PREDICTING STOCK PRICE USING MACHINE LEARNING

TEAM MEMBER

Phase-4 submission document

Project Title: STOCK PRICE PREDICTOR

Phase 4: *Development Part 2*

Topic: *Continue building the Stock price prediction model byfeature engineering, model training, and evaluation.*

Stock Price Prediction

Introduction:

- Stock price prediction is a challenging and vital task in the world of finance and investment. Accurate predictions can help traders and investors make informed decisions, mitigate risks, and maximize their returns. In this project, we will explore the development of a stock price prediction model using feature engineering, model training, and evaluation techniques.

- Feature engineering is a critical step that involves selecting and transforming relevant data to improve the model's predictive capabilities. It encompasses the creation of meaningful features from raw data, which can include factors like historical stock prices, trading volume, financial indicators, news sentiment, and more.

- The model training phase is where we leverage machine learning algorithms to build a predictive model. This involves selecting an appropriate algorithm, splitting the data into training and testing sets, and optimizing the model to ensure it can effectively capture patterns and relationships within the data.

- Finally, model evaluation is essential to assess the performance of our stock price prediction model. We will employ various evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) to measure the accuracy and reliability of our predictions. Additionally, we'll consider other factors like model interpretability and robustness.

- By combining feature engineering, model training, and rigorous evaluation, this project aims to create a stock price prediction model that can assist market participants in making more informed investment decisions, ultimately contributing to a better understanding of financial markets.

Given dataset:

Date	Open	High	Low	Close	Adj_Close	Volume
2000/3/27	3.8125	4.15625	3.8125	4.125	4.125	3675600
2000/3/28	4.125	4.125	4	4.015625	4.015625	1077600
2000/3/29	4	4.03125	3.953125	4	4	437200
2000/3/30	4	4	3.84375	3.84375	3.84375	1883600
2000/3/31	3.734375	3.734375	3.390625	3.390625	3.390625	7931600
2000/4/3	3.5	3.703125	3.4375	3.4375	3.4375	11486800
2000/4/4	3.53125	3.578125	3.09375	3.5	3.5	13136800
2000/4/5	3.46875	3.5625	3.453125	3.484375	3.484375	6349600
2000/4/6	3.5	3.59375	3.46875	3.578125	3.578125	7181200
2000/4/7	3.59375	3.8125	3.59375	3.609375	3.609375	13904800
2000/4/10	3.6875	3.75	3.625	3.640625	3.640625	5280800
2000/4/11	3.578125	3.65625	3.5625	3.578125	3.578125	6590000
2000/4/12	3.546875	3.640625	3.53125	3.578125	3.578125	8546400

Overview of the process:

The following is an overview of the process of building a stock price prediction model by feature selection, model training, and evaluation:

Data Collection:

- Gather historical stock price data, including open, high, low, close prices, and trading volume.

- Collect any additional data that might influence stock prices, such as economic indicators or news sentiment.

Data Preprocessing:

- Handle missing data by imputing or removing it.

- Ensure data is in chronological order and handle any data anomalies.

- Normalize or scale the data to make it consistent and suitable for modeling.

Model Training:

- Train the selected model using the training dataset. The input features are used to predict future stock prices.
- Optimize hyper parameters through techniques like cross-validation to improve model performance.

Feature Engineering:

- Create relevant features from the collected data.

This could involve:

- Lag features: Past prices, volumes, and returns.
- Technical indicators: Moving averages, Relative Strength Index (RSI), MACD, Bollinger Bands, etc.
- Sentiment analysis: News sentiment scores or social media sentiment related to the stock.
- Fundamental indicators: Earnings, P/E ratio, dividend yield, etc.
- Explore and analyze the data to identify potentially influential features.

Model Evaluation:

- Assess the model's performance using the testing dataset. Common evaluation metrics include:
  - Mean Absolute Error (MAE)
  - Mean Squared Error (MSE)
  - Root Mean Squared Error (RMSE)
  - Mean Absolute Percentage Error (MAPE)
- Visualize and compare the model's predictions with actual stock prices to understand its accuracy and behavior.

Deployment:

- Once satisfied with the model's performance, deploy it in a production environment where it can make real-time or future stock price predictions.

PROCEDURE:

Feature selection:

1. **Identify the target variable.** This is the variable that you want to predict, such as house price.
2. **Explore the data.** This will help you to understand the relationships between the different features and the target variable. You can use data visualization and correlation analysis to identify features that are highly correlated with the target variable.
3. **Remove redundant features.** If two features are highly correlated with each other, then you can remove one of the features, as they are likely to contain redundant information.
4. **Remove irrelevant features.** If a feature is not correlated with the target variable, then you can remove it, as it is unlikely to be useful for prediction.

PROGRAM:

In[1]:

```
import numpy as np # linear algebra
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In[2]:

```
df = pd.read_csv('D:\data\max.csv')
print(df)
```

Out[2]:

	Date	Open	High	...	Close	Adj_Close	Volume
0	2000/3/27	3.812500	4.156250	...	4.125000	4.125000	3675600
1	2000/3/28	4.125000	4.125000	...	4.015625	4.015625	1077600
2	2000/3/29	4.000000	4.031250	...	4.000000	4.000000	437200
3	2000/3/30	4.000000	4.000000	...	3.843750	3.843750	1883600
4	2000/3/31	3.734375	3.734375	...	3.390625	3.390625	7931600
...	...	...	...	...	...	...	...
4387	2017/9/1	113.790001	114.099998	...	113.309998	113.309998	950000
4388	2017/9/5	112.519997	113.529999	...	111.870003	111.870003	1805200
4389	2017/9/6	112.029999	112.489998	...	112.230003	112.230003	2136700
4390	2017/9/7	112.459999	112.900002	...	112.339996	112.339996	1251600
4391	2017/9/8	112.300003	114.790001	...	113.190002	113.190002	1611700

[4392 rows x 7 columns]

In[3]:

df.head()

Out[3]:

	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

In[4]:

df.tail()

Out[4]:

	Date	Open	High	...	Close	Adj_Close	Volume
4387	2017/9/1	113.790001	114.099998	...	113.309998	113.309998	950000
4388	2017/9/5	112.519997	113.529999	...	111.870003	111.870003	1805200
4389	2017/9/6	112.029999	112.489998	...	112.230003	112.230003	2136700
4390	2017/9/7	112.459999	112.900002	...	112.339996	112.339996	1251600
4391	2017/9/8	112.300003	114.790001	...	113.190002	113.190002	1611700

[5 rows x 7 columns]

In[5]:

df.info()

Out[5]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4392 entries, 0 to 4391
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       4392 non-null   object 
 1   Open       4392 non-null   float64
 2   High       4392 non-null   float64
 3   Low        4392 non-null   float64
 4   Close      4392 non-null   float64
 5   Adj_Close  4392 non-null   float64
 6   Volume     4392 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 240.3+ KB
None
```

In[6]:

pd.isnull(df).sum()

Out[6]:

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj_Close    0
Volume       0
dtype: int64
```

In[7]:

```
sorted_df = df.sort_values('Volume', ascending = True).head(10)
print(sorted_df)
```

Out[7]:

	Date	Open	High	Low	Close	Adj_Close	Volume
1425	2005/11/25	10.275000	10.352500	10.260000	10.325000	10.325000	190400
104	2000/8/23	5.718750	5.734375	5.640625	5.718750	5.718750	220000
93	2000/8/8	5.671875	5.687500	5.500000	5.625000	5.625000	224000
923	2003/11/28	7.500000	7.582500	7.477500	7.530000	7.530000	224800
1926	2007/11/23	12.282500	12.462500	12.257500	12.422500	12.422500	225600
320	2001/7/3	6.587500	6.637500	6.562500	6.562500	6.562500	226000
322	2001/7/6	6.535000	6.535000	6.370000	6.370000	6.370000	246400
150	2000/10/27	5.828125	5.875000	5.750000	5.750000	5.750000	262000
942	2003/12/26	7.450000	7.500000	7.425000	7.472500	7.472500	263600
116	2000/9/11	5.593750	5.593750	5.531250	5.578125	5.578125	264400

In[8]:

```
sorted_df = df.sort_values('High', ascending = True).head(10)
print(sorted_df)
```

Out[8]:

	Date	Open	High	Low	Close	Adj_Close	Volume
153	2000/11/1	3.375000	3.390625	3.000000	3.250000	3.250000	19073200
154	2000/11/2	3.296875	3.562500	3.265625	3.546875	3.546875	3819600
7	2000/4/5	3.468750	3.562500	3.453125	3.484375	3.484375	6349600
166	2000/11/20	3.578125	3.578125	3.390625	3.437500	3.437500	530800
6	2000/4/4	3.531250	3.578125	3.093750	3.500000	3.500000	13136800
8	2000/4/6	3.500000	3.593750	3.468750	3.578125	3.578125	7181200
165	2000/11/17	3.609375	3.609375	3.390625	3.546875	3.546875	1190000
15	2000/4/17	3.578125	3.609375	3.515625	3.562500	3.562500	2992000
16	2000/4/18	3.609375	3.625000	3.484375	3.515625	3.515625	2896000
14	2000/4/14	3.609375	3.625000	3.531250	3.609375	3.609375	2626000

In[9]:

df.describe().transpose()

Out[9]:

	count	mean	...	75%	max
Open	4392.0	3.056254e+01	...	4.254625e+01	1.210800e+02
High	4392.0	3.089362e+01	...	4.305125e+01	1.217500e+02
Low	4392.0	3.023883e+01	...	4.208625e+01	1.201700e+02
Close	4392.0	3.057258e+01	...	4.254000e+01	1.213600e+02
Adj_Close	4392.0	3.057258e+01	...	4.254000e+01	1.213600e+02
Volume	4392.0	1.884027e+06	...	2.188900e+06	4.641260e+07

[6 rows x 8 columns]

In[10]:

most_volume = df.query('Volume>1500000', inplace = False).sort_values('Volume', ascending = False)
print(most_volume[:10])

Out[10]:

	Date	Open	High	...	Close	Adj_Close	Volume
2770	2011/3/31	44.005001	44.095001	...	43.500000	43.500000	46412600
2103	2008/8/7	14.082500	14.547500	...	14.460000	14.460000	23767600
3288	2013/4/24	33.275002	33.299999	...	32.299999	32.299999	22986800
153	2000/11/1	3.375000	3.390625	...	3.250000	3.250000	19073200
152	2000/10/31	3.812500	4.000000	...	3.359375	3.359375	17944400
3155	2012/10/9	44.779999	44.794998	...	42.299999	42.299999	17734800
833	2003/7/23	8.000000	8.000000	...	6.875000	6.875000	16833200
4173	2016/10/26	99.300003	100.860001	...	94.250000	94.250000	16796600
3533	2014/4/14	40.965000	42.200001	...	40.500000	40.500000	14450200
9	2000/4/7	3.593750	3.812500	...	3.609375	3.609375	13904800

[10 rows x 7 columns]

In[11]:

```
df[["High"]].iloc[18]
```

Out[12]:

```
High    3.6875
Name: 18, dtype: float64
```

In[12]:

```
df[["Low"]].iloc[10]
```

Out[12]:

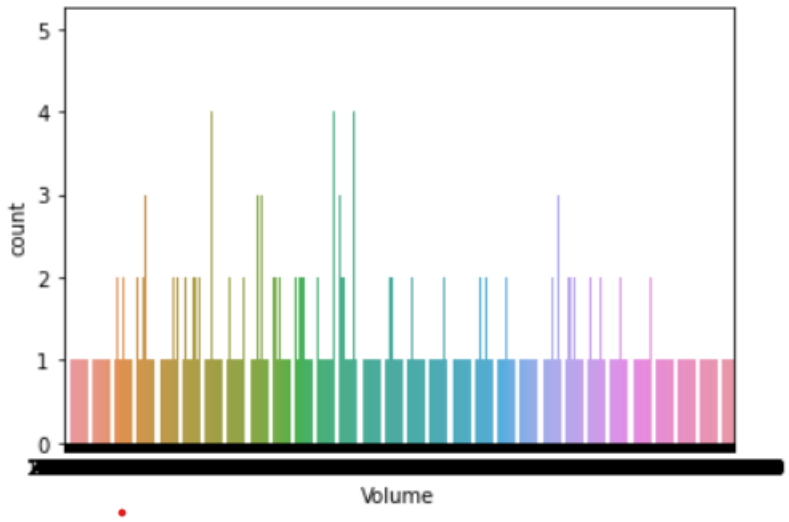
```
Low    3.625
Name: 10, dtype: float64
```

In[13]:

```
sns.countplot(df.Volume)
```

Out[13]:

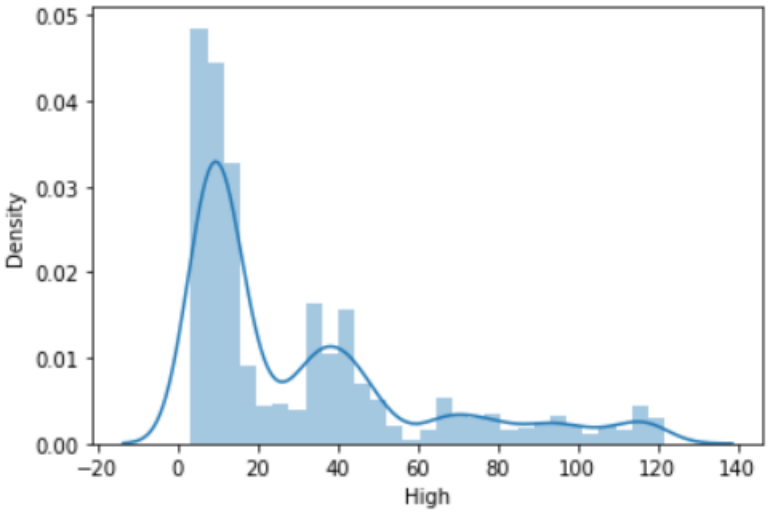
Axes(0.125,0.11;0.775x0.77)



In[14]:

sns.distplot(df.High)

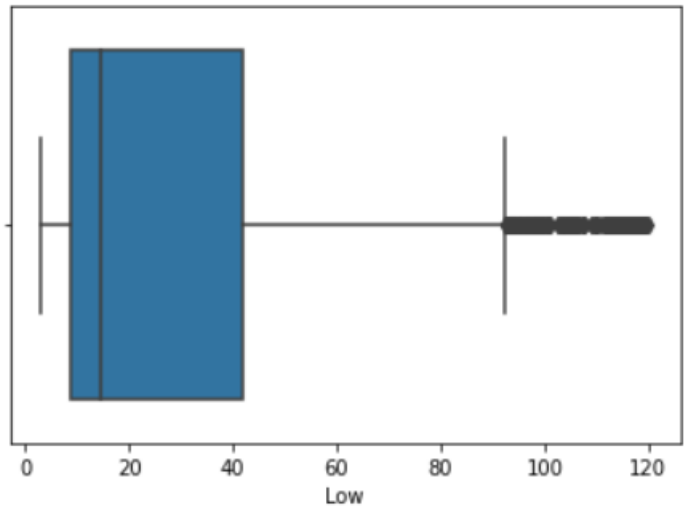
Out[14]:



In[15]:

sns.boxplot(df.Low)

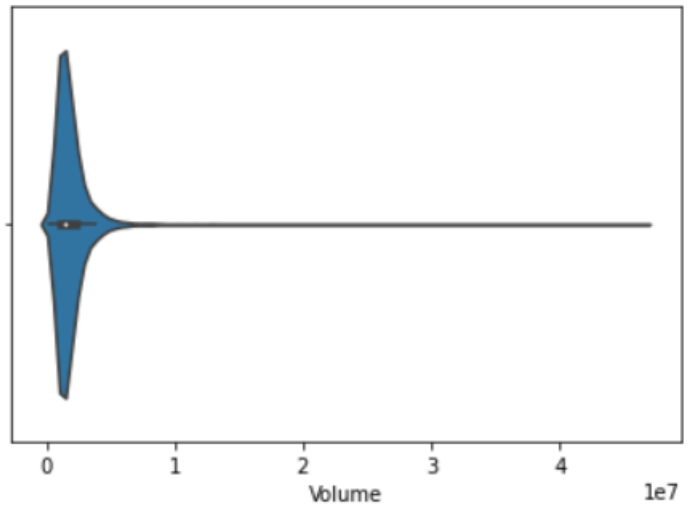
Out[15]:



In[16]:

sns.violinplot(df.Volume)

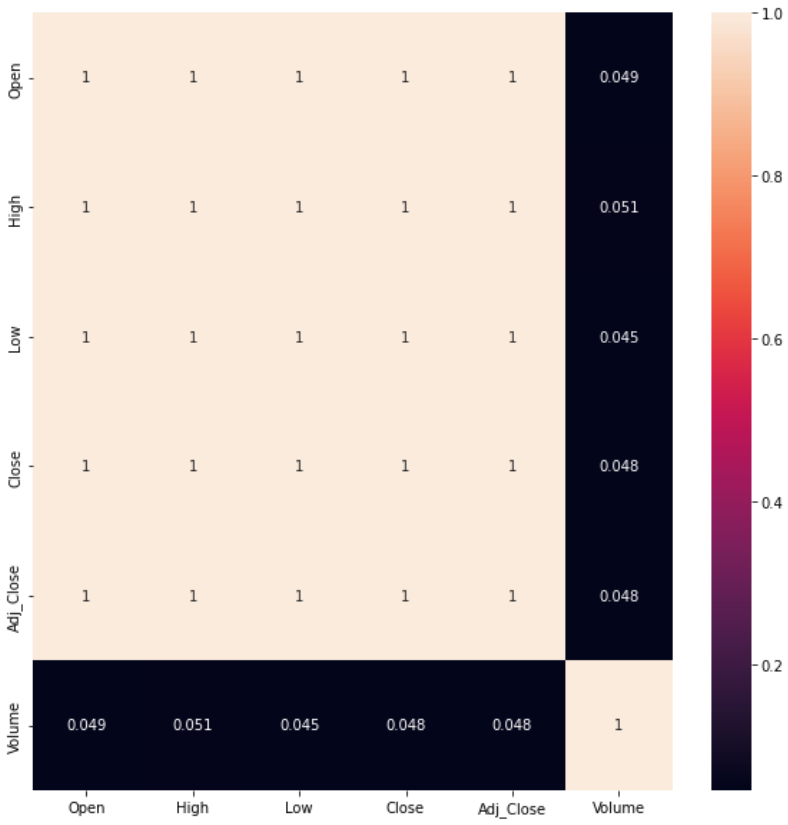
Out[16]:



In[17]:

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

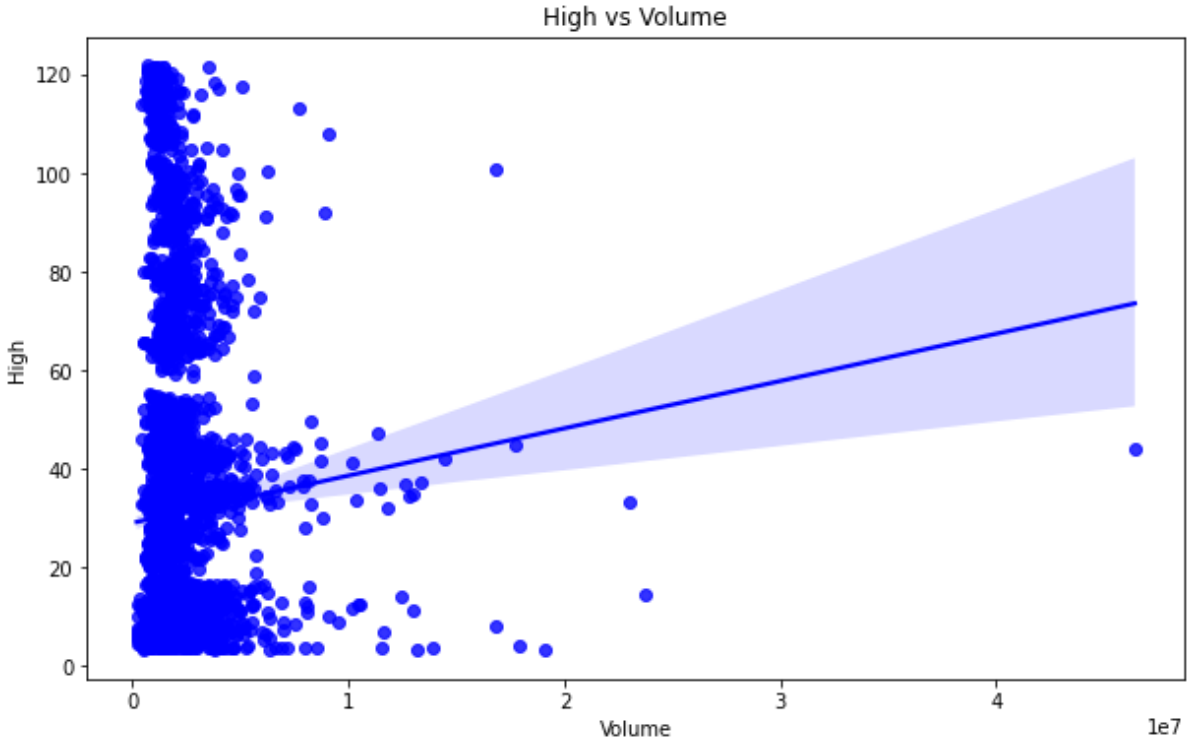
Out[17]:



In[18]:

```
plt.figure(figsize=(10,6))
sns.regplot(data = df, y = "High", x ="Volume", color = "b").set(title = "High vs Volume")
```

Out[18]:



Model training:

1. **Choose a machine learning algorithm.** There are a number of different machine learning algorithms that can be used for stock price prediction, such as linear regression, ridge regression, lasso regression, decision trees, and random forests are Covered above.

Machine Learning Model:

In[1]:

```
import pandas as pd
import matplotlib.pyplot as plt
from pylab import rcParams
import numpy as np
import seaborn as sns
import os
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score, train_test_split,
GridSearchCV
from sklearn.feature_selection import RFECV, SelectFromModel, SelectKBest
from sklearn.preprocessing import StandardScaler
```

```
from sklearn import metrics
%matplotlib inline
```

In[2]:

```
df_Stock.columns
```

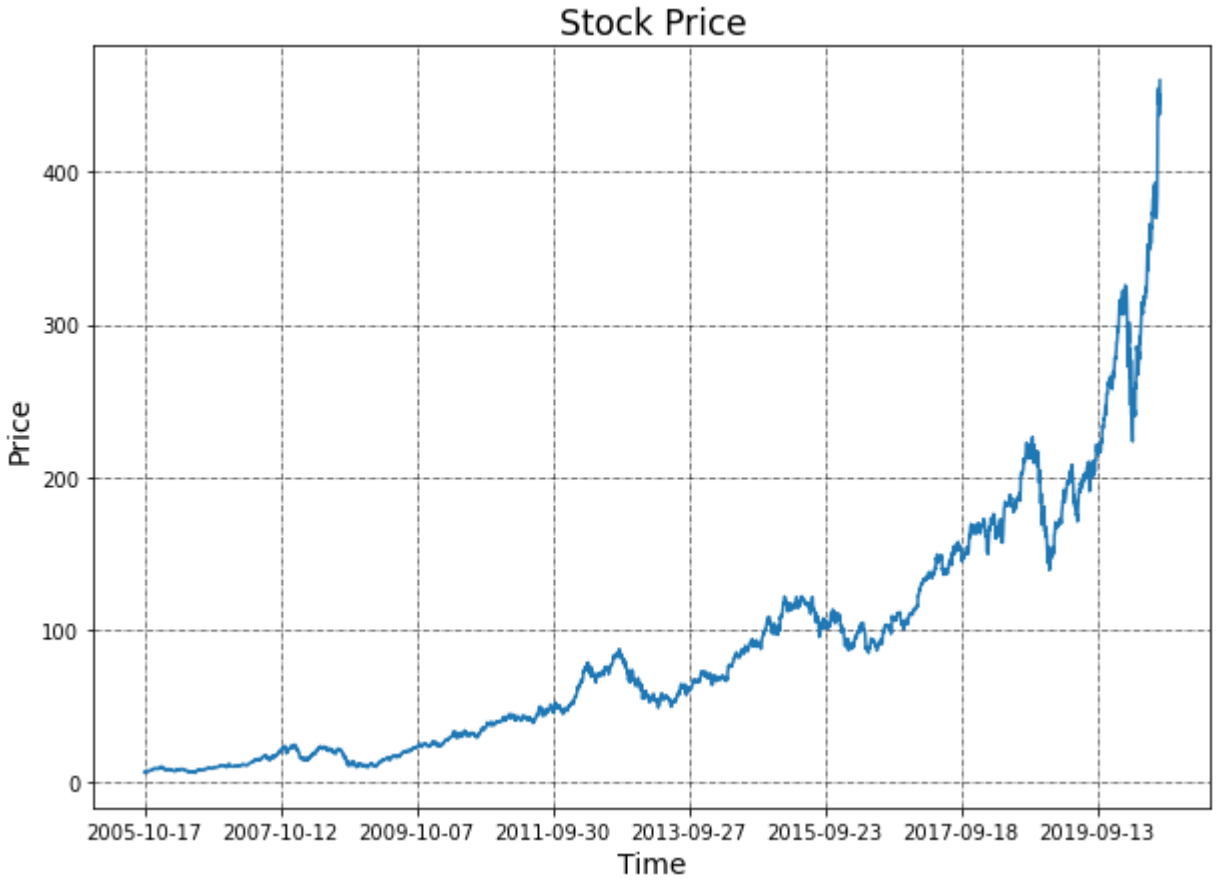
Out[2]:

```
Index(['Open', 'High', 'Low', 'Close', 'Volume', 'SD20', 'Upper_Band',
       'Lower_Band', 'S_Close(t-1)', 'S_Close(t-2)', 'S_Close(t-3)',
       'S_Close(t-5)', 'S_Open(t-1)', 'MA5', 'MA10', 'MA20', 'MA50', 'MA200',
       'EMA10', 'EMA20', 'EMA50', 'EMA100', 'EMA200', 'MACD', 'MACD_EMA',
       'ATR', 'ADX', 'CCI', 'ROC', 'RSI', 'William%R', 'SO%K', 'STD5',
       'ForceIndex1', 'ForceIndex20', 'Date_col', 'Day', 'DayofWeek',
       'DayofYear', 'Week', 'Is_month_end', 'Is_month_start', 'Is_quarter_end',
       'Is_quarter_start', 'Is_year_end', 'Is_year_start', 'Is_leap_year',
       'Year', 'Month', 'QQQ_Close', 'QQQ(t-1)', 'QQQ(t-2)', 'QQQ(t-5)',
       'QQQ_MA10', 'QQQ_MA20', 'QQQ_MA50', 'SnP_Close', 'SnP(t-1))',
       'SnP(t-5)', 'DJIA_Close', 'DJIA(t-1))', 'DJIA(t-5)', 'Close_forcast'],
      dtype='object')
```

In[3]:

```
df_Stock['Close'].plot(figsize=(10, 7))
plt.title("Stock Price", fontsize=17)
plt.ylabel('Price', fontsize=14)
plt.xlabel('Time', fontsize=14)
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```

Out[3]:



In[4]:

```
def create_train_test_set(df_Stock):

    features = df_Stock.drop(columns=['Close_forcast'], axis=1)
    target = df_Stock['Close_forcast']


    data_len = df_Stock.shape[0]
    print('Historical Stock Data length is - ', str(data_len))

    #create a chronological split for train and testing
    train_split = int(data_len * 0.88)
    print('Training Set length - ', str(train_split))

    val_split = train_split + int(data_len * 0.1)
    print('Validation Set length - ', str(int(data_len * 0.1)))

    print('Test Set length - ', str(int(data_len * 0.02)))
```

```
 X_train, X_val, X_test = features[:train_split], features[train_split:val_split], features[val_split:]
    Y_train, Y_val, Y_test = target[:train_split], target[train_split:val_split], target[val_split:]

    #print shape of samples
    print(X_train.shape, X_val.shape, X_test.shape)
    print(Y_train.shape, Y_val.shape, Y_test.shape)

    return X_train, X_val, X_test, Y_train, Y_val, Y_test
X_train, X_val, X_test, Y_train, Y_val, Y_test = create_train_test_set(df_Stock)
```

Out[4]:

```
Historical Stock Data length is -  3732
Training Set length -  3284
Validation Set length -  373
Test Set length -  74
(3284, 61) (373, 61) (75, 61)
(3284,) (373,) (75,)
```

In[5]:

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, Y_train)
```

Out[5]:

```
LinearRegression()
```

In[6]:

```
print('LR Coefficients: \n', lr.coef_)
```

print('LR Intercept: \n', lr.intercept_)

Out[6]:

```
LR Coefficients:
 [ 8.63711643e-03  1.86051940e-01  1.55487122e-01  1.12263751e+00
  1.27286803e-10  6.75248222e-03  1.40229140e-01  1.13219364e-01
  4.25628128e-02  8.96348608e-02  1.01914941e-01  5.94183582e-02
  7.95194237e-02  7.10399936e-02  2.71424998e-01  1.26724315e-01
  8.79333083e-02 -5.87980436e-03 -3.31643395e-01 -3.31643395e-01
 -3.31643395e-01 -3.31643395e-01 -3.31643395e-01  1.88650022e+00
 -1.27270733e+00 -1.65042221e-01 -4.36658019e-04  3.32218457e-13
 -5.07434686e-03  9.02936465e-03  5.78316992e-04  5.78316992e-04
 -5.57918343e-01 -2.02304395e-10  4.18932250e-11  1.69322436e-02
  1.61636707e-02 -1.75659581e-02  6.12165522e-03  2.15420350e-01
  1.13979656e-01 -2.41954674e-01  7.63050303e-02  3.73276599e-01
 -4.99600361e-16 -5.60843986e-02  4.08788810e-02  5.13473858e-01
 -2.94431537e-02 -8.41335090e-02  5.10939137e-02 -8.14435741e-03
 -1.95035187e-02  5.67587247e-02  4.39707790e-02  1.29311735e-02
 -9.99967545e-03 -3.89778364e-03 -1.62174821e-03  1.44436912e-03
  2.83455406e-04]
LR Intercept:
 -83.36486497274721
```

In[7]:

print("Performance (R^2): ", lr.score(X_train, Y_train))

Out[7]:

Performance (R^2): 0.9994516474373267

In[8]:

```
def get_mape(y_true, y_pred):
    """
    Compute mean absolute percentage error (MAPE)
```

```
   y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
Y_train_pred = lr.predict(X_train)
Y_val_pred = lr.predict(X_val)
Y_test_pred = lr.predict(X_test)

print("Training R-squared: ",round(metrics.r2_score(Y_train,Y_train_pred),2))
print("Training Explained Variation:
",round(metrics.explained_variance_score(Y_train,Y_train_pred),2))
print('Training MAPE:', round(get_mape(Y_train,Y_train_pred), 2))
print('Training Mean Squared Error:',
round(metrics.mean_squared_error(Y_train,Y_train_pred), 2))
print("Training RMSE:
",round(np.sqrt(metrics.mean_squared_error(Y_train,Y_train_pred)),2))
print("Training MAE:
",round(metrics.mean_absolute_error(Y_train,Y_train_pred),2))

print(' ')

print("Validation R-squared: ",round(metrics.r2_score(Y_val,Y_val_pred),2))
print("Validation Explained Variation:
",round(metrics.explained_variance_score(Y_val,Y_val_pred),2))
print('Validation MAPE:', round(get_mape(Y_val,Y_val_pred), 2))
print('Validation Mean Squared Error:',
round(metrics.mean_squared_error(Y_train,Y_train_pred), 2))
print("Validation RMSE:
",round(np.sqrt(metrics.mean_squared_error(Y_val,Y_val_pred)),2))
print("Validation MAE:
",round(metrics.mean_absolute_error(Y_val,Y_val_pred),2))

print(' ')

print("Test R-squared: ",round(metrics.r2_score(Y_test,Y_test_pred),2))
print("Test Explained Variation:
",round(metrics.explained_variance_score(Y_test,Y_test_pred),2))
print('Test MAPE:', round(get_mape(Y_test,Y_test_pred), 2))
```

```
print('Test Mean Squared Error:',
round(metrics.mean_squared_error(Y_test,Y_test_pred), 2))
print("Test RMSE:
",round(np.sqrt(metrics.mean_squared_error(Y_test,Y_test_pred)),2))
print("Test MAE: ",round(metrics.mean_absolute_error(Y_test,Y_test_pred),2))
```

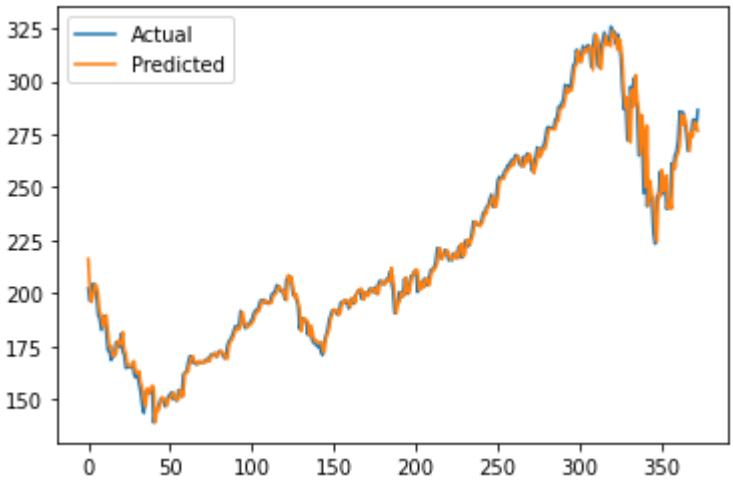
Out[8]:

```
Training R-squared:  1.0
Training Explained Variation:  1.0
Training MAPE: 1.45
Training Mean Squared Error: 1.48
Training RMSE:  1.22
Training MAE:  0.76

Validation R-squared:  0.99
Validation Explained Variation:  0.99
Validation MAPE: 1.68
Validation Mean Squared Error: 1.48
Validation RMSE:  5.91
Validation MAE:  3.75

Test R-squared:  0.96
Test Explained Variation:  0.97
Test MAPE: 1.77
Test Mean Squared Error: 79.21
Test RMSE:  8.9
Test MAE:  6.5
```

In[9]:

```
df_pred[['Actual', 'Predicted']].plot()
```

Out[9]:



Model evaluation:

1. **Calculate the evaluation metrics.** There are a number of different evaluation metrics that can be used to assess the performance of a machine learning model, such as ***R-squared, mean squared error (MSE), and root mean squared error (RMSE).***

2. **Interpret the evaluation metrics.** The evaluation metrics will give you an idea of how well the model is performing on unseen data. If the model is performing well, then you can be confident that it will generalize well to new data. However, if the model is performing poorly, then you may need to try a different model or retune the hyperparameters of the current model.

Model evaluation:

- Model evaluation is the process of assessing the performance of a machine learning model on unseen data. This is important to ensure that the model will generalize well to new data.
- There are a number of different metrics that can be used to evaluate the performance of a stock price prediction model. Some of the most common metrics include:

- **Mean squared error (MSE):** This metric measures the average squared difference between the predicted and actual stock prices.
- **Root mean squared error (RMSE):** This metric is the square root of the MSE.
- **Mean absolute error (MAE):** This metric measures the average absolute difference between the predicted and actual stock prices.
- **R-squared:** This metric measures how well the model explains the variation in the actual stock prices.

In addition to these metrics, it is also important to consider the following factors when evaluating a stock price prediction model:

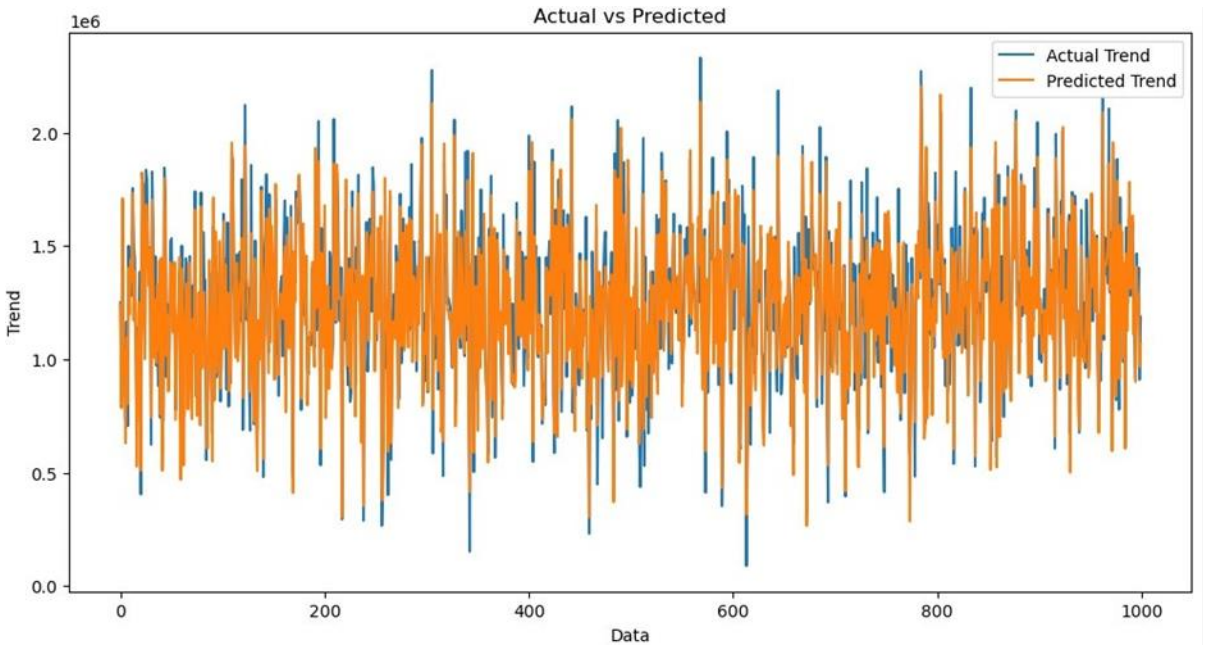
- **Bias:** Bias is the tendency of a model to consistently over- or underestimate stock prices.
- **Variance:** Variance is the measure of how much the predictions of a model vary around the true stock prices.
- **Interpretability:** Interpretability is the ability to understand how the model makes its predictions. This is important for stock price prediction models, as it allows users to understand the factors that influence the predicted stock prices.

In[1]:

```
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[1]:

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

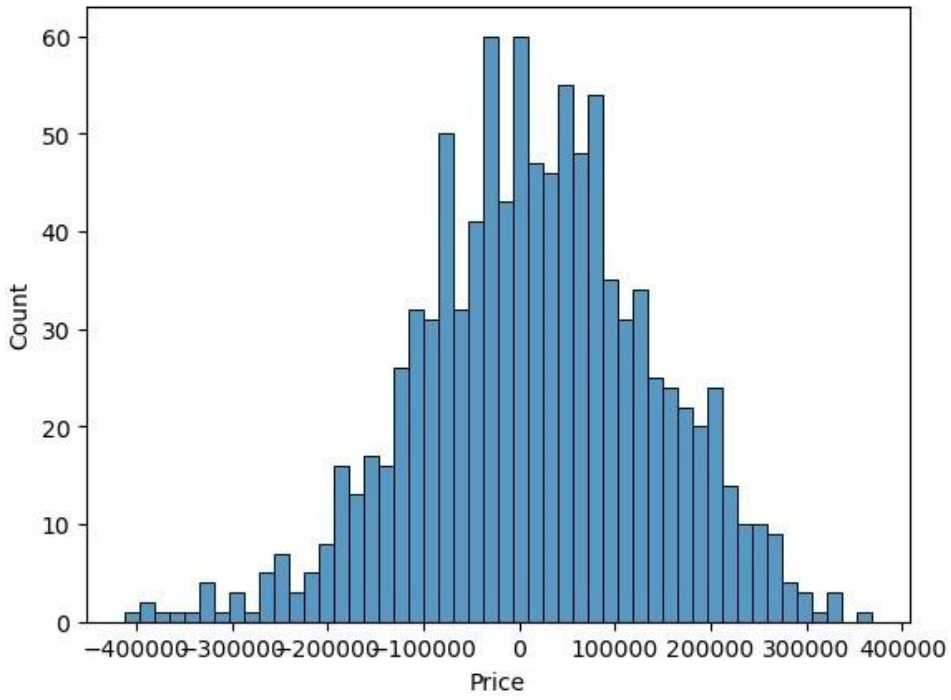


In [2]:

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

Out[2]:

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



In [3]:

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

Out[3]:

-0.0006222175925689744

286137.81086908665

128209033251.4034

Model Comparison:

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

In [4]:

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

Out[4]:

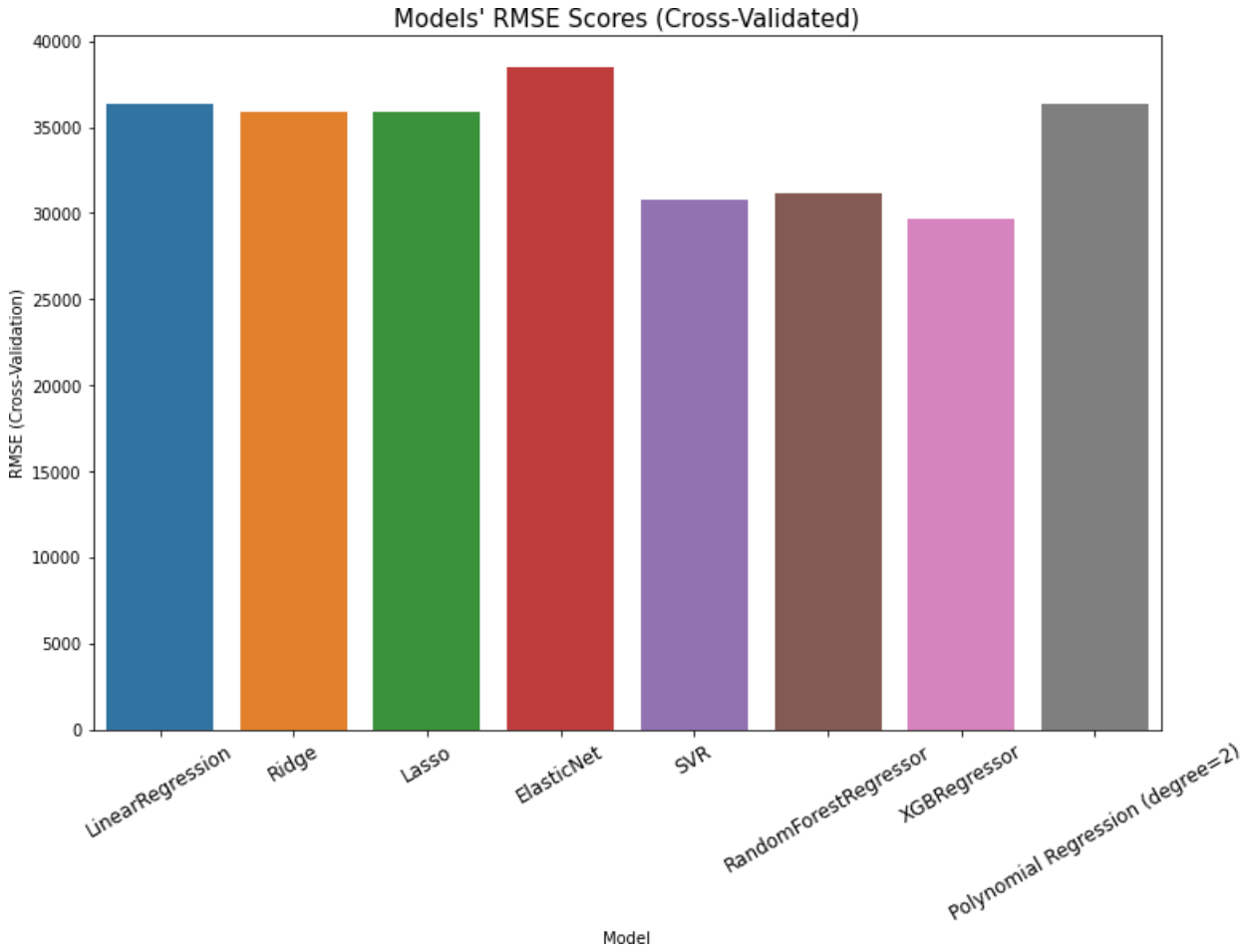
	Model	MAE	MSE	RMSE	R2 Score	RMSE (Cross-Validation)
6	XGBRegressor	1.743992e+04	7.165790e+08	2.676899e+04	9.065778e-01	29698.849618
4	SVR	1.784316e+04	1.132136e+09	3.364723e+04	8.524005e-01	30745.475239
5	RandomForestRegressor	1.811511e+04	1.004422e+09	3.169262e+04	8.690509e-01	31138.863315
1	Ridge	2.343550e+04	1.404264e+09	3.747351e+04	8.169225e-01	35887.852792

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

In [5]:

```
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()
```

Out[5]:



Features Engineering:

Feature engineering is a crucial step in stock price prediction. Here are some common features you can consider:

1. Technical Indicators: Calculate various indicators like Moving Averages, Relative Strength Index (RSI), MACD, Bollinger Bands, and Stochastic Oscillator. These can help capture price trends and momentum.

2. Fundamental Data: Include financial metrics like earnings, revenue, P/E ratios, and dividend yields. These can provide insights into a company's financial health.

3. Market Sentiment: Use sentiment analysis on news articles and social media data to gauge market sentiment. This can be valuable for short-term predictions.

4. Volume and Liquidity: Include trading volume and liquidity indicators. Sudden spikes or drops in volume can signal important market events.

5. Volatility: Measure historical price volatility, such as the standard deviation of returns, to assess risk.

6. Market Indexes: Consider the performance of broader market indexes like the S&P 500 or sector-specific indices. These can influence individual stock prices.

7. Seasonality: Account for seasonal patterns or anomalies in stock price movements, especially for companies affected by specific seasons or events.

8. Lagged Values: Include lagged values of the target variable (stock price) to capture autocorrelation in the data.

9. Economic Indicators: Incorporate economic indicators like GDP growth, unemployment rates, and interest rates. These can impact the overall market and, consequently, stock prices.

10. Feature Scaling: Normalize or standardize features to ensure they're on a similar scale, which is essential for some machine learning algorithms.

11. News and Events: Consider incorporating information about major news events, earnings reports, product launches, or mergers and acquisitions related to the company.

12. External Factors: Explore external factors like weather data (for certain industries), currency exchange rates (for multinational companies), and geopolitical events.

13. Social Media Trends: Analyze social media trends and sentiment related to the company or its products.

14. Correlations: Calculate and analyze the correlation between various features and the target stock price.

15. Feature Selection: Use techniques like feature selection or dimensionality reduction to identify the most relevant features.

Various feature to perform model training:

When building a model to predict stock price reductions, you can consider various features to train your model. These features can be broadly categorized into several types:

1. Historical Stock Data:
   - Previous stock prices (e.g., open, close, high, low).
   - Trading volume.
   - Price change percentages.
   - Moving averages (e.g., 50-day or 200-day moving averages).

2. Market Sentiment and News:
   - Sentiment analysis of news articles or social media related to the company or industry.
   - Economic indicators (e.g., GDP, inflation, unemployment rates).
   - Earnings reports and company announcements.

3. Technical Indicators:
   - Relative Strength Index (RSI).
   - Moving Average Convergence Divergence (MACD).
   - Bollinger Bands.
   - Stochastic Oscillator.

4. Fundamental Analysis:
   - Earnings per share (EPS).
   - Price-to-earnings ratio (P/E ratio).
   - Dividend yield.
   - Book value.

5. Market and Economic Indicators:
   - S&P 500 or other market indices.
   - Interest rates.
   - Exchange rates (if dealing with international stocks).
   - Consumer and producer price indices.

6. Volatility Measures:
   - Historical volatility.
   - Implied volatility (e.g., from options data).

7. Company-Specific Data:
   - Financial statements (e.g., balance sheet, income statement).
   - Business-specific metrics (e.g., sales growth, profit margins).
   - CEO or management changes.

8. Global Events:
   - Geopolitical events (e.g., elections, trade disputes).
   - Natural disasters and other global crises.

9. Technical Data:
   - Time of day or trading hours.
   - Trading volume patterns.
   - Price gaps and patterns.

10. External Factors:
    - Weather data (for certain industries like agriculture).
    - Legal or regulatory changes.

It's important to note that the relevance of features may vary depending on the stock, industry, and market conditions. Feature selection and engineering are critical steps in model development to identify which features have the most impact on predicting stock price reductions. Moreover, you should be cautious of overfitting and constantly validate your model's performance with appropriate evaluation metrics.

Conclusion:

- In the realm of stock price prediction, the process of feature engineering, model training, and model evaluation plays a pivotal role in making informed investment decisions. Through this comprehensive approach, we've endeavored to harness the power of data and machine learning to anticipate market trends.

- Feature engineering is the cornerstone of this endeavor. It involves crafting a rich set of features that encapsulate various aspects of the financial world, including historical stock data, technical indicators, fundamental metrics, and even market sentiment. These features, thoughtfully selected and engineered, provide the foundation upon which our predictive models are built.

- Model training, the next step, is where we employ machine learning algorithms to learn from historical data. The goal is to uncover patterns, correlations, and signals hidden within the features. We use techniques like regression, time series analysis, and deep learning to train our models. The success of this phase heavily relies on the quality of our feature engineering.

- The ultimate test of our models comes during the evaluation phase. We use various metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and others to assess the predictive accuracy of our models. Cross-validation is employed to ensure the robustness of our models in different market conditions.

- However, it's important to remember that stock price prediction is inherently challenging due to the complex and often unpredictable nature of financial markets. Past performance is not always indicative of future results. Therefore, it's crucial to approach these predictions with a healthy dose of caution.

- In conclusion, the process of stock price prediction is a multifaceted journey, from feature engineering to model training and evaluation. While it may not guarantee foolproof results, it equips us with valuable tools for making more informed investment decisions in an ever-fluctuating financial landscape. Careful selection of features, diligent model training, and rigorous evaluation can significantly enhance our ability to anticipate market movements, ultimately helping us navigate the world of finance more skillfully.