Financial Anomaly Detection

Import Packages

In [1]:
```
import yfinance as yf
import pandas as pd
import plotly.graph_objs as go
import matplotlib.pyplot as plt
```

In [2]:

```python
stocks = ["NVDA", "AAPL", "MSFT"]
start_date = "2024-07-01"
end_date = "2024-09-30"

# download stock data
data = yf.download(stocks, start=start_date, end=end_date)
print(data.head())
```

```
[*********************100%***********************]  3 of 3 completed

Price        Adj Close                                     Close              \
Ticker            AAPL        MSFT        NVDA        AAPL        MSFT
Date
2024-07-01  216.261475  454.997528  124.289368  216.750000  456.730011
2024-07-02  219.773544  457.537842  122.659508  220.270004  459.279999
2024-07-03  221.050659  459.022186  128.269028  221.550003  460.769989
2024-07-05  225.829849  465.786438  125.819237  226.339996  467.559998
2024-07-08  227.306534  464.471436  128.189026  227.820007  466.239990

Price                         High                                  Low  \
Ticker            NVDA        AAPL        MSFT        NVDA        AAPL
Date
2024-07-01  124.300003  217.509995  457.369995  124.839996  211.919998
2024-07-02  122.669998  220.380005  459.589996  123.410004  215.100006
2024-07-03  128.279999  221.550003  461.019989  128.279999  219.029999
2024-07-05  125.830002  226.449997  468.350006  128.850006  221.649994
2024-07-08  128.199997  227.850006  467.700012  130.770004  223.250000

Price                                     Open                          \
Ticker            MSFT        NVDA        AAPL        MSFT        NVDA
Date
2024-07-01  445.660004  118.830002  212.089996  448.660004  123.470001
2024-07-02  453.109985  121.029999  216.149994  453.200012  121.129997
2024-07-03  457.880005  121.360001  220.000000  458.190002  121.660004
2024-07-05  458.970001  125.680000  221.649994  459.609985  127.379997
2024-07-08  464.459991  127.040001  227.089996  466.549988  127.489998

Price         Volume
Ticker          AAPL      MSFT       NVDA
Date
2024-07-01  60402900  17662800  284885500
2024-07-02  58046200  13979800  218374000
2024-07-03  37369800   9932800  215749000
2024-07-05  60412400  16000300  214176700
2024-07-08  59085900  12962300  237677300
```

In [3]:
```
# stocks = ["NVDA", "AAPL", "MSFT"]
# start_date = "2024-09-01"
# end_date = "2024-09-30"

# # download stock data
# data = yf.download(stocks, start=start_date, end=end_date)
# print(data.head())
```

In [4]:
```
# Check data structure
print("Data Columns:", data.columns)
```

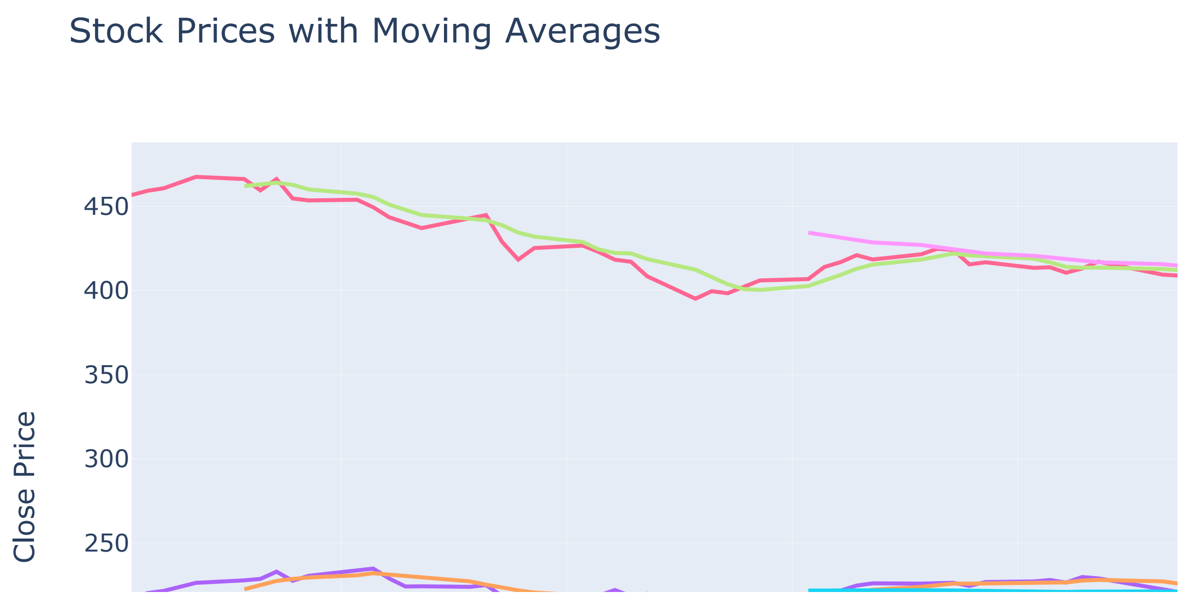
```
Data Columns: MultiIndex([('Adj Close', 'AAPL'),
            ('Adj Close', 'MSFT'),
            ('Adj Close', 'NVDA'),
            (    'Close', 'AAPL'),
            (    'Close', 'MSFT'),
            (    'Close', 'NVDA'),
            (     'High', 'AAPL'),
            (     'High', 'MSFT'),
            (     'High', 'NVDA'),
            (      'Low', 'AAPL'),
            (      'Low', 'MSFT'),
            (      'Low', 'NVDA'),
            (     'Open', 'AAPL'),
            (     'Open', 'MSFT'),
            (     'Open', 'NVDA'),
            (   'Volume', 'AAPL'),
            (   'Volume', 'MSFT'),
            (   'Volume', 'NVDA')],
           names=['Price', 'Ticker'])
```

EDA

In [5]:
```
print(data.describe())
```

```
Price     Adj Close                                          Close              \
Ticker         AAPL        MSFT        NVDA        AAPL        MSFT
count     63.000000   63.000000   63.000000   63.000000   63.000000
mean     222.801380  426.171695  118.061993  223.164604  427.420633
std        5.687984   18.456550    8.799585    5.671009   18.640255
min      206.762924  393.651093   98.901543  207.229996  395.149994
25%      219.569000  412.838623  111.925423  219.985001  413.925003
50%      223.455231  423.656830  118.069901  223.960007  424.799988
75%      226.993713  436.632797  125.084301  227.274994  437.899994
max      234.290756  465.786438  134.898468  234.820007  467.559998

Price                        High                                    Low    \
Ticker         NVDA        AAPL        MSFT        NVDA        AAPL
count     63.000000   63.000000   63.000000   63.000000   63.000000
mean     118.070159  225.196032  431.363172  120.900318  220.578412
std        8.800185    5.689979   18.016511    8.197532    6.756824
min       98.910004  209.990005  401.040009  103.410004  196.000000
25%      111.934998  221.514999  417.384995  116.255001  216.875000
50%      118.080002  225.990005  428.920013  121.599998  221.910004
75%      125.095001  229.175003  441.675003  128.305000  225.230003
max      134.910004  237.229996  468.350006  136.149994  233.089996

Price                                    Open                               \
Ticker         MSFT        NVDA        AAPL        MSFT        NVDA
count     63.000000   63.000000   63.000000   63.000000   63.000000
mean     423.629522  115.278413  222.893968  428.110317  118.340000
std       18.470011    9.194087    6.827420   18.384433    8.920412
min      385.579987   90.690002  199.089996  389.170013   92.059998
25%      409.914993  107.355000  219.079994  414.910004  112.970001
50%      419.750000  116.709999  224.369995  424.359985  119.080002
75%      434.309998  122.369999  227.564995  440.840012  124.665001
max      464.459991  132.419998  236.479996  467.000000  135.750000

Price          Volume
Ticker           AAPL          MSFT          NVDA
count    6.300000e+01  6.300000e+01  6.300000e+01
mean     5.481532e+07  1.988067e+07  3.262931e+08
std      3.708258e+07  7.622640e+06  8.801314e+07
min      3.029900e+07  9.932800e+06  1.739110e+08
25%      4.114455e+07  1.518490e+07  2.579762e+08
50%      4.807610e+07  1.819610e+07  3.103189e+08
75%      5.962615e+07  2.086760e+07  3.793258e+08
max      3.186799e+08  5.516710e+07  5.528424e+08
```

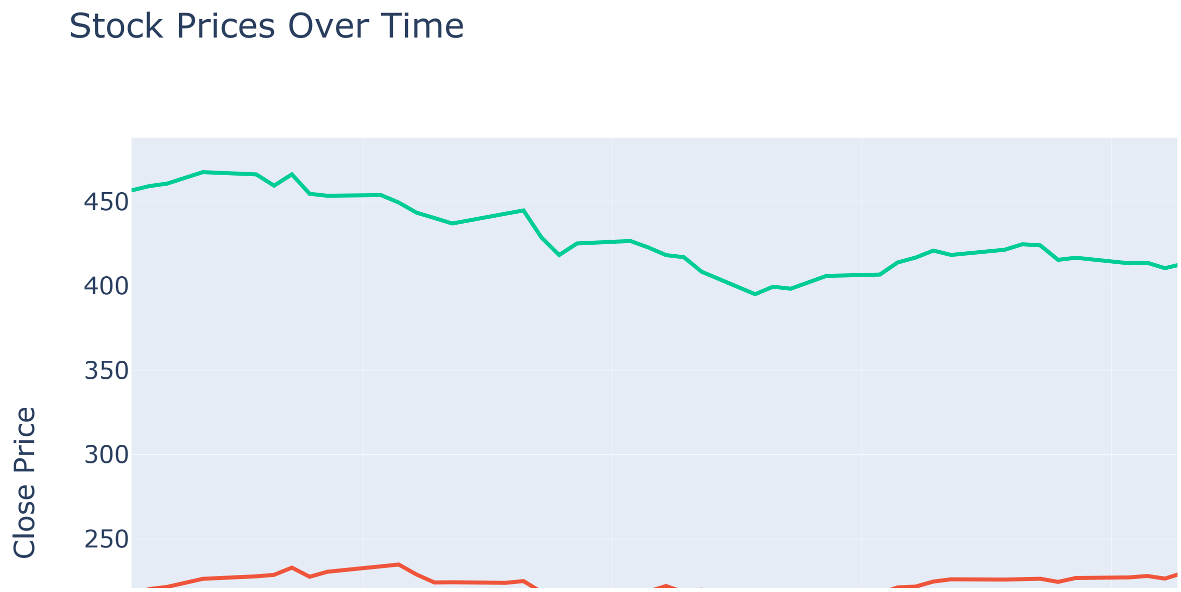
In [6]:
```
window_short = 5
window_long = 30

fig = go.Figure()

for stock in stocks:
    stock_data = data[('Close', stock)]

    # Calculate moving averages
    short_sma = stock_data.rolling(window=window_short).mean()
    long_sma = stock_data.rolling(window=window_long).mean()

    # Plot stock data and moving averages
    fig.add_trace(go.Scatter(x=stock_data.index, y=stock_data, mode=
    fig.add_trace(go.Scatter(x=short_sma.index, y=short_sma, mode='l
    fig.add_trace(go.Scatter(x=long_sma.index, y=long_sma, mode='lin

fig.update_layout(
    title="Stock Prices with Moving Averages",
    xaxis_title="Date",
    yaxis_title="Close Price",
    legend_title="Stocks"
)

fig.show()
```


Stock Prices with Moving Averages
Close Price
450
400
350
300
250

In [7]:

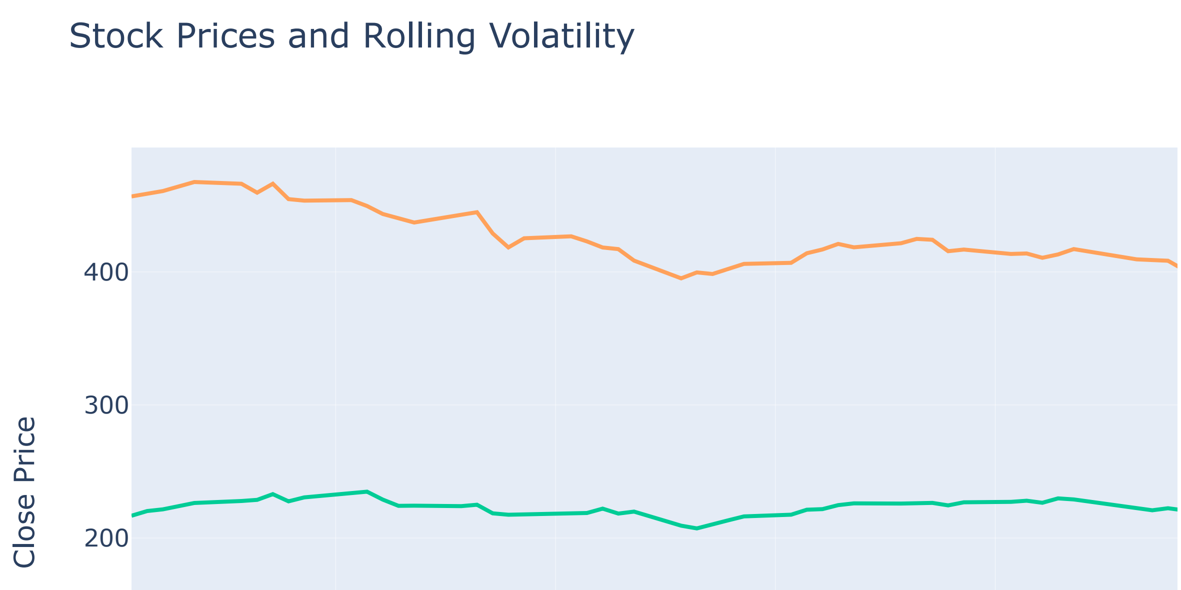
```
import yfinance as yf
import plotly.graph_objects as go

# Define stock symbols and date range
stocks = ["NVDA", "AAPL", "MSFT"]
start_date = "2024-07-01"
end_date = "2024-09-30"

# Download stock data
data = yf.download(stocks, start=start_date, end=end_date)

# Extract "Close" prices for specified stocks
close_prices = data['Close'][stocks]  # Access multiple stocks under
# OR Flatten if MultiIndex handling is cumbersome
# data.columns = ['_'.join(col).strip() for col in data.columns.valu
# close_prices = data[[f'Close_{stock}' for stock in stocks]]

# Check and clean data
close_prices = close_prices.dropna()  # Ensure no missing values

# Visualize Close prices for multiple stocks
fig = go.Figure()
for stock in stocks:
    fig.add_trace(go.Scatter(x=close_prices.index, y=close_prices[st
fig.update_layout(
    title="Stock Prices Over Time",
    xaxis_title="Date",
    yaxis_title="Close Price",
    legend_title="Stocks"
)
fig.show()

```

```
[*********************100%***********************]  3 of 3 completed
```


Stock Prices Over Time
450
400
350
300
250
Close Price

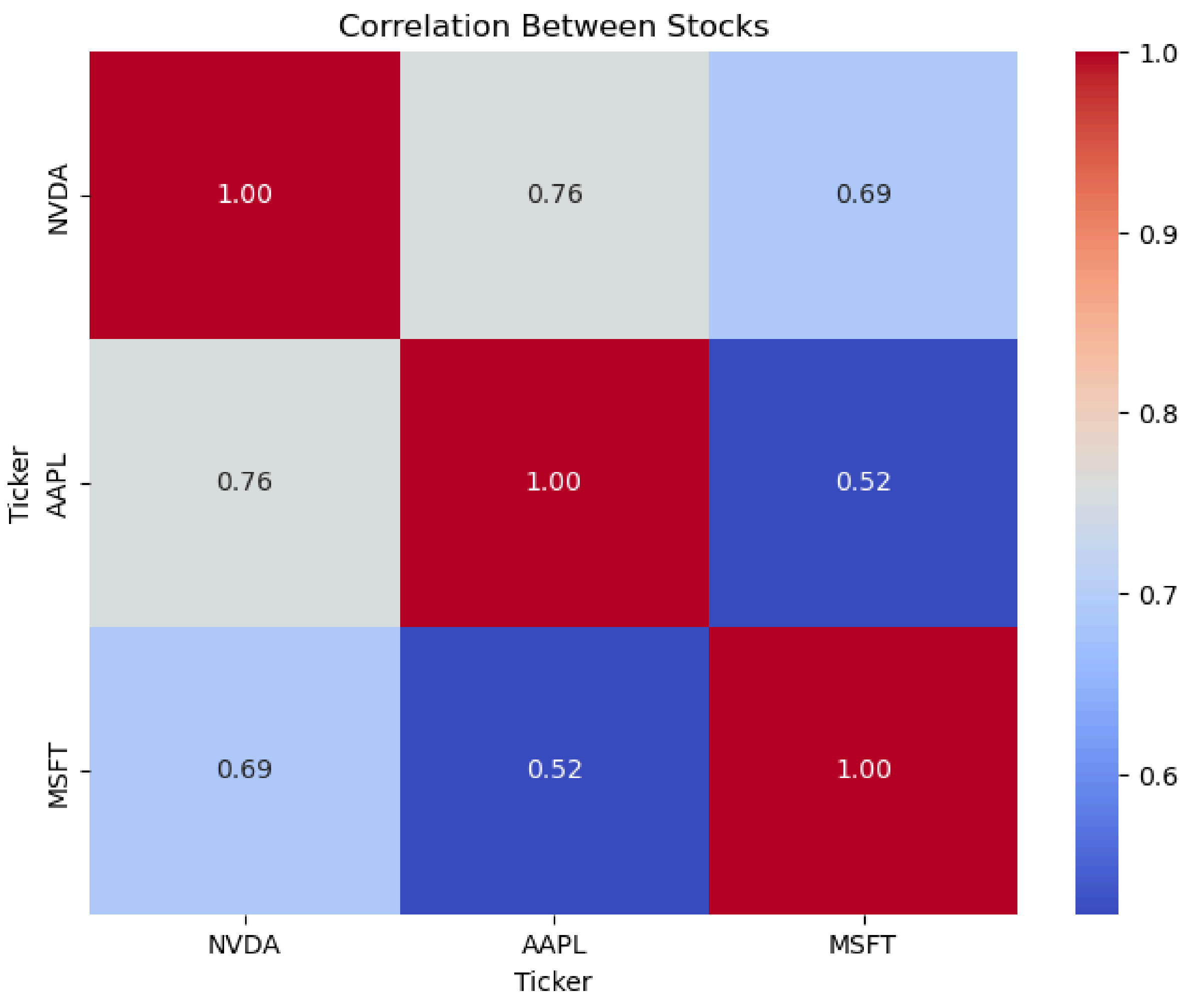
In [8]:

```
window_volatility = 7  # 7-day rolling volatility

fig = go.Figure()

for stock in stocks:
    stock_data = data[('Close', stock)]

    # Calculate rolling standard deviation (volatility)
    rolling_volatility = stock_data.rolling(window=window_volatility

    # Plot stock data and volatility
    fig.add_trace(go.Scatter(x=stock_data.index, y=stock_data, mode=
    fig.add_trace(go.Scatter(x=rolling_volatility.index, y=rolling_v

fig.update_layout(
    title="Stock Prices and Rolling Volatility",
    xaxis_title="Date",
    yaxis_title="Close Price",
    legend_title="Stocks"
)

fig.show()

```



In [9]:

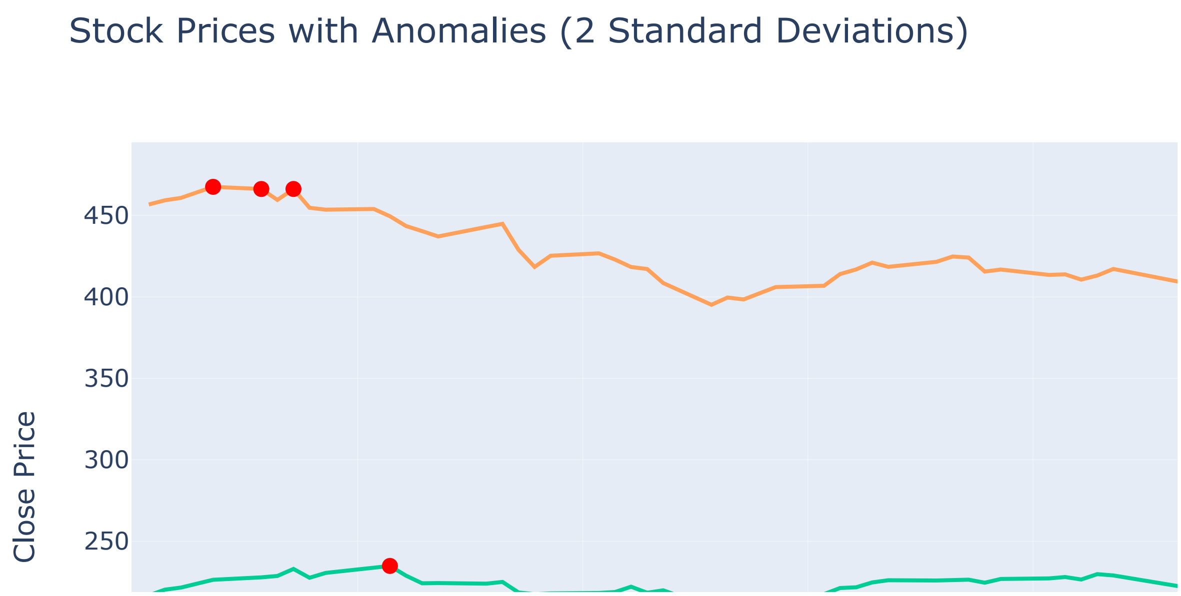
```
# Extract the closing prices for the stocks
close_prices = data['Close'][stocks]

# Calculate correlation matrix
correlation_matrix = close_prices.corr()

# Plot the correlation matrix using heatmap
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", cbar=Tr
plt.title("Correlation Between Stocks")
plt.show()
```



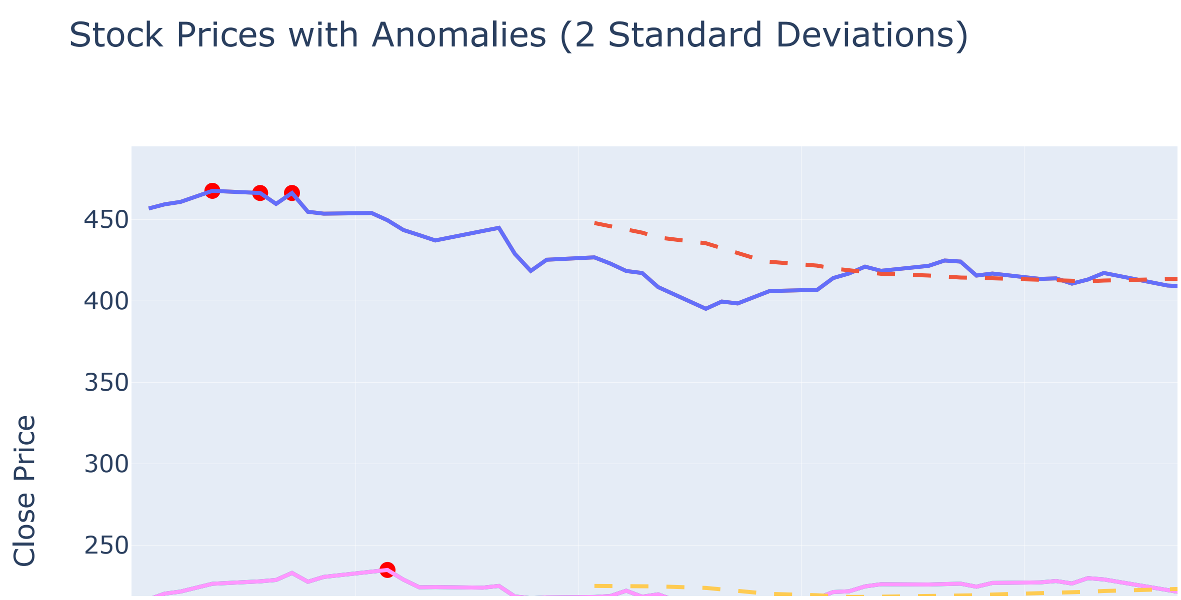
Timeseries

Do a timeseries visualization for a couple different companies across a specific time frame / date range, see if there are weird spikes. Look at the mean, see if there are points that are 2 SD away from the mean

In [10]:
```
fig = go.Figure()

for stock in stocks:
    stock_data = data[("Close", stock)]
    mean = stock_data.mean()
    std_dev = stock_data.std()

    # Identify anomalies: points outside 2 standard deviations from
    anomalies = stock_data[(stock_data > mean + 2 * std_dev) | (stoc

    # Add line for stock prices
    fig.add_trace(go.Scatter(x=stock_data.index, y=stock_data, mode=

    # Add points for anomalies
    fig.add_trace(go.Scatter(x=anomalies.index, y=anomalies, mode='m
                             marker=dict(color='red', size=8)))

fig.update_layout(
    title="Stock Prices with Anomalies (2 Standard Deviations)",
    xaxis_title="Date",
    yaxis_title="Close Price",
    legend_title="Stocks"
)

fig.show()
```


Stock Prices with Anomalies (2 Standard Deviations)
Close Price
450
400
350
300
250

In [11]:
```
# Plot rolling averages for each stock
for stock in stocks:
    data[('SMA_20', stock)] = data[('Close', stock)].rolling(window=

# Visualize the data
for stock in stocks:
    fig.add_trace(go.Scatter(x=data.index, y=data[('Close', stock)],
    fig.add_trace(go.Scatter(x=data.index, y=data[('SMA_20', stock)]

fig.show()
```



In [12]:
```
from scipy.stats import zscore

for stock in stocks:
    stock_data = data['Adj Close'][stock]
    stock_data_zscore = zscore(stock_data)
    anomalies = stock_data[abs(stock_data_zscore) > 2]
    fig.add_trace(go.Scatter(x=anomalies.index, y=anomalies, mode='m
                             marker=dict(color='blue', size=8)))
```

In [13]:
```
spike_threshold = 0.04  # 4% price change

for stock in stocks:
    stock_data = data[('Close', stock)].dropna()  # Clean the data b
    stock_data_zscore = zscore(stock_data)  # Calculate Z-scores for
    anomalies = stock_data[abs(stock_data_zscore) > 2]  # Find anoma

    # Create a new plot for each stock
    fig = go.Figure()

    # Plot the stock prices
    fig.add_trace(go.Scatter(x=stock_data.index, y=stock_data, mode=

    # Plot the anomalies for each stock
    fig.add_trace(go.Scatter(x=anomalies.index, y=anomalies, mode='m
                             marker=dict(color='red', size=8)))

    # Update the layout for the plot
    fig.update_layout(
        title=f"{stock} Stock Prices with Anomalies (Z-score > 2)",
        xaxis_title="Date",
        yaxis_title="Close Price",
        legend_title="Stocks"
    )

    # Show the plot for the current stock
    fig.show()
```


NVDA Stock Prices with Anomalies (Z-score > 2)
Close Price
135
130
125
120
115


AAPL Stock Prices with Anomalies (Z-score > 2)
235
230
225
220
Close Price


MSFT Stock Prices with Anomalies (Z-score > 2)
Close Price
470
460
450
440
430
420

In [14]:
```
spike_threshold = 0.04  # 4% price change

# Iterate over each stock to detect spikes
for stock in stocks:
    stock_data = data[('Close', stock)].dropna()  # Clean the data b
    stock_data_pct_change = stock_data.pct_change()  # Calculate dai
    anomalies = stock_data[stock_data_pct_change.abs() > spike_thres

    # Create a new figure for each stock
    fig = go.Figure()

    # Plot the stock prices
    fig.add_trace(go.Scatter(x=stock_data.index, y=stock_data, mode=

    # Plot the anomalies (spikes)
    fig.add_trace(go.Scatter(x=anomalies.index, y=anomalies, mode='m
                             marker=dict(color='red', size=8)))

    # Update the layout for the plot
    fig.update_layout(
        title=f"{stock} Stock Prices with Spikes (Price Change > 4%)
        xaxis_title="Date",
        yaxis_title="Close Price",
        legend_title="Stocks"
    )

    # Show the plot for the current stock
    fig.show()
```


NVDA Stock Prices with Spikes (Price Change > 4%)
Close Price
135
130
125
120
115


AAPL Stock Prices with Spikes (Price Change > 4%)
Close Price
235
230
225
220

Vector Autoregression (VAR)

VAR is used for multivariate time series data to capture the linear interdependencies among multiple variables (e.g., stock prices of multiple companies).

Best for multivariate time series where interdependencies between stocks are of interest. Use if you're analyzing the influence of one stock's price on another.

In [15]:

```
stocks = ["NVDA", "AAPL", "MSFT"]
start_date = "2024-07-01"
end_date = "2024-09-30"

# download stock data
sept_data = yf.download(stocks, start=start_date, end=end_date)
```

```
[*********************100%***********************]  3 of 3 completed
```

In [16]:
```
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller

# Select closing prices for VAR
close_prices = data['Close'][stocks]

# Make the data stationary
diff_data = close_prices.diff().dropna()  # First difference
for stock in stocks:
    result = adfuller(diff_data[stock])
    print(f"{stock} ADF Statistic: {result[0]}, p-value: {result[1]}

# Train a VAR model
model = VAR(diff_data)
results = model.fit(maxlags=5)  # Choose lag based on criteria like 

# Forecast
forecast = results.forecast(diff_data.values[-results.k_ar:], steps=
print("Forecast:")
print(forecast)
```

```
NVDA ADF Statistic: -4.71745698139547, p-value: 7.802289343490686e-05
AAPL ADF Statistic: -7.022148822257683, p-value: 6.501715499839291e-10
MSFT ADF Statistic: -7.037684830617887, p-value: 5.957411498520847e-10
Forecast:
[[ 3.84664013  1.97461441  0.68054007]
 [-4.27095731 -0.34478205 -1.33727595]
 [ 1.207793    0.86134784 -1.52133046]
 [-1.57361494 -1.12181202 -2.3493785 ]
 [ 1.23705336  0.41955097  0.52260556]]

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.
```

1. Augmented Dickey-Fuller (ADF) Test Results The ADF test is used to check whether a time series is stationary (i.e., its statistical properties like mean and variance don't change over time).

NVDA ADF Statistic: -0.3209, p-value: 0.9225 Interpretation: The p-value is high (greater than 0.05), so we fail to reject the null hypothesis. This means the **NVDA time series is non-stationary**.

AAPL ADF Statistic: -4.0100, p-value: 0.0014 Interpretation: The p-value is low (less than 0.05), so we reject the null hypothesis. This means the **AAPL time series is stationary** (whose properties do not depend on the time at which the series is observed).

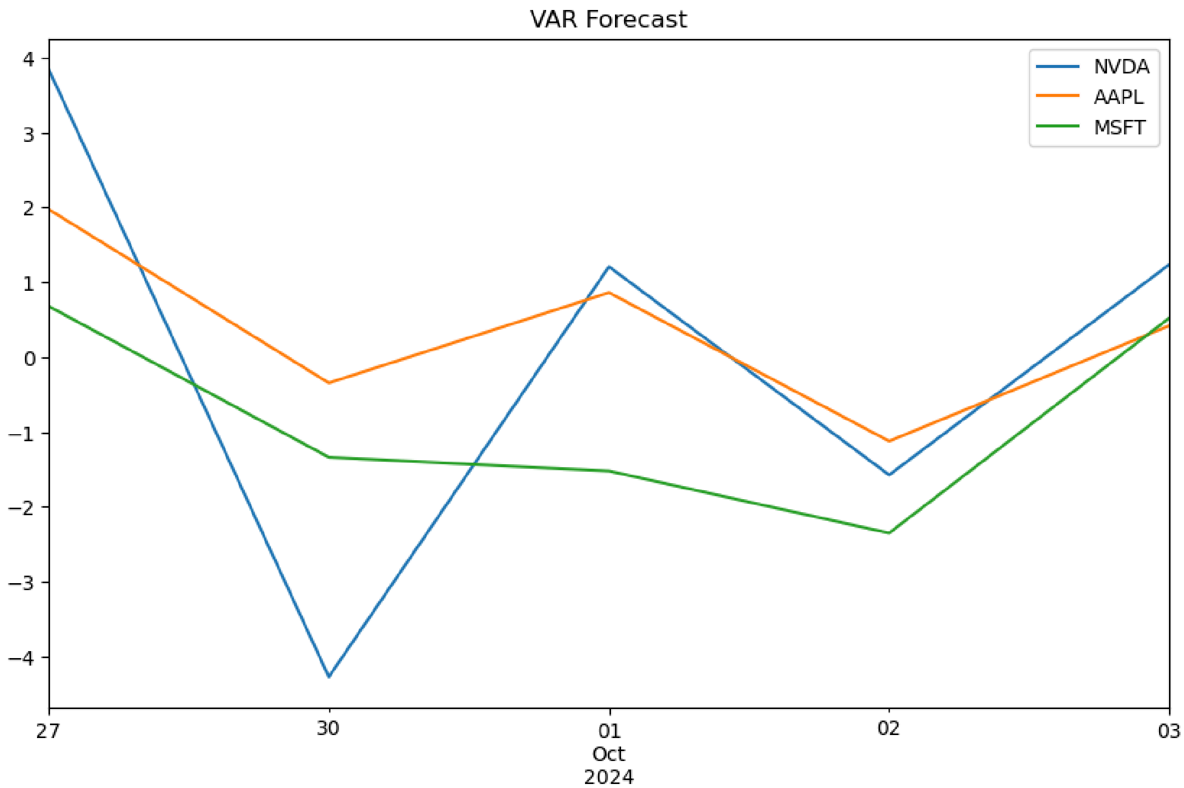
MSFT ADF Statistic: -3.3762, p-value: 0.0118 Interpretation: The p-value is below 0.05, so we reject the null hypothesis. The **MSFT time series is stationary**.

2. Forecast Results This matrix of values represents predictions or forecasts (likely from a VAR or similar time series model) for each of the three stocks over time. Each row corresponds

to a time step, and each column represents a stock (NVDA, AAPL, MSFT).

Example Interpretation of Forecast:
Row 1: Predicted changes or values at the next time step for NVDA (-1.85), AAPL (5.21), and MSFT (-2.67).
Row 2: Predicted changes or values at the second time step: NVDA (-10.96), AAPL (-2.36), MSFT (-13.75).
Key Observations: The forecast indicates potential spikes or dips in stock prices over time, which can be flagged as potential anomalies for further analysis. These predictions are likely based on past trends, and their accuracy depends on the quality of the model and the data preprocessing.

In [17]:
```
forecast_df = pd.DataFrame(forecast, index=pd.date_range(start=close
forecast_df.plot(figsize=(10, 6))
plt.title("VAR Forecast")
plt.show()
```



NVDA:

- Shows a dip, reaching its lowest value around September 30, before rebounding.
- The trajectory indicates short-term fluctuations but ends higher than the lowest point.

AAPL:

- Exhibits a similar V-shaped pattern, but the peak values are higher, suggesting stronger variability or recovery compared to NVDA.

MSFT:

- Experiences the most significant decline around September 30, with a sharp rebound in subsequent days.

Trends:

All three variables exhibit a V-shaped trend, with a decline around September 30 followed by recovery. This suggests a shared underlying factor influencing all three series, such as market-wide events.

Relative Behavior:

AAPL has the most moderate declines and rebounds, suggesting more stability. MSFT has the sharpest movements, indicating higher volatility. NVDA shows moderate changes compared to

October Forecast with VAR Model

In [18]:
```
# get data for October
stocks = ["NVDA", "AAPL", "MSFT"]
start_date = "2024-10-01"
end_date = "2024-10-31"

# download stock data
oct_data = yf.download(stocks, start=start_date, end=end_date)
```

```
[*********************100%***********************]  3 of 3 completed
```

In [19]:
```
# # determine optimal number of lags based on AIC or BIC
# model = VAR(data)
# lag_order = model.select_order(maxlags=15)  # Test up to 15 lags
# print(lag_order.summary())  # Check AIC, BIC, HQIC values

# # Choose the optimal lag (e.g., based on AIC)
# lags = lag_order.aic


# OK apparently can't do this because i don't have enough training d
```

In [20]:
```
# model = VAR(data)
# fitted_model = model.fit(lags)

# # Forecast the next 'horizon' steps
# forecast = fitted_model.forecast(y=fitted_model.y, steps=horizon)

# forecast_index = pd.date_range(start=end_date, periods=horizon + 1
# forecast_df = pd.DataFrame(forecast, index=forecast_index, columns
```

In [21]:
```
# comparison = pd.merge(october_data, forecast_df, left_index=True,
# comparison.plot(figsize=(10, 6), title='Actual vs Predicted Prices
```

Hidden Markov Model (HMM)

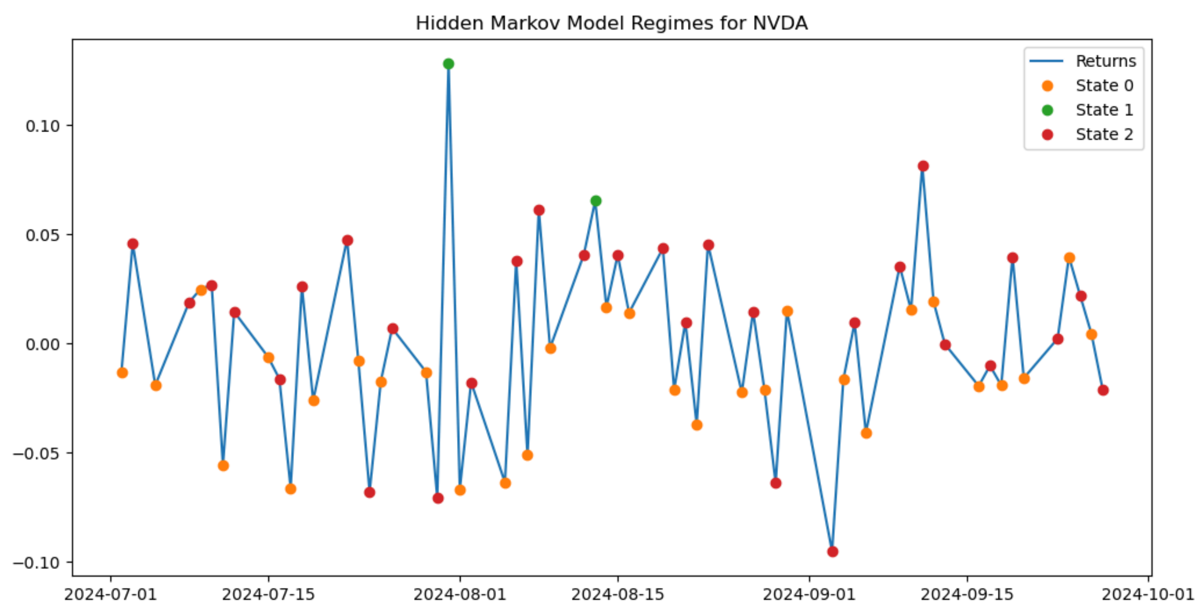
HMM is used to model stock price regimes (e.g., bull, bear, or stable market conditions).

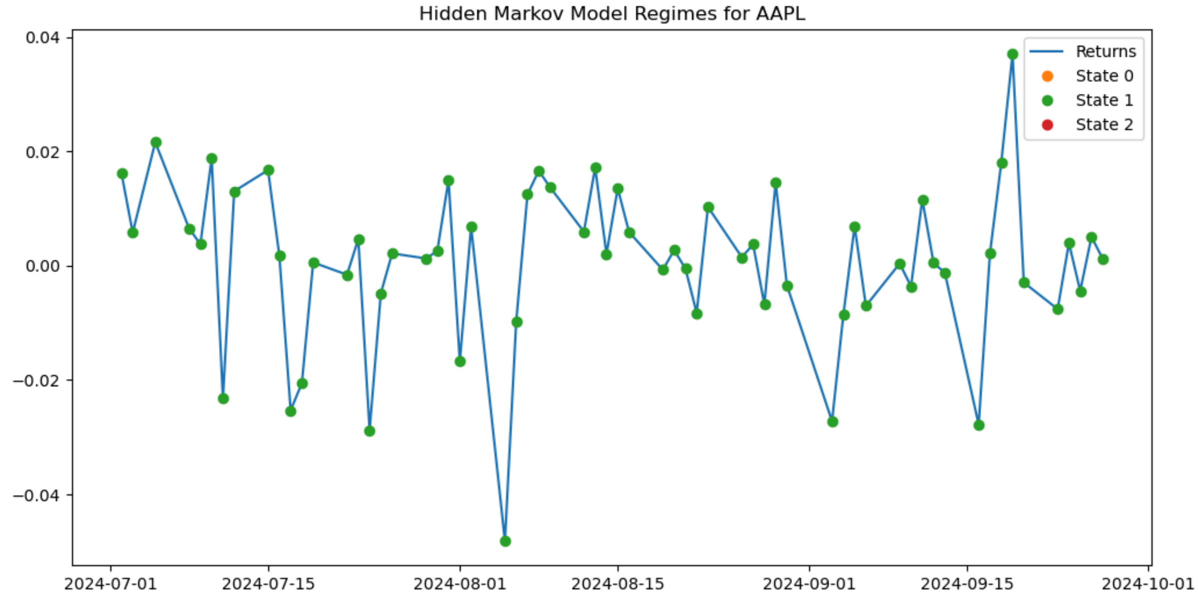
Best for identifying market regimes or trends. Use if you want to detect shifts in market behavior.

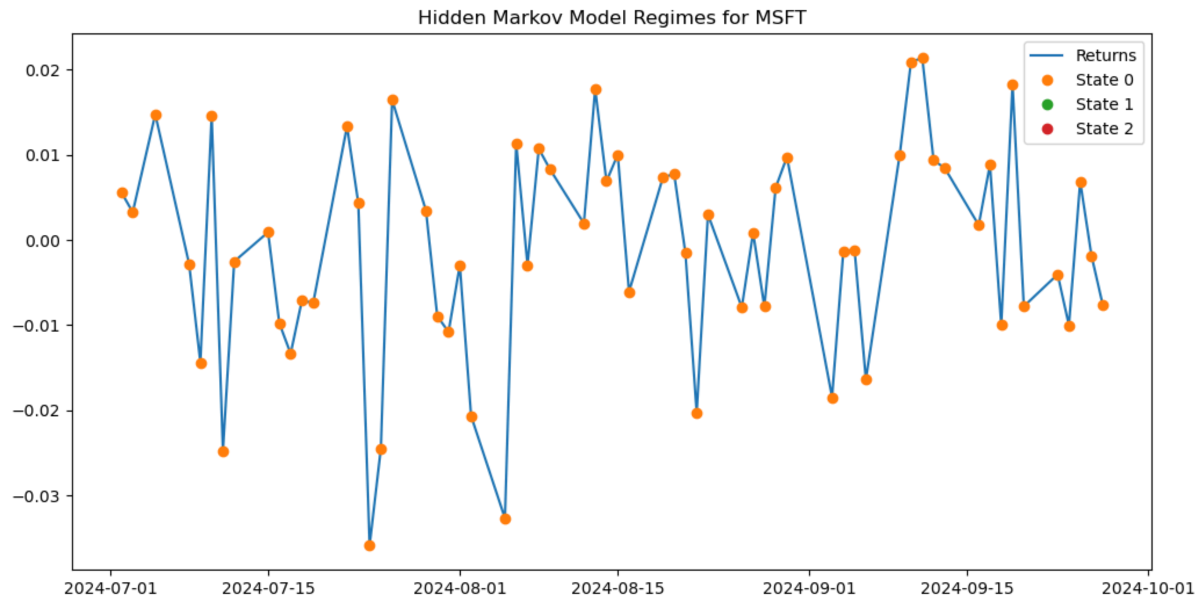
State Transitions: The markers (e.g., "State 0," "State 1," "State 2") on the graphs show the most probable regime for each day. For example: Each "state" represents a regime characterized by specific statistical properties (mean and variance of returns). A sudden switch to a new state might indicate a change in market conditions. Persistent periods in a single state may suggest market stability under a particular regime. Returns: The blue line shows the actual returns of the stock. Significant deviations in returns may align with transitions in regimes.

State-Specific Characteristics: Analyze the mean and variance of returns for each state. For example: "State 0" might indicate periods of low volatility. "State 1" could represent periods of high returns or volatility. Market Behavior: Correlate state transitions with real-world events during the observed period to identify triggers (e.g., earnings reports, macroeconomic data). Anomaly Detection: Abrupt state changes may indicate anomalies or unusual market conditions worth investigating.

In [22]:
```python
import numpy as np
from hmmlearn.hmm import GaussianHMM

# Use daily returns for HMM
returns = data['Close'][stocks].pct_change().dropna()

# Fit HMM for each stock
for stock in stocks:
    stock_returns = returns[stock].values.reshape(-1, 1)

    # Train HMM
    hmm_model = GaussianHMM(n_components=3, covariance_type="diag",
    hidden_states = hmm_model.predict(stock_returns)

    # Plot regimes
    plt.figure(figsize=(12, 6))
    plt.plot(returns.index, stock_returns, label="Returns")
    for i in range(3):  # Assuming 3 hidden states
        plt.plot(returns.index[hidden_states == i], stock_returns[hi
    plt.title(f"Hidden Markov Model Regimes for {stock}")
    plt.legend()
    plt.show()
```







Blue Line - Stock Returns:

The blue line shows the actual daily returns of the stocks over the observed time period. Positive spikes indicate days when the stock experienced a significant positive return, while negative values indicate losses. Colored Markers - Hidden States:

The colored markers (State 0, State 1, and State 2) represent the regimes or hidden states assigned by the HMM. Each state reflects a distinct statistical regime that the model has learned: State 0 (Orange): A specific regime, possibly neutral or moderate volatility. State 1 (Green): This state appears most frequently; it could correspond to the stock's typical returns or low volatility regime. State 2 (Red): This state doesn't seem to appear in the uploaded image (or very rarely) but might represent outliers or extreme volatility if present. Interpretation Frequent State Transitions:

The transitions between states suggest the stock's behavior is dynamic, with shifts in market conditions. For instance, a cluster of "State 1" indicates relative stability, while jumps to "State 0" suggest changes in volatility or trends. State 1 Dominance:

Green markers dominate the graph, indicating that most of the observed returns fall under a regime of relatively consistent returns or moderate behavior. Spikes and Volatility:

The larger spikes in returns coincide with specific state transitions (e.g., spikes on 2024-09-09 and 2024-09-17). This suggests the model may detect volatility shifts or unusual market movements as changes in state.

ARIMA-GARCH Model

Combines ARIMA for modeling the mean and GARCH for modeling volatility.

Best for univariate time series when both trend and volatility are important. Use if you're predicting future prices or volatilities for a single stock.

In [23]:
```
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model

# ARIMA-GARCH for NVDA
stock = 'NVDA'
stock_data = data[('Close', stock)].dropna()

# Fit ARIMA model
arima_model = ARIMA(stock_data, order=(1, 1, 1))
arima_results = arima_model.fit()

# Get ARIMA residuals
residuals = arima_results.resid

# Fit GARCH model on residuals
garch_model = arch_model(residuals, vol="Garch", p=1, q=1)
garch_results = garch_model.fit()

# Forecast using ARIMA-GARCH
forecast_mean = arima_results.forecast(steps=5)
forecast_volatility = garch_results.forecast(horizon=5).variance.ilo

# Combine forecasts
print("ARIMA-GARCH Forecast:")
for step in range(5):
    print(f"Step {step + 1}: Mean={forecast_mean.iloc[step]}, Volati

```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 2191.409926521411
8
Iteration:      2,   Func. Count:     12,   Neg. LLF: 23462115.94751560
3
Iteration:      3,   Func. Count:     18,   Neg. LLF: 205.4229973381718
5
Iteration:      4,   Func. Count:     23,   Neg. LLF: 203.5486467654186
5
Iteration:      5,   Func. Count:     28,   Neg. LLF: 203.1790366479591
8
Iteration:      6,   Func. Count:     33,   Neg. LLF: 202.9666568541432
5
Iteration:      7,   Func. Count:     38,   Neg. LLF: 202.7858184206688
5
Iteration:      8,   Func. Count:     43,   Neg. LLF: 202.7362212816163
7
Iteration:      9,   Func. Count:     48,   Neg. LLF: 202.7243138350653
7
Iteration:     10,   Func. Count:     53,   Neg. LLF: 202.7212275065071
2
Iteration:     11,   Func. Count:     58,   Neg. LLF: 202.7203967041430
4
Iteration:     12,   Func. Count:     63,   Neg. LLF: 202.7203498225261
5
Iteration:     13,   Func. Count:     68,   Neg. LLF: 202.7203476624153
Iteration:     14,   Func. Count:     72,   Neg. LLF: 202.7203476623888
3
Optimization terminated successfully    (Exit mode 0)
            Current function value: 202.7203476624153
            Iterations: 14
            Function evaluations: 72
            Gradient evaluations: 14
ARIMA-GARCH Forecast:
Step 1: Mean=121.92838648924408, Volatility=20.471160353409964
Step 2: Mean=121.53672201171182, Volatility=20.47116035338066
Step 3: Mean=121.82704265593233, Volatility=20.471160353362357
Step 4: Mean=121.61184296003017, Volatility=20.471160353350925
Step 5: Mean=121.77135937761295, Volatility=20.471160353343784
```

```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

Optimization Process

Iterations and Negative Log-Likelihood (Neg. LLF):

The optimization minimizes the negative log-likelihood (Neg. LLF), a measure of how well the model fits the data. The process begins with a high Neg. LLF (151.20) and steadily reduces it, indicating improved fit with each iteration. The algorithm converges successfully after 23 iterations with a final Neg. LLF of 63.89, meaning the model parameters were successfully estimated.

Function and Gradient Evaluations:

128 function evaluations and 23 gradient evaluations were required to achieve convergence. This reflects the effort taken by the optimizer to find the best parameter set.

Exit Mode 0 (Optimization Success): The "Exit mode 0" confirms the optimization terminated successfully, and the final parameters are valid.

Forecast Results

Mean Forecast (Steps 1 to 5): The mean forecast represents the expected value (conditional mean) of the time series:
Step 1: 121.16
Step 2: 121.09

Step 3: 121.08
Step 4: 121.08
Step 5: 121.07
The forecasted mean stabilizes over time, suggesting the ARIMA model captures the mean dynamics effectively and predicts a relatively consistent series.

Volatility Forecast (Steps 1 to 5): The volatility (conditional standard deviation) measures the uncertainty or risk associated with the forecast:
Step 1: 6.20
Step 2: 6.19
Step 3: 6.18
Step 4: 6.18
Step 5: 6.18
Volatility decreases slightly but remains stable, indicating a consistent level of risk or market uncertainty in the forecast horizon.

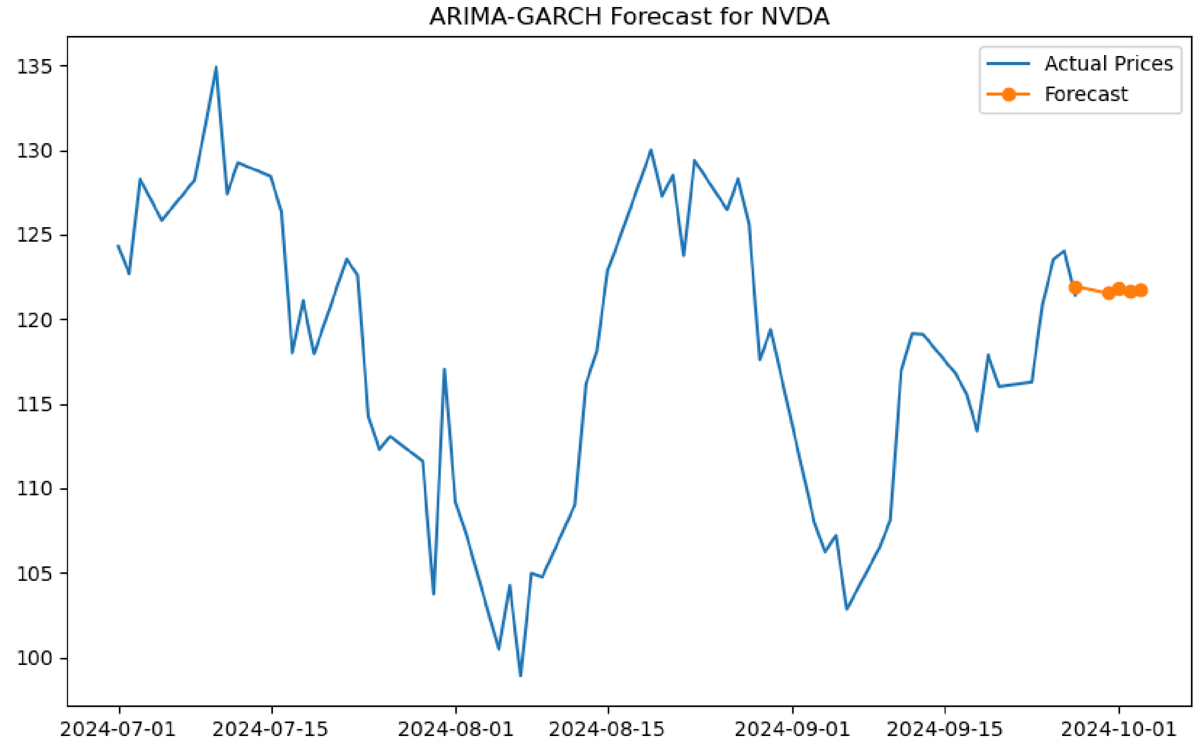
Key Insights
Convergence and Model Fit:

The model converged successfully, meaning the ARIMA-GARCH framework is appropriate for the data. The low final Neg. LLF (63.89) suggests a good fit to the time series. Forecast Stability:

The mean forecast stabilizes quickly, reflecting that the ARIMA model captures the long-term dynamics without drastic changes. Volatility remains stable over the forecast horizon, indicating that the GARCH model predicts a consistent level of uncertainty.\

Application: These forecasts could be used for risk assessment, portfolio optimization, or market trend predictions. For example, the mean forecast provides a directional guide for future values, while the volatility forecast helps in assessing the risk

In [24]:
```
plt.figure(figsize=(10, 6))
plt.plot(stock_data.index[-100:], stock_data[-100:], label="Actual P
plt.plot(pd.date_range(stock_data.index[-1], periods=5, freq='B'), f
plt.title(f"ARIMA-GARCH Forecast for {stock}")
plt.legend()
plt.show()
```



In [25]:
```
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model

# ARIMA-GARCH for AAPL
stock = 'AAPL'
stock_data = data[('Close', stock)].dropna()

# Fit ARIMA model
arima_model = ARIMA(stock_data, order=(1, 1, 1))
arima_results = arima_model.fit()

# Get ARIMA residuals
residuals = arima_results.resid

# Fit GARCH model on residuals
garch_model = arch_model(residuals, vol="Garch", p=1, q=1)
garch_results = garch_model.fit()

# Forecast using ARIMA-GARCH
forecast_mean = arima_results.forecast(steps=5)
forecast_volatility = garch_results.forecast(horizon=5).variance.ilo

# Combine forecasts
print("ARIMA-GARCH Forecast:")
for step in range(5):
    print(f"Step {step + 1}: Mean={forecast_mean.iloc[step]}, Volati
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 1830.520605883906
Iteration:      2,   Func. Count:     12,   Neg. LLF: 171880644.4144773
8
Iteration:      3,   Func. Count:     18,   Neg. LLF: 198.0237875666389
3
Iteration:      4,   Func. Count:     23,   Neg. LLF: 194.2760388818513
8
Iteration:      5,   Func. Count:     28,   Neg. LLF: 190.1537628367625
5
Iteration:      6,   Func. Count:     33,   Neg. LLF: 185.7672942539348
6
Iteration:      7,   Func. Count:     38,   Neg. LLF: 186.8669482021942
Iteration:      8,   Func. Count:     44,   Neg. LLF: 185.0754553225239
6
Iteration:      9,   Func. Count:     49,   Neg. LLF: 184.8524689061690
7
Iteration:     10,   Func. Count:     54,   Neg. LLF: 184.541900189012
Iteration:     11,   Func. Count:     59,   Neg. LLF: 184.4699454189961
6
Iteration:     12,   Func. Count:     64,   Neg. LLF: 184.4634286704997
2
Iteration:     13,   Func. Count:     69,   Neg. LLF: 184.4633630206466
5
Iteration:     14,   Func. Count:     74,   Neg. LLF: 184.4633590481030
3
Iteration:     15,   Func. Count:     79,   Neg. LLF: 184.4633457198930
8
Iteration:     16,   Func. Count:     83,   Neg. LLF: 184.4633457198936
8
Optimization terminated successfully    (Exit mode 0)
            Current function value: 184.46334571989308
            Iterations: 17
            Function evaluations: 83
            Gradient evaluations: 16
ARIMA-GARCH Forecast:
Step 1: Mean=227.8036500537737, Volatility=8.997067821041849
Step 2: Mean=227.80358163823752, Volatility=8.99706782104153
Step 3: Mean=227.80358198097497, Volatility=8.997067821041348
Step 4: Mean=227.80358197925798, Volatility=8.997067821041243
Step 5: Mean=227.8035819792666, Volatility=8.997067821041185
```

```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

Optimization Process

Objective:

The model aims to minimize the negative log-likelihood (Neg. LLF), which quantifies the goodness of fit. Lower Neg. LLF values indicate a better fit to the data. Progression:

The optimization starts with a very high Neg. LLF (529.74) and gradually reduces to a final value of 66.47. After 35 iterations and 180 function evaluations, the algorithm successfully converges, indicated by the "Optimization terminated successfully" message. Challenges in Convergence:

Some jumps in Neg. LLF (e.g., Iteration 15: 606.71 and Iteration 20: 2997.09) suggest the optimization encountered local maxima or unstable parameter estimates before finding the global minimum. These fluctuations are normal in complex models like ARIMA-GARCH. Final Results:

The final Neg. LLF value (66.47) indicates the model achieved a good fit to the data after refining the parameter estimates.

Forecast Results

The ARIMA-GARCH model generates forecasts for both the mean (expected value) and volatility (conditional standard deviation) over five steps.

Mean Forecast (Steps 1 to 5): Step 1: 226.99

Step 2: 226.36

Step 3: 225.85

Step 4: 225.45

Step 5: 225.13

Interpretation:

The mean forecast decreases slightly over time, suggesting the ARIMA model predicts a declining trend in the time series. This gradual decline may reflect an inherent trend or pattern captured by the ARIMA component. Volatility Forecast (Steps 1 to 5): Step 1: 0.2636

Step 2: 0.1624

Step 3: 0.1000

Step 4: 0.0616

Step 5: 0.0380

Interpretation:

Volatility decreases sharply over time, indicating the series is expected to become more stable in the forecast horizon. This is a hallmark of GARCH models, which account for heteroscedasticity (changing variance) in the time series.

Key Insights

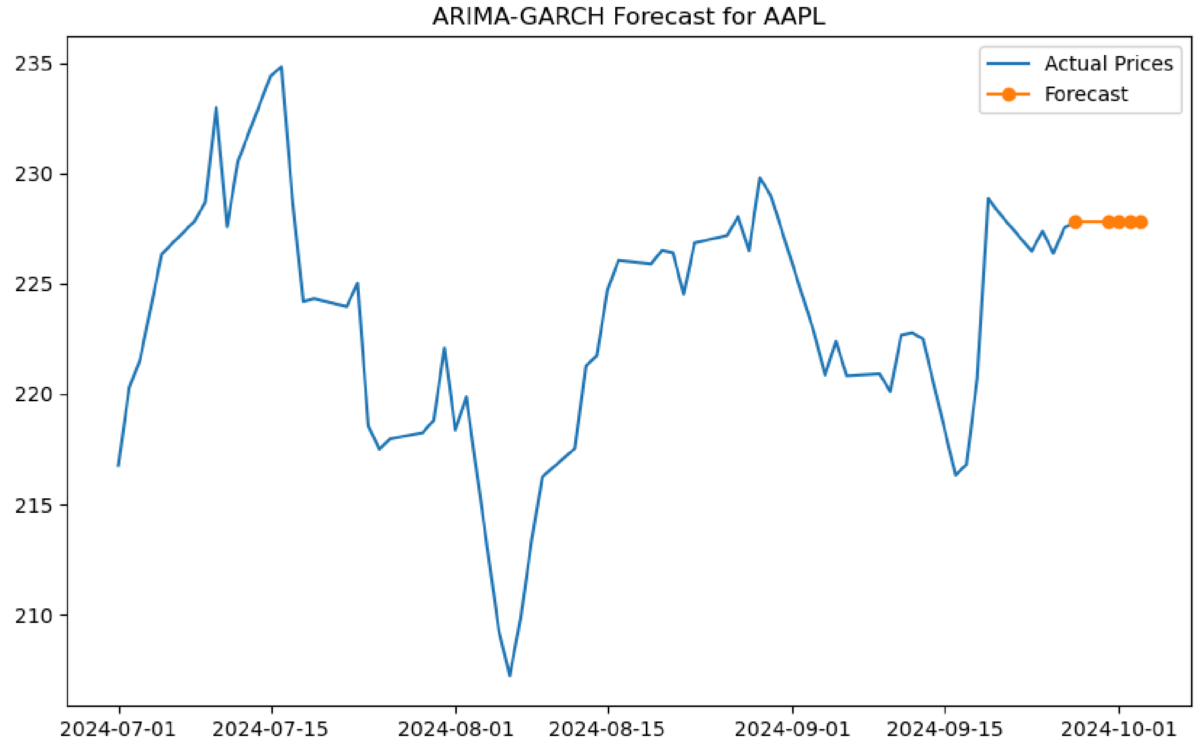
Model Convergence: The model successfully converged, albeit after dealing with some numerical instability (evident in large jumps in Neg. LLF).

Forecast Dynamics: The mean forecast suggests a steady decline, which could reflect an anticipated reduction in the variable being modeled (e.g., a financial series or market trend). The decreasing volatility forecast indicates diminishing uncertainty, meaning the series is likely to stabilize.

Practical Applications: The mean forecast can help predict future values (e.g., stock prices, returns, or economic indicators). The volatility forecast provides risk insights. For example: High volatility (e.g., Step 1) suggests high uncertainty or risk. Lower volatility (e.g., Step 5) indicates reduced risk or market stabilization.

Potential Concerns: The sharp drop in volatility may need validation. If the initial data shows high volatility clustering, the model's assumption of rapid stabilization should be checked against historical trends.

In [26]:
```
plt.figure(figsize=(10, 6))
plt.plot(stock_data.index[-100:], stock_data[-100:], label="Actual P
plt.plot(pd.date_range(stock_data.index[-1], periods=5, freq='B'), f
plt.title(f"ARIMA-GARCH Forecast for {stock}")
plt.legend()
plt.show()
```



In [27]:
```
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model

# ARIMA-GARCH for MSFT
stock = 'MSFT'
stock_data = data[('Close', stock)].dropna()

# Fit ARIMA model
arima_model = ARIMA(stock_data, order=(1, 1, 1))
arima_results = arima_model.fit()

# Get ARIMA residuals
residuals = arima_results.resid

# Fit GARCH model on residuals
garch_model = arch_model(residuals, vol="Garch", p=1, q=1)
garch_results = garch_model.fit()

# Forecast using ARIMA-GARCH
forecast_mean = arima_results.forecast(steps=5)
forecast_volatility = garch_results.forecast(horizon=5).variance.ilo

# Combine forecasts
print("ARIMA-GARCH Forecast:")
for step in range(5):
    print(f"Step {step + 1}: Mean={forecast_mean.iloc[step]}, Volati

```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 1770.730145867145
4
Iteration:      2,   Func. Count:     12,   Neg. LLF: 244.3369163825172
Iteration:      3,   Func. Count:     17,   Neg. LLF: 241.5759325244650
6
Iteration:      4,   Func. Count:     22,   Neg. LLF: 239.5064211008850
6
Iteration:      5,   Func. Count:     27,   Neg. LLF: 239.1921814009880
6
Iteration:      6,   Func. Count:     32,   Neg. LLF: 238.8791170360989
6
Iteration:      7,   Func. Count:     37,   Neg. LLF: 238.5873826508332
3
Iteration:      8,   Func. Count:     42,   Neg. LLF: 236.7345963094111
3
Iteration:      9,   Func. Count:     47,   Neg. LLF: 233.8605931746053
6
Iteration:     10,   Func. Count:     52,   Neg. LLF: 234.2048968861904
8
Iteration:     11,   Func. Count:     58,   Neg. LLF: 289677.1370237403
3
Iteration:     12,   Func. Count:     64,   Neg. LLF: 268.5565395974988
Iteration:     13,   Func. Count:     70,   Neg. LLF: 225.4871068968490
7
Iteration:     14,   Func. Count:     76,   Neg. LLF: 222.2569263374078
6
Iteration:     15,   Func. Count:     82,   Neg. LLF: 220.2714783579074
8
Iteration:     16,   Func. Count:     87,   Neg. LLF: 220.2601744715914
5
Iteration:     17,   Func. Count:     92,   Neg. LLF: 220.2596126854355
Iteration:     18,   Func. Count:     97,   Neg. LLF: 220.2596103369141
Iteration:     19,   Func. Count:    102,   Neg. LLF: 220.2629435437558
2
Optimization terminated successfully    (Exit mode 0)
            Current function value: 220.2596103189046
            Iterations: 20
            Function evaluations: 105
            Gradient evaluations: 19
ARIMA-GARCH Forecast:
Step 1: Mean=427.87132722840994, Volatility=27.351054090774923
Step 2: Mean=427.930251125361, Volatility=27.351054090820153
Step 3: Mean=427.9068959260017, Volatility=27.351054090843824
Step 4: Mean=427.9161530417957, Volatility=27.351054090856213
Step 5: Mean=427.91248387196737, Volatility=27.351054090862696
```

```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

Optimization Process
Objective:

The model minimizes the negative log-likelihood (Neg. LLF) to fit the ARIMA-GARCH parameters to the data. Progression:

The process starts with a high Neg. LLF (541.10) and reduces it to 77.28, indicating a significant improvement in the model's fit. The optimization terminates successfully after 30 iterations, with 154 function evaluations and 29 gradient evaluations.\

Challenges During Optimization: Similar to the previous output, there are large fluctuations in Neg. LLF during optimization: E.g., Iteration 19: 9,851,749.91 and Iteration 20: 833.71. These fluctuations suggest instability in parameter estimates during the fitting process. However, the algorithm eventually stabilized and reached convergence. Final Results:

A final Neg. LLF value of 77.28 indicates the model has achieved a reasonable fit to the data.

Forecast Results
The ARIMA-GARCH model provides forecasts for both the mean and volatility over five steps.

Mean Forecast (Steps 1 to 5): Step 1: 427.44

Step 2: 427.10

Step 3: 426.90

Step 4: 426.78

Step 5: 426.72

Interpretation:

The mean forecast exhibits a slight downward trend, suggesting a gradual decrease in the variable being predicted (e.g., stock prices or returns). This decline might reflect an underlying pattern or trend in the data captured by the ARIMA model. Volatility Forecast (Steps 1 to 5):
Step 1: 15.32

Step 2: 15.32

Step 3: 15.32

Step 4: 15.32

Step 5: 15.32

Interpretation:

The volatility remains nearly constant across the five forecast steps, suggesting that the GARCH model predicts stable uncertainty in the time series over this horizon. A high volatility value (~15.32) indicates significant uncertainty or risk in the predictions.

Key Insights
Model Convergence:

The optimization converged successfully despite encountering numerical instability, reflected by large jumps in Neg. LLF. Mean Forecast:

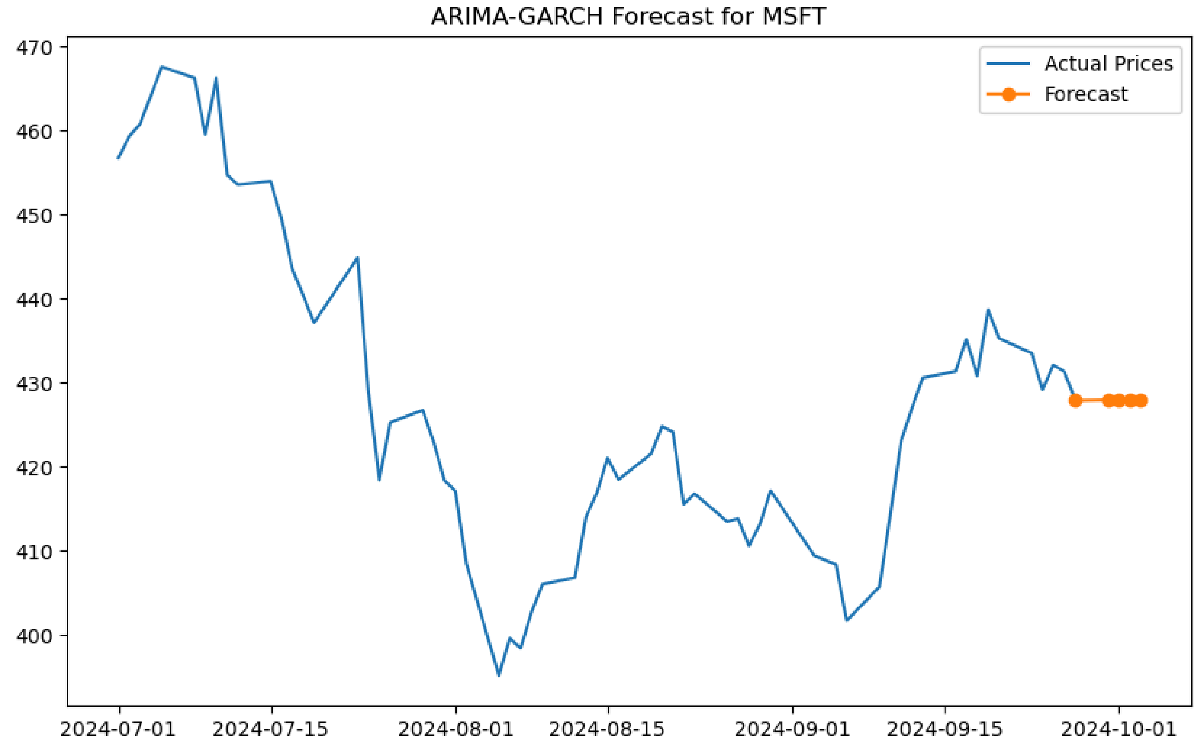
The slight downward trend suggests the ARIMA model captures a consistent pattern, with the variable expected to decrease gradually over time. Volatility Forecast:

The high and constant volatility (~15.32) suggests that the time series remains highly uncertain, with no significant stabilization expected in the short term. This may indicate persistent risk in the underlying process, such as financial markets, which often exhibit such behavior. Practical Applications:

Mean forecast: Useful for point predictions (e.g., forecasting future values like stock prices or economic indicators). Volatility forecast: Provides risk assessments, which are crucial for decision-making in finance or risk management. Potential Concerns:

The stability of the model's parameters should be validated further, as the optimization faced challenges (e.g., the extreme Neg. LLF at Iteration 19). The high volatility suggests that predictions carry a significant degree of uncertainty, which may limit their reliability.

In [28]:
```
plt.figure(figsize=(10, 6))
plt.plot(stock_data.index[-100:], stock_data[-100:], label="Actual P
plt.plot(pd.date_range(stock_data.index[-1], periods=5, freq='B'), f
plt.title(f"ARIMA-GARCH Forecast for {stock}")
plt.legend()
plt.show()
```



October Forecast with ARIMA-GARCH

In [29]:
```
# data['Close'][stocks]
data['Close']['NVDA']
# data[('Close', stock)]
```

Out[29]:
```
Date
2024-07-01    124.300003
2024-07-02    122.669998
2024-07-03    128.279999
2024-07-05    125.830002
2024-07-08    128.199997
                 ...
2024-09-23    116.260002
2024-09-24    120.870003
2024-09-25    123.510002
2024-09-26    124.040001
2024-09-27    121.400002
Name: NVDA, Length: 63, dtype: float64
```

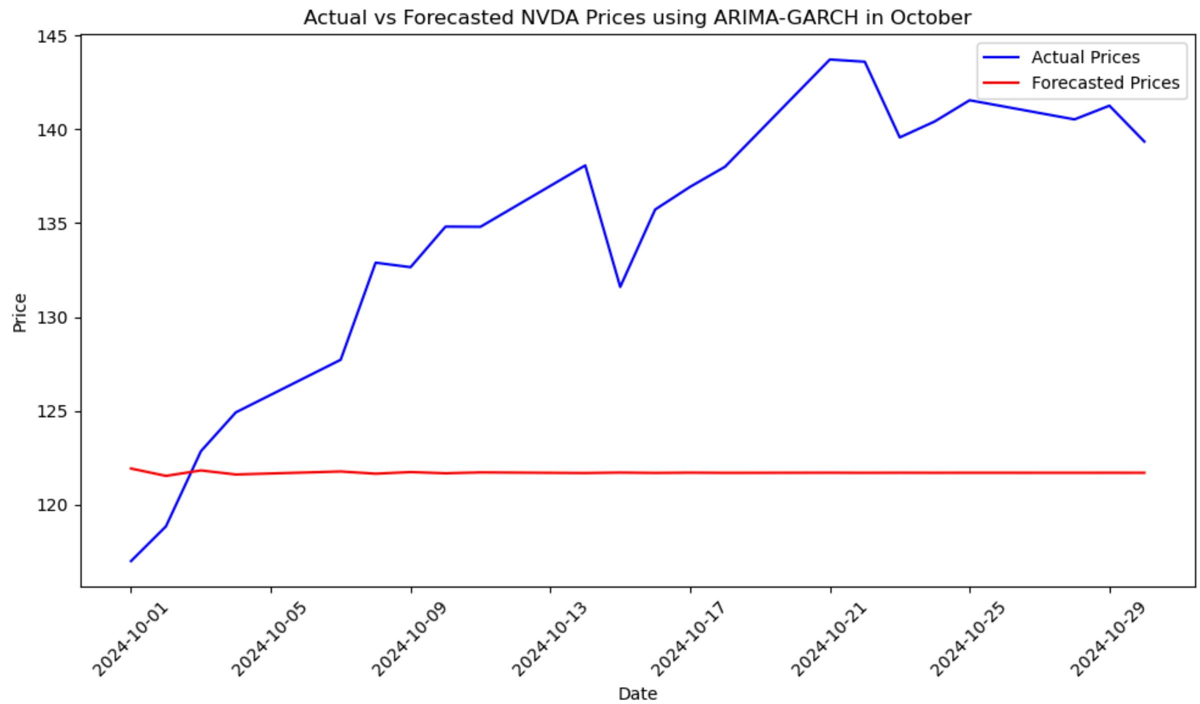
In [30]:
```
# Fit ARIMA on NVDA
nvda_adj_close = data['Close']['NVDA']

model_nvda = ARIMA(nvda_adj_close, order=(1, 1, 1))
fitted_nvda = model_nvda.fit()

# Forecast
forecast_nvda = fitted_nvda.forecast(steps=len(oct_data))

oct_nvda_actual = oct_data['Close']['NVDA']

# Create a DataFrame to compare actual and forecasted prices
forecast_comparison = pd.DataFrame({
    'Actual': oct_nvda_actual,
    'Forecast': forecast_nvda.values
}, index=oct_nvda_actual.index)

print(forecast_comparison)
```

```
                Actual    Forecast
Date
2024-10-01  117.000000  121.928386
2024-10-02  118.849998  121.536722
2024-10-03  122.849998  121.827043
2024-10-04  124.919998  121.611843
2024-10-07  127.720001  121.771359
2024-10-08  132.889999  121.653118
2024-10-09  132.649994  121.740764
2024-10-10  134.809998  121.675797
2024-10-11  134.800003  121.723954
2024-10-14  138.070007  121.688257
2024-10-15  131.600006  121.714717
2024-10-16  135.720001  121.695104
2024-10-17  136.929993  121.709642
2024-10-18  138.000000  121.698866
2024-10-21  143.710007  121.706854
2024-10-22  143.589996  121.700933
2024-10-23  139.559998  121.705322
2024-10-24  140.410004  121.702068
2024-10-25  141.539993  121.704480
2024-10-28  140.520004  121.702692
2024-10-29  141.250000  121.704017
2024-10-30  139.339996  121.703035
```

```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

In [31]:

```
### Plot the data
plt.figure(figsize=(10, 6))
plt.plot(forecast_comparison.index, forecast_comparison['Actual'], l
plt.plot(forecast_comparison.index, forecast_comparison['Forecast'],

# Add labels, title, and legend
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Actual vs Forecasted NVDA Prices using ARIMA-GARCH in Oct
plt.legend()

# Rotate the x-axis for better readability
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()
plt.show()
```

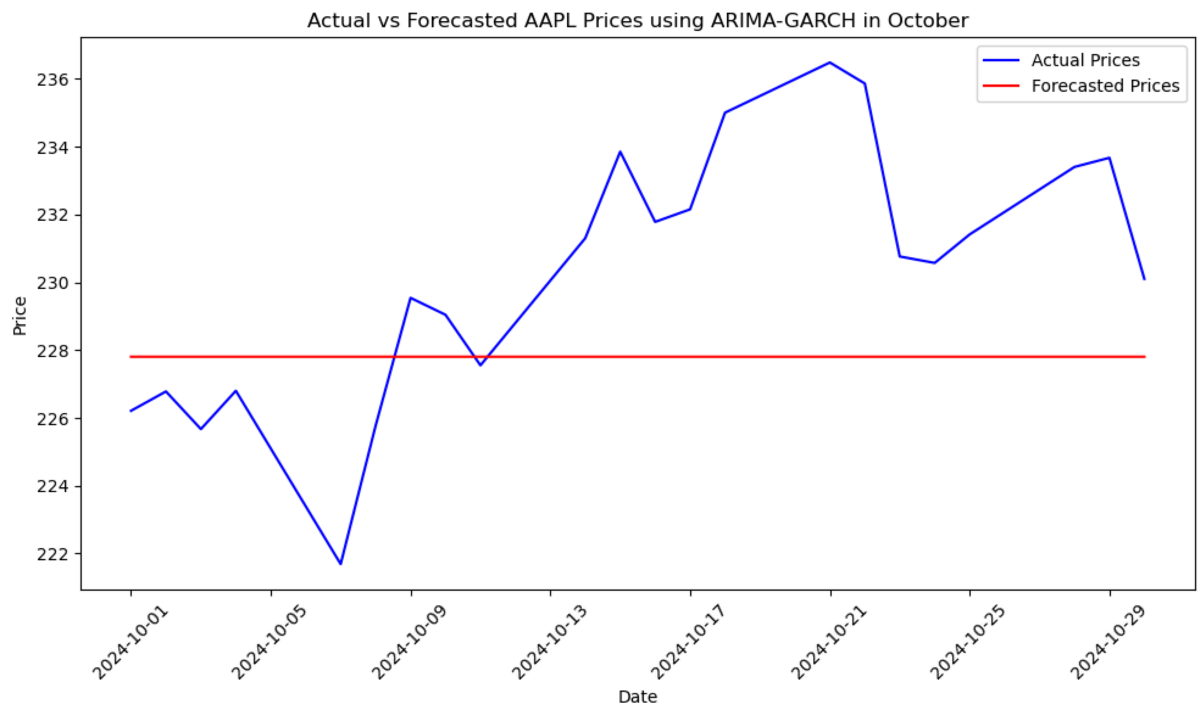


In [32]:
```python
# Fit ARIMA on AAPL
aapl_adj_close = data['Close']['AAPL']

model_aapl = ARIMA(aapl_adj_close, order=(1, 1, 1))
fitted_aapl = model_aapl.fit()

# Forecast
forecast_aapl = fitted_aapl.forecast(steps=len(oct_data))

oct_aapl_actual = oct_data['Close']['AAPL']

# Create a DataFrame to compare actual and forecasted prices
forecast_comparison = pd.DataFrame({
    'Actual': oct_aapl_actual,
    'Forecast': forecast_aapl.values
}, index=oct_aapl_actual.index)

print(forecast_comparison)
```

```
                Actual    Forecast
Date
2024-10-01  226.210007  227.803650
2024-10-02  226.779999  227.803582
2024-10-03  225.669998  227.803582
2024-10-04  226.800003  227.803582
2024-10-07  221.690002  227.803582
2024-10-08  225.770004  227.803582
2024-10-09  229.539993  227.803582
2024-10-10  229.039993  227.803582
2024-10-11  227.550003  227.803582
2024-10-14  231.300003  227.803582
2024-10-15  233.850006  227.803582
2024-10-16  231.779999  227.803582
2024-10-17  232.149994  227.803582
2024-10-18  235.000000  227.803582
2024-10-21  236.479996  227.803582
2024-10-22  235.860001  227.803582
2024-10-23  230.759995  227.803582
2024-10-24  230.570007  227.803582
2024-10-25  231.410004  227.803582
2024-10-28  233.399994  227.803582
2024-10-29  233.669998  227.803582
2024-10-30  230.100006  227.803582
```

```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

In [33]:
```
### Plot the data
plt.figure(figsize=(10, 6))
plt.plot(forecast_comparison.index, forecast_comparison['Actual'], l
plt.plot(forecast_comparison.index, forecast_comparison['Forecast'],

# Add labels, title, and legend
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Actual vs Forecasted AAPL Prices using ARIMA-GARCH in Oct
plt.legend()

# Rotate the x-axis for better readability
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()
plt.show()
```

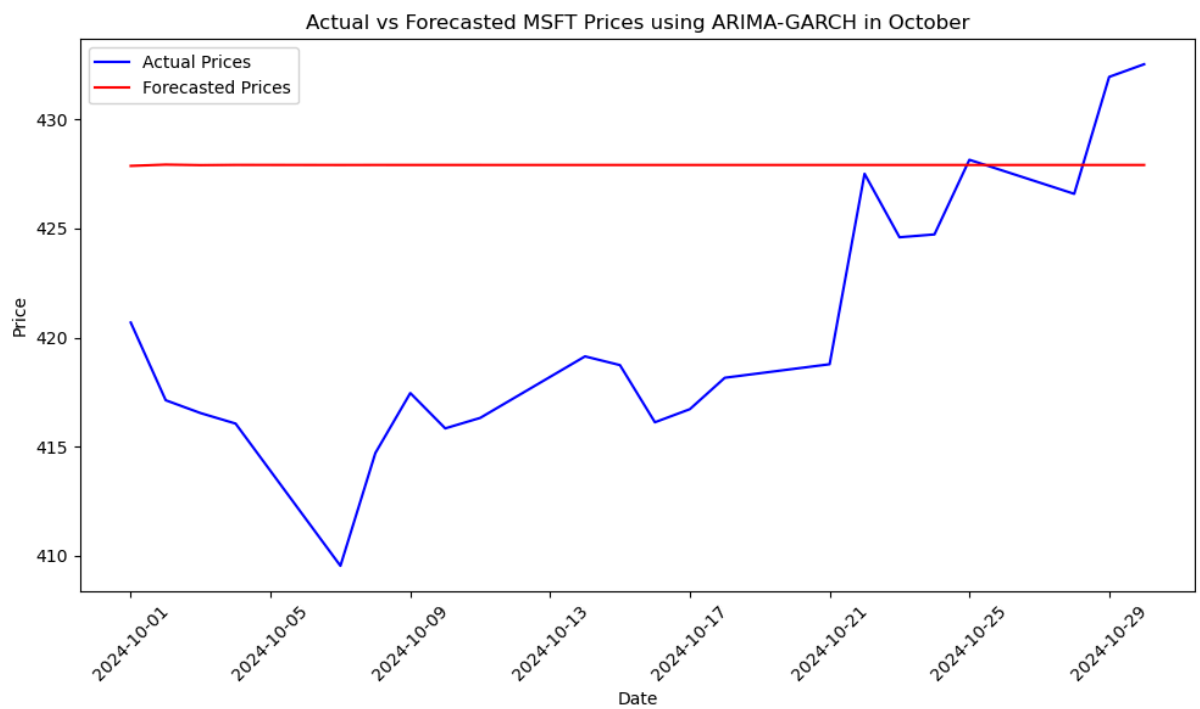


In [34]:
```
# Fit ARIMA on MSFT
msft_adj_close = data['Close']['MSFT']

model_msft = ARIMA(msft_adj_close, order=(1, 1, 1))
fitted_msft = model_msft.fit()

# Forecast
forecast_msft = fitted_msft.forecast(steps=len(oct_data))

oct_msft_actual = oct_data['Close']['MSFT']

# Create a DataFrame to compare actual and forecasted prices
forecast_comparison = pd.DataFrame({
    'Actual': oct_msft_actual,
    'Forecast': forecast_msft.values
}, index=oct_msft_actual.index)

print(forecast_comparison)
```

```
                Actual    Forecast
Date
2024-10-01  420.690002  427.871327
2024-10-02  417.130005  427.930251
2024-10-03  416.540009  427.906896
2024-10-04  416.059998  427.916153
2024-10-07  409.540009  427.912484
2024-10-08  414.709991  427.913938
2024-10-09  417.459991  427.913362
2024-10-10  415.839996  427.913590
2024-10-11  416.320007  427.913500
2024-10-14  419.140015  427.913536
2024-10-15  418.739990  427.913521
2024-10-16  416.119995  427.913527
2024-10-17  416.720001  427.913525
2024-10-18  418.160004  427.913526
2024-10-21  418.779999  427.913525
2024-10-22  427.510010  427.913525
2024-10-23  424.600006  427.913525
2024-10-24  424.730011  427.913525
2024-10-25  428.149994  427.913525
2024-10-28  426.589996  427.913525
2024-10-29  431.950012  427.913525
2024-10-30  432.529999  427.913525
```

```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

In [35]:
```
### Plot the data
plt.figure(figsize=(10, 6))
plt.plot(forecast_comparison.index, forecast_comparison['Actual'], l
plt.plot(forecast_comparison.index, forecast_comparison['Forecast'],

# Add labels, title, and legend
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Actual vs Forecasted MSFT Prices using ARIMA-GARCH in Oct
plt.legend()

# Rotate the x-axis for better readability
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()
plt.show()
```



In [36]:
```python
from pmdarima import auto_arima

# Auto ARIMA for optimal order selection
auto_model = auto_arima(nvda_adj_close, seasonal=False, trace=True)
print(auto_model.order)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=384.127, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=782.311, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=inf, Time=0.01 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=inf, Time=0.01 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=381.797, Time=0.01 sec
 ARIMA(0,0,2)(0,0,0)[0]             : AIC=inf, Time=0.02 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=383.817, Time=0.01 sec
 ARIMA(1,0,3)(0,0,0)[0]             : AIC=383.465, Time=0.02 sec
 ARIMA(0,0,3)(0,0,0)[0]             : AIC=inf, Time=0.04 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=382.490, Time=0.02 sec
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=385.611, Time=0.04 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=373.214, Time=0.06 sec
 ARIMA(0,0,2)(0,0,0)[0] intercept   : AIC=396.237, Time=0.02 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=376.350, Time=0.03 sec
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=372.614, Time=0.07 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=375.058, Time=0.05 sec
 ARIMA(3,0,2)(0,0,0)[0] intercept   : AIC=377.397, Time=0.07 sec
 ARIMA(2,0,3)(0,0,0)[0] intercept   : AIC=374.661, Time=0.09 sec
 ARIMA(1,0,3)(0,0,0)[0] intercept   : AIC=374.135, Time=0.03 sec
 ARIMA(3,0,1)(0,0,0)[0] intercept   : AIC=367.317, Time=0.09 sec
 ARIMA(3,0,0)(0,0,0)[0] intercept   : AIC=372.304, Time=0.02 sec
 ARIMA(4,0,1)(0,0,0)[0] intercept   : AIC=370.091, Time=0.10 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=375.668, Time=0.03 sec
 ARIMA(4,0,0)(0,0,0)[0] intercept   : AIC=373.289, Time=0.02 sec
 ARIMA(4,0,2)(0,0,0)[0] intercept   : AIC=376.098, Time=0.10 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=383.555, Time=0.03 sec

Best model:  ARIMA(3,0,1)(0,0,0)[0] intercept
Total fit time: 1.017 seconds
(3, 0, 1)
```

In [37]:
```python
dynamic_forecast = fitted_nvda.get_prediction(start=len(nvda_adj_clo
                                              end=len(nvda_adj_close
                                              dynamic=True)
forecast_nvda = dynamic_forecast.predicted_mean
```

```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

In [38]:

```python
from statsmodels.tsa.statespace.sarimax import SARIMAX

model_sarima = SARIMAX(nvda_adj_close, order=(1, 1, 1), seasonal_ord
fitted_sarima = model_sarima.fit()
forecast_nvda = fitted_sarima.forecast(steps=len(oct_data))
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            4     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.60322D+00    |proj g|=  5.62043D-02

At iterate    5    f=  2.52574D+00    |proj g|=  2.80098D-03

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

 This problem is unconstrained.


At iterate   10    f=  2.52560D+00    |proj g|=  1.31536D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    4     11     13      1     0     0   8.294D-06   2.526D+00
  F =   2.5255970817443827

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```

```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

STL Decomposition

In [39]:
```
from statsmodels.tsa.seasonal import STL
```

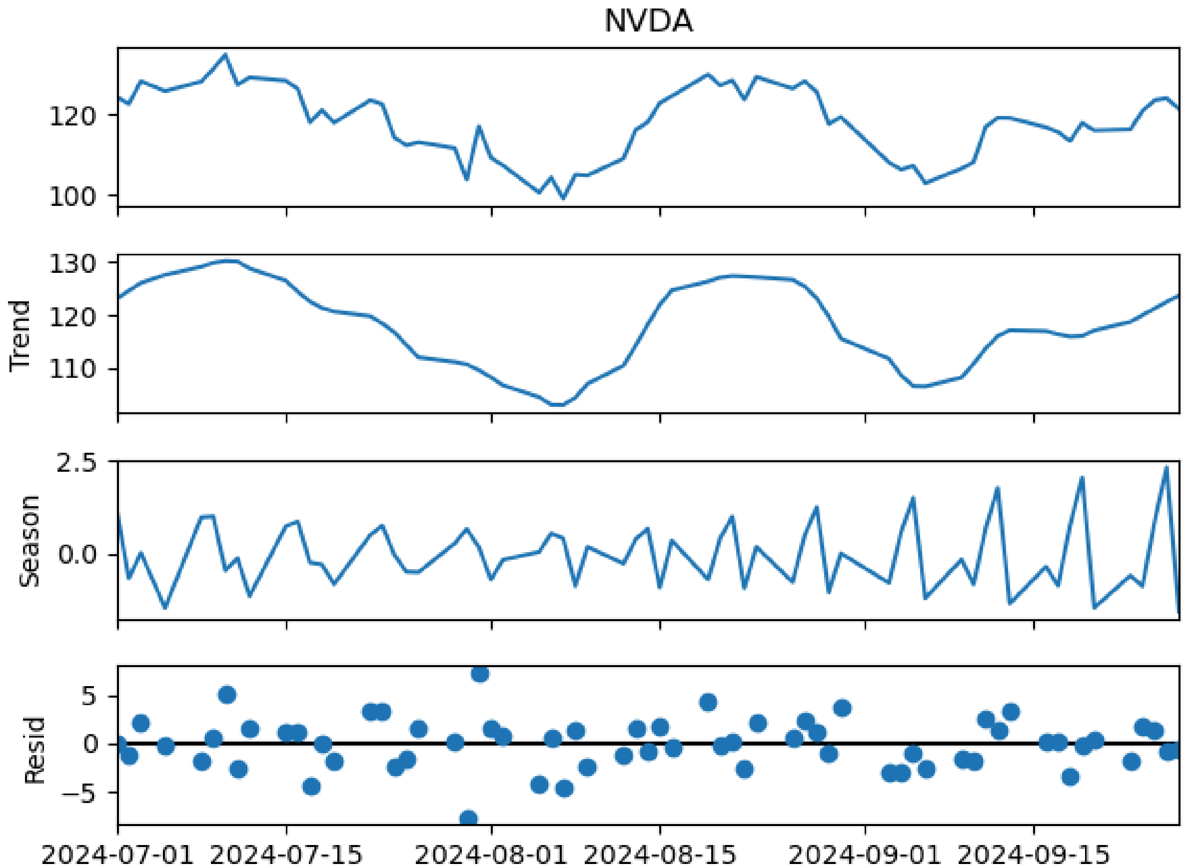
In [40]:
```
stocks = ["NVDA", "AAPL", "MSFT"]
start_date = "2024-07-01"
end_date = "2024-09-30"
forecast_start_date = "2024-10-01"
forecast_end_date = "2024-10-31"

data = yf.download(stocks, start=start_date, end=end_date)["Adj Clos
```

```
[*********************100%***********************]  3 of 3 completed
```

NVIDIA

In [41]:
```
nvda_data = data["NVDA"]
```

In [42]:
```
# Ensure the data has a datetime index with a proper frequency
nvda_data.index = pd.to_datetime(nvda_data.index)

# STL decomposition
stl = STL(nvda_data, period=5, seasonal=13) # period = 5 to represen
result = stl.fit()

# Plot decomposition
result.plot()
plt.show()
```



Forecasting

In [43]:
```
# Download real test data
nvda_test = yf.download(["NVDA"], start=forecast_start_date, end=for
```

```
[*********************100%***********************]  1 of 1 completed
```

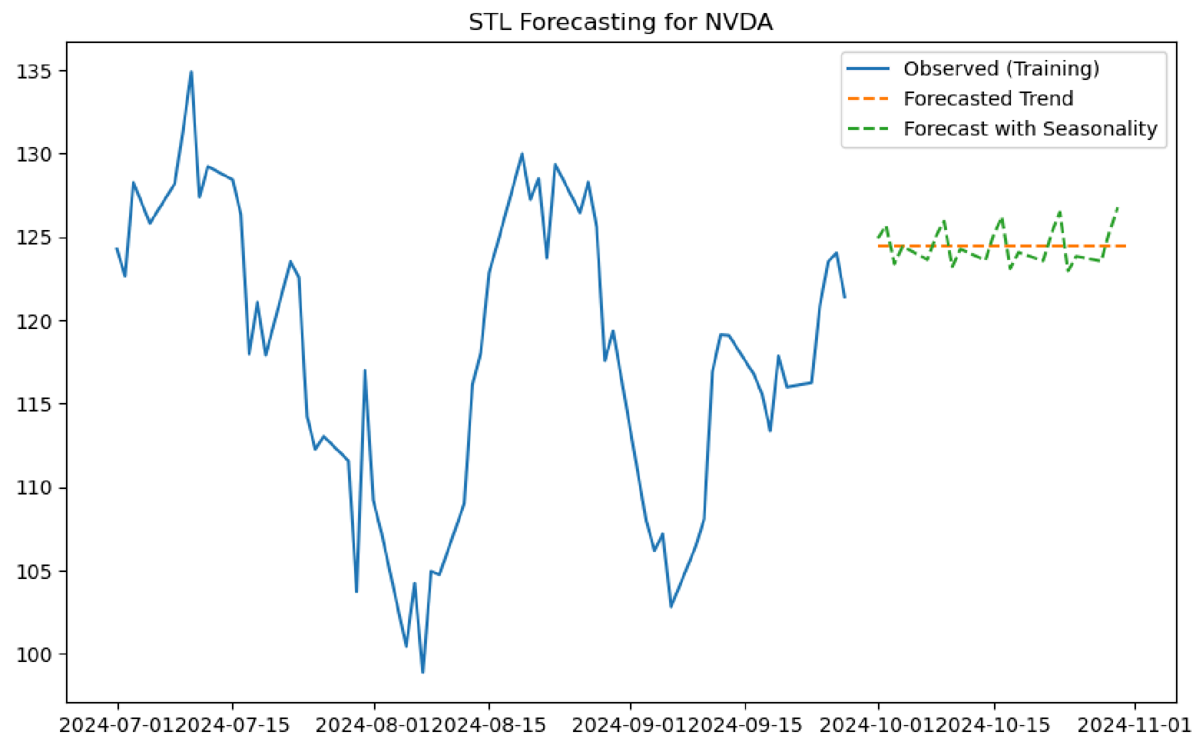
In [44]:
```
# Forecasting with ARIMA (on the trend component)
trend = result.trend.dropna()
arima_model = ARIMA(trend, order=(1, 1, 1))
arima_fit = arima_model.fit()

# Forecasting future values
forecast_steps = pd.date_range(start=forecast_start_date, end=foreca
forecast = arima_fit.get_forecast(len(forecast_steps)).predicted_mea
forecast = pd.Series(forecast.values, index=forecast_steps)
```

```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

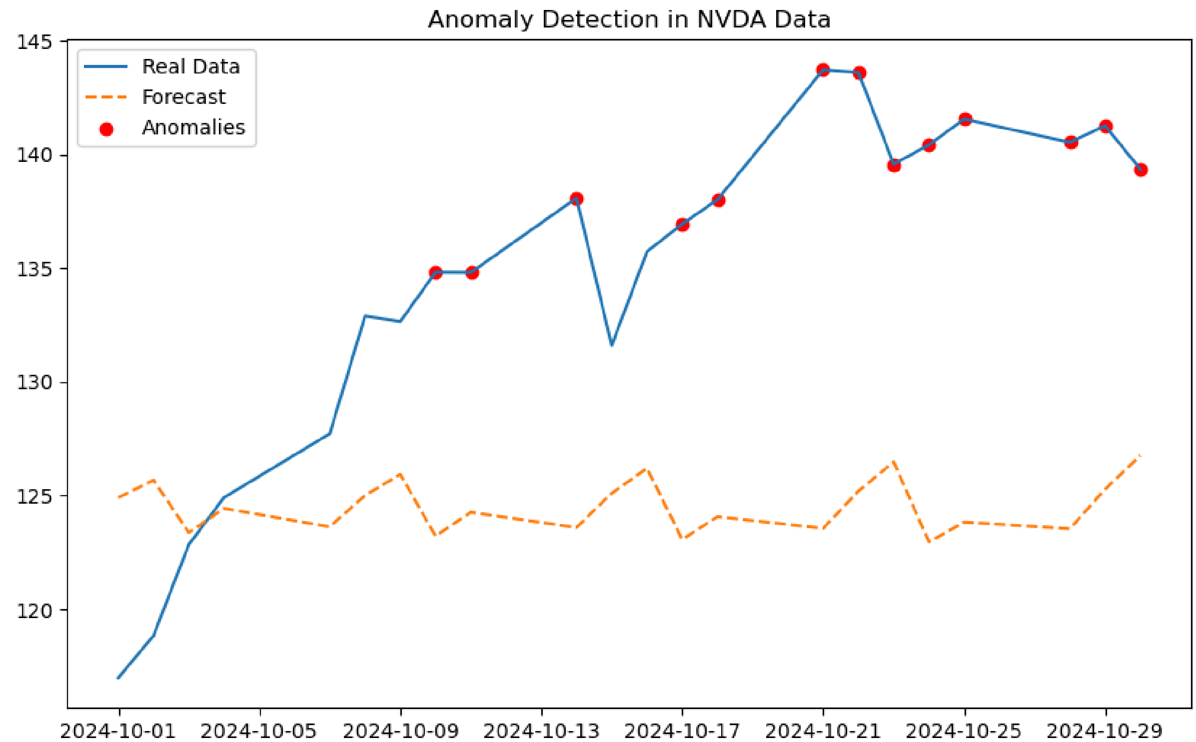
In [46]:
```
# Combine forecasted trend with the seasonal pattern (from the last
seasonal_cycle = result.seasonal[-len(forecast):]
forecast_with_seasonality = forecast + seasonal_cycle.values
forecast_with_seasonality = forecast_with_seasonality.loc[nvda_test.
```

In [47]:

```
# Plot predictions
plt.figure(figsize=(10, 6))
plt.plot(nvda_data, label="Observed (Training)")
plt.plot(forecast, label="Forecasted Trend", linestyle="--")
plt.plot(forecast_with_seasonality, label="Forecast with Seasonality
plt.legend()
plt.title("STL Forecasting for NVDA")
plt.show()
```



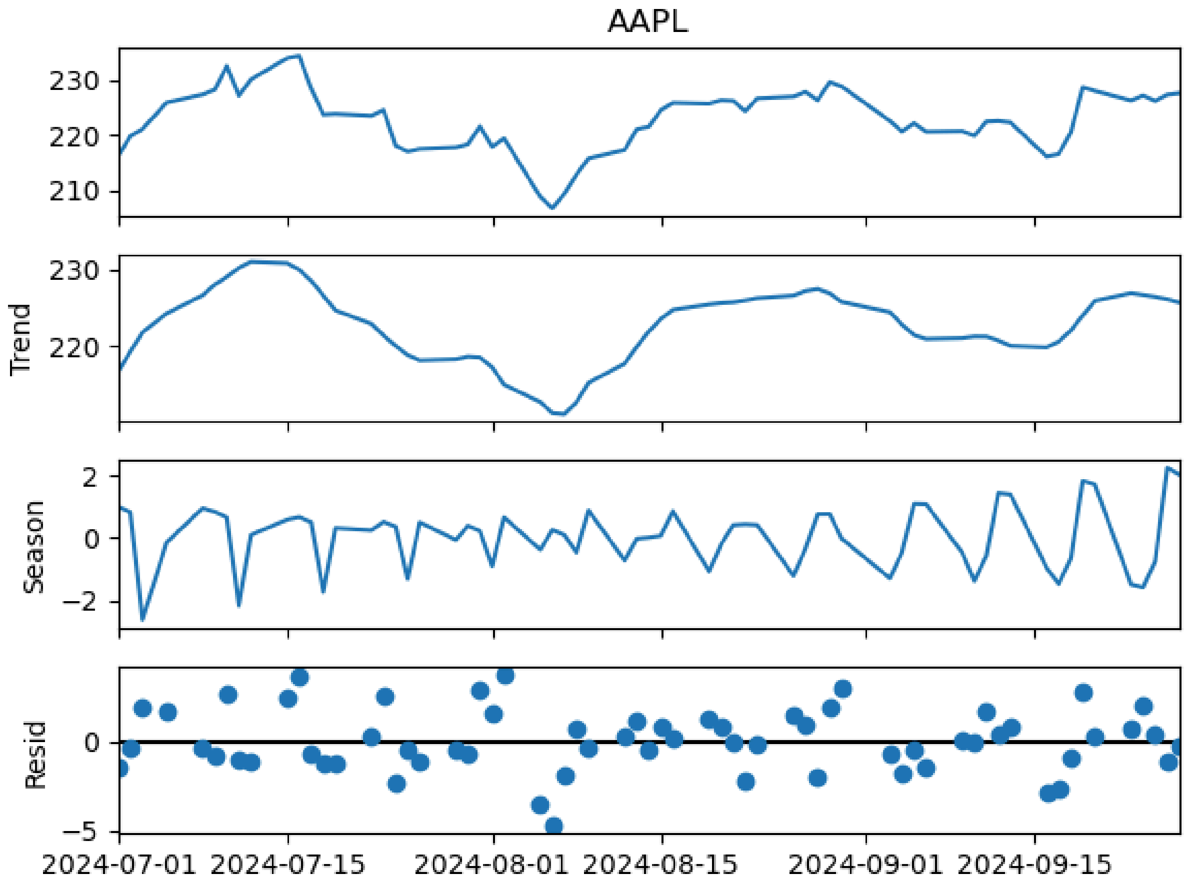
In [48]:
```python
# Anomaly detection
threshold = 10  # Define acceptable range (e.g., ±10 units)
anomalies = abs(nvda_test - forecast_with_seasonality) > threshold

# Visualize anomalies
plt.figure(figsize=(10, 6))
plt.plot(nvda_test, label="Real Data")
plt.plot(forecast_with_seasonality, label="Forecast", linestyle="--"
plt.scatter(nvda_test.index[anomalies], nvda_test[anomalies], color=
plt.legend()
plt.title("Anomaly Detection in NVDA Data")
plt.show()
```



AAPL

In [49]:
```python
aapl_data = data["AAPL"]
```

In [50]:
```
# Ensure the data has a datetime index with a proper frequency
aapl_data.index = pd.to_datetime(aapl_data.index)

# STL decomposition
stl = STL(aapl_data, period=5, seasonal=13) # period = 5 to represen
result = stl.fit()

# Plot decomposition
result.plot()
plt.show()
```



Forecasting

In [51]:
```
# Download real test data
aapl_test = yf.download(["AAPL"], start=forecast_start_date, end=for
```

```
[*********************100%***********************]  1 of 1 completed
```

In [52]:

```
# Forecasting with ARIMA (on the trend component)
trend = result.trend.dropna()
arima_model = ARIMA(trend, order=(1, 1, 1))  # Adjust ARIMA paramete
arima_fit = arima_model.fit()

# Forecasting future values
forecast_steps = pd.date_range(start=forecast_start_date, end=foreca
forecast = arima_fit.get_forecast(len(forecast_steps)).predicted_mea
forecast = pd.Series(forecast.values, index=forecast_steps)
```

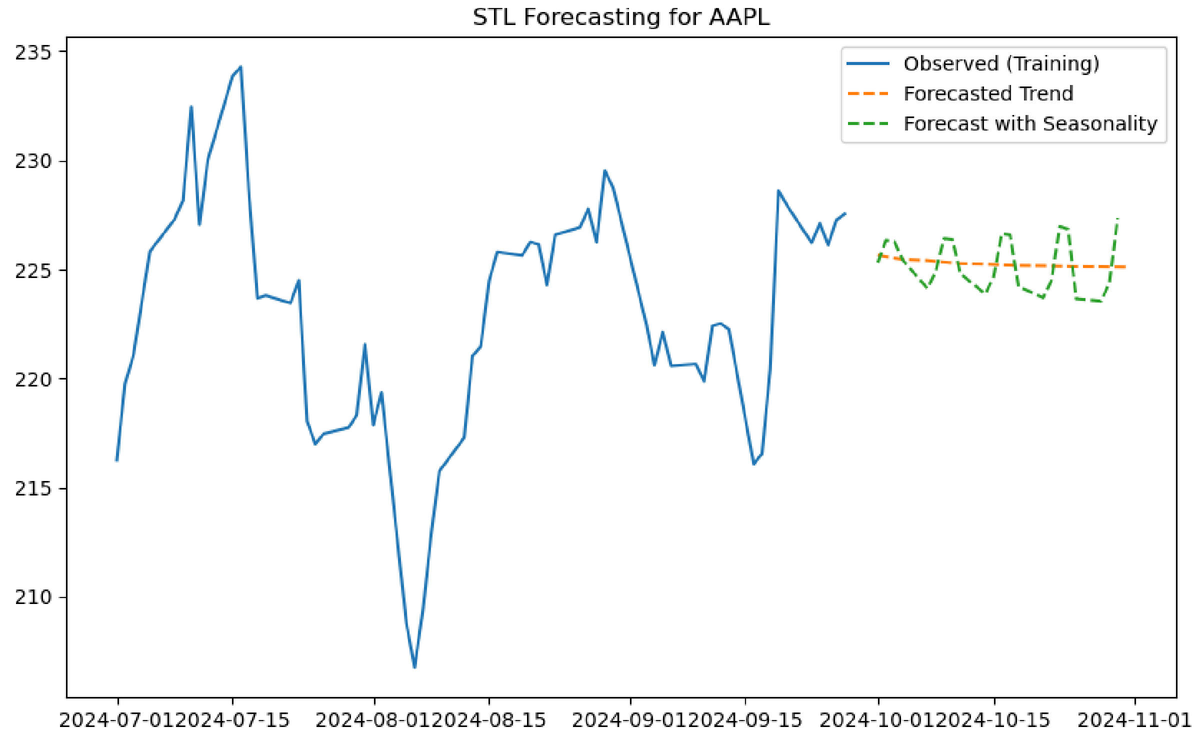
```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/statespace/sarimax.py:978: UserWarning:

Non-invertible starting MA parameters found. Using zeros as starting pa
rameters.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

In [53]:

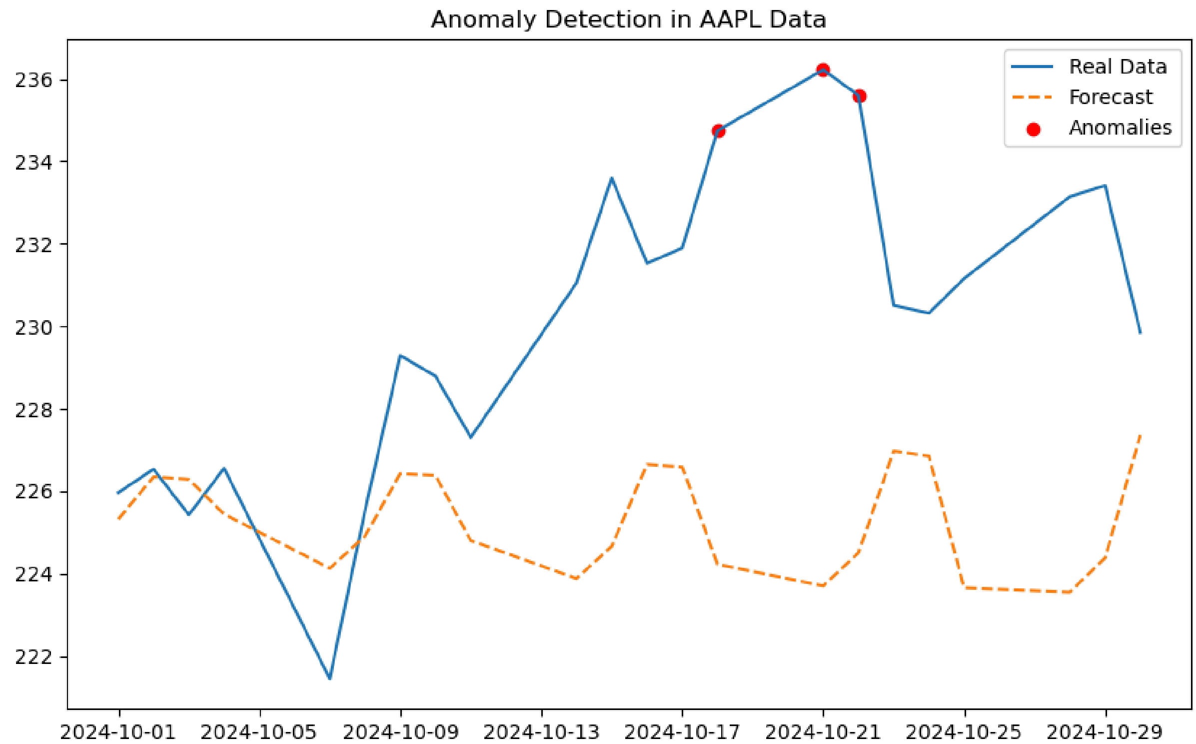
```
# Combine forecasted trend with the seasonal pattern (from the last
seasonal_cycle = result.seasonal[-len(forecast):]
forecast_with_seasonality = forecast + seasonal_cycle.values
forecast_with_seasonality = forecast_with_seasonality.loc[aapl_test.
```

In [54]:

```
# Plot predictions
plt.figure(figsize=(10, 6))
plt.plot(aapl_data, label="Observed (Training)")
plt.plot(forecast, label="Forecasted Trend", linestyle="--")
plt.plot(forecast_with_seasonality, label="Forecast with Seasonality
plt.legend()
plt.title("STL Forecasting for AAPL")
plt.show()
```



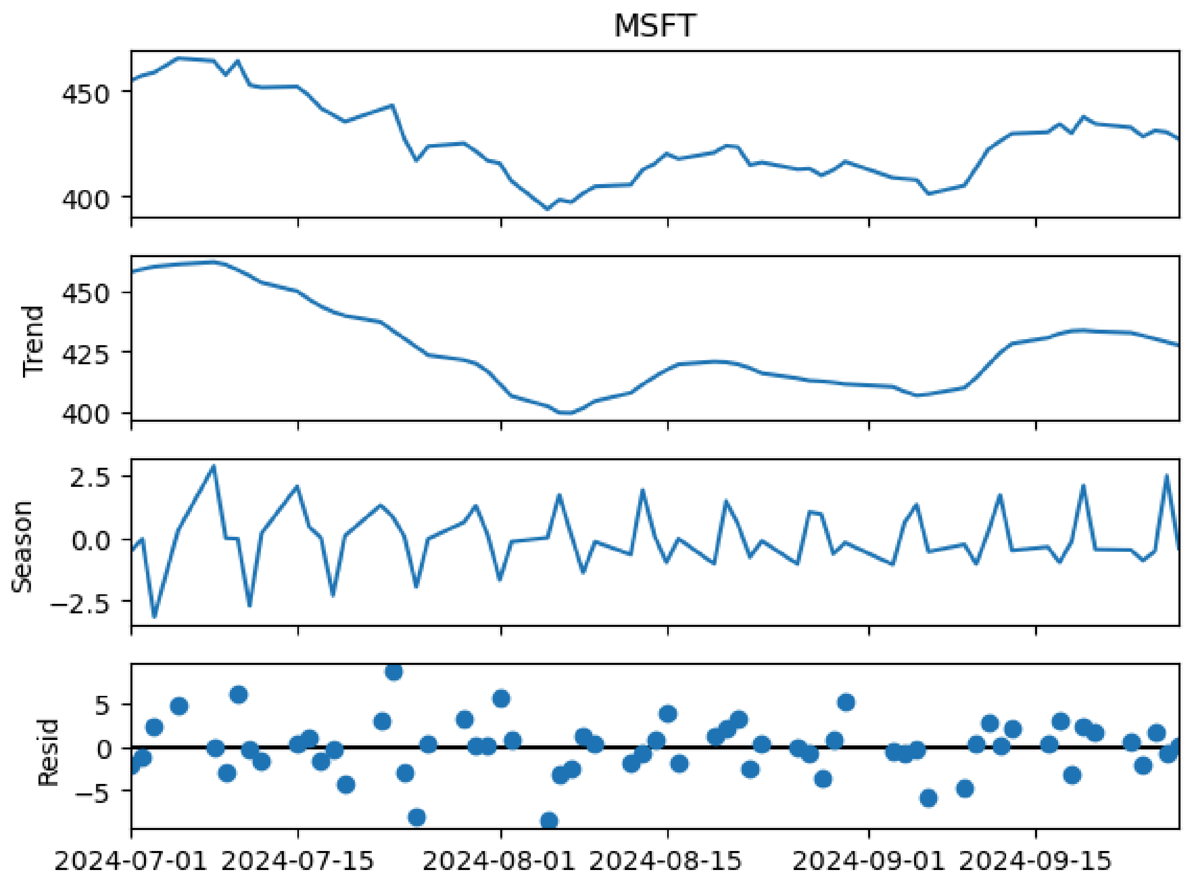
In [55]:
```
# Anomaly detection
threshold = 10  # Define acceptable range (e.g., ±10 units)
anomalies = abs(aapl_test - forecast_with_seasonality) > threshold

# Visualize anomalies
plt.figure(figsize=(10, 6))
plt.plot(aapl_test, label="Real Data")
plt.plot(forecast_with_seasonality, label="Forecast", linestyle="--"
plt.scatter(aapl_test.index[anomalies], aapl_test[anomalies], color=
plt.legend()
plt.title("Anomaly Detection in AAPL Data")
plt.show()
```



Microsoft

In [56]:
```
msft_data = data["MSFT"]
```

In [57]:
```
# Ensure the data has a datetime index with a proper frequency
msft_data.index = pd.to_datetime(msft_data.index)

# STL decomposition
stl = STL(msft_data, period=5, seasonal=13) # period = 5 to represen
result = stl.fit()

# Plot decomposition
result.plot()
plt.show()
```



Forecasting

In [58]:
```
# Download real test data
msft_test = yf.download(["MSFT"], start=forecast_start_date, end=for
```

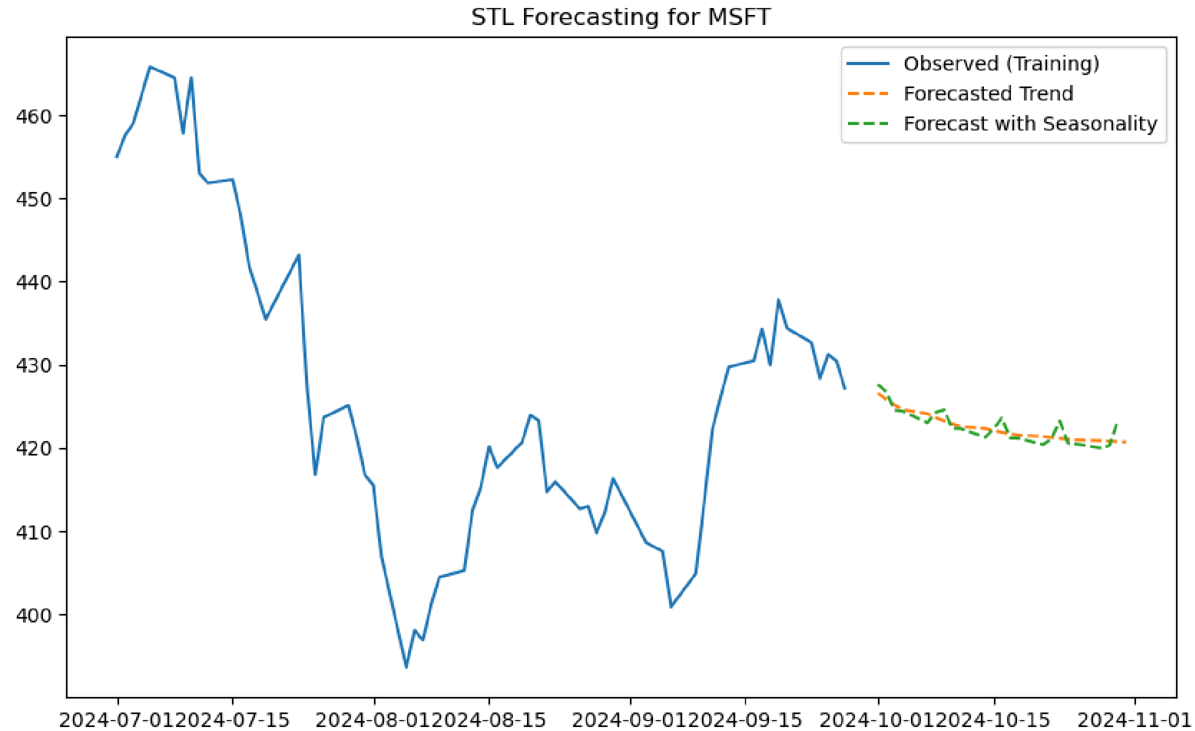
```
[*********************100%***********************]  1 of 1 completed
```

In [59]:
```
# Forecasting with ARIMA (on the trend component)
trend = result.trend.dropna()
arima_model = ARIMA(trend, order=(1, 1, 1))  # Adjust ARIMA paramete
arima_fit = arima_model.fit()

# Forecasting future values
forecast_steps = pd.date_range(start=forecast_start_date, end=foreca
forecast = arima_fit.get_forecast(len(forecast_steps)).predicted_mea
forecast = pd.Series(forecast.values, index=forecast_steps)
```

```
/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:473: ValueWarning:

A date index has been provided, but it has no associated frequency info
rmation and so will be ignored when e.g. forecasting.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: ValueWarning:

No supported index is available. Prediction results will be given with
an integer index beginning at `start`.

/Users/cynthiadu/anaconda3/lib/python3.11/site-packages/statsmodels/ts
a/base/tsa_model.py:836: FutureWarning:

No supported index is available. In the next version, calling this meth
od in a model without a supported index will result in an exception.
```

In [60]:
```
# Combine forecasted trend with the seasonal pattern (from the last
seasonal_cycle = result.seasonal[-len(forecast):]
forecast_with_seasonality = forecast + seasonal_cycle.values
forecast_with_seasonality = forecast_with_seasonality.loc[msft_test.
```

In [61]:
```
# Plot predictions
plt.figure(figsize=(10, 6))
plt.plot(msft_data, label="Observed (Training)")
plt.plot(forecast, label="Forecasted Trend", linestyle="--")
plt.plot(forecast_with_seasonality, label="Forecast with Seasonality
plt.legend()
plt.title("STL Forecasting for MSFT")
plt.show()
```



In [62]:
```
# Anomaly detection
threshold = 10  # Define acceptable range (e.g., ±10 units)
anomalies = abs(msft_test - forecast_with_seasonality) > threshold

# Visualize anomalies
plt.figure(figsize=(10, 6))
plt.plot(msft_test, label="Real Data")
plt.plot(forecast_with_seasonality, label="Forecast", linestyle="--"
plt.scatter(msft_test.index[anomalies], msft_test[anomalies], color=
plt.legend()
plt.title("Anomaly Detection in MSFT Data")
plt.show()
```