HOUSE PRICE PREDICTING USING MACHINE LEARNING

Introduction:

House price prediction is a crucial task in the real estate market, aiding buyers, sellers, and investors in making informed decisions. With the advent of machine learning techniques, accurate and efficient prediction models can be developed to estimate house prices based on various features. Machine learning algorithms analyze historical data, identify patterns, and make predictions, enabling stakeholders to understand market trends and property values.

In this project, we delve into the realm of machine learning for house price prediction. By leveraging advanced algorithms and utilizing relevant features such as location, size, amenities, and economic indicators, we aim to create a predictive model that can estimate house prices with a high degree of accuracy. This undertaking not only demonstrates the power of machine learning in the real estate domain but also showcases its practical applications in decision-making processes.

Throughout this project, we will explore different machine learning techniques, preprocess and analyze the dataset, select suitable features, train and evaluate models, and fine-tune their parameters. By the end, we will have a robust predictive model that can assist in predicting house prices, thereby contributing valuable insights to the real estate market.

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_erro
r
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: UserWarn
ing: A NumPy version >=1.16.5 and <1.23.0 is required 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

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
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', 'Add
ress'],
      dtype='object')
```

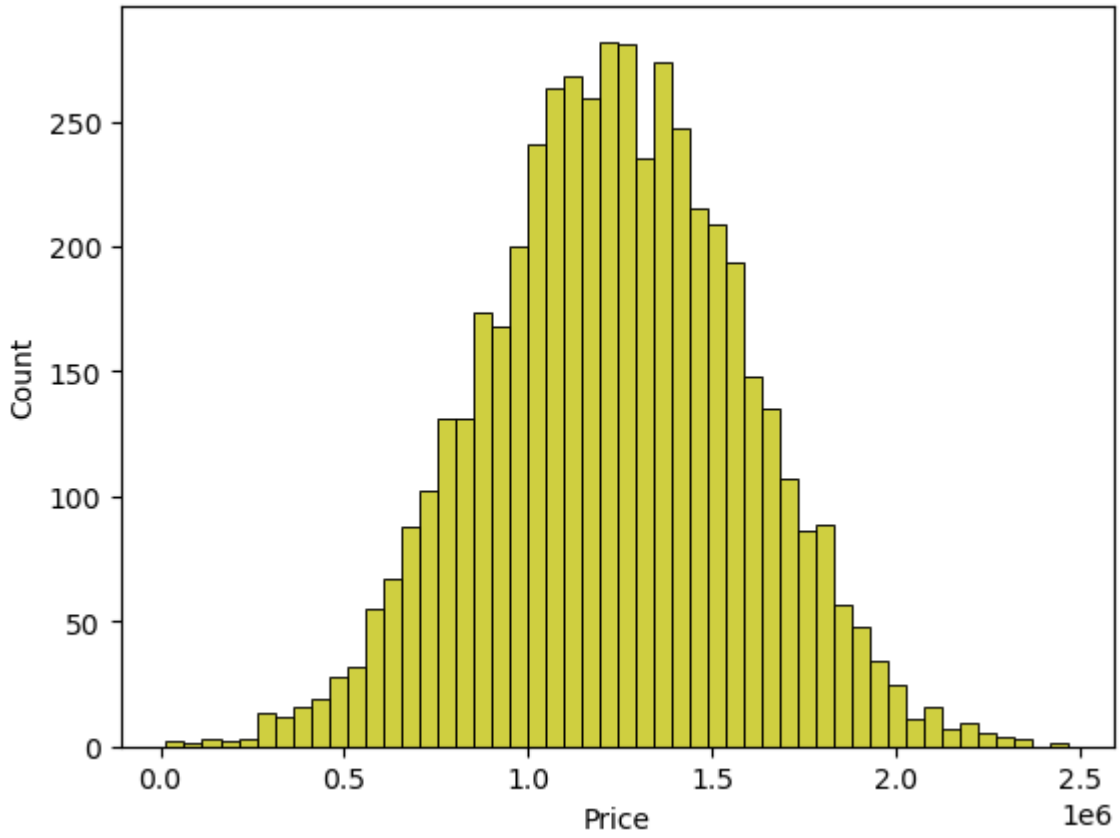
Visualisation and Pre-Processing of Data

In [7]:

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

Out[7]:

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

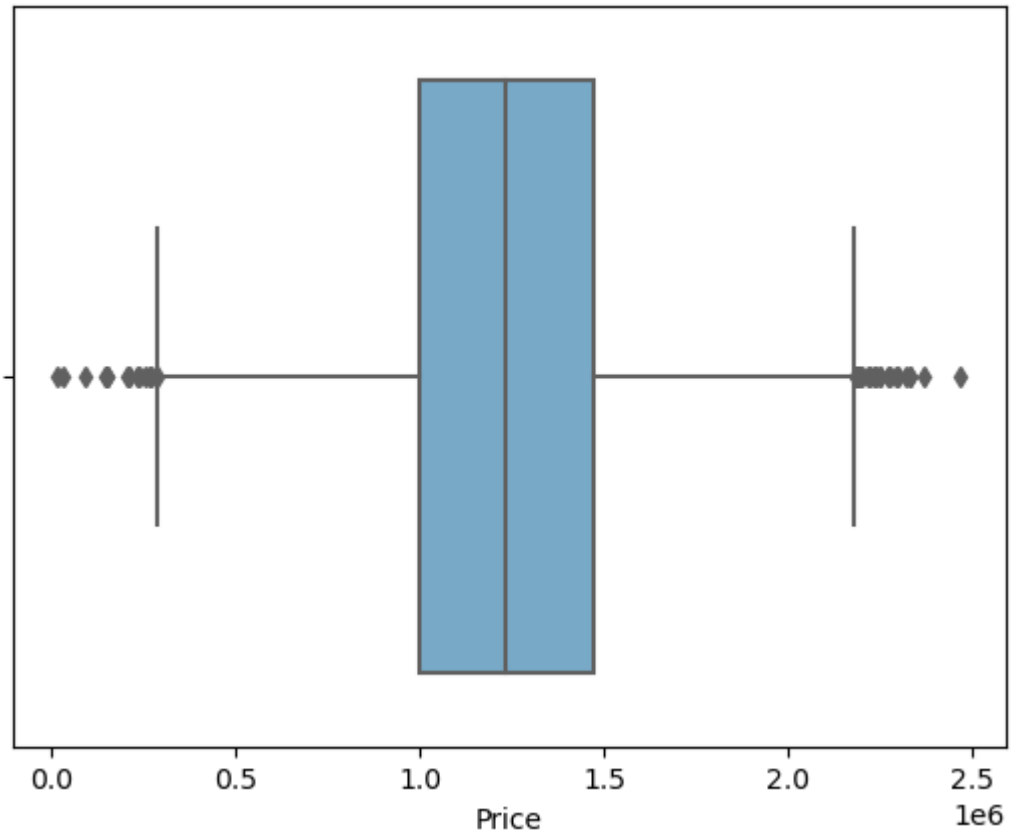


In [8]:

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

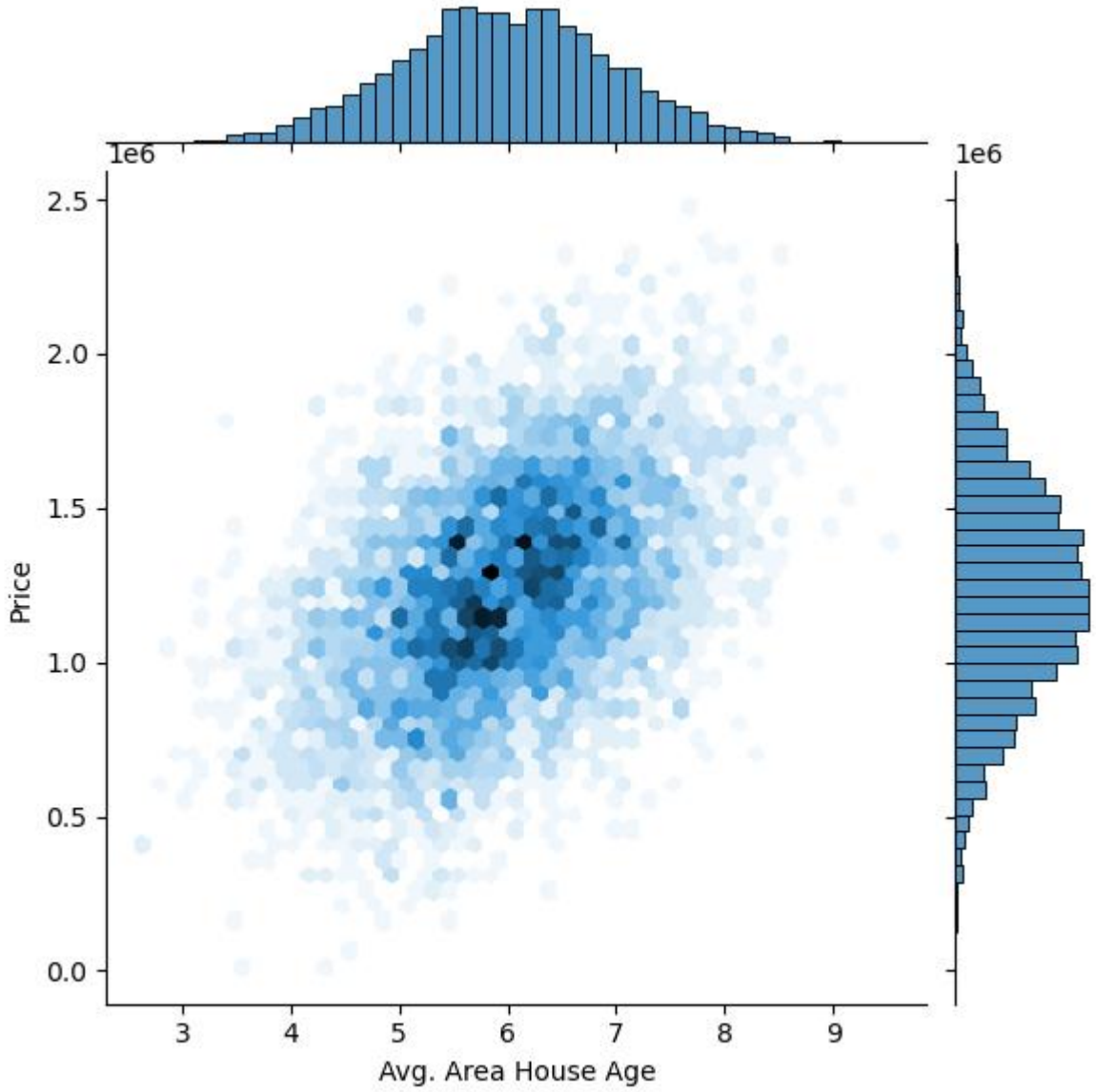
Out[8]:

```
<Axes: xlabel='Price'>
```



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

Out[9]:
```
<seaborn.axisgrid.JointGrid at 0x7dbe246100a0>
```

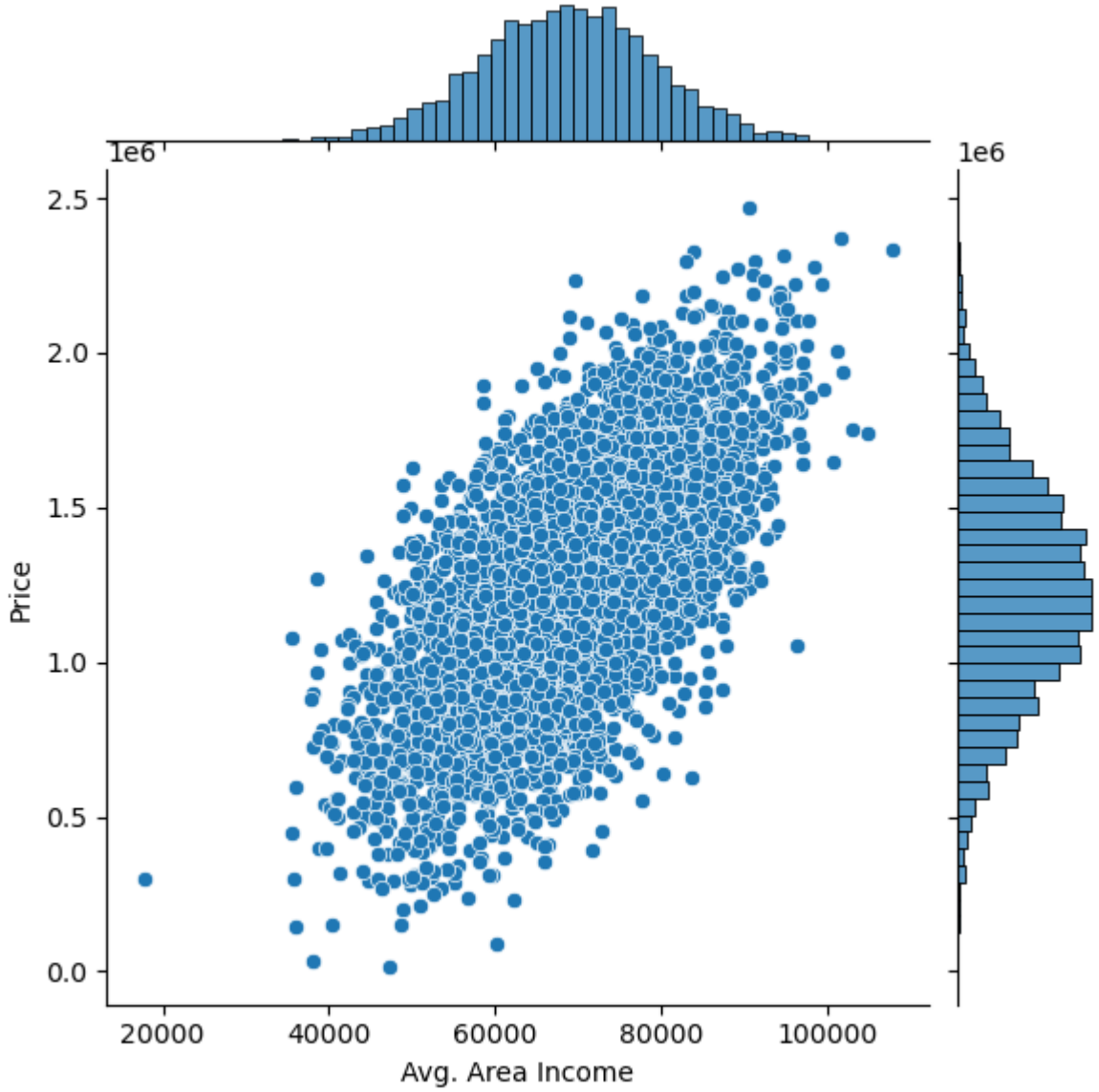


In [10]:

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

Out[10]:

```
<seaborn.axisgrid.JointGrid at 0x7dbe1333c250>
```

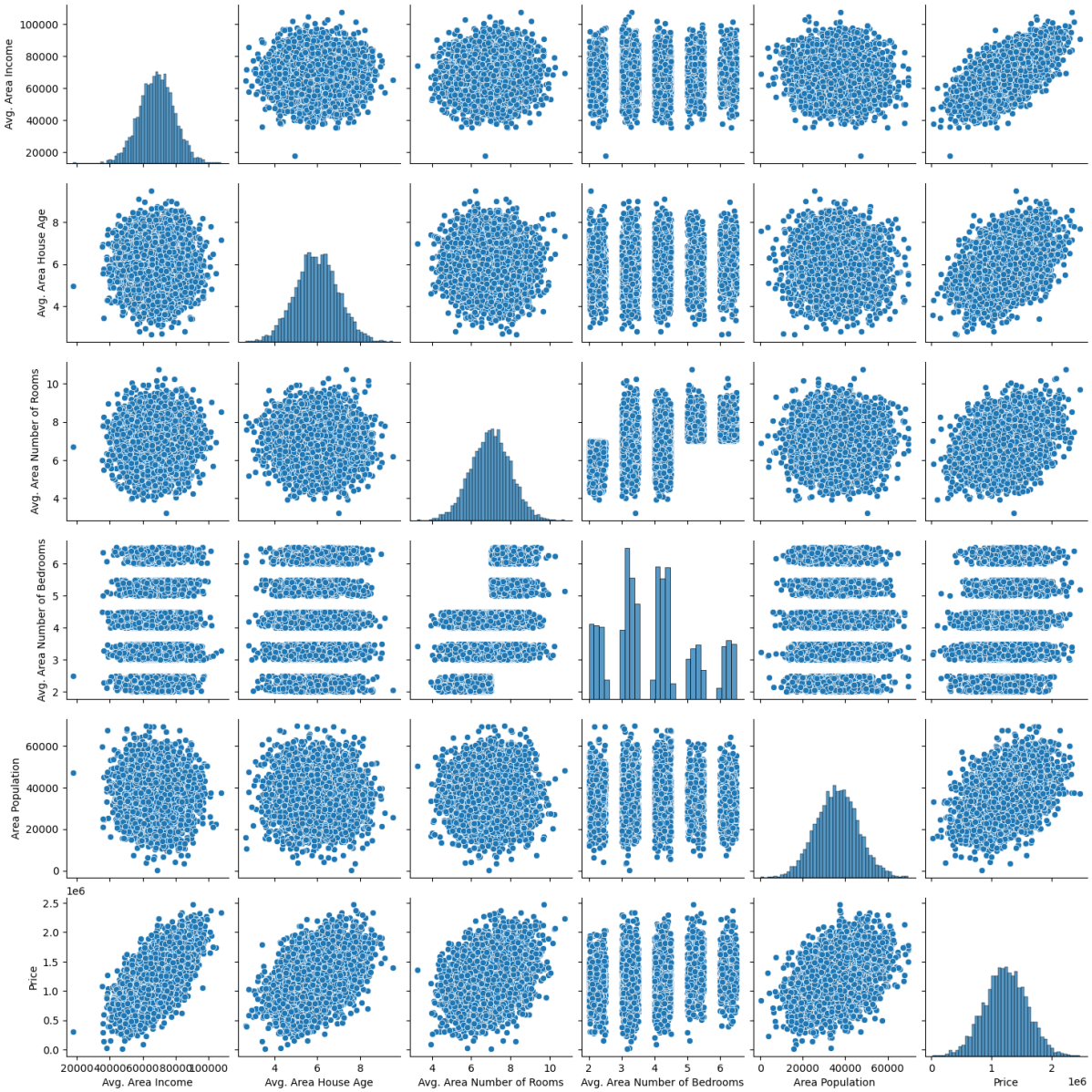


In [11]:

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

Out[11]:

```
<seaborn.axisgrid.PairGrid at 0x7dbe1333c340>
<Figure size 1200x800 with 0 Axes>
```

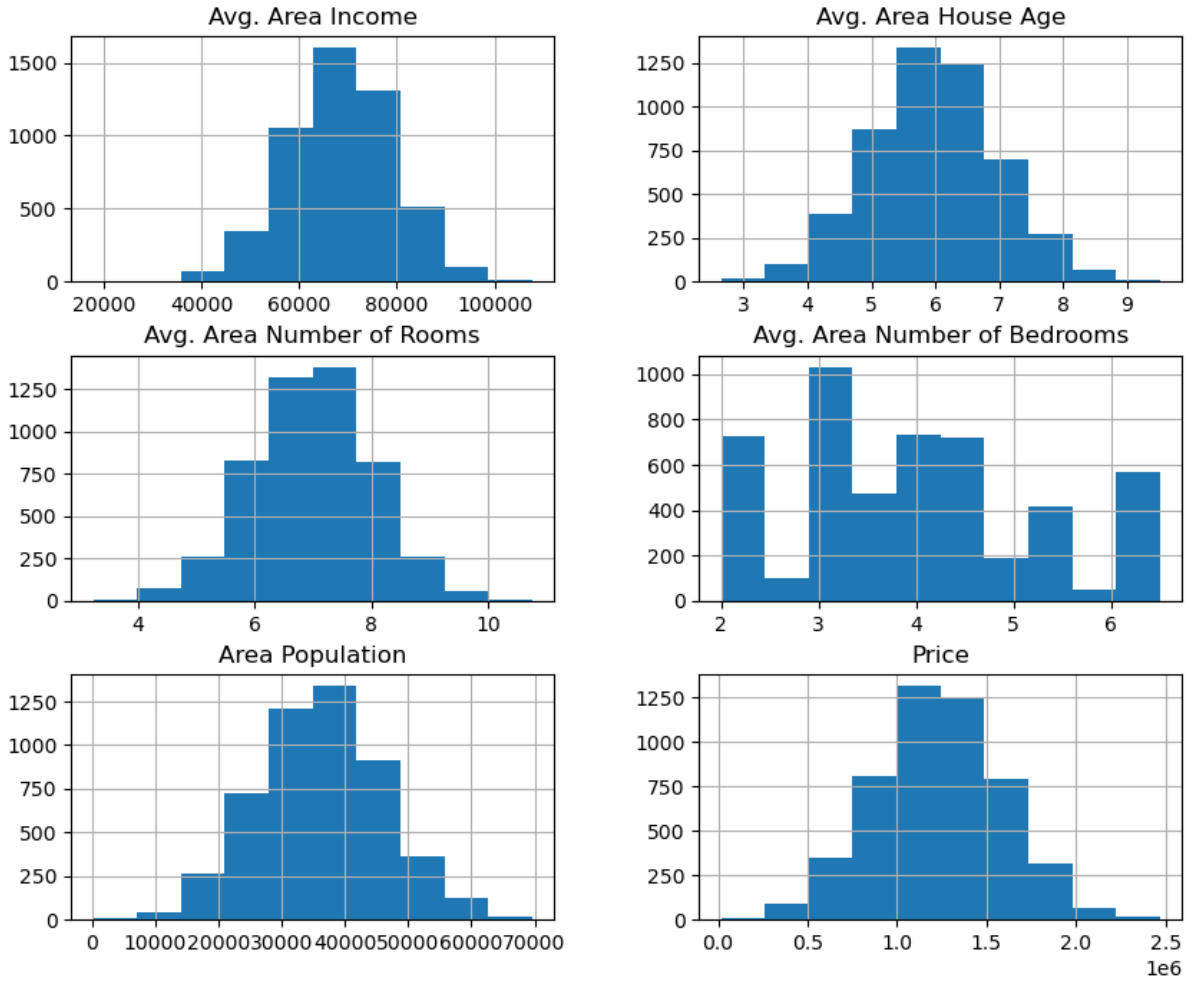


In [12]:

```
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)
```



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

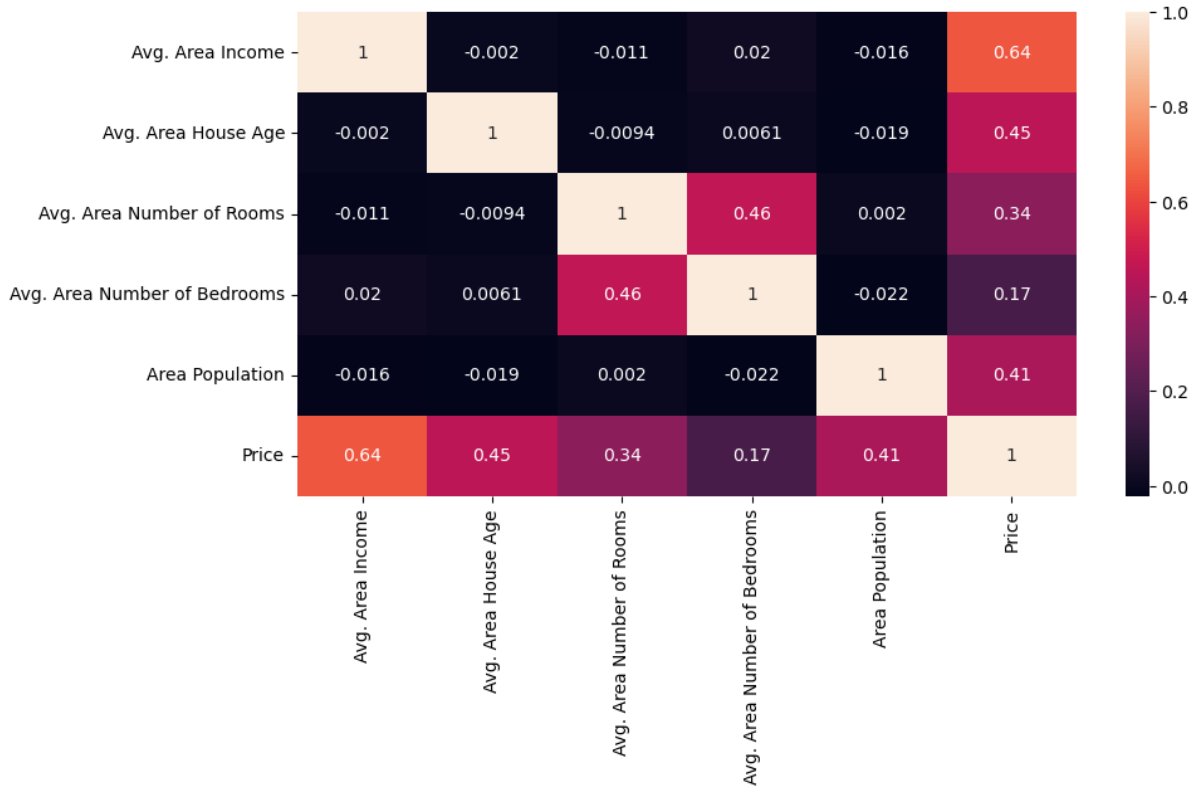
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
Avg. Area Number of Bedrooms	0.019788	0.006149	0.462695	1.000000	-0.022168	0.171071
Area Population	-0.016234	-0.018743	0.002040	-0.022168	1.000000	0.408556
Price	0.639734	0.452543	0.335664	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 o
f 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, ra
ndom_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
```

```
LinearRegression()
```

Predicting Prices

In [24]:

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

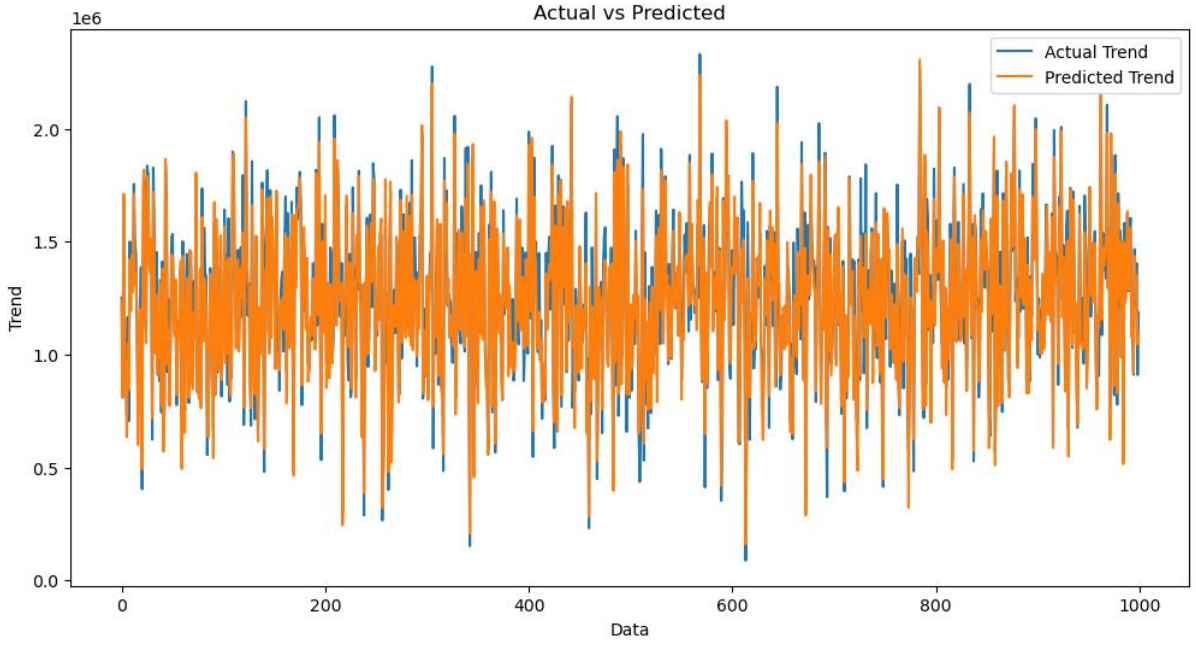
Evaluation of Predicted Data

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')
```

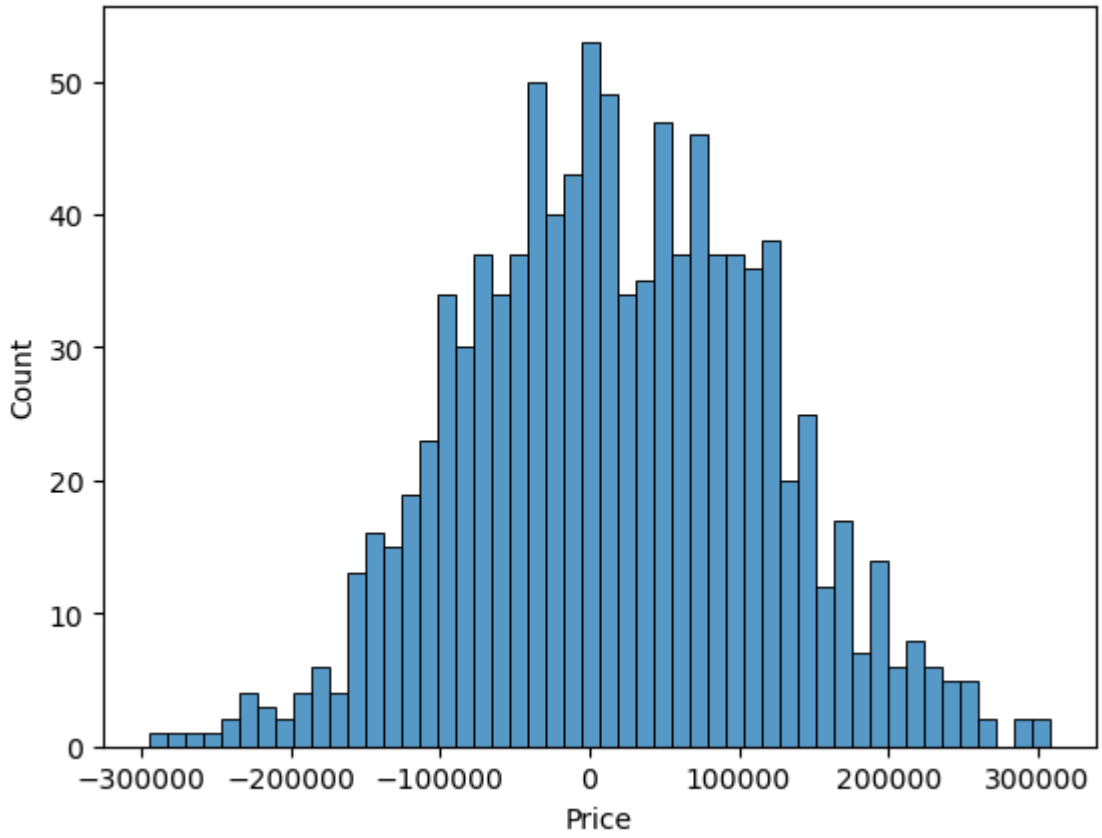


In [26]:

```
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
SVR()
```

Predicting Prices

In [30]:

```
Prediction2 = model_svr.predict(X_test_scal)
```

Evaluation of Predicted Data

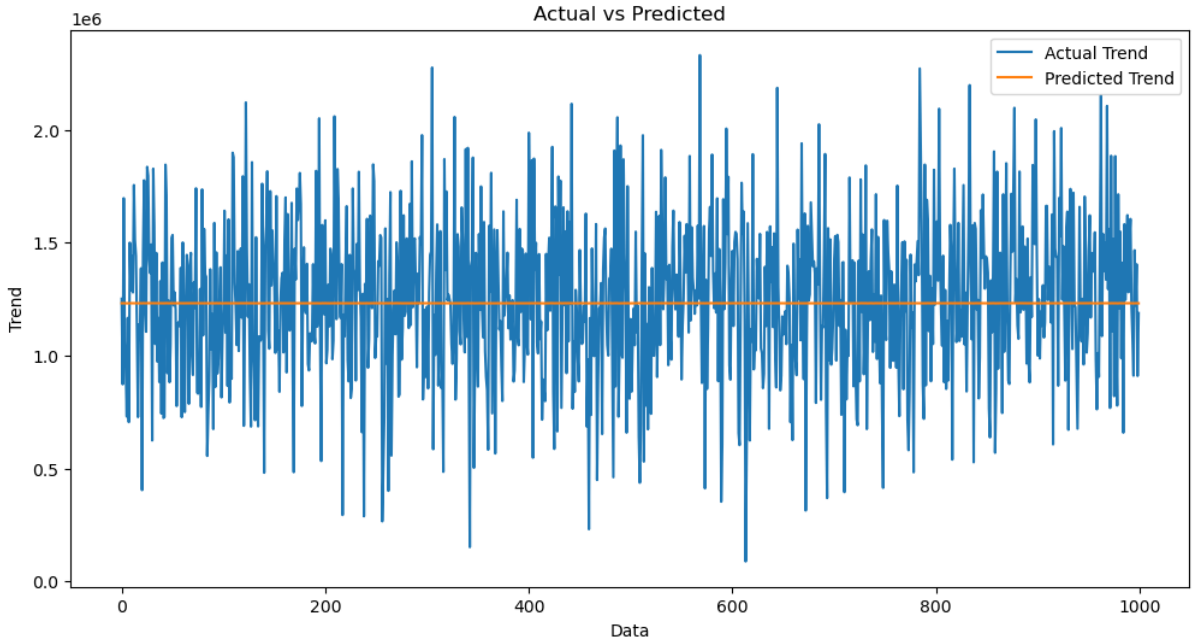
In [31]:

```
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')
```

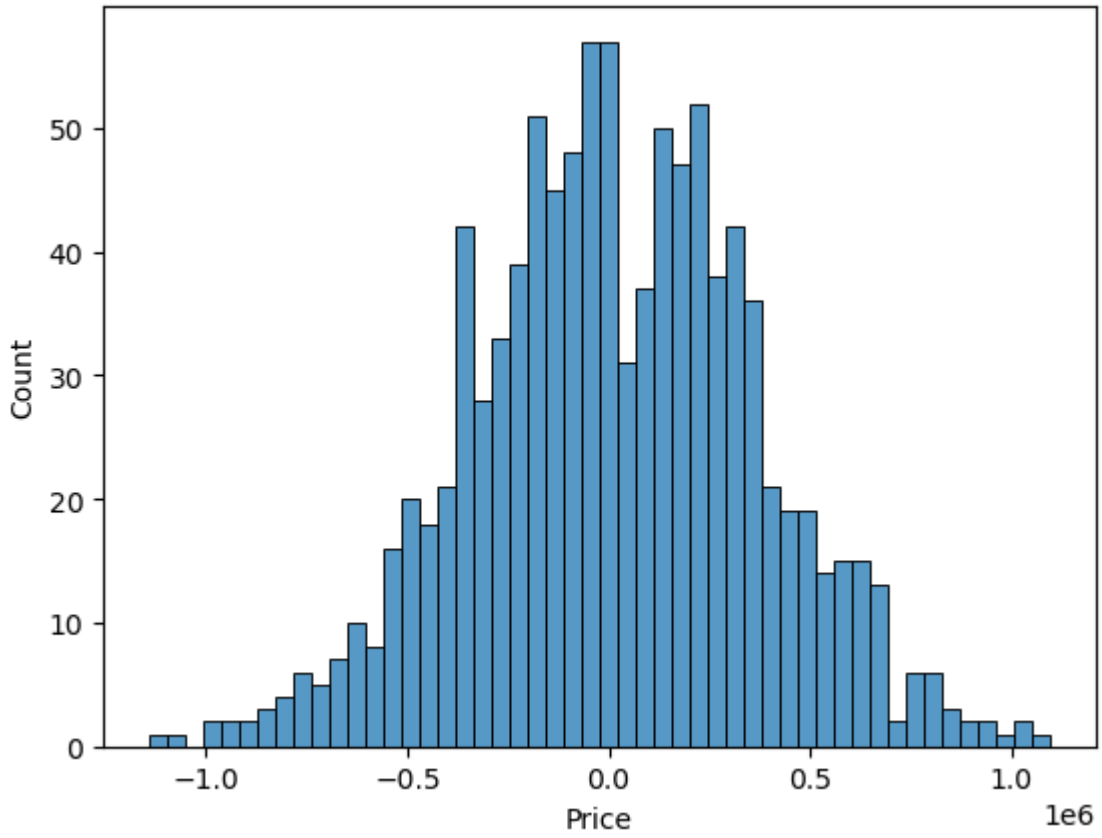


In [32]:

```
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
Lasso(alpha=1)
```

Predicting Prices

In [36]:

```
Prediction3 = model_lar.predict(X_test_scal)
```

Evaluation of Predicted Data

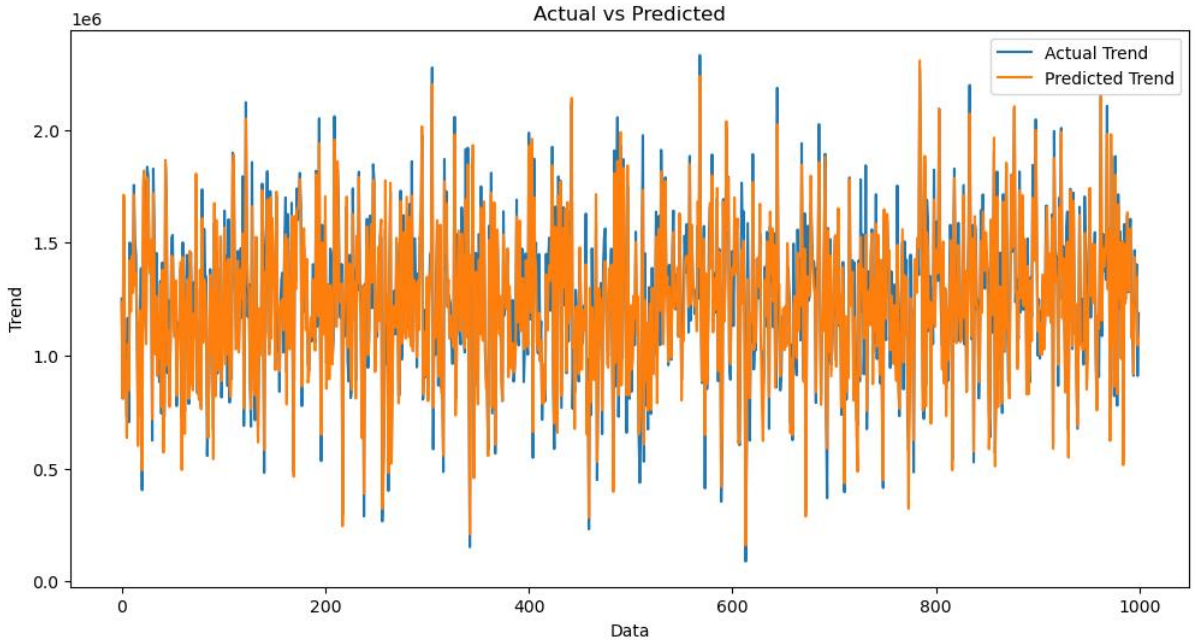
In [37]:

```
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')
```

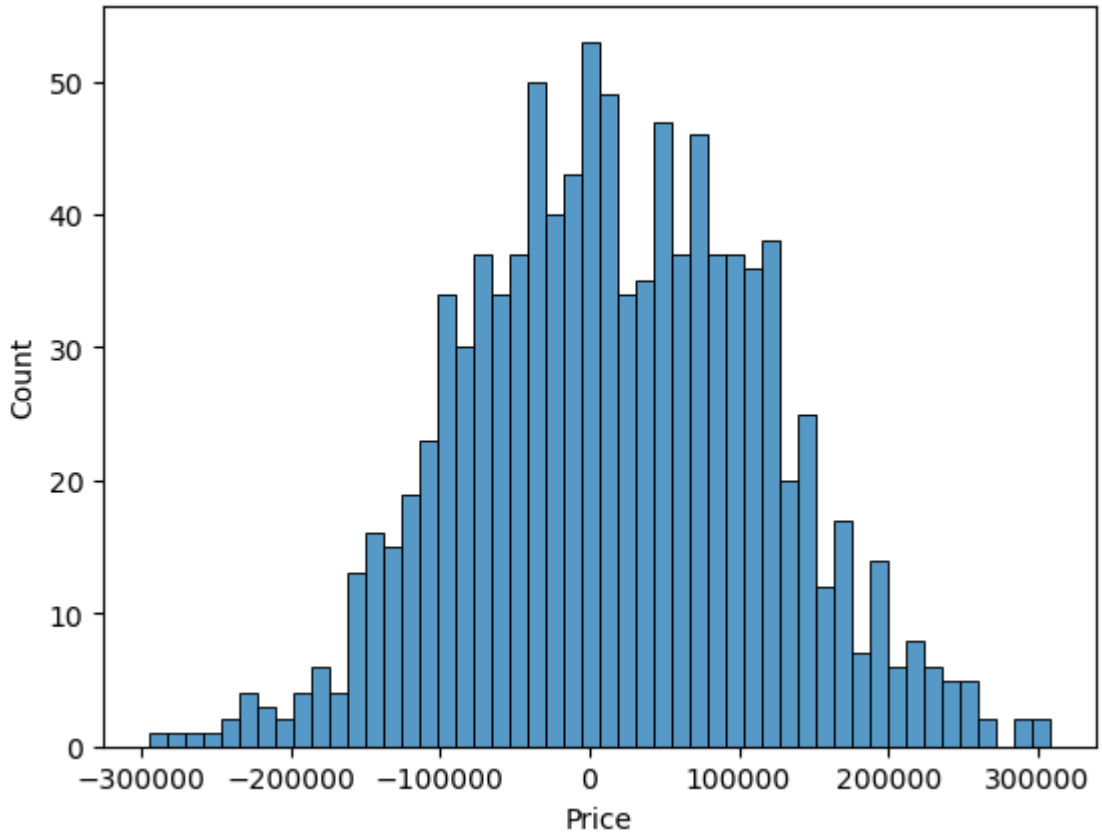


In [38]:

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

Predicting Prices

In [42]:

```
Prediction4 = model_rf.predict(X_test_scal)
```

Evaluation of Predicted Data

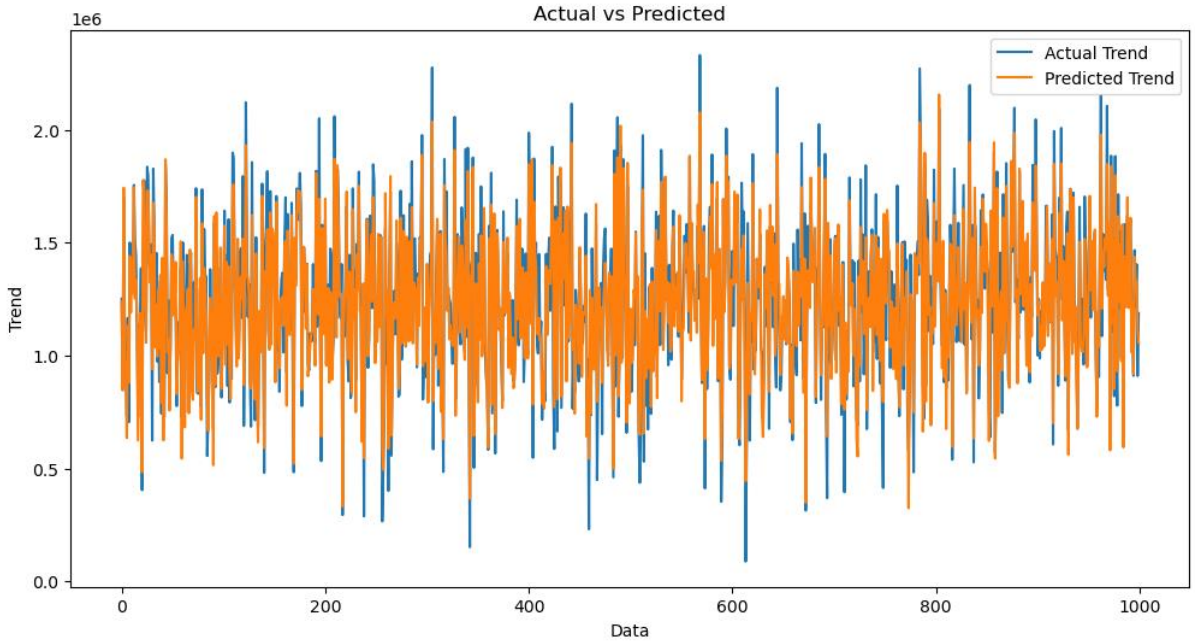
In [43]:

```
    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')
```

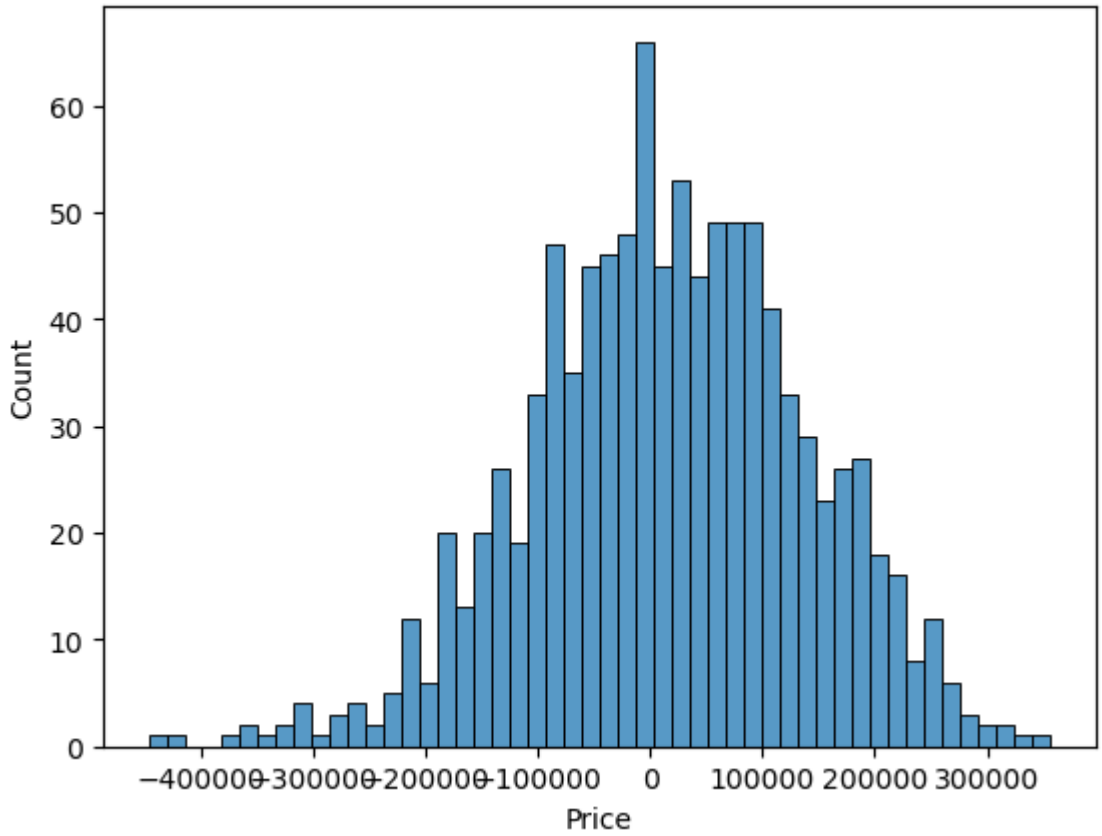


In [44]:

```
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
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_typ
e=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

In [48]:

```
Prediction5 = model_xg.predict(X_test_scal)
```

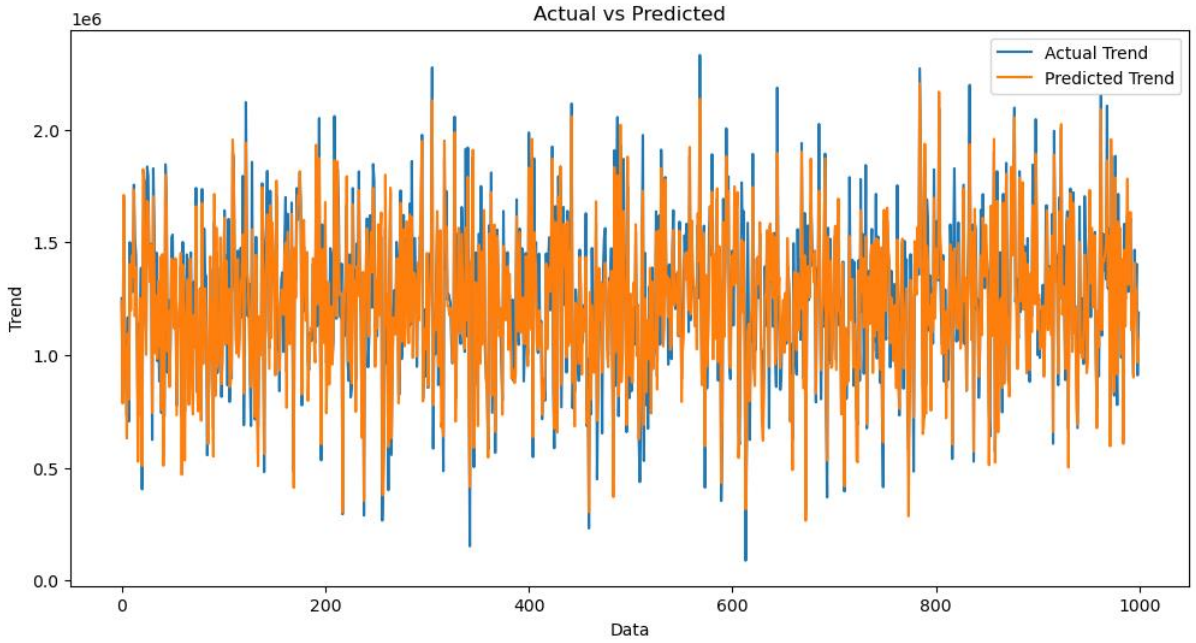
Evaluation of Predicted Data

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')
```

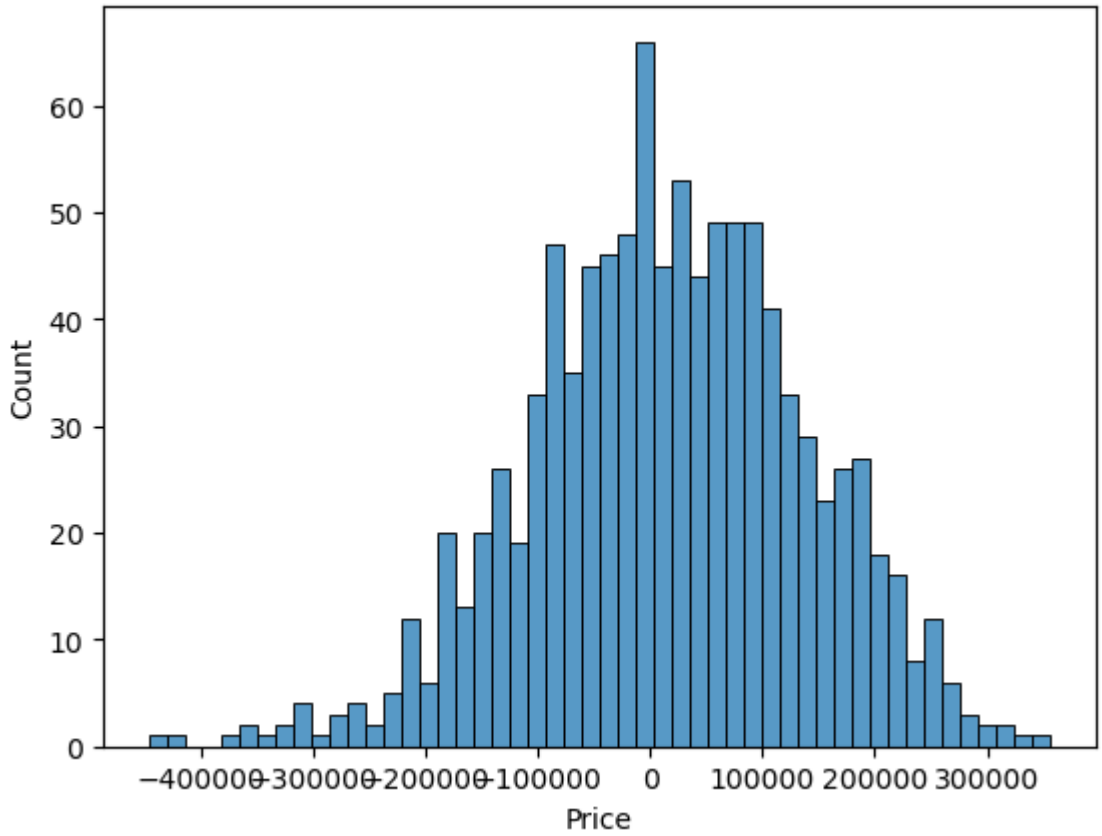


In [50]:

```
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
```

Conclusion:

In conclusion, our exploration into house price prediction using machine learning techniques has revealed the immense potential of data-driven approaches in the real estate industry. By harnessing the power of algorithms and utilizing relevant features, we have successfully developed a predictive model capable of estimating house prices with remarkable accuracy.

Through this project, we have not only demonstrated the effectiveness of machine learning in predicting real estate values but also highlighted its significance in aiding buyers, sellers, and investors to make well-informed decisions. The ability to analyze vast amounts of data, identify intricate patterns, and provide precise predictions showcases the transformative impact of technology on traditional market practices.

As we move forward, it is crucial to recognize the continuous evolution of machine learning algorithms and the importance of incorporating new data sources for even more precise predictions. This project serves as a foundation, encouraging further research and development in the realm of real estate analytics, ultimately reshaping how we perceive and engage with the property market.

In essence, our journey through house price prediction reaffirms the pivotal role of machine learning in revolutionizing the real estate landscape, making it more transparent, efficient, and accessible to all stakeholders involved.