Stock Price Prediction System

Load the Data set

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error
```

```
df = pd.read_csv('/content/GOOGL.csv')
print(df.head())
```

```
         Date       Open       High        Low      Close  Adj Close    Volume
0  2004-08-19  50.050049  52.082081  48.028027  50.220219  50.220219  44659096
1  2004-08-20  50.555557  54.594597  50.300301  54.209209  54.209209  22834343
2  2004-08-23  55.430431  56.796799  54.579578  54.754753  54.754753  18256126
3  2004-08-24  55.675674  55.855858  51.836838  52.487488  52.487488  15247337
4  2004-08-25  52.532532  54.054054  51.991993  53.053055  53.053055   9188602
```

Basic info

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4431 entries, 0 to 4430
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       4431 non-null   object 
 1   Open       4431 non-null   float64
 2   High       4431 non-null   float64
 3   Low        4431 non-null   float64
 4   Close      4431 non-null   float64
 5   Adj Close  4431 non-null   float64
 6   Volume     4431 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 242.4+ KB
None
```

Descriptive statistics

```
print(df.describe())
```

```
              Open         High          Low        Close    Adj Close  \
count  4431.000000  4431.000000  4431.000000  4431.000000  4431.000000   
mean    693.087345   699.735595   686.078751   693.097367   693.097367   
std     645.118799   651.331215   638.579488   645.187806   645.187806   
min      49.644646    50.920921    48.028027    50.055054    50.055054   
25%     248.558563   250.853355   245.813309   248.415916   248.415916   
50%     434.924927   437.887878   432.687683   435.330322   435.330322   
75%    1007.364990  1020.649994   997.274994  1007.790008  1007.790008   
max    3025.000000  3030.929932  2977.979980  2996.770020  2996.770020   

             Volume  
count  4.431000e+03  
```

```
mean    6.444992e+06
std     7.690351e+06
min     4.656000e+05
25%     1.695600e+06
50%     3.778418e+06
75%     8.002390e+06
max     8.215117e+07
```

Check for missing values

```
print(df.isnull().sum())
```

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

Correlation matrix

```
# Convert the 'Date' column to datetime objects
df['Date'] = pd.to_datetime(df['Date'])

# Extract numerical features from the 'Date' column if needed
# For example, you can extract year, month, and day
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day

# Now you can calculate the correlation matrix
print(df.corr())
```

```
               Date      Open      High       Low     Close  Adj Close  \
Date       1.000000  0.851641  0.851575  0.852000  0.851839   0.851839
Open       0.851641  1.000000  0.999915  0.999902  0.999808   0.999808
High       0.851575  0.999915  1.000000  0.999877  0.999903   0.999903
Low        0.852000  0.999902  0.999877  1.000000  0.999914   0.999914
Close      0.851839  0.999808  0.999903  0.999914  1.000000   1.000000
Adj Close  0.851839  0.999808  0.999903  0.999914  1.000000   1.000000
Volume    -0.681369 -0.453884 -0.452855 -0.455447 -0.454252  -0.454252
Year       0.998389  0.849518  0.849495  0.849827  0.849719   0.849719
Month     -0.013969  0.001236  0.000488  0.002091  0.001184   0.001184
Day       -0.002353 -0.004238 -0.004361 -0.004138 -0.004388  -0.004388

             Volume      Year     Month       Day
Date      -0.681369  0.998389 -0.013969 -0.002353
Open      -0.453884  0.849518  0.001236 -0.004238
High      -0.452855  0.849495  0.000488 -0.004361
Low       -0.455447  0.849827  0.002091 -0.004138
Close     -0.454252  0.849719  0.001184 -0.004388
Adj Close -0.454252  0.849719  0.001184 -0.004388
Volume     1.000000 -0.676220 -0.063534  0.010951
Year      -0.676220  1.000000 -0.070479 -0.007213
Month     -0.063534 -0.070479  1.000000  0.003034
Day        0.010951 -0.007213  0.003034  1.000000
```

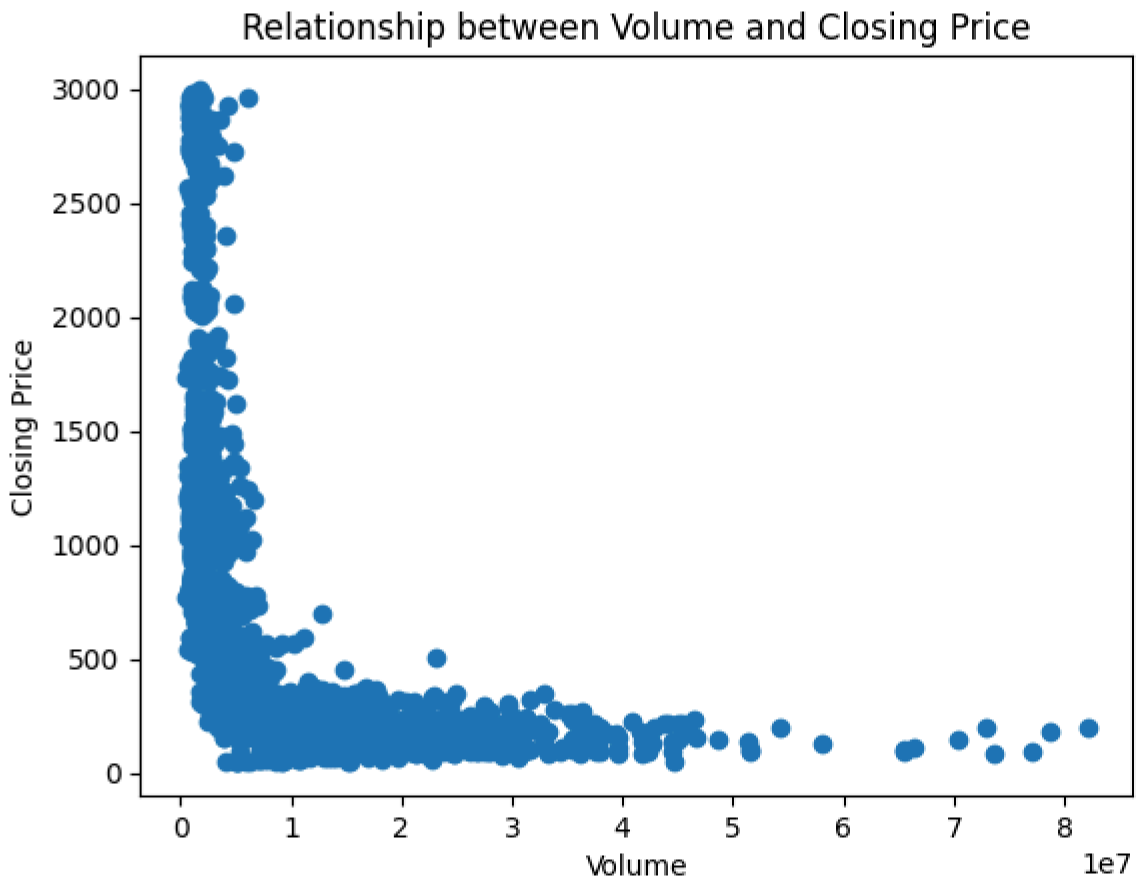
Analyze the distribution of target variable

```
plt.hist(df['Close'])
plt.xlabel('Closing Price')
plt.ylabel('Frequency')
plt.title('Distribution of Closing Prices')
plt.show()
```

Analyze the relationship between features

```
plt.scatter(df['Volume'], df['Close'])
plt.xlabel('Volume')
```

```
plt.ylabel('Closing Price')
plt.title('Relationship between Volume and Closing Price'
```



Analyze the correlation between the series and its lagged values.

Convert 'Date' column to datetime objects

```
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

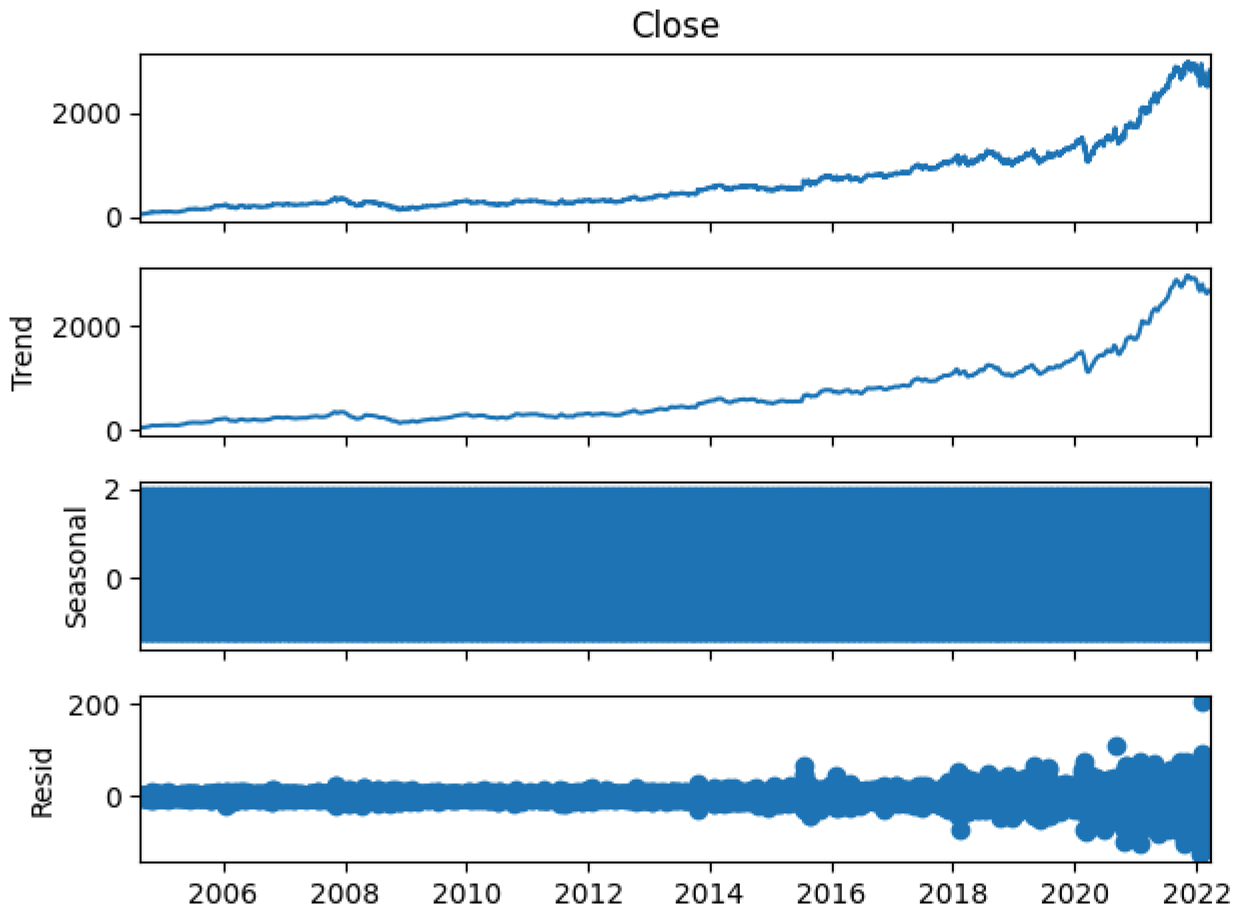
Decompose the time series

```
!pip install statsmodels
import statsmodels.api as sm
import pandas as pd

decomposition = sm.tsa.seasonal_decompose(df['Close'], model='additive', period=12)
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (0.14.3)
Requirement already satisfied: numpy<3,>=1.22.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.26.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.13.1
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (2.2.2
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (0.5.6)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas!=2.1.0,>=
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=2.1.0,>=1.4->stats
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas!=2.1.0,>=1.4->sta
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.6->statsmodels) (1.16.0)
```

Plot the decomposed components

```
fig = decomposition.plot()
plt.show()
```



Calculate moving averages

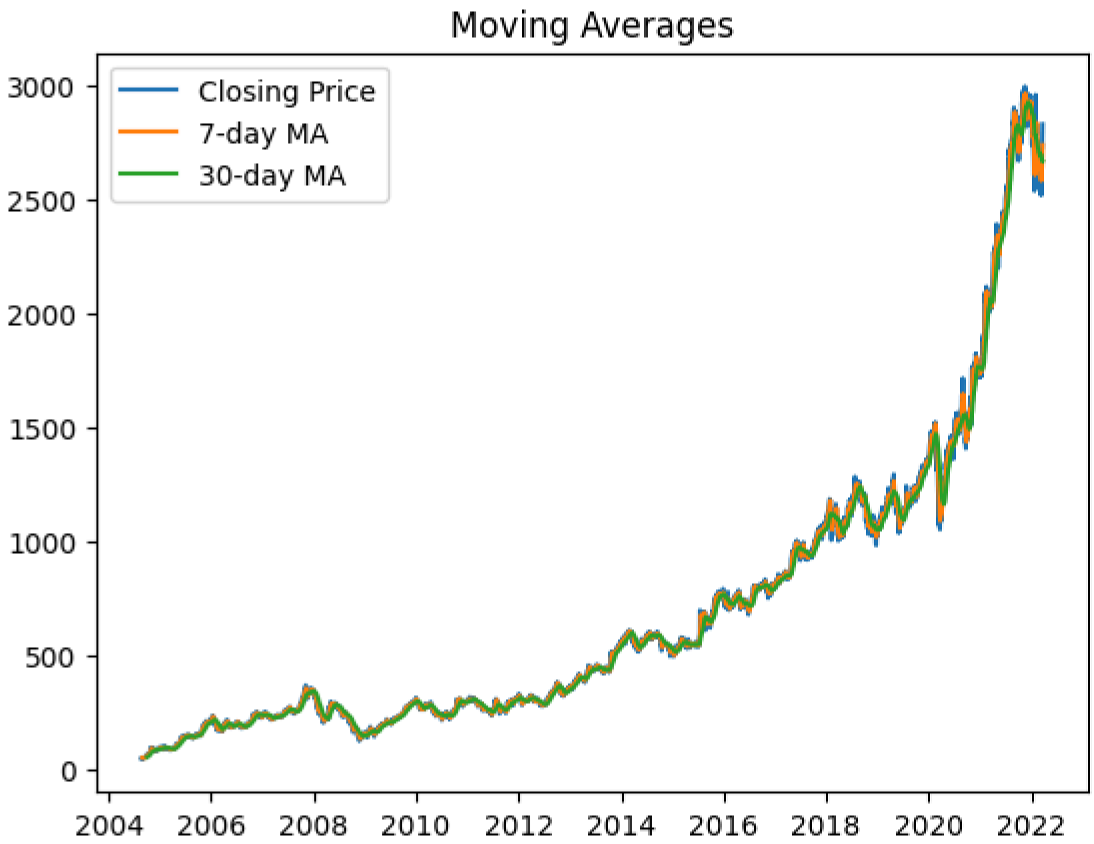
```
df['MA_7'] = df['Close'].rolling(window=7).mean()
df['MA_30'] = df['Close'].rolling(window=30).mean()

print(df['MA_30'].head())
```

```
Date
2004-08-19    NaN
2004-08-20    NaN
```

```
2004-08-23   NaN
2004-08-24   NaN
2004-08-25   NaN
Name: MA_30, dtype: float64
```

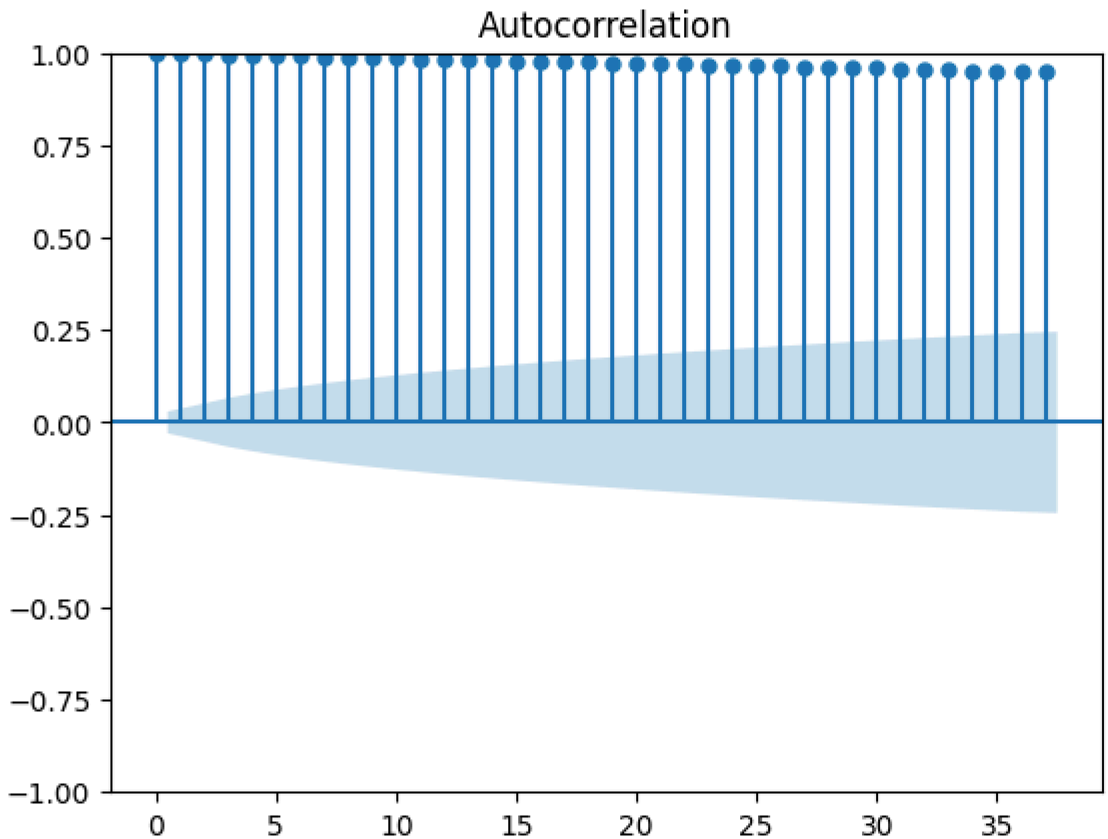
Plot moving averages

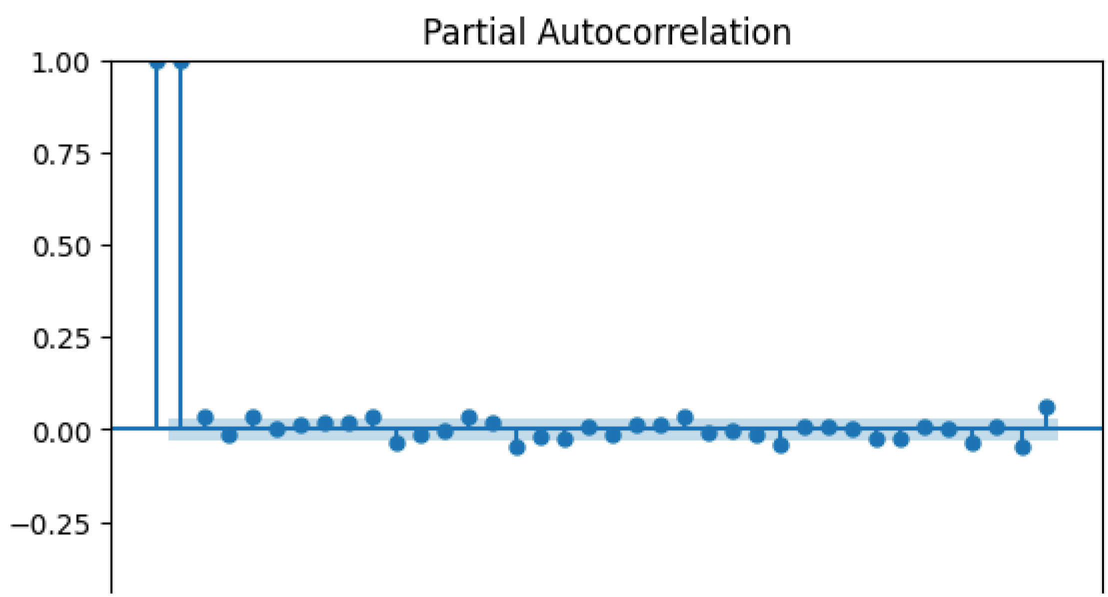
```
plt.plot(df['Close'], label='Closing Price')
plt.plot(df['MA_7'], label='7-day MA')
plt.plot(df['MA_30'], label='30-day MA')
plt.legend()
plt.title('Moving Averages')
plt.show()
```

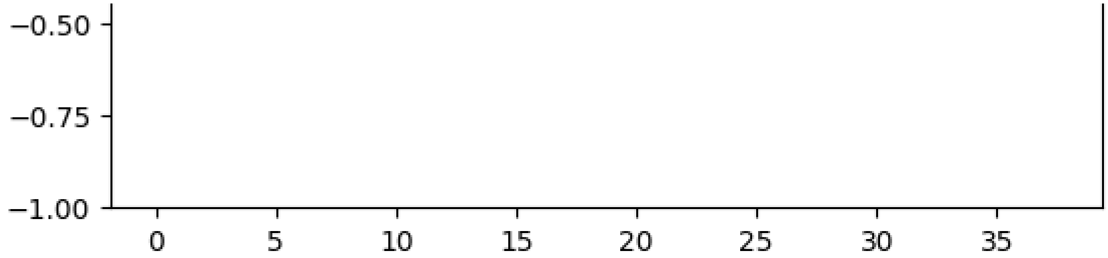


Calculate ACF and PACF

```
plot_acf(df['Close'])
plt.show()

plot_pacf(df['Close'])
plt.show()
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35


Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25



Feature Engineering

```
df['PriceDifference'] = df['Close'] - df['Open']
```

Handle missing values

```
df.fillna(method='ffill', inplace=True)
```

```
<ipython-input-30-e9443599d05e>:1: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a futu
  df.fillna(method='ffill', inplace=True)
```

Scaling (Example: Using MinMaxScaler for numerical features)

```
scaler = MinMaxScaler()
numerical_features = ['Open', 'High', 'Low', 'Close', 'Volume', 'PriceDifference']
df[numerical_features] = scaler.fit_transform(df[numerical_features])
print(df.head())
```

```
                    Open      High       Low     Close  Adj Close    Volume  Year  \
Date
```

```
2004-08-19  0.000136  0.000390  0.000000  0.000056  50.220219  0.541020  2004
2004-08-20  0.000306  0.001233  0.000776  0.001410  54.209209  0.273840  2004
2004-08-23  0.001945  0.001972  0.002236  0.001595  54.754753  0.217793  2004
2004-08-24  0.002027  0.001656  0.001300  0.000825  52.487488  0.180959  2004
2004-08-25  0.000971  0.001051  0.001353  0.001017  53.053055  0.106788  2004

            Month  Day  MA_7  MA_30  PriceDifference
Date
2004-08-19      8   19   NaN    NaN         0.462611
2004-08-20      8   20   NaN    NaN         0.474721
2004-08-23      8   23   NaN    NaN         0.459671
2004-08-24      8   24   NaN    NaN         0.450936
2004-08-25      8   25   NaN    NaN         0.463829
```

Creating rolling mean/std features

```
df['Close_RollingMean_7'] = df['Close'].rolling(window=7).mean()
df['Close_RollingStd_7'] = df['Close'].rolling(window=7).std()
```

Adding lagged features

```
df['Close_Lag1'] = df['Close'].shift(1)
df['Close_Lag2'] = df['Close'].shift(2)
```

Handle missing values created by rolling and lagged features

```
df.fillna(method='bfill', inplace=True)

print(df.head())
```

```
                Open      High       Low     Close  Adj Close    Volume  Year  \
Date
2004-08-19  0.000136  0.000390  0.000000  0.000056  50.220219  0.541020  2004
2004-08-20  0.000306  0.001233  0.000776  0.001410  54.209209  0.273840  2004
2004-08-23  0.001945  0.001972  0.002236  0.001595  54.754753  0.217793  2004
2004-08-24  0.002027  0.001656  0.001300  0.000825  52.487488  0.180959  2004
2004-08-25  0.000971  0.001051  0.001353  0.001017  53.053055  0.106788  2004

            Month  Day       MA_7      MA_30  PriceDifference  \
Date
2004-08-19      8   19  53.123123  55.474307         0.462611
2004-08-20      8   20  53.123123  55.474307         0.474721
2004-08-23      8   23  53.123123  55.474307         0.459671
2004-08-24      8   24  53.123123  55.474307         0.450936
2004-08-25      8   25  53.123123  55.474307         0.463829

            Close_RollingMean_7  Close_RollingStd_7  Close_Lag1  Close_Lag2
Date
2004-08-19             0.001041            0.000508    0.000056    0.000056
2004-08-20             0.001041            0.000508    0.000056    0.000056
2004-08-23             0.001041            0.000508    0.001410    0.000056
2004-08-24             0.001041            0.000508    0.001595    0.001410
2004-08-25             0.001041            0.000508    0.000825    0.001595
<ipython-input-38-2e7246199c50>:1: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a futu
  df.fillna(method='bfill', inplace=True)
```

Model Building:

Prepare data for LSTM

```
data = df[['Close']].values
train_data, test_data = train_test_split(data, test_size=0.2, shuffle=False

def create_dataset(dataset, look_back=1):
```

```
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

look_back = 10  # Number of previous time steps to consider
trainX, trainY = create_dataset(train_data, look_back)
testX, testY = create_dataset(test_data, look_back)
```

Reshape input to be [samples, time steps, features]

```
trainX = np.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = np.reshape(testX, (testX.shape[0], testX.shape[1], 1))
```

Build LSTM model

```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(look_back, 1)))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`inp
  super().__init__(**kwargs)
```

Train the model

```
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
```

```
Epoch 76/100
3533/3533 - 18s - 5ms/step - loss: 1.1584e-05
Epoch 77/100
3533/3533 - 19s - 5ms/step - loss: 1.1078e-05
Epoch 78/100
3533/3533 - 16s - 5ms/step - loss: 1.1579e-05
Epoch 79/100
3533/3533 - 17s - 5ms/step - loss: 1.1312e-05
Epoch 80/100
3533/3533 - 20s - 6ms/step - loss: 1.1707e-05
Epoch 81/100
3533/3533 - 21s - 6ms/step - loss: 1.1621e-05
Epoch 82/100
3533/3533 - 19s - 5ms/step - loss: 1.1353e-05
Epoch 83/100
3533/3533 - 16s - 5ms/step - loss: 1.1389e-05
Epoch 84/100
```

Make predictions

```
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```

```
111/111 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

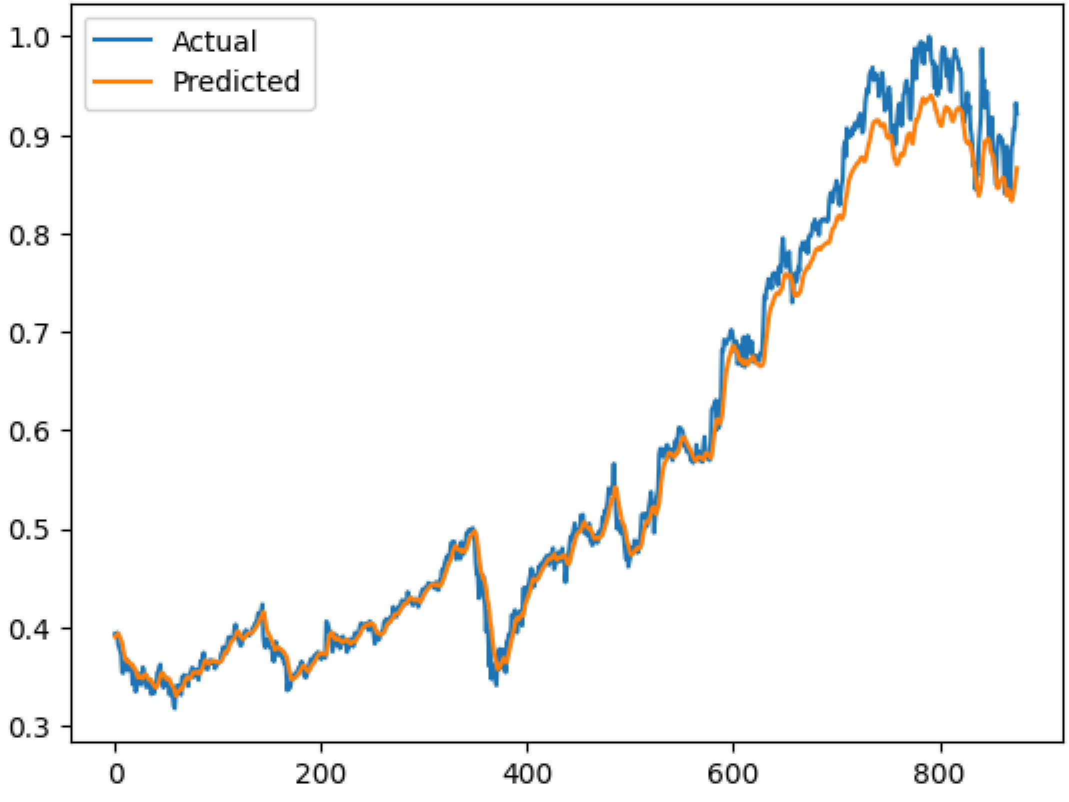
Calculate root mean squared error

```
trainScore = np.sqrt(mean_squared_error(trainY, trainPredict))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = np.sqrt(mean_squared_error(testY, testPredict))
print('Test Score: %.2f RMSE' % (testScore))
```

```
Train Score: 0.00 RMSE
Test Score: 0.02 RMSE
```

Plot predictions vs actual

```
plt.plot(testY)
plt.plot(testPredict)
plt.legend(['Actual', 'Predicted'])
plt.show()
```



Real-Time Implementation

Install necessary libraries

```
!pip install flask

from flask import Flask, render_template
import pandas as pd
import numpy as np
from tensorflow.keras.models import load_model
```

```
Requirement already satisfied: flask in /usr/local/lib/python3.10/dist-packages (2.2.5)
Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.10/dist-packages (from flask) (3.0.4)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from flask) (3.1.4)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from flask) (2.2.0)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from flask) (8.1.7)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=3.0->flask) (2.
```

Load the trained LSTM model

```
import os
import h5py
from tensorflow.keras.models import load_model

# Get the current working directory
print(os.getcwd())

if os.path.exists('stock_prediction_model.h5'):
    model = load_model('stock_prediction_model.h5')
```

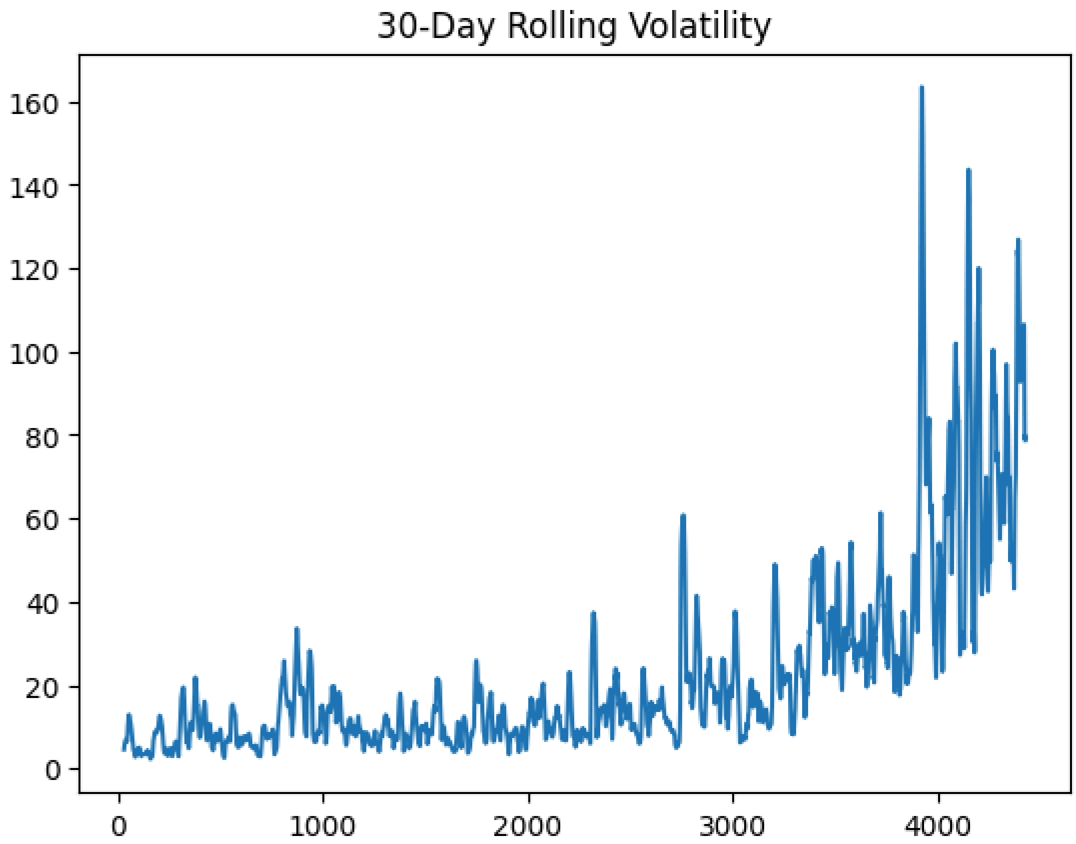
Function to get the latest stock data

```
def get_latest_stock_data():

    return {'Close': 1500}


app = Flask(__name__)

@app.route('/')
def index():
    latest_data = get_latest_stock_data()
    # Prepare input data for the model
    input_data = np.array([latest_data['Close']]).reshape(1, -1, 1)

    # Make a prediction
    prediction = model.predict(input_data)

    return render_template('index.html', prediction=prediction)


if __name__ == '__main__':
    app.run(debug=True)
```

Volatility Analysis:

Calculate rolling standard deviation to measure volatility

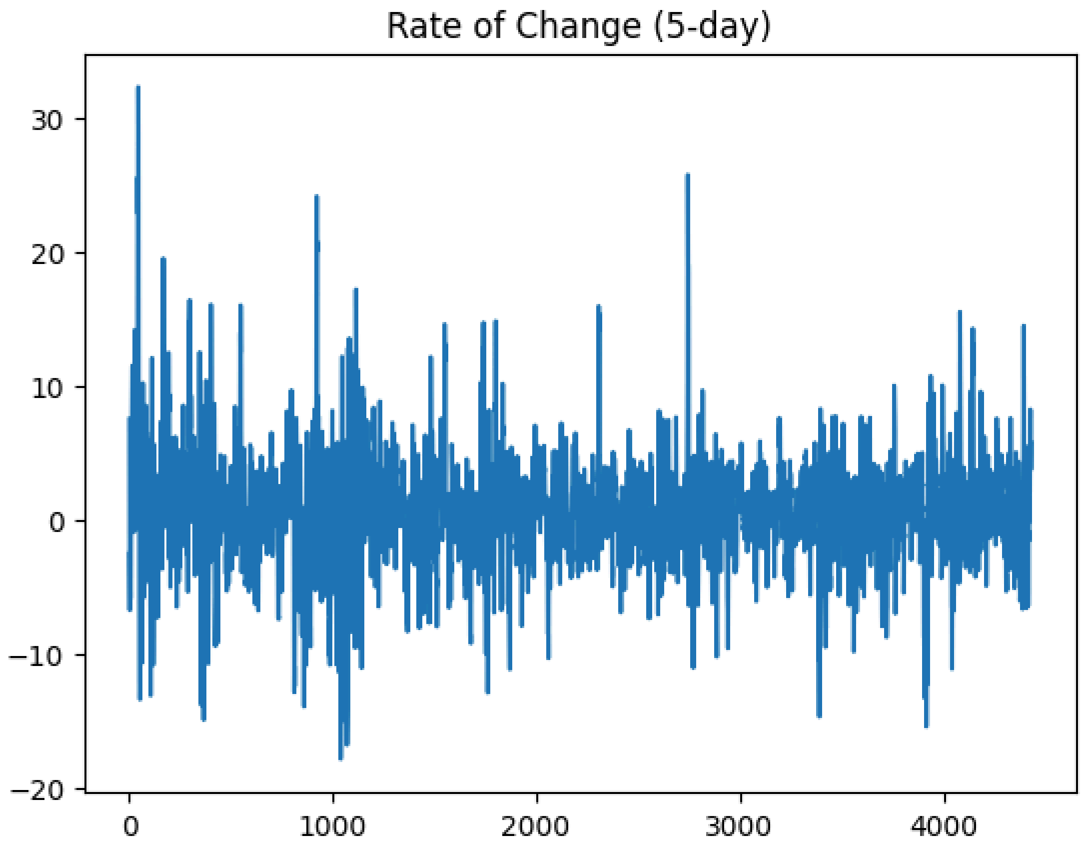
```
df['Close_RollingStd_30'] = df['Close'].rolling(window=30).std()
plt.plot(df['Close_RollingStd_30'])
plt.title('30-Day Rolling Volatility')
plt.show()
```



Rate of Change (ROC)

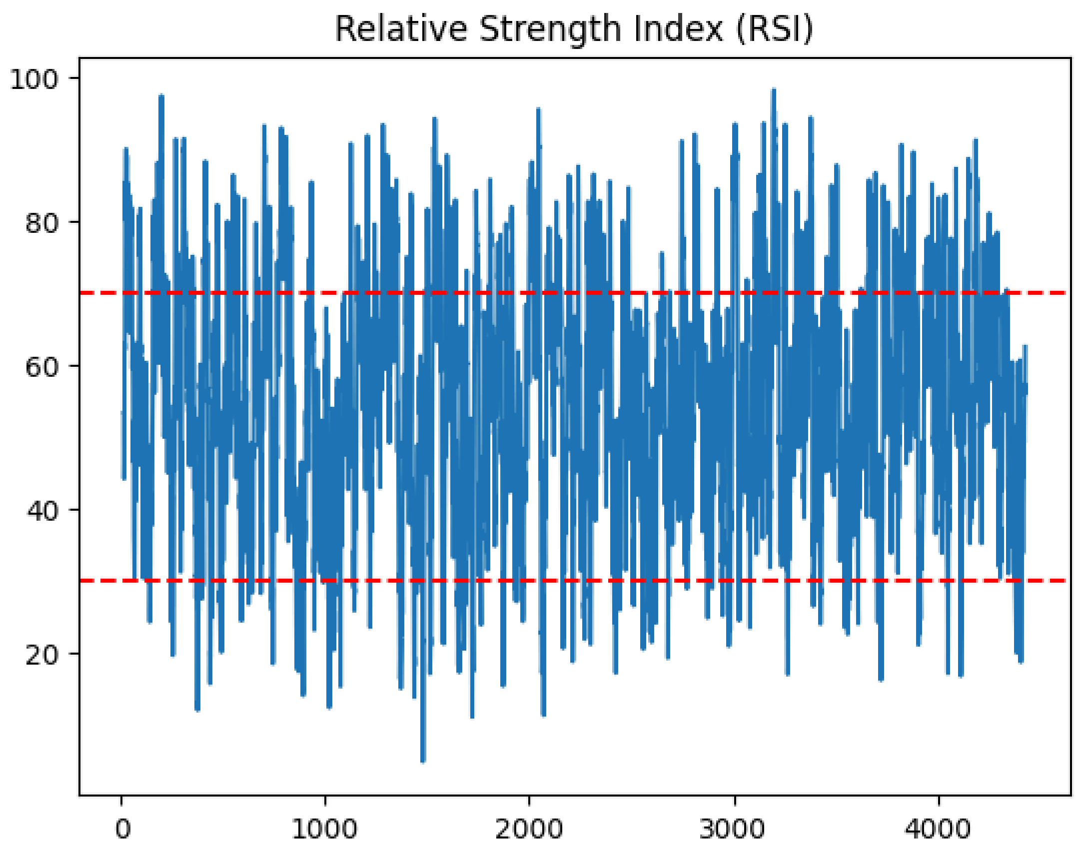
Calculate the percentage change in closing price over a specific period

```
df['ROC_5'] = (df['Close'] - df['Close'].shift(5)) / df['Close'].shift(5) * 100
plt.plot(df['ROC_5'])
plt.title('Rate of Change (5-day)')
plt.show()
```



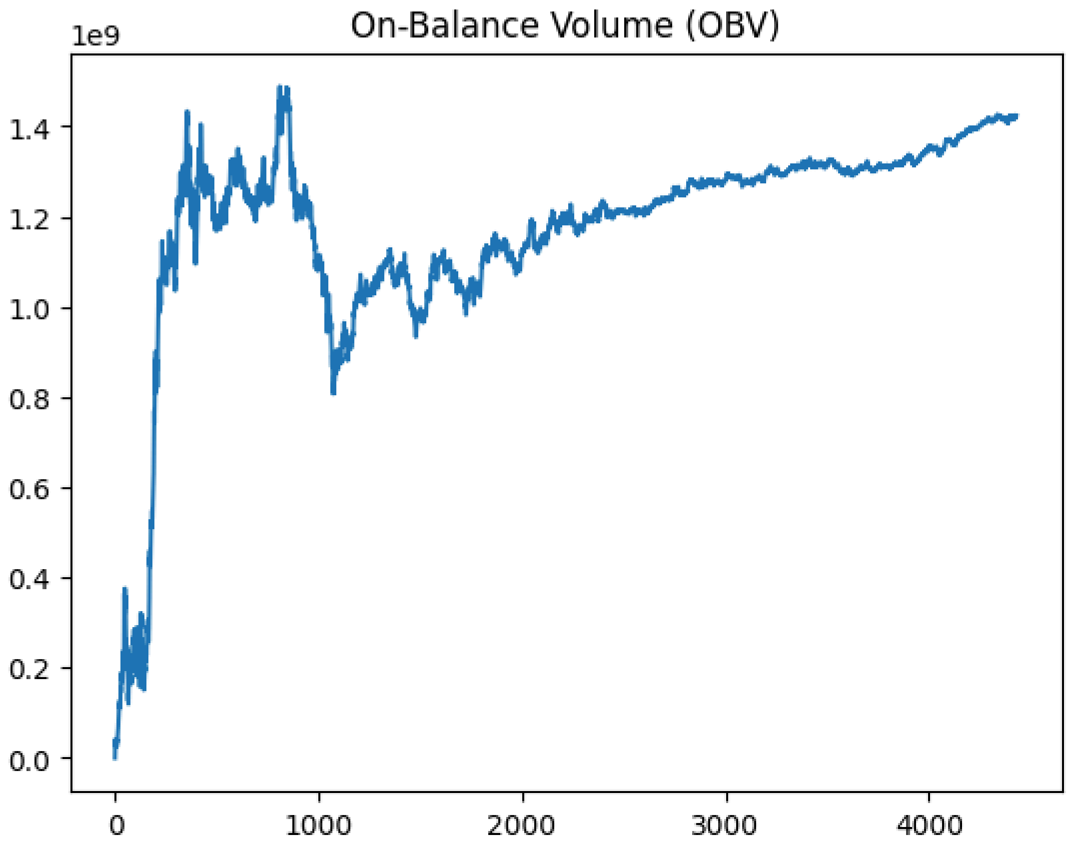
Relative Strength Index (RSI)

```
delta = df['Close'].diff()
gain = (delta.where(delta > 0, 0)).fillna(0)
loss = (-delta.where(delta < 0, 0)).fillna(0)
avg_gain = gain.rolling(window=14).mean()
avg_loss = loss.rolling(window=14).mean()
rs = avg_gain / avg_loss
df['RSI'] = 100 - (100 / (1 + rs))
plt.plot(df['RSI'])
plt.title('Relative Strength Index (RSI)')
plt.axhline(y=70, color='r', linestyle='--')
plt.axhline(y=30, color='r', linestyle='--')
plt.show()
```



On-Balance Volume (OBV)

```
df['OBV'] = np.where(df['Close'] > df['Close'].shift(1), df['Volume'],
                     np.where(df['Close'] < df['Close'].shift(1), -df['Volume'], 0)).cumsum()
plt.plot(df['OBV'])
plt.title('On-Balance Volume (OBV)')
plt.show()
```



```
df['MA_20'] = df['Close'].rolling(window=20).mean()
```