In [2]:
```
!pip install yfinance pandas_datareader
```

```
Requirement already satisfied: yfinance in c:\users\hp\anaconda3\lib\site-packages (0.2.37)
Requirement already satisfied: pandas_datareader in c:\users\hp\anaconda3\lib\site-packages (0.10.0)
Requirement already satisfied: pandas>=1.3.0 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (1.5.3)
Requirement already satisfied: numpy>=1.16.5 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (1.24.3)
Requirement already satisfied: requests>=2.31 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (4.9.2)
Requirement already satisfied: appdirs>=1.4.4 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (1.4.4)
Requirement already satisfied: pytz>=2022.5 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (2022.7)
Requirement already satisfied: frozendict>=2.3.4 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (2.4.0)
Requirement already satisfied: peewee>=3.16.2 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (3.17.1)
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (4.12.2)
Requirement already satisfied: html5lib>=1.1 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\hp\anaconda3\lib\site-packages (from beautifulsoup4>=4.11.1->yfinance) (2.4)
Requirement already satisfied: six>=1.9 in c:\users\hp\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in c:\users\hp\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\hp\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2023.5.7)
```

In [8]:
```
import yfinance as yf
!pip install yfinance
stock = yf.Ticker("AAPL")  # Using Apple's stock symbol
data = stock.history(period="1y")  # Fetching data for 1 year
```

```
Requirement already satisfied: yfinance in c:\users\hp\anaconda3\lib\site-packages (0.2.37)
Requirement already satisfied: pandas>=1.3.0 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (1.5.3)
Requirement already satisfied: numpy>=1.16.5 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (1.24.3)
Requirement already satisfied: requests>=2.31 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (4.9.2)
Requirement already satisfied: appdirs>=1.4.4 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (1.4.4)
Requirement already satisfied: pytz>=2022.5 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (2022.7)
Requirement already satisfied: frozendict>=2.3.4 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (2.4.0)
Requirement already satisfied: peewee>=3.16.2 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (3.17.1)
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (4.12.2)
Requirement already satisfied: html5lib>=1.1 in c:\users\hp\anaconda3\lib\site-packages (from yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\hp\anaconda3\lib\site-packages (from beautifulsoup4>=4.11.1->yfinance) (2.4)
Requirement already satisfied: six>=1.9 in c:\users\hp\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in c:\users\hp\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\hp\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\hp\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2023.5.7)
```

In [13]:
```
import pandas as pd
data=pd.read_csv('C:\\Users\\hp\\Downloads\\stockdata.csv')
```

In [14]:
```
data
```

Out[14]:

	Ticker	Date	Open	High	Low	Close	Adj Close	Volume
0	AAPL	07-02-2023	150.639999	155.229996	150.639999	154.649994	154.414230	83322600
1	AAPL	08-02-2023	153.880005	154.580002	151.169998	151.919998	151.688400	64120100
2	AAPL	09-02-2023	153.779999	154.330002	150.419998	150.869995	150.639999	56007100
3	AAPL	10-02-2023	149.460007	151.339996	149.220001	151.009995	151.009995	57450700
4	AAPL	13-02-2023	150.949997	154.259995	150.919998	153.850006	153.850006	62199000
...	...	...	...	...	...	...	...	...
243	GOOG	01-05-2023	107.720001	108.680000	107.500000	107.709999	107.709999	20926300
244	GOOG	02-05-2023	107.660004	107.730003	104.500000	105.980003	105.980003	20343100
245	GOOG	03-05-2023	106.220001	108.129997	105.620003	106.120003	106.120003	17116300
246	GOOG	04-05-2023	106.160004	106.300003	104.699997	105.209999	105.209999	19780600
247	GOOG	05-05-2023	105.320000	106.440002	104.738998	106.214996	106.214996	20705300

248 rows × 8 columns

In [15]:
```
data.head()
```

Out[15]:

	Ticker	Date	Open	High	Low	Close	Adj Close	Volume
0	AAPL	07-02-2023	150.639999	155.229996	150.639999	154.649994	154.414230	83322600
1	AAPL	08-02-2023	153.880005	154.580002	151.169998	151.919998	151.688400	64120100
2	AAPL	09-02-2023	153.779999	154.330002	150.419998	150.869995	150.639999	56007100
3	AAPL	10-02-2023	149.460007	151.339996	149.220001	151.009995	151.009995	57450700
4	AAPL	13-02-2023	150.949997	154.259995	150.919998	153.850006	153.850006	62199000

In [16]:
```
data.tail()
```

Out[16]:

	Ticker	Date	Open	High	Low	Close	Adj Close	Volume
243	GOOG	01-05-2023	107.720001	108.680000	107.500000	107.709999	107.709999	20926300
244	GOOG	02-05-2023	107.660004	107.730003	104.500000	105.980003	105.980003	20343100
245	GOOG	03-05-2023	106.220001	108.129997	105.620003	106.120003	106.120003	17116300
246	GOOG	04-05-2023	106.160004	106.300003	104.699997	105.209999	105.209999	19780600
247	GOOG	05-05-2023	105.320000	106.440002	104.738998	106.214996	106.214996	20705300

In [17]:
```
data.describe()
```

Out[17]:

	Open	High	Low	Close	Adj Close	Volume
count	248.000000	248.000000	248.000000	248.000000	248.000000	2.480000e+02
mean	215.252093	217.919662	212.697452	215.381674	215.362697	3.208210e+07
std	91.691315	92.863023	90.147881	91.461989	91.454750	2.233590e+07
min	89.540001	90.129997	88.860001	89.349998	89.349998	2.657900e+06
25%	135.235004	137.440004	134.822495	136.347498	136.347498	1.714180e+07
50%	208.764999	212.614998	208.184998	209.920006	209.920006	2.734000e+07
75%	304.177506	307.565002	295.437500	303.942505	303.942505	4.771772e+07
max	372.410004	373.829987	361.739990	366.829987	366.829987	1.133164e+08

In [21]:
```
data.dropna(inplace=True)  # This removes any rows with missing values
data
```

Out[21]:

	Ticker	Date	Open	High	Low	Close	Adj Close	Volume
0	AAPL	07-02-2023	150.639999	155.229996	150.639999	154.649994	154.414230	83322600
1	AAPL	08-02-2023	153.880005	154.580002	151.169998	151.919998	151.688400	64120100
2	AAPL	09-02-2023	153.779999	154.330002	150.419998	150.869995	150.639999	56007100
3	AAPL	10-02-2023	149.460007	151.339996	149.220001	151.009995	151.009995	57450700
4	AAPL	13-02-2023	150.949997	154.259995	150.919998	153.850006	153.850006	62199000
...	...	...	...	...	...	...	...	...
243	GOOG	01-05-2023	107.720001	108.680000	107.500000	107.709999	107.709999	20926300
244	GOOG	02-05-2023	107.660004	107.730003	104.500000	105.980003	105.980003	20343100
245	GOOG	03-05-2023	106.220001	108.129997	105.620003	106.120003	106.120003	17116300
246	GOOG	04-05-2023	106.160004	106.300003	104.699997	105.209999	105.209999	19780600
247	GOOG	05-05-2023	105.320000	106.440002	104.738998	106.214996	106.214996	20705300

248 rows × 8 columns

In [22]:
```
#checking -date format is correct
data.index = pd.to_datetime(data.index)
data
```

Out[22]:

	Ticker	Date	Open	High	Low	Close	Adj Close	Volume
1970-01-01 00:00:00.000000000	AAPL	07-02-2023	150.639999	155.229996	150.639999	154.649994	154.414230	83322600
1970-01-01 00:00:00.000000001	AAPL	08-02-2023	153.880005	154.580002	151.169998	151.919998	151.688400	64120100
1970-01-01 00:00:00.000000002	AAPL	09-02-2023	153.779999	154.330002	150.419998	150.869995	150.639999	56007100
1970-01-01 00:00:00.000000003	AAPL	10-02-2023	149.460007	151.339996	149.220001	151.009995	151.009995	57450700
1970-01-01 00:00:00.000000004	AAPL	13-02-2023	150.949997	154.259995	150.919998	153.850006	153.850006	62199000
...	...	...	...	...	...	...	...	...
1970-01-01 00:00:00.000000243	GOOG	01-05-2023	107.720001	108.680000	107.500000	107.709999	107.709999	20926300
1970-01-01 00:00:00.000000244	GOOG	02-05-2023	107.660004	107.730003	104.500000	105.980003	105.980003	20343100
1970-01-01 00:00:00.000000245	GOOG	03-05-2023	106.220001	108.129997	105.620003	106.120003	106.120003	17116300
1970-01-01 00:00:00.000000246	GOOG	04-05-2023	106.160004	106.300003	104.699997	105.209999	105.209999	19780600
1970-01-01 00:00:00.000000247	GOOG	05-05-2023	105.320000	106.440002	104.738998	106.214996	106.214996	20705300

248 rows × 8 columns

In [23]:
```
#calculating daily returns
data['Daily Return'] = data['Close'].pct_change()
data
```

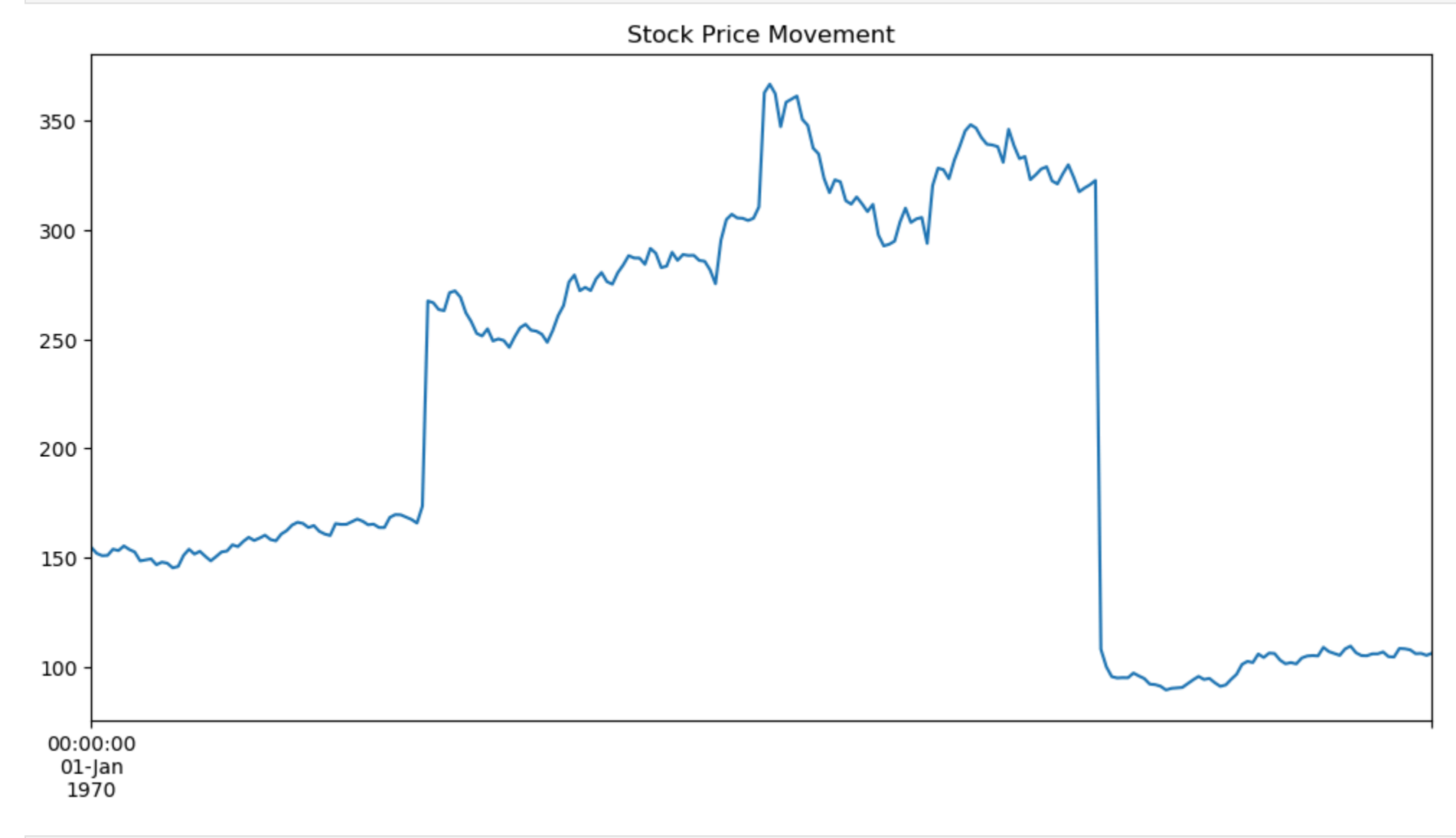
Out[23]:

	Ticker	Date	Open	High	Low	Close	Adj Close	Volume	Daily Return
1970-01-01 00:00:00.000000000	AAPL	07-02-2023	150.639999	155.229996	150.639999	154.649994	154.414230	83322600	NaN
1970-01-01 00:00:00.000000001	AAPL	08-02-2023	153.880005	154.580002	151.169998	151.919998	151.688400	64120100	-0.017653
1970-01-01 00:00:00.000000002	AAPL	09-02-2023	153.779999	154.330002	150.419998	150.869995	150.639999	56007100	-0.006912
1970-01-01 00:00:00.000000003	AAPL	10-02-2023	149.460007	151.339996	149.220001	151.009995	151.009995	57450700	0.000928
1970-01-01 00:00:00.000000004	AAPL	13-02-2023	150.949997	154.259995	150.919998	153.850006	153.850006	62199000	0.018807
...	...	...	...	...	...	...	...	...	...
1970-01-01 00:00:00.000000243	GOOG	01-05-2023	107.720001	108.680000	107.500000	107.709999	107.709999	20926300	-0.004713
1970-01-01 00:00:00.000000244	GOOG	02-05-2023	107.660004	107.730003	104.500000	105.980003	105.980003	20343100	-0.016062
1970-01-01 00:00:00.000000245	GOOG	03-05-2023	106.220001	108.129997	105.620003	106.120003	106.120003	17116300	0.001321
1970-01-01 00:00:00.000000246	GOOG	04-05-2023	106.160004	106.300003	104.699997	105.209999	105.209999	19780600	-0.008575
1970-01-01 00:00:00.000000247	GOOG	05-05-2023	105.320000	106.440002	104.738998	106.214996	106.214996	20705300	0.009552

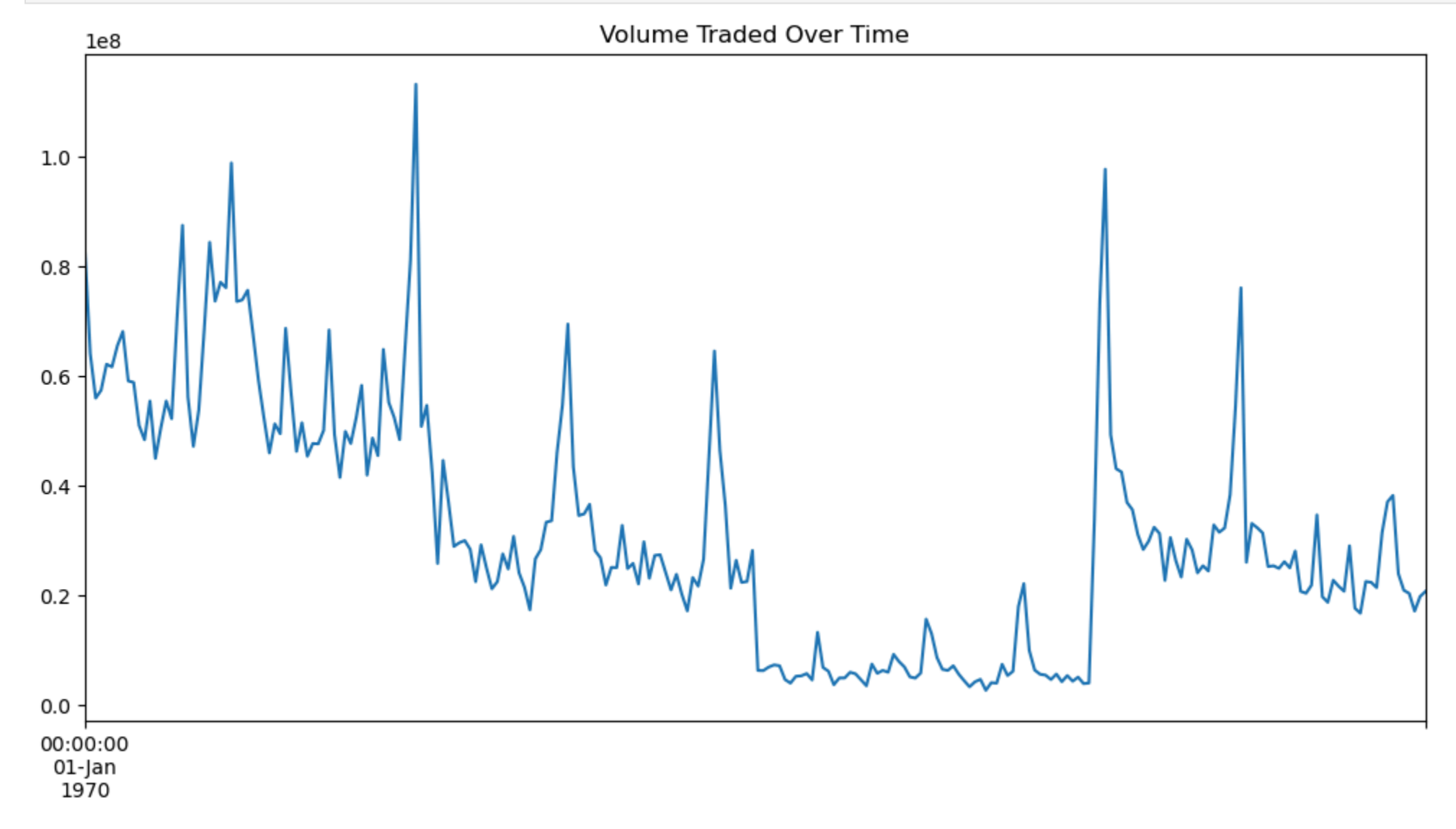
248 rows × 9 columns

In [ ]:
```
# Exploratory Data Analysis (EDA)
# EDA helps us understand the nature and structure of our data. It's the first step in identifying trends or anomalies.


# Visualize stock price movements:

# A simple line plot can show us how the stock's closing price has moved over time.
```

In [24]:
```
import matplotlib.pyplot as plt

data['Close'].plot(figsize=(12, 6), title="Stock Price Movement")
plt.show()
```

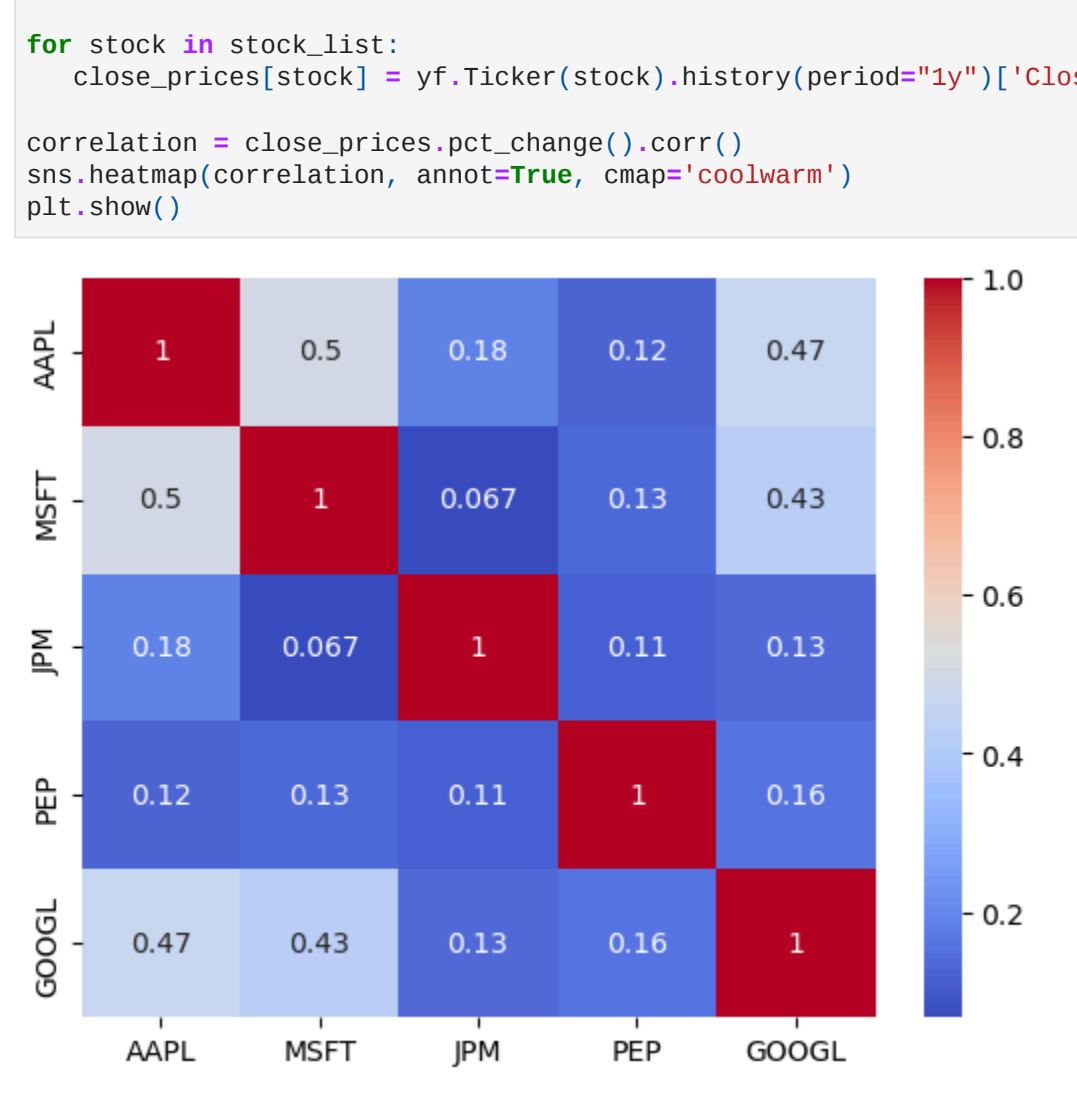


In [25]:
```
# Analyze volume of stocks traded:
# Volume indicates the number of shares traded in a given period. High volume can suggest significant news or events affecting the stock.
data['Volume'].plot(figsize=(12, 6), title="Volume Traded Over Time")
plt.show()
```



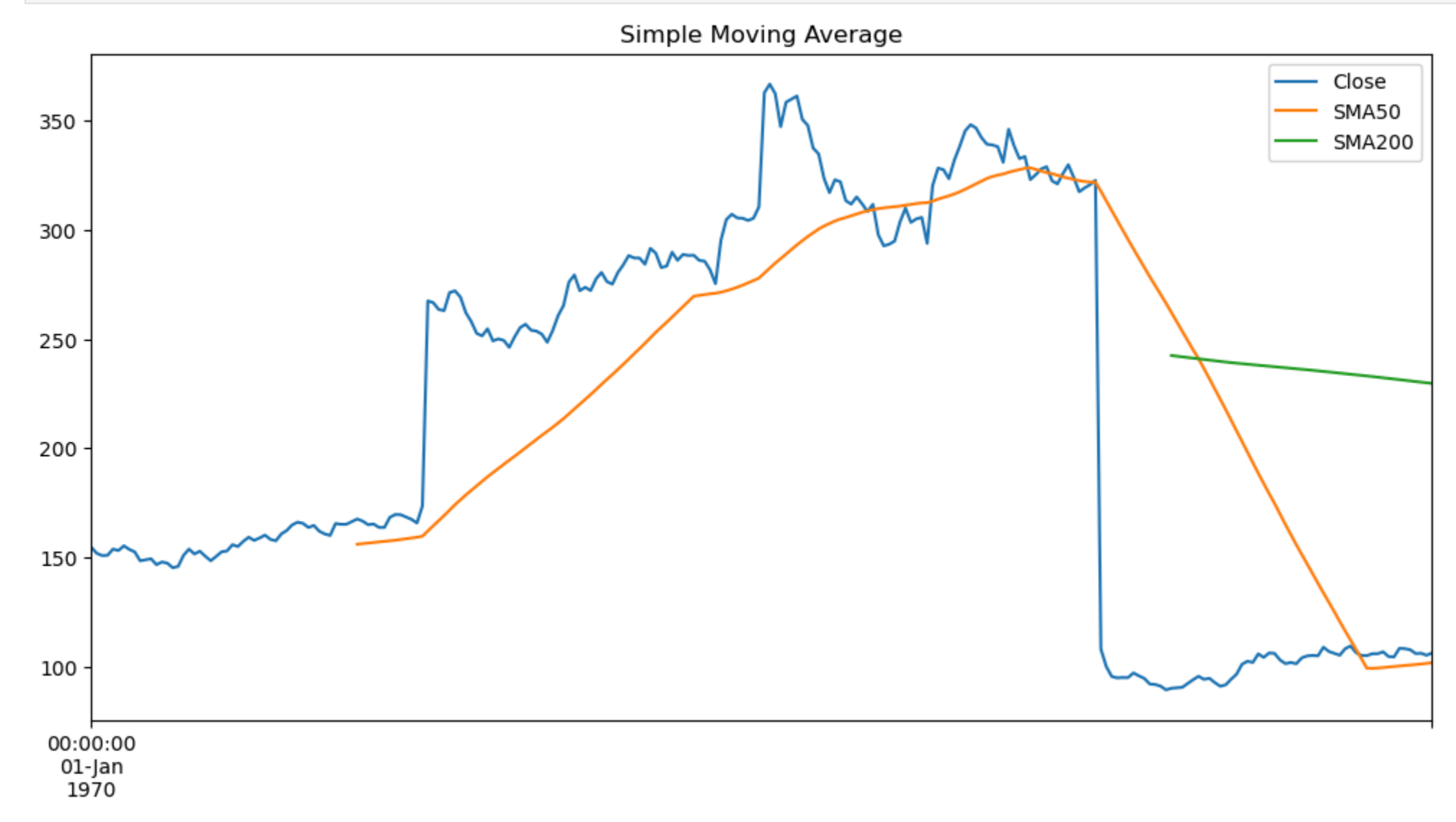
In [ ]:
```
# Correlation between stocks:


# When analyzing multiple stocks, it's essential to see how they move in relation to one another. A heatmap can visualize this relationship.
```

In [27]:
```
import seaborn as sns

stock_list = ["AAPL", "MSFT", "JPM", "PEP", "GOOGL"]
close_prices = pd.DataFrame()

for stock in stock_list:
   close_prices[stock] = yf.Ticker(stock).history(period="1y")['Close']

correlation = close_prices.pct_change().corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.show()
```

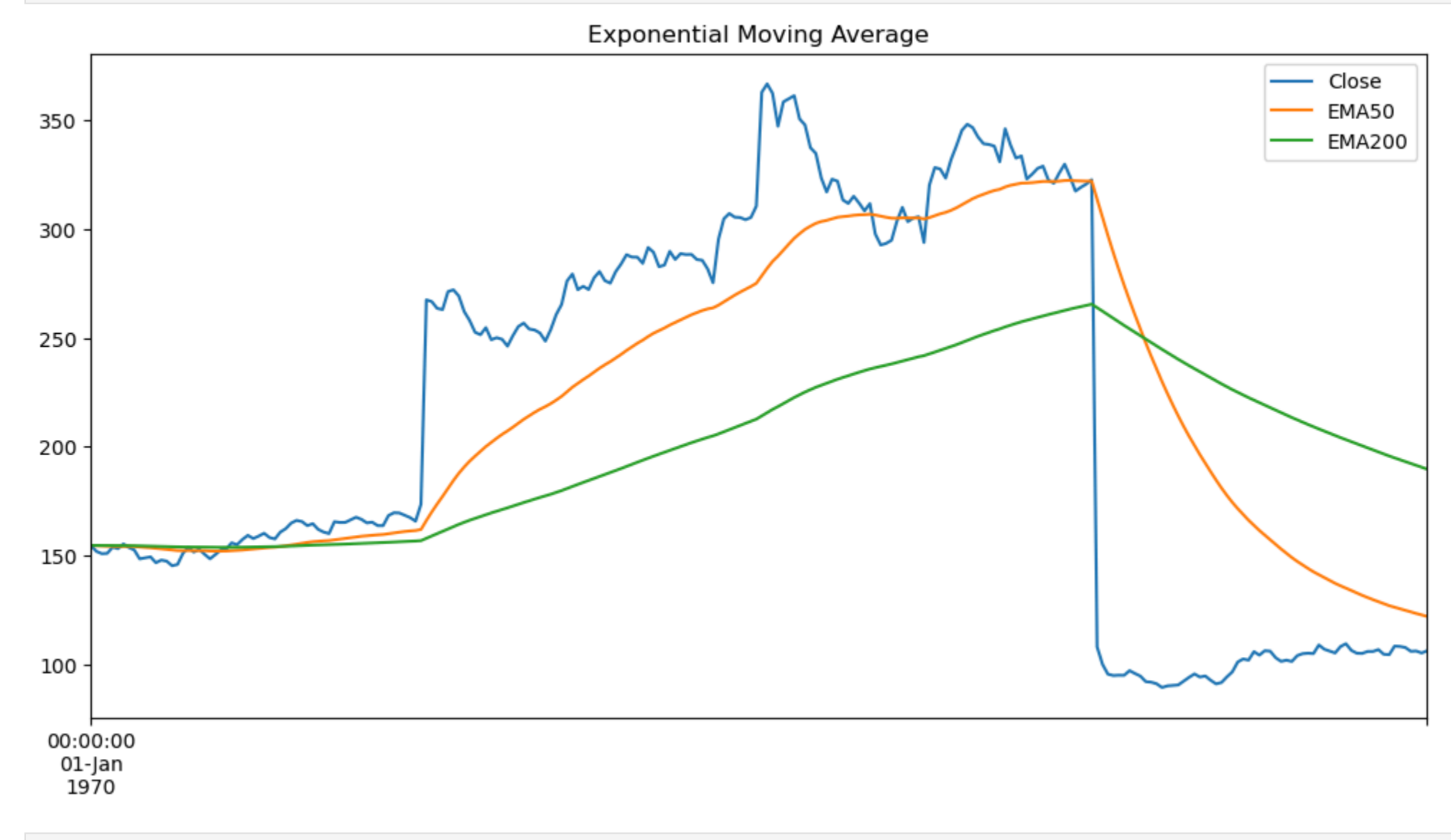


In [ ]:
```
# Technical Analysis
# Technical analysis involves studying past market data, primarily price and volume, to forecast future price movements. This analysis can be performed on any security with historic


# Moving Averages:

# A moving average smoothens price data to create a single flowing line, which makes it easier to identify the direction of the trend. The two most common types of moving averages a
```

In [28]:
```
# Simple moving average
data['SMA50'] = data['Close'].rolling(window=50).mean()
data['SMA200'] = data['Close'].rolling(window=200).mean()

data[['Close', 'SMA50', 'SMA200']].plot(figsize=(12,6))
plt.title('Simple Moving Average')
plt.show()
```

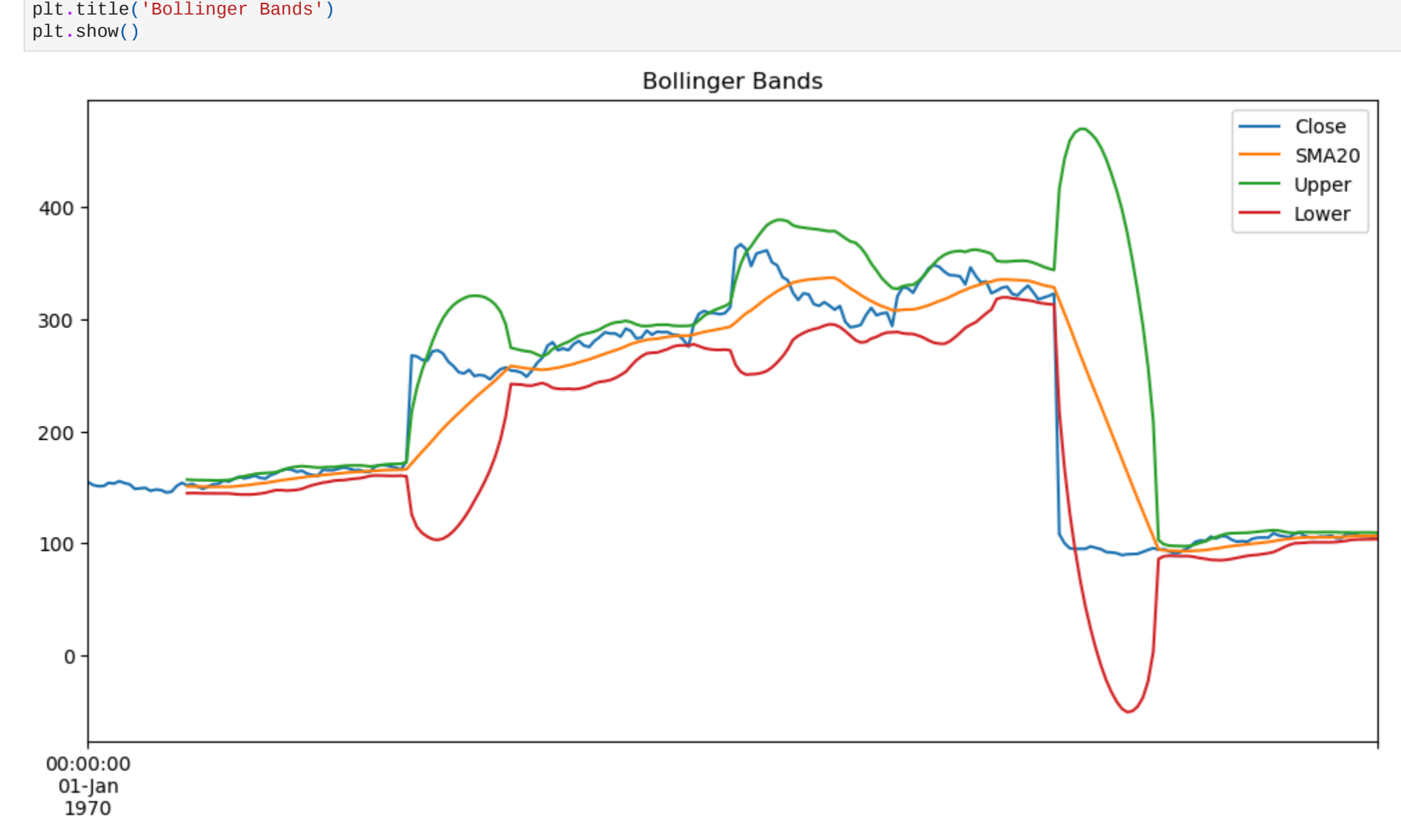


In [29]:
```
# Exponential Moving Average (EMA)
data['EMA50'] = data['Close'].ewm(span=50, adjust=False).mean()
data['EMA200'] = data['Close'].ewm(span=200, adjust=False).mean()

data[['Close', 'EMA50', 'EMA200']].plot(figsize=(12,6))
plt.title('Exponential Moving Average')
plt.show()
```



In [ ]:
```
# Bollinger Bands:


# Bollinger Bands consist of a middle band being an N-period simple moving average (SMA), an upper band at K times an N-period standard deviation above the middle band, and a lower
```

In [31]:
```
data['SMA20'] = data['Close'].rolling(window=20).mean()
data['Upper'] = data['SMA20'] + 2*data['Close'].rolling(window=20).std()
data['Lower'] = data['SMA20'] - 2*data['Close'].rolling(window=20).std()

data[['Close', 'SMA20', 'Upper', 'Lower']].plot(figsize=(12,6))
plt.title('Bollinger Bands')
plt.show()
```



In [36]:
```
from scipy.optimize import minimize

# Assuming stock_list contains the symbols of the stocks in the portfolio
returns = close_prices.pct_change()
mean_returns = returns.mean()
cov_matrix = returns.cov()
num_portfolios = 10000
risk_free_rate = 0.0178 # example value

# Portfolio optimization function
def portfolio_annualised_performance(weights, mean_returns, cov_matrix):
   returns = np.sum(mean_returns*weights ) *252
   std = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights))) * np.sqrt(252)
   return std, returns

# Minimize negative Sharpe Ratio to get optimal portfolio
def neg_sharpe_ratio(weights, mean_returns, cov_matrix, risk_free_rate):
   p_var, p_ret = portfolio_annualised_performance(weights, mean_returns, cov_matrix)
   return -(p_ret - risk_free_rate) / p_var

# Constraints for optimization
constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})

# Running the optimization
initial = [1./len(stock_list) for stock in stock_list]
bounds = tuple((0, 1) for asset in range(len(stock_list)))
result = minimize(neg_sharpe_ratio, initial, args=(mean_returns, cov_matrix, risk_free_rate), bounds=bounds, constraints=constraints)
```

In [ ]:
```
# Conclusion


# Summarize the insights gathered from the analysis:


# Stock price trends over the past year.
# Technical indicators and their implications.
# Potential investment strategies based on historical data.
# Performance of the developed trading strategy.
```