypes(object)
```

In [16]:
```
#To get information about the categorical features present in the dataset

cat = df.select_dtypes(object).columns
object_cols = list(cat)
print("Categorical variables:",object_cols)
print('No. of categorical features: ', len(object_cols))
```

```
Categorical variables: ['Address']
No. of categorical features:  1
```

Here we have only categorical variables i.e. 'Address'

In [17]:
```
#To display the names of the categorical columns in your DataFrame

object_cols
```

Out[17]:
```
['Address']
```

In [18]:
```
#Assuming 'object_cols' contains the names of categorical columns

le = LabelEncoder()
for i in object_cols:
    df[i]=le.fit_transform(df[i])
le
```

Out[18]:
```
LabelEncoder()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

The categorical variables in a DataFrame are converted into numerical labels

In [19]:
```
#To get the information about the data

df.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   int32  
dtypes: float64(6), int32(1)
memory usage: 254.0 KB
```

After converting the categorical column into numerical we have 6 float and 1 integer

In [20]:
```
#Splitting the data into X and Y

from sklearn.model_selection import train_test_split
```

Here we are using

sklearn.model_selection:Essential function for assessing and improving the performance of machine learning models.

train_test_split: To split a dataset into training and testing sets, facilitating the evaluation of machine learning models.

In [21]:
```
#Features (independent variables)

X = df.drop(['Price'], axis=1)
X
```

Out[21]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	962
1	79248.642455	6.002900	6.730821	3.09	40173.072174	863
2	61287.067179	5.865890	8.512727	5.13	36882.159400	4069
3	63345.240046	7.188236	5.586729	3.26	34310.242831	4794
4	59982.197226	5.040555	7.839388	4.23	26354.109472	4736
...	...	...	...	...	...	...
4995	60567.944140	7.830362	6.137356	3.46	22837.361035	4750
4996	78491.275435	6.999135	6.576763	4.02	25616.115489	4636
4997	63390.686886	7.250591	4.805081	2.13	33266.145490	1897
4998	68001.331235	5.534388	7.130144	5.44	42625.620156	4833
4999	65510.581804	5.992305	6.792336	4.07	46501.283803	1703

5000 rows × 6 columns

In [22]:
```
#Target variable (dependent variable)

Y = df['Price']
Y
```

Out[22]:
```
0       1.059034e+06
1       1.505891e+06
2       1.058988e+06
3       1.260617e+06
4       6.309435e+05
            ...
4995    1.060194e+06
4996    1.482618e+06
4997    1.030730e+06
4998    1.198657e+06
4999    1.298950e+06
Name: Price, Length: 5000, dtype: float64
```

In [23]:
```
# Split the training set into training and testing set

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
print(X_train, X_test)
print(Y_train, Y_test)
```

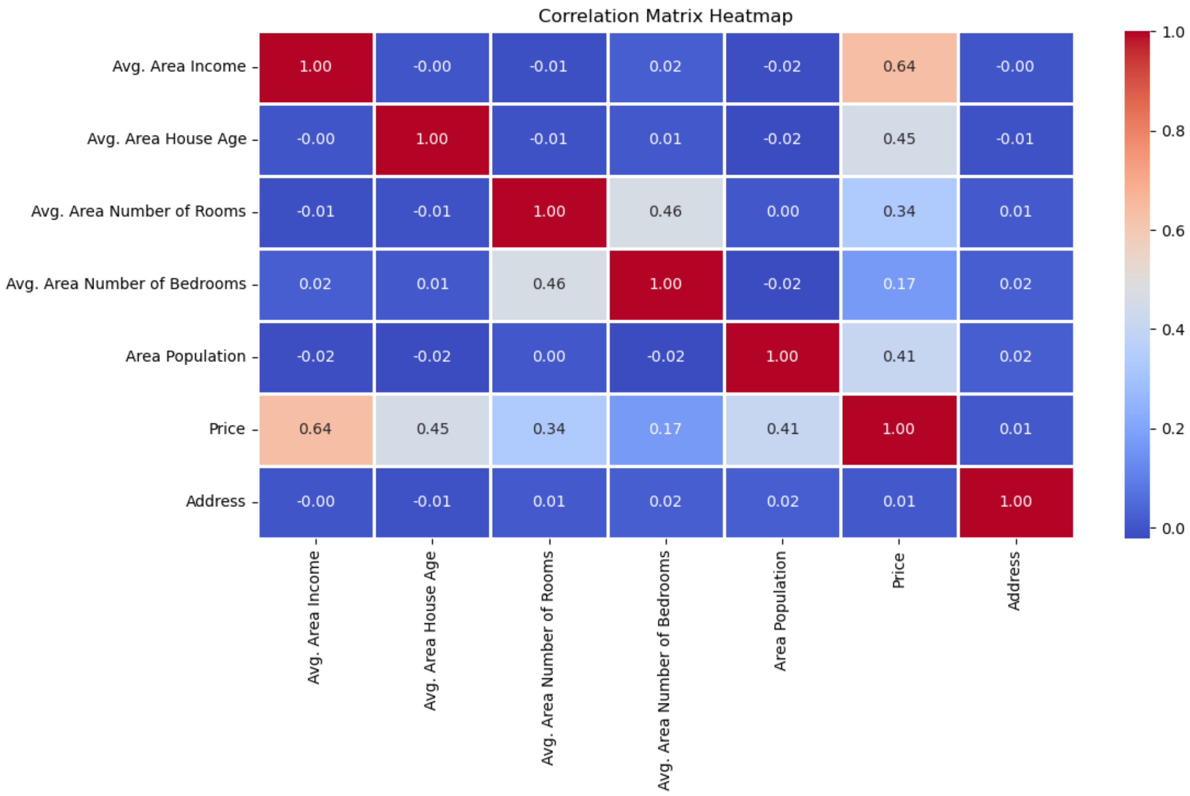
```
      Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
2913      80196.242251             6.675697                   7.275193   
3275      74130.606324             6.919663                   8.266994   
775       67384.000373             7.224281                   7.809919   
217       59569.537340             6.279537                   7.325380   
1245      58385.215373             7.588559                   6.406118   
...                ...                  ...                        ...   
4931      77622.958116             6.738014                   6.043040   
3264      80051.847123             5.872678                   6.019018   
1653      67094.197072             5.346437                   7.374607   
2607      52541.319847             4.885243                   7.225522   
2732      86762.882864             6.530193                   5.106962   

      Avg. Area Number of Bedrooms  Area Population  Address  
2913                          3.17     48694.864144     4800  
3275                          3.24     49958.580994     4126  
775                           6.43     48918.055356     4179  
217                           4.24     31294.652460     3367  
1245                          2.30     41930.375009      777  
...                            ...              ...      ...  
4931                          3.34     51102.441950     1997  
3264                          3.39     35254.128316     2771  
1653                          4.18     30022.537173      280  
2607                          3.20     41258.262292     1822  
2732                          2.09     47724.581355     2199  

[4000 rows x 6 columns]       Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
398       61200.726175             5.299694                   6.234615   
3833      63380.814670             5.344664                   6.001574   
4836      71208.269301             5.300326                   6.077989   
4572      50343.763518             6.027468                   5.160240   
636       54535.453719             5.278065                   6.871038   
...                ...                  ...                        ...   
4228      72472.366736             5.801879                   5.374962   
2367      58909.313436             5.714293                   7.703920   
788       49424.267124             7.053473                   5.110956   
1452      70138.512558             6.319457                   6.599789   
3265      69835.563996             6.419843                   7.670983   

      Avg. Area Number of Bedrooms  Area Population  Address  
398                           4.23     42789.692217     2034  
3833                          2.45     40217.333577     1574  
4836                          4.01     25696.361741     3544  
4572                          4.35     27445.876739     1804  
636                           4.41     30852.207006      679  
...                            ...              ...      ...  
4228                          2.45     19745.492789     2657  
2367                          6.38     40865.817888     4803  
788                           2.27     18656.642432      134  
1452                          4.37     33434.112589     4521  
3265                          3.03     19376.318935     2100  

[1000 rows x 6 columns]
2913    1.616937e+06
3275    1.881075e+06
775     1.930344e+06
217     8.859206e+05
1245    1.266210e+06
            ...     
4931    1.599997e+06
3264    1.354609e+06
1653    1.202993e+06
2607    8.429859e+05
2732    1.571254e+06
Name: Price, Length: 4000, dtype: float64 398     8.942511e+05
3833    9.329794e+05
4836    9.207479e+05
4572    6.918549e+05
636     7.327332e+05
            ...     
4228    7.549606e+05
2367    1.205568e+06
788     6.682555e+05
1452    1.398760e+06
3265    1.277381e+06
Name: Price, Length: 1000, dtype: float64
```

Splitting the training set into training and testing sets is essential for evaluating how well a machine learning model performs on new, unseen data. It helps prevent overfitting, allows for model evaluation, and supports hyperparameter tuning.

Data Visualization

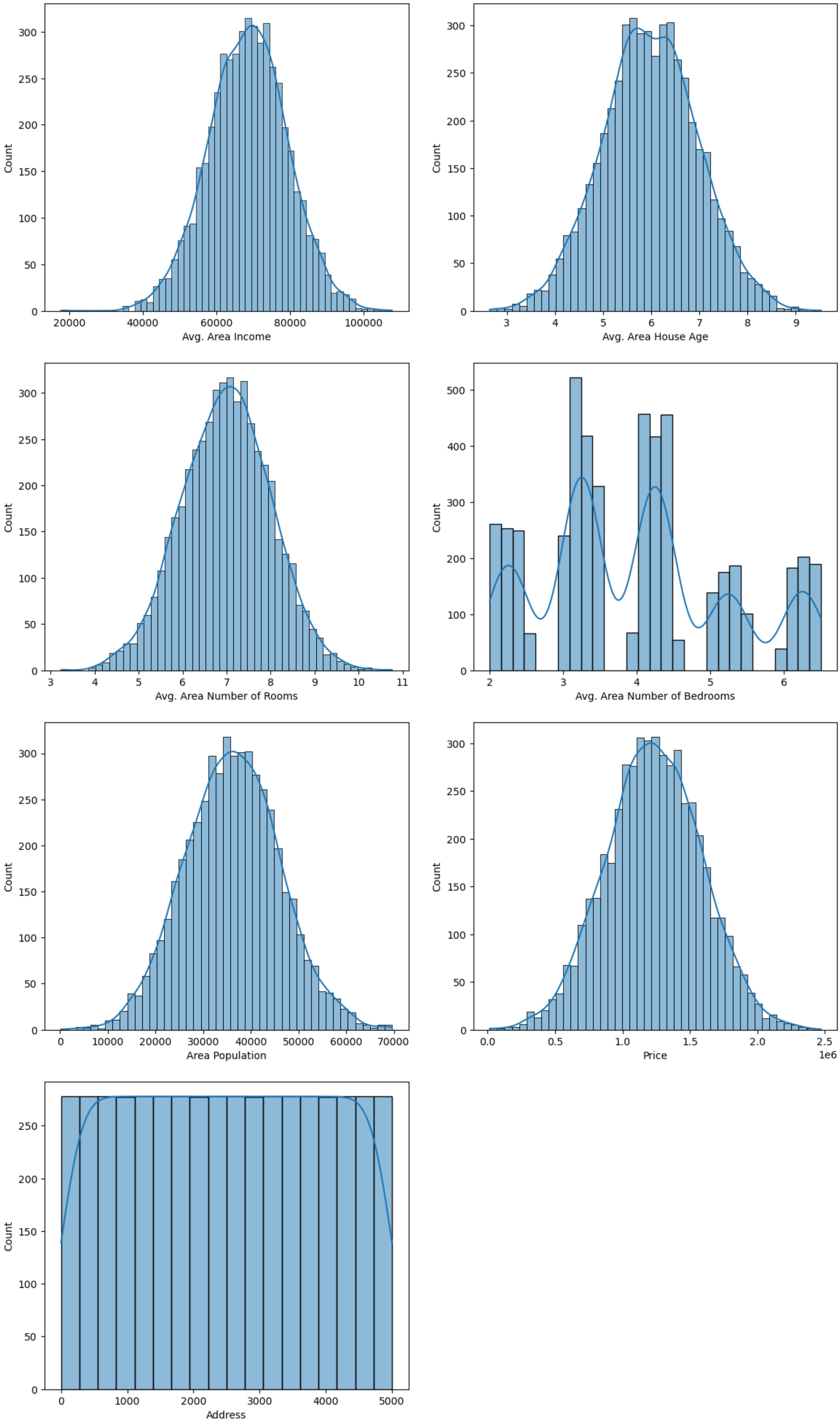
Essential because it transforms raw data into a format that is accessible, understandable, and actionable. It plays a crucial role in the data analysis process, helping individuals at various levels of expertise derive insights, make informed decisions, and communicate findings effectively.

In [24]:
```
plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(),
            cmap='coolwarm',
            annot=True,
            fmt='.2f',
            linewidths=2)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



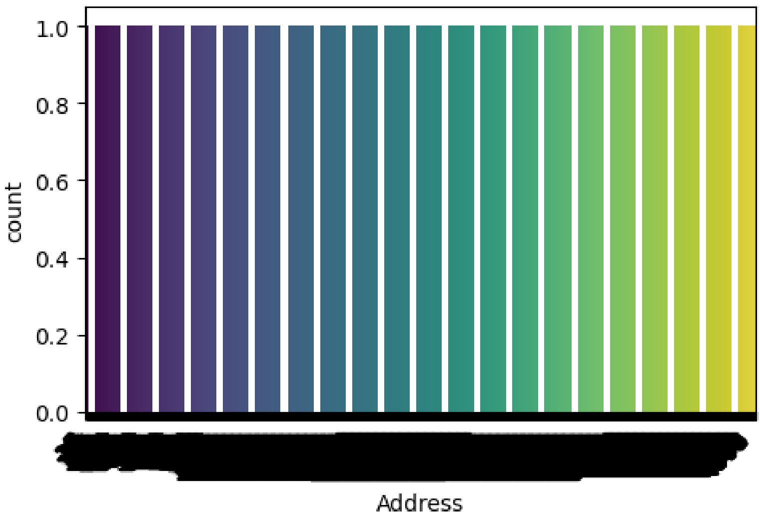
Here we have used Heat maps, primarily to visually represent patterns, relationships, and variations in data.

In [28]:
```
data_num=df.columns
plt.figure(figsize=(12,20))

for i, col in enumerate(data_num):

    plt.subplot((len(data_num) + 1) // 2, 2, i+1 )
    sns.histplot(x=col,data=df,kde=True)
plt.tight_layout(pad = 2)
```


Count
Avg. Area Income
Count
Avg. Area House Age
Count
Avg. Area Number of Rooms
Count
Avg. Area Number of Bedrooms
Count
Area Population
Count
Price
1e6
Count
Address

We have used Histograms as it helps in the data exploration and analysis phase.

In [27]:
```
plt.figure(figsize=(18, 36))
plt.title('Categorical Features: Distribution')
plt.xticks(rotation=90)
index = 1

for index, col in enumerate(categorical_cols, start=1):
    plt.subplot(11, 4, index)
    plt.xticks(rotation=45, ha='right')
    sns.countplot(x=col, data=df, palette='viridis')

plt.tight_layout()
plt.show()
```



We have used Subplots for visualizing multiple aspects of the data or model performance simultaneously.

Data Scaling

Data scaling is crucial in machine learning to ensure fair contributions from all features, speed up training, and improve model performance by reducing sensitivity to initial conditions and facilitating optimization algorithms.

In [29]:
```
#Data normalization with sklearn

from sklearn.preprocessing import MinMaxScaler
```

Normalizing data is a common preprocessing step in machine learning, and it refers to scaling the features to a standard range.

In [30]:
```
#Fit scaler on training data

minmax = MinMaxScaler()
X_train = minmax.fit_transform(X_train)
X_test = minmax.transform(X_test)
```

To insure that the training (X_train) and testing (X_test) data using the same MinMaxScaler instance for consistency.

In [31]:
```
#Libraries Imported

from sklearn.metrics import mean_absolute_percentage_error, r2_score
```

Here we are using

sklearn.metrics:Provides a wide range of tools for evaluating the performance of machine learning models.

mean_absolute_percentage_error: Measures the mean absolute percentage difference between true and predicted values essential for assessing accuracy, especially in forecasting.

r2_score:Computes the coefficient of determination (R-squared) to evaluate how well predictions approximate true values ued for measuring the proportion of variance explained.

Linear Regression

Linear Regression is a statistical method that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation. It aims to find the best-fit line that minimizes the sum of squared differences between predicted and actual values.

In [32]:
```
#Libraries Imported

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Here we are using

mean_squared_error: Measures the average squared difference between predicted and actual values.

In [33]:
```
#Assuming X, Y are your features and target

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

Dividing the dataset (X for features, Y for target) into training (X_train, Y_train) and testing (X_test, Y_test) sets. Before creating a linear regreesion model.

In [34]:
```
#Create a linear regression model

linear_model = LinearRegression()
linear_model.fit(X_train, Y_train)
linear_model
```

Out[34]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [35]:
```
#Making predictions

Y_pred = linear_model.predict(X_test)
Y_pred
```

Out[35]:
```
array([1308873.02491858, 1236482.91146815, 1244191.16222025,
       1229581.5060611 , 1062596.71654063, 1543367.06912373,
       1095539.51788942,  832757.4559343 ,  788938.670355  ,
       1470466.04894102,  671478.10781278, 1606804.67125503,
       1004213.10142048, 1797057.57501641, 1288684.09320991,
       1087010.23781192, 1423759.49046864, 1077921.96079049,
        802662.19752494,  930444.54059401, 1135261.66338185,
        916446.32317661, 1490429.16850068, 1284673.64996594,
       1581573.93519775, 1131714.43155715, 1089744.85030452,
        974597.05104371,  923520.71411839, 1740965.15866107,
       1286506.89116102, 1620891.7100839 , 1435620.27949892,
       1234692.13747052, 1485275.19786486, 1718584.16340224,
       1539390.43732017,  777228.63143043, 1764745.43562396,
       1175680.34037078, 1553245.99825664,  897085.10446464,
       1370505.95306381,  845718.14577768, 1201467.05257403,
       1133445.13100691, 1363296.60635036, 1449274.88878057,
       1574538.24324263, 1234101.05116862, 1484592.14139739,
       1295690.9284083 , 1222294.90932427,  990630.3601567 ,
       1693986.49548772, 1823216.96696513, 1135730.16536688,
```

In [36]:
```
#Finding accuraccy

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

mse = mean_squared_error(Y_test, Y_pred)
mae = mean_absolute_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'Accuracy Score: {r2}')
```

```
Mean Squared Error: 10090909840.450602
Mean Absolute Error: 80880.0471705781
Accuracy Score: 0.9179817232181546
```

Random Forest

Random Forest is an ensemble of decision trees, combining their predictions for improved accuracy and robustness. It is widely used for both classification and regression tasks.

In [43]:
```
#Libraries Imported

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

Here we are using

sklearn.ensembl:These models build multiple decision trees to enhance predictive performance.

RandomForestRegressor: Constructs an ensemble of decision trees to predict continuous target variables.

In [44]:
```
#Assuming X_train and Y_train are your feature and target variables

rf_regressor = RandomForestRegressor()
rf_regressor.fit(X_train, Y_train)
Y_pred=rf_regressor.predict(X_test)
```

Here, we are assuming X_train and Y_train represent our feature and target variables, and fitting in a RandomForestRegressor model to the training data and making predictions on the test data.

In [45]:
```
#Libraries Imported

from sklearn import metrics
from sklearn.metrics import mean_squared_error
```

In [46]:
```
#Finding accuraccy

mse = mean_squared_error(Y_test, Y_pred)
mae = mean_absolute_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'Accuracy Score: {r2}')
```

```
Mean Squared Error: 14676257541.453365
Mean Absolute Error: 95029.4363584592
Accuracy Score: 0.8807123071963929
```

Conclusion:

The RandomForest algorithm gives us maximum Accuracy score is 0.8807123071963929. compared to the other machine learning classification algorithm.

The best accuracy with an accuracy score of 88%.