PREDICTING HOUSE PRICES USING MACHINE LEARNING

Phase 5 Submission

PROBLEM STATEMENT:

* The problem is to create a machine learning model that can accurately predict house prices based on various features such as square footage, number of bedrooms, location, and other relevant factors. This is valuable for real estate professionals, buyers, and sellers who want to estimate the price of a house without relying solely on traditional methods like appraisals.

DESIGN THINKING PROCESS:

**1.Empathize:**

* Understand the needs and pain points of potential users (e.g., homebuyers, sellers, real estate agents).
* Gather data and feedback on what features or factors are most important in predicting house prices.

**2.Define:**

* Clearly define the problem and objectives, such as the target accuracy for price predictions.
* Identify the key features and data sources required for the model.

**3.Ideate:**

* Brainstorm different machine learning algorithms and techniques suitable for regression tasks.
* Consider feature engineering, data preprocessing, and model evaluation approaches.

**4.Prototype:**

* Develop a prototype machine learning model using a subset of the data.
* Experiment with various algorithms (e.g., Linear Regression, Decision Trees, Random Forest) to determine the best-performing one.

**5.Test:**

* Evaluate the prototype model's performance through metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared.
* Gather user feedback and iterate on the model as needed.

**6.Implement:**

* Build the full-scale machine learning model using the chosen algorithm and all available data.
* Create a user-friendly interface or integration for end-users to access the model's predictions.

**7.Feedback:**

* Continuously gather feedback from users to improve the model and its predictions.
* Make adjustments to the model and its features based on changing market conditions.

PHASES OF DEVELOPMENT:

**1.Data Collection:**

* Gather a comprehensive dataset that includes features like square footage, number of bedrooms, bathrooms, location, historical sales data, and other relevant variables.

**2.Data Preprocessing:**

* Clean the data by handling missing values, outliers, and ensuring data quality.
* Feature engineering: Create new features or transform existing ones to improve the model's predictive power.

**3.Model Selection:**

* Choose a regression algorithm (e.g., Linear Regression, Random Forest, Gradient Boosting) that is suitable for the problem.
* Split the data into training and testing sets for model evaluation.

**4.Model Training:**

* Train the selected model on the training data, optimizing hyperparameters as needed.
* Cross-validation can be used to fine-tune the model.

**5.Model Evaluation:**

* Assess the model's performance using evaluation metrics such as MAE, MSE, and R-squared on the testing dataset.
* Iterate on the model or try different algorithms if necessary.

**6.Deployment:**

* Deploy the trained model to a platform or application where users can input house features and receive price predictions.

**7.Maintenance:**

* Regularly update the model with new data to adapt to changing market conditions.
* Continuously monitor its performance and address issues as they arise.

DATASET DESCRIPTION:

* The dataset for predicting house prices typically includes features related to properties (e.g., size, number of bedrooms, location) and the corresponding sale prices. The dataset can be obtained from sources like real estate websites, government housing agencies, or compiled datasets.

DATA PREPROCESSING STEPS:

1.Data Collection:

* Gather the dataset with information on various houses, including features such as square footage, number of bedrooms, bathrooms, location, year built, and sale prices.

2.Data Cleaning:

* Handle missing values, remove duplicates, and deal with outliers. Missing data can be filled using imputation techniques, and outliers can be treated through methods like removing extreme values or transforming them.

3.Data Exploration:

* Analyze the dataset to understand data distributions and relationships. Visualizations can help identify patterns, correlations, and potential features that influence house prices.

4.Feature Engineering:

* Create new features or transform existing ones to make them more suitable for modeling. For instance, you can create features like price per square foot, age of the house, or one-hot encode categorical variables like the type of neighborhood.

5.Data Splitting:

* Split the dataset into training, validation, and test sets to evaluate the model's performance. Common splits include 70-80% for training, 10-15% for validation, and 10-15% for testing.

6.Scaling:

* Normalize or standardize numerical features to ensure that they have similar scales, which can help gradient descent-based algorithms converge faster.

MODEL TRAINING PROCESS

1.Choose a Model:

* Select a regression model suitable for predicting house prices. Common choices include linear regression, decision trees, random forests, or more advanced models like gradient boosting, support vector regression, or neural networks.

2.Model Architecture:

* For more complex models like neural networks, define the architecture with appropriate layers, units, and activation functions. Simpler models like linear regression may not require extensive architecture design.

3.Loss Function:

* Choose a loss function suitable for regression tasks, such as mean squared error (MSE) or mean absolute error (MAE), which measures the difference between predicted and actual prices.

4.Optimizer:

* Select an optimization algorithm, typically gradient descent variants, to update model parameters during training.

5.Training:

* Train the model on the training data, minimizing the chosen loss function. The model will learn to predict house prices based on the features.

6.Validation:

* Monitor the model's performance on the validation set during training to detect overfitting. Adjust hyperparameters like learning rate, regularization strength, or model complexity to improve performance.

7.Testing:

* Evaluate the model's performance on the test set to assess its generalization capabilities. Metrics like RMSE (Root Mean Squared Error) or MAE can quantify the prediction accuracy.

8.Deployment:

* If the model performs well, you can deploy it in a real estate application to predict house prices based on input features.

Remember that the specific preprocessing steps and choice of model can vary based on the dataset and problem requirements, and iterative refinement is often necessary to achieve the best predictive performance.

REGRESSION ALGORITHMS:

1.Linear Regression:

* This is a simple and interpretable choice. It assumes a linear relationship between features and the target variable. It can work well if the relationship is approximately linear.

2.Decision Trees and Random Forests:

* These can capture non-linear relationships and interactions between features. Random Forests, in particular, are robust and provide feature importance scores.

3.Gradient Boosting Regressors (e.g., XGBoost, LightGBM, AdaBoost):

* These ensemble methods often yield state-of-the-art results. They can handle complex relationships, outliers, and missing data effectively.

4.Support Vector Regression:

* It's useful when you need to deal with high-dimensional data and complex relationships.

5.Neural Networks:

* Deep learning models can be used for regression, especially for very complex, large datasets. They may require extensive data preprocessing and tuning.

EVALUATION METRICS:

1.Mean Absolute Error (MAE):

* This measures the average absolute difference between predicted and actual values. It provides a straightforward interpretation in the same unit as the target variable.

2.Mean Squared Error (MSE):

* MSE penalizes larger errors more than MAE. It's more sensitive to outliers. RMSE (square root of MSE) has the same unit as the target variable.

3.R-squared (R2):

* R-squared measures the proportion of the variance in the target variable explained by the model. A higher R-squared indicates a better fit.

4.Root Mean Squared Logarithmic Error (RMSLE):

* Useful when the target variable has a wide range. It logarithmically scales the target variable before calculating the error.

5.Median Absolute Error:

* It's less sensitive to outliers compared to MAE. It's a robust metric when dealing with skewed datasets.

6.Custom Metrics:

* Depending on the specific problem, you may create custom evaluation metrics. For example, you could penalize underestimation and overestimation differently if your application has asymmetric costs.

The choice of algorithm and metrics should involve experimentation and cross-validation. You can use techniques like k-fold cross-validation to assess the model's performance and choose the best algorithm and metrics combination based on the specific problem and data.

PREDICTING HOUSE PRICES USING LINEAR REGRESSION ALGORITHM

Source:

* Jupyter

Coding:

*# Importing Libraries*

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.simplefilter('ignore')

*# Loading Data*

df=pd.read\_csv("../input/usa-housing/USA\_Housing.csv")

*# Type Info*

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

*# Showing Column names*

df.columns

Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',

'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],

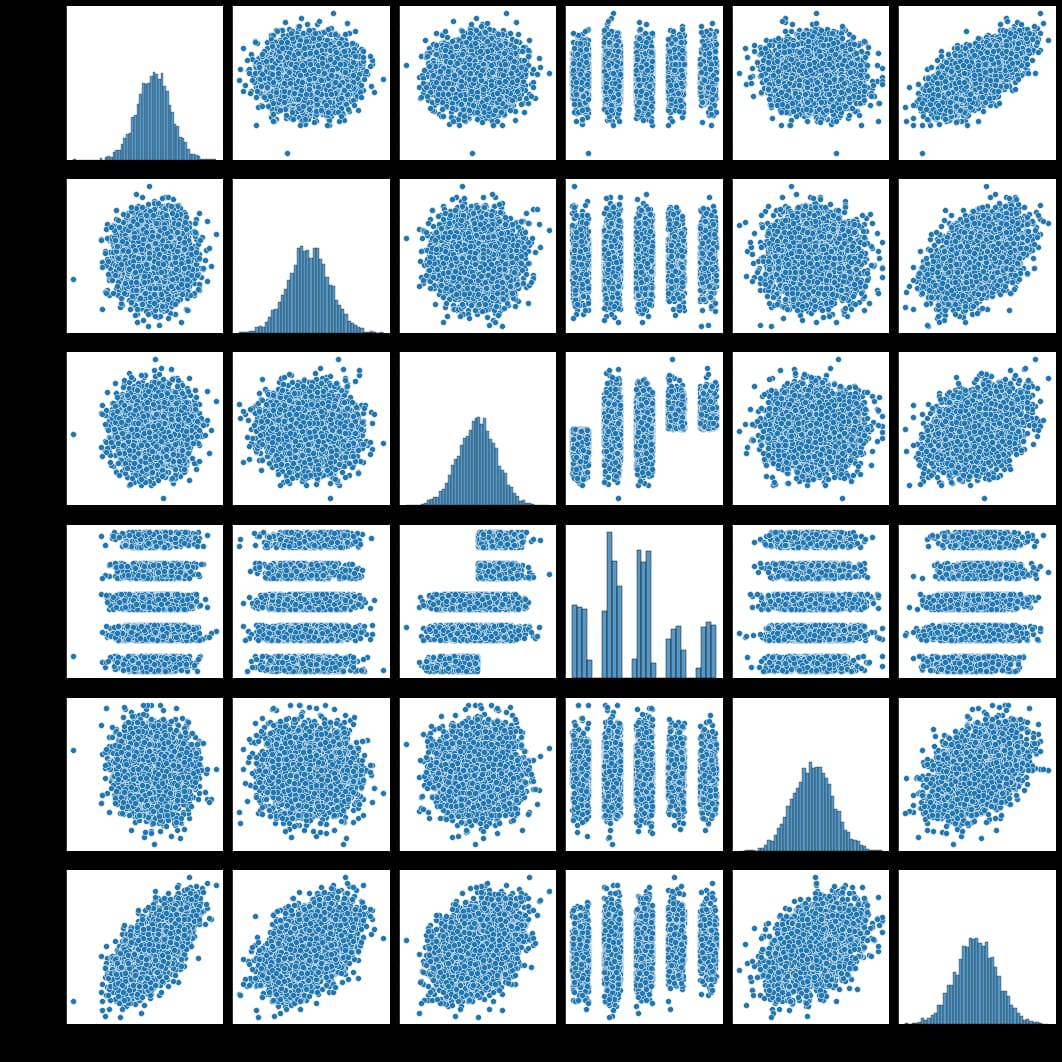
dtype='object')

EXPLORATORY DATA ANALYSIS

*# Pairplot*

sns.pairplot(df)

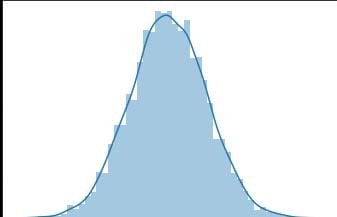
<seaborn.axisgrid.PairGrid at 0x7f13c545fe10>



*# Distribution Plot*

sns.distplot(df["Price"])

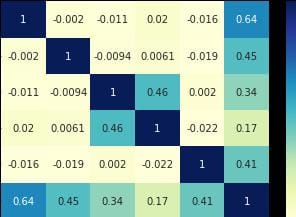
<AxesSubplot:xlabel='Price', ylabel='Density'>



*# Heatmamp by Correlation of Columns*

sns.heatmap(df.corr(),cmap="YlGnBu",annot=True)

<AxesSubplot:>



*# Y*

y=df["Price"]

y

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

*# Importing train\_test\_split to split our data train and test set.*

from sklearn.model\_selection import train\_test\_split

*# Train and Test Set*

x\_train, x\_test, y\_train, y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=42)

*# Importing LinearRegression Model*

from sklearn.linear\_model import LinearRegression

model=LinearRegression()

*# Fitting Model To Our Train Set*

model.fit(x\_train,y\_train)

LinearRegression()

*# Intercept and Coefs of Model*

print(model.intercept\_)

print(model.coef\_)

-2635072.900933358

[2.16522058e+01 1.64666481e+05 1.19624012e+05 2.44037761e+03

1.52703134e+01]

*# Making Predictions by Test Set*

y\_preds=model.predict(x\_test)

*# Importing mean\_absolute\_error to Evaluate Model*

from sklearn.metrics import mean\_absolute\_error

*# We Look Difference Between Actual Values and Predictions to Evaluate Model*

mean\_absolute\_error(y\_test, y\_preds)

80879.0972348982

mean\_absolute\_error(y\_test, y\_preds)/df["Price"].mean()

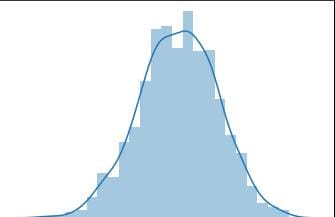
0.06564474664945628

There is %6 difference between the actual and predictions.

*# Creating Residuals Distribution Plot*

sns.distplot(y\_preds-y\_test)

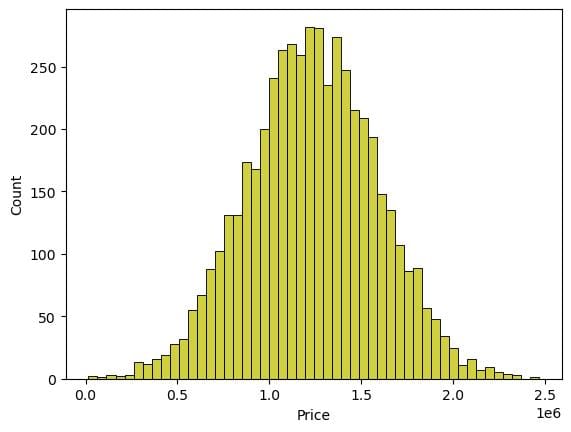
<AxesSubplot:xlabel='Price', ylabel='Density'>



Visualisation and Pre-Processing of Data

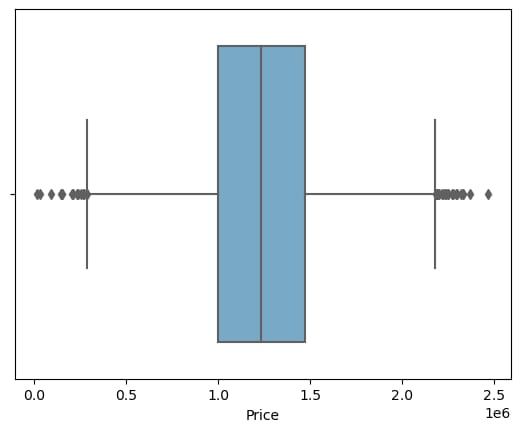
sns.histplot(dataset, x='Price', bins=50, color='y')

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



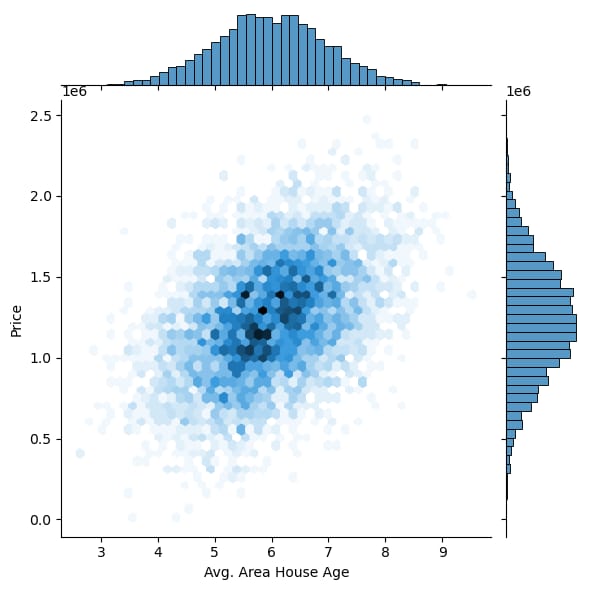
sns.boxplot(dataset, x='Price', palette='Blues')

<Axes: xlabel='Price'>



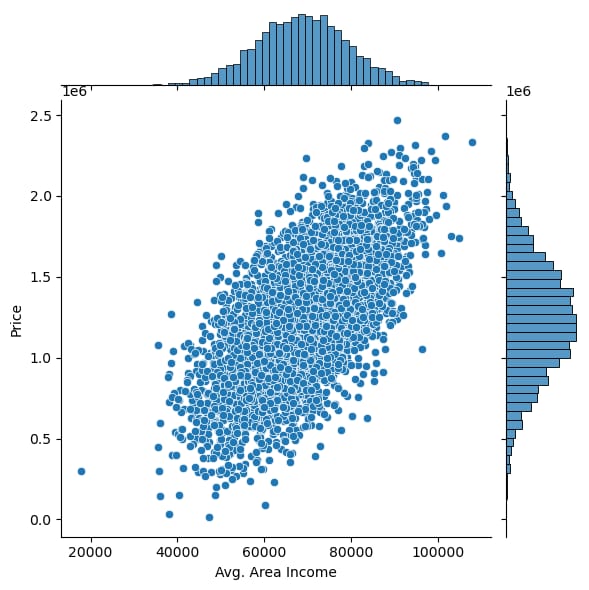
sns.jointplot(dataset, x='Avg. Area House Age', y='Price', kind='hex')

<seaborn.axisgrid.JointGrid at 0x7b5e769d94b0>



sns.jointplot(dataset, x='Avg. Area Income', y='Price')

<seaborn.axisgrid.JointGrid at 0x7b5e65a10340>



dataset.hist(figsize=(10,8))

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

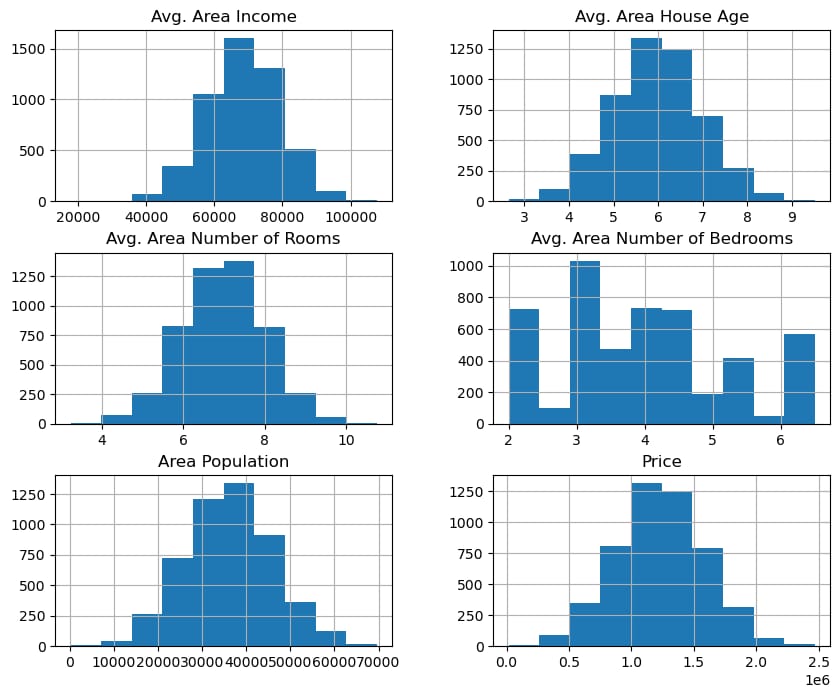
<Axes: title={'center': 'Avg. Area House Age'}>],

[<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)



## Visualising Correlation

plt.figure(figsize=(10,5))

sns.heatmap(dataset.corr(numeric\_only = True), annot=True)



# Dividing Dataset in to features and target variable

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

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=101)

Y\_train.head()

3413 1.305210e+06

1610 1.400961e+06

3459 1.048640e+06

4293 1.231157e+06

1039 1.391233e+06

Name: Price, dtype: float64

Y\_train.shape

(4000,)

Y\_test.head()

1718 1.251689e+06

2511 8.730483e+05

345 1.696978e+06

2521 1.063964e+06

54 9.487883e+05

Name: Price, dtype: float64

Y\_test.shape

(1000,)

# Standardizing the data

sc = StandardScaler()

X\_train\_scal = sc.fit\_transform(X\_train)

X\_test\_scal = sc.fit\_transform(X\_test)

# Model Building and Evaluation

## Random Forest Regressor

model\_rf = RandomForestRegressor(n\_estimators=50)

model\_rf.fit(X\_train\_scal, Y\_train)

## Predicting Prices

Prediction4 = model\_rf.predict(X\_test\_scal)

## Evaluation of Predicted Data

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')

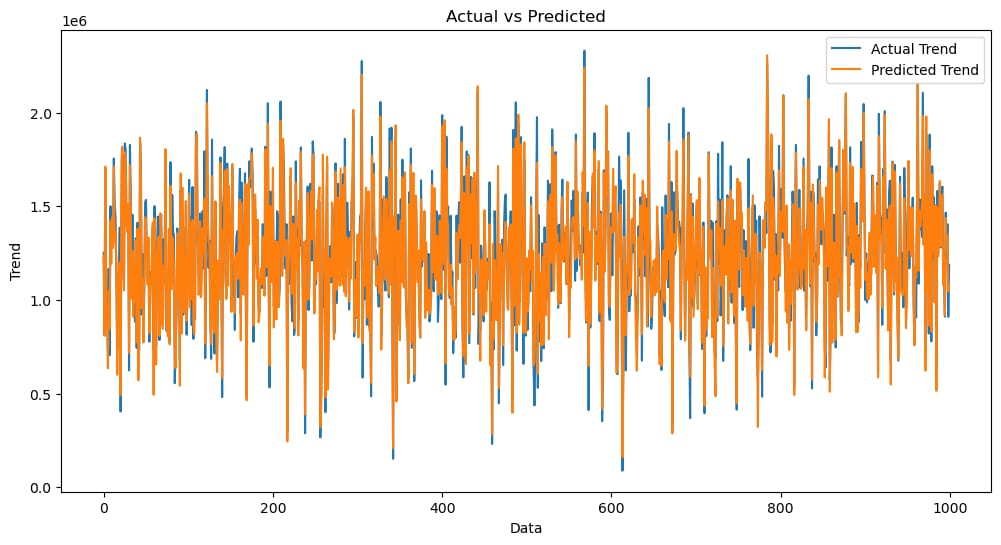
plt.xlabel('Data')

plt.ylabel('Trend')

plt.legend()

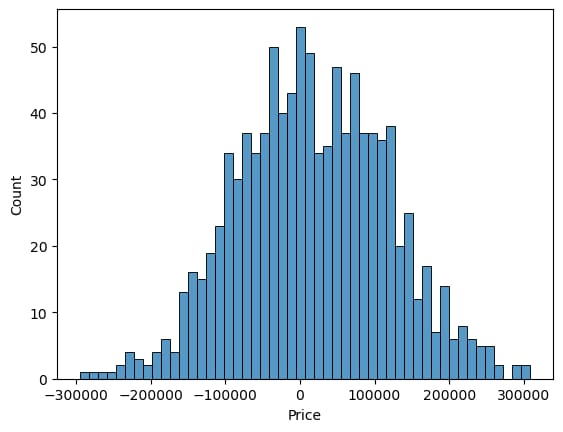
plt.title('Actual vs Predicted')

Text(0.5, 1.0, 'Actual vs Predicted')



sns.histplot((Y\_test-Prediction4), bins=50)

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



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