Prediction of House Price using ML

Table of Contents

1. Importing Dependencies

Introduction

The first step in any data science or machine learning project is to import the necessary libraries and packages. In this project, we will rely on a set of powerful and versatile libraries to handle data, build models, and visualize results.

Importance of Dependencies

- NumPy: NumPy is a fundamental library for numerical operations in Python. It provides support for arrays and matrices, making it essential for data manipulation.

- Pandas: Pandas is a data manipulation library that simplifies data handling by providing data structures like DataFrames. It's indispensable for data exploration and preprocessing.

- Scikit-Learn: Scikit-Learn is a machine learning library that offers a wide range of algorithms for regression, classification, and more. It streamlines the implementation of machine learning models.

- Matplotlib and Seaborn: Data visualization is crucial for understanding the dataset and model results. Matplotlib and Seaborn are excellent for creating charts and graphs.

2. Data Exploration

Introduction

Data exploration is the process of understanding the dataset's characteristics. This stage is crucial to grasp the nature of the data and identify any peculiarities that could affect model performance.

Key Aspects of Data Exploration

- Descriptive Statistics: Calculating and visualizing measures such as mean, median, standard deviation, and quartiles to understand the data's central tendencies and spread.

- Exploratory Data Analysis (EDA): Utilizing EDA techniques to visualize data relationships, patterns, and anomalies through histograms, scatter plots, and more.

3. Visualization and Pre-Processing of Data

Introduction

Data visualization and pre-processing are essential steps to prepare the dataset for machine learning. Visualization helps you understand

the data's distribution and relationships, while pre-processing ensures the data is in a suitable format for model training.

Data Visualization Techniques

- Histograms: Used to visualize the distribution of numerical data, histograms provide insights into data spread and skewness.

- Box Plots: Box plots offer information about data distribution, central tendency, and potential outliers.

- Scatter Plots: Scatter plots help visualize relationships between two numerical variables.

Data Pre-Processing

- Handling Missing Values: Strategies to address missing data, including imputation or removal.

- Categorical Data Encoding: Techniques like one-hot encoding or label encoding for dealing with categorical features.

- Feature Scaling: Ensuring that features have the same scale through standardization or normalization.

4. Dividing Dataset into Features and Target Variable

Introduction

To apply machine learning algorithms, we need to divide the dataset into two key components: features and the target variable.

Features and Target Variable

- Features (Independent Variables): These are the attributes used to make predictions. For house price prediction, features could include property size, number of bedrooms, location, and more.

- Target Variable (Dependent Variable): This is what we aim to predict – in this case, the house price.

5. Using Train Test Split

Introduction

The division of the dataset into training and testing sets is critical to assess a model's performance. This is done to ensure that the model's evaluation is based on data it has not seen during training.

Train Test Split

- Training Set: This is the portion of the data used for model training.

- Testing Set: The testing set is reserved for evaluating the model's performance.

6. Standardizing the Data

Introduction

Standardization is a crucial step in data pre-processing. It ensures that all features have the same scale, which can improve the accuracy and performance of many machine learning algorithms.

Standardization Techniques

- Mean Centering: Subtracting the mean value of each feature from the data. This centers the data around zero.

- Feature Scaling: Scaling the data to have a standard deviation of 1. Common scaling methods include Min-Max scaling and Z-score normalization.

Significance of Standardization

Standardization is particularly important when working with machine learning models that rely on distance or magnitude, such as Support Vector Machines and k-Nearest Neighbors.

7. Model Building and Evaluation

Introduction

In this pivotal section, we embark on the process of building and evaluating machine learning models for house price prediction. We will explore various algorithms and assess their performance.

Model Building Steps

- Data Splitting: Separating the data into training and testing sets.

- Model Selection: Choosing the appropriate model for the task.

- Model Training: Fitting the model to the training data.

- Hyper parameter Tuning: Optimizing model parameters for better performance.

- Model Evaluation Metrics: Using appropriate metrics to assess the model's performance.

7.1 Model 1 - Linear Regression

Introduction

Linear regression is a foundational and interpretable model used for predicting continuous values, such as house prices. It assumes a linear relationship between the features and the target variable.

Building and Evaluating a Linear Regression Model

- Model Training: Demonstrating how to train a linear regression model.

- Assumptions: Explaining the key assumptions of linear regression.

- Evaluation Metrics: Employing metrics like Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) to evaluate the model's performance.

7.2 Model 2 - Support Vector Regressor

Introduction

Support Vector Regressor (SVR) is a powerful regression model. It excels in capturing non-linear relationships between features and the target variable.

Working with SVR

- Kernel Functions: Understanding kernel functions in SVR, such as linear, polynomial, and radial basis function kernels.

- Parameter Tuning: Discussing the role of hyperparameter tuning in SVR.

- Performance Evaluation: Using metrics like R-squared and Mean Absolute Percentage Error (MAPE) to assess SVR's performance.

7.3 Model 3 - Lasso Regression

Introduction

Lasso regression is a variant of linear regression that includes L1 regularization. It is valuable when feature selection or feature reduction is necessary.

Lasso Regression in Action

- L1 Regularization: Explaining how L1 regularization works to shrink coefficients and eliminate less important features.

- Parameter Selection: Discussing the importance of selecting the regularization strength.

- Performance Evaluation: Assessing the performance using metrics like Mean Squared Error (MSE) and the coefficient of determination (R-squared).

7.4 Model 4 - Random Forest Regressor

Introduction

Random Forest Regressor is an ensemble model known for its ability to handle complex relationships in the data. It is highly robust and less prone to overfitting.

Leveraging Random Forest

- Ensemble Learning: Understanding the ensemble concept, which combines multiple decision trees.

- Hyperparameter Tuning: Exploring key hyperparameters like the number of trees and tree depth.

- Performance Evaluation: Using metrics such as explained variance score and feature importance to assess Random Forest's performance.

7.5 Model 5 - XGBoost Regressor

Key Features of XGBoost

Gradient Boosting: Understanding the concept of gradient boosting and how XGBoost uses it to build an ensemble of decision trees.

Regularization: Discussing the role of regularization in preventing overfitting and enhancing model generalization.

Tree Pruning: Explaining how XGBoost prunes decision trees to improve model efficiency.

8. Code

House Price Prediction

Importing Dependencies

In [1]:
```
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")
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is r
equired for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Loading Dataset

In [2]:
```
dataset = pd.read_csv('/kaggle/input/usa-housing/USA_Housing.csv')
```

Data Exploration

In [3]:
```
dataset
```

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
...	...	...	...	...	...	...	...
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 [4]:
```
dataset.info()
```

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

In [5]:
```
dataset.describe()
```

Out[5]:

	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

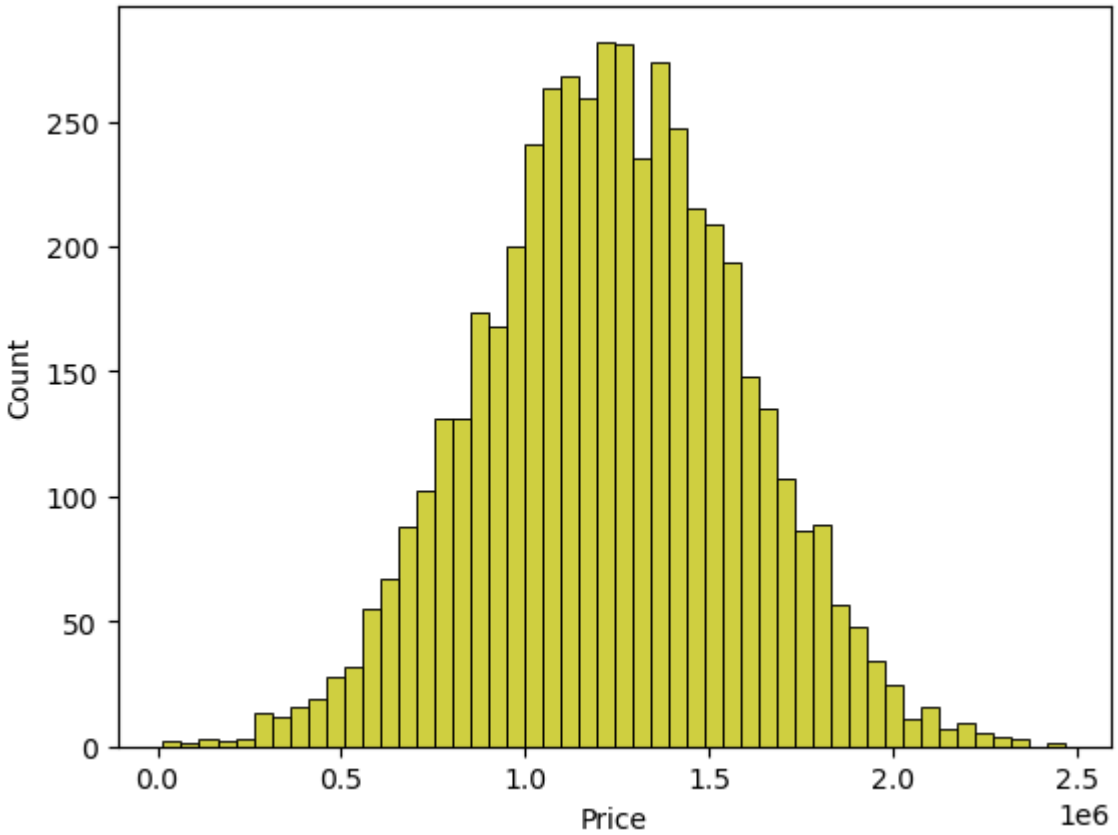
In [6]:
```
dataset.columns
```

Out[6]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

Visualisation and Pre-Processing of Data

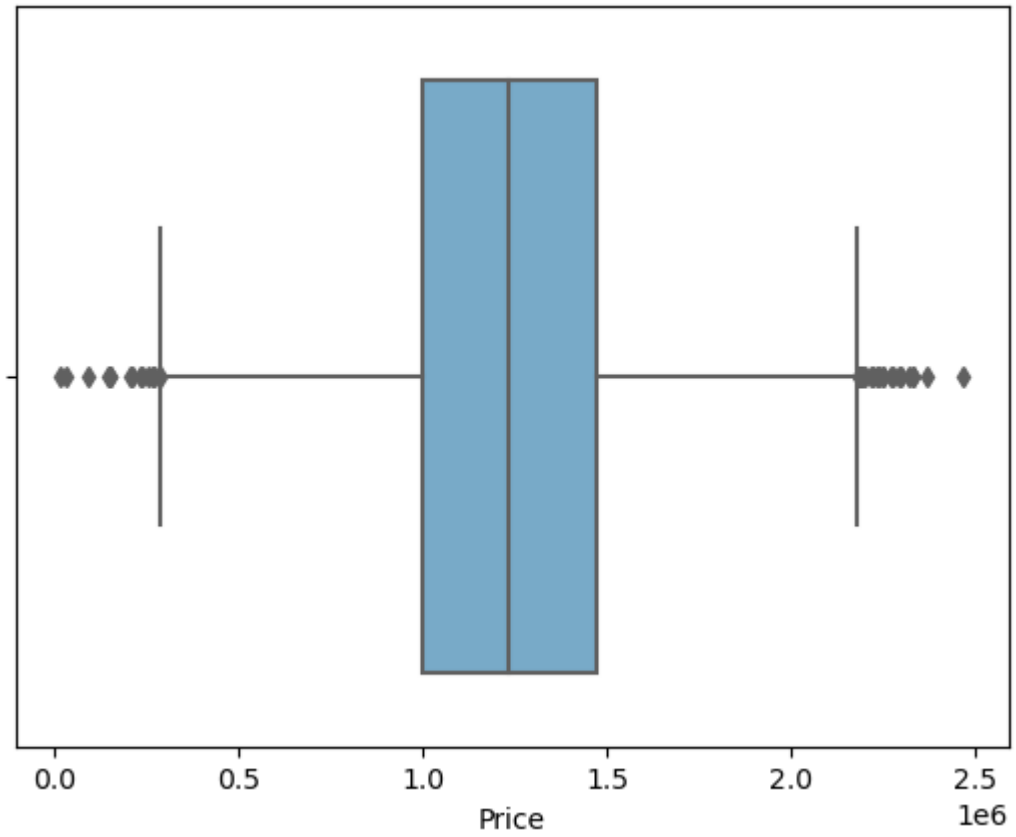
In [7]:
```
sns.histplot(dataset, x='Price', bins=50, color='y')
```

Out[7]:
```
<Axes: xlabel='Price', ylabel='Count'>
```



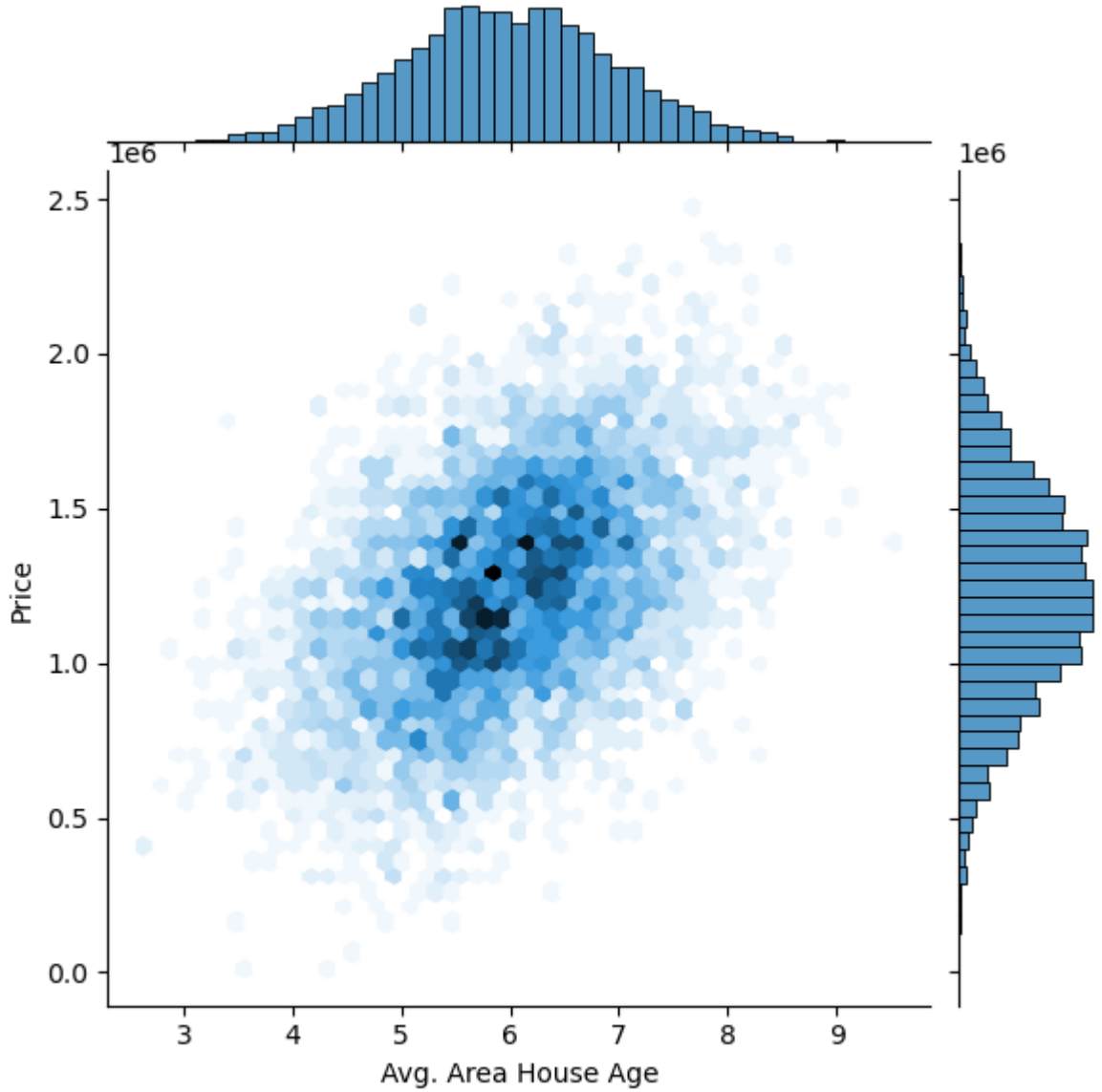
In [8]:
```
sns.boxplot(dataset, x='Price',  palette='Blues')
```

Out[8]:
```
<Axes: xlabel='Price'>
```



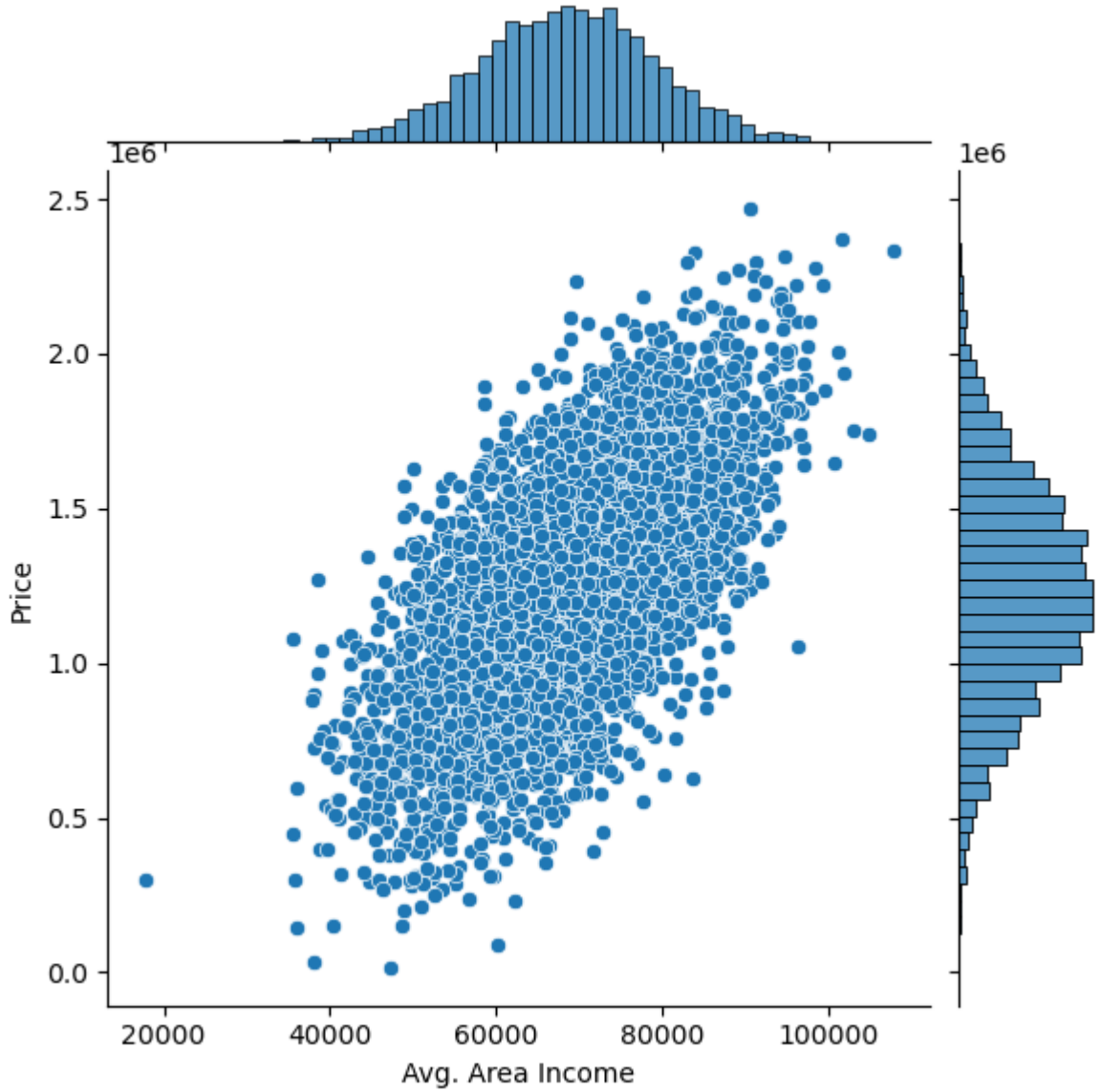
In [9]:
```
sns.jointplot(dataset, x='Avg. Area House Age', y='Price', kind='hex')
```

Out[9]:
```
<seaborn.axisgrid.JointGrid at 0x7caf1d571810>
```



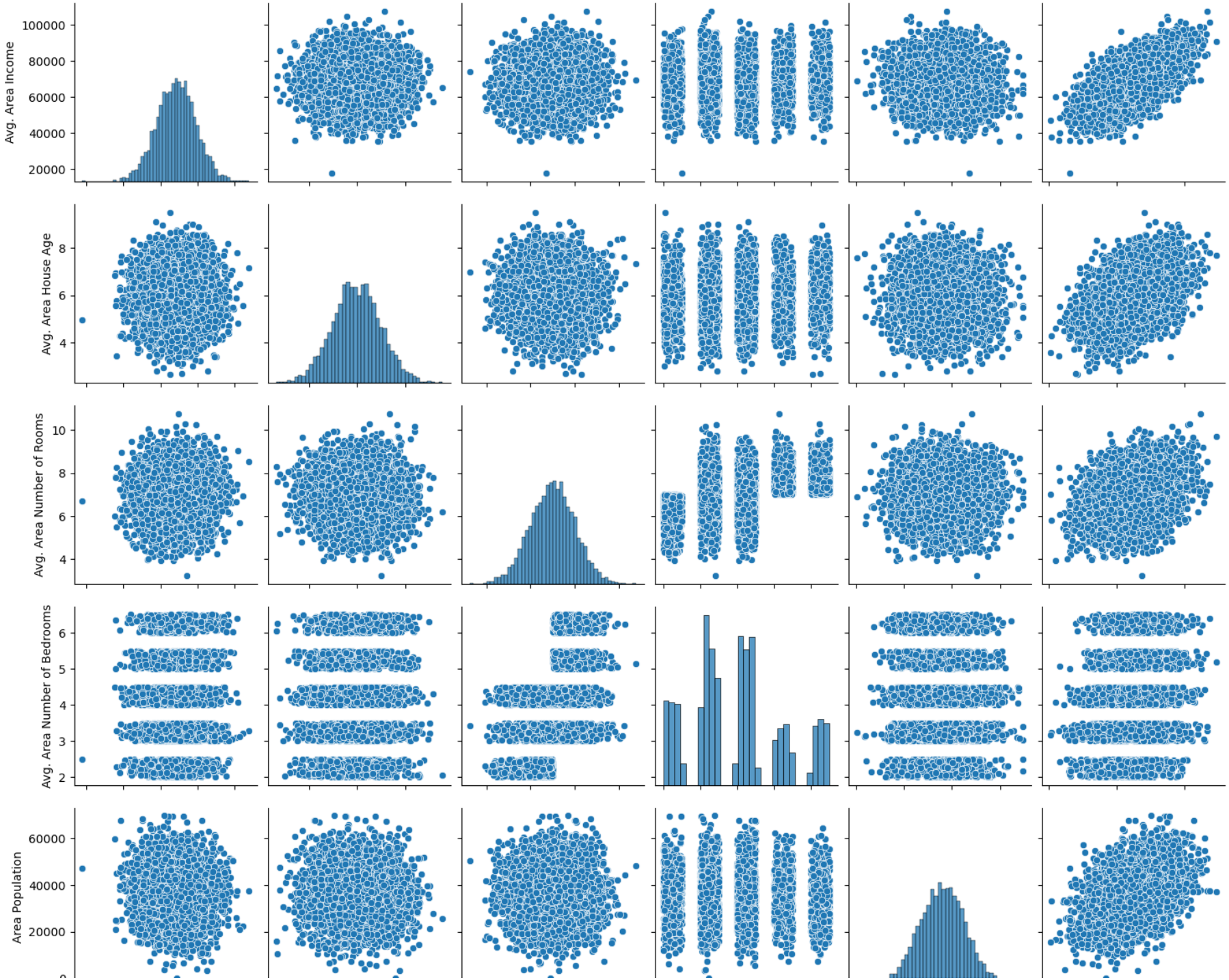
In [10]:
```
sns.jointplot(dataset, x='Avg. Area Income', y='Price')
```

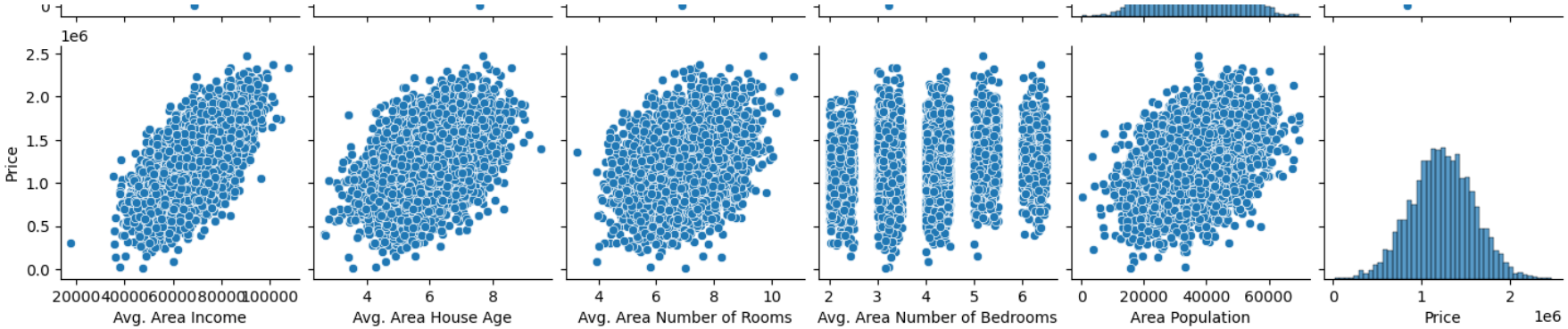
Out[10]:
```
<seaborn.axisgrid.JointGrid at 0x7caf1d8bf7f0>
```



In [11]:
```
plt.figure(figsize=(12,8))
sns.pairplot(dataset)
```

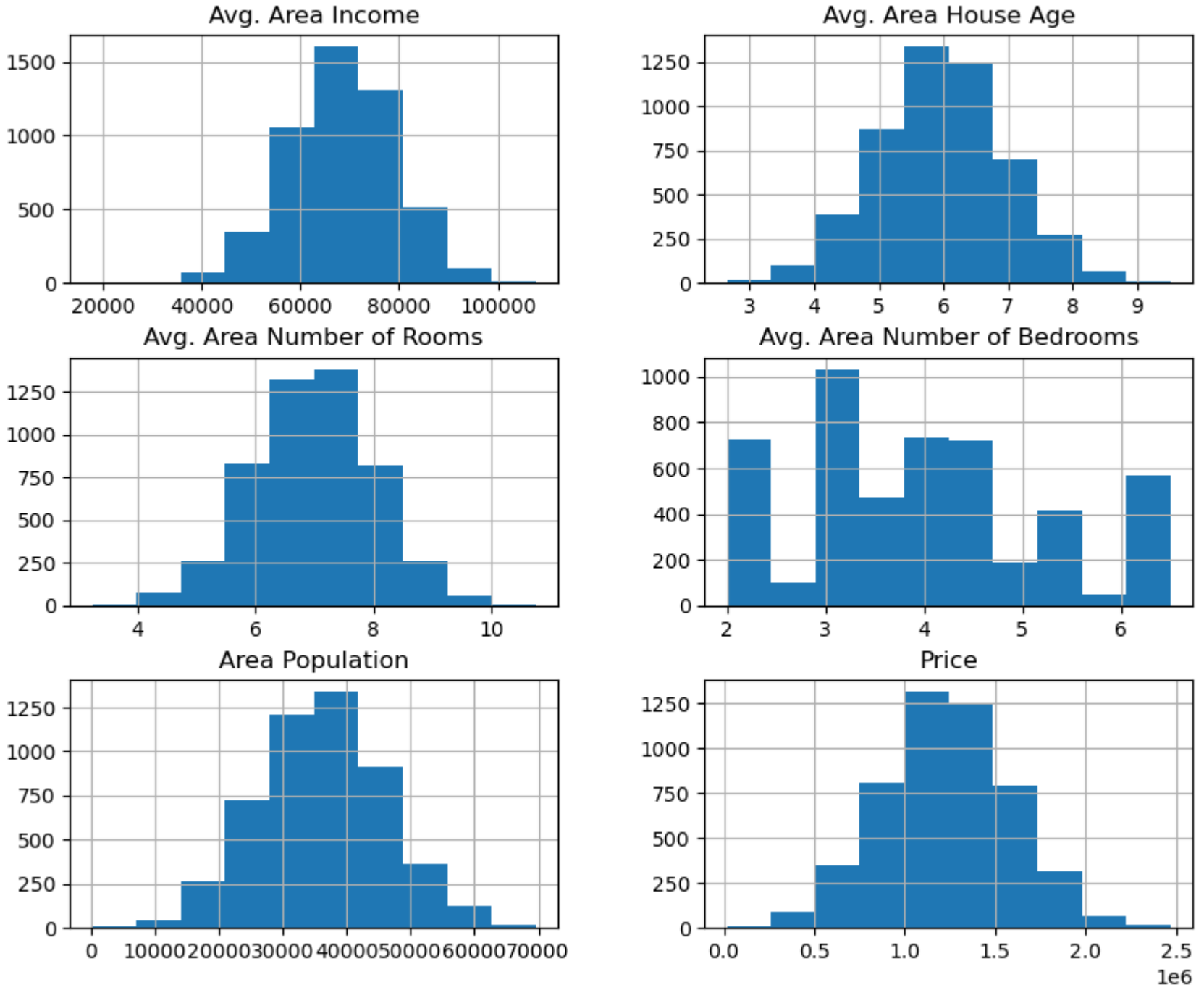
Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x7caf0c2ac550>

<Figure size 1200x800 with 0 Axes>
```


Avg. Area Income
Avg. Area House Age
Avg. Area Number of Rooms
Avg. Area Number of Bedrooms
Area Population


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

In [12]:
```
dataset.hist(figsize=(10,8))
```

Out[12]:
```
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
1e6

Visualising Correlation

In [13]:
```
dataset.corr(numeric_only=True)
```

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

In [14]:
```
plt.figure(figsize=(10,5))
sns.heatmap(dataset.corr(numeric_only = True), annot=True)
```

Out[14]: <Axes: >

Dividing Dataset in to features and target variable

In [15]:
```
X = dataset[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population']]
Y = dataset['Price']
```

Using Train Test Split

In [16]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=101)
```

In [17]:
```
Y_train.head()
```

Out[17]:
```
3413    1.305210e+06
1610    1.400961e+06
3459    1.048640e+06
4293    1.231157e+06
1039    1.391233e+06
Name: Price, dtype: float64
```

In [18]:
```
Y_train.shape
```

Out[18]:
```
(4000,)
```

In [19]:
```
Y_test.head()
```

Out[19]:
```
1718    1.251689e+06
2511    8.730483e+05
345     1.696978e+06
2521    1.063964e+06
54      9.487883e+05
Name: Price, dtype: float64
```

In [20]:
```
Y_test.shape
```

Out[20]:
```
(1000,)
```

Standardizing the data

In [21]:
```
sc = StandardScaler()
X_train_scal = sc.fit_transform(X_train)
X_test_scal = sc.fit_transform(X_test)
```

Model Building and Evaluation

Model 1 - Linear Regression

In [22]:
```
model_lr=LinearRegression()
```

In [23]:
```
model_lr.fit(X_train_scal, Y_train)
```

Out[23]:
```
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.

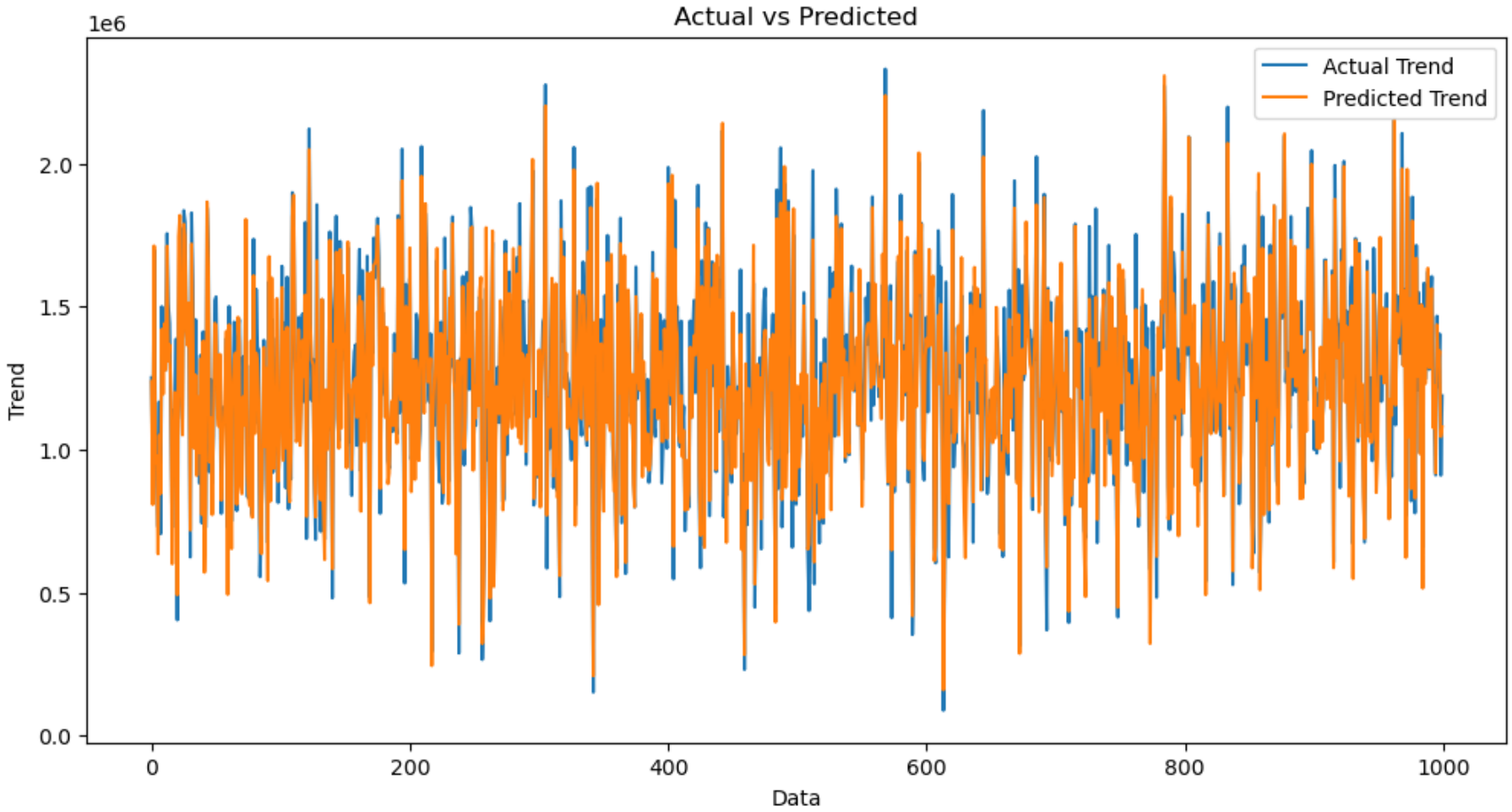
Predicting Prices

In [24]:
```
Prediction1 = model_lr.predict(X_test_scal)
```

Evaluation of Predicted Data

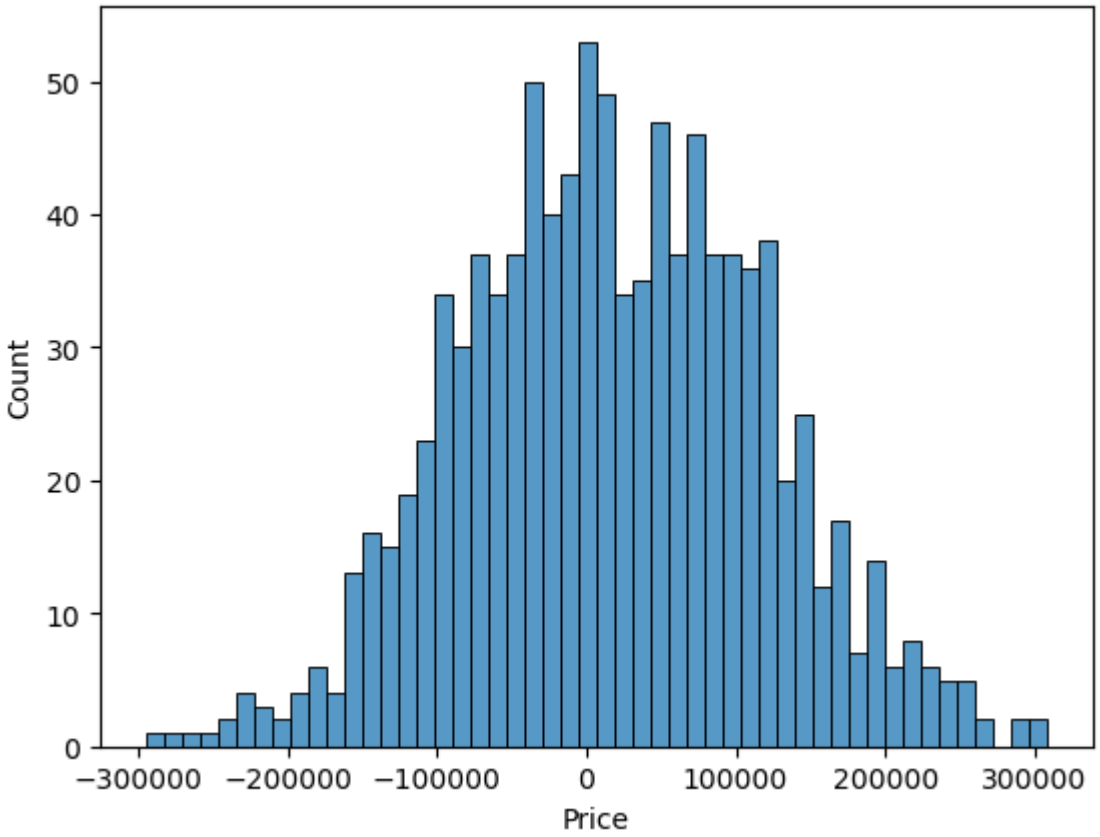
In [25]:
```
    plt.figure(figsize=(12,6))
    plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
    plt.plot(np.arange(len(Y_test)), Prediction1, label='Predicted Trend')
    plt.xlabel('Data')
    plt.ylabel('Trend')
    plt.legend()
    plt.title('Actual vs Predicted')
```

Out[25]:
```
Text(0.5, 1.0, 'Actual vs Predicted')
```


Actual vs Predicted
1e6
Actual Trend
Predicted Trend
Trend
Data
0.0
0.5
1.0
1.5
2.0
0
200
400
600
800
1000

In [26]:
```
sns.histplot((Y_test-Prediction1), bins=50)
```

Out[26]:
```
<Axes: xlabel='Price', ylabel='Count'>
```



In [27]:
```
print(r2_score(Y_test, Prediction1))
print(mean_absolute_error(Y_test, Prediction1))
print(mean_squared_error(Y_test, Prediction1))
```

```
0.9182928179392918
82295.49779231755
10469084772.975954
```

Model 2 - Support Vector Regressor

In [28]:
```
model_svr = SVR()
```

In [29]:
```
model_svr.fit(X_train_scal, Y_train)
```

Out[29]:
```
SVR()
```
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.

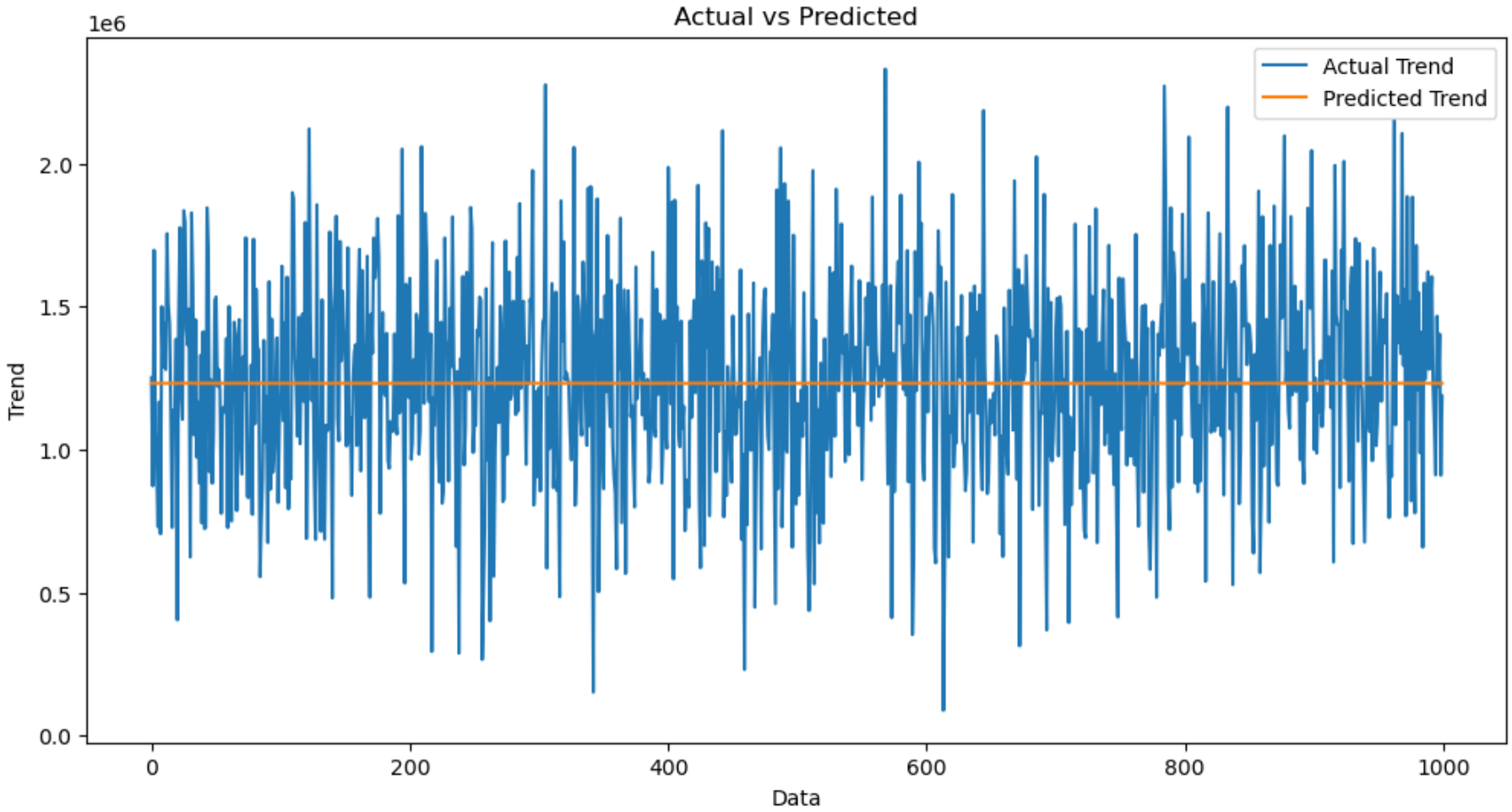
Predicting Prices

In [30]:
```
Prediction2 = model_svr.predict(X_test_scal)
```

Evaluation of Predicted Data

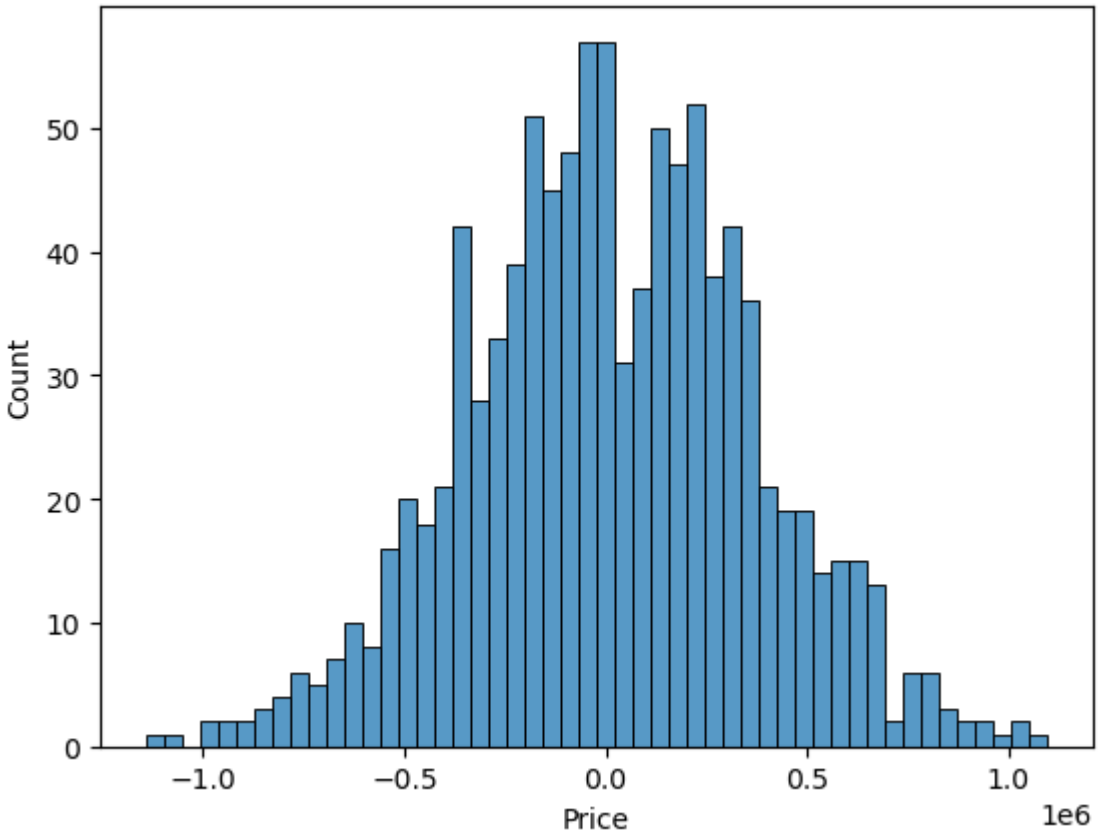
In [31]:
```
    plt.figure(figsize=(12,6))
    plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
    plt.plot(np.arange(len(Y_test)), Prediction2, label='Predicted Trend')
    plt.xlabel('Data')
    plt.ylabel('Trend')
    plt.legend()
    plt.title('Actual vs Predicted')
```

Out[31]:
```
Text(0.5, 1.0, 'Actual vs Predicted')
```


Actual vs Predicted
1e6
Actual Trend
Predicted Trend
Trend
Data
0.0
0.5
1.0
1.5
2.0
0
200
400
600
800
1000

In [32]:
```
sns.histplot((Y_test-Prediction2), bins=50)
```

Out[32]:
```
<Axes: xlabel='Price', ylabel='Count'>
```



In [33]:
```
print(r2_score(Y_test, Prediction2))
print(mean_absolute_error(Y_test, Prediction2))
print(mean_squared_error(Y_test, Prediction2))
```

```
-0.0006222175925689744
286137.81086908665
128209033251.4034
```

Model 3 - Lasso Regression

In [34]:
```
model_lar = Lasso(alpha=1)
```

In [35]:
```
model_lar.fit(X_train_scal,Y_train)
```

Out[35]:
```
Lasso(alpha=1)
```

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.

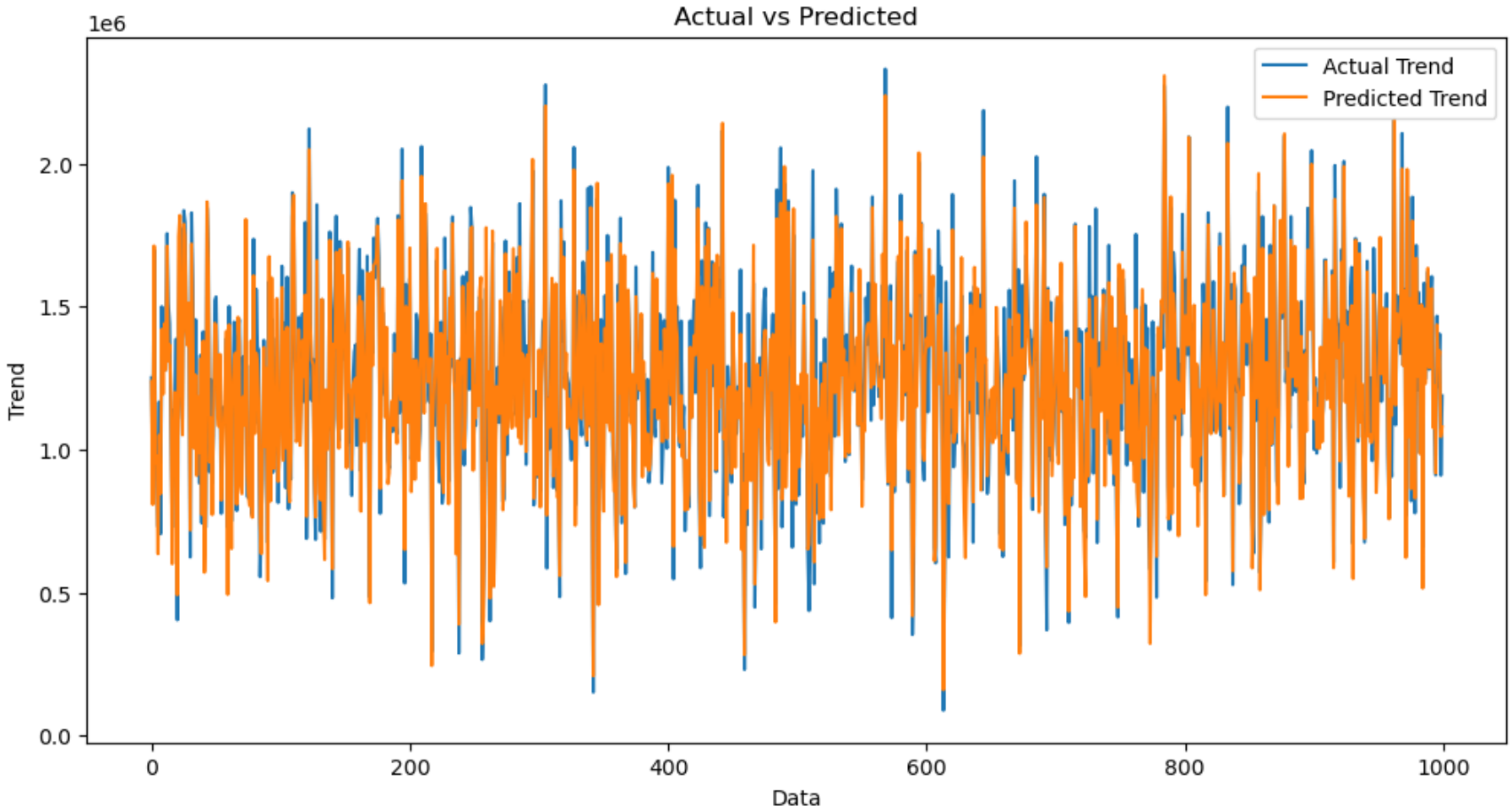
Predicting Prices

In [36]:
```
Prediction3 = model_lar.predict(X_test_scal)
```

Evaluation of Predicted Data

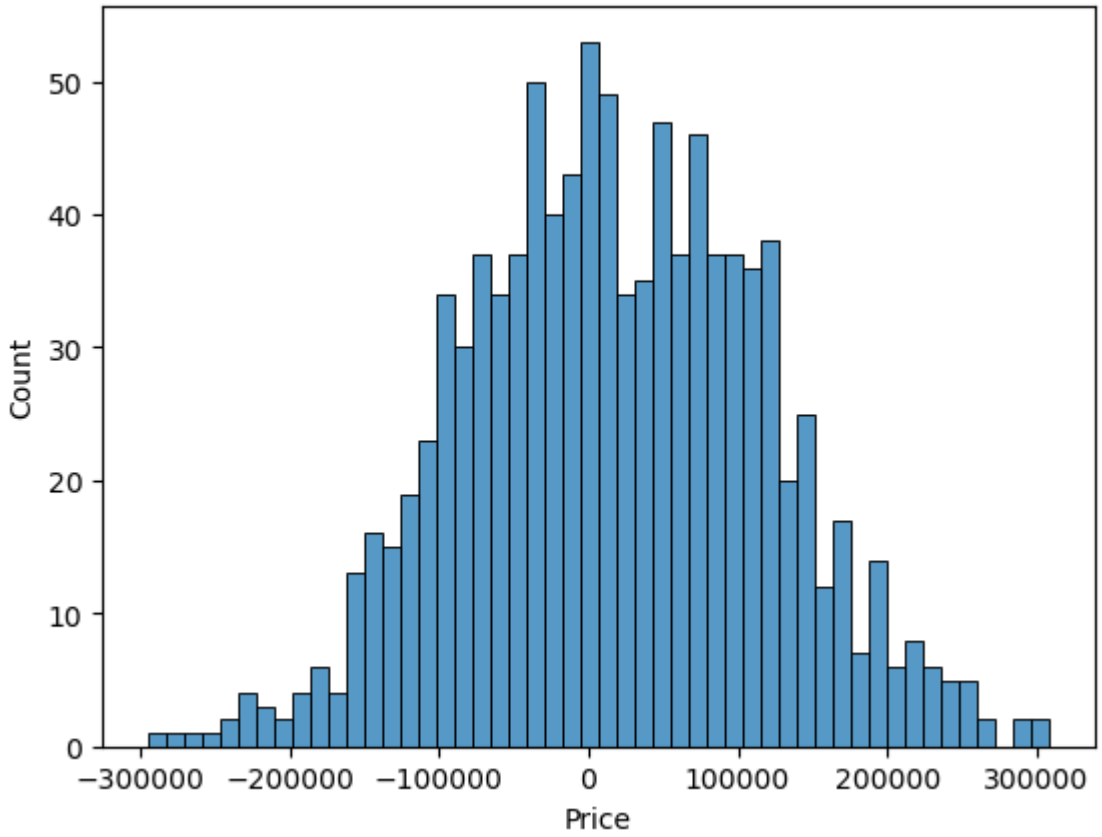
In [37]:
```
    plt.figure(figsize=(12,6))
    plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
    plt.plot(np.arange(len(Y_test)), Prediction3, label='Predicted Trend')
    plt.xlabel('Data')
    plt.ylabel('Trend')
    plt.legend()
    plt.title('Actual vs Predicted')
```

Out[37]:
```
Text(0.5, 1.0, 'Actual vs Predicted')
```


Actual vs Predicted
1e6
Actual Trend
Predicted Trend
Trend
Data
0.0
0.5
1.0
1.5
2.0
0
200
400
600
800
1000

In [38]:
```
sns.histplot((Y_test-Prediction3), bins=50)
```

Out[38]: <Axes: xlabel='Price', ylabel='Count'>



In [39]:
```
print(r2_score(Y_test, Prediction2))
print(mean_absolute_error(Y_test, Prediction2))
print(mean_squared_error(Y_test, Prediction2))
```

```
-0.0006222175925689744
286137.81086908665
128209033251.4034
```

Model 4 - Random Forest Regressor

In [40]:
```
model_rf = RandomForestRegressor(n_estimators=50)
```

In [41]:
```
model_rf.fit(X_train_scal, Y_train)
```

Out[41]:
```
RandomForestRegressor(n_estimators=50)
```
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.

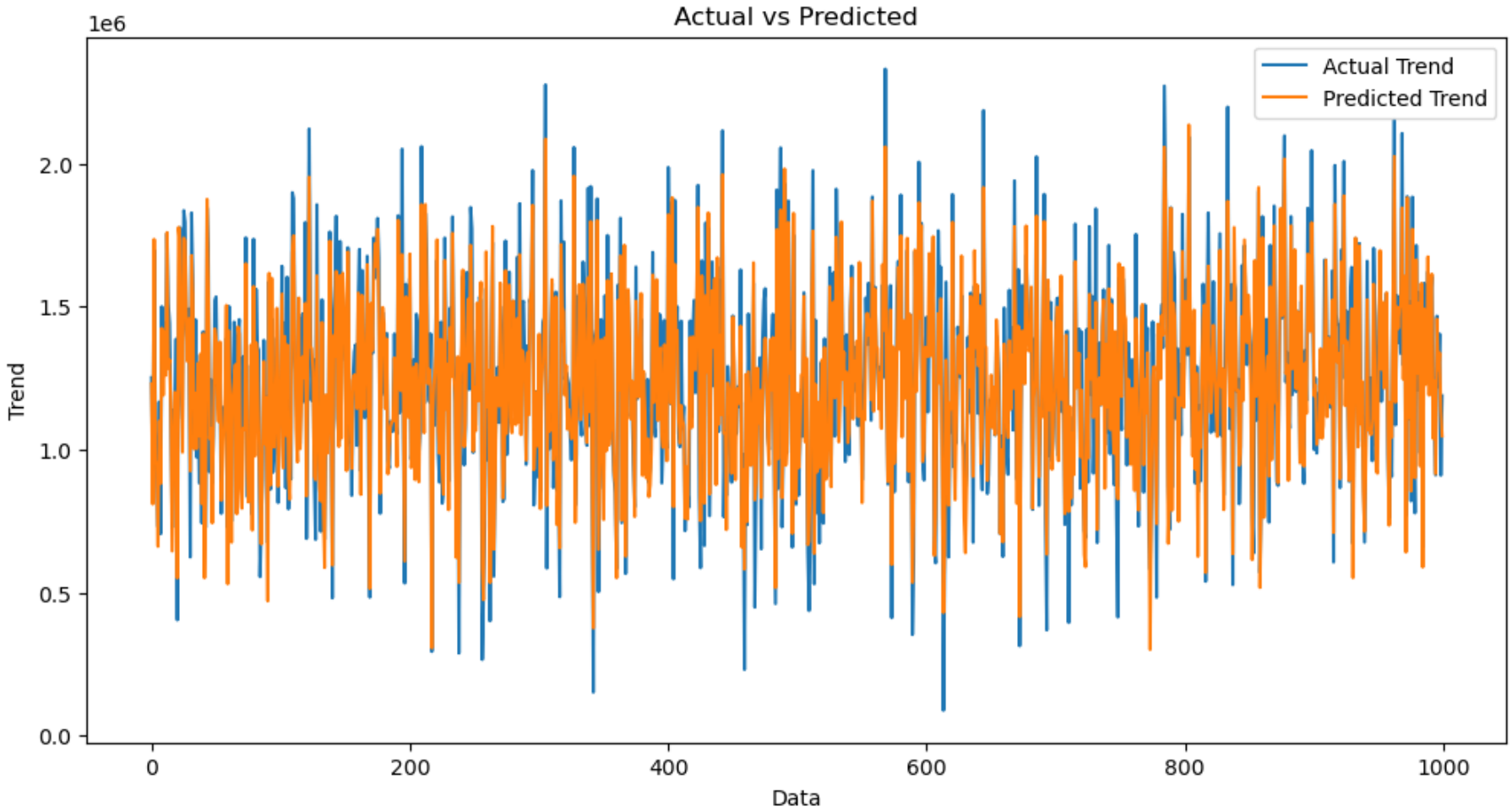
Predicting Prices

In [42]:
```
Prediction4 = model_rf.predict(X_test_scal)
```

Evaluation of Predicted Data

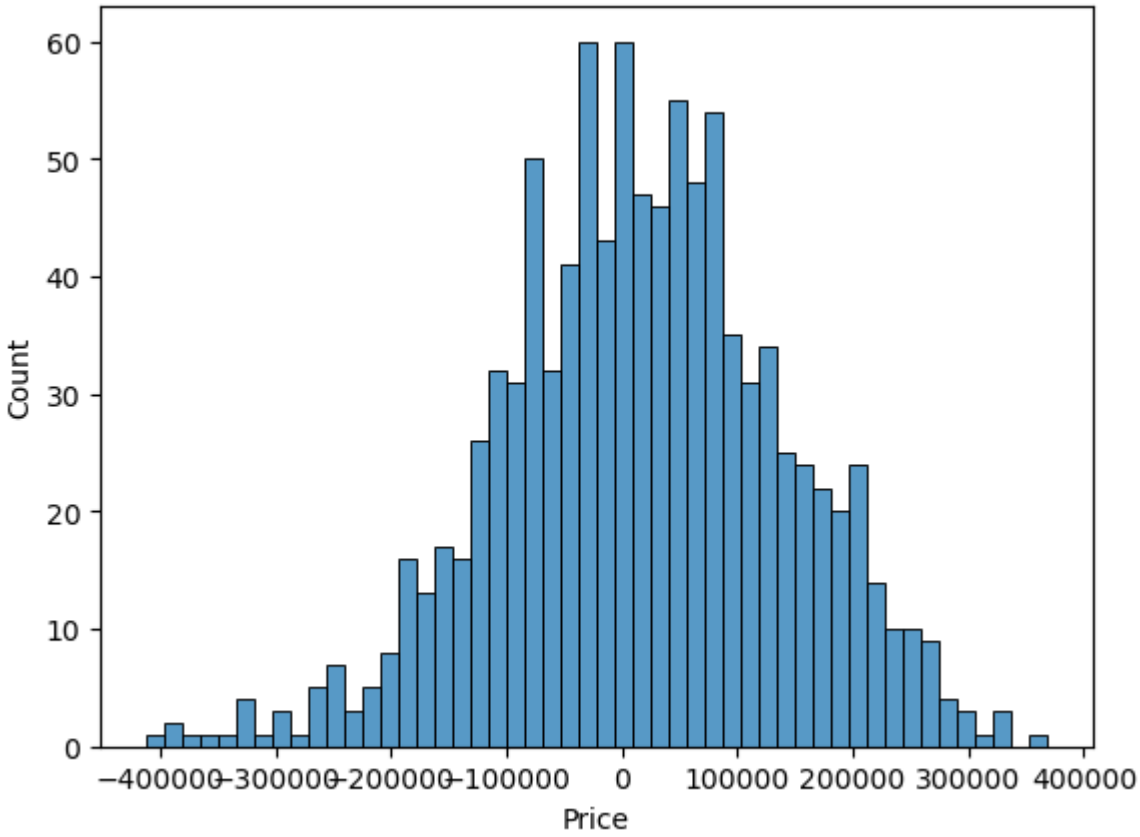
In [43]:
```
    plt.figure(figsize=(12,6))
    plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
    plt.plot(np.arange(len(Y_test)), Prediction4, label='Predicted Trend')
    plt.xlabel('Data')
    plt.ylabel('Trend')
    plt.legend()
    plt.title('Actual vs Predicted')
```

Out[43]:
```
Text(0.5, 1.0, 'Actual vs Predicted')
```


Actual vs Predicted
1e6
Actual Trend
Predicted Trend
Trend
Data
0.0
0.5
1.0
1.5
2.0
0
200
400
600
800
1000

In [44]:
```
sns.histplot((Y_test-Prediction4), bins=50)
```

Out[44]: <Axes: xlabel='Price', ylabel='Count'>



In [45]:
```
print(r2_score(Y_test, Prediction2))
print(mean_absolute_error(Y_test, Prediction2))
print(mean_squared_error(Y_test, Prediction2))
```

```
-0.0006222175925689744
286137.81086908665
128209033251.4034
```

Model 5 - XGboost Regressor

In [46]:
```
model_xg = xg.XGBRegressor()
```

In [47]:
```
model_xg.fit(X_train_scal, Y_train)
```

Out[47]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

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.

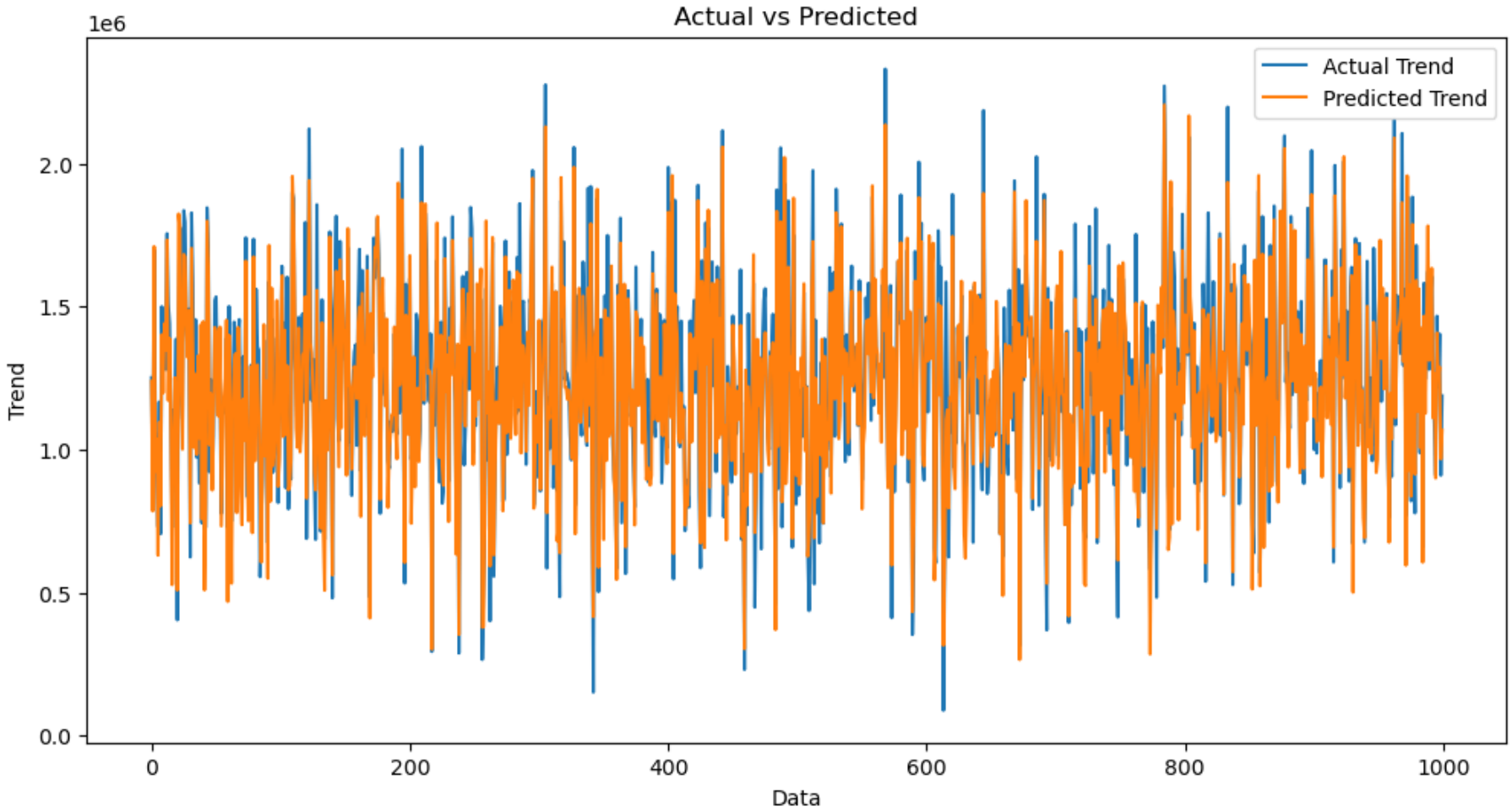
Predicting Prices

In [48]:
```
Prediction5 = model_xg.predict(X_test_scal)
```

Evaluation of Predicted Data

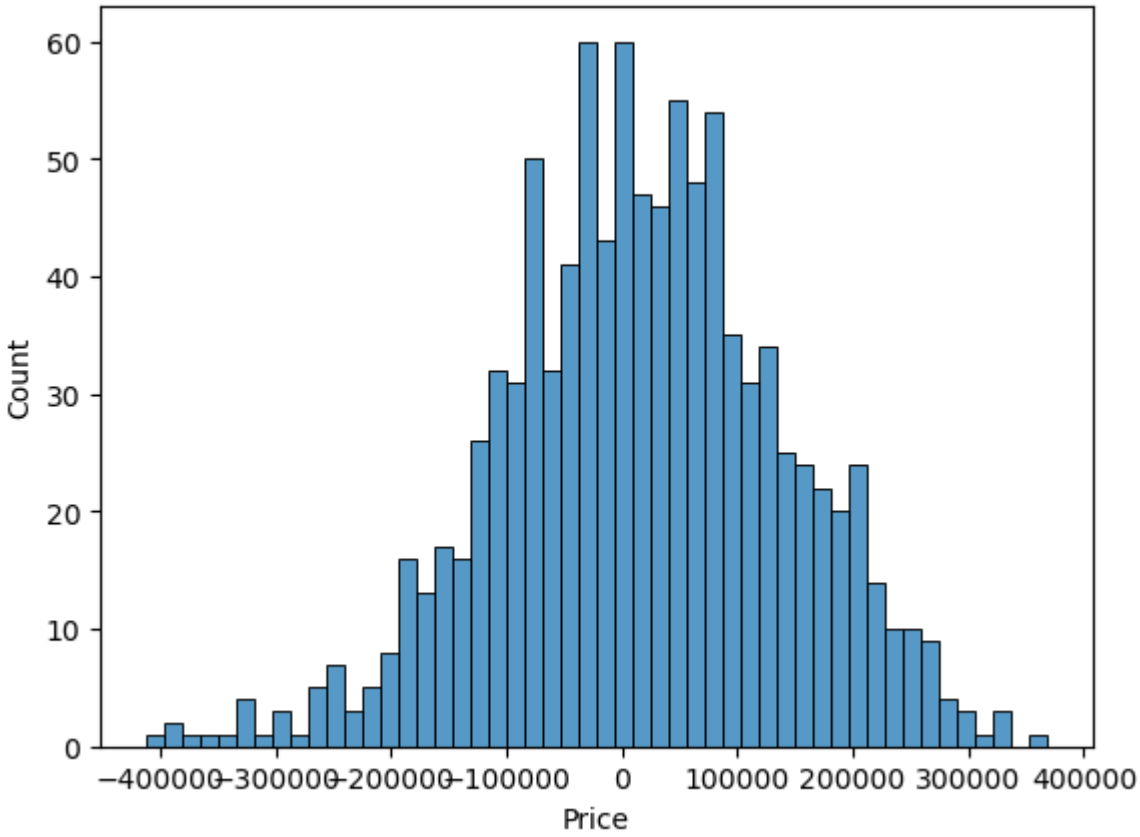
In [49]:
```
    plt.figure(figsize=(12,6))
    plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
    plt.plot(np.arange(len(Y_test)), Prediction5, label='Predicted Trend')
    plt.xlabel('Data')
    plt.ylabel('Trend')
    plt.legend()
    plt.title('Actual vs Predicted')
```

Out[49]:
```
Text(0.5, 1.0, 'Actual vs Predicted')
```


Actual vs Predicted
1e6
Actual Trend
Predicted Trend
Trend
Data
0.0
0.5
1.0
1.5
2.0
0
200
400
600
800
1000

In [50]:
```
sns.histplot((Y_test-Prediction4), bins=50)
```

Out[50]:
```
<Axes: xlabel='Price', ylabel='Count'>
```



In [51]:
```
print(r2_score(Y_test, Prediction2))
print(mean_absolute_error(Y_test, Prediction2))
print(mean_squared_error(Y_test, Prediction2))
```

```
-0.0006222175925689744
286137.81086908665
128209033251.4034
```

Linear Regression is giving us best Accuracy.

9. Conclusion

Summary and Insights

In the conclusion, we summarize the findings from our model evaluations. We provide insights into the strengths and weaknesses of each model, helping the reader make an informed decision about which model to use for house price prediction based on the dataset and the evaluation results. We also offer recommendations for next steps and further areas of exploration in the field of house price prediction.