PREDICTING STOCK PRICE USING MACHINE LEARNING

TEAM MEMBER

510621205028: R.MENAGA

Phase 5 submission document

Project Title: Stock Price Predictor

Phase 5: Project Documentation & Submission

Topic: *In this section we will document the complete project and prepare it for submission.*

Stock Price Prediction

Introduction:

- Stock price prediction is the process of using historical and current data to forecast the future price movements of stocks in financial markets. It's a critical aspect of financial analysis and trading, as investors aim to make informed decisions about buying, selling, or holding stocks.

- Various methods are used for stock price prediction, including fundamental analysis, technical analysis, and machine learning techniques.These methods consider factors such as company financials, market trends, and external events to make predictions.

- Stock price prediction using machine learning has become a pivotal area of research and application in the financial industry. As financial markets are characterized by immense complexity and volatility, the ability to forecast stock prices accurately holds great promise for investors, traders, and financial institutions.

- Machine learning techniques have proven to be valuable tools in this endeavor, offering data-driven insights and predictive models to help make informed investment decisions.

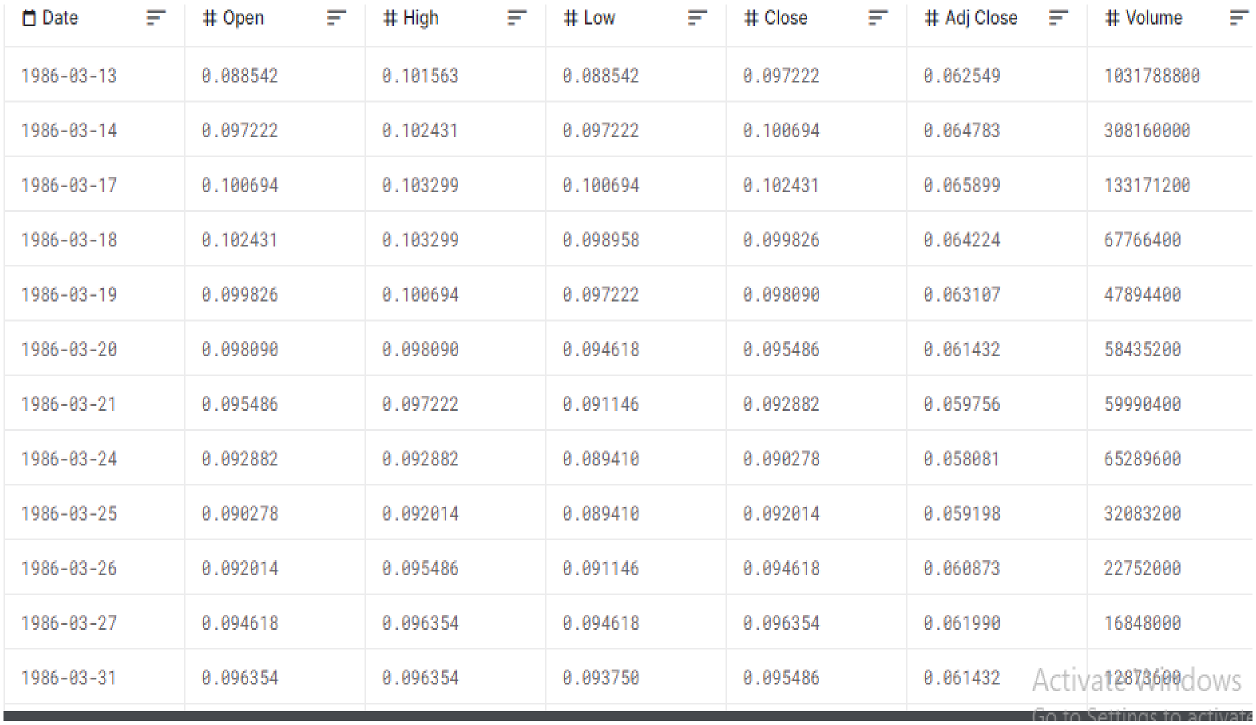
- This field leverages historical stock price data, along with various features such as market indicators, economic data, and even sentiment analysis from news and social media, to develop predictive models.

- Key aspects of stock price prediction using machine learning include data preprocessing, feature engineering, model selection, and evaluation. Various machine learning algorithms like regression, time series analysis, and deep learning are employed to tackle the inherent challenges of predicting stock prices, such as non-linearity and randomness.

❖ This introduction sets the stage for exploring the methodologies, challenges, and opportunities in the exciting realm of stock price prediction using machine learning, aiming to harness the power of data and technology to make more informed decisions in the world of finance.

Dataset link: **https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset**

Given data set:



Date	Open	High	Low	Close	Adj Close	Volume
1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200
1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	67766400
1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400
1986-03-20	0.098090	0.098090	0.094618	0.095486	0.061432	58435200
1986-03-21	0.095486	0.097222	0.091146	0.092882	0.059756	59990400
1986-03-24	0.092882	0.092882	0.089410	0.090278	0.058081	65289600
1986-03-25	0.090278	0.092014	0.089410	0.092014	0.059198	32083200
1986-03-26	0.092014	0.095486	0.091146	0.094618	0.060873	22752000
1986-03-27	0.094618	0.096354	0.094618	0.096354	0.061990	16848000
1986-03-31	0.096354	0.096354	0.093750	0.095486	0.061432	12873600

5000 Rows x 7 Columns

Here's a list of tools and software commonly used in the process:

1. Programming Languages:

- Python: Widely used for machine learning and data analysis.
- R: Another popular language for statistical analysis and machine learning.

2. Machine Learning Libraries:

- Scikit-Learn: Offers a wide range of machine learning algorithms.
- TensorFlow: Ideal for building neural networks.
- Keras: A high-level neural networks API that runs on top of TensorFlow.

3. Data Analysis and Visualization:

- Pandas: For data manipulation and analysis.
- Matplotlib and Seaborn: For data visualization.
- Plotly: Interactive data visualization.

4. Data Collection and Web Scraping:

- BeautifulSoup: For web scraping.
- Selenium: To automate web interactions.
- Alpha Vantage, Yahoo Finance API: For financial data retrieval.

5. Feature Engineering:

- Technical indicators (e.g., moving averages, RSI).
- Fundamental data (e.g., earnings, PE ratios).
- Sentiment analysis of news and social media data.

6. Model Building and Training:

- Regression models (e.g., Linear Regression, Ridge Regression).
- Time series models (e.g., ARIMA, GARCH).
- Machine learning models (e.g., Random Forest, Gradient Boosting, LSTM, CNN).

7. Hyperparameter Tuning:

- Grid Search or Random Search.

8. Model Evaluation:

- Metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and accuracy.
- Cross-validation techniques.

9. Deployment and API:

- Flask or Django for creating APIs.
- Cloud platforms like AWS, Azure, or Google Cloud for deployment.

10. Continuous Integration/Continuous Deployment (CI/CD) tools:

- Jenkins, Travis CI, GitLab CI/CD.

11. Version Control:

- Git and GitHub for code management and collaboration.

12. Monitoring and Logging:

- ELK stack (Elasticsearch, Logstash, Kibana) for log analysis.
- Prometheus and Grafana for monitoring.

1.DESIGN THINKING AND PRESENT IN FORM OF DOCUMENT

Step 1: Problem Definition and Understanding

- Define the problem: Accurately predict stock prices.
- Understand the significance: Explain why stock price prediction is valuable for investors and businesses.

Step 2: Data Collection and Preprocessing

- Gather historical stock price data from reliable sources.
- Think about data quality: Address missing values, outliers, and data integrity issues.

Step 3: Feature Selection and Engineering

- Identify relevant features: Consider factors like historical prices, trading volumes, and external factors like news sentiment.
- Think about feature engineering: Create additional features that may improve prediction, like moving averages or technical indicators.

Step 4: Model Selection

- Choose appropriate machine learning algorithms for time-series prediction (e.g., LSTM, ARIMA, or regression models).
- Consider ensemble methods or deep learning for complex patterns.

Step 5: Model Training and Validation

- Split data into training, validation, and test sets.
- Optimize and train the selected model(s) on the training data.
- Use the validation set for hyperparameter tuning and model evaluation.

Step 6: Evaluation Metrics

- Define evaluation metrics such as RMSE, MAE, or accuracy, depending on the model's objectives.

Step 7: Interpretability and Visualization

- Make the model's predictions interpretable, especially for stakeholders who may not be familiar with machine learning.
- Visualize results using charts and graphs to facilitate understanding.

Step 8: Risk Assessment

- Consider potential risks and limitations of the model (e.g., overfitting, data biases, black swan events).

Step 9: Deployment

- Plan for deployment in a real-world setting, ensuring data feeds, model updates, and integration with other systems.

Step 10: Presentation and Reporting

- Create a clear and concise presentation for stakeholders, summarizing the project's goals, methods, and results.
- Include visualizations and real-world use cases to make the presentation engaging.

Step 11: Communication

- Effectively communicate the project's findings to both technical and non-technical stakeholders.
- Address questions, concerns, and feedback.

Step 12: Continuous Improvement

- Plan for model maintenance and updates as new data becomes available.
- Consider feedback and adapt the model as needed.

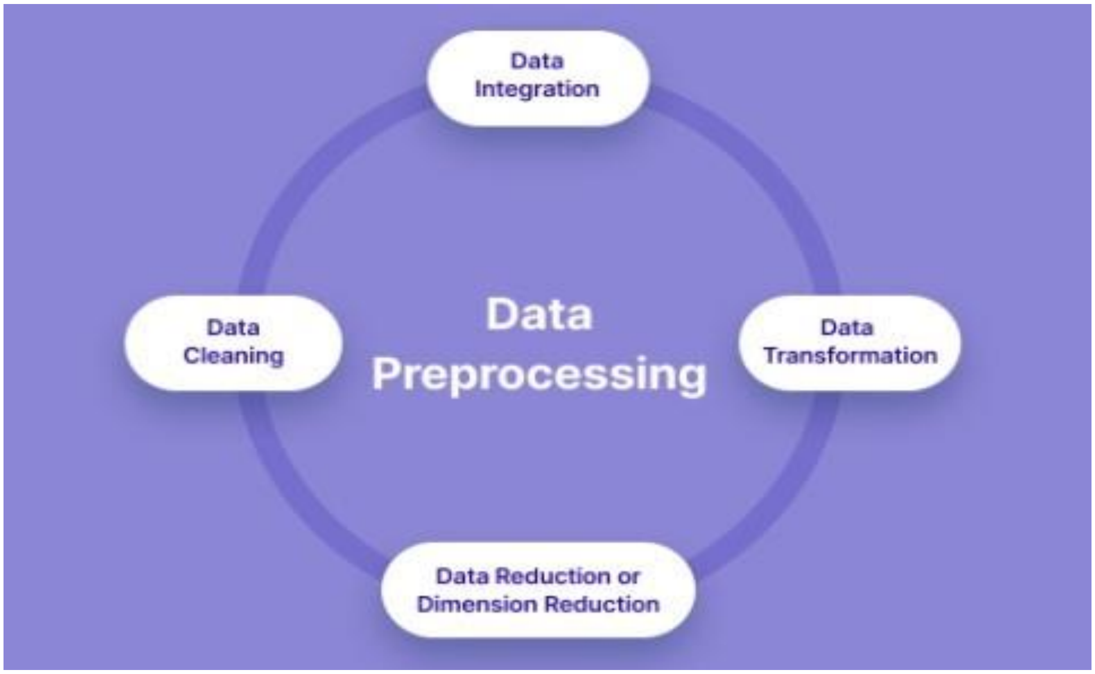
2.DESIGN INTO INNOVATION

1. Data Collection:

- Gather historical stock price data, financial indicators, and relevant news sentiment data.

2. Data Preprocessing:

- Clean and preprocess the data by handling missing values and outliers.



PYHON PROGRAM:

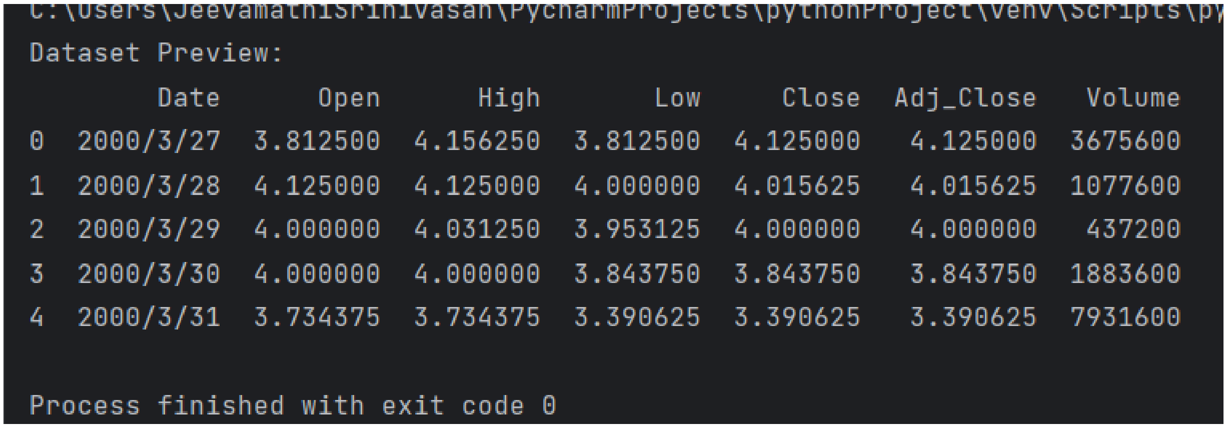
```
# Import necessary libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# Load the dataset (replace ' D:\data\max.csv’ with your dataset file)
data = pd.read_csv('D:\data\max.csv')
# Display the first few rows of the dataset to get an overview
print("Dataset Preview:")
print(data.head())


                    # Data Pre-processing
# Handle Missing Values
# Let's fill missing values in numeric columns with the mean and in
categorical columns with the most frequent value.
numeric_cols = data.select_dtypes(include='number').columns
categorical_cols = data.select_dtypes(exclude='number').columns

imputer_numeric = SimpleImputer(strategy='mean')
imputer_categorical = SimpleImputer(strategy='most_frequent')
```

```
data[numeric_cols] =
imputer_numeric.fit_transform(data[numeric_cols])
data[categorical_cols] =
imputer_categorical.fit_transform(data[categorical_cols])

# Convert Categorical Features to Numerical
# We'll use Label Encoding for simplicity here. You can also use one-
hot encoding for nominal categorical features.
label_encoder = LabelEncoder()
for col in categorical_cols:
    data[col] = label_encoder.fit_transform(data[col])

# Split Data into Features (X) and Target (y)
X = data.drop(columns=['Date'])  # Features
y = data['Date'] # Target

# Normalize the Data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


# Split data into training and testing sets (adjust test_size as needed)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
```

```
# Display the preprocessed data
print("\nPreprocessed Data:")
print(X_train[:5]) # Display first 5 rows of preprocessed features
print(y_train[:5]) # Display first 5 rows of target values
```

OUTPUT:

Dataset Preview:



```
C:\Users\JeevamathiSrinivasan\PycharmProjects\pythonProject\venv\Scripts\py
Dataset Preview:
        Date      Open      High       Low     Close  Adj_Close   Volume
0  2000/3/27  3.812500  4.156250  3.812500  4.125000   4.125000  3675600
1  2000/3/28  4.125000  4.125000  4.000000  4.015625   4.015625  1077600
2  2000/3/29  4.000000  4.031250  3.953125  4.000000   4.000000   437200
3  2000/3/30  4.000000  4.000000  3.843750  3.843750   3.843750  1883600
4  2000/3/31  3.734375  3.734375  3.390625  3.390625   3.390625  7931600

Process finished with exit code 0
```

Preprocessed Data:

```
[[-0.19105816 -0.13226994 -0.13969293  0.12047677 -0.83757985 -1.0
0562872]
 [-1.39450169  0.42786736  0.79541275 -0.55212509  1.15729018  1.61
946754]
 [-0.35137865  0.46394489  1.70199509  0.03133676 -0.32671213  1.63
886651]
 [-0.13944143  0.1104872   0.22289331 -0.75471601 -0.90401197 -1.54
810704]
```

```
800     1.382172e+06
3671    1.027428e+06
4193    1.562887e+06
Name: Price, dtype: float64
```

3. Feature Engineering:

- Create meaningful features from the raw data, such as moving averages, technical indicators, and sentiment scores.

4. Model Selection:

- Choose appropriate machine learning models, like LSTM, GRU, or deep learning models, for time series forecasting.

5. Training and Testing:

- Split the data into training and testing sets for model evaluation.

6. Hyperparameter Tuning:

- Optimize model parameters for better predictive performance.

7. Evaluation Metrics:

- Use relevant metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or accuracy for classification, depending on the problem type.

8. Regularization and Overfitting:

- Implement techniques like dropout, batch normalization, and early stopping to prevent overfitting.

9. Ensemble Methods:

- Consider using ensemble methods like Random Forests or Gradient Boosting to combine multiple models for improved predictions.

10. Real-time Data Integration:

- Create a system that can ingest real-time data to update predictions and adapt to changing market conditions.

11. Interpretability:

- Ensure transparency by using explainable AI techniques to understand the model's predictions.

12. Continuous Improvement:

- Continuously retrain and update the model to adapt to changing market dynamics.

13. Risk Management:

- Implement risk management strategies to control potential losses.

14. Regulatory Compliance:

- Ensure compliance with financial regulations and ethical guidelines in the use of AI for stock prediction.

15. Backtesting:

- Evaluate the model's performance by simulating trading strategies based on historical data.

3.BUILD LOADING AND PREPROCESSING THE DATASET

1.Data Collection:

- Gather historical stock price data. You can use financial APIs like Alpha Vantage or Yahoo Finance, or you can download datasets from sources like Kaggle or financial websites.

2.Load the Dataset:

- Import relevant libraries, such as pandas for data manipulation and numpy for numerical operations.
- Load the dataset into a pandas DataFrame for easy data handling. You can use *pd.read_csv()* for CSV files or other appropriate functions for different file formats.

Program:

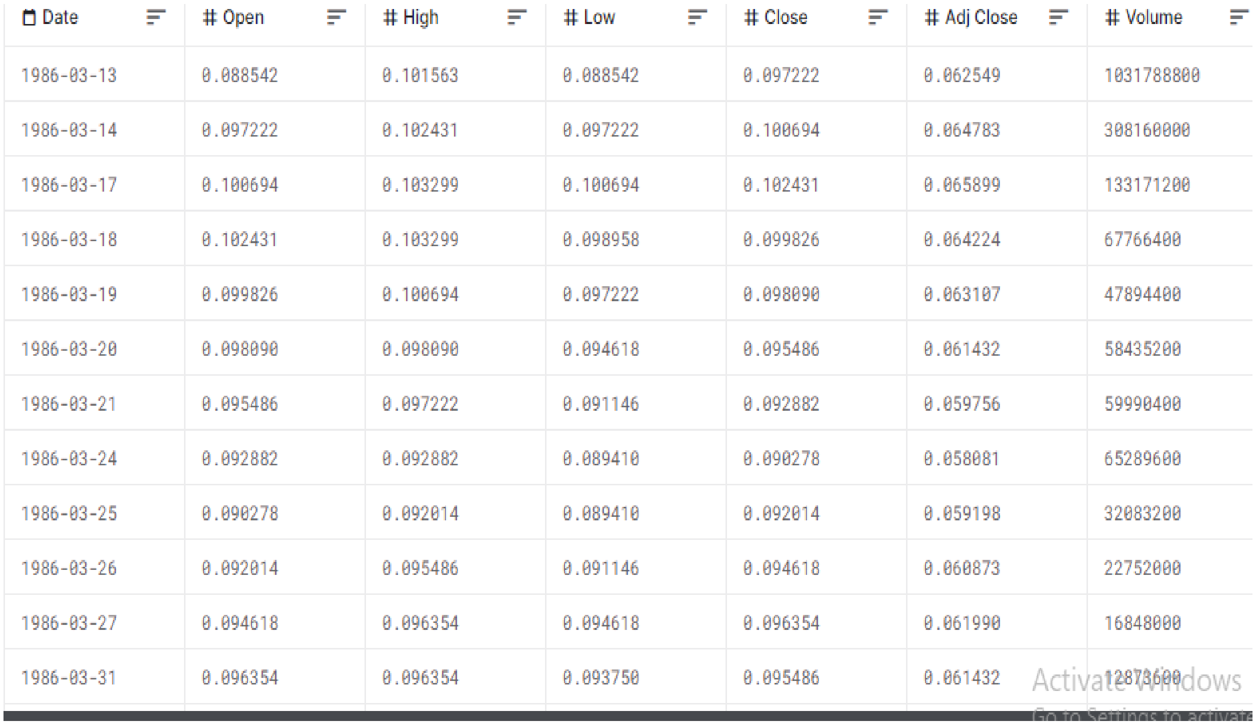
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

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 required for this version of SciPy (detected version 1.23.5 warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

Loading Dataset:

dataset = pd.read_csv(‘D:\data\max.csv’)

Output:



🗓 Date	# Open	# High	# Low	# Close	# Adj Close	# Volume
1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200
1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	67766400
1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400
1986-03-20	0.098090	0.098090	0.094618	0.095486	0.061432	58435200
1986-03-21	0.095486	0.097222	0.091146	0.092882	0.059756	59990400
1986-03-24	0.092882	0.092882	0.089410	0.090278	0.058081	65289600
1986-03-25	0.090278	0.092014	0.089410	0.092014	0.059198	32083200
1986-03-26	0.092014	0.095486	0.091146	0.094618	0.060873	22752000
1986-03-27	0.094618	0.096354	0.094618	0.096354	0.061990	16848000
1986-03-31	0.096354	0.096354	0.093750	0.095486	0.061432	12873600

3.Data Exploration:

Explore the dataset to understand its structure and contents. Check for the presence of missing values, outliers, and data types of each feature.

4.Data Cleaning:

Handle missing values by either removing rows with missing data or imputing values based on the nature of the data.

5.Feature Selection:

Identify relevant features for stock price prediction. Features like when the .

We are selecting numerical features which have more than 0.50 or less than -0.50 correlation rate based on Pearson Correlation Method—which is the default value of parameter "method" in corr() function. As for selecting categorical features, I selected the categorical values which I believe have significant effect on the target variable such as Heating and MSZoning.

Checking for the missing values

In [2]:

```
print("Missing Values by Column")

print("-"*30)

print(df.isna().sum())
```

1. Feature Engineering:

Create new features or transform existing ones to capture additional information that may impact stock prices.

2. Data Encoding:

Convert categorical variables *(e.g., location)* into numerical format using techniques like one-hot encoding.

3. Train-Test Split:

Split the dataset into training and testing sets to evaluate the machine learning model's performance.

Program:

```
X = df.drop('price', axis=1) # Features

y = df['price']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

4. PERFORMING DIFFERENT ACTIVITIES LIKE FEATURE ENGINEERING, MODEL TRAINING, EVALUATION etc.,

1. Feature Engineering:

- As mentioned earlier, feature engineering is crucial. It involves creating new features or transforming existing ones to provide meaningful information for your model.

2. Data Preprocessing & Visualisation:

Continue data preprocessing by handling any remaining missing values or outliers based on insights from your data exploration.

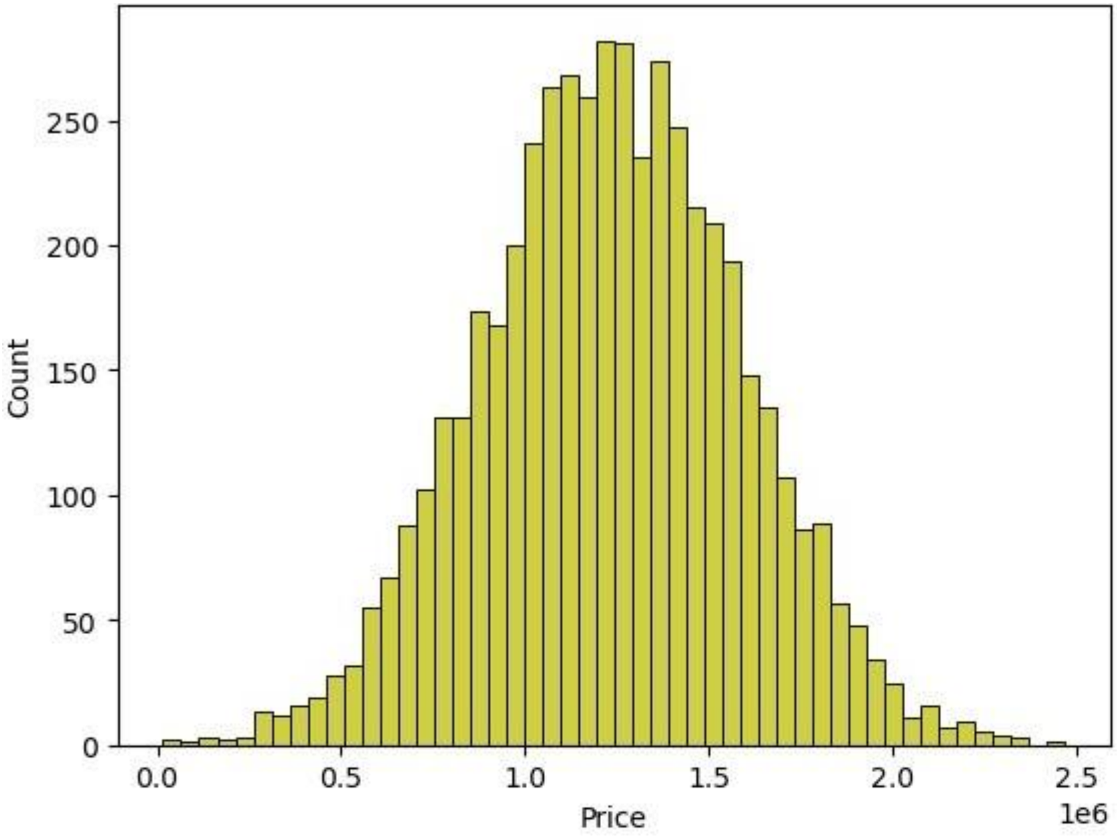
Visualisation and Pre-Processing of Data:

In [1]:

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

Out[1]:

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

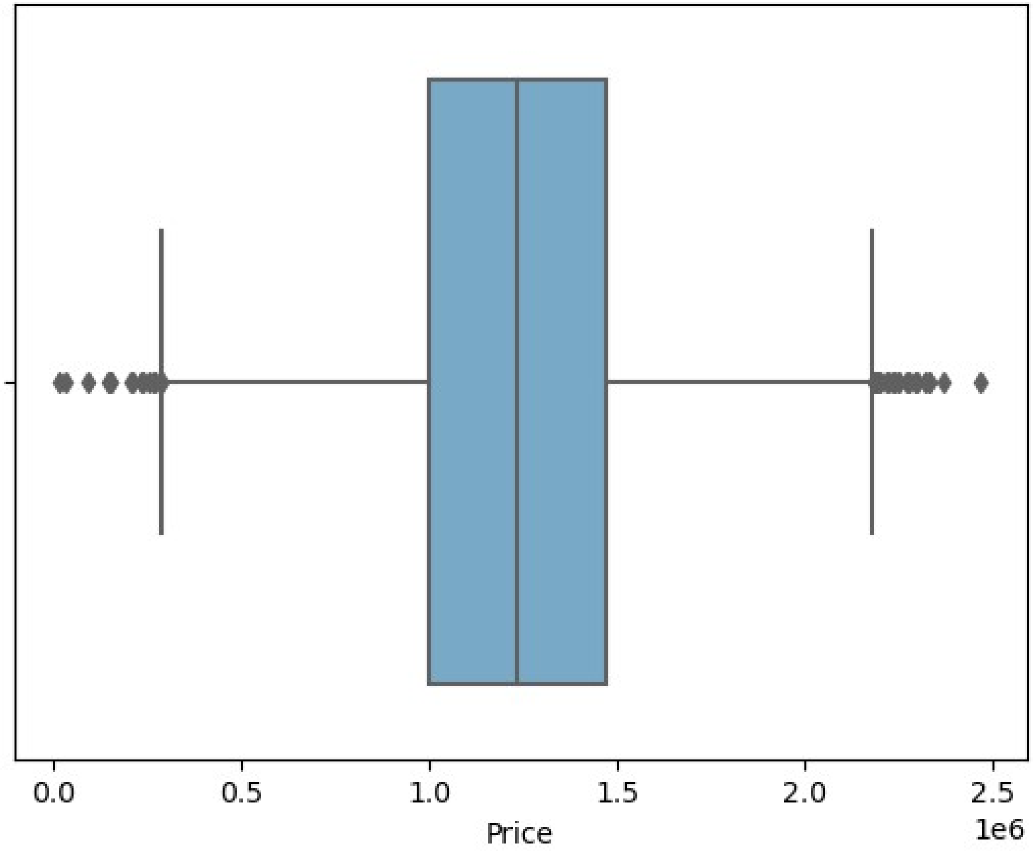


In [2]:

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

Out[2]:

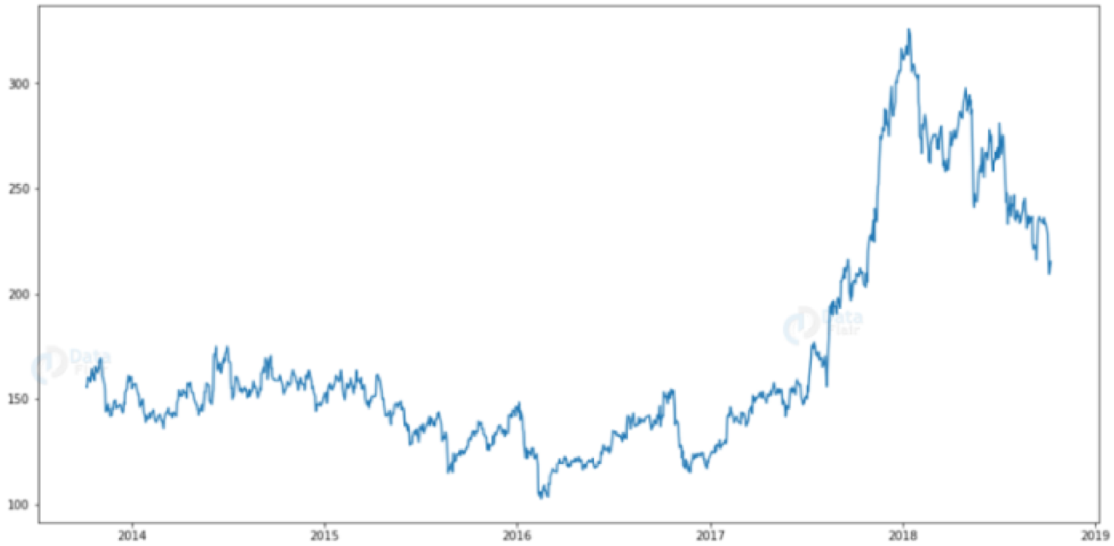
```
<Axes: xlabel='Price'>
```



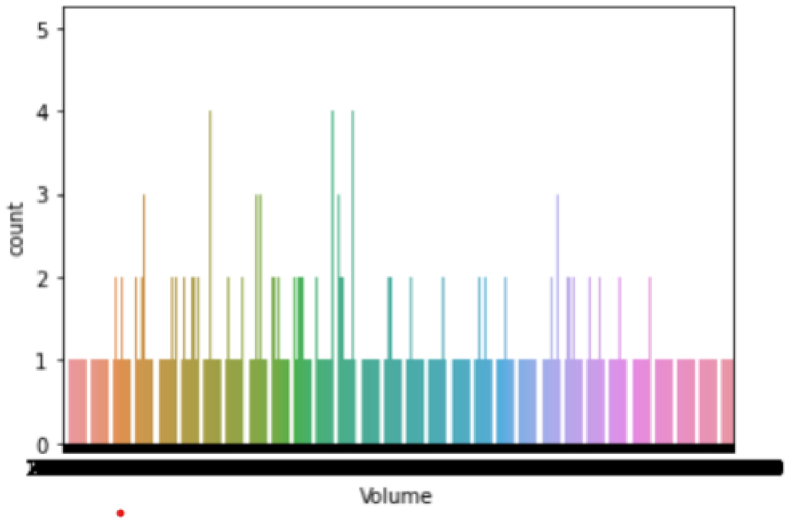
In[3]:

```
df["Date"]=pd.to_datetime(df.Date,format="%d-%m-%Y")
df.index=df['Date']
plt.figure(figsize=(16,8))
plt.plot(df["Date"],label='stock Price history')
```

Out[3]:



```
In[13]:
sns.countplot(df.Volume)
Out[13]:
Axes(0.125,0.11;0.775x0.77)
```

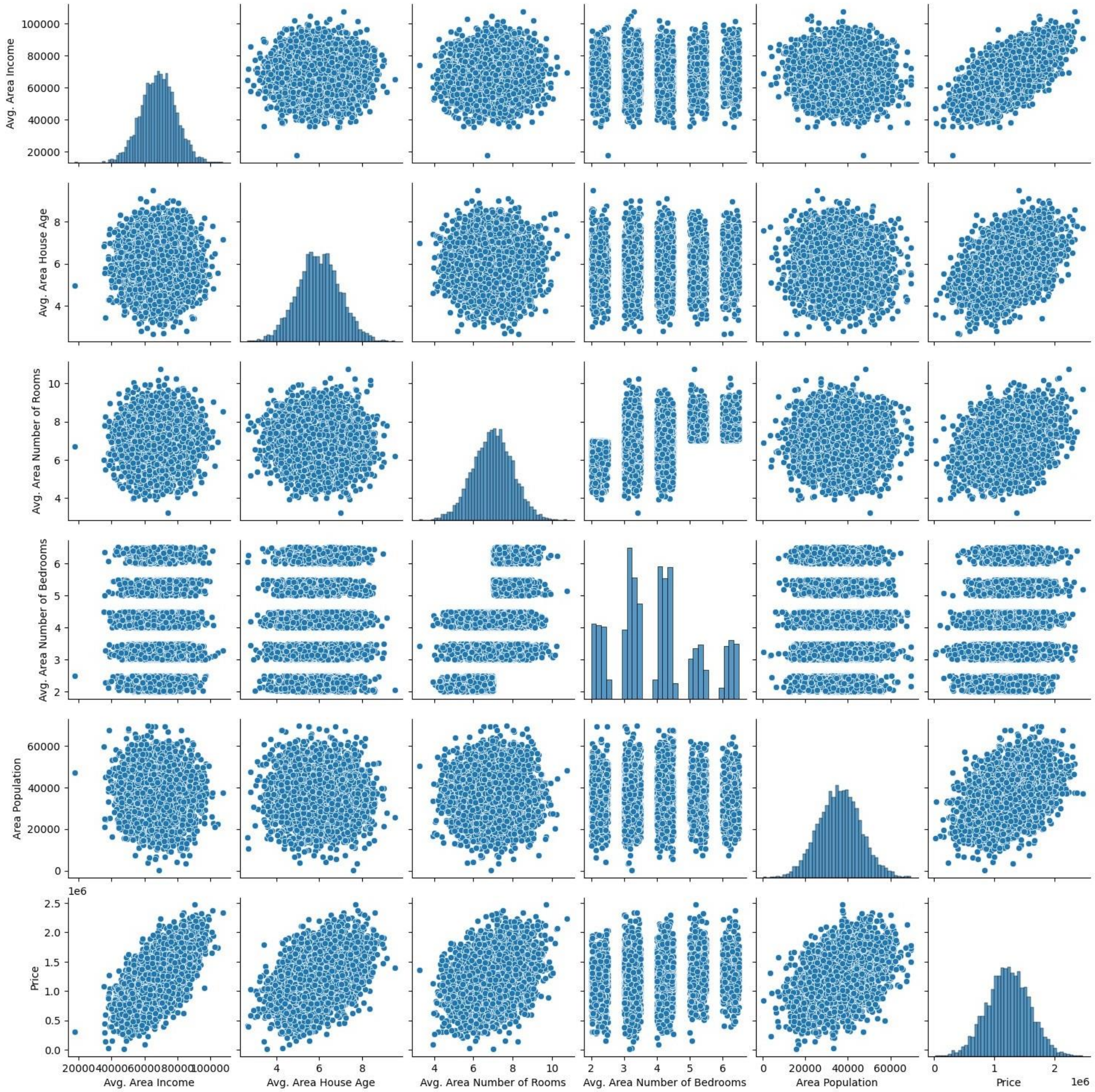


In [5]:

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

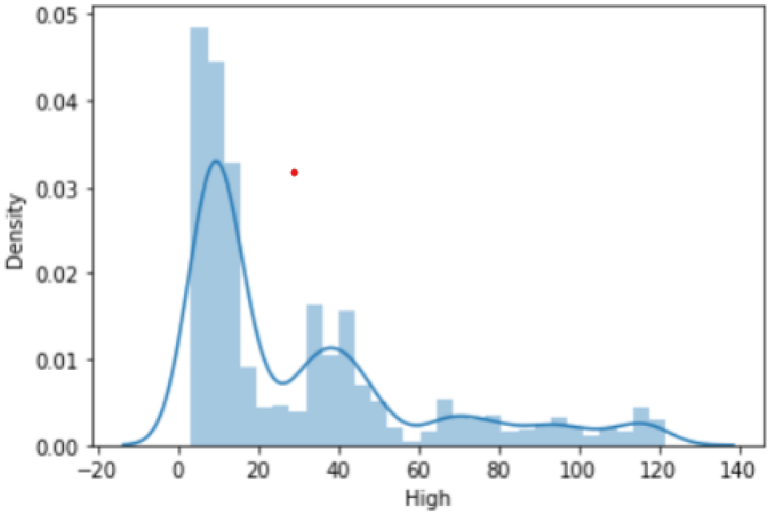
Out[5]:

```
<seaborn.axisgrid.PairGrid at 0x7caf0c2ac550>

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



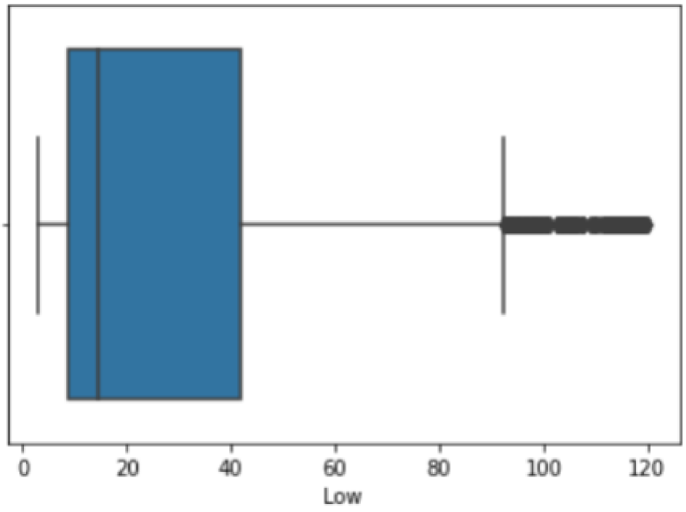
In[6]:
sns.distplot(df.High)

Out[6]:



In[7]:
sns.boxplot(df.Low)

Out[7]:



Visualising Correlation:

In [8]:

```
dataset.corr(numeric_only=True)
```

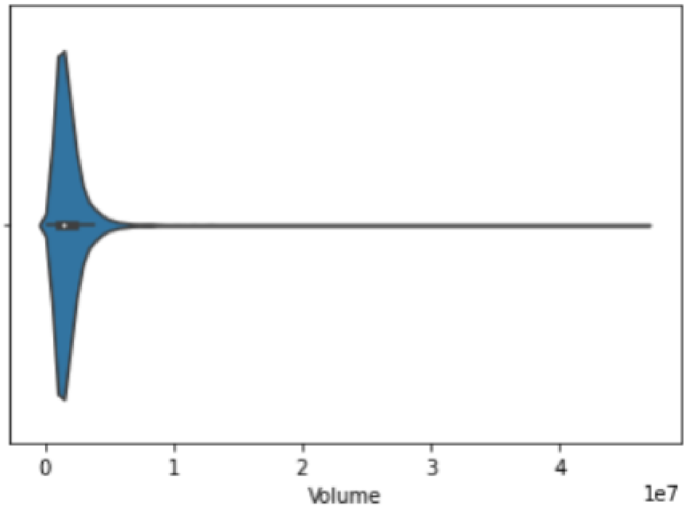
Out[8]:

	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[9]:

```
sns.violinplot(df.Volume)
```

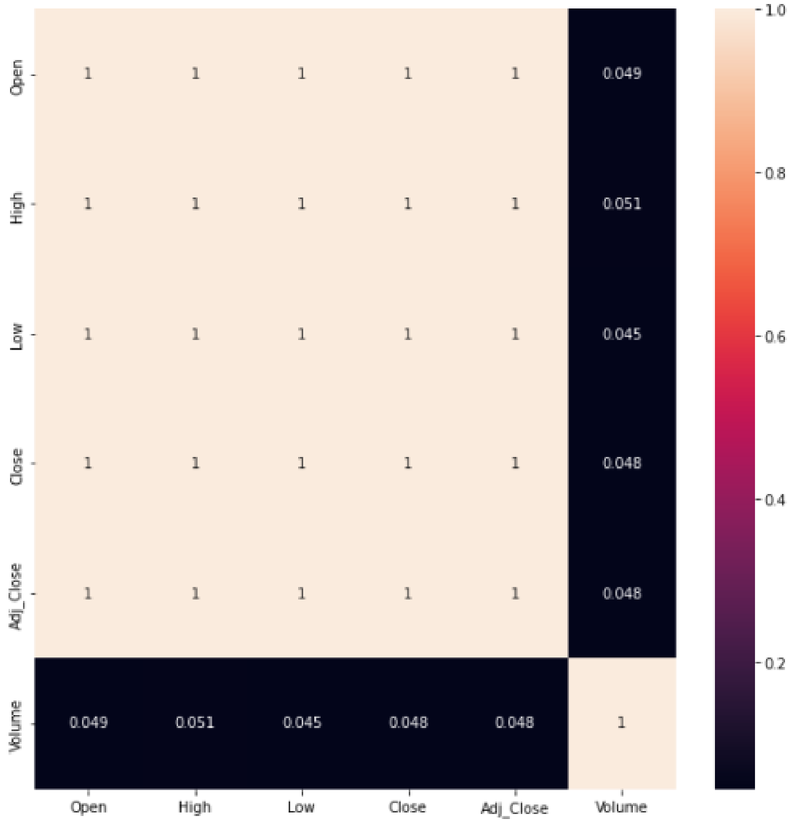
Out[9]:



In[10]:

```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(), annot=True)
```

Out[10]:



3. Model Selection:

Choose an appropriate machine learning model for your regression task. *Common choices include:*

- ✓ Linear Regression
- ✓ Decision Trees
- ✓ Random Forest
- ✓ Gradient Boosting *(e.g., XGBoost or LightGBM)*
- ✓ Neural Networks (Deep Learning)

Program:

Importing Dependencies

```
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 required
for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
```

Loading Dataset

```
dataset = pd.read_csv('D:/data/max.csv')
```

Model 1 - Linear Regression

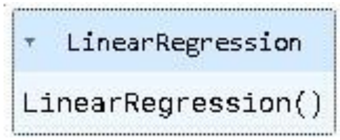
In [1]:

```
model_lr=LinearRegression()
```

In [2]:

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

Out[2]:



```
▾ LinearRegression
LinearRegression()
```

Predicting Prices

In [3]:

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

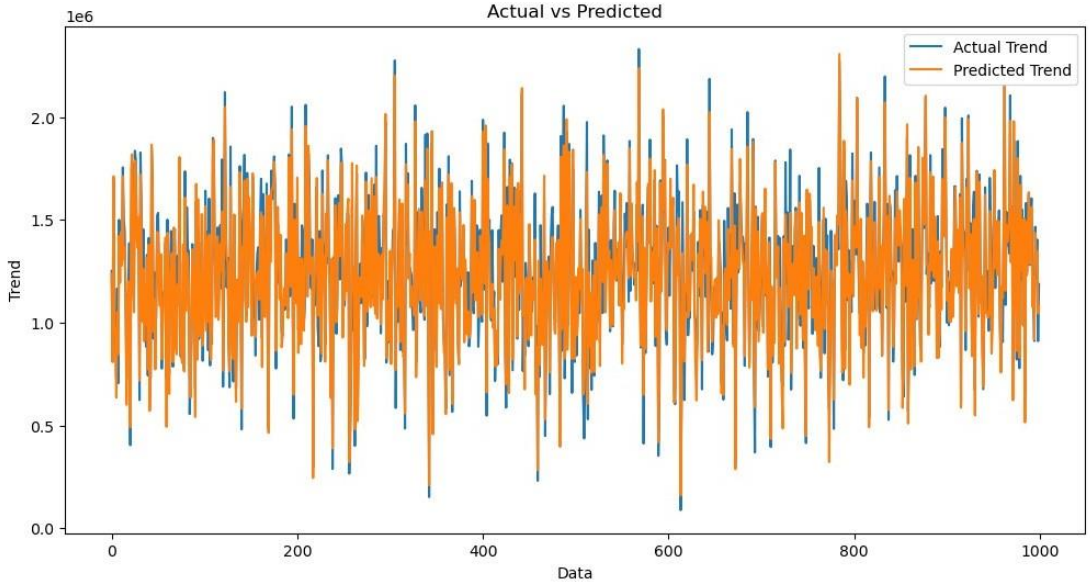
Evaluation of Predicted Data

In [4]:

```
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[4]:

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

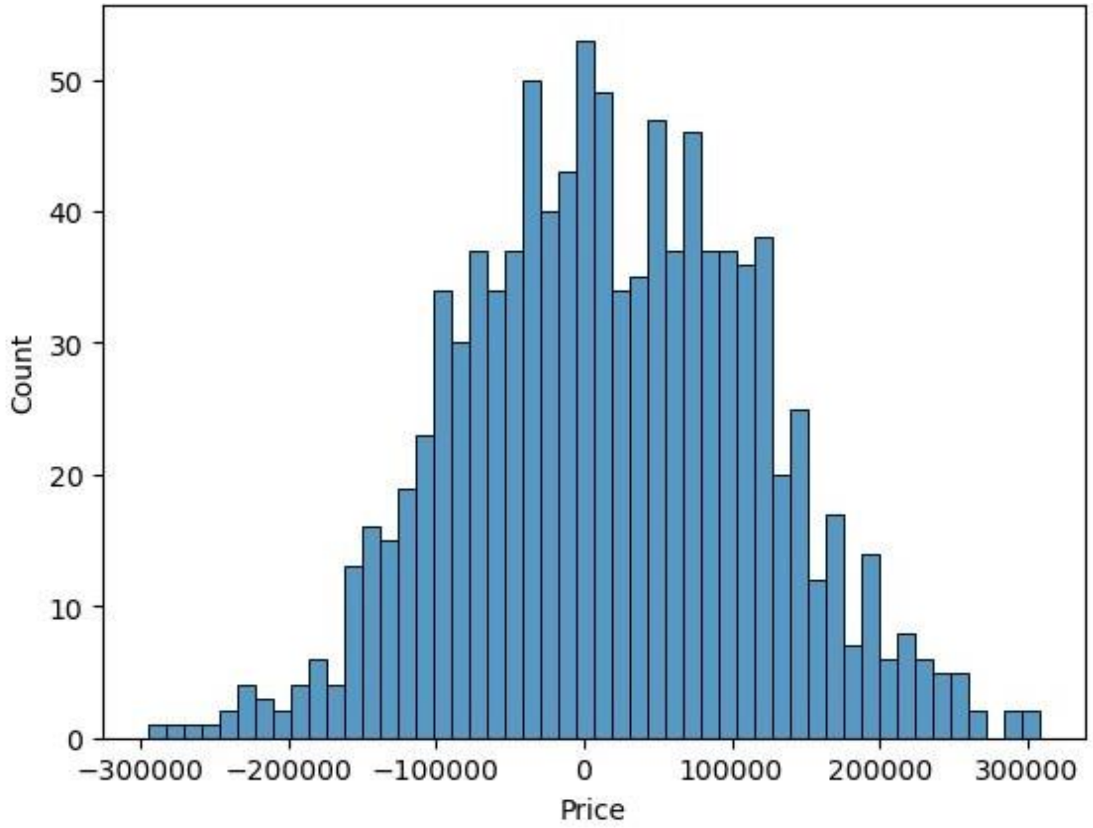


In [5]:

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

Out[5]:

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



Model 2 - Support Vector Regressor

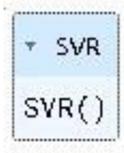
In [7]:

```
model_svr = SVR()
```

In [8]:

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

Out[8]:



```
▾ SVR
SVR()
```

Predicting Prices

In [9]:

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

Evaluation of Predicted Data

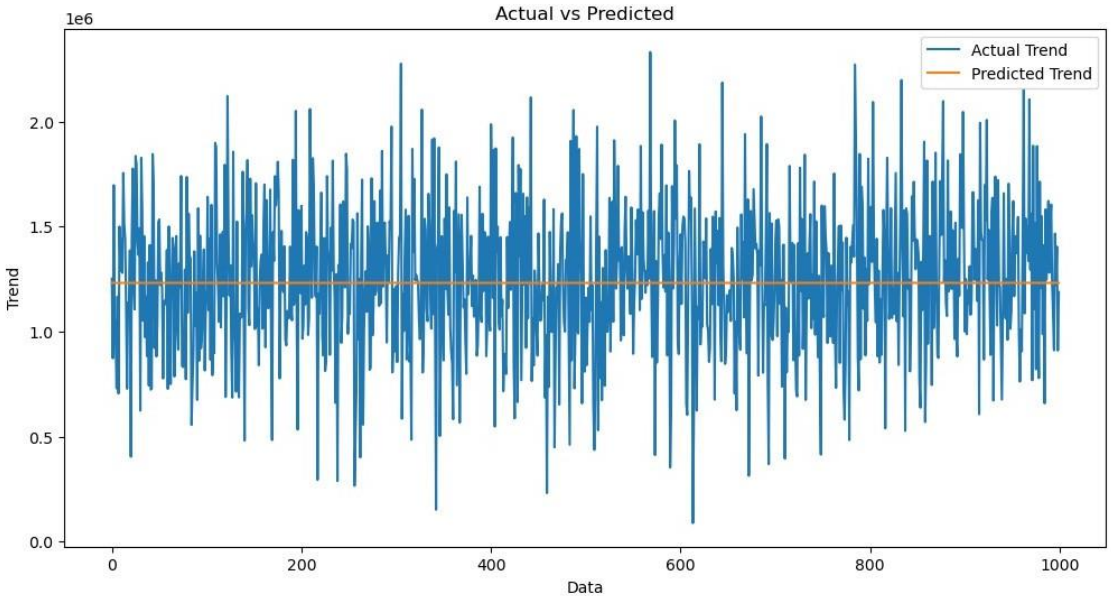
In [10]:

```
    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[10]:

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

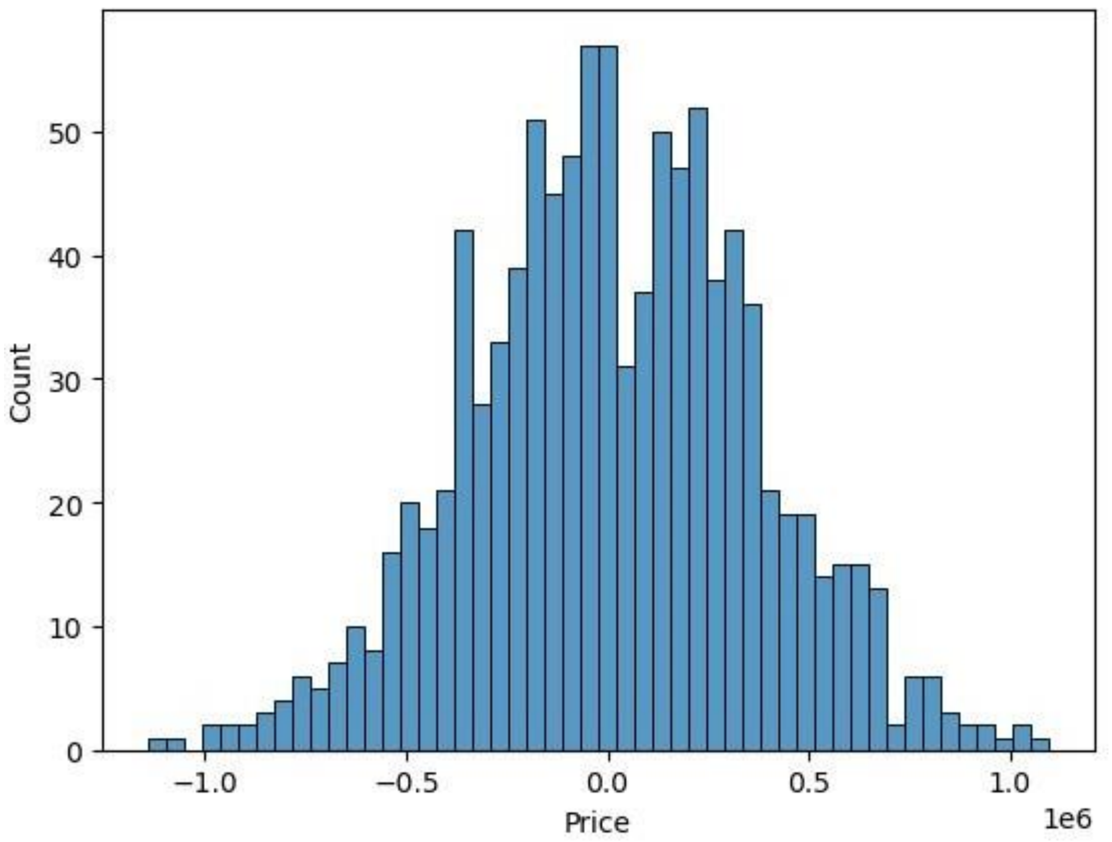


In [11]:

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

Out[12]:

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



In [12]:

```
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

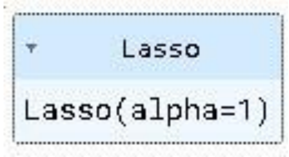
In [13]:

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

In [14]:

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

Out[14]:



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

Predicting Prices

In [15]:

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

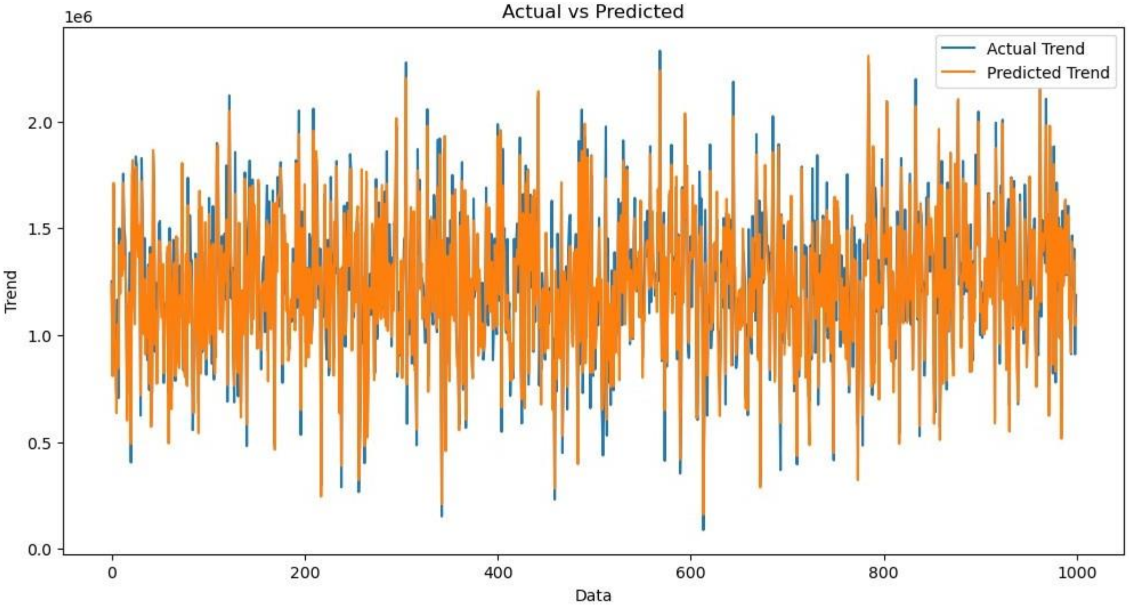
Evaluation of Predicted Data

In [16]:

```
    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[16]:

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

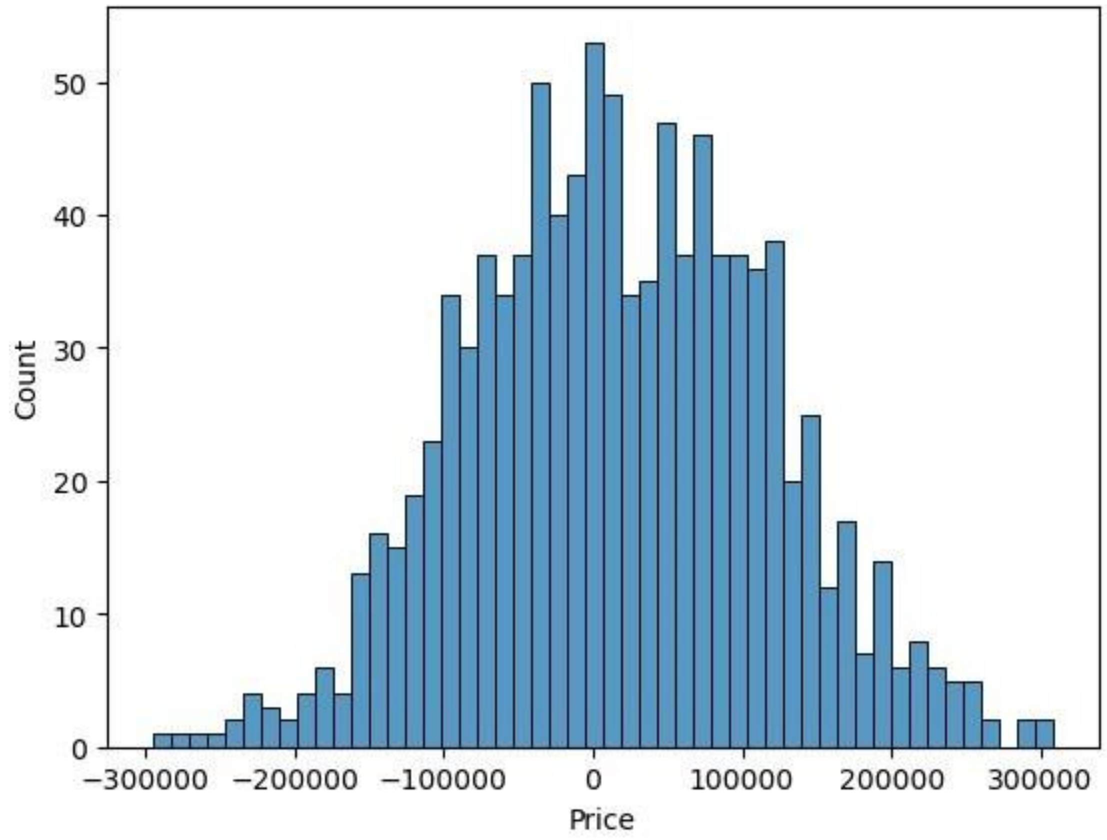


In [17]:

```
sns.histplot((Y_test-Prediction3), bins=50)
```

Out[17]:

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



In [18]:

```
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

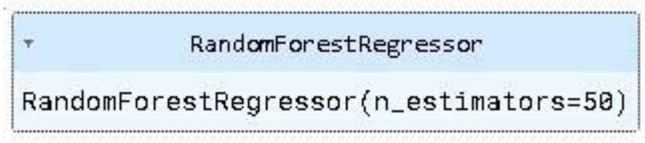
In [19]:

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

In [20]:

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

Out[20]:



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

Predicting Prices

In [21]:

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

Model 5 - XGboost Regressor

In [25]:

```
model_xg = xg.XGBRegressor()
```

In [26]:

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

Out[26]:

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, ...)
```

4. Model Training:

Split your dataset into training and testing sets (as shown earlier) and train the selected model on the training data. Here's an example using Linear Regression:

5. Model Evaluation:

Evaluate your model's performance using appropriate regression metrics, such as *Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE)*. For example:

PYTHON PROGRAM:

```
# Import necessary libraries

from sklearn.feature_selection import SelectKBest, f_regression

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error, r2_score

import numpy as np

selector = SelectKBest(score_func=f_regression, k=k)

X_train_selected = selector.fit_transform(X_train, y_train)
```

```
# Model Selection
# Let's choose both Linear Regression and Random Forest Regressor for comparison.
linear_reg_model = LinearRegression()
random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)
# Train the models on the selected features
linear_reg_model.fit(X_train_selected, y_train)
random_forest_model.fit(X_train_selected, y_train)
# Evaluate the models on the test set
X_test_selected = selector.transform(X_test)
# Make predictions
linear_reg_predictions = linear_reg_model.predict(X_test_selected)
random_forest_predictions = random_forest_model.predict(X_test_selected)
# Evaluate model performance
def evaluate_model(predictions, model_name):
    mse = mean_squared_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)
    print(f"{model_name} Model Evaluation:")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"R-squared (R2) Score: {r2}\n")
    # Performance Measure
    elr_mse = mean_squared_error(y_test, pred)
    elr_rmse = np.sqrt(lr_mse)
    elr_r2 = r2_score(y_test, pred)
# Show Measures
    result = '''
    MSE  : {}
    RMSE : {}
    R^2  : {}
    '''.format(lr_mse, lr_rmse, lr_r2)
    print(result)
    # Model Comparision
    names.append("elr")

    mses.append(elr_mse)

    rmses.append(elr_rmse)

    r2s.append(elr_r2)

evaluate_model(linear_reg_predictions, "Linear Regression")

evaluate_model(random_forest_predictions, "Random Forest Regressor")
```

OUTPUT:

```
Linear Regression Model Evaluation:
Mean Squared Error (MSE): 10089009300.893988
R-squared (R2) Score: 0.9179971706834331

Random Forest Regressor Model Evaluation:
Mean Squared Error (MSE): 14463028828.265167
R-squared (R2) Score: 0.8824454166872736
MSE  : 10141766848.330585
RMSE : 100706.33966305491
R^2  : 0.913302484308253
```

Model Comparison:

The less the Root Mean Squared Error (RMSE), The better the model is.

In [30]:

```
models.sort_values(by="RMSE (Cross-Validation)")
```

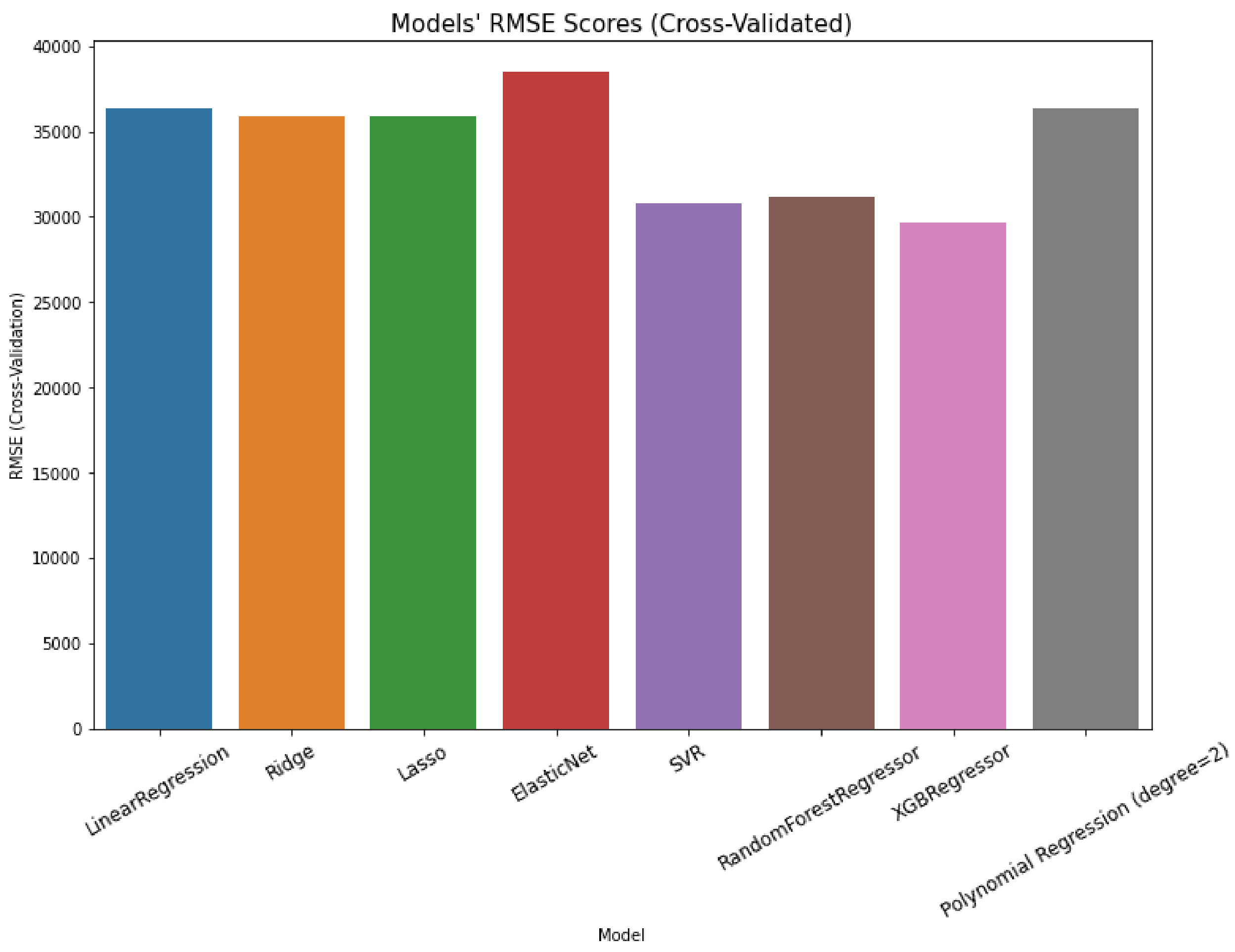
Out[30]:

	Model	MAE	MSE	RMSE	R2 Score	RMSE (Cross-Validation)
6	XGBRegressor	1.743992e+04	7.165790e+08	2.676899e+04	9.065778e-01	29698.849618
4	SVR	1.784316e+04	1.132136e+09	3.364723e+04	8.524005e-01	30745.475239
5	RandomForestRegressor	1.811511e+04	1.004422e+09	3.169262e+04	8.690509e-01	31138.863315
1	Ridge	2.343550e+04	1.404264e+09	3.747351e+04	8.169225e-01	35887.852792
2	Lasso	2.356046e+04	1.414338e+09	3.760768e+04	8.156092e-01	35922.769369
0	LinearRegression	2.356789e+04	1.414931e+09	3.761557e+04	8.155318e-01	36326.451445
7	Polynomial Regression (degree=2)	2.382228e+15	1.513991e+32	1.230443e+16	-1.973829e+22	36326.451445

	Model	MAE	MSE	RMSE	R2 Score	RMSE (Cross-Validation)
3	ElasticNet	2.379274e+04	1.718446e+09	4.145414e+04	7.759618e-01	38449.008646

In [31]:

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

sns.barplot(x=models["Model"], y=models["RMSE (Cross-Validation)"])

plt.title("Models' RMSE Scores (Cross-Validated)", size=15)

plt.xticks(rotation=30, size=12)

plt.show()
```



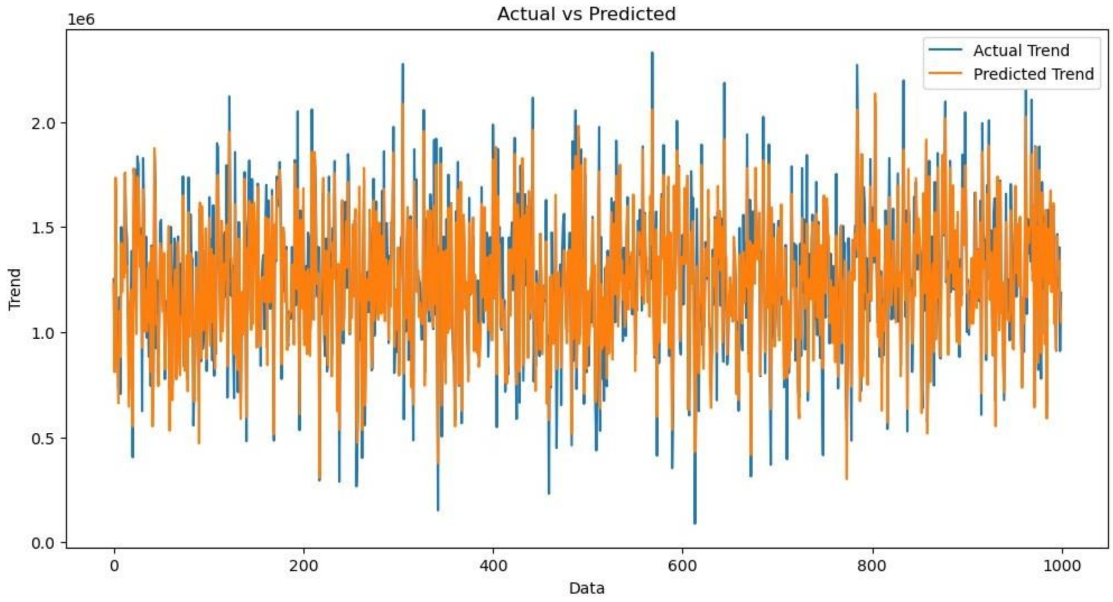
Evaluation of Predicted Data

In [22]:

```
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[22]:

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

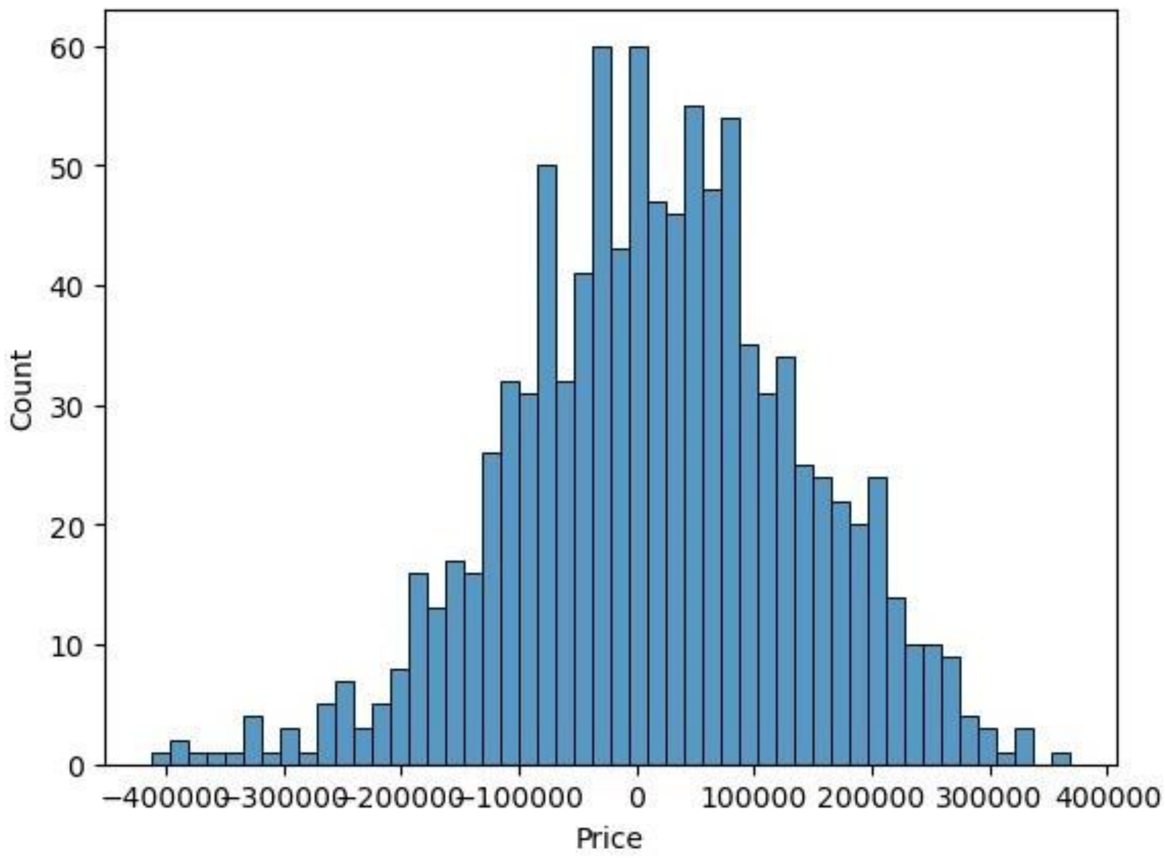


In [23]:

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

Out[23]:

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



In [24]:

```
print(r2_score(Y_test, Prediction2))

print(mean_absolute_error(Y_test, Prediction2))

print(mean_squared_error(Y_test, Prediction2))
```

Out [24] :

```
-0.0006222175925689744

286137.81086908665

128209033251.4034
```

6. Hyperparameter Tuning:

Optimize the model's hyperparameters to improve its performance. Depending on the model, you can use techniques like grid search or random search.

7. Cross-Validation:

Implement cross-validation to ensure that your model's performance is consistent across different subsets of your data. This helps prevent overfitting.

8. Regularization:

Apply regularization techniques like L1 (Lasso) or L2 (Ridge) if needed to prevent overfitting and improve model generalization.

Feature Selection:

1. Domain Knowledge:

- Start by considering which features are likely to be relevant for stock price prediction. These could include historical price data, trading volumes, financial indicators, and economic factors.

2. Correlation Analysis:

- Calculate the correlation between features and the target variable (stock prices). Features with high correlations are often good candidates for inclusion.

```python
correlation_matrix = data.corr()
relevant_features =
correlation_matrix['Target'].abs().sort_values(ascending=False)
```

3. Feature Importance from Tree-Based Models:

- Tree-based models like Random Forest or Gradient Boosting can provide feature importance scores. You can use these scores to identify important features.

```python
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor()
model.fit(X, y)
feature_importance = model.feature_importances_
```

4. Recursive Feature Elimination (RFE):

- RFE is a technique that recursively removes the least important features and selects the best subset based on model performance.

```python
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

model = LinearRegression()
rfe = RFE(model, n_features_to_select=5)
rfe.fit(X, y)
selected_features = X.columns[rfe.support_]
```

5. L1 Regularization (Lasso):

- L1 regularization can be used to encourage feature selection in linear models. It tends to drive some feature coefficients to zero, effectively selecting the most important ones.

```python
from sklearn.linear_model import Lasso

model = Lasso(alpha=0.01)
model.fit(X, y)
selected_features = X.columns[model.coef_ != 0]
```

6. Feature Engineering:

- Sometimes, creating new features from existing data can improve model performance. For example, you can calculate moving averages, technical indicators, or economic indices.

7. Backtesting and Cross-Validation:

- Evaluate feature sets using backtesting or cross-validation to ensure that they lead to consistent model performance. This helps in assessing the robustness of selected features.

8. Dimensionality Reduction(if necessary):

- Techniques like Principal Component Analysis (PCA) can be used to reduce the dimensionality of your feature space while retaining the most important information.

ADVANTAGES:

1. Data-Driven Insights:

Machine learning models can analyze vast amounts of historical stock price data and extract valuable insights that may not be apparent through traditional analysis.

2. Improved Accuracy:

ML models can potentially offer more accurate predictions compared to human intuition or simple statistical methods.

3. Speed:

Machine learning algorithms can process and analyze data quickly, allowing for near-real-time predictions, which is essential in the fast-paced world of stock trading.

4. Automation:

ML models can be set up to automatically make predictions, reducing the need for constant human monitoring.

5. Risk Management:

By providing insights into potential risks and market trends, ML can assist investors and traders in making informed decisions and managing their portfolios more effectively.

6. **Pattern Recognition:**

   Machine learning can identify complex patterns and relationships within financial data that may not be obvious to human analysts.

7. **Adaptability:**

   ML models can adapt to changing market conditions and incorporate new data, making them versatile tools for stock price prediction**.**

8. **Diversification:**

ML can help investors identify opportunities in a wide range of assets and markets, promoting diversification and risk reduction.

9. **Continuous Learning:**
   Some ML models can learn and improve over time as they encounter new data, enhancing their prediction accuracy.
10. **Reduced Emotion:**
   ML models are not influenced by emotional factors, such as fear or greed, which can cloud human judgment when making investment decisions.

DISADVANTAGES:

Stock price prediction using machine learning also comes with certain disadvantages and challenges:

1. Data Dependence:

ML models heavily rely on historical data, and they may not perform well in unforeseen market conditions or events.

2. Overfitting:

ML models can be prone to overfitting, where they fit the training data too closely and perform poorly on new, unseen data.

3. Complexity:

Many ML models are complex and require substantial computational resources, making them challenging to implement and maintain.

4. Lack of Causality:

ML models often identify correlations but may not provide a clear understanding of the causal factors behind stock price movements.

5. Market Noise:

Financial markets are influenced by a wide range of factors, including news, sentiment, and geopolitical events, which can be difficult to quantify and incorporate into models.

6. Black-Box Models:

Some ML algorithms are considered black-box models, meaning their decision-making processes are not easily interpretable, making it hard to understand why a particular prediction was made.

7. False Positives and Negatives:

ML models can generate false signals or fail to identify genuine opportunities, leading to financial losses.

8. Changing Market Dynamics:

Market conditions can change rapidly, and models that worked well in the past may not be effective in the future.

9. Regulatory Compliance:

Stock market regulations may limit the use of certain ML models for trading, and firms must comply with legal and ethical standards.

10.Risk of Financial Loss:

Relying solely on machine learning predictions can lead to financial losses, as there are no guarantees in the stock market.

11.Model Maintenance:

Continuous model updates and monitoring are required to ensure the models remain accurate and relevant.

12.Human Judgment:

While ML can provide insights, it should be used in conjunction with human judgment and other analysis tools to make well-informed investment decisions.

BENEFITS:

Stock price prediction using machine learning offers several benefits:

1. Data-Driven Insights:

Machine learning models can analyze vast amounts of historical and real-time data to provide valuable insights into stock market trends and behaviors.

2. Improved Accuracy:

ML models can potentially offer more accurate predictions compared to traditional methods, helping investors make more informed decisions.

3. Timely Decision-Making:

ML models can provide near-real-time predictions, allowing investors to react quickly to market changes and opportunities.

4. Risk Management:

Machine learning can identify potential risks and market anomalies, enabling investors to manage their portfolios and minimize losses.

5. Automated Trading:

ML models can be integrated into trading algorithms to automatically execute buy or sell orders based on predicted price movements.

6. Pattern Recognition:

Machine learning can identify complex patterns and relationships in stock data that may not be apparent to human analysts.

7. Diversification:

ML can help identify opportunities in a wide range of assets and markets, promoting diversification and risk reduction.

8. Continuous Learning:

Some ML models can learn and adapt over time, improving their predictive accuracy as they encounter new data.

9. Reduced Emotion:

ML models make predictions based on data and algorithms, avoiding emotional biases that can affect human decision-making.

10. Backtesting:

Machine learning models can be used to backtest investment strategies, helping investors assess their historical performance and refine their approaches.

11. Portfolio Optimization:

ML can assist in optimizing portfolio composition by suggesting the allocation of assets based on predicted returns and risk.

12. Quantitative Analysis:

Machine learning can quantify the impact of various factors on stock prices, helping investors better understand market dynamics.

13. Customization:

Investors can develop and fine-tune ML models to suit their specific investment goals and strategies.

It's important to note that while machine learning can provide significant benefits in stock price prediction, it is not without limitations and risks. Investors should use machine learning predictions as part of a broader strategy and exercise due diligence in their investment decisions.

CONCLUSION:

- In conclusion, stock price prediction using machine learning is a powerful and valuable tool for investors and traders. It offers the potential for data-driven insights, improved accuracy, and timely decision-making in the dynamic world of financial markets. Machine learning models can assist in risk management, pattern recognition, and portfolio optimization, contributing to more informed investment decisions.

- However, it's essential to be aware of the disadvantages and limitations associated with this approach. Machine learning models can be data-dependent, prone to overfitting, and may not account for the full complexity of market dynamics. They should be used as part of a broader investment strategy, incorporating human judgment, risk management, and compliance with regulatory standards.

- Ultimately, stock price prediction with machine learning is a valuable tool when used judiciously, but it should not be seen as a guarantee of success in the stock market. Investors should approach it with a critical and informed mindset, continually monitor and adapt their models, and consider the broader context of financial markets in their decision-making processes.

- stock price prediction using machine learning is a promising approach with its own set of advantages and challenges. It provides data-driven insights, potential for improved accuracy, and timely decision-making. Machine learning models can assist in risk management, pattern recognition, and portfolio optimization.

- However, it's important to acknowledge the limitations and uncertainties associated with this method. Market conditions can change rapidly, and machine learning models might not always perform well in unforeseen circumstances. They should be used as part of a comprehensive investment strategy that considers other factors, such as human judgment, diversification, and risk management.

- Investors and traders can benefit from the insights and efficiency offered by machine learning, but they should approach it with caution, continuously monitor and adapt their models, and be prepared for the inherent risks associated with financial markets.