Name: Siddharth Singh

Net_ID: sms10221

```
!pip install alpha_vantage
```

```
Requirement already satisfied: alpha_vantage in /usr/local/lib/python3.10/dist-packages (2.3.1)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from alpha_vantage) (3.9.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from alpha_vantage) (2.31.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->alpha_vantage) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->alpha_vantage) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->alpha_vantage) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->alpha_vantage) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->alpha_vantage) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->alpha_vantage) (4.0.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->alpha_vantage) (3.3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->alpha_vantage) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->alpha_vantage) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->alpha_vantage) (2024.2.2)
```

Setting up API and testing Data

```
from alpha_vantage.timeseries import TimeSeries
import pandas as pd

# Initialize with your API key
ts = TimeSeries(key='Your_key', output_format='pandas')

# Get stock data for a specific stock, e.g., 'META'
data, meta_data = ts.get_daily(symbol='META', outputsize='full')
stock_prices = data['4. close']  # Close price column
stock_prices = stock_prices.astype(float)
```

```
# Get stock data for 'META' (META.)
data, meta_data = ts.get_daily(symbol='META', outputsize='full')
print(data.head())
```

```
            1. open  2. high  3. low  4. close   5. volume
date
2024-04-26  441.460   446.44  431.96    443.29  32691443.0
2024-04-25  421.400   445.77  414.50    441.38  82890741.0
2024-04-24  508.060   510.00  484.58    493.50  37772677.0
2024-04-23  491.250   498.76  488.97    496.10  15079196.0
2024-04-22  489.715   492.01  473.40    481.73  17271125.0
```

ARIMA Model Setup and Forecast:

The initial phase of the project employed the Autoregressive Integrated Moving Average (ARIMA) model as a baseline for forecasting stock prices. ARIMA is a widely recognized statistical method designed to analyze and predict time-series data, making it highly applicable to financial markets where data points are sequential and time-dependent.

Rationale for Choosing ARIMA:

1. Handling Non-Stationarity
2. Flexibility in Modeling
3. Simplicity and Efficiency

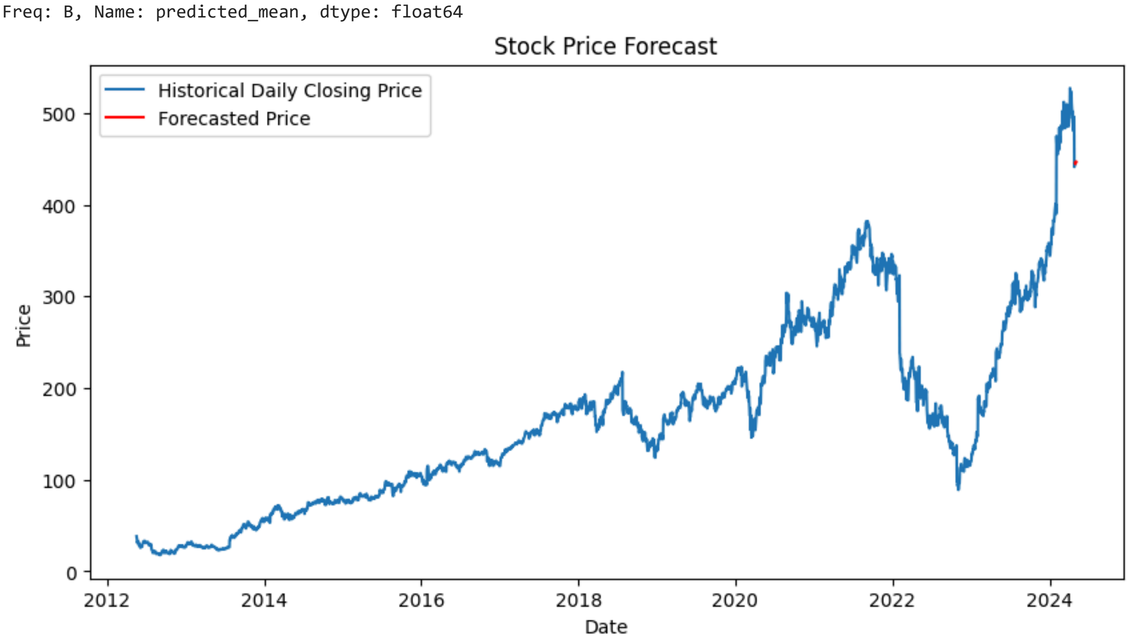
```
!pip install statsmodels
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (0.14.2)
Requirement already satisfied: numpy>=1.22.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.25.2)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.11.4)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (2.0.3)
```

```
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) (24.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.6->statsmodels) (1.16.0)
```

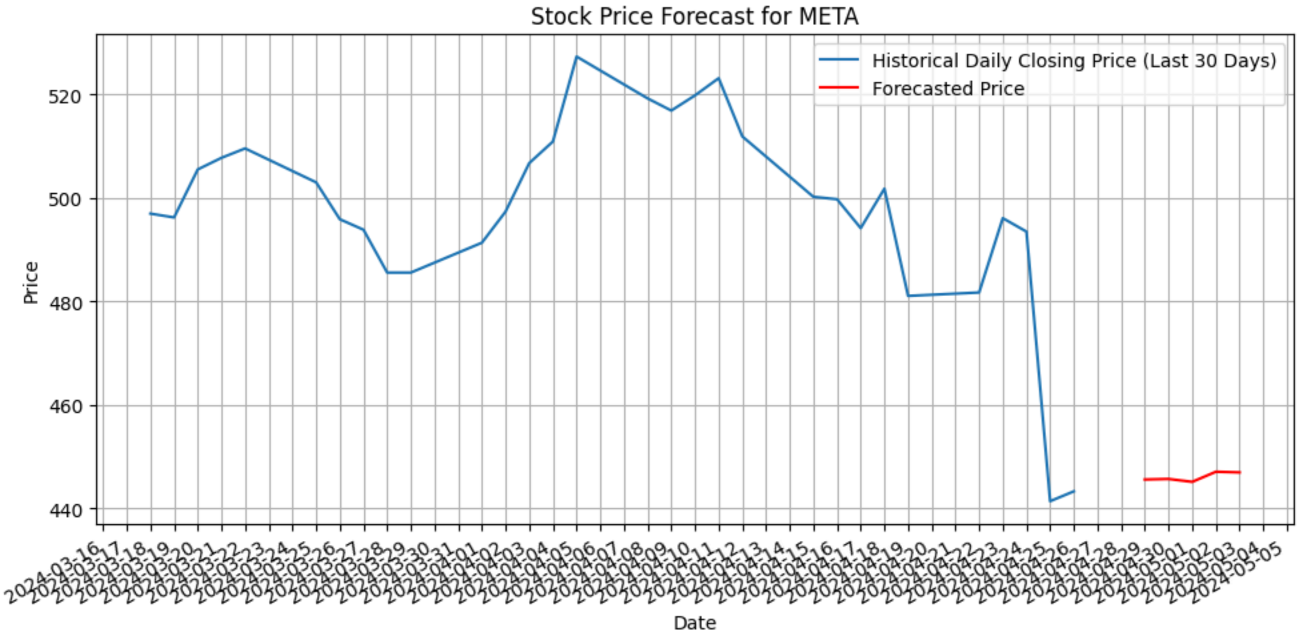
```python
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
# Get stock data for a specific stock, e.g., 'META'
data, meta_data = ts.get_daily(symbol='META', outputsize='full')
stock_prices = data['4. close']  # Close price column
stock_prices = stock_prices.astype(float)

# Sort the data by date (if not already sorted)
stock_prices.sort_index(inplace=True)

# Set the frequency of the data - assuming daily data excluding weekends
stock_prices.index = pd.to_datetime(stock_prices.index)
stock_prices = stock_prices.asfreq('B')  # 'B' denotes business day frequency

# Filling any missing values that might appear after setting the frequency
stock_prices.fillna(method='ffill', inplace=True)  # Forward fill

# Define and fit the ARIMA model
model = ARIMA(stock_prices, order=(5, 1, 0))  # Adjust these parameters as needed
fitted_model = model.fit()

# Forecasting the next 5 business days
forecast = fitted_model.forecast(steps=5)
print(forecast)

# Plotting the results
plt.figure(figsize=(10,5))
plt.plot(stock_prices.index, stock_prices, label='Historical Daily Closing Price')
plt.plot(forecast.index, forecast, color='red', label='Forecasted Price')
plt.title('Stock Price Forecast')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
2024-04-29    445.579791
2024-04-30    445.681594
2024-05-01    445.120655
2024-05-02    447.075520
2024-05-03    446.952670
Freq: B, Name: predicted_mean, dtype: float64
```



```
symbol='META'
# Get the full stock data for a specific stock, e.g., 'META'
data, meta_data = ts.get_daily(symbol=symbol, outputsize='full')

# The '4. close' column has the closing prices
stock_prices = data['4. close'].iloc[::-1]  # Reverse the order to have the oldest prices first

# Ensure the date index is a datetime type and sort it
stock_prices.index = pd.to_datetime(stock_prices.index)
stock_prices.sort_index(inplace=True)

# Set the frequency of the data to business days and fill any missing values
stock_prices = stock_prices.asfreq('B', method='ffill')

# Take the last 30 days for the plot
last_30_days_prices = stock_prices.last('30B')

# Define and fit the ARIMA model on the full dataset
model = ARIMA(stock_prices, order=(5, 1, 0))  # The order may need to be adjusted based on model diagnostics
fitted_model = model.fit()

# Forecast the next 5 business days
forecast = fitted_model.forecast(steps=5)
print(forecast)

# Preparing the dates for the forecast
last_date = stock_prices.index[-1]
forecast_dates = pd.date_range(start=last_date, periods=6, freq='B')[1:]  # exclude the last date of the known data

# Plotting the results
plt.figure(figsize=(10,5))
plt.plot(last_30_days_prices.index, last_30_days_prices, label='Historical Daily Closing Price (Last 30 Days)')
plt.plot(forecast_dates, forecast, color='red', label='Forecasted Price')

# Formatting the plot
plt.title(f'Stock Price Forecast for {symbol}')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)

# Setting x-axis major locator and formatter for better date display
plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gcf().autofmt_xdate()  # Auto rotate date labels

plt.tight_layout()
plt.show()
```

```
2024-04-29    445.579791
2024-04-30    445.681594
2024-05-01    445.120655
2024-05-02    447.075520
2024-05-03    446.952670
Freq: B, Name: predicted_mean, dtype: float64
```



Upon implementing the ARIMA model, forecasted stock prices were generated, providing a benchmark against which the performance of more complex models could be assessed. This step was crucial for establishing a foundational understanding of how well traditional time-series analysis could perform in predicting stock prices before integrating more sophisticated methods and additional variables such as ESG factors.

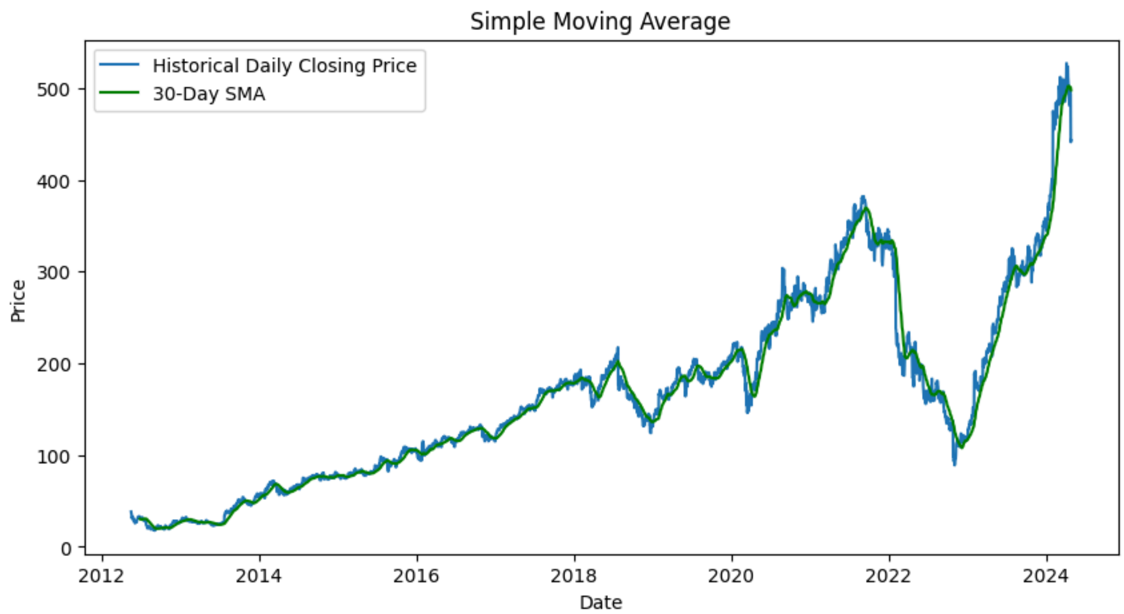
Simple Moving Average (SMA):

The Simple Moving Average (SMA) model is an elementary yet powerful tool used in time series forecasting, particularly in stock price analysis. It calculates the average stock price over a specified number of time periods, sliding forward with each new period.

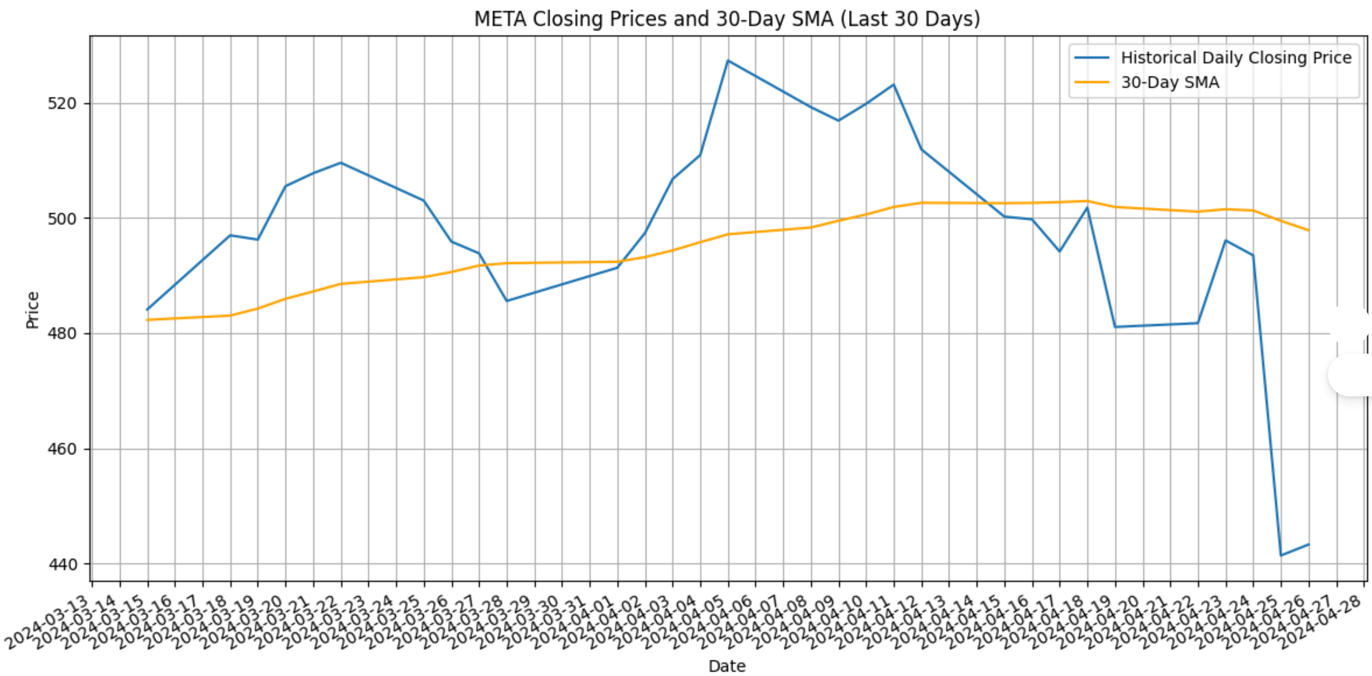
Rationale for Choosing SMA:

1. Trend Identification
2. Simplicity

```
# Calculate the 30-day simple moving average
sma_30 = stock_prices.rolling(window=30).mean()

# Plotting SMA
plt.figure(figsize=(10,5))
plt.plot(stock_prices.index, stock_prices, label='Historical Daily Closing Price')
plt.plot(sma_30.index, sma_30, color='green', label='30-Day SMA')
plt.title('Simple Moving Average')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```




```
symbol = 'META'
# Get the stock data
data, meta_data = ts.get_daily(symbol=symbol, outputsize='compact')

# The '4. close' column has the closing prices
closing_prices = data['4. close'].iloc[::-1]  # Reverse the order to have the oldest prices first

# Calculate the 30-day SMA
sma_30 = closing_prices.rolling(window=30).mean()

# Select the last 30 days of closing prices and SMA
last_30_days_prices = closing_prices.tail(30)
last_30_days_sma = sma_30.tail(30)

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(last_30_days_prices.index, last_30_days_prices, label='Historical Daily Closing Price')
plt.plot(last_30_days_sma.index, last_30_days_sma, label='30-Day SMA', color='orange')

# Formatting the plot
plt.title(f'{symbol} Closing Prices and 30-Day SMA (Last 30 Days)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)

# Setting x-axis major locator to show one tick per day and formatting date labels
plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gcf().autofmt_xdate()  # Auto rotate date labels

plt.tight_layout()
plt.show()
```



Exponential Smoothing:

Exponential Smoothing is a technique used to forecast time series data by assigning exponentially decreasing weights over time. It is more responsive to recent changes in the data than SMA, making it suitable for data with more fluctuations.

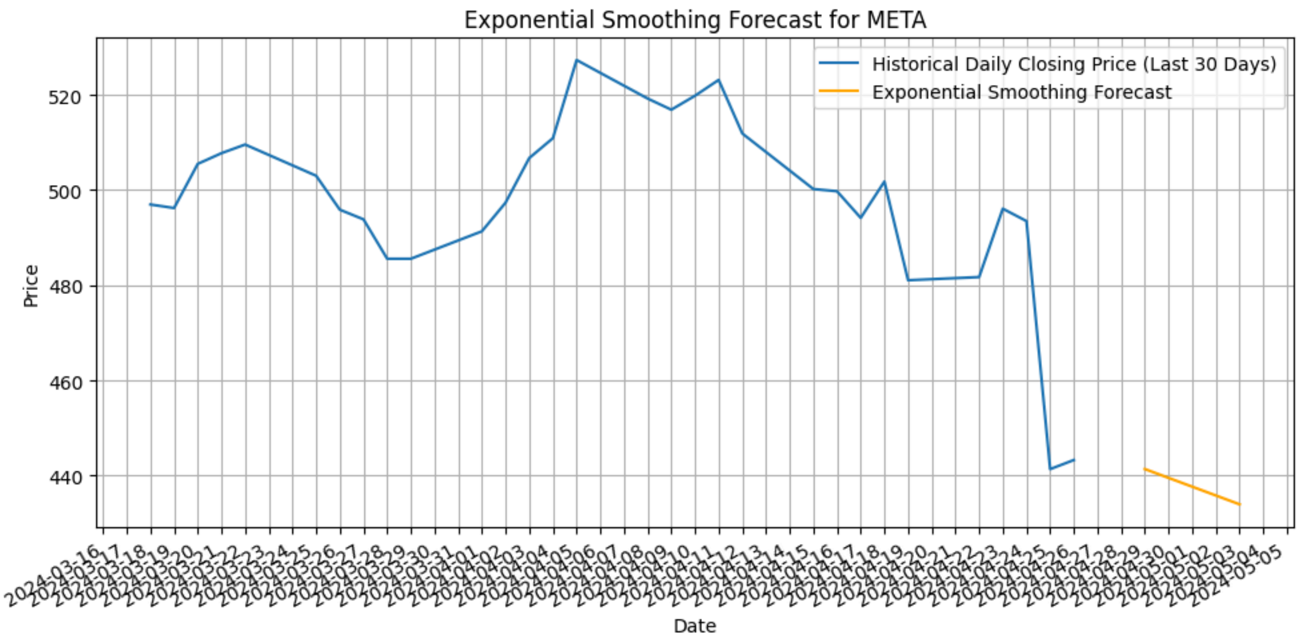
Rationale for Choosing Exponential Smoothing:

1. Responsiveness to New Data
2. Flexibility

```
# Import the ExponentialSmoothing class
from statsmodels.tsa.statespace.exponential_smoothing import ExponentialSmoothing

# Get the stock data
data, meta_data = ts.get_daily(symbol=symbol, outputsize='compact')
data.index = pd.to_datetime(data.index)  # Ensure the index is datetime

# Sort the data by date
stock_prices = data['4. close'].sort_index()

# Set the frequency of the index to business days
stock_prices = stock_prices.asfreq('B')

# Filling missing values if there are any
stock_prices.fillna(method='ffill', inplace=True)

# Select the last 30 days of closing prices
last_30_days_prices = stock_prices.last('30B')  # 'B' stands for business day frequency

# Define and fit the Exponential Smoothing model
model = ExponentialSmoothing(last_30_days_prices, trend='add', seasonal=None)
fitted_model = model.fit()

# Forecast the next 5 business days
forecast = fitted_model.forecast(steps=5)
print(forecast)

# Preparing the dates for the forecast
last_date = last_30_days_prices.index[-1]
forecast_dates = pd.date_range(start=last_date, periods=6, freq='B')[1:]  # exclude the last date of the known data
```

```
# Plotting the results
plt.figure(figsize=(10,5))
plt.plot(last_30_days_prices.index, last_30_days_prices, label='Historical Daily Closing Price (Last 30 Days)')
plt.plot(forecast_dates, forecast, color='orange', label='Exponential Smoothing Forecast')

# Formatting the plot
plt.title(f'Exponential Smoothing Forecast for {symbol}')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)

# Setting x-axis major locator and formatter for better date display
plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gcf().autofmt_xdate()  # Auto rotate date labels

plt.tight_layout()
plt.show()
```

```
2024-04-29    441.395821
2024-04-30    439.543081
2024-05-01    437.690341
2024-05-02    435.837601
2024-05-03    433.984861
Freq: B, Name: predicted_mean, dtype: float64
```



Start coding or generate with AI.

Start coding or generate with AI.

Data collection

Stock data is being collected

```
import requests
import pandas as pd
#get api key from alphavantage
# each api key makes 25 requests
# api documentation:https://www.alphavantage.co/documentation/
apikey='Your_key'

url = 'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=META&outputsize=full&apikey='+apikey
r = requests.get(url)
data = r.json()
```

```
print(data)
```

{'Meta Data': {'1. Information': 'Daily Prices (open, high, low, close) and Volumes', '2. Symbol': 'META', '3. Last Refreshed': '2024-0

```
daily_data=pd.DataFrame.from_dict(data['Time Series (Daily)']).T
```

```
daily_data.columns
```

```
Index(['1. open', '2. high', '3. low', '4. close', '5. volume'], dtype='object')
```

```
daily_data.rename(columns={'1. open':'open','2. high': 'high','3. low': 'low','4. close': 'close','5. volume': 'volume'},inplace=True)
```

```
daily_data
```

	open	high	low	close	volume
2024-04-26	441.4600	446.4400	431.9600	443.2900	32691443
2024-04-25	421.4000	445.7700	414.5000	441.3800	82890741
2024-04-24	508.0600	510.0000	484.5800	493.5000	37772677
2024-04-23	491.2500	498.7600	488.9700	496.1000	15079196
2024-04-22	489.7150	492.0100	473.4000	481.7300	17271125
...	...	...	...	...	...
2012-05-24	32.9500	33.2100	31.7700	33.0300	50237200
2012-05-23	31.3700	32.5000	31.3600	32.0000	73600000
2012-05-22	32.6100	33.5900	30.9400	31.0000	101786600
2012-05-21	36.5300	36.6600	33.0000	34.0300	168192700
2012-05-18	42.0500	45.0000	38.0000	38.2318	573576400

3004 rows × 5 columns

```
daily_data=daily_data[::-1]
```

Collecting Sentiment data from the news API

```
from datetime import datetime, timedelta
import requests
import pandas as pd

apikey = 'Your_key'  # Replace with your actual API key

# Calculate 30 days back from today
time_to = datetime.now()
time_from = time_to - timedelta(days=30)

# Format dates in the required format
time_to = time_to.strftime('%Y%m%dT%H%M')
time_from = time_from.strftime('%Y%m%dT%H%M')

# Sentiment API URL from Alpha Vantage
url = f'https://www.alphavantage.co/query?function=NEWS_SENTIMENT&time_from={time_from}&time_to={time_to}&sort=EARLIEST&symbol=META&limit=1(

r = requests.get(url)
data = r.json()
print(data)
```

{'items': '1000', 'sentiment_score_definition': 'x <= -0.35: Bearish; -0.35 < x <= -0.15: Somewhat-Bearish; -0.15 < x < 0.15: Neutral;

```
data
```

{'items': '1000',
 'sentiment_score_definition': 'x <= -0.35: Bearish; -0.35 < x <= -0.15: Somewhat-Bearish; -0.15 < x < 0.15: Neutral; 0.15 <= x <
0.35: Somewhat_Bullish; x >= 0.35: Bullish',

```
 'relevance_score_definition': '0 < x <= 1, with a higher score indicating higher relevance.',
 'feed': [{'title': "Is It Smart to Take Social Security if I'm Still Working?",
   'url': 'https://www.fool.com/retirement/2024/03/30/is-it-smart-to-take-social-security-if-im-still-wo/',
   'time_published': '20240330T091800',
   'authors': ['Maurie Backman'],
   'summary': "You're allowed to collect Social Security even if you have a job. Whether it's a good idea will depend on your circumstances.",
   'banner_image': 'https://g.foolcdn.com/image/?url=https%3A%2F%2Fg.foolcdn.com%2Feditorial%2Fimages%2F770882%2Folder-woman-taking-notes-at-laptop-gettyimages-1407163041.jpg&op=resize&w=700',
   'source': 'Motley Fool',
   'category_within_source': 'n/a',
   'source_domain': 'www.fool.com',
   'topics': [{'topic': 'Earnings', 'relevance_score': '0.576289'}],
   'overall_sentiment_score': 0.154475,
   'overall_sentiment_label': 'Somewhat-Bullish',
   'ticker_sentiment': []},
  {'title': '3 Reliable Dividend Growth Stocks With Yields Above 3% That You Can Buy Now and Hold for at Least a Decade',
   'url': 'https://www.fool.com/investing/2024/03/30/3-reliable-dividend-growth-stocks-with-yields-abov/',
   'time_published': '20240330T091900',
   'authors': ['Cory Renauer'],
   'summary': 'These advantaged businesses have what it takes to keep raising their payouts.',
   'banner_image': 'https://g.foolcdn.com/image/?url=https%3A%2F%2Fg.foolcdn.com%2Feditorial%2Fimages%2F770568%2Fgroup-of-investors-looking-at-charts.jpg&op=resize&w=700',
   'source': 'Motley Fool',
   'category_within_source': 'n/a',
   'source_domain': 'www.fool.com',
   'topics': [{'topic': 'Earnings', 'relevance_score': '0.967321'},
    {'topic': 'Life Sciences', 'relevance_score': '1.0'},
    {'topic': 'Financial Markets', 'relevance_score': '0.980922'}],
   'overall_sentiment_score': 0.238472,
   'overall_sentiment_label': 'Somewhat-Bullish',
   'ticker_sentiment': [{'ticker': 'ABBV',
     'relevance_score': '0.146916',
     'ticker_sentiment_score': '-0.045412',
     'ticker_sentiment_label': 'Neutral'},
    {'ticker': 'ABT',
     'relevance_score': '0.049221',
     'ticker_sentiment_score': '0.0',
     'ticker_sentiment_label': 'Neutral'},
    {'ticker': 'MDT',
     'relevance_score': '0.098255',
     'ticker_sentiment_score': '0.049573',
     'ticker_sentiment_label': 'Neutral'},
    {'ticker': 'SWAV',
     'relevance_score': '0.049221',
     'ticker_sentiment_score': '0.133067',
     'ticker_sentiment_label': 'Neutral'},
    {'ticker': 'JNJ',
     'relevance_score': '0.098255',
     'ticker_sentiment_score': '0.0',
     'ticker_sentiment_label': 'Neutral'}]},
  {'title': 'Huawei Begins Mass Deliveries Of Luxeed S7 Electric Sedan, Touted As Rival To Tesla Model S In Various Aspects',
   'url': 'https://www.benzinga.com/news/24/03/38012562/huawei-begins-mass-deliveries-of-luxeed-s7-electric-sedan-touted-as-rival-to-tesla-model-s-in-variou',
```


```
print(pd.DataFrame.from_dict(data))
```

```
    items                          sentiment_score_definition  \
0    1000  x <= -0.35: Bearish; -0.35 < x <= -0.15: Somew...
1    1000  x <= -0.35: Bearish; -0.35 < x <= -0.15: Somew...
2    1000  x <= -0.35: Bearish; -0.35 < x <= -0.15: Somew...
3    1000  x <= -0.35: Bearish; -0.35 < x <= -0.15: Somew...
4    1000  x <= -0.35: Bearish; -0.35 < x <= -0.15: Somew...
..    ...                                                ...
995  1000  x <= -0.35: Bearish; -0.35 < x <= -0.15: Somew...
996  1000  x <= -0.35: Bearish; -0.35 < x <= -0.15: Somew...
997  1000  x <= -0.35: Bearish; -0.35 < x <= -0.15: Somew...
998  1000  x <= -0.35: Bearish; -0.35 < x <= -0.15: Somew...
999  1000  x <= -0.35: Bearish; -0.35 < x <= -0.15: Somew...

                              relevance_score_definition  \
0    0 < x <= 1, with a higher score indicating hig...
1    0 < x <= 1, with a higher score indicating hig...
2    0 < x <= 1, with a higher score indicating hig...
3    0 < x <= 1, with a higher score indicating hig...
4    0 < x <= 1, with a higher score indicating hig...
..                                                 ...
995  0 < x <= 1, with a higher score indicating hig...
996  0 < x <= 1, with a higher score indicating hig...
997  0 < x <= 1, with a higher score indicating hig...
998  0 < x <= 1, with a higher score indicating hig...
999  0 < x <= 1, with a higher score indicating hig...
```

```
                                               feed
0    {'title': 'Is It Smart to Take Social Security...
1    {'title': '3 Reliable Dividend Growth Stocks W...
2    {'title': 'Huawei Begins Mass Deliveries Of Lu...
3    {'title': '2 Tech Stocks That Are Still No-Bra...
4    {'title': 'Huawei says partner Chery Automobil...
..                                              ...
995  {'title': 'LeBron James ties career high with ...
996  {'title': 'Will Nifty hit a new high above 22,...
997  {'title': 'Trading strategies for Nifty, Nifty...
998  {'title': 'IMM CAREHUB - REDEFINING HEALTHY AG...
999  {'title': 'Bragar Eagel & Squire, P.C. Reminds...

[1000 rows x 4 columns]
```

Extracting the Data

Extracting the data to create them a data frame

```
import pandas as pd
from datetime import datetime

all_date = {}
news_data = pd.DataFrame.from_dict(data)  # Ensure 'data' is defined and formatted properly

for feed in news_data['feed']:

    #print(feed)
    date = datetime.strptime(feed['time_published'].split('T')[0], '%Y%m%d').date()
    date_str = str(date)

    if date_str not in all_date:
        all_date[date_str] = {
            'sentiment_score': float(feed['overall_sentiment_score']),
            'sentiment_score_count': 1
        }
        #print(all_date[date_str])
    else:
        all_date[date_str]['sentiment_score'] += float(feed['overall_sentiment_score'])
        all_date[date_str]['sentiment_score_count'] += 1

    for topic in feed['topics']:
        topic_key = topic['topic']
        #print(topic_key)
        if topic_key not in all_date[date_str]:
            all_date[date_str][topic_key] = float(topic['relevance_score'])
            all_date[date_str][topic_key + '_count'] = 1
        else:
            all_date[date_str][topic_key] += float(topic['relevance_score'])
            all_date[date_str][topic_key + '_count'] += 1

print(all_date)
# Converting dictionary to DataFrame
date_data = pd.DataFrame.from_dict(all_date, orient='index')
```


```
{'2024-03-30': {'sentiment_score': 37.62475300000001, 'sentiment_score_count': 333, 'Earnings': 41.84654499999997, 'Earnings_count': 72
```


```
date_data
```

	sentiment_score	sentiment_score_count	Earnings	Earnings_count	Life Sciences	Life Sciences_count	Financial Markets	Financial Markets_count	Manufact
2024-03-30	37.624753	333	41.846545	72.0	13.059523	18	89.979550	150	43.8
2024-03-31	103.225390	652	124.404233	205.0	42.249994	68	233.179619	398	74.4
2024-04-01	3.247553	15	NaN	NaN	1.000000	1	4.549338	6	

3 rows × 32 columns

```
temp=date_data
```

```
# average out sentiment data with count columns and then drop count columns
for i in temp.columns:
    if 'count' not in i:
            temp[i]=temp[i]/temp[i+'_count']
            temp.drop(columns=[i+'_count'],inplace=True)
```

```
daily_data
```

	open	high	low	close	volume
2012-05-18	42.0500	45.0000	38.0000	38.2318	573576400
2012-05-21	36.5300	36.6600	33.0000	34.0300	168192700
2012-05-22	32.6100	33.5900	30.9400	31.0000	101786600
2012-05-23	31.3700	32.5000	31.3600	32.0000	73600000
2012-05-24	32.9500	33.2100	31.7700	33.0300	50237200
...	...	...	...	...	...
2024-04-22	489.7150	492.0100	473.4000	481.7300	17271125
2024-04-23	491.2500	498.7600	488.9700	496.1000	15079196
2024-04-24	508.0600	510.0000	484.5800	493.5000	37772677
2024-04-25	421.4000	445.7700	414.5000	441.3800	82890741
2024-04-26	441.4600	446.4400	431.9600	443.2900	32691443

3004 rows × 5 columns

```
# join sentiment data and stock price data
result =  temp.join(daily_data, how='outer')
```

```
result.index
```

```
Index(['2012-05-18', '2012-05-21', '2012-05-22', '2012-05-23', '2012-05-24',
       '2012-05-25', '2012-05-29', '2012-05-30', '2012-05-31', '2012-06-01',
       ...
       '2024-04-15', '2024-04-16', '2024-04-17', '2024-04-18', '2024-04-19',
       '2024-04-22', '2024-04-23', '2024-04-24', '2024-04-25', '2024-04-26'],
      dtype='object', length=3006)
```

```
final_data=result
```

```
# fill null values
final_data.fillna(method='ffill',inplace=True)
final_data.fillna(method='bfill',inplace=True)
```

Merging the stock data and new data in single dataframe

```
final_data
```

	sentiment_score	Earnings	Life Sciences	Financial Markets	Manufacturing	Technology	Energy & Transportation	Real Estate & Construction	Finance	Blockchain	...
2012-05-18	0.112987	0.581202	0.725529	0.599864	0.655117	0.748884	0.822129	0.629819	0.633145	0.445596	..
2012-05-21	0.112987	0.581202	0.725529	0.599864	0.655117	0.748884	0.822129	0.629819	0.633145	0.445596	..
2012-05-22	0.112987	0.581202	0.725529	0.599864	0.655117	0.748884	0.822129	0.629819	0.633145	0.445596	..
2012-05-23	0.112987	0.581202	0.725529	0.599864	0.655117	0.748884	0.822129	0.629819	0.633145	0.445596	..
2012-05-24	0.112987	0.581202	0.725529	0.599864	0.655117	0.748884	0.822129	0.629819	0.633145	0.445596	..
...	...	...	...	...	...	...	...	...	...	...	..
2024-04-22	0.216504	0.606850	1.000000	0.758223	0.620139	0.916667	0.724359	0.694444	1.000000	0.402078	..
2024-04-23	0.216504	0.606850	1.000000	0.758223	0.620139	0.916667	0.724359	0.694444	1.000000	0.402078	..
2024-04-24	0.216504	0.606850	1.000000	0.758223	0.620139	0.916667	0.724359	0.694444	1.000000	0.402078	..
2024-04-25	0.216504	0.606850	1.000000	0.758223	0.620139	0.916667	0.724359	0.694444	1.000000	0.402078	..
2024-04-26	0.216504	0.606850	1.000000	0.758223	0.620139	0.916667	0.724359	0.694444	1.000000	0.402078	..

3006 rows × 21 columns

Generating CSV file

```
final_data.to_csv('meta_sentiment.csv')
```

Reading the dateset

```
data = pd.read_csv('/content/meta_sentiment.csv')
```

```
data.columns
```

```
Index(['Unnamed: 0', 'sentiment_score', 'Economy - Monetary',
       'Financial Markets', 'Earnings', 'Mergers & Acquisitions', 'Technology',
       'Finance', 'Real Estate & Construction', 'Energy & Transportation',
       'Economy - Fiscal', 'Retail & Wholesale', 'Manufacturing', 'Blockchain',
       'Life Sciences', 'IPO', 'Economy - Macro', 'open', 'high', 'low',
       'close', 'volume'],
      dtype='object')
```

Making ESG sentiment columns i.e e_sentiment, s_sentiment and g_sentiment

Environment

1. Blockchain: Blockchain technology has significant environmental implications, particularly in terms of energy consumption. Cryptocurrencies like Bitcoin, which are based on blockchain, are known for their high energy usage during the mining process1.
2. Energy & Transportation: This category directly relates to the environment as it involves the production and consumption of energy, as well as transportation systems, both of which have significant environmental impacts.
3. Manufacturing: Manufacturing processes often have significant environmental impacts, including pollution and resource depletion.
4. Real Estate & Construction: These sectors can have significant environmental impacts, including land use changes, resource consumption, and waste generation.

Social

1. Earnings: Earnings relate to income and wealth distribution, which are key social issues.

2. Life Sciences: This field includes healthcare and biotechnology, which have significant social implications in terms of health outcomes and ethical considerations.
3. Retail & Wholesale: These sectors are part of the consumer economy and relate to social issues such as consumer behavior and employment.
4. Technology: Technology has significant social implications, including impacts on communication, privacy, and employment.

Government

1. IPO: Initial Public Offerings (IPOs) are regulated by government entities like the Securities and Exchange Commission in the U.S., making them a government aspect.
2. Mergers & Acquisitions: These are also regulated by government entities to prevent anti-competitive practices.
3. Financial Markets: Financial markets are heavily regulated by government entities to maintain stability and protect consumers.
4. Economy - Fiscal Policy (e.g., tax reform, government spending): Fiscal policy is a direct function of government, involving decisions about government spending and taxation.
5. Economy - Monetary Policy (e.g., interest rates, inflation): Monetary policy is also a direct function of government, typically managed by a central bank.
6. Economy - Macro/Overall: The overall economy is influenced by government policies and regulations.
7. Finance: While finance has social and environmental aspects, it is also heavily regulated by government entities, making it a government aspect as well.

```
import pandas as pd
import numpy as np
from statsmodels.tsa.statespace.varmax import VARMAX
from statsmodels.regression.linear_model import OLS
from statsmodels.tsa.stattools import grangercausalitytests
from statsmodels.tools.tools import add_constant
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

Load and preprocess the data

```
data = pd.read_csv('/content/meta_sentiment.csv')
data.rename(columns={'Unnamed: 0': 'date'}, inplace=True)
data['date'] = pd.to_datetime(data['date'])
data.set_index('date', inplace=True)
```

```
# Fill missing data
data = data.fillna(method='ffill').fillna(method='bfill')
```

```
data.head()
```

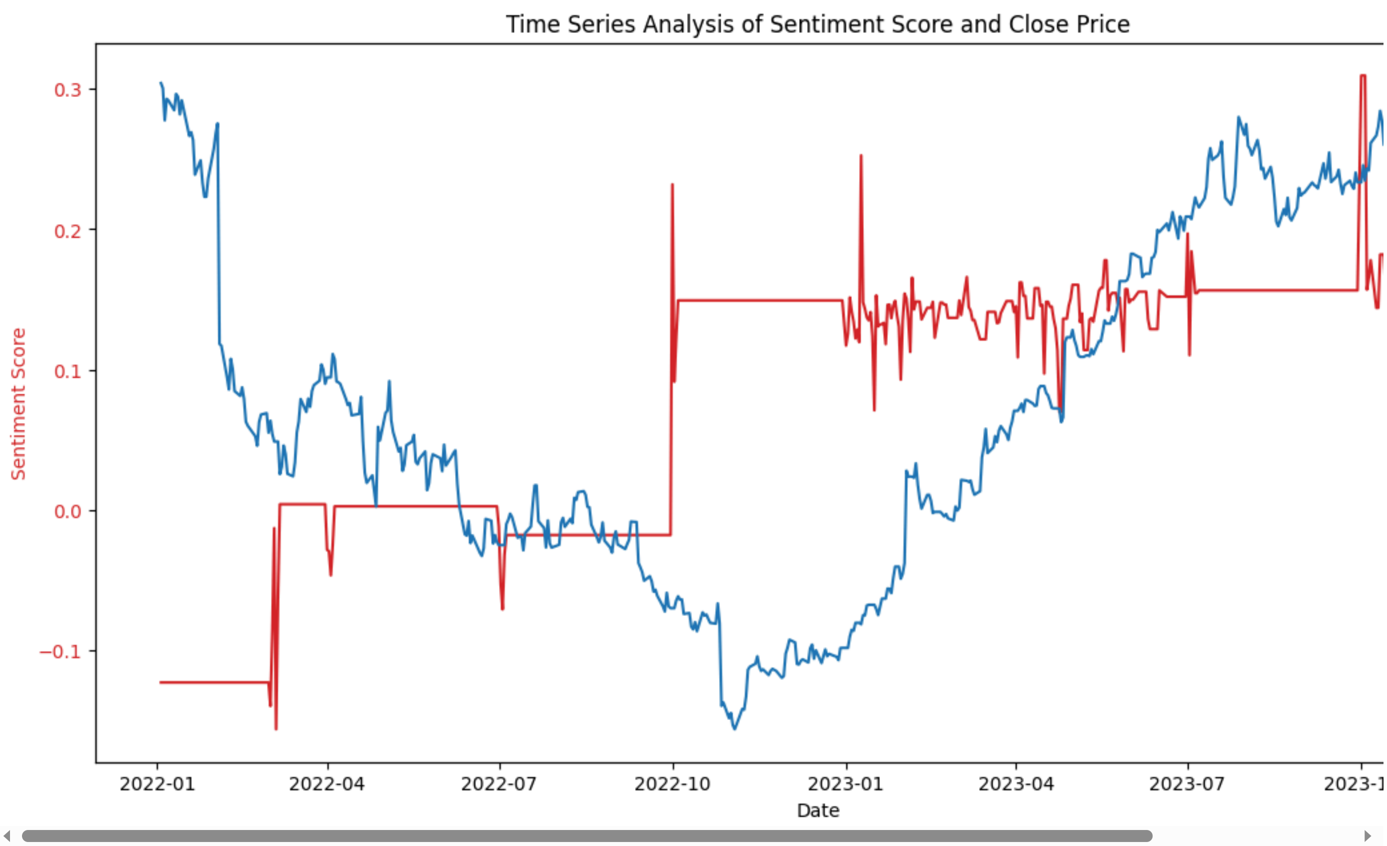
	sentiment_score	Economy - Monetary	Financial Markets	Earnings	Mergers & Acquisitions	Technology	Finance	Real Estate & Construction	Energy & Transportation	Economy - Fiscal	...	M
date												
2022-01-03	-0.123265	0.414848	0.497703	0.656845	0.158519	0.686458	0.676042	0.44	0.67	0.158519	...	
2022-01-04	-0.123265	0.414848	0.497703	0.656845	0.158519	0.686458	0.676042	0.44	0.67	0.158519	...	
2022-01-05	-0.123265	0.414848	0.497703	0.656845	0.158519	0.686458	0.676042	0.44	0.67	0.158519	...	
2022-01-06	-0.123265	0.414848	0.497703	0.656845	0.158519	0.686458	0.676042	0.44	0.67	0.158519	...	
2022-01-07	-0.123265	0.414848	0.497703	0.656845	0.158519	0.686458	0.676042	0.44	0.67	0.158519	...	

5 rows × 21 columns

EDA

Time Series Plot for Sentiment Score and Close Price:

```
# Plotting sentiment score and closing price
fig, ax1 = plt.subplots(figsize=(14, 7))
color = 'tab:red'
ax1.set_xlabel('Date')
ax1.set_ylabel('Sentiment Score', color=color)
ax1.plot(data.index, data['sentiment_score'], color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()
color = 'tab:blue'
ax2.set_ylabel('Close Price', color=color)
ax2.plot(data.index, data['close'], color=color)
ax2.tick_params(axis='y', labelcolor=color)

plt.title('Time Series Analysis of Sentiment Score and Close Price')
plt.show()
```

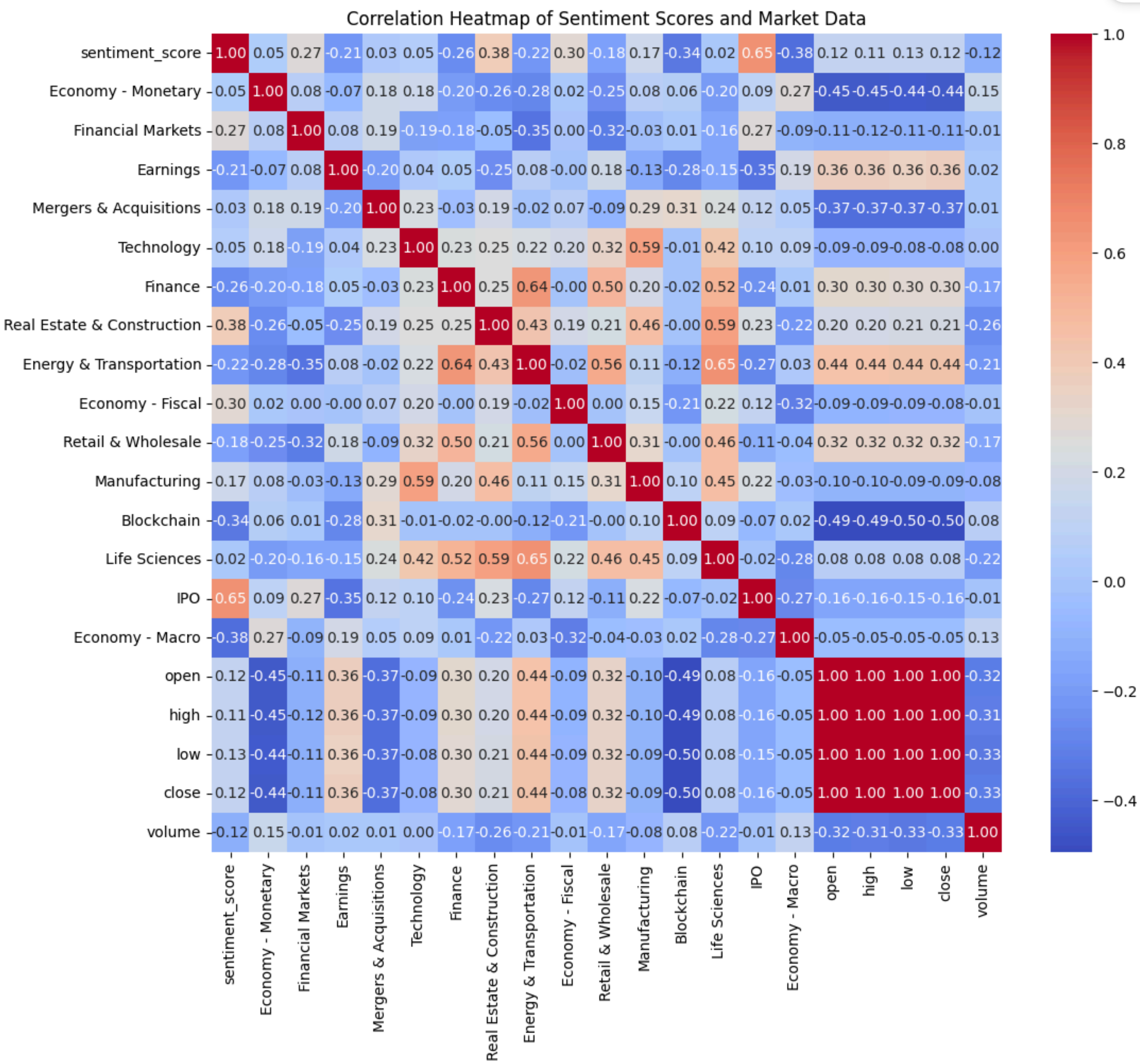


Here's the time series plot showing both the overall sentiment score and the closing price over time. The sentiment score is shown in red, and the closing price is in blue.

This visualization can help you analyze how changes in sentiment might correlate with changes in market price. For instance, significant drops or increases in sentiment might precede similar movements in the market.

Correlation Heatmap:

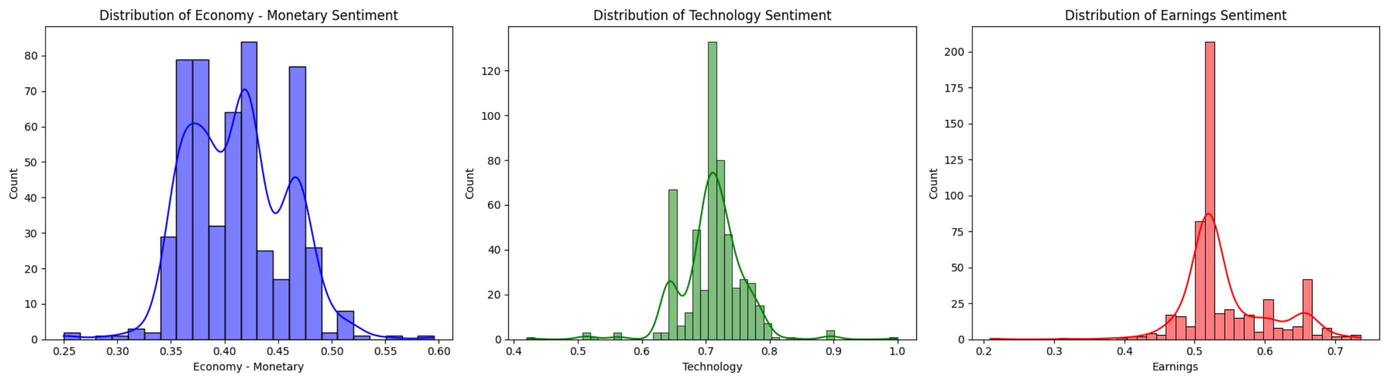
```
import seaborn as sns
import numpy as np

# Select columns that are numerical
correlation_data = data.select_dtypes(include=[np.number])

# Calculate the correlation matrix
correlation_matrix = correlation_data.corr()

# Generate a heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap of Sentiment Scores and Market Data')
plt.show()
```



Here's the correlation heatmap of sentiment scores and market data. The values range from -1 to 1, where 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation. This visualization can help identify which factors are most closely related.

Distribution Plots for Selected Sentiment Scores:

```
# Distribution plots for 'Economy - Monetary', 'Technology', 'Earnings'
plt.figure(figsize=(18, 5))

plt.subplot(1, 3, 1)
sns.histplot(data['Economy - Monetary'], kde=True, color='blue')
plt.title('Distribution of Economy - Monetary Sentiment')

plt.subplot(1, 3, 2)
sns.histplot(data['Technology'], kde=True, color='green')
plt.title('Distribution of Technology Sentiment')

plt.subplot(1, 3, 3)
sns.histplot(data['Earnings'], kde=True, color='red')
plt.title('Distribution of Earnings Sentiment')

plt.tight_layout()
plt.show()
```



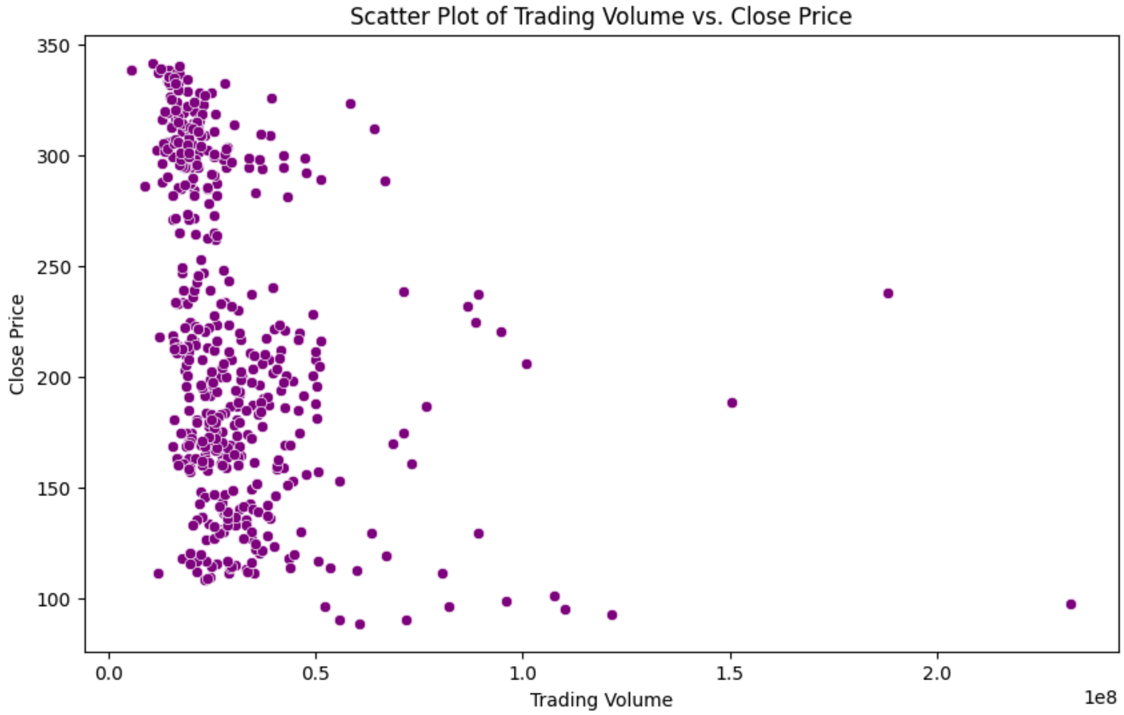
Here are the distribution plots for three different sentiment scores:

1. Economy - Monetary Sentiment: Displayed in blue.
2. Technology Sentiment: Displayed in green.
3. Earnings Sentiment: Displayed in red.

These plots show the frequency distribution of each sentiment score, helping us understand the typical values and variability within each theme. For example, you can see whether a sentiment tends to be mostly positive, negative, or neutral.

Scatter Plot of Trading Volume vs. Close Price:

```
# Scatter plot of volume and close price
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data, x='volume', y='close', color='purple')
plt.title('Scatter Plot of Trading Volume vs. Close Price')
plt.xlabel('Trading Volume')
plt.ylabel('Close Price')
plt.show()
```



Here's the scatter plot showing the relationship between trading volume and the closing price. This plot can help identify any patterns or trends, such as whether higher volumes are associated with certain price levels.

From this scatter plot, we can visually assess the spread and concentration of data points, giving us insight into how these two variables might interact.

Calculating percentage changes for different time frames before differencing

```
# Calculate percentage changes for different time frames before differencing
for days in [3, 7, 30]:
    data[f'close_pct_change_{days}d'] = data['close'].pct_change(periods=days)
    data[f'sentiment_score_change_{days}d'] = data['sentiment_score'].diff(periods=days)


# First differencing to enforce stationarity
data_diff = data.diff().dropna()
```

Aggregate sentiment scores into categories in the differenced data

```
data_diff['e_sentiment'] = (data_diff['Blockchain'] + data_diff['Energy & Transportation'] +
                            data_diff['Manufacturing'] + data_diff['Real Estate & Construction']) / 4
data_diff['s_sentiment'] = (data_diff['Earnings'] + data_diff['Life Sciences'] +
                            data_diff['Retail & Wholesale'] + data_diff['Technology']) / 4
data_diff['g_sentiment'] = (data_diff['IPO'] + data_diff['Mergers & Acquisitions'] +
                            data_diff['Financial Markets'] + data_diff['Economy - Monetary'] +
                            data_diff['Economy - Fiscal'] + data_diff['Economy - Macro'] +
                            data_diff['Finance']) / 7
```

```
data_diff.columns
```

```
Index(['sentiment_score', 'Economy - Monetary', 'Financial Markets',
       'Earnings', 'Mergers & Acquisitions', 'Technology', 'Finance',
       'Real Estate & Construction', 'Energy & Transportation',
       'Economy - Fiscal', 'Retail & Wholesale', 'Manufacturing', 'Blockchain',
       'Life Sciences', 'IPO', 'Economy - Macro', 'open', 'high', 'low',
       'close', 'volume', 'close_pct_change_3d', 'sentiment_score_change_3d',
       'close_pct_change_7d', 'sentiment_score_change_7d',
       'close_pct_change_30d', 'sentiment_score_change_30d', 'e_sentiment',
```

```
            's_sentiment', 'g_sentiment'],
           dtype='object')
```

```
# Setup endogenous and exogenous variables
endog = data_diff[['close', 'volume']]
exog = data_diff[['e_sentiment', 's_sentiment', 'g_sentiment', 'sentiment_score'] +
                 [f'close_pct_change_{d}d' for d in [3, 7, 30]]]
```

Scaling the Data

```
# Scale the endogenous variables
scaler_endog = StandardScaler()
endog_scaled = scaler_endog.fit_transform(endog)
```

Splitting the data

```
# Splitting the dataset in a time-series way
split_ratio = 0.8
split_idx = int(len(data_diff) * split_ratio)
X_train = exog.iloc[:split_idx]
X_test = exog.iloc[split_idx:]
y_train = endog_scaled[:split_idx]
y_test = endog_scaled[split_idx:]
```

Granger Causality Tests on the Time Series Data

doing granger casuality test of each variable with close variable. it helps to identify th ecorrect lag to pick

```
# Granger Causality Tests
max_lags = 50
for i in exog.columns:
#gc_results = grangercausalitytests(data_diff[['close', 'e_sentiment']], max_lags, verbose=True)
  print("\n Column_name:",i)
  gc_results = grangercausalitytests(data_diff[['close',i]], max_lags, verbose=True)
```

```
parameter F test:         F=1.1564  , p=0.3259  , df_denom=493, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=1.3152  , p=0.2632  , df_denom=490, df_num=4
ssr based chi2 test:   chi2=5.3574  , p=0.2526  , df=4
likelihood ratio test: chi2=5.3289  , p=0.2552  , df=4
parameter F test:         F=1.3152  , p=0.2632  , df_denom=490, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=1.0858  , p=0.3673  , df_denom=487, df_num=5
ssr based chi2 test:   chi2=5.5516  , p=0.3523  , df=5
```

```
parameter F test:         F=1.0464  , p=0.4000  , df_denom=478, df_num=8

Granger Causality
number of lags (no zero) 9
ssr based F test:         F=0.9786  , p=0.4568  , df_denom=475, df_num=9
ssr based chi2 test:   chi2=9.1596  , p=0.4227  , df=9
likelihood ratio test: chi2=9.0757  , p=0.4303  , df=9
parameter F test:         F=0.9786  , p=0.4568  , df_denom=475, df_num=9

Granger Causality
number of lags (no zero) 10
ssr based F test:         F=0.9766  , p=0.4629  , df_denom=472, df_num=10
ssr based chi2 test:   chi2=10.2001 , p=0.4231  , df=10
likelihood ratio test: chi2=10.0961 , p=0.4321  , df=10
parameter F test:         F=0.9766  , p=0.4629  , df_denom=472, df_num=10

Granger Causality
number of lags (no zero) 11
ssr based F test:         F=0.9284  , p=0.5125  , df_denom=469, df_num=11
ssr based chi2 test:   chi2=10.7134 , p=0.4676  , df=11
likelihood ratio test: chi2=10.5984 , p=0.4775  , df=11
parameter F test:         F=0.9284  , p=0.5125  , df_denom=469, df_num=11
```

```
# Grid search for VARMAX parameters
best_aic = np.inf
best_order = None
best_model = None
```

Running VARMAX Model

VARMAX is an advanced form of the vector autoregressive model and is used for multivariate time series data where the variables influence each other.

Rationale for Choosing VARMAX:

1. Multivariate Time Series Analysis: It allows the simultaneous modeling of more than one time-dependent variable, which is ideal for analyzing stocks from multiple companies or indices.
2. Complex Dynamics: VARMAX can capture the interdependencies between variables alongside external influences, offering a comprehensive framework for dynamic relationships in financial data.

here im only doing lags from (1,8) due to lack of compute resources, if compute power was there then (1,50) could have been taken

```
for p in range(1, 8):
    for q in range(1, 8):
        try:
            model = VARMAX(y_train, exog=X_train, order=(p, q), trend='n', enforce_stationarity=True)
            fitted_model = model.fit(disp=False, maxiter=200)
            if fitted_model.aic < best_aic:
                best_aic = fitted_model.aic
                best_order = (p, q)
                best_model = fitted_model
        except Exception as e:
            print(f"Failed to fit VARMAX with order (p={p}, q={q}): {e}")
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/varmax.py:160: EstimationWarning: Estimation of VARMA(p,q) models is
  warn('Estimation of VARMA(p,q) models is not generically robust,'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but it has
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/varmax.py:160: EstimationWarning: Estimation of VARMA(p,q) models is
  warn('Estimation of VARMA(p,q) models is not generically robust,'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but it has
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to co
  warnings.warn("Maximum Likelihood optimization failed to "
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/varmax.py:160: EstimationWarning: Estimation of VARMA(p,q) models is
  warn('Estimation of VARMA(p,q) models is not generically robust,'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but it has
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to co
  warnings.warn("Maximum Likelihood optimization failed to "
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/varmax.py:160: EstimationWarning: Estimation of VARMA(p,q) models is
  warn('Estimation of VARMA(p,q) models is not generically robust,'
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but it has
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to co
  warnings.warn("Maximum Likelihood optimization failed to "
```

```
if best_model is not None:
    print(f"Best AIC: {best_aic} for model order: {best_order}")
    print(best_model.summary())
else:
    print("No suitable model was found.")
```

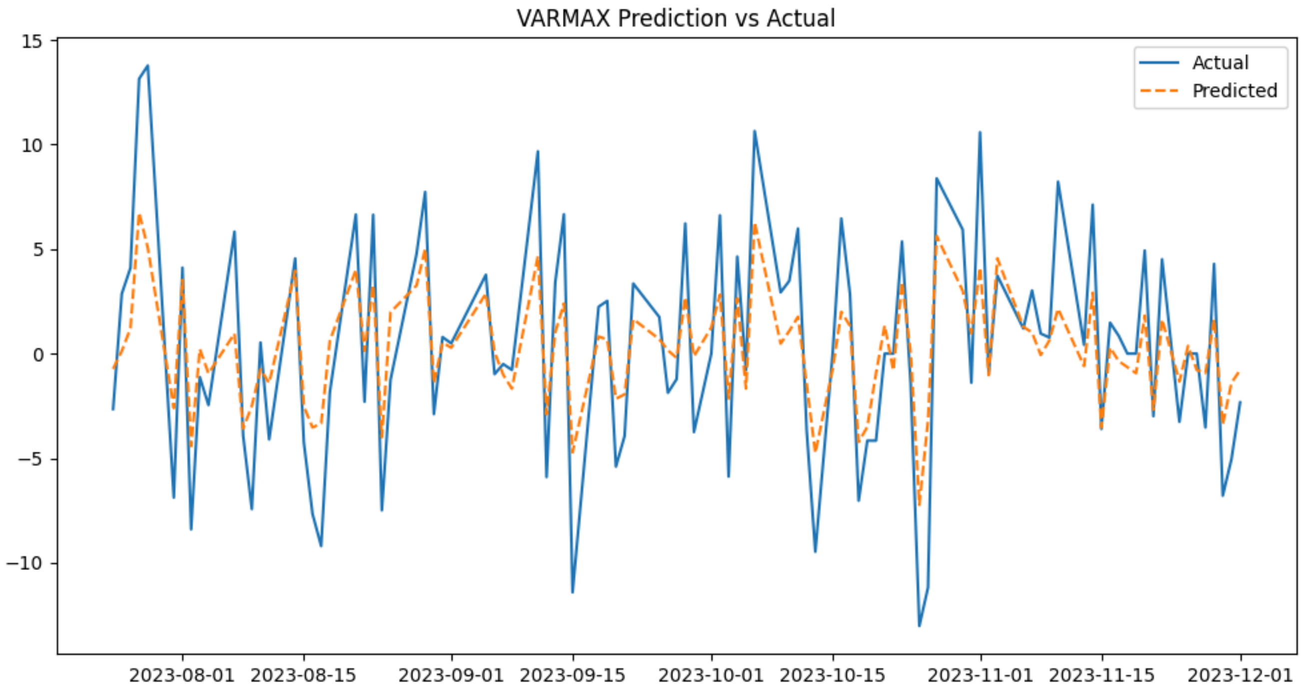
```
------------------------------------------------------------------------------------
L1.y1                        -0.0420      0.018     -2.377      0.017      -0.077      -0.007
L1.y2                        -0.0444      0.116     -0.384      0.701      -0.271       0.182
L2.y1                        -0.0522      0.020     -2.667      0.008      -0.091      -0.014
L2.y2                         0.2367      0.109      2.171      0.030       0.023       0.450
L3.y1                         0.7170      0.025     28.618      0.000       0.668       0.766
L3.y2                         0.1841      0.105      1.758      0.079      -0.021       0.389
L1.e(y1)                      0.1772      0.174      1.021      0.307      -0.163       0.517
L1.e(y2)                      0.0157      0.115      0.136      0.892      -0.210       0.242
L2.e(y1)                      0.2605      0.528      0.493      0.622      -0.774       1.295
L2.e(y2)                     -0.2589      0.147     -1.765      0.078      -0.546       0.029
L3.e(y1)                     -0.2931      0.212     -1.383      0.167      -0.709       0.122
L3.e(y2)                     -0.0732      0.086     -0.849      0.396      -0.242       0.096
L4.e(y1)                      0.3052      0.095      3.215      0.001       0.119       0.491
L4.e(y2)                      0.1563      0.060      2.614      0.009       0.039       0.274
beta.e_sentiment             -0.6347      0.421     -1.506      0.132      -1.461       0.191
beta.s_sentiment              0.4533      0.358      1.265      0.206      -0.249       1.156
beta.g_sentiment              0.2918      0.674      0.433      0.665      -1.029       1.613
beta.sentiment_score         -0.2974      0.662     -0.449      0.653      -1.595       1.000
beta.close_pct_change_3d     22.0793      0.535     41.292      0.000      21.031      23.127
beta.close_pct_change_7d      3.9633      0.471      8.409      0.000       3.039       4.887
beta.close_pct_change_30d     1.4615      0.300      4.873      0.000       0.874       2.049
                                   Results for equation y2
====================================================================================
                                coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------
L1.y1                        -0.0211      0.052     -0.405      0.686      -0.123       0.081
L1.y2                        -0.4210      0.208     -2.019      0.043      -0.830      -0.012
L2.y1                        -0.0079      0.060     -0.132      0.895      -0.126       0.110
L2.y2                        -0.7274      0.122     -5.973      0.000      -0.966      -0.489
L3.y1                        -0.1694      0.066     -2.573      0.010      -0.298      -0.040
L3.y2                         0.0539      0.176      0.305      0.760      -0.292       0.400
L1.e(y1)                     -0.2422      0.738     -0.328      0.743      -1.688       1.204
L1.e(y2)                     -0.0391      0.225     -0.173      0.862      -0.481       0.403
L2.e(y1)                     -0.3288      0.633     -0.520      0.603      -1.569       0.911
L2.e(y2)                      0.4704      0.170      2.763      0.006       0.137       0.804
L3.e(y1)                      0.1799      0.673      0.267      0.789      -1.139       1.499
L3.e(y2)                     -0.3936      0.196     -2.009      0.045      -0.778      -0.010
L4.e(y1)                     -0.3284      0.253     -1.296      0.195      -0.825       0.168
L4.e(y2)                     -0.0254      0.105     -0.242      0.809      -0.231       0.180
beta.e_sentiment             -0.0382      0.472     -0.081      0.936      -0.964       0.888
beta.s_sentiment              0.0405      0.350      0.116      0.908      -0.646       0.727
beta.g_sentiment              1.3059      0.682      1.914      0.056      -0.031       2.643
beta.sentiment_score          1.7704      0.904      1.959      0.050      -0.001       3.542
beta.close_pct_change_3d      0.8960      1.027      0.872      0.383      -1.117       2.909
beta.close_pct_change_7d     -1.8288      0.970     -1.885      0.059      -3.730       0.073
beta.close_pct_change_30d     0.0337      0.614      0.055      0.956      -1.170       1.237
                              Error covariance matrix
==================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
sqrt.var.y1       0.2876      0.074      3.871      0.000       0.142       0.433
sqrt.cov.y1.y2    0.2383      0.315      0.757      0.449      -0.378       0.855
sqrt.var.y2       0.9498      0.160      5.937      0.000       0.636       1.263
==================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Performance Metrics and Results:

1. The model demonstrated a Best AIC (Akaike Information Criterion) of 1338.4427341119303, suggesting a good fit to the data with respect to the complexity of the model. The AIC helps in balancing the model's fit against its complexity, with a lower AIC indicating a more efficient model.
2. The model's BIC (Bayesian Information Criterion) and HQIC (Hannan-Quinn Information Criterion) scores were also considered, which further supported the selection of the (3,4) order due to its better balance between explanatory power and simplicity.

```
# Prediction for VARMAX model
varmax_pred = fitted_model.get_forecast(steps=len(X_test), exog=X_test)
varmax_pred_mean = scaler_endog.inverse_transform(varmax_pred.predicted_mean)  # Correct inverse scaling
plt.figure(figsize=(12, 6))
plt.plot(data.index[-len(y_test):], scaler_endog.inverse_transform(y_test)[:, 0], label='Actual')
plt.plot(data.index[-len(y_test):], varmax_pred_mean[:, 0], label='Predicted', linestyle='--')
plt.title('VARMAX Prediction vs Actual')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836: FutureWarning: No supported index is available. In the n
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/varmax.py:160: EstimationWarning: Estimation of VARMA(p,q) models is
  warn('Estimation of VARMA(p,q) models is not generically robust,'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836: ValueWarning: No supported index is available. Predictio
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/varmax.py:160: EstimationWarning: Estimation of VARMA(p,q) models
  warn('Estimation of VARMA(p,q) models is not generically robust,'
```



OLS Model(Base model before VARMAX)

Ordinary Least Squares (OLS) is a type of linear regression technique used for estimating the unknown parameters in a linear regression model. It is one of the most basic and commonly used predictive techniques.

Rationale for Choosing OLS:

1. Baseline Comparisons: OLS provides a baseline to assess the impact of ESG factors on stock prices without the complexities of time-series models. This makes it particularly useful for initial exploratory analysis.
2. Simplicity and Transparency: The simplicity of the OLS model allows for clear interpretation and straightforward analysis of the relationship between stock prices and explanatory variables.

```
# OLS Model
X_ols = add_constant(X_train)
ols_model = OLS(data_diff['close'].iloc[:split_idx], X_ols)
ols_fitted = ols_model.fit()
```

```
# Predict and evaluate OLS model
X_test_ols = add_constant(X_test)
y_pred_ols = ols_fitted.predict(X_test_ols)
```

```
# Display OLS results and performance metrics
print(ols_fitted.summary())
```

```
print(ols_fitted.summary())

                            OLS Regression Results                            
==============================================================================
Dep. Variable:                  close   R-squared:                       0.698
Model:                            OLS   Adj. R-squared:                  0.692
Method:                 Least Squares   F-statistic:                     129.9
Date:                Mon, 29 Apr 2024   Prob (F-statistic):           3.00e-98
Time:                        10:22:50   Log-Likelihood:                -1029.9
No. Observations:                 402   AIC:                             2076.
Df Residuals:                     394   BIC:                             2108.
Df Model:                           7                                         
Covariance Type:            nonrobust                                         
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const                0.1460      0.158      0.923      0.356      -0.165       0.457
e_sentiment         -8.6710      4.593     -1.888      0.060     -17.701       0.359
s_sentiment          0.7178      4.451      0.161      0.872      -8.033       9.468
g_sentiment          0.3298      8.792      0.038      0.970     -16.955      17.615
```