HOUSE PRICE PREDICTION

INTRODUCTION:

House price prediction using machine learning is a powerful application within the real estate domain, offering stakeholders valuable insights into property valuation. The journey commences with loading and preprocessing the dataset, a critical stage where raw data is transformed into a structured format suitable for training machine learning models. This introduction outlines key points in understanding the significance and process of loading and preprocessing the dataset for house price prediction.

1. Objective of House Price Prediction:

House price prediction employs machine learning techniques to estimate the market value of residential properties.

Essential for assisting buyers, sellers, real estate agents, and investors in making informed decisions.

2. Significance of Dataset Loading:

Loading the dataset involves importing historical data, typically sourced from real estate databases, property records, and market reports.

The dataset serves as the foundation for training models that can make accurate predictions about house prices.

3. Role of Preprocessing:

Preprocessing is crucial for transforming raw data into a format conducive to machine learning model training.

Involves handling missing values, feature engineering, and scaling features for improved model performance.

4. Initial Dataset Exploration:

Understanding the dataset's structure and content is paramount.

Exploration involves inspecting the first few rows, checking for missing values, and obtaining basic statistical information.

5. Handling Missing Values:

Imputation strategies, such as filling missing numerical values with means or medians and categorical values with modes, ensure a complete dataset.

6. Feature Engineering:

Creating new features or transforming existing ones can enhance the predictive power of the model.

Examples include extracting temporal information or creating dummy variables for categorical features.

7. Feature Scaling:

Standardizing or normalizing numerical features ensures that all variables are on a comparable scale.

Important for algorithms sensitive to varying feature magnitudes.

8. Data Splitting:

Dividing the dataset into features (X) and target variable (y) is essential for model training.

Further splitting the data into training and testing sets facilitates model evaluation and validation.

9. Optional: Save Processed Data:

Saving the preprocessed data to a new file allows for reproducibility and avoids repeating preprocessing steps in subsequent analyses.

Given data set:

	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	23086.80003	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...
						37778	

NESSARY STEP TO FOLLOW:

Loading and preprocessing:

1. Import Libraries:

Program:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer
```

2. Load the Dataset:

Load the dataset into a Pandas DataFrame using pd.read_csv(). Replace 'your_dataset.csv' with the actual file path or URL of your dataset.

Program:

```
file_path = 'your_dataset.csv'

data = pd.read_csv(file_path)
```

3. Explore the Dataset:

Explore the dataset to understand its structure, check for missing values, and get basic statistics. This step helps you identify issues that need to be addressed during preprocessing.

Program:

```
print(data.head()) # Display the first few rows

print(data.isnull().sum()) # Check for missing values

print(data.describe()) # Get basic statistics
```

4. Handle Missing Values:

Handle missing values using imputation strategies. For numerical columns, you might use the mean or median, and for categorical columns, you might use the mode.

Program:

```
numerical_imputer = SimpleImputer(strategy='mean')

data['numerical_column'] = numerical_imputer.fit_transform(data[['numerical_column']])

categorical_imputer = SimpleImputer(strategy='most_frequent')

data['categorical_column'] = categorical_imputer.fit_transform(data[['categorical_column']])
```

5. Feature Engineering:

Create new features or transform existing ones. Here, we extract the year and month from a date column and create dummy variables for categorical features.

Program:

```
data['year'] = pd.to_datetime(data['date_column']).dt.year

data['month'] = pd.to_datetime(data['date_column']).dt.month

data = pd.get_dummies(data, columns=['categorical_column'])
```

6. Feature Scaling:

Standardize or normalize numerical features to ensure they are on a similar scale. This is important for certain machine learning algorithms.

Program:

```
scaler = StandardScaler()

data[['numerical_feature1', 'numerical_feature2']] = scaler.fit_transform(data[['numerical_feature1',
'numerical_feature2']])
```

7. Split the Data:

Split the dataset into features (X) and the target variable (y). Also, split the data into training and testing sets using train_test_split.

Program:

```
X = data.drop('target_column', axis=1)

y = data['target_column']

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

8. Save Processed Data (Optional):

If you make significant changes during preprocessing and want to save the modified dataset, you can use the to_csv() function.

Program:

```
data.to_csv('processed_data.csv', index=False)
```

IMPORTANCE OF LOADING AND PROCESSING DATASET:

Loading and preprocessing the dataset is an important first step building any machine learning model. However, it is especially important for house price prediction models, as house price datasets are often complex and noisy.

By loading and preprocessing the dataset, we can ensure that the machine learning algorithm is able to learn from the data effectively and accurately.

Challenges involved in loading and preprocessing a house price dataset:

There are a number of challenges involved in loading and preprocessing a house price dataset, including:

Handling missing values:

House price datasets often contain missing values, which can be due to a variety of factors, such as human error or incomplete data collection. Common methods for handling missing values include dropping the rows with missing values, imputing the missing values

with the mean or median of the feature, or using a more sophisticated method such as multiple imputation.

Encoding categorical variables:

House price datasets often contain categorical features, such as the type of house, the neighborhood, and the school district. These features need to be encoded before they can be used by machine learning models.

One common way to encode categorical variables is to use one-hot encoding.

Scaling the features:

It is often helpful to scale the features before training a machine learning model. This can help to improve the performance of the model and make it more robust to outliers. There are a variety of ways to scale the features, such as min-max scaling and standard scaling.

Splitting the dataset into training and testing sets:

Once the data has been pre-processed, we need to split the dataset into training and testing sets. The training set will be used to train the model, and the testing set will be used to evaluate the performance of the model on unseen data. It is important to split the dataset in a way that is representative of the real-world distribution of the data.

How to overcome the challenges of loading and preprocessing a house price dataset:

There are a number of things that can be done to overcome the challenges of loading and preprocessing a house price dataset, including:

Use a data preprocessing library:

There are a number of libraries available that can help with data preprocessing tasks, such as handling missing values, encoding categorical variables, and scaling the features.

Carefully consider the specific needs of your model:

The best way to preprocess the data will depend on the specific machine learning algorithm that you are using. It is important to carefully consider the requirements of the algorithm and to preprocess the data in a way that is compatible with the algorithm.

Validate the preprocessed data:

It is important to validate the preprocessed data to ensure that it is in a format that can be used by the machine learning algorithm and that it is of high quality. This can be done by inspecting the data visually or by using statistical methods.

1.Loading the dataset:

Loading the dataset using machine learning is the process of bringing the data into the machine learning environment so that it can be used to train and evaluate a model.

The specific steps involved in loading the dataset will vary depending on the machine learning library or framework that is being used.

However, there are some general steps that are common to most machine learning frameworks:

a. ### Identify the dataset:

The first step is to identify the dataset that you want to load. This dataset may be stored in a local file, in a database, or in a cloud storage service.

b. ### Load the data base:

Once you have identified the dataset, you need to load it into the machine learning environment. This may involve using a built-in function in the machine learning library, or it may involve writing your own code.

c. Preprocess the dataset:

Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model. This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training and test sets.

Program:

```
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
from sklearn.metrics import r2_score, mean_absolute_error,mean_squared_error
from sklearn.linear_model import LinearRegression from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR import xgboost as xg

%matplotlib inline import warnings
warnings.filterwarnings("ignore")
dataset = pd.read_csv('/kaggle/input/usa-housing/USA_Housing.csv')
          dataset dataset.info() dataset.describe() sns.histplot(dataset, x='Price',
          bins=50, color='y') sns.boxplot(dataset, x='Price', palette='Blues') sns.jointplot(dataset,
x='Avg.Area House Age', y='Price', kind='hex') sns.jointplot(dataset, x='Avg. Area Income',
y='Price') plt.figure(figsize=(12,8))

sns.pairplot(dataset) dataset.hist(figsize=(10,8)) dataset.corr(numeric_only=True)
plt.figure(figsize=(10,5))
sns.heatmap(dataset.corr(numeric_only = True), annot=True)
X = dataset[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
'Avg. Area Number of Bedrooms', 'Area Population']]
Y_train.head()
Y = dataset['Price']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=101)
Y_train.head()
Y_train.shape
Y_test.head()
Y_test.shape

sc = StandardScaler()
X_train_scal = sc.fit_transform(X_train) X_test_scal = sc.fit_transform(X_test)
model_lr=LinearRegression() model_lr.fit(X_train_scal, Y_train)
Prediction1 = model_lr.predict(X_test_scal)
    plt.figure(figsize=(12,6)) plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
plt.plot(np.arange(len(Y_test)), Prediction1, label='Predicted Trend')
```

```
plt.xlabel('Data') plt.ylabel('Trend') plt.legend() plt.title('Actual vs Predicted')
sns.histplot((Y_testPrediction1), bins=50) print(r2_score(Y_test,
Prediction1)) print(mean_absolute_error(Y_test,
Prediction1)) print(mean_squared_error(Y_test, Prediction1)) model_svr = SVR()
model_svr.fit(X_train_scal, Y_train) Prediction2 = model_svr.predict(X_test_scal)
 plt.figure(figsize=(12,6))

plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend') plt.plot(np.arange(len(Y_test)),
Prediction2, label='Predicted Trend')
plt.xlabel('Data') plt.ylabel('Trend') plt.legend() plt.title('Actual vs Predicted')
sns.histplot((Y_testPrediction2), bins=50) print(r2_score(Y_test,
Prediction2)) print(mean_absolute_error(Y_test,
Prediction2)) print(mean_squared_error(Y_test,
Prediction2)) model_lar = Lasso(alpha=1)
model_lar.fit(X_train_scal,Y_train) Prediction3
 = model_lar.predict(X_test_scal)
plt.figure(figsize=(12,6)) plt.plot(np.arange(len(Y_test)),
Y_test, label='Actual Trend')


   plt.plot(np.arange(len(Y_test)), Prediction3, label='Predicted Trend') plt.xlabel('Data')
plt.ylabel('Trend') plt.legend()
 plt.title('Actual vs Predicted') sns.histplot((Y_test
Prediction3), bins=50) print(r2_score(Y_test,
Prediction2)) print(mean_absolute_error(Y_test,
Prediction2)) print(mean_squared_error(Y_test,
Prediction2)) model_rf =
RandomForestRegressor(n_estimators=50) model_rf.fit(X_train_scal,
 Y_train)         Prediction4 =    model_rf.predict(X_test_scal)
plt.figure(figsize=(12,6)) plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
plt.plot(np.arange(len(Y_test)), Prediction4, label='Predicted Trend') plt.xlabel('Data')
plt.ylabel('Trend') plt.legend() plt.title('Actual vs Predicted')

sns.histplot((Y_test-Prediction4), bins=50) print(r2_score(Y_test, Prediction2))
print(mean_absolute_error(Y_test, Prediction2)) print(mean_squared_error(Y_test,
Prediction2))

model_xg = xg.XGBRegressor() model_xg.fit(X_train_scal, Y_train)



Prediction5 = model_xg.predict(X_test_scal)
plt.figure(figsize=(12,6)) plt.plot(np.arange(len(Y_test)),
 Y_test, label='Actual Trend') plt.plot(np.arange(len(Y_test)),
Prediction5, label='Predicted Trend')
plt.xlabel('Data') plt.ylabel('Trend') plt.legend() plt.title('Actual vs Predicted')
sns.histplot((Y_testPrediction4), bins=50) print(r2_score(Y_test, Prediction2))
print(mean_absolute_error(Y_test, Prediction2)) print(mean_squared_error(Y_test,
Prediction2)) ### Linear Regression is giving us best Accuracy.
```

OUTPUT:

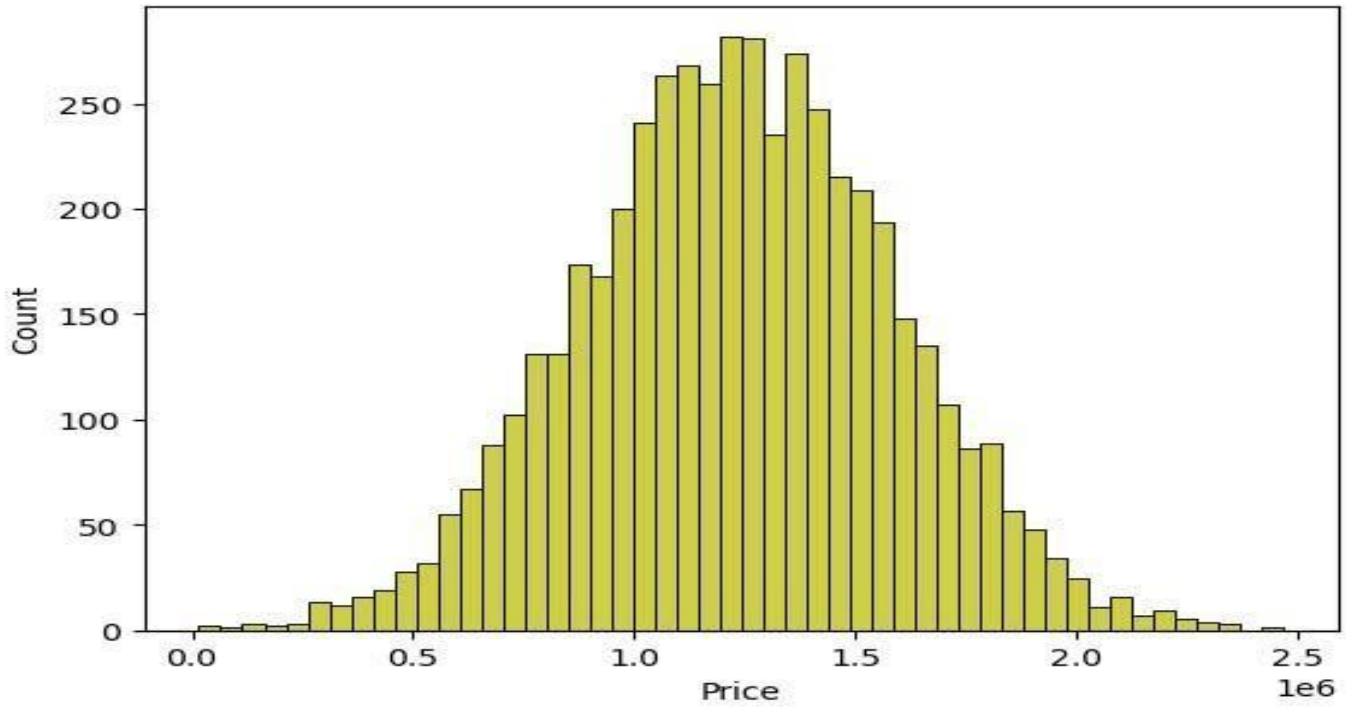
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.80003	1.059034e+06	208 Michael Ferry Apt. 674\n[illegible] NE 37051...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\n[illegible] Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\n[illegible]own, 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 Wilson\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\n[illegible], 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...

2.Preprocessing the dataset:

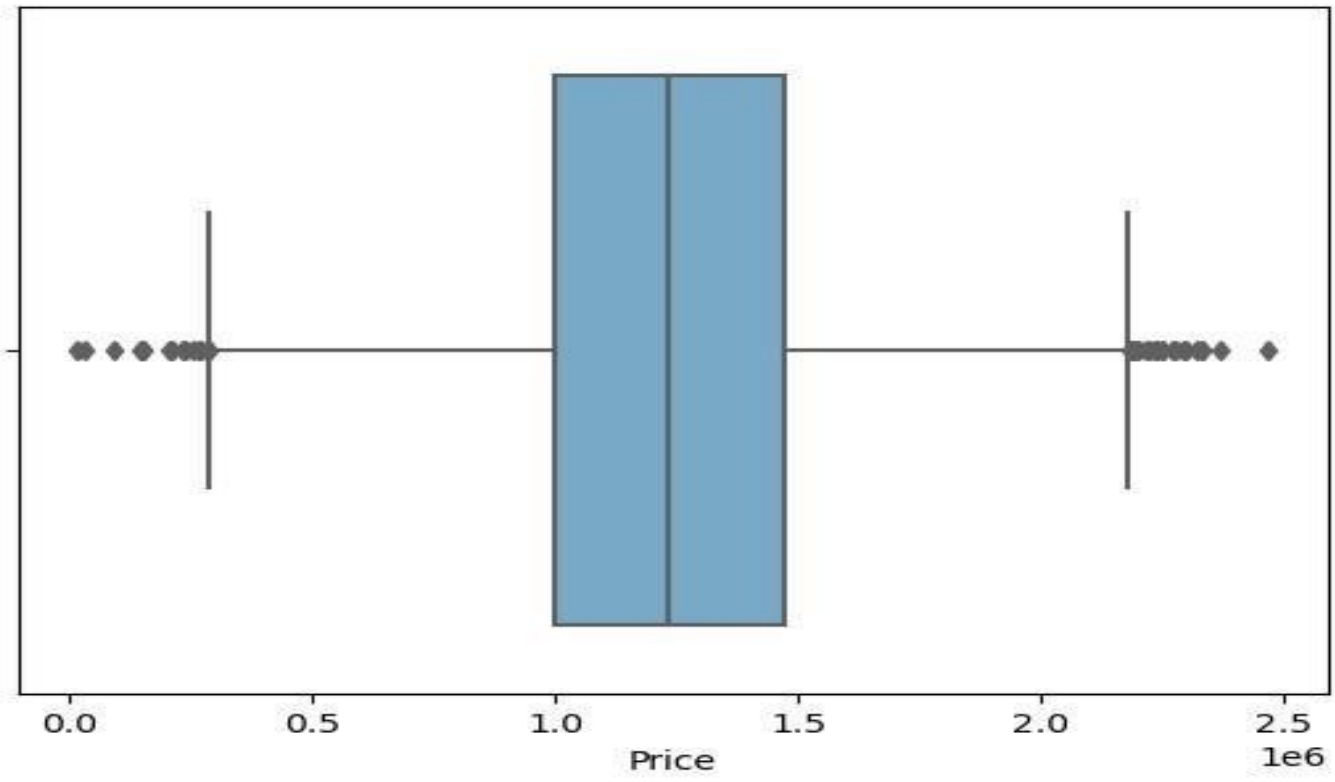
- Data preprocessing is the process of cleaning, transforming, and integrating data in order to make it ready for analysis.
- This may involve removing errors and inconsistencies, handling missing values, transforming the data into a consistent format, and scaling the data to a suitable range.

Visualization and Pre-Processing of Data:

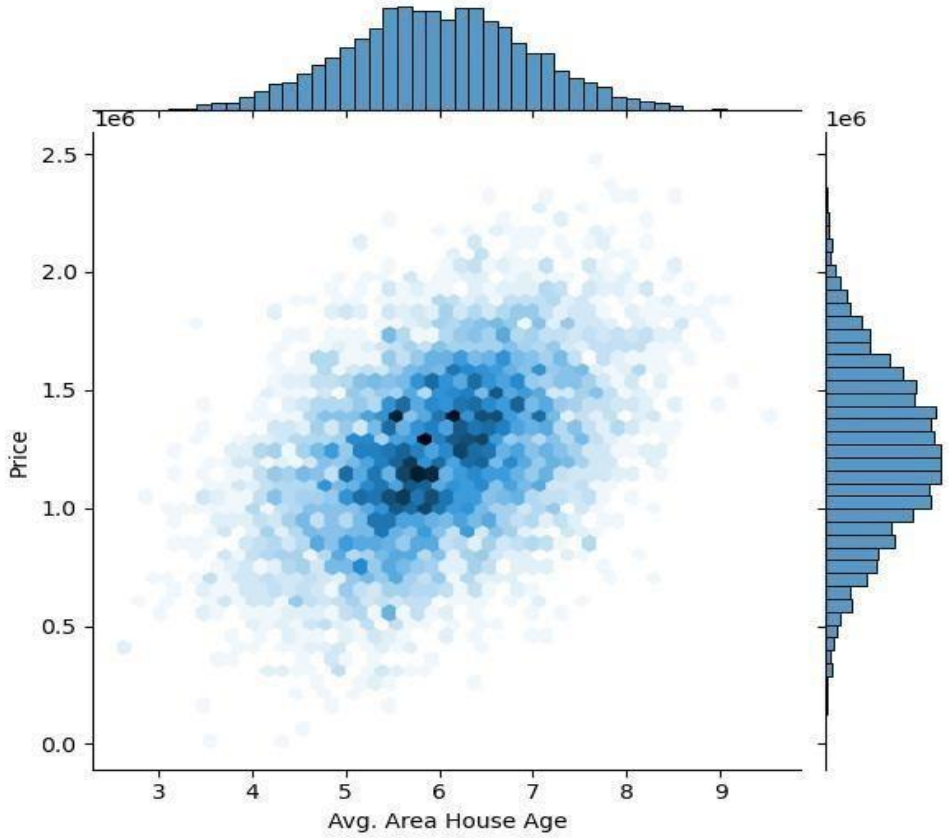
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area
Number of Bedrooms', 'Area Population', 'Price', 'Address'], dtype='object')
<Axes: xlabel='Price', ylabel='Count'>
```



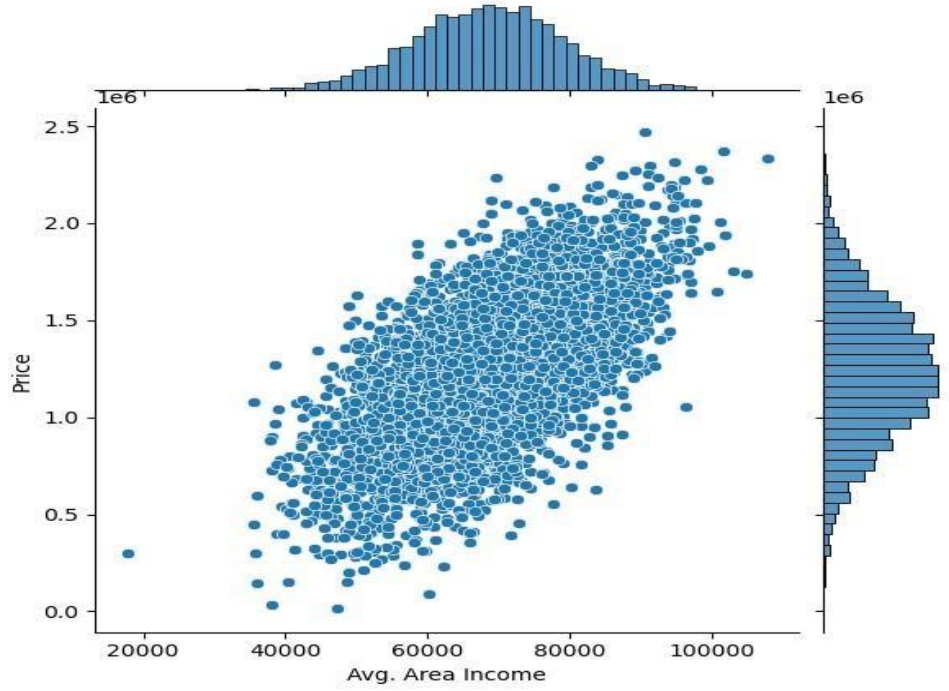
<Axes: xlabel='Price'>



<seaborn.axisgrid.JointGrid at 0x7dbe246100a0>

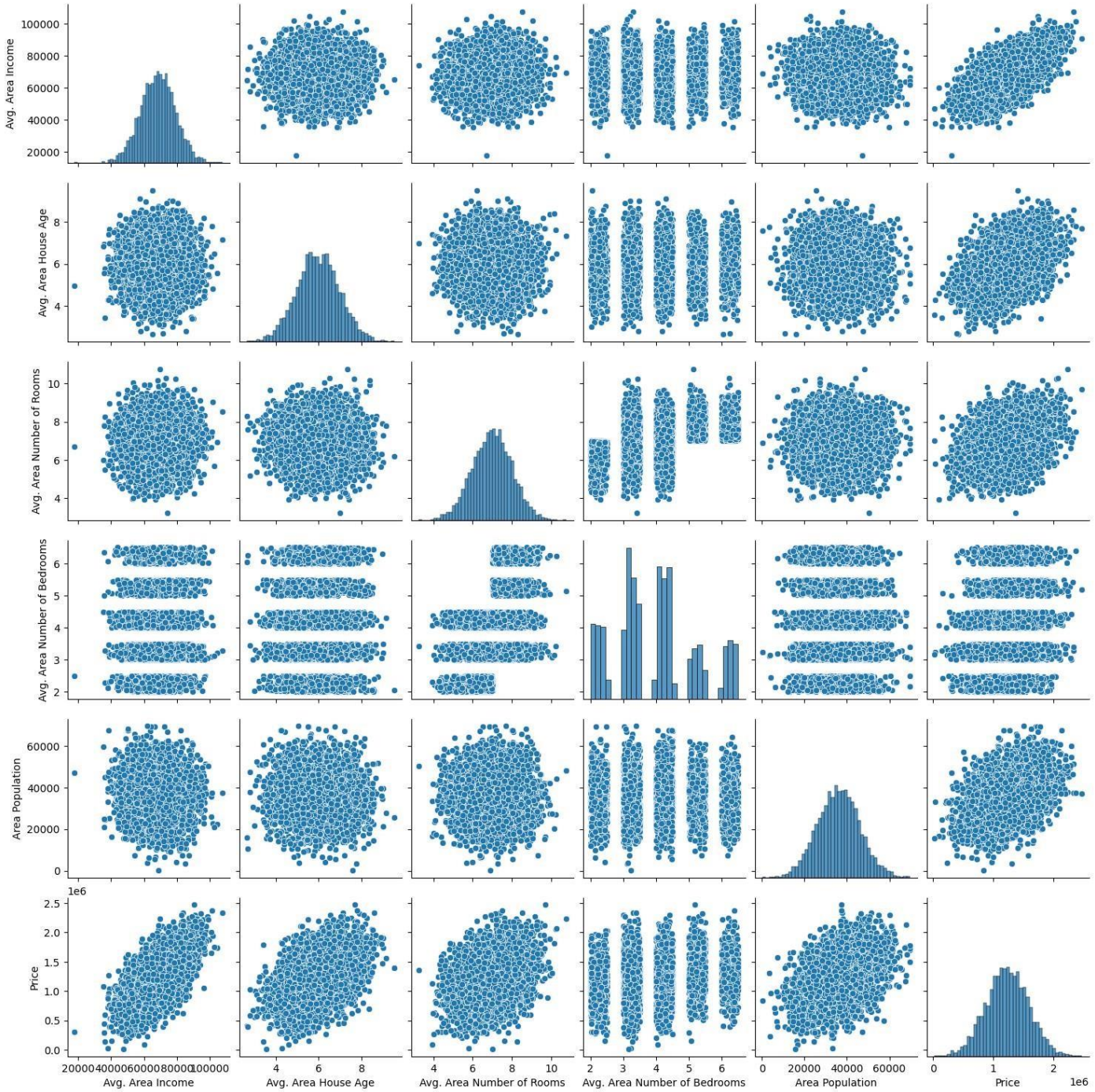


<seaborn.axisgrid.JointGrid at 0x7dbe1333c250>



<seaborn.axisgrid.PairGrid at 0x7dbe1333c340>

<Figure size 1200x800 with 0 Axes>



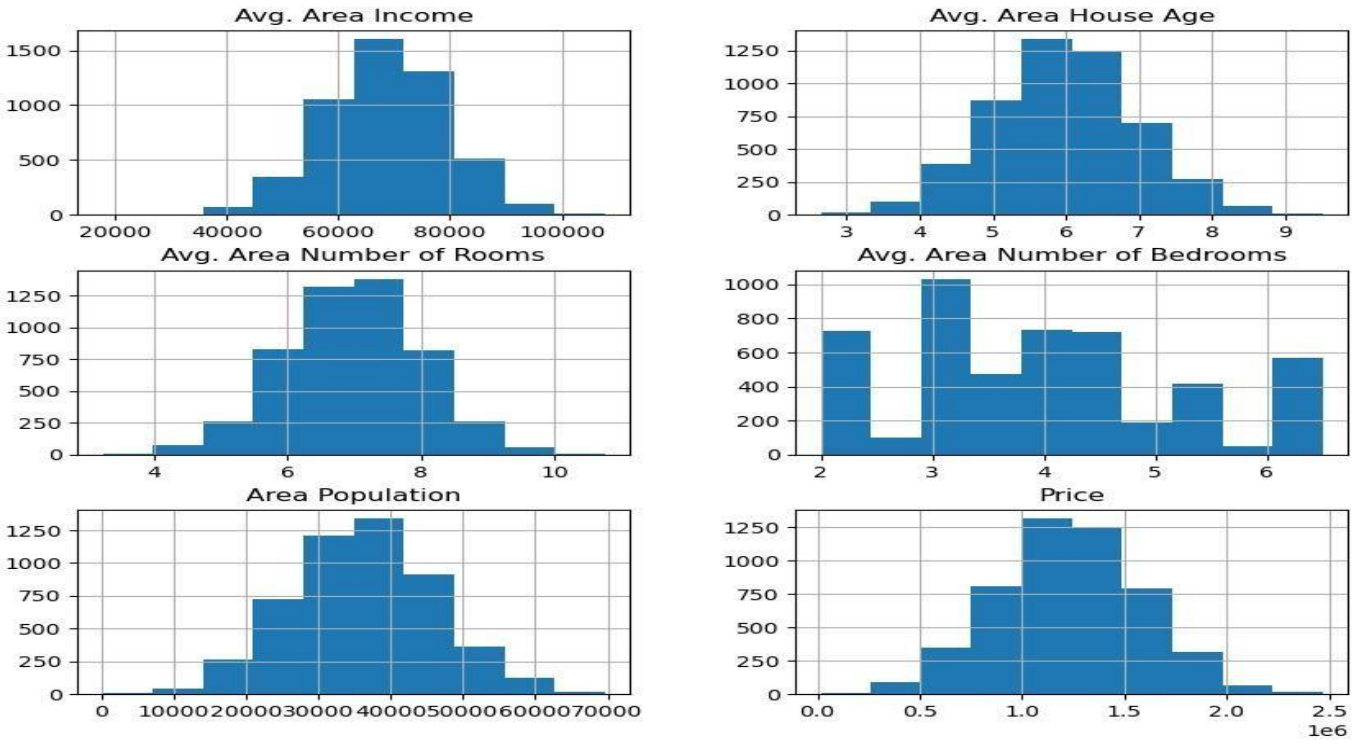
array([[<Axes: title={'center': 'Avg. Area Income'}>, <Axes: title={'center': 'Avg. Area House Age'}>],

[<Axes: title={'center': 'Avg. Area Number of Rooms'}>,

<Axes: title={'center': 'Avg. Area Number of Bedrooms'}>],

[<Axes: title={'center': 'Area Population'}>,

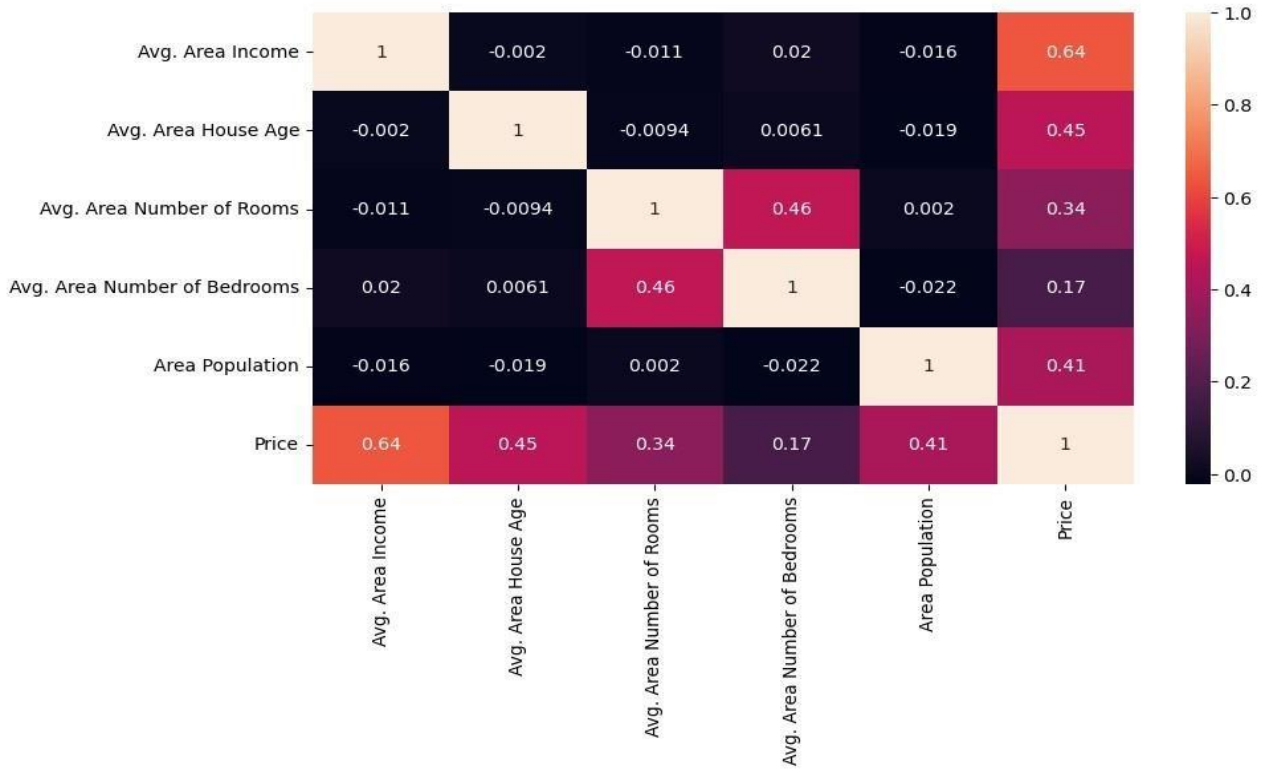
<Axes: title={'center': 'Price'}>]], dtype=object)



	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
	-0.002007			0.006149		
Avg. Area House Age		1.000000	-0.009428		-0.018743	0.452543
	-0.011032			0.462695		
Avg. Area Number of Rooms		-0.009428	1.000000		0.002040	0.335664

Avg. Area Number of Bedrooms	0.019788 -0.016234	0.006149	0.462695	1.000000 -0.022168	-0.022168	0.171071
Area Population	0.639734	-0.018743	0.002040	0.171071	1.000000	0.408556
Price		0.452543	0.335664		0.408556	1.000000

<Axes: >



3413 1.305210e+06

1610 1.400961e+06

3459 1.048640e+06

4293 1.231157e+06

1039 1.391233e+06

Name: Price, dtype: float64

(4000,)

1718 1.251689e+06

2511 8.730483e+05

345 1.696978e+06

2521 1.063964e+06

54 9.487883e+05

Name: Price, dtype: float64

(1000,)

Some common data preprocessing tasks include:

Data cleaning: This involves identifying and correcting errors and inconsistencies in the data. For example, this may involve removing duplicate records, correcting typos, and filling in missing values.

Data transformation: This involves converting the data into a format that is suitable for the analysis task. For example, this may involve converting categorical data to numerical data, or scaling the data to a suitable range.

Feature engineering: This involves creating new features from the existing data. For example, this may involve creating features that represent interactions between variables, or features that represent summary statistics of the data.

Data integration: This involves combining data from multiple sources into a single dataset. This may involve resolving inconsistencies in the data, such as different data formats or different variable names.

Data preprocessing is an essential step in many data science projects. By carefully preprocessing the data, data scientists can improve the accuracy and reliability of their results.

Program:

```
# Importing necessary libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline


# Step 1: Load the dataset

data = pd.read_csv('E:\USA_Housing.csv')


# Step 2: Exploratory Data Analysis (EDA)

print("--- Exploratory Data Analysis ---")

print("1. Checking for Missing Values:")

missing_values = data.isnull().sum()

print(missing_values)


print("\n2. Descriptive Statistics:")

description = data.describe()
```

```
print(description)


# Step 3: Feature Engineering

print("\n--- Feature Engineering ---")


# Separate features and target variable

X = data.drop('price', axis=1)

y = data['price']


# Define which columns should be one-hot encoded (categorical)

categorical_cols = [' Avg. Area House Age']


# Define preprocessing steps using ColumnTransformer and Pipeline

preprocessor = ColumnTransformer(

transformers=[

('num', StandardScaler(), [' Avg. Area Number of Rooms ', ' Avg.

Area Number of Bedrooms ', ' Area Population ', ' Avg. Area Income ']),

('cat', OneHotEncoder(), categorical_cols)

])

# Step 4: Data Splitting

print("\n--- Data Splitting ---")

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

random_state=42)

print(f"X_train shape: {X_train.shape}")

print(f"X_test shape: {X_test.shape}")

print(f"y_train shape: {y_train.shape}")
```

```
print(f"y_test shape: {y_test.shape}")



# Step 5: Preprocessing and Feature Scaling using Pipeline

print("\n--- Feature Scaling ---")

model = Pipeline([

('preprocessor', preprocessor),

])

# Fit the preprocessing pipeline on the training data

X_train = model.fit_transform(X_train)

# Transform the testing data using the fitted pipeline

X_test = model.transform(X_test)



print("--- Preprocessing Complete! ---")
```

Output:

Exploratory Data Analysis:

1. Checking for Missing Values:

```
<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
```

2. Descriptive Statistics:

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

Data Splitting;

Y_train.shape = (4000,)

Y_test.shape = (1000,)

Conclusion:

In the quest to build a house price prediction model, we have embarked on a critical journey that begins with loading and pre-processing the dataset. We have traversed through essential steps, starting with importing the necessary libraries to facilitate data manipulation and analysis.

Understanding the data's structure, characteristics, and any potential issues through exploratory data analysis (EDA) is essential for informed decision-making.

Data pre-processing emerged as a pivotal aspect of this process. It involves cleaning, transforming, and refining the dataset to ensure that it aligns with the requirements of machine learning algorithms.

With these foundational steps completed, our dataset is now primed for the subsequent stages of building and training a house price prediction model.