SC1015 MINI PROJECT

$VOO Stock Price Prediction

TAN UEI HORNG
LSTM Model

CHONG JIA CHERN
ARIMA Model

CHARLTON SIAW
Sentimental Analysis

Contributors

TAN UEI HORNG

LSTM Model

CHONG JIA CHERN

ARIMA Model

CHARLTON SIAW

Sentimental Analysis

MOTIVATION

WHY $VOO

Vanguard S&P 500 ETF (VOO)

widely recognized and traded ETF

1 trillion
assets under management

500 LARGEST US COMPANIES

csv dataset pulled from Yahoo Finance

Problem Statement

Our project aims to **predict the future stock prices of VOO using ARIMA, LSTM models, and sentiment analysis**.

The specific problem we are addressing is to determine which of these techniques is most effective for predicting stock prices in a real-world, high-volume trading environment.

Data Preparation & Cleaning

- verify its completeness and integrity. We checked for any missing values across different columns.
- convert the 'Date' column to a datetime format, which is crucial for time-series analysis
- For forecasting, extract 'Close' price, as it represents the final trading price of the day and is commonly used as a primary indicator in stock price analysis.

```
# Check for Missing Values
print(voo_data.isnull().sum())
```

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

```
# Convert the 'Date' column to datetime format and set it as the index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)

# Now, focus on the 'Close' column
df_close = df[['Close']]
df_close
```

trends?

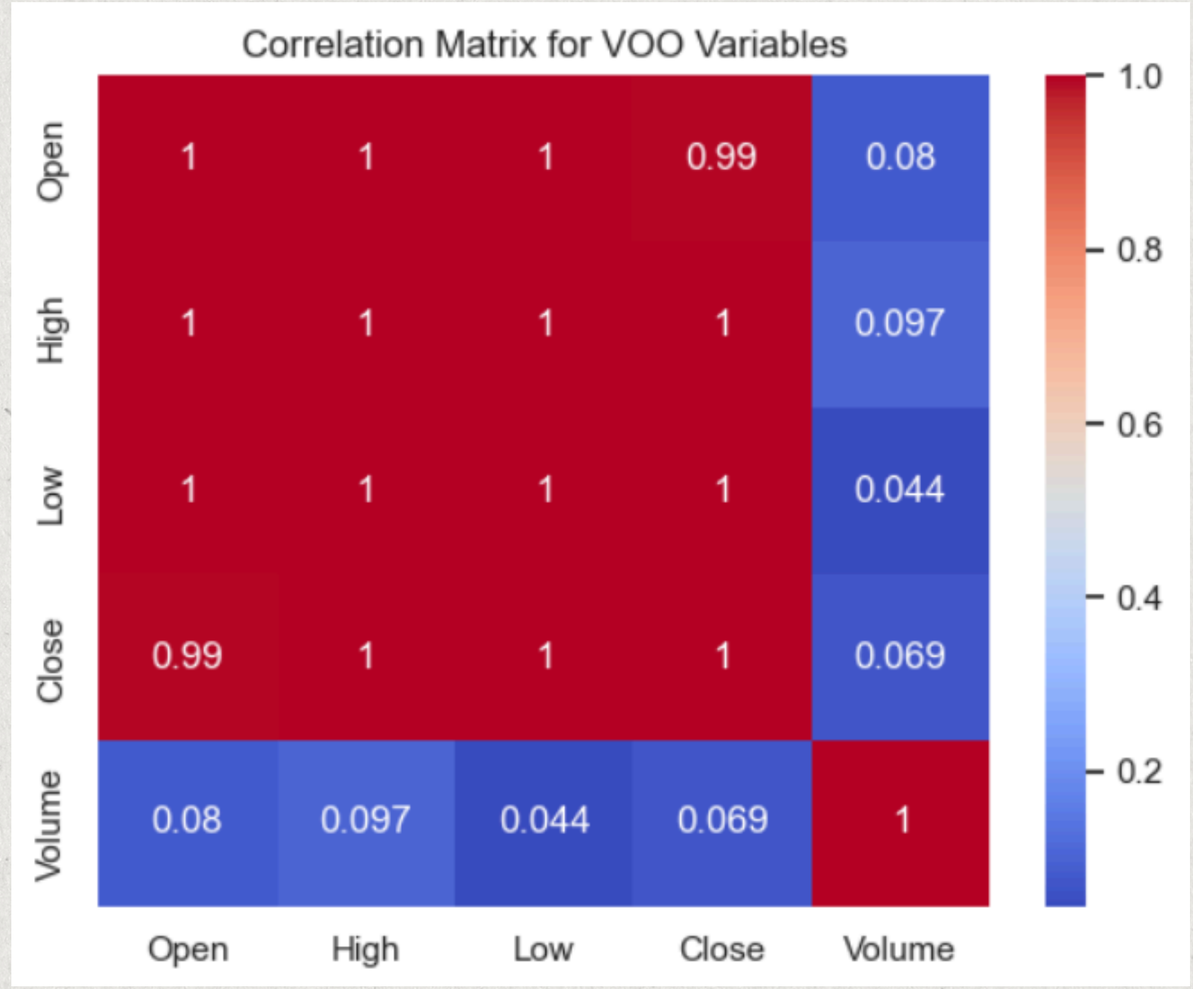
seasonality?

Exploratory Data Analysis (EDA)

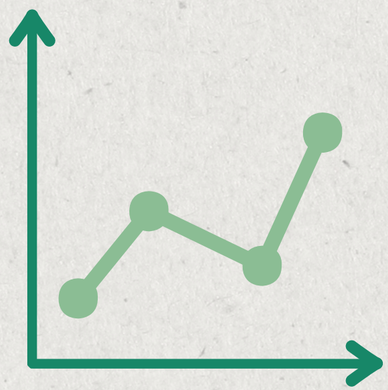
volatility?

Multi-variate Exploration

highly correlated!

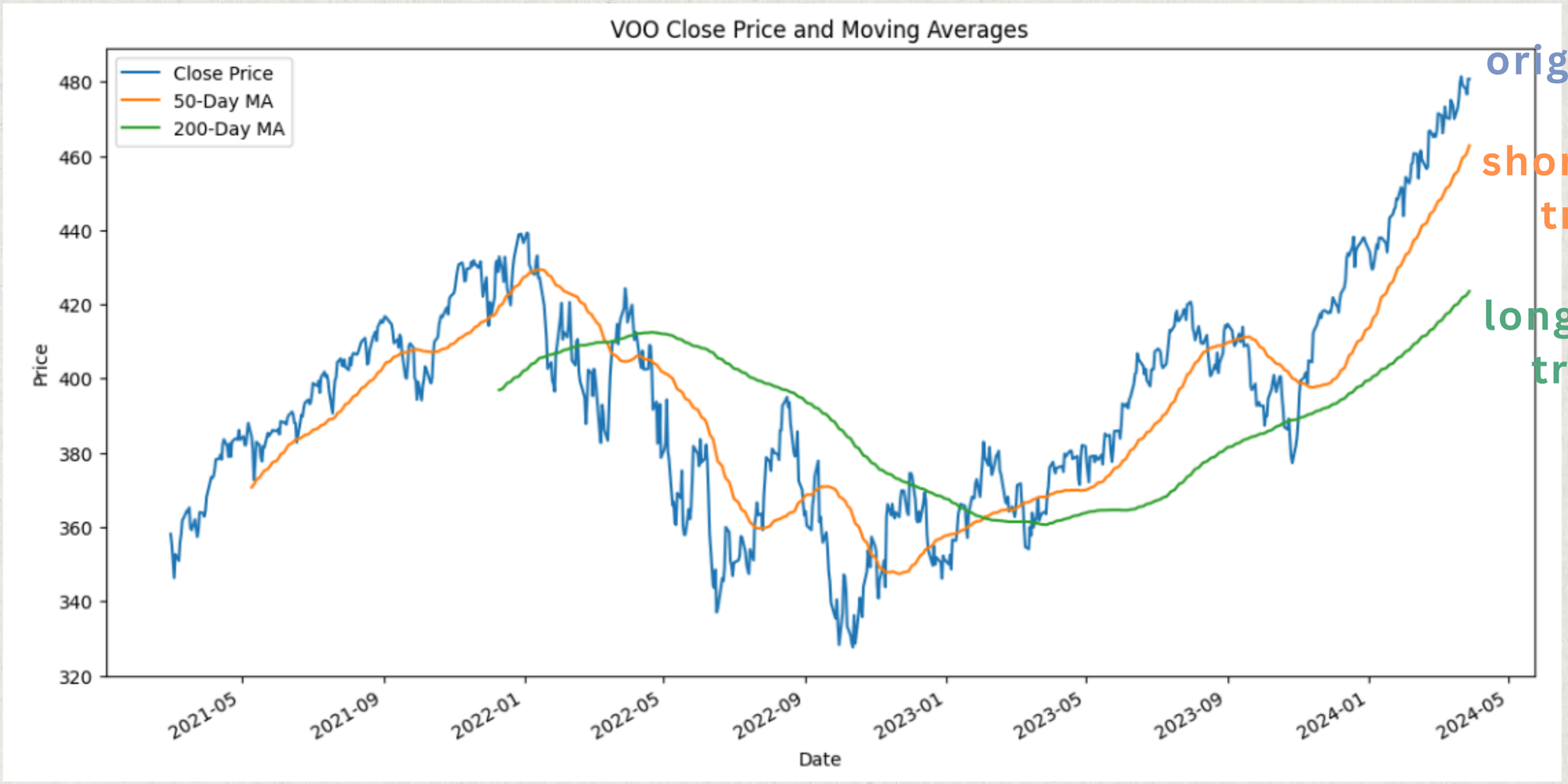


Data Visualisation



- Closing price and moving average
- Daily returns and volatility

Closing Price & Moving Averages

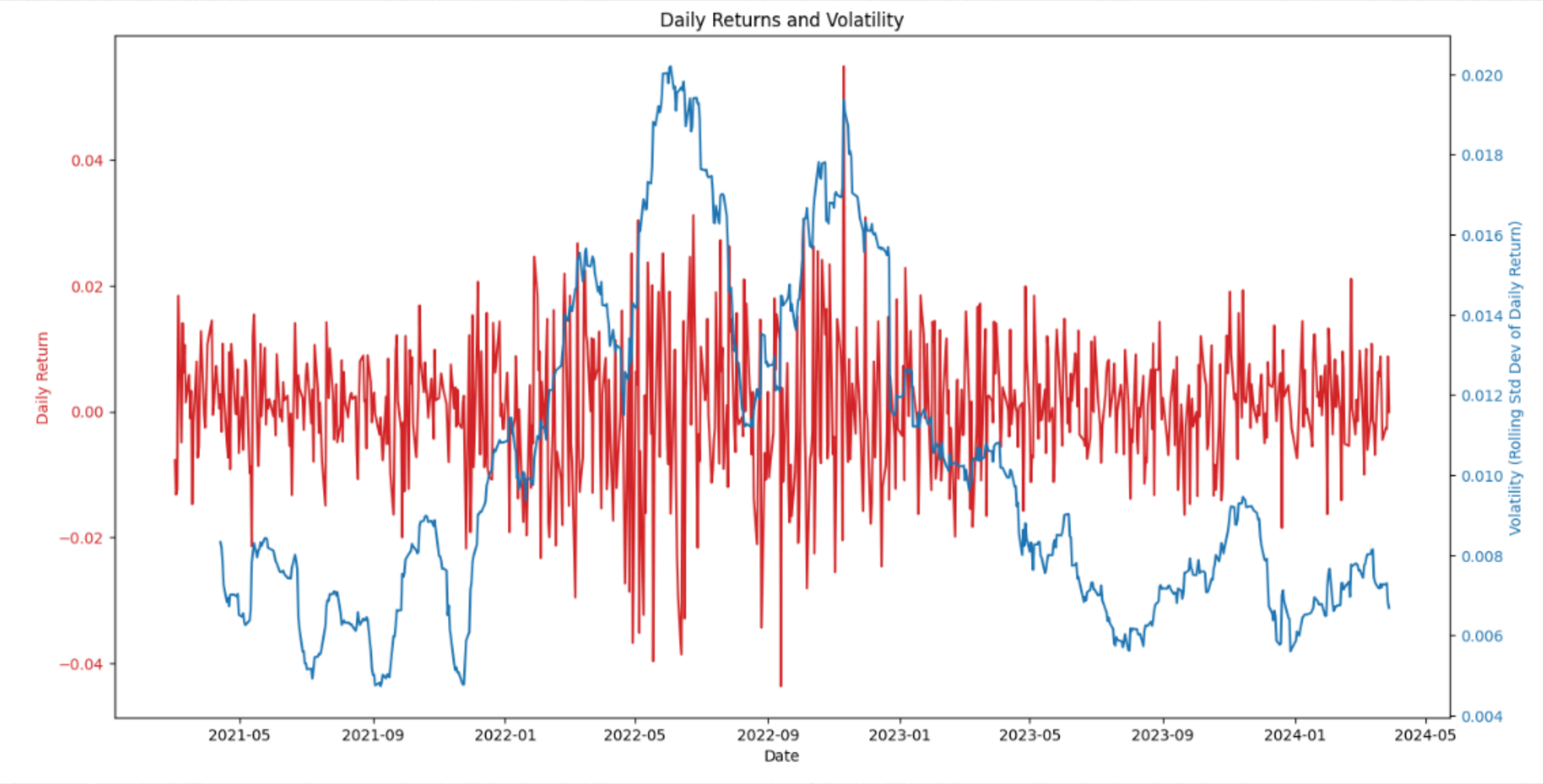


original data

short-term trend

long-term trend

Daily Returns & Volatility



Stationarity (ADF) Test

```
# Perform the Augmented Dickey-Fuller test
result = adfuller(df_close)

# Get the p-value
p_value = result[1]
print("p-value:", round(p_value, DP))
```

Result:

```
p-value: 0.7566
The series is non-stationary.
```

Core Analysis

ARIMA

Auto **R**egressive **I**ntegrated
Moving **A**verage

“AR” Component

ARIMA(**p**, d, q)

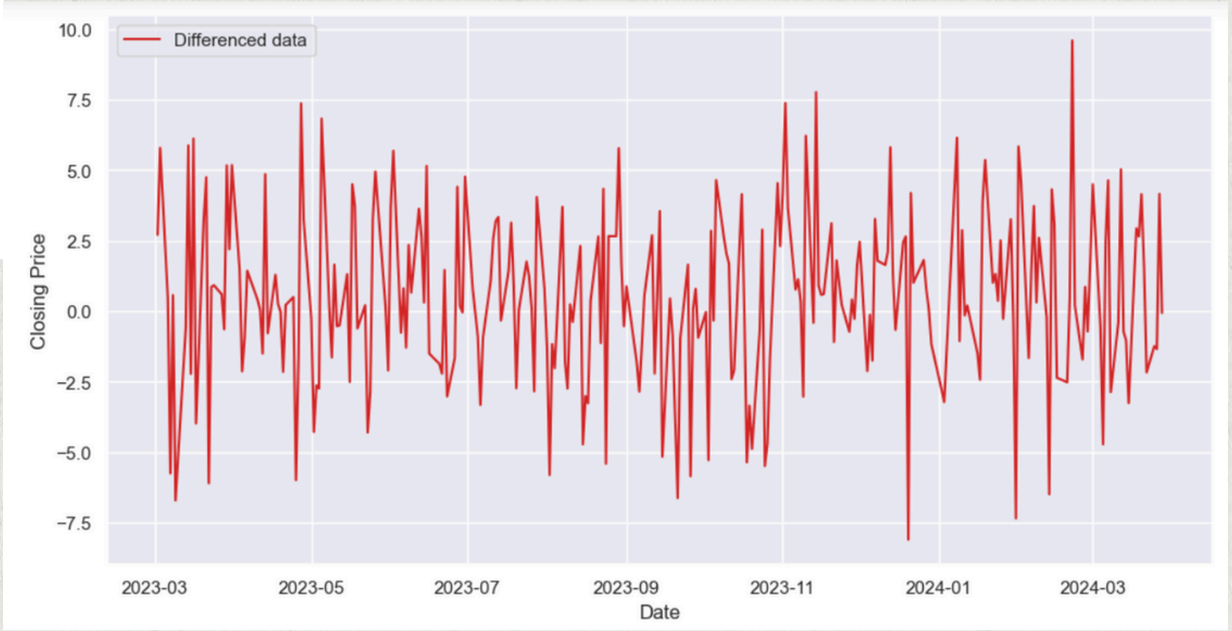
$$P_t = \beta_0 + \beta_1 P_{t-i} + \beta_2 P_{t-j} + \beta_3 P_{t-k} + \dots$$

lagged values **highly correlated** to current value

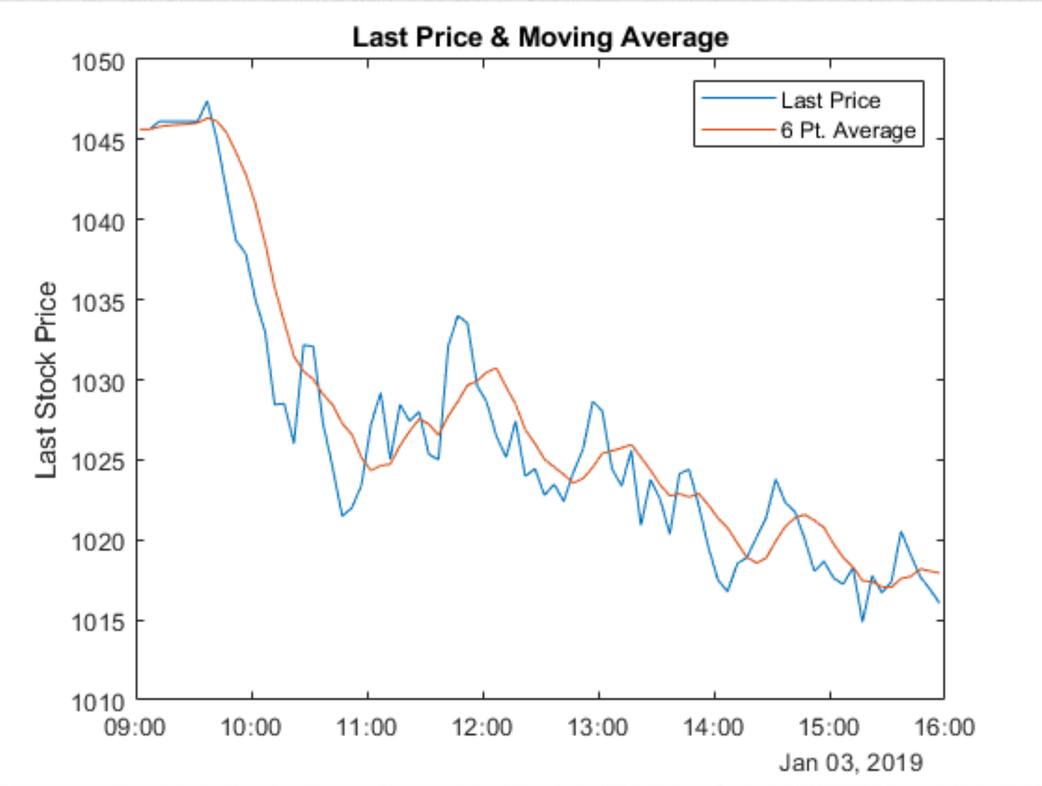
“I” Component

$$P_t = P_t - P_{t-1}$$

ARIMA(p, **d**, q)



“MA” Component



Disclaimer: Diagram not taken from project

ARIMA(p, d, **q**)

smooths out **short-term fluctuations**

Finding the best model

```
# Using Auto ARIMA to find the best model
model = auto_arima(train_df, seasonal=False, trace=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=2969.124, Time=0.40 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=2963.297, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=2965.207, Time=0.05 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=2965.198, Time=0.05 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=2961.300, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=2965.594, Time=0.16 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 0.679 seconds
```

ARIMA: Best model?

- **not ideal** for long-term forecasting
- assume **future patterns** resemble the **past**

LSTM

(Long Short-Term Memory)

Type of Recurrent Neural Network

LSTM Data Normalisation

MinMaxScaler to normalize the 'Close' prices between 0 and 1, which ensures stable training and better convergence in our LSTM neural network by standardizing the input scale

```
# Normalize the data
scaler = MinMaxScaler(feature_range=(0,1))
scaled_train_data = scaler.fit_transform(train_df['Close'].values. Reshape(-1,1))
```

Training Data Preparation

This window size helps the model to 'remember' and 'learn' from the last 50 days of stock price movements, which includes short-term fluctuations and longer-term trends.

```
# Prepare the training data with a specified number of days to use for the prediction
N_PRED_DAYS = 50
x_train, y_train = [], []

for d in range(N_PRED_DAYS, len(scaled_train_data)):
    x_train.append(scaled_train_data[d - N_PRED_DAYS:d, 0])
    y_train.append(scaled_train_data[d, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

LSTM Model Building

LSTM model features multiple layers, each with 50 units and dropout techniques to prevent overfitting, enhancing the model's generalizability.

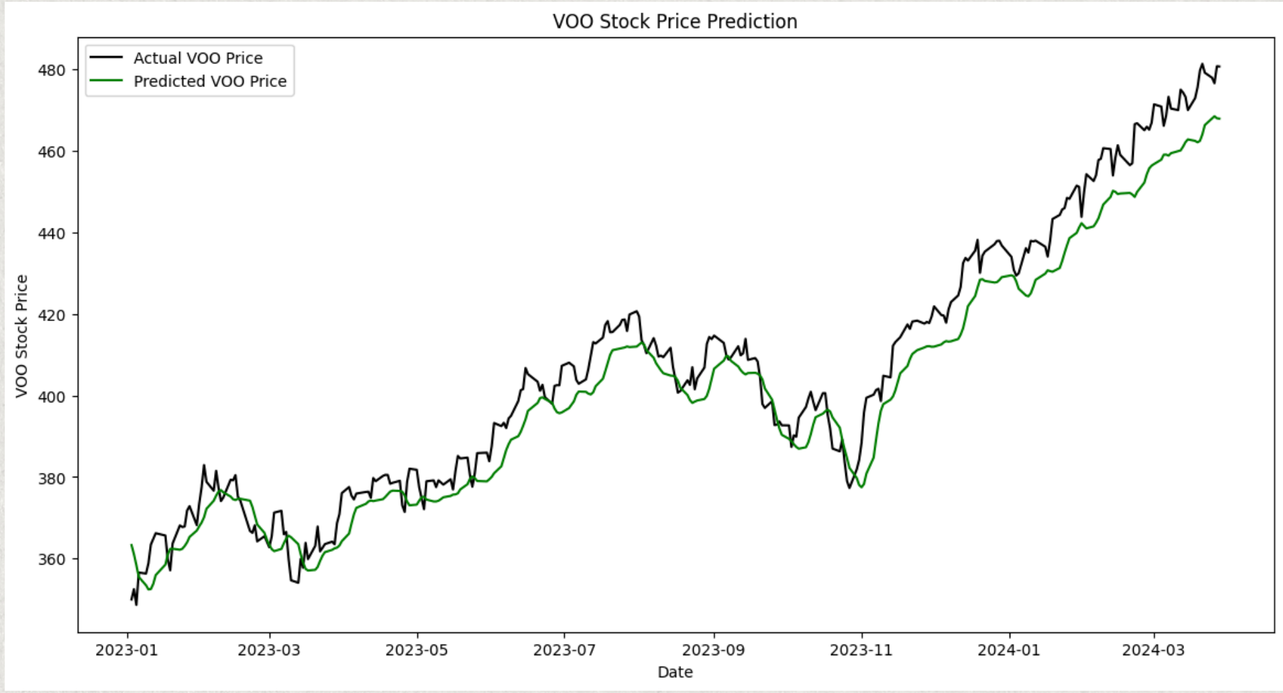
```
# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1))
```

The training showed a consistent decrease in loss, validating the model's learning efficiency.

```
Epoch 33/35
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 0.0073
Epoch 34/35
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0079
Epoch 35/35
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0074
<keras.src.callbacks.history.History at 0x22533b6dd90>
```

Prediction & Visualisation

LSTM relies on past patterns to predict future ones, which is not always the case in volatile and adaptive stock markets. External factors such as market sentiment or political events can dramatically influence stock prices.



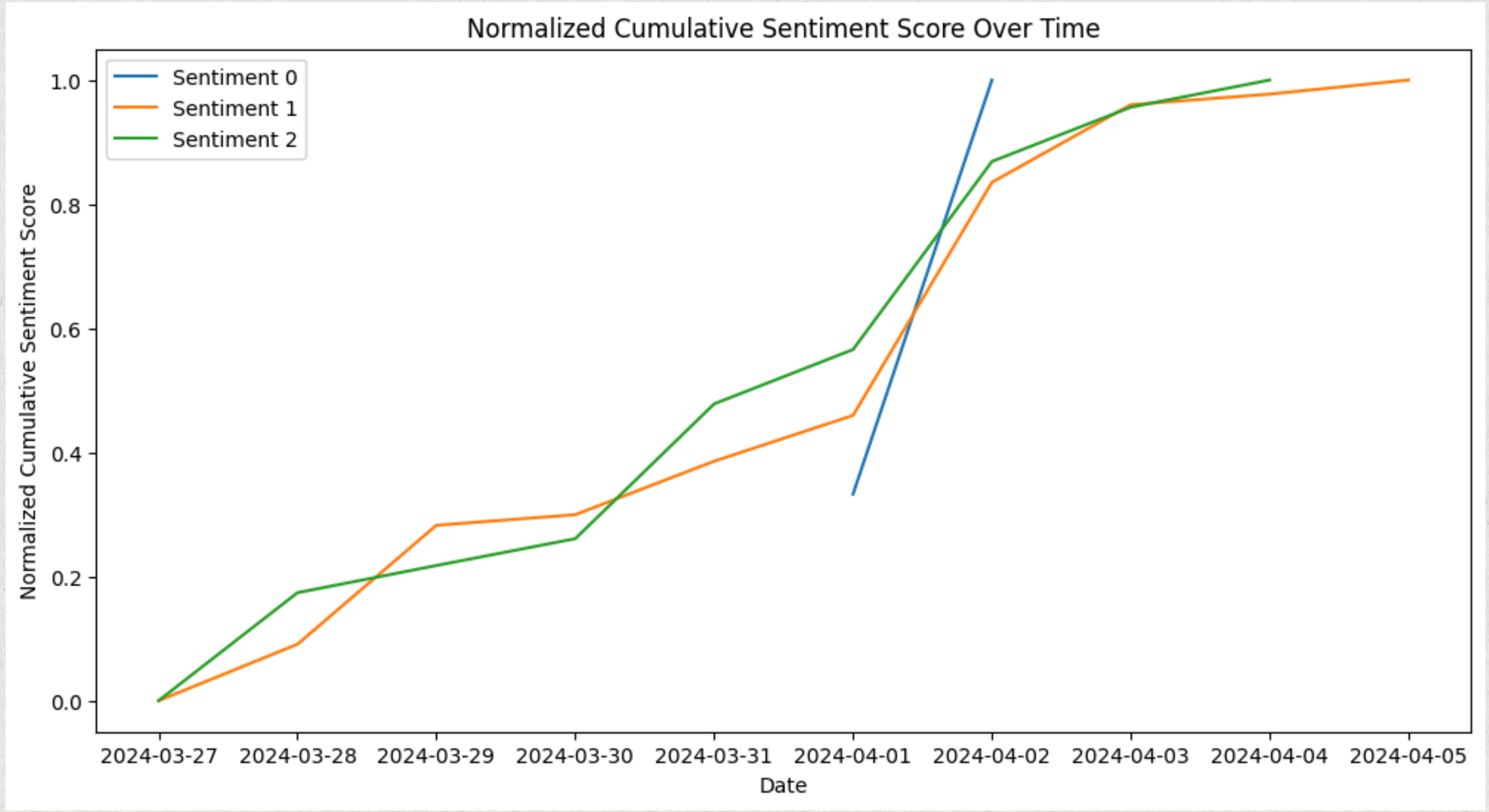
Sentiment Analysis

Sentiment Analysis

Classify Sentiment

```
model_name = "yiyanghkust/finbert-tone"
sentiment_pipeline = pipeline("sentiment-analysis", model=model_name)
def classify_sentiment(text):
    result = sentiment_pipeline(text)[0]
    sentiment_labels = {'Positive': 2, 'Neutral': 1, 'Negative': 0}
    label = sentiment_labels.get(result['label'], -1)
    score = result['score']
    return label, score
```

Normalized Cumulative Sentiment Over Time



Analyzing the Overall Sentiment

Normalized Net Sentiment Score

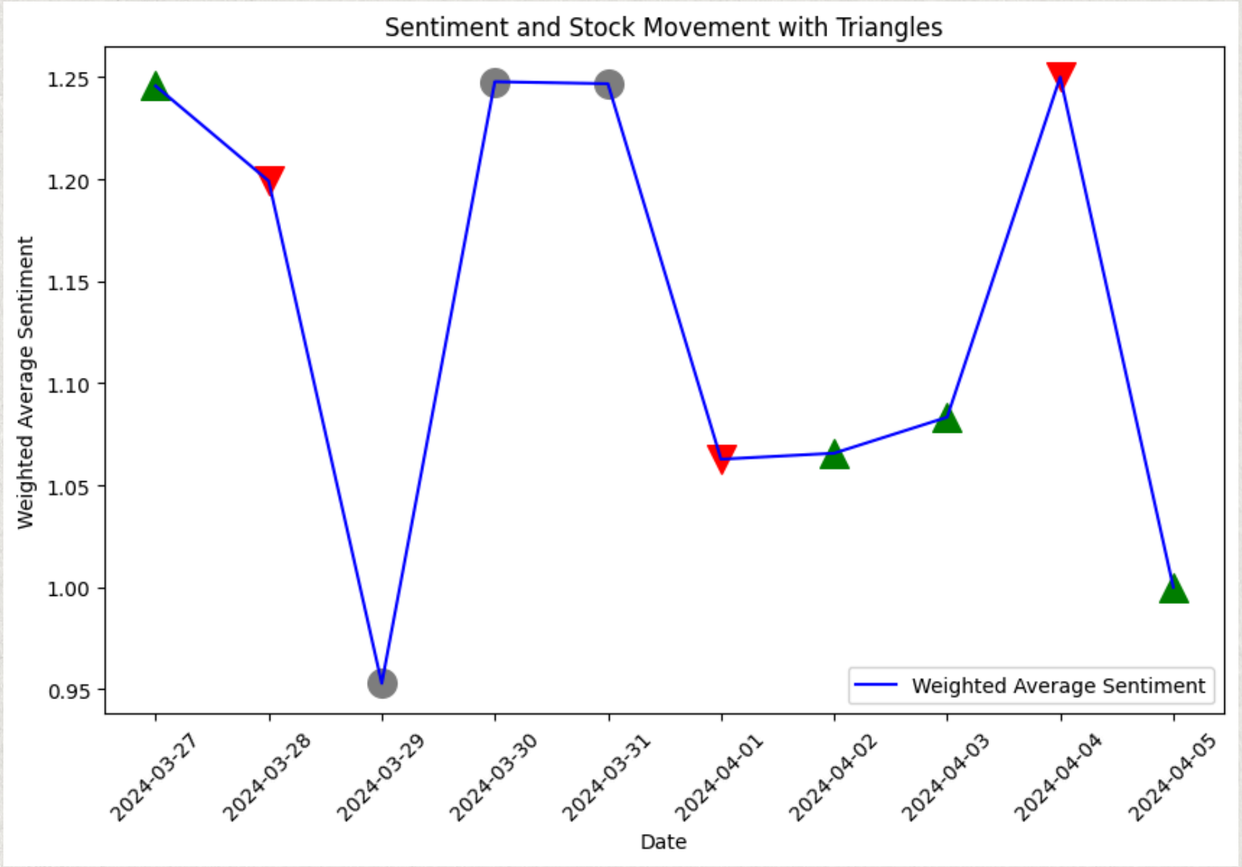
```
nss = nss_df.groupby('date').apply(
    lambda x: x[x['sentiment_label'] == 2]['sentiment_score'].sum()
              - x[x['sentiment_label'] == 0]['sentiment_score'].sum()
)
...
scaler = MinMaxScaler(feature_range=(0, 1))
nss_df['normalized_nss'] = scaler.fit_transform(nss_df['nss'].values.reshape(-1, 1))
```

Weighted Average

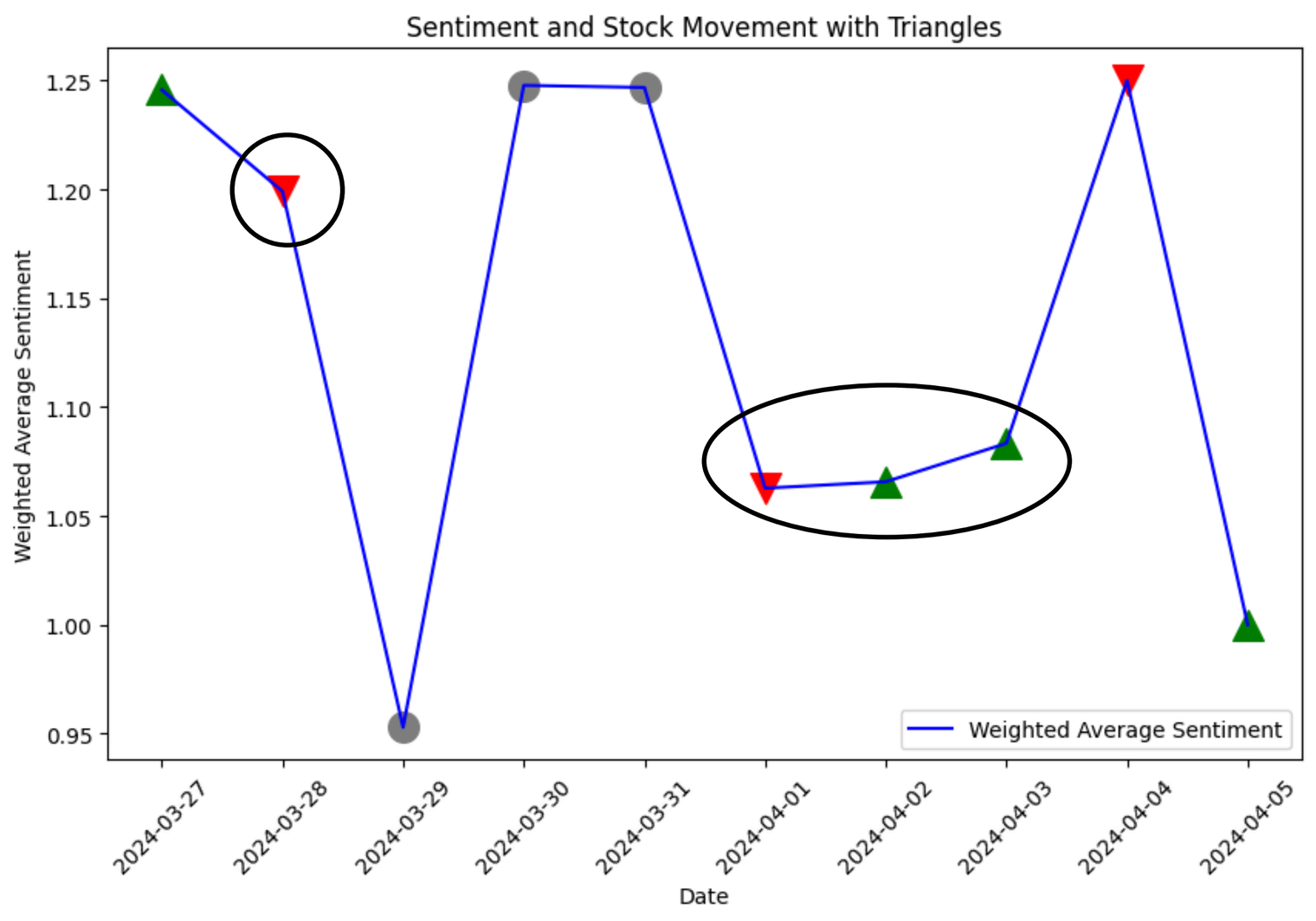
```
weighted_avg = weighted_avg_df.groupby('date').apply(
    lambda x: (x['sentiment_label'] * x['sentiment_score']).sum() / x['sentiment_score'].sum())
```

Visualizing Weighted Average Sentiment with Stock Movement

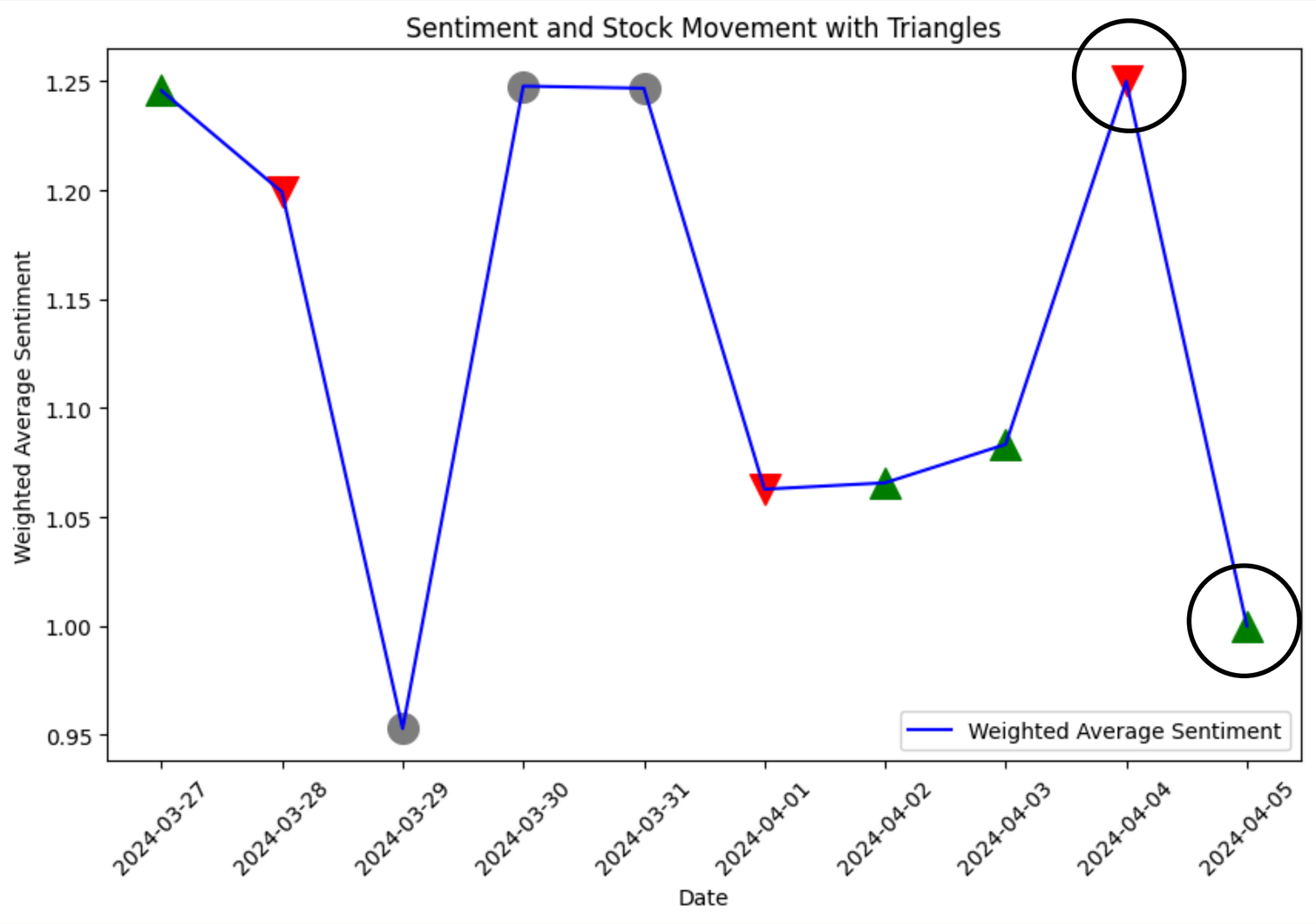
```
plt.figure(figsize=(10, 6))
plt.plot(weighted_avg_df['date'], weighted_avg_df['weighted_average'], label='Weighted Average
Sentiment', color='blue')

for i, row in weighted_avg_df.iterrows():
    if row['actual_label'] == 1:
        plt.scatter(row['date'], row['weighted_average'], color='green', marker='^', s=200)
    elif row['actual_label'] == 0:
        plt.scatter(row['date'], row['weighted_average'], color='red', marker='v', s=200)
    else:
        plt.scatter(row['date'], row['weighted_average'], color='grey', marker='o', s=200)
```



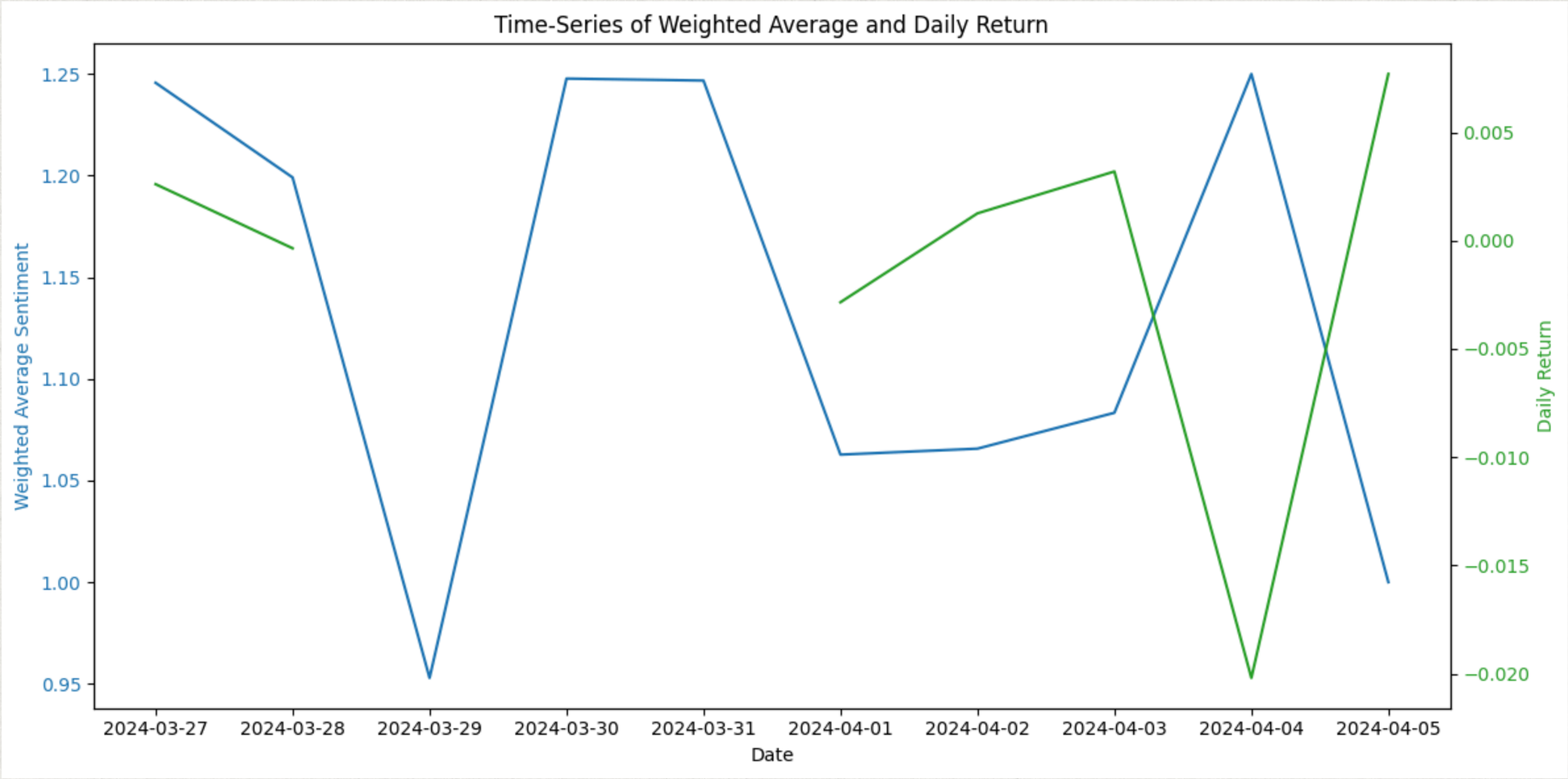
Alignments


Sentiment and Stock Movement with Triangles
Weighted Average Sentiment
1.25
1.20
1.15
1.10
1.05
1.00
0.95
2024-03-27
2024-03-28
2024-03-29
2024-03-30
2024-03-31
2024-04-01
2024-04-02
2024-04-03
2024-04-04
2024-04-05
Date
Weighted Average Sentiment

Anomalies

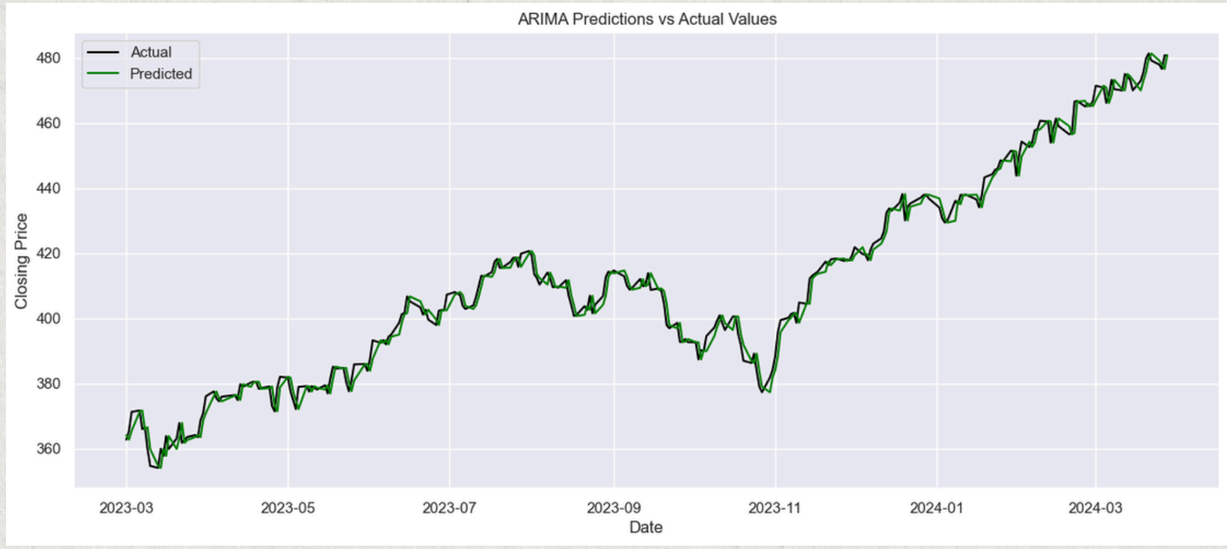

Sentiment and Stock Movement with Triangles
Weighted Average Sentiment
1.25
1.20
1.15
1.10
1.05
1.00
0.95
2024-03-27
2024-03-28
2024-03-29
2024-03-30
2024-03-31
2024-04-01
2024-04-02
2024-04-03
2024-04-04
2024-04-05
Date
Weighted Average Sentiment

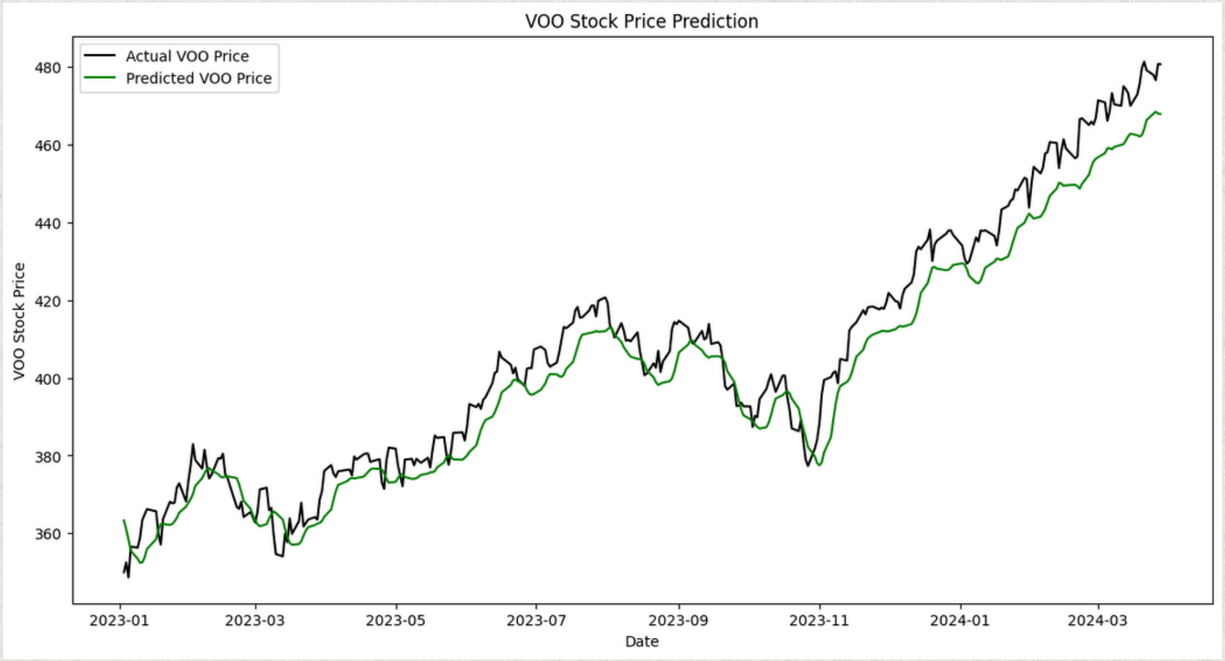
Dual Y-Axis Time-Series Plot for Weighted Average Sentiment and Daily Stock Returns



Outcomes and Insights

ARIMA Model





LSTM Model

Model Comparison and Evaluation

Metric	ARIMA	LSTM
Mean Squared Error (MSE)	9.6799	63.41
Root Mean Squared Error (RMSE)	3.1113 (0.7572%)	7.9676 (1.96%)
Mean Absolute Percentage Error (MAPE)	0.006%	1.64%
Coefficient of Determination (R squared)	0.9901	0.9406

Recommendations

ARIMA

while most accurate but assumes linearity and stationarity in the time series data

IF

- clear trend or seasonal patterns
- easier and faster to develop

*LSTM

*increase number of days (N_PRED_DAYS) for the model to learn

IF

- access to large datasets
- consists of non-linear patterns

Presented by FCSF Team 2

May the Market be with you!