**PREDICTING THE HOUSE PRICES USING MACHINE LEARNING**

**INTRODUCTION:**

* Predicting house prices using machine learning involves using algorithms and data to estimate the market value of a property based on its features, such as location, size, and amenities.
* Key steps include data pre processing, feature engineering, model selection, and rigorous evaluation using metrics like Mean Absolute Error (MAE) or Mean Squared Error (MSE).
* Ensuring model interpretability and addressing ethical considerations, such as bias, are essential. Deployment in a user-friendly interface and continuous monitoring for updates and market trends are crucial for maintaining accuracy in real-world applications.

**Loading Dataset** :

**1. Data Collection:**

* Gather a comprehensive dataset that includes relevant information about houses. This could encompass data sources such as real estate list property tax records or publicly available datasets.
* Ensure that your dataset has a variety of features including both numerical (e.g., square footages number of bedrooms) and categorical (e.g., locations type of house) variables.
* Pay attention to data quality as missing or inaccurate data can significantly impact the performance of your model.

|  | 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... |

**INPUT:**

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

**INPUT :**

dataset.describe()

**OUTPUT:**

|  | **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 |

INPUT :

dataset.columns

OUTPUT :

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:**

**2. Data Pre-processing:**

* **Data Cleaning:**
* Identify and handle missing data: You can choose to input missing values with averages or medians for numerical features and use mode for categorical features or you may decide to remove rows or columns with excessive missing data.
* Outlier detection and treatment: Identify outliers in the data and decide whether to remove them transform them or leave them as-is based on domain knowledge.
* Error correction: Check for data entry errors and correct them if necessary.
* **Scaling Feature and Transformation:**
* Standardization: Standardize numerical features to have a mean of 0 and a standard deviation of 1. This is important for algorithms sensitive to feature scales like gradient descent-based methods.
* Normalization: Normalize features to a specific ranges like [0s 1s] if needed.
* on transformation: Apply on transformations to features that exhibit skewed distributions which can help improve model performance.
* **Categorical Encoding:**
* One-Hot Encoding: Convert categorical variables into binary vectors where each category becomes a binary feature.
* Label Encoding: Assign a unique integer to each category. This is suitable for ordinal categorical data.

**INPUT :**

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

**OUTPUT :**

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

**INPUT :**

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

**OUTPUT :**

<Axes: xlabel='Price'>



**INPUT :**

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

**OUTPUT :**

<seaborn.axisgrid.JointGrid at 0x7dbe246100a0>



**INPUT :**

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

**OUTPUT:**

<seaborn.axisgrid.JointGrid at 0x7dbe1333c250>



**INPUT :**

plt.figure(figsize=(12,8))

sns.pairplot(dataset)

**OUTPUT :**

<seaborn.axisgrid.PairGrid at 0x7dbe1333c340>

<Figure size 1200x800 with 0 Axes>

**INPUT :**

dataset.hist(figsize=(10,8)

**OUTPUT :**

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)



VISUAL CORRELATION

**INPUT :**

dataset.corr(numeric\_only=True)

**OUTPUT :**

|  | **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 |

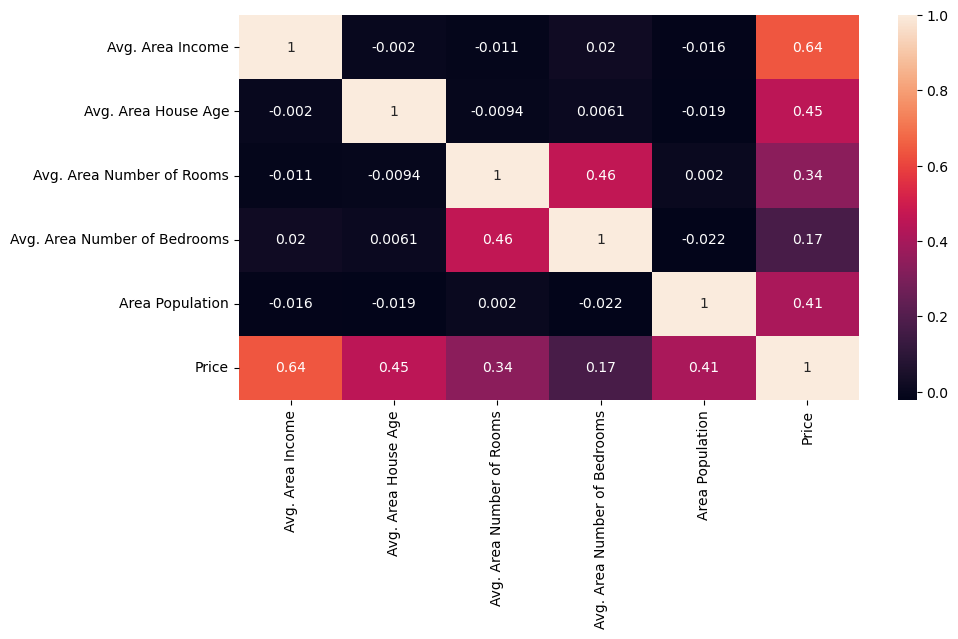
**INPUT :**

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

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

**OUTPUT :**

<Axes: >



# 

**INPUT :**

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**

**INPUT :**

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

**INPUT :**

Y\_train.head()

**OUTPUT :**

3413 1.305210e+06

1610 1.400961e+06

3459 1.048640e+06

4293 1.231157e+06

1039 1.391233e+06

Name: Price, dtype: float64

**INPUT :**

Y\_train.shape

**OUTPUT :**

(4000,)

**INPUT :**

Y\_test.head()

**OUTPUT :**

1718 1.251689e+06

2511 8.730483e+05

345 1.696978e+06

2521 1.063964e+06

54 9.487883e+05

Name: Price, dtype: float64

**INPUT :**

Y\_test.shape

**OUTPUT :**

(1000,)

# **Standardizing the data:**

**INPUT :**

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

**INPUT :**

model\_lr=LinearRegression()

**INPUT :**

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

**OUTPUT :**

LinearRegression

LinearRegression()

## Predicting Prices

**3. Feature Engineering:**

* **Creating New Features:**
* Generate new features that may be more informative such as the one of the house (current year minus year built).
* Calculate ratios or proportions between features like the price per square foot.
* **Feature Selection:**
* Utilize techniques like correlation analysis to identify relationships between features and the target variable.
* Employ feature importance scores from tree-based models like Random Forests or Gradient Boosting to select the most relevant features.
* **Model Selection:**
* Choose appropriate algorithms for regression tasks:
* Linear Regression: Simple and interpretable but assumes a linear relationship between features and target.
* Decision Threes and Random Forests: Non-linear models that can capture complex relationships in the data.
* Gradient Boosting: Ensemble method that combines multiple weak learners to create a strong predictive model.
* Support Vector Machines (SVM): Effective for high dimensional data.
* Neural Networks: Deep learning models that can capture intricate patterns but may require more data and computational resources.

**INPUT :**

Prediction1=model\_lr.predict(X\_test\_scal)

## Evaluation of Predicted Data:

**INPUT :**

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')

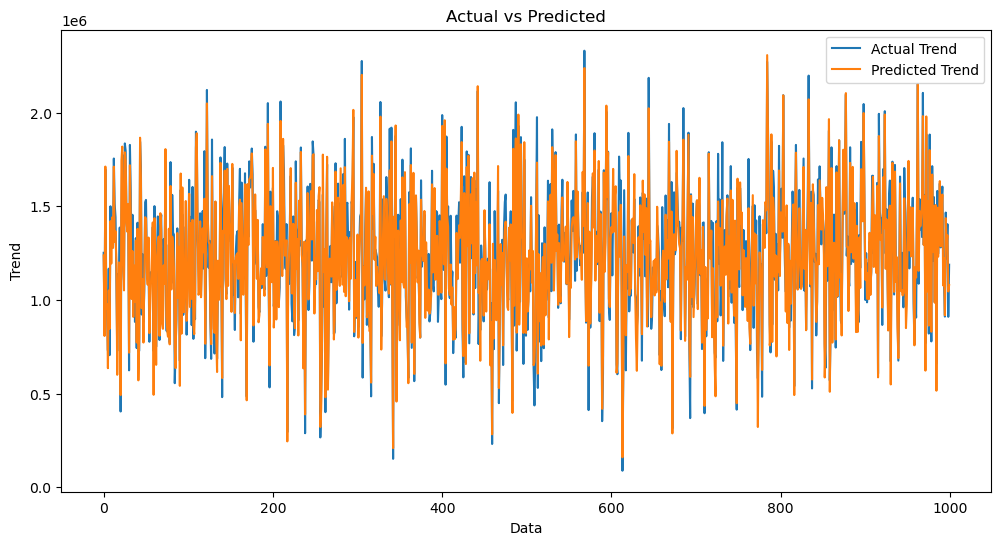
plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

**OUTPUT :**

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



**INPUT :**

sns.histplot((Y\_test-Prediction1),bins=50)

**OUTPUT :**

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



**INPUT :**

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 :

**INPUT :**

model\_svr=SVR()

**INPUT :**

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

**OUTPUT :**

SVR

SVR()

## Predicting Prices

**INPUT :**

Prediction2=model\_svr.predict(X\_test\_scal)

## Evaluation of Predicted Data

**INPUT :**

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')

**OUTPUT :**

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



**INPUT :**

sns.histplot((Y\_test-Prediction2),bins=50)

**OUTPUT :**

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



**INPUT :**

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

**INPUT :**

model\_lar=Lasso(alpha=1)

**INPUT :**

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

**OUTPUT :**

Lasso

Lasso(alpha=1)

## Predicting Prices

**INPUT :**

Prediction3=model\_lar.predict(X\_test\_scal)

## Evaluation of Predicted Data

**INPUT :**

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')

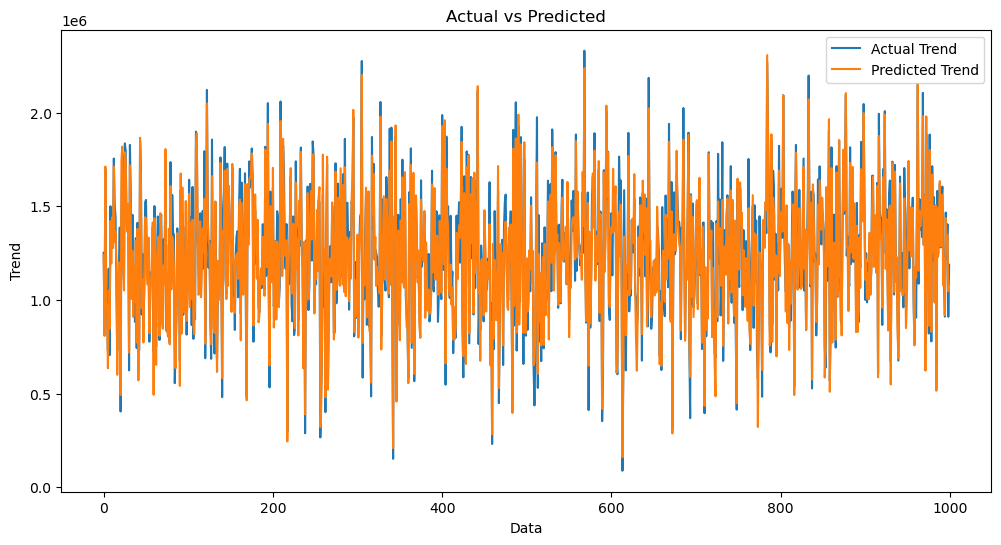
plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

**OUTPUT :**

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



**INPUT :**

sns.histplot((Y\_test-Prediction3),bins=50)

**OUTPUT :**

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



**INPUT :**

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:

**INPUT :**

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

**INPUT :**

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

**OUTPUT :**

RandomForestRegressor

RandomForestRegressor(n\_estimators=50)

## Predicting Prices

**INPUT :**

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

## Evaluation of Predicted Data

**INPUT :**

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')

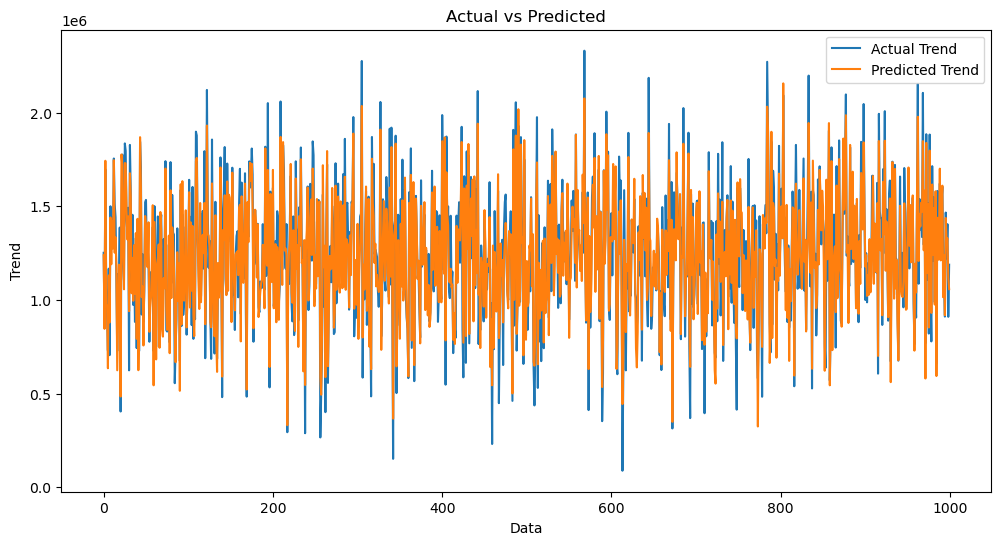
plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

**OUTPUT :**

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

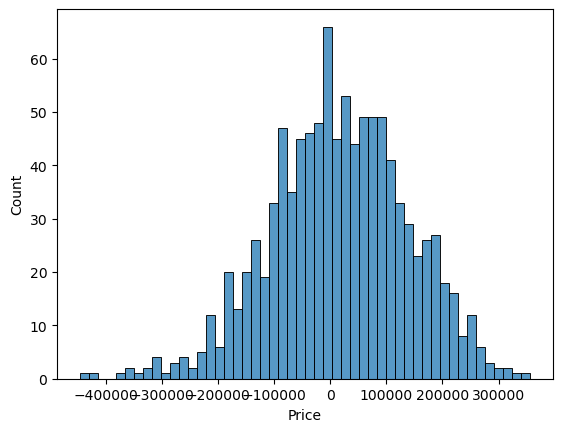


**INPUT :**

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

**OUTPUT :**

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



**INPUT :**

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 – XgboostRegressor :

**INPUT :**

model\_xg=xg.XGBRegressor()

**INPUT :**

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

**OUTPUT :**

XGBRegressor

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, ...)

## Predicting Prices

## Model 3 - Lasso Regression

1. **Hyper parameter Tuning:**
   * Experiment with different hyper parameter settings to optimize model performance. You can use techniques like grid search or random search to find the best combination of hyper parameters.

**INPUT :**

model\_lar=Lasso(alpha=1)

**INPUT :**

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

**OUTPUT :**

Lasso

Lasso(alpha=1)

## Predicting Prices

**INPUT :**

Prediction3=model\_lar.predict(X\_test\_scal)

## Evaluation of Predicted Data

**INPUT :**

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')

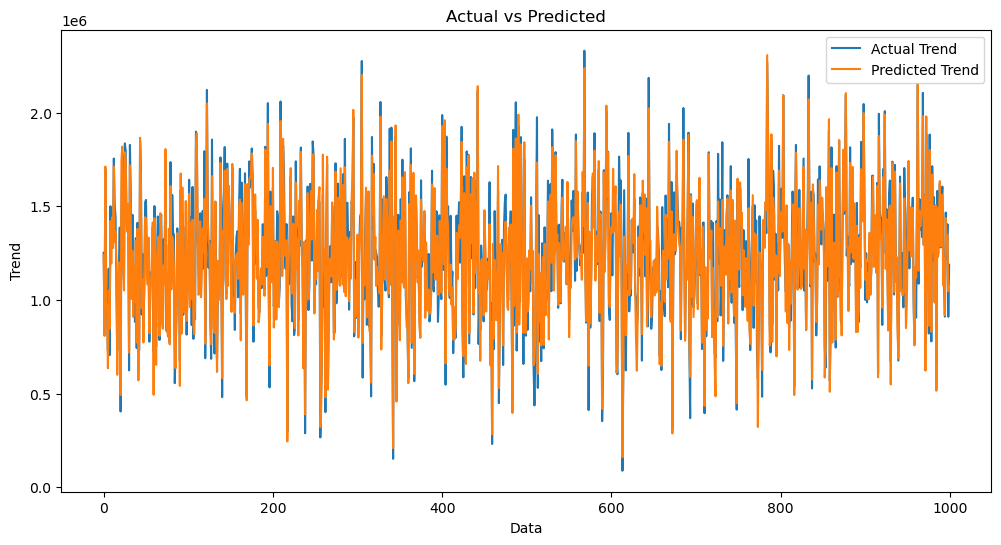
plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

**OUTPUT :**

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



**INPUT :**

sns.histplot((Y\_test-Prediction3),bins=50)

**OUTPUT :**

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



**INPUT :**

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

**INPUT :**

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

**INPUT :**

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

**OUTPUT :**

RandomForestRegressor

RandomForestRegressor(n\_estimators=50)

## Predicting Prices

**INPUT :**

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

## Evaluation of Predicted Data

**INPUT :**

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')

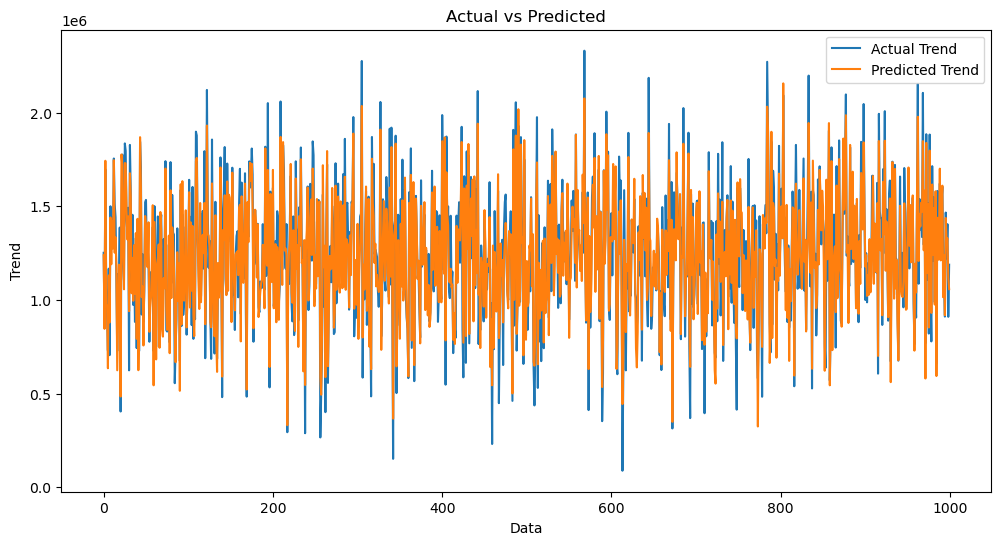
plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

**OUTPUT :**

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

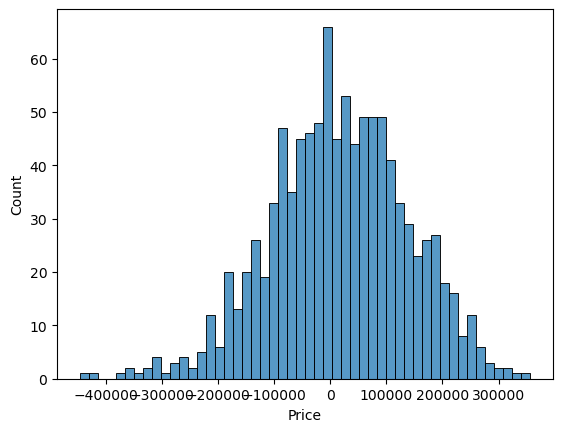


**INPUT :**

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

**OUTPUT :**

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



**INPUT :**

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 - XGboostRegressor

**INPUT :**

model\_xg=xg.XGBRegressor()

**INPUT :**

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

**OUTPUT :**

XGBRegressor

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, ...)

## Predicting Prices

**INPUT :**

Prediction5=model\_xg.predict(X\_test\_scal)

## Evaluation of Predicted Data

**INPUT :**

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')

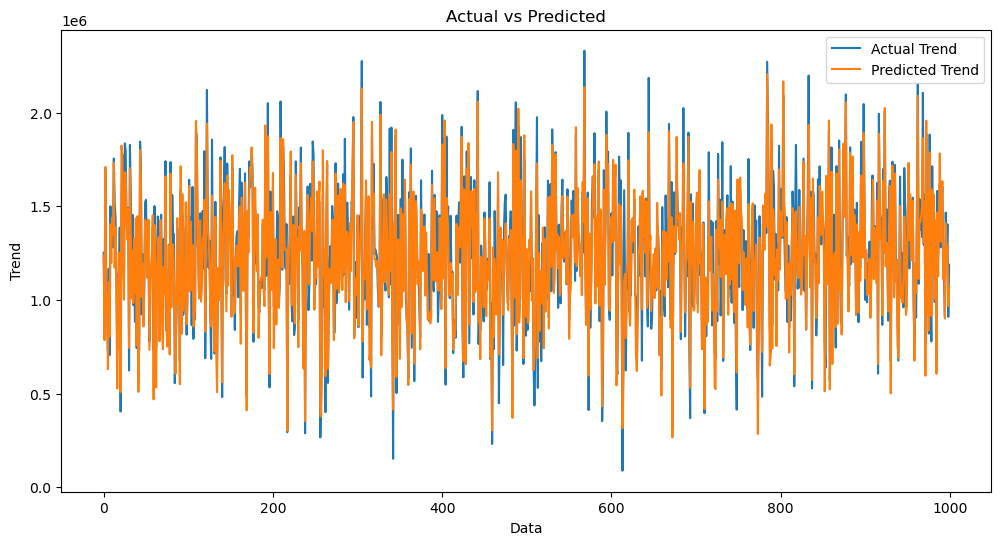
plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

**OUTPUT :**

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

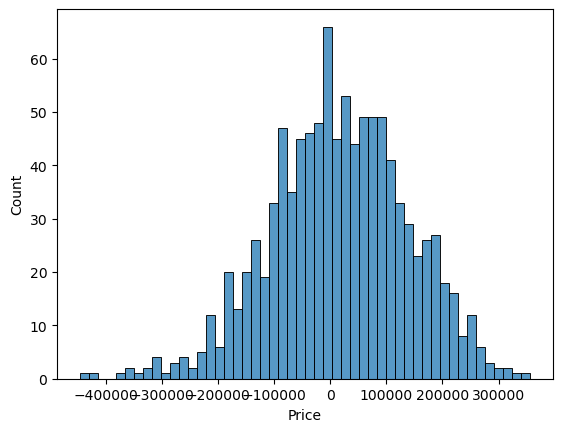


**INPUT :**

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

**OUTPUT :**

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



**INPUT :**

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

**Conclusion :**

**Thus the prediction of house prices using machine learning has been explained clearly with the techniques and related example of code with output are explained with the given dataset as the model for the prediction of house prices using machine learning.**