HOUSE PRICE PREDICTION ANALYSIS USING MACHINE LEARNING

PHASE 4: DEVELOPMENT PART 2

Continue building the house price prediction model by feature selection, model training, and evaluation.

Project Title: House Price Predictor

INTRODUCTION:

The housing market is an important and complex sector that impacts people's lives in many ways. For many individuals and families, buying a house is one of the biggest investments they will make in their lifetime. Therefore, it is essential to accurately predict the prices of houses so that buyers and sellers can make informed decisions. This project aims to use machine learning techniques to predict house prices based on various features such as location, square footage, number of bedrooms and bathrooms, and other relevant factors

COLAB LINK:

https://colab.research.google.com/drive/1bE0cFUY6tFo317FFmzpjncu8Mxi1TkC-#scrollTo=sS0klj7JrPJd

PROGRAM:

Importing dataset:

Loading data is a crucial step in any data analysis or machine learning task. It involves bringing external datasets into your programming environment so that you can manipulate, analyze, and draw insights from the data.

```
[ ] 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
```

```
[ ] dataset = pd.read_csv('USA_Housing.csv')
```

```
dataset.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386

Data cleaning techniques:

Data cleaning is the process of identifying and correcting errors, inconsistencies, and inaccuracies in datasets. It's a crucial step in the data analysis pipeline, as the quality of your analysis depends heavily on the quality of your data.

- Handling missing values
- Dealing with duplicates
- Handling Outliers
- Data type Conversion

```
dataset.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
4995    False
4996    False
4997    False
4998    False
4999    False
Length: 5000, dtype: bool
```

```
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
```

```
dataset.describe()
```

	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%	61480.562390	5.322283	6.299250	3.140000	29403.928700	9.975771e+05
50%	68804.286405	5.970429	7.002902	4.050000	36199.406690	1.232669e+06
75%	75783.338665	6.650808	7.665871	4.490000	42861.290770	1.471210e+06
max	107701.748400	9.519088	10.759588	6.500000	69621.713380	2.469066e+06

```
# Categorical columns
cat_col = [col for col in dataset.columns if dataset[col].dtype == 'object']
print('Categorical columns :',cat_col)
# Numerical columns
num_col = [col for col in dataset.columns if dataset[col].dtype != 'object']
print('Numerical columns :',num_col)
```

```
Categorical columns : ['Address']
Numerical columns : ['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population', 'Price']
```

```
[ ] dataset[cat_col].nunique()
```

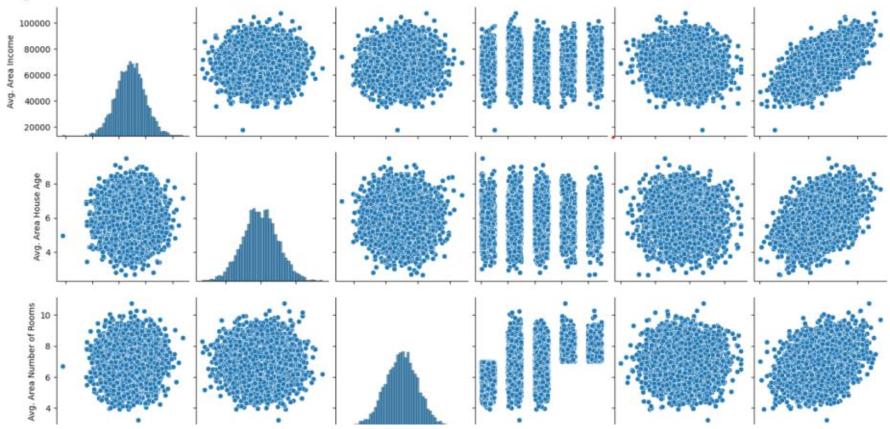
```
Address     5000
dtype: int64
```

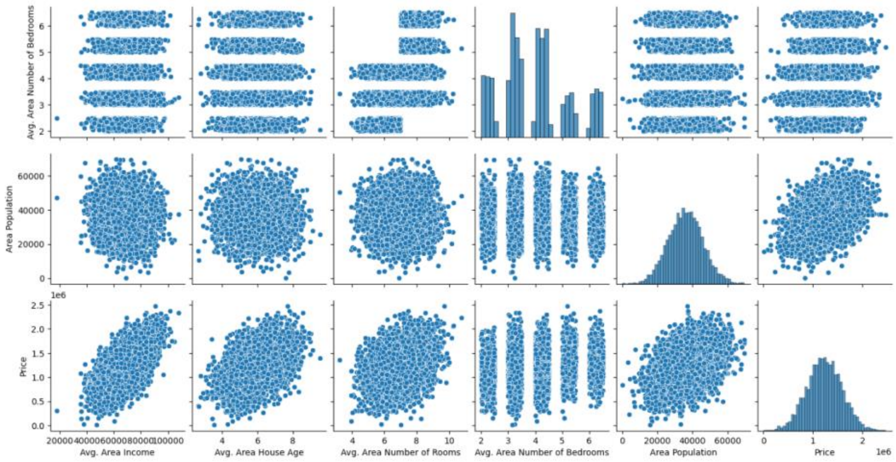
```
round((dataset.isnull().sum()/dataset.shape[0])*100,2)
```

```
Avg. Area Income                0.0
Avg. Area House Age             0.0
Avg. Area Number of Rooms       0.0
Avg. Area Number of Bedrooms    0.0
Area Population                 0.0
Price                           0.0
Address                         0.0
dtype: float64
```

```
plt.figure(figsize=(12,8))
sns.pairplot(dataset)
```

```
<seaborn.axisgrid.PairGrid at 0x79e7fe5cf100>
<Figure size 1200x800 with 0 Axes>
```





Data Analysis:

Data analysis is the process of inspecting, cleaning, transforming, and modeling data to uncover useful information, draw conclusions, and support decision-making.

- Data visualization
- Exploratory Data Analysis

```
[ ] def mean(df):
        return sum(dataset.Price)/len(dataset)
    print(mean(dataset))
```

```
1232072.6541452995
```

```
median_value = np.median(dataset['Price'])
print(median_value)
```

```
1232669.378
```

```
import statistics
mode_result = statistics.mode(dataset['Price'])

print(f'Mode: {mode_result}')
```

```
Mode: 1059033.558
```

```
std_deviation = np.std(dataset['Price'])
print(f"Standard Deviation: {std_deviation}")
```

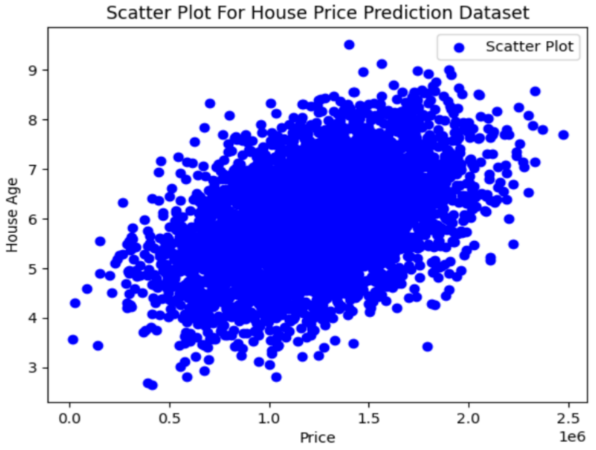
```
Standard Deviation: 353082.3130552725
```

```
percentiles = np.percentile(dataset['Price'], [25, 50, 75])
print(f"25th Percentile (Q1): {percentiles[0]}")
print(f"50th Percentile (Q2 or Median): {percentiles[1]}")
print(f"75th Percentile (Q3): {percentiles[2]}")
```

```
25th Percentile (Q1): 997577.135075
50th Percentile (Q2 or Median): 1232669.378
75th Percentile (Q3): 1471210.2045
```

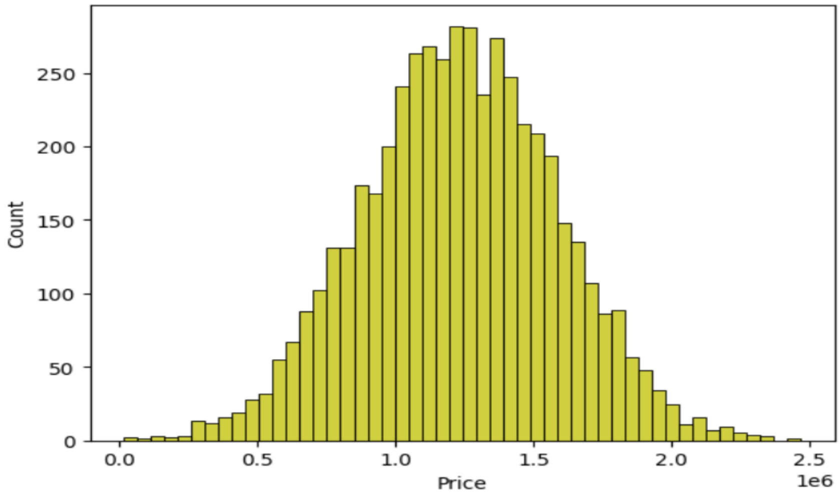
```python
x_values = dataset['Price']
y_values = dataset['Avg. Area House Age']

plt.scatter(x_values, y_values, c='blue', label='Scatter Plot')
plt.title('Scatter Plot For House Price Prediction Dataset')
plt.xlabel('Price')
plt.ylabel('House Age')
plt.legend()
plt.show()
```




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

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



Data Preprocessing techniques:

Data preprocessing is a crucial step in the data analysis and machine learning pipeline. It involves cleaning and transforming raw data into a format suitable for analysis or model training. The goal is to enhance the quality of the data, address issues like missing values and outliers, and prepare it for effective exploration and modeling.

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
for column in dataset.columns:
    dataset[column] = labelencoder.fit_transform(dataset[column])
```

```
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   int64
 1   Avg. Area House Age        5000 non-null   int64
 2   Avg. Area Number of Rooms  5000 non-null   int64
 3   Avg. Area Number of Bedrooms  5000 non-null   int64
 4   Area Population            5000 non-null   int64
 5   Price                      5000 non-null   int64
 6   Address                    5000 non-null   int64
dtypes: int64(7)
memory usage: 273.6 KB
```

Splitting the data:

To do this, you split your dataset into two main parts: a training set and a testing set.

1.Training Set:

•This is the portion of your data used to train your model. The model learns patterns, relationships, and features from this set.

•The idea is that by exposing your model to a sufficient amount of data, it should be able to learn and understand the underlying patterns in the information.

2.Testing Set:

•This set is reserved to evaluate how well your model performs on unseen data.

•Once your model is trained, you use the testing set to see how accurately it can make predictions or classifications.

•The testing set serves as a simulation of real-world scenarios where your model encounters new, previously unseen examples.

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

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

```
Y_train.head()
```

```
3413    2915
1610    3410
3459    1503
4293    2491
1039    3360
Name: Price, dtype: int64
```

```
[40] Y_train.shape
```

```
(4000,)
```

```
Y_test.head()
```

```
1718    2628
2511     786
345     4524
2521    1596
54      1049
Name: Price, dtype: int64
```

```
Y_test.shape
```

```
(1000,)
```

Standard Scalar:

Standard Scalar, or standardization, is a technique used in machine learning to scale and center the attributes or features of a dataset. The goal is to ensure that the features have the same scale or distribution.

```
sc = StandardScaler()
X_train_scal = sc.fit_transform(X_train)
X_test_scal = sc.fit_transform(X_test)
```

Linear regression:

Linear regression is like fitting a straight line through a cloud of points. It's a simple yet powerful method in statistics and machine learning used for predicting a continuous outcome variable (dependent variable) based on one or more predictor variables (independent variables).

```
[45] from sklearn.metrics import r2_score, mean_absolute_error,mean_squared_error
     from sklearn.linear_model import LinearRegression
```

The objective is to find the best-fitting line that minimizes the sum of squared differences between the observed and predicted values.

```
[46] model_lr=LinearRegression()
```

Predictions:

Once trained, the model can make predictions for new, unseen data. You input a value for x, and the model predicts the corresponding y.

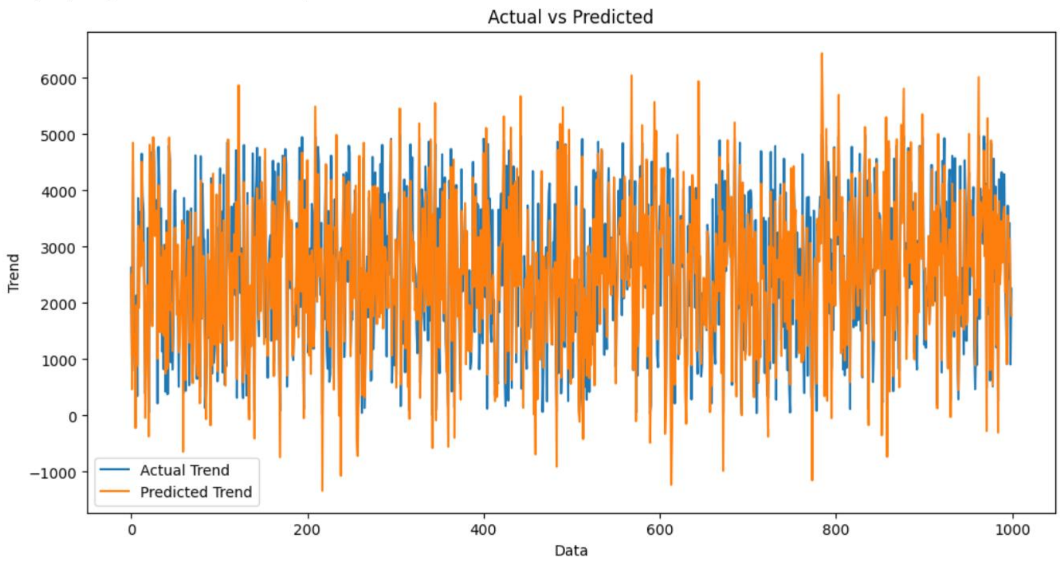
```
model_lr.fit(X_train_scal, Y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
Prediction1 = model_lr.predict(X_test_scal)
```

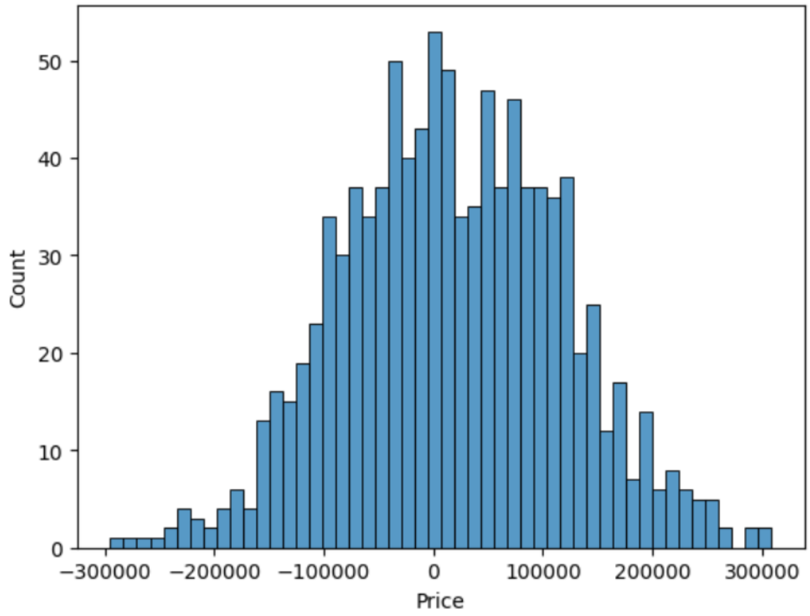
```
[50] plt.figure(figsize=(12,6))
     plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
     plt.plot(np.arange(len(Y_test)), Prediction1, label='Predicted Trend')
     plt.xlabel('Data')
     plt.ylabel('Trend')
     plt.legend()
     plt.title('Actual vs Predicted')
```

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




```
sns.histplot((Y_test-Prediction1), bins=50)
```

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



Evaluation:

Common metrics for evaluating linear regression models include Mean Squared Error (MSE) and R-squared. MSE measures the average squared difference between predicted and actual values, while R-squared represents the proportion of the variance in the dependent variable that is predictable from the independent variables.

```
print(r2_score(Y_test, Prediction1))
print(mean_absolute_error(Y_test, Prediction1))
print(mean_squared_error(Y_test, Prediction1))
```

```
0.9182928179527469
82295.49777553751
10469084771.329184
```

Linear regression is a great starting point for many predictive modeling tasks, and it forms the foundation for more complex models. It's widely used in various fields due to its simplicity and interpretability.

Support vector Regressor:

Support Vector Regression (SVR) is a type of machine learning model that utilizes support vector machines for regression tasks. Similar to Support Vector Machines (SVM) for classification, SVR aims to find a hyperplane that best fits the data while minimizing the error between the predicted and actual values.

```
[35] from sklearn.svm import SVR
```

```
[36] model_svr = SVR()
```

The primary goal of SVR is to find a hyperplane that best represents the trend in the data, while allowing for a margin of error.

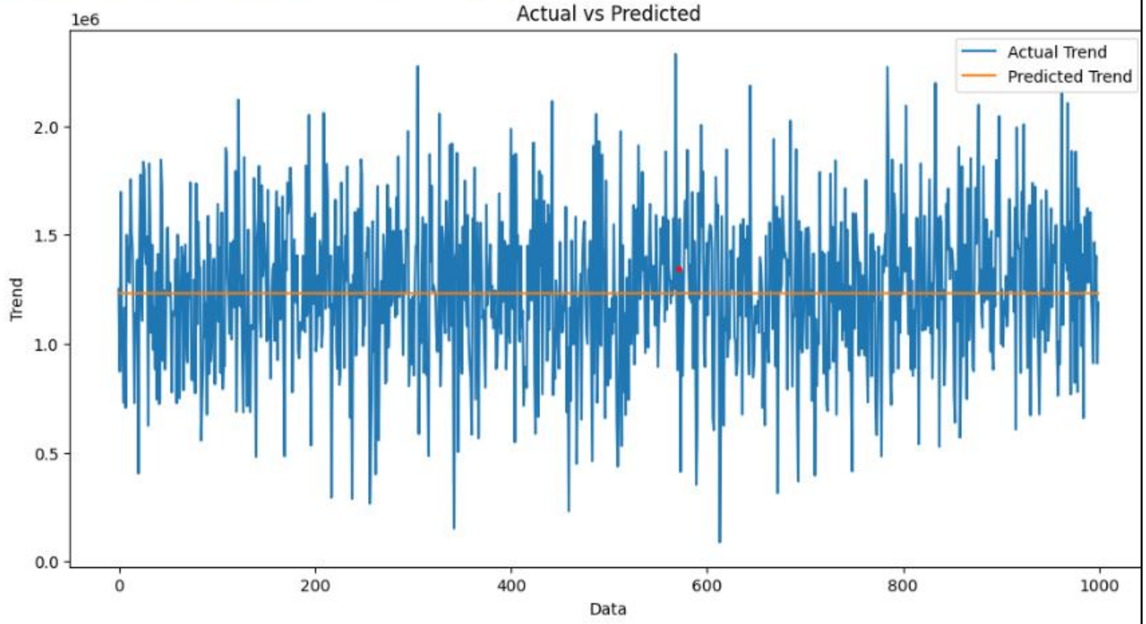
```
[37] model_svr.fit(X_train_scal, Y_train)
```

```
▾ SVR
SVR()
```

```
[38] Prediction2 = model_svr.predict(X_test_scal)
```

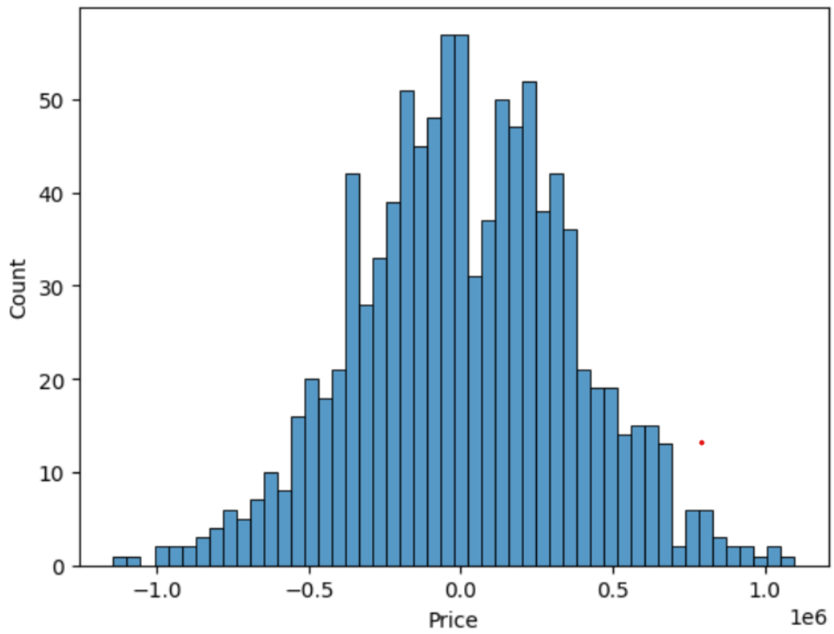
```
[39] plt.figure(figsize=(12,6))
     plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
     plt.plot(np.arange(len(Y_test)), Prediction2, label='Predicted Trend')
     plt.xlabel('Data')
     plt.ylabel('Trend')
     plt.legend()
     plt.title('Actual vs Predicted')
```

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




```
sns.histplot((Y_test-Prediction2), bins=50)
```

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




```
[41] print(r2_score(Y_test, Prediction2))
     print(mean_absolute_error(Y_test, Prediction2))
     print(mean_squared_error(Y_test, Prediction2))
```

```
-0.000622217544275383
286137.8108616177
128209033246.16103
```

SVR is particularly useful when dealing with datasets where the relationship between the features and the target variable is complex and nonlinear. It's a powerful regression technique, and the choice of the kernel function (linear, polynomial, radial basis function, etc.) can significantly impact its performance on different types of data.

Random Forest Regressor:

Random Forest Regression is like the wisdom of the crowd applied to predicting numbers. It's an ensemble learning method that combines the predictions of multiple decision trees to improve accuracy and robustness in regression tasks.

```
[42] from sklearn.ensemble import RandomForestRegressor
```

Objective:

The primary objective of a Random Forest Regressor is to build an ensemble of decision trees that collectively make accurate predictions for a regression task. Each decision tree is trained on a subset of the data and features, and the final prediction is an average or a weighted average of the predictions of individual trees.

```
[43] model_rf = RandomForestRegressor(n_estimators=50)
```

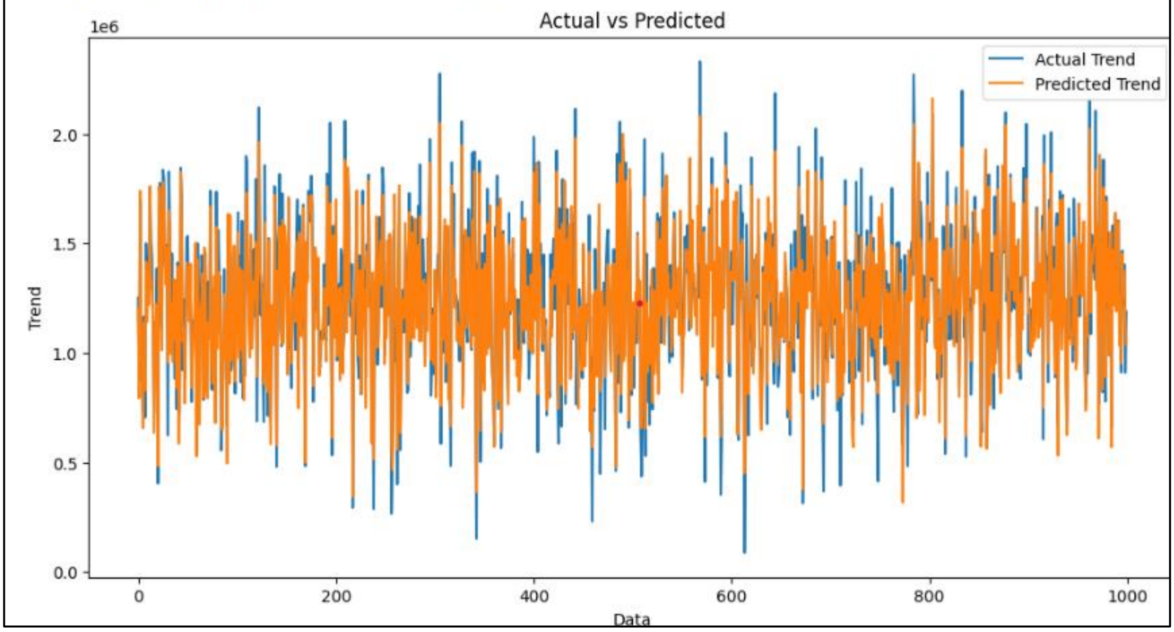
```
model_rf.fit(X_train_scal, Y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor(n_estimators=50)
```

```
[45] Prediction4 = model_rf.predict(X_test_scal)
```

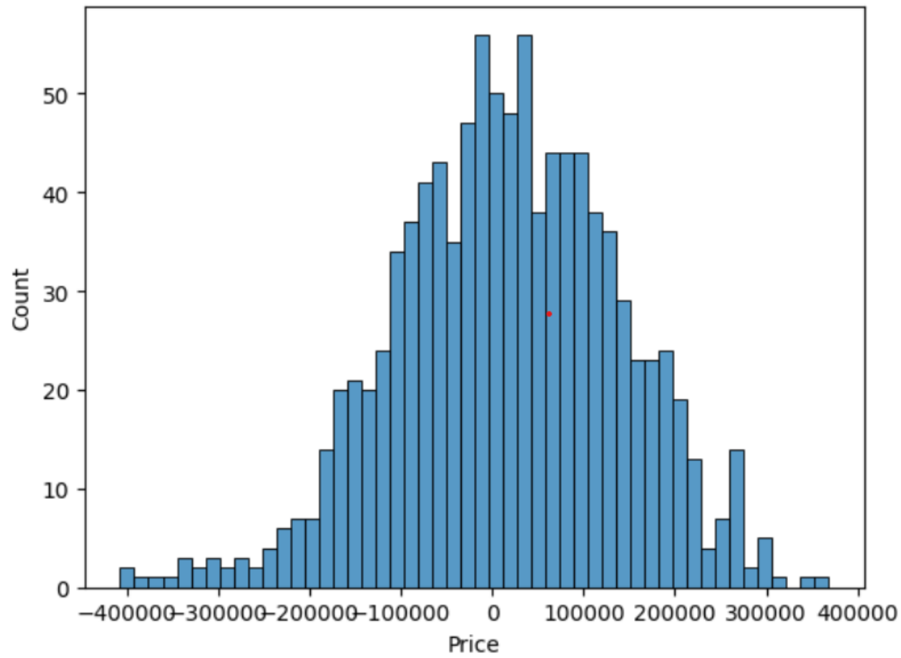
```
[46] plt.figure(figsize=(12,6))
     plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
     plt.plot(np.arange(len(Y_test)), Prediction4, label='Predicted Trend')
     plt.xlabel('Data')
     plt.ylabel('Trend')
     plt.legend()
     plt.title('Actual vs Predicted')
```

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




```
sns.histplot((Y_test-Prediction4), bins=50)
```

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




```
[48] print(r2_score(Y_test, Prediction4))
     print(mean_absolute_error(Y_test, Prediction4))
     print(mean_squared_error(Y_test, Prediction4))
```

```
0.8785992367029718
99585.8014007088
15554995906.303984
```

The "random" in Random Forest is key to its success—it adds an element of diversity that prevents overfitting and improves generalization.

XGBoost:

XGBoost is the gradient boosting algorithms—it's powerful, versatile, and can tackle a wide range of machine learning tasks

```
[49] import xgboost as xg
```

Objective:

The fundamental objective of gradient boosting is to create a strong predictive model by combining the outputs of multiple weak models (typically decision trees) in an additive manner.

```
[50] model_xg = xg.XGBRegressor()
```

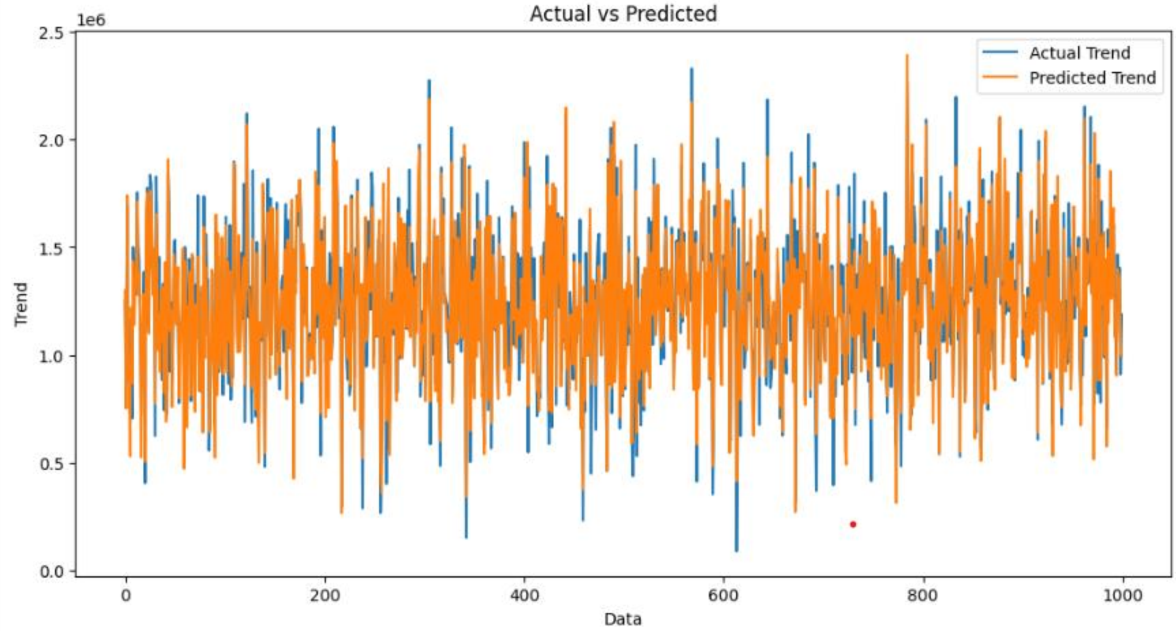
```
model_xg.fit(X_train_scal, Y_train)
```

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=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,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```
[52] Prediction5 = model_xg.predict(X_test_scal)
```

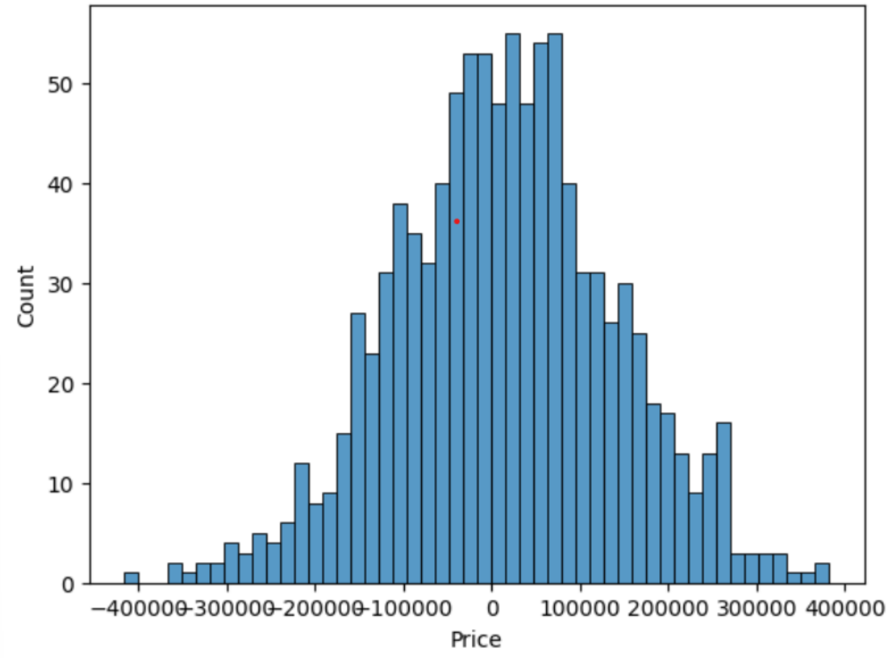
```
plt.figure(figsize=(12,6))
plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
plt.plot(np.arange(len(Y_test)), Prediction5, label='Predicted Trend')
plt.xlabel('Data')
plt.ylabel('Trend')
plt.legend()
plt.title('Actual vs Predicted')
```

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




```
54] sns.histplot((Y_test-Prediction5), bins=50)
```

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




```
print(r2_score(Y_test, Prediction5))
print(mean_absolute_error(Y_test, Prediction5))
print(mean_squared_error(Y_test, Prediction5))
```

```
0.8749027860724268
100138.43696774
16028619571.134682
```

XGBoost optimizes an objective function, which includes a loss function that measures the difference between predicted and actual values, regularization terms to control model complexity, and a component for each tree that corrects errors in the current ensemble. The iterative process of adding trees and optimizing the objective function results in a highly accurate and robust predictive model.

```
Linear Regression is giving us best Accuracy.
```

Linear regression is a straightforward and interpretable model that works well in many scenarios, particularly when the relationship between the input features and the target variable is approximately linear.