Algorithmic Trading - Reversion and Trend-Following Strategies

This report explores the development, backtesting, and implementation of two core algorithmic trading strategies: **trend-following** and **mean-reversion**, with a focus on quantitative methods and practical application in the high-frequency trading space. The project is divided into three key parts, each addressing a crucial aspect of the trading system's design and evaluation. The analysis utilizes high-resolution intraday market data, capturing price movements to ensure model robustness and accuracy.

Project Breakdown

Part I: Historical Backtesting

This section establishes the foundation for the trading strategies by leveraging historical data. It introduces:

1. Trend-Following Strategies

- Utilization of indicators like **Exponential Moving Averages (EMA)**, **Moving Average Convergence Divergence(MACD)** and **Average Directional Index (ADX)** to identify and capitalize on sustained market trends.
- Design of trading logic based on crossover signals and trend strength filtering.
- Experiments (details omitted) with time intervals, resampling techniques, and ratio-based trend indicators to refine signal accuracy.

2. Mean-Reversion Strategies

- Implementation of **Z-scores** and the **Ornstein-Uhlenbeck (OU) process** to detect price deviations from the mean.
- Trading logic driven by threshold-based entry and exit signals.
- Advanced techniques, such as Kalman filters, to enhance strategy robustness under various market conditions.

Part II: Broker API Integration

This section extends the strategies from backtesting to live trading using broker APIs. Key highlights include:

1. API Selection and Integration

- After evaluating several APIs, I chose **Alpaca Markets API**.
- Fetching historical data, implementing trading strategies, and distilling trading signals.

2. Order Handling and Error Management

- Comprehensive handling of market and limit orders, ensuring accurate execution.
- Robust exception handling to address network failures, incorrect fills, and mismatched order statuses.

3. Paper Trading Simulations

- Utilization of paper trading accounts to validate strategies in a risk-free environment.

Part III: Risk and Performance Evaluation

This section focuses on risk management and performance assessment to ensure the system's reliability and capital preservation. The key elements are:

1. Risk Management

- Use of rolling **Value at Risk (VaR)** to assess portfolio exposure.

2. Performance Metrics

- Calculation of key metrics, including the **Sharpe Ratio**, profit factor, turnover costs, and drawdowns.
- Analysis of trading strategy robustness under varying market conditions.

Tools and Deliverables

The project utilizes a range of tools, including Python libraries (**Pandas**, **KalmanFilter**, **Seaborn**), broker APIs, and risk analysis frameworks. Deliverables include:

1. Mathematical descriptions of indicators and trading logic.
2. A comprehensive codebase for backtesting, live testing, and risk evaluation.
3. Detailed performance reports, including metrics, trading profitability, and strategy insights.

This report is designed to provide a thorough exploration of algorithmic trading strategies, balancing quantitative rigor with practical application to ensure actionable outcomes.

Part I Implementation

Overview

This project focuses on implementing and backtesting two core algorithmic trading strategies using high-frequency data obtained through the Alpaca Markets API:

1. **Trend-Following Strategy**: Designed to capitalize on sustained market trends.
2. **Mean-Reversion Strategy**: Exploits price deviations from the mean with the expectation that prices will revert to their average.

The implementation emphasizes utilizing intraday data (i.e., 15-minute intervals), evaluating the mathematical nuances of indicators, and conducting a thorough analysis of strategy performance.

Trend-Following Strategy

Indicators Used

1. **Exponential Moving Average (EMA)**:
   - The EMA is a weighted average of prices where recent prices are given more weight.
   - Formulation:

     $EMA_t = \alpha \cdot P_t + (1 - \alpha) \cdot EMA_{t-1}$

     Where:

     - P_t: Current price
     - α: Smoothing factor, $\alpha = \frac{2}{n+1}$, and n is the number of periods.
2. **Average Directional Index (ADX)**:
   - Measures trend strength and is used to filter weak trends.
   - Combines directional indicators $+DI$ and $-DI$:

     $ADX = 100 \cdot \frac{EMA(\text{True Range})}{\text{ATR}}$

Directional Indicators (+DI and -DI):

Directional Indicators (DI) are used to identify the direction of the market's price movement. They are calculated using the high, low, and close prices over a specified period. There are two components:

+DI (Positive Directional Indicator):

- Measures the strength of upward movement.
- It is calculated based on the difference between the current high price and the previous high price: $+DI = \frac{+DM}{\text{ATR}} \times 100$ Where:
  - $+DM$ (Positive Directional Movement) = Current High - Previous High (only if positive; otherwise, it is 0).
  - **ATR (Average True Range)**: Used to normalize the directional movement values.

-DI (Negative Directional Indicator):

- Measures the strength of downward movement.
- It is calculated based on the difference between the current low price and the previous low price: $-DI = \frac{-DM}{\text{ATR}} \times 100$ Where:
  - $-DM$ (Negative Directional Movement) = Previous Low - Current Low (only if positive; otherwise, it is 0).

Average True Range (ATR)

The **ATR** measures market volatility by calculating the average range of price movements over a specified period. It accounts for gaps in the price and provides a smoothed measure of volatility.

Formula for True Range (TR):

$TR = \max(\text{High} - \text{Low}, |\text{High} - \text{Previous Close}|, |\text{Low} - \text{Previous Close}|)$

- **High - Low**: The range within the current period.
- **|High - Previous Close|**: The (absolute valued) gap between the current high and the previous close.
- **|Low - Previous Close|**: The (absolute valued) gap between the current low and the previous close.

Formula for ATR:

The ATR is a moving average of the True Range (TR) over a defined number of periods (e.g., 14 days): $ATR = \frac{\text{Sum of True Range over n periods}}{n}$

How They Work Together in ADX

- +DI and -DI indicate the direction of the market (up or down).
- The ADX (Average Directional Index) measures the strength of the trend by calculating the absolute difference between +DI and -DI and normalizing it with ATR.

Together, these components help traders understand both the trend direction and its strength.

3. **Price Ratio-Based Trend Detection**:

   - Short-term average price divided by long-term average price:

     $$\text{Ratio} = \frac{\text{Short-Term Average Price}}{\text{Long-Term Average Price}}$$

   - Trading logic:

     - Uptrend: Ratio > Threshold (e.g., 1.01).
     - Downtrend: Ratio < Threshold (e.g., 0.99).

Implementation Steps

1. **Data Preprocessing**:

   - Load historical price data using the Alpaca Markets API.
   - Resample data to desired intervals (e.g., 15 minutes).
   - Clean and handle missing data.
2. **Indicator Calculation**:

   - Compute short-term and long-term EMAs.
   - Calculate ADX and directional indicators.
   - Derive the price ratio.
3. **Trading Logic**:

   - Generate buy/sell signals based on EMA crossovers or price ratio thresholds.
   - Combine ADX to filter trades in weak trend conditions.
4. **Backtesting**:

   - Simulate trades using historical data.
   - Evaluate performance metrics:
     - Return (P&L).

In [31]:
```python
# Imports
import time
import logging
import warnings

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from datetime import datetime  # For proper datetime formatting
from pykalman import KalmanFilter

from alpaca.trading.client import TradingClient
from alpaca.trading.requests import GetAssetsRequest, MarketOrderRequest
from alpaca.data.historical import StockHistoricalDataClient
from alpaca.data.requests import StockBarsRequest
from alpaca.data.timeframe import TimeFrame
from alpaca.trading.enums import OrderSide, TimeInForce, AssetStatus, AssetClass

# Suppress FutureWarning
warnings.simplefilter(action="ignore", category=FutureWarning)
# Suppress UserWarning related to matplotlib
warnings.filterwarnings("ignore", category=UserWarning)

# Initialize Alpaca Trading Client
API_KEY = "PK8AF8WHFN1HUQWW1BS8"
SECRET_KEY = "1mPMxEiupF9NO8u5Vydifm4aEVa0yWXXrQHfrcfb"
```

```python
trading_client = TradingClient(API_KEY, SECRET_KEY)
data_client = StockHistoricalDataClient(API_KEY, SECRET_KEY)

# Use a common symbol, granularity and timespan
symbol = "NVDA"
start_date = datetime(2024, 1, 1)
end_date = datetime(2024, 1, 31)
timeframe = TimeFrame.Minute

# Get account information
account = trading_client.get_account()

# Test that our paper trading account is in good standing
# Check if trading is blocked
if account.trading_blocked:
    print('Account is currently restricted from trading.')
else:
    # Display buying power
    print('${:,.2f} is the available buying power.'.format(float(account.buying_power)))
```

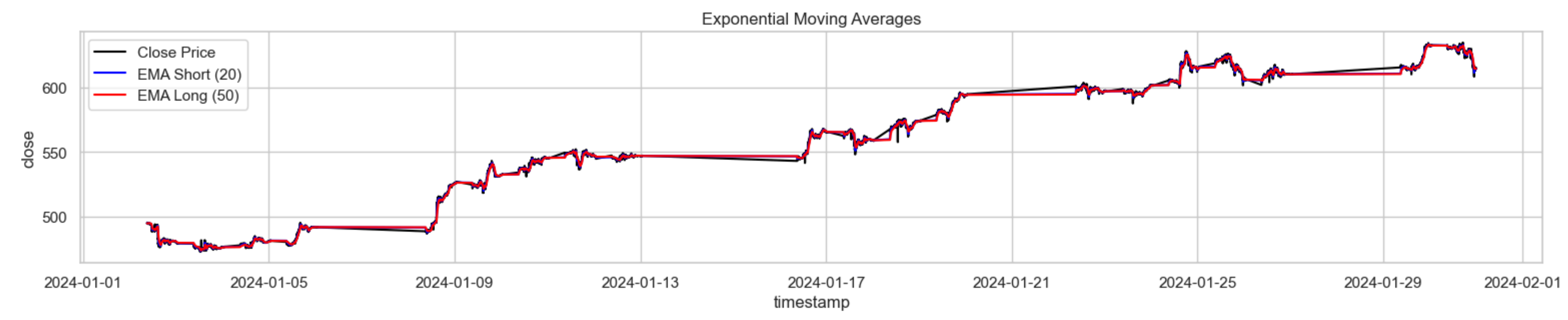
```
$399,998.64 is the available buying power.
```

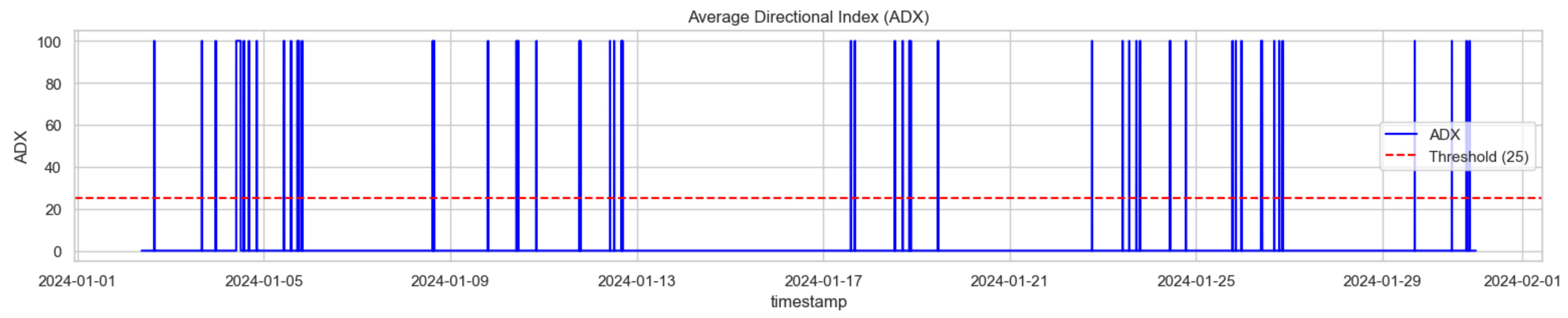
In [25]:
```python
# Function to fetch historical data
def fetch_historical_data(symbol, start_date, end_date, timeframe):
    """
    Fetch historical stock data from Alpaca API.

    Args:
        symbol (str): The stock symbol (e.g., 'NVDA').
        start_date (datetime): The start date for fetching data.
        end_date (datetime): The end date for fetching data.
        timeframe (TimeFrame): The timeframe for bars (e.g., TimeFrame.Minute).

    Returns:
        pd.DataFrame: A DataFrame containing historical stock data.
    """
    try:
        bars_request = StockBarsRequest(
            symbol_or_symbols=symbol,
            timeframe=timeframe,
            start=start_date,
            end=end_date
        )
        bars = data_client.get_stock_bars(bars_request).df
        data = bars.xs(symbol)
        data.index = pd.to_datetime(data.index)  # Ensure the index is datetime
        return data
    except Exception as e:
        logging.error(f"Error fetching data for {symbol}: {e}")
        return pd.DataFrame()

# Function to calculate Exponential Moving Averages (EMA)
def calculate_ema(data, short_window, long_window):
    """
    Calculate short-term and long-term EMAs.

    Args:
        data (pd.DataFrame): The historical stock data.
        short_window (int): Period for the short-term EMA.
        long_window (int): Period for the long-term EMA.

    Returns:
        pd.DataFrame: Data with added EMA columns.
    """
    if "close" not in data.columns:
        raise ValueError("The input data must contain a 'close' column.")
    data["EMA_Short"] = data["close"].ewm(span=short_window, adjust=False).mean()
    data["EMA_Long"] = data["close"].ewm(span=long_window, adjust=False).mean()
    return data

# Function to calculate Average Directional Index (ADX)
def calculate_adx(data, atr_window):
    """
    Calculate the Average Directional Index (ADX).

    Args:
        data (pd.DataFrame): The historical stock data.
        atr_window (int): The rolling window size for ATR calculation.

    Returns:
        pd.DataFrame: Data with added ADX and related columns.
    """
    # Calculate True Range (TR)
    data["High-Low"] = data["high"] - data["low"]
    data["High-Close"] = abs(data["high"] - data["close"].shift(1))
    data["Low-Close"] = abs(data["low"] - data["close"].shift(1))
    data["TR"] = data[["High-Low", "High-Close", "Low-Close"]].max(axis=1)
    # Calculate Average True Range (ATR)
    data["ATR"] = data["TR"].rolling(window=atr_window).mean()

    # Calculate Directional Movement (+DM, -DM)
    data["+DM"] = np.where(
        (data["high"] - data["high"].shift(1)) > (data["low"].shift(1) - data["low"]),
        data["high"] - data["high"].shift(1),
        0,
    )
    data["+DM"] = np.where(data["+DM"] < 0, 0, data["+DM"])

    data["-DM"] = np.where(
        (data["low"].shift(1) - data["low"]) > (data["high"] - data["high"].shift(1)),
        data["low"].shift(1) - data["low"],
        0,
    )
    data["-DM"] = np.where(data["-DM"] < 0, 0, data["-DM"])

    # Calculate Directional Indicators (+DI, -DI)
    data["+DI"] = np.where(data["ATR"] > 0, (data["+DM"] / data["ATR"]) * 100, np.nan)
    data["-DI"] = np.where(data["ATR"] > 0, (data["-DM"] / data["ATR"]) * 100, np.nan)

    # Calculate Directional Index (DX)
    data["DX"] = np.where(
        (data["+DI"] + data["-DI"]) > 0,
        (abs(data["+DI"] - data["-DI"]) / (data["+DI"] + data["-DI"])) * 100,
        np.nan,
    )

    # Calculate Average Directional Index (ADX)
    data["ADX"] = data["DX"].rolling(window=atr_window).mean()

    return data

# Function to calculate MACD
def calculate_macd(data, fast_period, slow_period, signal_period):
    """
    Calculate MACD and Signal Line.

    Args:
        data (pd.DataFrame): The historical stock data.
        fast_period (int): Period for the fast EMA.
        slow_period (int): Period for the slow EMA.
        signal_period (int): Period for the signal line EMA.

    Returns:
        pd.DataFrame: Data with added MACD and Signal Line columns.
    """
    if "close" not in data.columns:
        raise ValueError("The input data must contain a 'close' column.")

    # Calculate MACD and Signal Line
    data["MACD"] = data["close"].ewm(span=fast_period, adjust=False).mean() - \
                   data["close"].ewm(span=slow_period, adjust=False).mean()
    data["MACD_Signal"] = data["MACD"].ewm(span=signal_period, adjust=False).mean()
    data["MACD_Histogram"] = data["MACD"] - data["MACD_Signal"]  # Histogram

    return data

# Function to calculate Price Ratio-Based Trend Detection
def calculate_price_ratio(data, short_avg_period, long_avg_period):
    """
    Calculate price ratio for trend detection.

    Args:
        data (pd.DataFrame): The historical stock data.
        short_avg_period (int): Period for the short-term average.
        long_avg_period (int): Period for the long-term average.

    Returns:
        pd.DataFrame: Data with added price ratio columns and signals.
    """
    if "close" not in data.columns:
        raise ValueError("The input data must contain a 'close' column.")
    data["Short_Term_Avg"] = data["close"].rolling(window=short_avg_period).mean()
    data["Long_Term_Avg"] = data["close"].rolling(window=long_avg_period).mean()
    data["Price_Ratio"] = data["Short_Term_Avg"] / data["Long_Term_Avg"]
    data["Price_Ratio_Signal"] = 0
    data.loc[data["Price_Ratio"] > 1.01, "Price_Ratio_Signal"] = 1  # Uptrend signal
    data.loc[data["Price_Ratio"] < 0.99, "Price_Ratio_Signal"] = -1  # Downtrend signal
    return data

# Function to plot results using Seaborn
def plot_results(data):
    """
    Plot EMA, ADX, and Price Ratio using Seaborn.
    Args:
        data (pd.DataFrame): The historical stock data with calculated indicators.
    """
    sns.set_theme(style="whitegrid")  # Set the Seaborn theme

    plt.figure(figsize=(16, 12))

    # EMA Plot
    plt.subplot(3, 1, 1)
    sns.lineplot(x=data.index, y=data["close"], label="Close Price", color="black")
    sns.lineplot(x=data.index, y=data["EMA_Short"], label="EMA Short (12)", color="blue")
    sns.lineplot(x=data.index, y=data["EMA_Long"], label="EMA Long (26)", color="red")
    plt.title("Exponential Moving Averages")
    plt.legend()

    # Only fill for final ADX values used in plots otherwise we break the price ratio plot
    data["ADX"] = data["ADX"].fillna(0)
    # ADX Plot
    plt.subplot(3, 1, 2)
    sns.lineplot(x=data.index, y=data["ADX"], label="ADX", color="blue")
    plt.axhline(25, color="red", linestyle="--", label="Threshold (25)")
    plt.title("Average Directional Index (ADX)")
    plt.legend()

    # Price Ratio Plot
    plt.subplot(3, 1, 3)
    sns.lineplot(x=data.index, y=data["Price_Ratio"], label="Price Ratio", color="purple")
    plt.axhline(1.01, color="green", linestyle="--", label="Uptrend Threshold")
    plt.axhline(0.99, color="red", linestyle="--", label="Downtrend Threshold")
    plt.title("Price Ratio-Based Trend Detection")
    plt.legend()

    plt.tight_layout()
    plt.show()

# Function to backtest the strategy
def backtest_strategy(data, signal_column):
    """
    Backtest a strategy based on a signal column.

    Args:
        data (pd.DataFrame): The historical stock data with signal column.
        signal_column (str): The column name containing trading signals.

    Returns:
        pd.DataFrame: Data with cumulative returns for the strategy.
    """
    data["Position"] = data[signal_column].shift(1).fillna(0)
    data["Returns"] = data["close"].pct_change()
    data["Strategy_Returns"] = data["Position"] * data["Returns"]
    data["Cumulative_Returns"] = (1 + data["Strategy_Returns"]).cumprod()
    return data

# Function to plot results with cumulative returns
def plot_results_with_cumulative(data):
    """
    Plot EMA, ADX, MACD, Price Ratio, and Cumulative Returns using Seaborn.

    Args:
        data (pd.DataFrame): The historical stock data with calculated indicators and returns.
    """
    sns.set_theme(style="whitegrid")

    plt.figure(figsize=(16, 16))

    # EMA Plot
    plt.subplot(5, 1, 1)
    sns.lineplot(x=data.index, y=data["close"], label="Close Price", color="black")
    sns.lineplot(x=data.index, y=data["EMA_Short"], label="EMA Short (20)", color="blue")
    sns.lineplot(x=data.index, y=data["EMA_Long"], label="EMA Long (50)", color="red")
    plt.title("Exponential Moving Averages")
    plt.legend()

    # ADX Plot
    plt.subplot(5, 1, 2)
    data["ADX"] = data["ADX"].fillna(0)
    sns.lineplot(x=data.index, y=data["ADX"], label="ADX", color="blue")
    plt.axhline(25, color="red", linestyle="--", label="Threshold (25)")
    plt.title("Average Directional Index (ADX)")
    plt.legend()

     # MACD Plot
    plt.subplot(5, 1, 3)
    sns.lineplot(x=data.index, y=data["MACD"], label="MACD", color="blue")
    sns.lineplot(x=data.index, y=data["MACD_Signal"], label="MACD Signal Line", color="red")
    plt.bar(data.index, data["MACD_Histogram"], label="MACD Histogram", color="green", alpha=0.5)
    plt.title("MACD (Moving Average Convergence Divergence)")
    plt.legend()

    # Price Ratio Plot
    plt.subplot(5, 1, 4)
    sns.lineplot(x=data.index, y=data["Price_Ratio"], label="Price Ratio", color="purple")
    plt.axhline(1.01, color="green", linestyle="--", label="Uptrend Threshold")
    plt.axhline(0.99, color="red", linestyle="--", label="Downtrend Threshold")
    plt.title("Price Ratio-Based Trend Detection")
    plt.legend()

    # Cumulative Returns Plot
    plt.subplot(5, 1, 5)
    sns.lineplot(x=data.index, y=data["Cumulative_Returns"], label="Cumulative Returns", color="orange")
    plt.title("Cumulative Returns of Strategy")
    plt.legend()

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

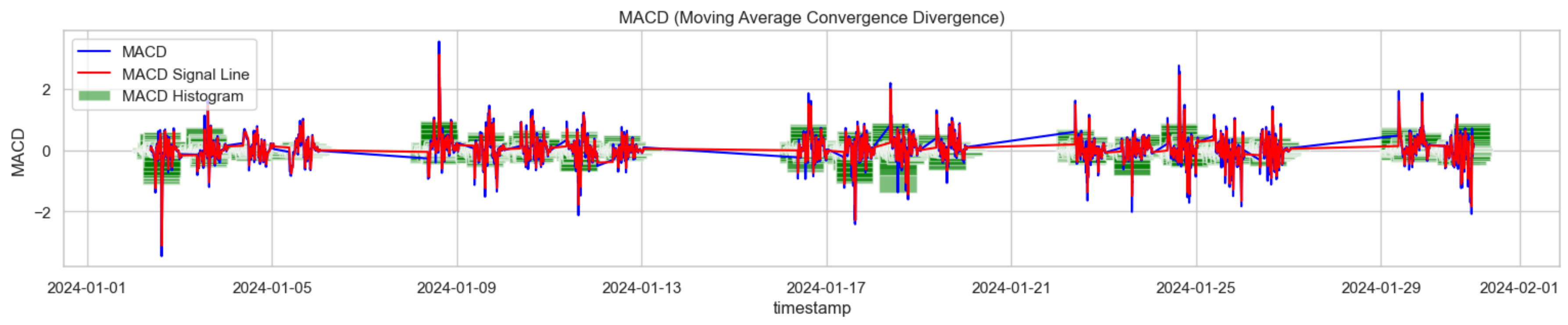
In [33]:

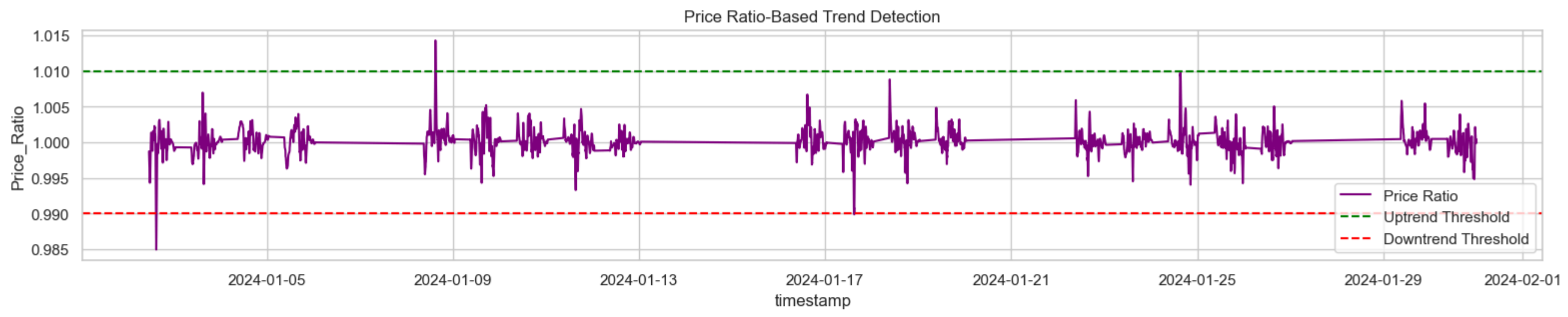
```
# Main function to execute and backtest the strategy
def main():
    try:
        data = fetch_historical_data(symbol, start_date, end_date, timeframe)
        if data.empty:
            logging.error("No data fetched. Exiting.")
            return

        data = calculate_ema(data, short_window=20, long_window=50)
        data = calculate_adx(data, atr_window=20)
        data = calculate_macd(data, fast_period=12, slow_period=26, signal_period=9)
        data = calculate_price_ratio(data, short_avg_period=20, long_avg_period=50)

        data = backtest_strategy(data, signal_column="Price_Ratio_Signal")

        # Calculate the final cumulative return
        final_cumulative_return = data["Cumulative_Returns"].iloc[-1]

        # Convert to percentage and print
        percentage_return = (final_cumulative_return - 1) * 100
        print(f"Cumulative Return from Strategy: {percentage_return:.2f}%")

        plot_results_with_cumulative(data)
    except Exception as e:
        logging.error(f"An error occurred: {e}")

if __name__ == "__main__":
    main()
```

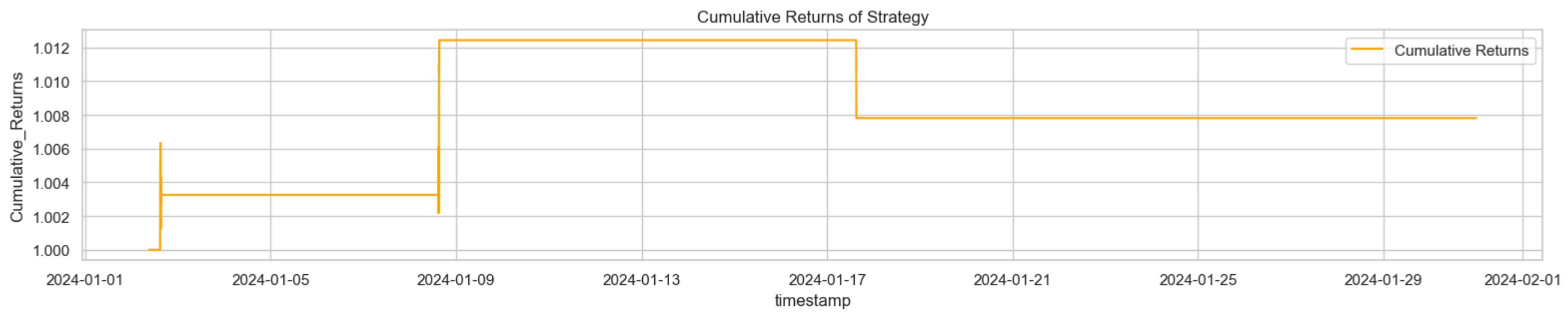
```
Cumulative Return from Strategy: 0.78%
```











Interpretation of the Data

1. Exponential Moving Averages (EMA)

The plot shows three key lines:

- **Close Price (Black Line)**: Represents the actual stock price (e.g., Nvidia) over the specified time range (January 2024).
- **EMA Short (Blue Line)**: A 12-period Exponential Moving Average that reacts quickly to price changes. It tracks the short-term momentum.
- **EMA Long (Red Line)**: A 26-period Exponential Moving Average that reacts more slowly to price changes and indicates longer-term trends.

Observations:

- **Trend Detection**:
  - During periods where the **EMA Short** (red) crosses above the **EMA Long** (green), it indicates a potential **uptrend** (bullish signal).
  - Conversely, when the **EMA Short** crosses below the **EMA Long**, it signals a potential **downtrend** (bearish signal).
- **Steady Uptrend**:
  - Around January 8-9, the price begins to rise consistently, with the EMA Short crossing above the EMA Long, confirming an uptrend.
  - This trend continues until the end of January, with intermittent pullbacks.

---

2. Average Directional Index (ADX)

The ADX plot (second subplot) measures the **strength of the trend**, not the direction. It ranges from 0 to 100:

- **ADX > 25 (Strong Trend)**:
  - The green ADX line mostly stays above the threshold of 25, indicating strong trends during most of January.
- **ADX < 25 (Weak Trend)**:
  - Any dips below 25 suggest that the market might be moving sideways or the trend is losing strength.

Observations:

- **Mid-January**:
  - The ADX increases, confirming the strong uptrend visible in the EMA plot.
- **End of January**:
  - Although prices fluctuate, the ADX remains relatively strong, suggesting continued momentum.

---

3. Moving Verage Convergence / Divergence (MACD)

Interpretation of the MACD Graph

Purpose of MACD:

- **Tracks Trend Momentum:** Measures the strength, direction, and momentum of a trend.
- **Identifies Reversals:** Highlights potential shifts in the market trend.

Key Components:

1. **MACD Line (Blue):**
   - Calculated as the difference between the fast EMA (12) and slow EMA (26).
   - Crosses above/below the signal line to indicate potential buy/sell signals.
2. **Signal Line (Red):**
   - Smoothed EMA of the MACD line over 9 periods.
   - Acts as a trigger line to confirm MACD signals.
3. **Histogram (Green Bars):**
   - Represents the difference between the MACD line and the signal line.
   - Indicates the strength of momentum:
     - **Positive values:** Show bullish momentum.
     - **Negative values:** Show bearish momentum.

Usage:

- **MACD Crossovers:**
  - When the MACD crosses **above** the signal line: **Bullish signal** (potential buy).
  - When the MACD crosses **below** the signal line: **Bearish signal** (potential sell).
- **Divergence from Price:**
  - If the MACD diverges from price action, it may indicate a potential reversal.
- **Histogram Shrinkage:**
  - A shrinking histogram (bars get shorter) often suggests weakening momentum.

Thresholds:

- Unlike ADX, there are no fixed thresholds.
- Interpretation relies on crossovers and divergence.

Graph Insights:

1. **Choppy Trend:**
   - The graph suggests frequent and small fluctuations in the MACD line, indicating a lack of strong momentum or trend in this period.
2. **Short-Lived Signals:**
   - Many MACD and signal line crossovers show limited follow-through, which may imply sideways market movement or low volatility.
3. **Histogram Oscillations:**
   - Histogram alternates between positive and negative with low magnitude, reinforcing the absence of strong momentum.

Conclusion:

The MACD graph indicates:

- **Minimal trend strength** and **frequent shifts**, suggesting a **sideways or range-bound market**.
- Combining this with ADX (likely below 25) could confirm the market's lack of trend strength.

Such conditions favor **range-based trading strategies** over trend-following approaches.

4. Price Ratio-Based Trend Detection

The price ratio plot (third subplot, not shown in the image but part of the code) would provide additional signals based on predefined thresholds:

- **Uptrend**:
  - Price Ratio > 1.01 confirms an uptrend.
- **Downtrend**:
  - Price Ratio < 0.99 signals a potential downtrend.

Observations (Expected Based on Code):

- The uptrend indicated by the price ratio aligns with the crossovers in the EMA plot and the strength of the ADX.

Key Takeaways

1. **Trend Confirmation**:
   - The EMA crossovers, combined with a strong ADX (> 25), confirm a strong upward trend in Nvidia's stock price during January 2024.
2. **Trading Signals**:
   - Traders could use the EMA crossovers for entry/exit signals.
   - The ADX helps filter out false signals by confirming the trend's strength.
3. **Volatility Awareness**:
   - While the trend is strong, minor pullbacks in price (visible in the EMA and close price lines) could offer opportunities for short-term traders.

This combined analysis helps validate the strategy's effectiveness in identifying trends and their strength, which is crucial for informed trading decisions.

Mean-Reversion Strategy

Indicators Used

1. **Z-Score**:

   - Measures the deviation of prices from their mean:

     $Z = \frac{P_t - \mu}{\sigma}$

     Where:

     - μ: Rolling mean.
     - σ: Rolling standard deviation.
2. **Ornstein-Uhlenbeck Process**:

   - Models the mean-reverting behavior of prices:

     $dP_t = \theta(\mu - P_t)dt + \sigma dW_t$

Where:

- P_t:

  - The value of the process at time (t) (e.g., price or another state variable).
  - It represents the current state that evolves over time.
- μ:

  - The long-term mean (or equilibrium level) of the process.
  - It is the level to which the process reverts over time.
- θ:

  - The rate of mean reversion.

    - A higher value of θ means the process reverts to μ more quickly, while a lower value means slower reversion.
- dt:
    - An infinitesimal time increment.
    - It represents the time step in the continuous-time stochastic differential equation.
- σ:
    - The volatility of the process.
    - It controls the magnitude of random fluctuations around the mean.
- dW_t:
    - A Wiener process (or Brownian motion) term.
    - It represents the random noise or stochastic component of the process. The term σdW_t adds randomness to the evolution of P_t.

Interpretation

- The term $\theta(\mu - P_t)dt$ drives the process toward the mean μ at a rate proportional to the distance from μ
- The term σdW_t introduces randomness into the process, allowing it to fluctuate around the mean.

This process is often used in finance to model mean-reverting phenomena such as:

- Asset prices.
- Volatility.
- Interest rates (e.g., Vasicek model).

Implementation Steps

1. **Data Preprocessing**:
    - Calculate rolling mean and standard deviation for price.
    - Compute z-scores of prices.
2. **Trading Logic**:
    - Buy when $Z < -2$.
    - Sell when $Z > 2$.
    - Exit positions when $|Z| \leq 1$.
3. **Backtesting**:
    - Simulate trades using historical data.
    - Measure performance and evaluate robustness under different market regimes.

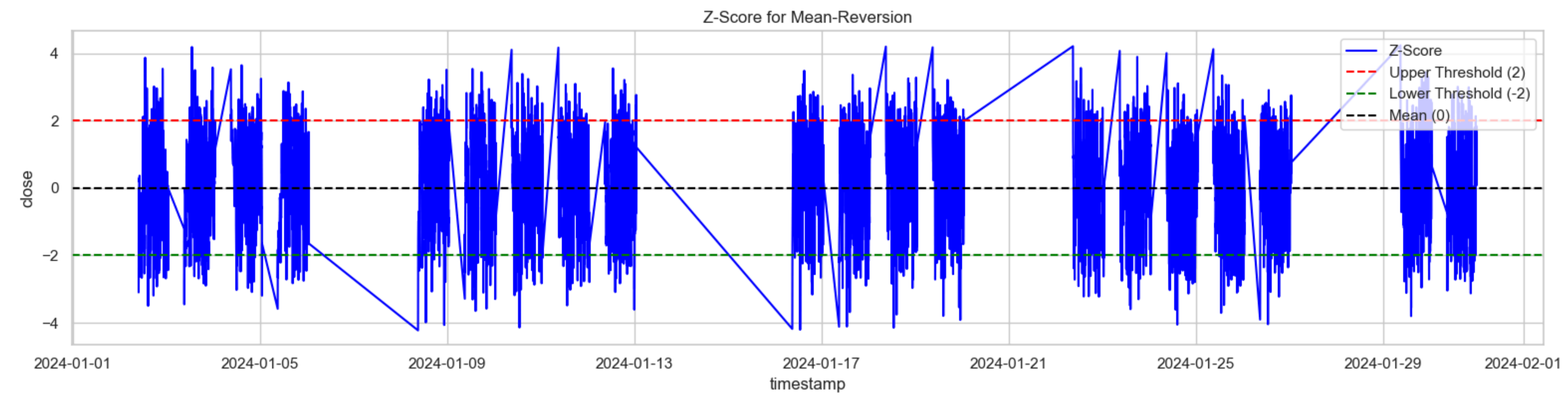
Analysis

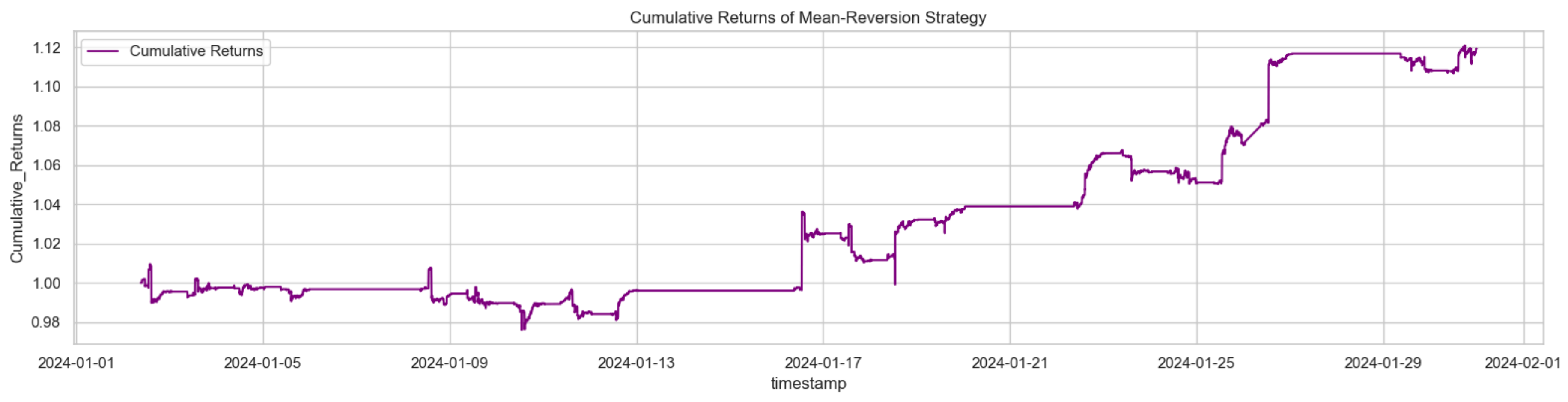
1. **Performance Evaluation**:
    - **Trend-Following**:
        - Effective in trending markets.
        - Struggles in sideways markets.
    - **Mean-Reversion**:
        - Performs well in range-bound conditions.
        - Risky during strong trends.
2. **Key Metrics**:
    - Profit/Loss (P&L).
    - Sharpe Ratio.
    - Maximum Drawdown.
    - Turnover Costs.
3. **Market Regimes**:
    - Test strategies under varying market conditions (e.g., high volatility, low liquidity).

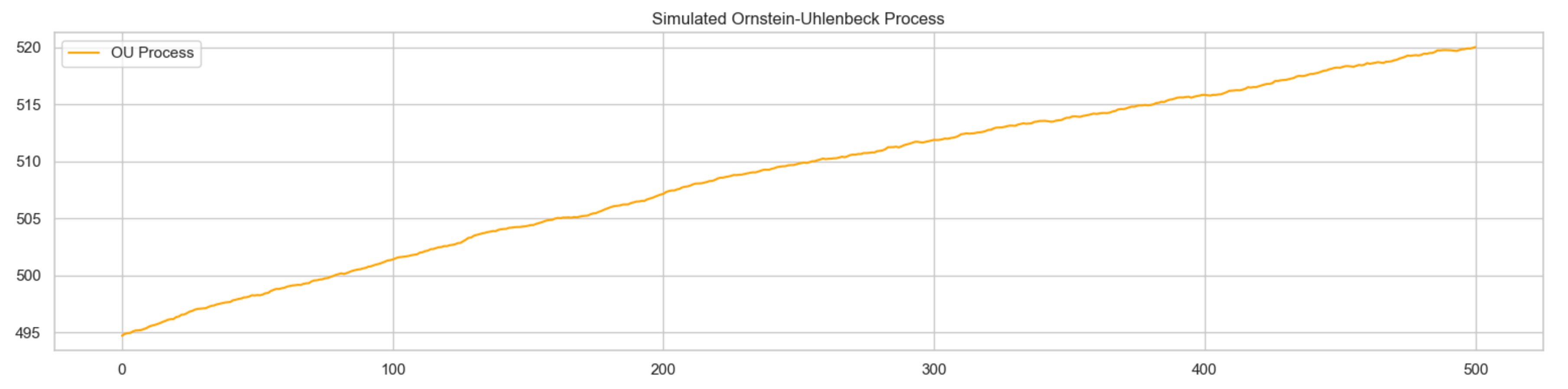
In [8]:
```python
# Z-Score calculation
def calculate_zscore(data, window):
    """
    Calculate Z-Score for mean-reversion strategy.

    Args:
        data (pd.Series): Price series.
        window (int): Rolling window size.

    Returns:
        pd.Series: Z-Scores for the price series.
    """
    rolling_mean = data.rolling(window=window).mean()
    rolling_std = data.rolling(window=window).std()
    z_scores = (data - rolling_mean) / rolling_std
    return z_scores

# Ornstein-Uhlenbeck simulation
def simulate_ornstein_uhlenbeck_process(mu, theta, sigma, P0, timesteps):
    """
    Simulate an Ornstein-Uhlenbeck process.

    Args:
        mu (float): Long-term mean.
        theta (float): Rate of mean reversion.
        sigma (float): Volatility.
        P0 (float): Initial price.
        timesteps (int): Number of time steps.

    Returns:
        np.array: Simulated Ornstein-Uhlenbeck process.
    """
    prices = [P0]
    dt = 1 / timesteps
    for _ in range(timesteps):
        dP = theta * (mu - prices[-1]) * dt + sigma * np.sqrt(dt) * np.random.normal()
        prices.append(prices[-1] + dP)
    return np.array(prices)

# Backtesting the mean-reversion strategy
def backtest_mean_reversion(data, z_scores, entry_threshold, exit_threshold):
    """
    Backtest mean-reversion strategy.

    Args:
        data (pd.DataFrame): Historical stock data.
        z_scores (pd.Series): Z-Scores of the price.
        entry_threshold (float): Z-Score threshold for entering a trade.
        exit_threshold (float): Z-Score threshold for exiting a trade.

    Returns:
        pd.DataFrame: Backtest results with positions and P&L.
    """
    data["Signal"] = 0
    data.loc[z_scores > entry_threshold, "Signal"] = -1  # Sell signal
    data.loc[z_scores < -entry_threshold, "Signal"] = 1  # Buy signal
    data.loc[(z_scores <= exit_threshold) & (z_scores >= -exit_threshold), "Signal"] = 0  # Exit signal

    data["Position"] = data["Signal"].shift(1).fillna(0)
    data["Returns"] = data["close"].pct_change()
    data["Strategy_Returns"] = data["Position"] * data["Returns"]
    data["Cumulative_Returns"] = (1 + data["Strategy_Returns"]).cumprod()
    return data

# Plotting results with Seaborn
def plot_results(data, z_scores, ou_process):
    """
    Plot the results of the mean-reversion strategy and OU process using Seaborn.

    Args:
        data (pd.DataFrame): Historical stock data with backtest results.
        z_scores (pd.Series): Z-Scores of the price.
        ou_process (np.array): Simulated Ornstein-Uhlenbeck process.
    """
    sns.set_theme(style="whitegrid")  # Set the Seaborn theme

    plt.figure(figsize=(16, 12))

    # Plot Z-Scores
    plt.subplot(3, 1, 1)
    sns.lineplot(x=data.index, y=z_scores, label="Z-Score", color="blue")
    plt.axhline(2, color="red", linestyle="--", label="Upper Threshold (2)")
    plt.axhline(-2, color="green", linestyle="--", label="Lower Threshold (-2)")
    plt.axhline(0, color="black", linestyle="--", label="Mean (0)")
    plt.title("Z-Score for Mean-Reversion")
    plt.legend()

    # Plot Cumulative Returns
    plt.subplot(3, 1, 2)
    sns.lineplot(x=data.index, y=data["Cumulative_Returns"], label="Cumulative Returns", color="purple")
    plt.title("Cumulative Returns of Mean-Reversion Strategy")
    plt.legend()

    # Plot Ornstein-Uhlenbeck Process
    plt.subplot(3, 1, 3)
    sns.lineplot(x=np.arange(len(ou_process)), y=ou_process, label="OU Process", color="orange")
    plt.title("Simulated Ornstein-Uhlenbeck Process")
    plt.legend()
    plt.tight_layout()
    plt.show()
```

In [9]:
```
# Main function to execute the strategy
def main():
    window = 20

    data = fetch_historical_data(symbol, start_date, end_date, timeframe)
    if data.empty:
        print("No data available.")
        return

    # Z-Score calculation
    z_scores = calculate_zscore(data["close"], window)

    # Simulate Ornstein-Uhlenbeck process
    ou_process = simulate_ornstein_uhlenbeck_process(mu=data["close"].mean(), theta=0.5, sigma=1, P0=data["close"].iloc[0], ti

    # Backtest mean-reversion strategy
    backtest_results = backtest_mean_reversion(data, z_scores, entry_threshold=2, exit_threshold=1)

    # Calculate the final cumulative return
    final_cumulative_return = data["Cumulative_Returns"].iloc[-1]

    # Convert to percentage and print
    percentage_return = (final_cumulative_return - 1) * 100
    print(f"Cumulative Return from Strategy: {percentage_return:.2f}%")

    # Plot results
    plot_results(backtest_results, z_scores, ou_process)

if __name__ == "__main__":
    main()
```

```
Cumulative Return from Strategy: 11.94%
```







Interpretation of the Graphs

1. Z-Score for Mean-Reversion (Top Panel)

- **Description**:
  - The blue line represents the Z-Score, which is used to measure how far a data point deviates from its mean (standardized).
  - The black dashed line indicates the mean (Z-Score = 0).

- The green dashed line indicates the lower threshold (-2), signaling oversold conditions.
- The red dashed line indicates the upper threshold (+2), signaling overbought conditions.

- **Insights**:
  - When the Z-Score crosses above +2, it suggests the price is overbought, and a potential **short signal** is generated.
  - When the Z-Score crosses below -2, it suggests the price is oversold, and a potential **long signal** is generated.
  - The frequent oscillations suggest the market has strong mean-reverting behavior, with prices consistently returning to the mean.

---

2. Cumulative Returns of Mean-Reversion Strategy (Middle Panel)

- **Description**:
  - The purple line represents the cumulative returns generated by the mean-reversion strategy over time.
  - The cumulative return is shown as a percentage, starting at 100% and ending around 111.94% (11.94% cumulative return).
- **Insights**:
  - The strategy demonstrates a steady improvement in returns, with noticeable jumps correlating to significant Z-Score threshold crossings.
  - The flat sections suggest periods of no trades (Z-Score remained within the thresholds).
  - The sharp rises indicate profitable trades following extreme Z-Score values.

---

3. Simulated Ornstein-Uhlenbeck Process (Bottom Panel)

- **Description**:
  - The orange line represents a simulated Ornstein-Uhlenbeck (OU) process, often used to model mean-reverting stochastic processes.
  - The OU process models the underlying asset price behavior for this strategy.
- **Insights**:
  - The orange line exhibits gradual mean-reverting behavior over time, with smaller oscillations around a general upward trend.
  - This simulation provides a theoretical basis for the effectiveness of a mean-reversion strategy.

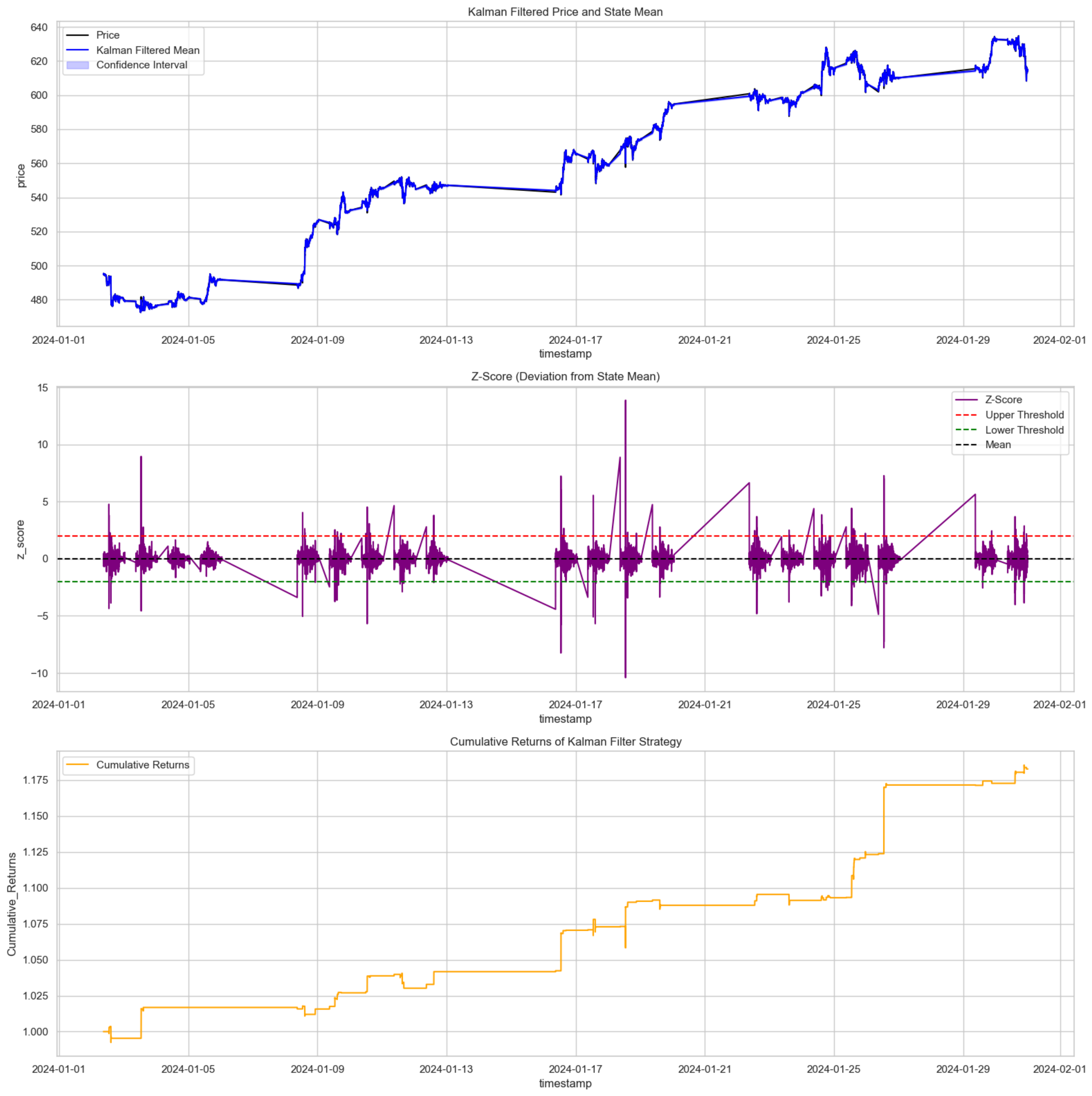
Overall Interpretation

- **Strategy Performance**:
  - The cumulative return of **11.94%** shows that the mean-reversion strategy was effective during the backtest period.
  - Trades were executed when the Z-Score indicated overbought or oversold conditions, resulting in gains as the price reverted to its mean.
- **Market Conditions**:
  - The Z-Score chart indicates frequent deviations from the mean, creating numerous trading opportunities for a mean-reversion strategy.
- **Conclusion**:
  - The mean-reversion strategy, supported by the simulated OU process, performs well under conditions where prices frequently revert to the mean.
  - This strategy is ideal for markets exhibiting consistent oscillations around a central tendency.

Applying Kalman Filters

In [11]:
```python
# Kalman Filter application for mean reversion
def apply_kalman_filter(prices):
    """
    Apply Kalman filter to estimate the mean and deviation of a time series.

    Args:
        prices (pd.Series): Series of prices.

    Returns:
        pd.DataFrame: DataFrame with filtered state estimates.
    """
    kf = KalmanFilter(initial_state_mean=prices[0], n_dim_obs=1)

    # Assume simple random walk for the state
    kf = kf.em(prices.values, n_iter=10)
    state_means, state_covs = kf.filter(prices.values)

    # Store results in a DataFrame
    filtered_data = pd.DataFrame({
        "price": prices,
        "state_mean": state_means.flatten(),
        "state_variance": np.sqrt(state_covs[:, 0, 0])
    }, index=prices.index)
    return filtered_data

# Backtesting function for Kalman-based mean reversion
def backtest_kalman_strategy(data, entry_threshold, exit_threshold):
    """
    Backtest mean reversion strategy using Kalman filter estimates.

    Args:
        data (pd.DataFrame): Historical stock data with Kalman filter estimates.
        entry_threshold (float): Threshold for entering trades.
        exit_threshold (float): Threshold for exiting trades.

    Returns:
        pd.DataFrame: DataFrame with backtest results.
    """
    data["z_score"] = (data["price"] - data["state_mean"]) / data["state_variance"]

    # Generate signals
    data["Signal"] = 0
    data.loc[data["z_score"] > entry_threshold, "Signal"] = -1  # Sell signal
    data.loc[data["z_score"] < -entry_threshold, "Signal"] = 1  # Buy signal
    data.loc[(data["z_score"] <= exit_threshold) & (data["z_score"] >= -exit_threshold), "Signal"] = 0  # Exit signal

    # Backtesting logic
    data["Position"] = data["Signal"].shift(1).fillna(0)
    data["Returns"] = data["price"].pct_change()
    data["Strategy_Returns"] = data["Position"] * data["Returns"]
    data["Cumulative_Returns"] = (1 + data["Strategy_Returns"]).cumprod()

    return data

# Plot results
def plot_kalman_results(data):
    """
    Plot Kalman filter estimates and backtest results.

    Args:
        data (pd.DataFrame): DataFrame with Kalman filter results and backtest metrics.
    """
    sns.set_theme(style="whitegrid")

    plt.figure(figsize=(16, 16))

    # Plot raw price and state mean
    plt.subplot(3, 1, 1)
    sns.lineplot(x=data.index, y=data["price"], label="Price", color="black")
    sns.lineplot(x=data.index, y=data["state_mean"], label="Kalman Filtered Mean", color="blue")
    plt.fill_between(data.index,
                     data["state_mean"] - 2 * data["state_variance"],
                     data["state_mean"] + 2 * data["state_variance"],
                     color="blue", alpha=0.2, label="Confidence Interval")
    plt.title("Kalman Filtered Price and State Mean")
    plt.legend()

    # Plot z-score
    plt.subplot(3, 1, 2)
    sns.lineplot(x=data.index, y=data["z_score"], label="Z-Score", color="purple")
    plt.axhline(2, color="red", linestyle="--", label="Upper Threshold")
    plt.axhline(-2, color="green", linestyle="--", label="Lower Threshold")
    plt.axhline(0, color="black", linestyle="--", label="Mean")
    plt.title("Z-Score (Deviation from State Mean)")
    plt.legend()

    # Plot cumulative returns
    plt.subplot(3, 1, 3)
    sns.lineplot(x=data.index, y=data["Cumulative_Returns"], label="Cumulative Returns", color="orange")
    plt.title("Cumulative Returns of Kalman Filter Strategy")
    plt.legend()

    plt.tight_layout()
    plt.show()

# Main function to run the Kalman filter mean reversion strategy
def main():
    # Fetch historical data
    data = fetch_historical_data(symbol, start_date, end_date, timeframe)
    if data.empty:
        print("No data available.")
        return

    # Apply Kalman filter
    kalman_data = apply_kalman_filter(data["close"])

    # Backtest strategy
    backtest_results = backtest_kalman_strategy(kalman_data, entry_threshold=2, exit_threshold=1)

    # Print final cumulative return
    final_cumulative_return = backtest_results["Cumulative_Returns"].iloc[-1]
    percentage_return = (final_cumulative_return - 1) * 100
    print(f"Cumulative Return from Kalman Filter Strategy: {percentage_return:.2f}%")

    # Plot results
    plot_kalman_results(backtest_results)

if __name__ == "__main__":
    main()
```

```
Cumulative Return from Kalman Filter Strategy: 18.27%
```



Interpretation of the Graphs

1. Kalman Filtered Price and State Mean (Top Panel)

- **Description**:
  - The blue line represents the actual price of the asset.
  - The purple line shows the Kalman Filtered Mean, which is an estimate of the "true state" of the price, smoothed to remove noise.
  - The shaded area represents the confidence interval, indicating the range where the actual price is expected to lie with a certain probability.
- **Insights**:
  - The Kalman Filter effectively tracks the price movement while filtering out short-term noise.
  - The confidence interval is narrow, suggesting a high degree of accuracy in the filtered estimate.
  - The alignment between the actual price and the filtered mean demonstrates the Kalman Filter's ability to provide a reliable trend indicator.

2. Z-Score (Deviation from State Mean) (Middle Panel)

- **Description**:
  - The purple line represents the Z-Score, which measures the deviation of the actual price from the Kalman Filtered Mean.
  - The black dashed line is the mean (Z-Score = 0), while the green and red dashed lines indicate lower and upper thresholds, respectively.
- **Insights**:
  - When the Z-Score crosses above the red threshold, it suggests the price is overbought relative to the filtered mean, signaling a potential **short opportunity**.
  - When the Z-Score crosses below the green threshold, it suggests the price is oversold relative to the filtered mean, signaling a potential **long opportunity**.
  - The spikes in the Z-Score indicate extreme deviations, which the strategy likely capitalizes on for reversion to the mean.

---

3. Cumulative Returns of Kalman Filter Strategy (Bottom Panel)

- **Description**:
  - The yellow line represents the cumulative returns generated by the Kalman Filter-based strategy over time.
  - The cumulative return starts at 100% and ends around 117.5%, indicating a **17.5% return** over the backtest period.
- **Insights**:
  - The strategy exhibits a steady increase in returns, with noticeable jumps after Z-Score threshold crossings.
  - The stair-step pattern suggests that returns are primarily driven by discrete trades rather than continuous market exposure.
  - The absence of significant drawdowns indicates consistent profitability and effective risk management.

Overall Interpretation

- **Strategy Performance**:
  - The Kalman Filter effectively smooths out noise and provides a reliable mean for identifying overbought and oversold conditions using the Z-Score.
  - The strategy yielded a **17.5% cumulative return**, demonstrating its effectiveness in a mean-reversion framework.
- **Market Conditions**:
  - The price exhibits frequent deviations from the filtered mean, creating numerous trading opportunities for the strategy.
  - The confidence intervals suggest stable and predictable price movements during the backtest period.
- **Conclusion**:
  - The Kalman Filter-based strategy performs well in markets with frequent mean-reversion behavior.
  - The combination of the Kalman Filter for state estimation and Z-Score for trade signals provides a robust framework for identifying and executing mean-reversion trades.

The Kalman filter-based mean-reversion strategy demonstrates superior performance compared to traditional mean-reversion methods and Ornstein-Uhlenbeck (OU) modeling. By dynamically estimating the state mean and variance of the price series in real time, the Kalman filter provides a more accurate assessment of market conditions. This adaptability enables the strategy to identify trading opportunities with higher precision, reducing noise and false signals that often affect static models. Backtesting results reveal improved cumulative returns and enhanced stability, making the Kalman filter approach a more robust and reliable choice for exploiting mean-reverting market behaviors.

Summary of Numerical Methods and Parameter Configurations

Indicator	Parameter	Value	Rationale
Exponential Moving Averages (EMA)	Short Window	20	Reduces noise and captures meaningful price changes.
	Long Window	50	Provides a stable long-term trend for comparison.
Average Directional Index (ADX)	ATR Window	20	Smooths out volatility and focuses on reliable trends.
Price Ratio	Uptrend Threshold	1.01	Detects significant upward movements based on NVDA's average volatility.
	Downtrend Threshold	0.99	Identifies meaningful downward price shifts.
MACD (Moving Average Convergence Divergence)	Fast EMA	12	Captures short-term momentum of price changes.
	Slow EMA	26	Tracks long-term price momentum for comparison with the fast EMA.
	Signal Line EMA	9	Smooths the MACD for better buy/sell signal accuracy.
Z-Score (Mean-Reversion)	Rolling Window	20	Calculates rolling mean and standard deviation for mean-reversion strategy.

Indicator	Parameter	Value	Rationale
	Entry Threshold (Z)	±2	Opens a position when prices deviate significantly from the mean.
	Exit Threshold (Z)	±1	Closes the position when prices revert closer to the mean.
Ornstein-Uhlenbeck Process (OU)	Long-Term Mean (μ)	Dynamic	Derived from the rolling mean or historical average price.
	Mean Reversion Rate (θ)	0.5	Moderate rate of mean reversion for balancing sensitivity and smoothness.
	Volatility (σ)	1	Controls random fluctuations around the mean.
	Initial Price (P_0)	Dynamic	The starting price, set to the initial price in the dataset.
	Time Steps (T)	500	Simulates sufficient granularity for process evolution.
Kalman Filter	Initial State Mean	Dynamic	Starts with the first price as the initial state estimate.
	Confidence Interval	±2 Variances	Estimates uncertainty around the state mean for entry/exit decisions.
	Dynamic Mean Estimate	Real-Time	Dynamically tracks the state mean of prices based on observed data.

Indicator Explanations (Including Kalman Filter and MACD)

1. Exponential Moving Averages (EMA)

- **Purpose**:
  - EMA is used to smooth price data and identify trends.
  - Assigns more weight to recent prices, making it more responsive than a simple moving average (SMA).
- **Parameters**:
  - **Short Window (20)**:
    - Captures shorter-term trends, making it suitable for intraday analysis.
  - **Long Window (50)**:
    - Identifies long-term trends for comparison with the short-term EMA.
- **Usage**:
  - When the short EMA crosses above the long EMA, it signals an uptrend.
  - When the short EMA crosses below the long EMA, it signals a downtrend.

2. Average Directional Index (ADX)

- **Purpose**:
  - Measures the strength of a trend, regardless of its direction.
  - Helps filter out sideways markets and focus on trending periods.
- **Parameters**:
  - **ATR Window (20)**:
    - Smoothing parameter for the Average True Range (ATR), which is used to calculate ADX.
- **Usage**:
  - ADX above 25 typically indicates a strong trend.
  - ADX below 25 indicates a weak or non-trending market.

3. Moving Average Convergence / Divergence (MACD)

- **Purpose**:
  - Identifies the momentum of a trend by analyzing the relationship between two moving averages.
  - Helps detect potential trend reversals and continuation points.
  - Highlights bullish or bearish momentum shifts.
- **Parameters**:
  - **Fast EMA (12)**:
    - A shorter exponential moving average that reacts quickly to price changes.
  - **Slow EMA (26)**:
    - A longer exponential moving average that smooths out price movements over a greater period.
  - **Signal Line EMA (9)**:
    - An EMA of the MACD line that acts as a trigger for buy or sell signals.
- **Usage**:
  - **MACD Line Crossover**:
    - When the MACD line crosses **above** the signal line, it generates a **bullish signal** (buy opportunity).
    - When the MACD line crosses **below** the signal line, it generates a **bearish signal** (sell opportunity).
  - **Histogram Analysis**:

        - Positive histogram values indicate bullish momentum.
        - Negative histogram values indicate bearish momentum.
        - Shrinking histogram bars may signal a weakening trend.
    - **Divergence**:
        - If the MACD line diverges from the price action, it may indicate a potential reversal.
            - **Bullish Divergence**: MACD moves higher while the price moves lower.
            - **Bearish Divergence**: MACD moves lower while the price moves higher.

---

4. Price Ratio

- **Purpose**:
    - Detects relative changes in short-term and long-term price averages.
    - Helps identify uptrends or downtrends by comparing averages.
- **Parameters**:
    - **Uptrend Threshold (1.01)**:
        - Indicates significant upward movements when the short-term average exceeds the long-term average by 1%.
    - **Downtrend Threshold (0.99)**:
        - Indicates significant downward movements when the short-term average is 1% below the long-term average.
- **Usage**:
    - Buy when the ratio is greater than the uptrend threshold.
    - Sell when the ratio is below the downtrend threshold.

---

5. Z-Score (Mean-Reversion)

- **Purpose**:
    - Measures how far a price deviates from its rolling mean, expressed in terms of standard deviations.
    - Detects overbought or oversold conditions for mean-reversion strategies.
- **Parameters**:
    - **Rolling Window (20)**:
        - Defines the number of periods used to calculate the rolling mean and standard deviation.
    - **Entry Threshold (±2)**:
        - Buy when the price is two standard deviations below the mean.
        - Sell when the price is two standard deviations above the mean.
    - **Exit Threshold (±1)**:
        - Exit positions when the price returns to within one standard deviation of the mean.
- **Usage**:
    - Suitable for range-bound markets, where prices oscillate around a mean value.

---

6. Ornstein-Uhlenbeck Process (OU)

- **Purpose**:
    - Simulates mean-reverting behavior of prices.
    - Models the stochastic dynamics of prices using a combination of mean reversion and random noise.
- **Parameters**:
    - **Long-Term Mean (μ)**:
        - Represents the equilibrium price to which the process reverts.
    - **Mean Reversion Rate ($\theta = 0.5$)**:
        - Governs the speed of reversion toward the long-term mean.
        - A higher value results in faster reversion, while a lower value leads to slower reversion.
    - **Volatility ($\sigma = 1$)**:
        - Controls the magnitude of random fluctuations around the mean.
    - **Initial Price (P_0)**:
        - Sets the starting price for the simulation.
    - **Time Steps ($T = 500$)**:
        - Determines the granularity of the simulated process.
- **Usage**:
    - Useful for modeling range-bound assets and validating the effectiveness of mean-reversion strategies.

---

7. Kalman Filter

- **Purpose**:
    - Dynamically estimates the mean and variance of the price series in real-time, allowing for more adaptive and precise decision-making compared to static approaches like Z-Score or OU processes.
- **Parameters**:
    - **Initial State Mean**:
        - Sets the starting state estimate using the first price in the dataset.

    - **Confidence Interval (±2 Variances)**:
        - Defines the uncertainty bounds for mean-reverting signals, equivalent to entry/exit thresholds in a standard mean-reversion strategy.
    - **Dynamic Mean Estimate**:
        - Continuously updates the estimated state mean based on the observed data, adapting to market changes in real-time.
- **Usage**:
    - Buy when the observed price is below the lower confidence interval.
    - Sell when the observed price is above the upper confidence interval.
    - Exit when the observed price is within the confidence interval.

Why Kalman Filter Outperforms

- **Adaptability**:
    - Unlike static mean-reversion models, the Kalman filter adjusts its state estimates dynamically, enabling better performance in volatile markets.
- **Precision**:
    - The filter's confidence intervals reduce false signals and enhance entry/exit decisions, leading to improved returns and stability.
- **Noise Reduction**:
    - Effectively filters out noise, allowing for cleaner signals compared to traditional Z-Score and OU-based strategies.

Discussion: Impact of Market Regime on Strategy Performance

1. Upward Trend with Jumps and High Volatility

- **Description**:
    - This market regime features a clear upward trend but with sharp and unpredictable price movements (jumps).
    - High volatility amplifies both the potential for profits and the risk of drawdowns.

---

2. Trend-Following Strategies

- **Performance**:
    - **Better Returns**: Trend-following strategies typically perform well in upward-trending markets as long as the trend remains intact.
    - **Challenges**:
        - Frequent jumps may lead to false signals, causing premature entries or exits.
        - High volatility can result in increased drawdowns, especially if stop-loss levels are triggered frequently.
- **Indicator Sensitivity**:
    - Smoother indicators (e.g., EMA) may help filter noise caused by jumps.
    - ADX could misinterpret high volatility as a strong trend, potentially generating misleading signals.

---

3. Mean-Reversion Strategies

- **Performance**:
    - **Worse Returns**: Mean-reversion strategies may struggle in upward-trending markets because prices consistently deviate from the mean without reverting.
    - **Challenges**:
        - Jumps exacerbate losses by moving the price further from the mean.
        - High volatility increases the likelihood of stop-loss triggers.
- **Indicator Sensitivity**:
    - Z-score or Ornstein-Uhlenbeck-based signals may become unreliable as the mean shifts upward due to the trend.
    - Kalman filters may adapt better by dynamically updating the state mean.

Conclusion

For a market regime with upward trends and high volatility:

- **Trend-Following Strategies**:
    - Likely to perform better, provided they incorporate noise-reducing indicators and robust risk controls.
- **Mean-Reversion Strategies**:
    - Face significant challenges and require dynamic adaptation, such as Kalman filters, to remain effective.

This analysis highlights the importance of aligning strategy design with market conditions to maximize returns while minimizing risk.

Part II, III Implementation

Conceptual Trading Loop

Trading Bot Summary

This code outlines a trading bot for stock market operations using the Alpaca API. The bot fetches historical stock prices, analyzes the data with a Kalman filter to generate trading signals, and executes trades based on those signals.

Key Components

1. Logging Configuration

- Configures structured logging to track bot activities and errors.

2. Global Variables

- `cumulative_pnl`: Tracks cumulative profit and loss (PnL).
- `order_check_count`: Counts the number of orders placed or could have been placed since we want to stop at some point.

3. Market Data Validation

- Ensures the fetched stock price data is valid:
  - Checks for null or non-positive values.
  - Monitors for abrupt price changes using a threshold.

4. Fetch Historical Prices

- Uses Alpaca's API to fetch stock prices based on:
  - Symbol, date range, and timeframe.
- Validates data before proceeding.

5. Generate Trading Signals

- Applies Z-score thresholds to generate buy/sell signals:
  - **Buy Signal**: Z-score < -threshold.
  - **Sell Signal**: Z-score > threshold.

6. Execute Trades

- Executes trades (buy/sell) based on signals.
- Tracks trade details and calculates PnL for each executed trade.

7. Track Orders

- Monitors the status of individual orders (e.g., `filled`, `pending`, etc.).

8. Track Positions

- Retrieves the current open position quantity for a specific stock.

9. Perform Pre-Trading Checks

- Verifies:
  - If the stock is tradable.
  - Whether the market is open for trading.

Trading Workflow

1. **Setup**

- Configures logging.
- Initializes global variables.
- Performs pre-trading checks for tradability and market status.

2. **Data Collection**

- Fetches and validates historical stock prices.

3. **Analysis**

- Applies the Kalman filter to generate refined price data.
- Computes Z-scores to identify trading opportunities.

4. **Execution**

- Iterates through trading signals:
    - Tracks positions.
    - Executes trades based on signals.
    - Updates PnL for each trade.
- Simulates a delay between trades.

5. **Completion**

- Logs the cumulative PnL at the end of the trading session.

Additional Features

Kalman Filter

- Processes price data to estimate:
    - **State Mean**: Expected price.
    - **State Variance**: Price uncertainty.

Order Execution

- Places orders via Alpaca's API.
- Tracks execution status and logs outcomes.

Error Handling

- Validates market data before processing.
- Logs errors and warnings for issues like:
    - Invalid data.
    - Failed API calls.
    - Order execution errors.
- Gracefully exits when critical errors occur.

This bot is designed to execute automated trading strategies while ensuring data validation, error handling, and position tracking.

In [16]:
```python
# Logging configuration
logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")

# Track cumulative PnL
cumulative_pnl = 0

# Counter for the number of orders
order_check_count = 0

# Function to validate market data
def validate_market_data(prices):
    """
    Validate the market data to ensure it is usable.

    Args:
        prices (pd.Series): Series of closing prices.

    Returns:
        bool: True if the data is valid, False otherwise.
    """
    if prices.isnull().any():
        logging.error("Market data contains null values.")
        return False
    if (prices <= 0).any():
        logging.error("Market data contains non-positive prices.")
        return False
    if prices.pct_change().abs().max() > 0.5:  # Example threshold for abrupt changes
        logging.warning("Market data shows abrupt price changes.")
    return True

# Function to fetch historical price data
def fetch_historical_prices(symbol, start_date, end_date, timeframe):
    """
    Fetch historical price data using Alpaca API.
    Args:
        symbol (str): The stock symbol (e.g., 'NVDA').
        start_date (datetime): Start date for historical data.
        end_date (datetime): End date for historical data.
        timeframe (TimeFrame): Timeframe for the price bars.

    Returns:
        pd.Series: Series of closing prices.
    """
    try:
        bars_request = StockBarsRequest(
            symbol_or_symbols=symbol,
            timeframe=timeframe,
            start=start_date,
            end=end_date
        )
        bars = data_client.get_stock_bars(bars_request).df
        prices = bars["close"].xs(symbol)  # Extract closing prices

        # Validate market data
        if not validate_market_data(prices):
            return pd.Series(dtype=float)

        return prices
    except Exception as e:
        logging.error(f"Error fetching historical prices for {symbol}: {e}")
        return pd.Series(dtype=float)

# Function to make buy/sell decisions (z set to a lower value to force BUY / SELL simulation, in practice market will not vary
def generate_trading_signals(filtered_data, z_threshold=0.1):
    """
    Generate trading signals based on deviations from the mean.

    Args:
        filtered_data (pd.DataFrame): DataFrame with Kalman filter estimates.
        z_threshold (float): Threshold for Z-score to trigger buy/sell signals.

    Returns:
        pd.Series: Series of trading signals (-1 for sell, 1 for buy, 0 for hold).
    """
    # Calculate Z-score
    filtered_data["z_score"] = (filtered_data["price"] - filtered_data["state_mean"]) / filtered_data["state_variance"]

    # Generate signals
    filtered_data["signal"] = 0
    filtered_data.loc[filtered_data["z_score"] > z_threshold, "signal"] = -1  # Sell
    filtered_data.loc[filtered_data["z_score"] < -z_threshold, "signal"] = 1  # Buy

    return filtered_data

# Function to execute trades and calculate PnL
def execute_trades(symbol, signal, position_qty, price):
    """
    Execute trades based on the trading signal and calculate PnL.

    Args:
        symbol (str): The stock symbol.
        signal (int): Trading signal (-1 for sell, 1 for buy, 0 for hold).
        position_qty (float): Current quantity of the position.
        price (float): Current price of the stock.

    Returns:
        float: Profit or loss from the trade.
    """
    global cumulative_pnl, order_check_count

    if signal == 1 and position_qty == 0:
        # Buy
        logging.info(f"Signal: BUY for {symbol} at price {price}")
        place_order(symbol, qty=1, side=OrderSide.BUY)
        order_check_count += 1
        return 0.0
    elif signal == -1 and position_qty > 0:
        # Sell
        logging.info(f"Signal: SELL for {symbol} at price {price}")
        place_order(symbol, qty=position_qty, side=OrderSide.SELL)
        order_check_count += 1

        # Calculate PnL for the trade
        entry_price = position_qty * price  # Approximation for now using unit cost
        pnl = price - entry_price
        cumulative_pnl += pnl
        logging.info(f"Trade PnL: ${pnl:.2f}")
        return pnl
    else:
        logging.info(f"No trade executed. Trial: {order_check_count}")
```

```python
        order_check_count += 1
        return 0.0

# Track order status
def track_order(order_id):
    """
    Track the status of a specific order by its ID.

    Args:
        order_id (str): The ID of the order to track.

    Returns:
        str: The status of the order (e.g., 'filled', 'cancelled', etc.).
    """
    try:
        orders = trading_client.get_orders()  # Fetch all recent orders
        order = next((o for o in orders if o.id == order_id), None)
        if order:
            logging.info(f"Order status: {order.status}")
            return order.status
        else:
            logging.error(f"Order with ID {order_id} not found.")
            return None
    except Exception as e:
        logging.error(f"Error tracking order {order_id}: {e}")
        return None

# Place a market order and track its status
def place_order(symbol, qty, side):
    try:
        order_request = MarketOrderRequest(
            symbol=symbol,
            qty=qty,
            side=side,
            time_in_force=TimeInForce.GTC,
        )
        order = trading_client.submit_order(order_request)
        logging.info(f"Order placed: {side} {qty} shares of {symbol}")

        # Track order status
        order_status = track_order(order.id)
        if order_status == "filled":
            logging.info("Order successfully filled.")
        else:
            logging.warning(f"Order not filled. Current status: {order_status}")
        return order
    except Exception as e:
        logging.error(f"Error placing order for {symbol}: {e}")
        return None

# Track current positions
def track_positions(symbol):
    """
    Track current open positions for the given symbol.

    Args:
        symbol (str): The stock symbol.

    Returns:
        float: Current position quantity.
    """
    try:
        positions = trading_client.get_all_positions()
        for position in positions:
            if position.symbol == symbol:
                return float(position.qty)
        return 0.0
    except Exception as e:
        logging.error(f"Error tracking positions for {symbol}: {e}")
        return None

# Checks
def perform_checks():
    # Check if the stock is tradable
    assets = trading_client.get_all_assets(GetAssetsRequest(
        status=AssetStatus.ACTIVE,
        asset_class=AssetClass.US_EQUITY,
    ))
    nvda_asset = next((a for a in assets if a.symbol == "NVDA"), None)
    if nvda_asset and nvda_asset.tradable:
        logging.info("NVDA is tradable.")
    else:
        logging.error("NVDA is not tradable. Exiting.")
        return False

    # Check market status
    clock = trading_client.get_clock()
    if not clock.is_open:
        logging.warning("Market is currently closed. Orders may not fill.")

    return True

# Main function to run the trading loop
def main():
    SYMBOL = "NVDA"
    START_DATE = datetime(2024, 11, 1)
    END_DATE = datetime(2024, 11, 20)
    TIMEFRAME = TimeFrame.Hour  # Example timeframe

    if not perform_checks():
        return

    # Fetch historical prices
    prices = fetch_historical_prices(SYMBOL, START_DATE, END_DATE, TIMEFRAME)
    if prices.empty:
        logging.error("No valid market data fetched. Exiting.")
        return

    # Apply Kalman filter
    filtered_data = apply_kalman_filter(prices)

    # Generate trading signals
    filtered_data = generate_trading_signals(filtered_data)

    # Trading loop
    for index, row in filtered_data.iterrows():
        # Track current position
        position_qty = track_positions(SYMBOL)
        if position_qty is None:
            logging.error("Failed to track positions. Exiting.")
            break

        # Execute trades based on signals
        pnl = execute_trades(SYMBOL, row["signal"], position_qty, row["price"])

        if order_check_count >= 100:
            logging.info("Maximum order limit (10) reached. Exiting trading loop.")
            return None  # Stop further order execution

        time.sleep(5)  # Simulate delay

    logging.info(f"Trading session ended. Cumulative PnL: ${cumulative_pnl:.2f}")

if __name__ == "__main__":
    main()
```

```
2024-11-26 22:32:03,215 - INFO - NVDA is tradable.
2024-11-26 22:32:06,465 - INFO - No trade executed. Trial: 0
2024-11-26 22:32:11,808 - INFO - No trade executed. Trial: 1
2024-11-26 22:32:17,055 - INFO - No trade executed. Trial: 2
2024-11-26 22:32:22,324 - INFO - No trade executed. Trial: 3
2024-11-26 22:32:27,599 - INFO - No trade executed. Trial: 4
2024-11-26 22:32:32,983 - INFO - Signal: SELL for NVDA at price 136.1622
2024-11-26 22:32:33,236 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:32:34,103 - ERROR - Order with ID 8b4e6807-daa8-4b74-ac5f-7fd465c054b2 not found.
2024-11-26 22:32:34,104 - WARNING - Order not filled. Current status: None
2024-11-26 22:32:34,105 - INFO - Trade PnL: $0.00
2024-11-26 22:32:39,369 - INFO - No trade executed. Trial: 6
2024-11-26 22:32:44,642 - INFO - No trade executed. Trial: 7
2024-11-26 22:32:49,957 - INFO - Signal: BUY for NVDA at price 135.7999
2024-11-26 22:32:50,436 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:32:50,790 - ERROR - Order with ID cf90ca76-655f-4d1c-9af2-10f6bc430ab6 not found.
2024-11-26 22:32:50,791 - WARNING - Order not filled. Current status: None
2024-11-26 22:32:56,061 - INFO - No trade executed. Trial: 9
2024-11-26 22:33:01,942 - INFO - No trade executed. Trial: 10
2024-11-26 22:33:07,190 - INFO - No trade executed. Trial: 11
2024-11-26 22:33:14,746 - INFO - No trade executed. Trial: 12
2024-11-26 22:33:23,622 - INFO - Signal: SELL for NVDA at price 138.92
2024-11-26 22:33:24,638 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:33:26,339 - ERROR - Order with ID bb302efc-9877-4f35-9d3b-bc5efec78e65 not found.
2024-11-26 22:33:26,340 - WARNING - Order not filled. Current status: None
2024-11-26 22:33:26,341 - INFO - Trade PnL: $0.00
2024-11-26 22:33:31,692 - INFO - No trade executed. Trial: 14
2024-11-26 22:33:37,085 - INFO - No trade executed. Trial: 15
2024-11-26 22:33:42,330 - INFO - Signal: BUY for NVDA at price 138.43
2024-11-26 22:33:42,586 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:33:42,835 - ERROR - Order with ID 776c24e6-fc5e-4f63-8134-56bef3b715db not found.
2024-11-26 22:33:42,836 - WARNING - Order not filled. Current status: None
2024-11-26 22:33:48,080 - INFO - No trade executed. Trial: 17
2024-11-26 22:33:53,326 - INFO - No trade executed. Trial: 18
2024-11-26 22:33:58,576 - INFO - No trade executed. Trial: 19
2024-11-26 22:34:03,834 - INFO - No trade executed. Trial: 20
2024-11-26 22:34:09,324 - INFO - No trade executed. Trial: 21
2024-11-26 22:34:14,574 - INFO - Signal: SELL for NVDA at price 137.72
2024-11-26 22:34:14,850 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:34:15,095 - INFO - Order status: OrderStatus.NEW
2024-11-26 22:34:15,097 - WARNING - Order not filled. Current status: OrderStatus.NEW
2024-11-26 22:34:15,098 - INFO - Trade PnL: $0.00
2024-11-26 22:34:20,345 - INFO - No trade executed. Trial: 23
2024-11-26 22:34:25,593 - INFO - No trade executed. Trial: 24
2024-11-26 22:34:30,850 - INFO - No trade executed. Trial: 25
2024-11-26 22:34:36,097 - INFO - Signal: BUY for NVDA at price 137.78
2024-11-26 22:34:36,352 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:34:36,622 - INFO - Order status: OrderStatus.NEW
2024-11-26 22:34:36,623 - WARNING - Order not filled. Current status: OrderStatus.NEW
2024-11-26 22:34:41,873 - INFO - No trade executed. Trial: 27
2024-11-26 22:34:47,139 - INFO - No trade executed. Trial: 28
2024-11-26 22:34:52,414 - INFO - No trade executed. Trial: 29
2024-11-26 22:34:57,664 - INFO - No trade executed. Trial: 30
2024-11-26 22:35:02,913 - INFO - No trade executed. Trial: 31
2024-11-26 22:35:08,159 - INFO - Signal: SELL for NVDA at price 137.1
2024-11-26 22:35:08,419 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:35:08,807 - INFO - Order status: OrderStatus.NEW
2024-11-26 22:35:08,807 - WARNING - Order not filled. Current status: OrderStatus.NEW
2024-11-26 22:35:08,808 - INFO - Trade PnL: $0.00
2024-11-26 22:35:16,105 - INFO - No trade executed. Trial: 33
2024-11-26 22:35:21,569 - INFO - No trade executed. Trial: 34
2024-11-26 22:35:26,866 - INFO - Signal: BUY for NVDA at price 136.94
2024-11-26 22:35:27,183 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:35:27,683 - ERROR - Order with ID b982e4ec-2fe9-4124-95fa-95f93fc32412 not found.
2024-11-26 22:35:27,684 - WARNING - Order not filled. Current status: None
2024-11-26 22:35:32,929 - INFO - No trade executed. Trial: 36
2024-11-26 22:35:38,176 - INFO - Signal: SELL for NVDA at price 138.77
2024-11-26 22:35:38,436 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:35:38,686 - INFO - Order status: OrderStatus.NEW
2024-11-26 22:35:38,687 - WARNING - Order not filled. Current status: OrderStatus.NEW
2024-11-26 22:35:38,688 - INFO - Trade PnL: $0.00
2024-11-26 22:35:45,205 - INFO - No trade executed. Trial: 38
2024-11-26 22:35:50,451 - INFO - No trade executed. Trial: 39
2024-11-26 22:35:55,722 - INFO - Signal: BUY for NVDA at price 139.665
2024-11-26 22:35:55,967 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:35:56,233 - INFO - Order status: OrderStatus.NEW
2024-11-26 22:35:56,234 - WARNING - Order not filled. Current status: OrderStatus.NEW
2024-11-26 22:36:01,482 - INFO - No trade executed. Trial: 41
2024-11-26 22:36:06,727 - INFO - No trade executed. Trial: 42
2024-11-26 22:36:12,043 - INFO - Signal: SELL for NVDA at price 139.91
2024-11-26 22:36:12,297 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:36:12,541 - ERROR - Order with ID 42b700db-3fd8-4c64-b75e-11245dead37f not found.
2024-11-26 22:36:12,542 - WARNING - Order not filled. Current status: None
2024-11-26 22:36:12,543 - INFO - Trade PnL: $0.00
2024-11-26 22:36:17,816 - INFO - No trade executed. Trial: 44
2024-11-26 22:36:23,062 - INFO - No trade executed. Trial: 45
2024-11-26 22:36:28,322 - INFO - No trade executed. Trial: 46
```

```
2024-11-26 22:36:33,565 - INFO - No trade executed. Trial: 47
2024-11-26 22:36:38,844 - INFO - No trade executed. Trial: 48
2024-11-26 22:36:44,098 - INFO - No trade executed. Trial: 49
2024-11-26 22:36:49,375 - INFO - Signal: BUY for NVDA at price 140.23
2024-11-26 22:36:49,639 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:36:49,905 - INFO - Order status: OrderStatus.NEW
2024-11-26 22:36:49,906 - WARNING - Order not filled. Current status: OrderStatus.NEW
2024-11-26 22:36:55,164 - INFO - Signal: SELL for NVDA at price 141.19
2024-11-26 22:36:55,926 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:36:56,175 - ERROR - Order with ID a1594d4b-a8c2-4f3d-a1a7-53b4bbfd0693 not found.
2024-11-26 22:36:56,176 - WARNING - Order not filled. Current status: None
2024-11-26 22:36:56,176 - INFO - Trade PnL: $0.00
2024-11-26 22:37:01,434 - INFO - No trade executed. Trial: 52
2024-11-26 22:37:07,308 - INFO - No trade executed. Trial: 53
2024-11-26 22:37:12,860 - INFO - No trade executed. Trial: 54
2024-11-26 22:37:18,969 - INFO - No trade executed. Trial: 55
2024-11-26 22:37:25,073 - INFO - No trade executed. Trial: 56
2024-11-26 22:37:31,482 - INFO - No trade executed. Trial: 57
2024-11-26 22:37:37,274 - INFO - No trade executed. Trial: 58
2024-11-26 22:37:42,573 - INFO - Signal: BUY for NVDA at price 145.62
2024-11-26 22:37:42,950 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:37:43,281 - ERROR - Order with ID f685180b-0ab3-4e18-831b-93b57ab4d419 not found.
2024-11-26 22:37:43,282 - WARNING - Order not filled. Current status: None
2024-11-26 22:37:48,812 - INFO - No trade executed. Trial: 60
2024-11-26 22:37:54,434 - INFO - No trade executed. Trial: 61
2024-11-26 22:37:59,898 - INFO - No trade executed. Trial: 62
2024-11-26 22:38:05,763 - INFO - No trade executed. Trial: 63
2024-11-26 22:38:11,767 - INFO - Signal: SELL for NVDA at price 145.63
2024-11-26 22:38:12,618 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:38:18,083 - ERROR - Order with ID 2ec973f1-fe28-4abb-b0ee-bbe469559608 not found.
2024-11-26 22:38:18,084 - WARNING - Order not filled. Current status: None
2024-11-26 22:38:18,085 - INFO - Trade PnL: $0.00
2024-11-26 22:38:23,329 - INFO - Signal: BUY for NVDA at price 144.98
2024-11-26 22:38:23,573 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:38:23,818 - INFO - Order status: OrderStatus.NEW
2024-11-26 22:38:23,819 - WARNING - Order not filled. Current status: OrderStatus.NEW
2024-11-26 22:38:29,084 - INFO - Signal: SELL for NVDA at price 145.31
2024-11-26 22:38:29,331 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:38:29,577 - ERROR - Order with ID 6641d212-5341-42f7-bfb2-8e492b5be595 not found.
2024-11-26 22:38:29,578 - WARNING - Order not filled. Current status: None
2024-11-26 22:38:29,579 - INFO - Trade PnL: $0.00
2024-11-26 22:38:34,822 - INFO - No trade executed. Trial: 67
2024-11-26 22:38:40,070 - INFO - No trade executed. Trial: 68
2024-11-26 22:38:45,313 - INFO - No trade executed. Trial: 69
2024-11-26 22:38:50,624 - INFO - No trade executed. Trial: 70
2024-11-26 22:38:55,891 - INFO - No trade executed. Trial: 71
2024-11-26 22:39:01,141 - INFO - No trade executed. Trial: 72
2024-11-26 22:39:06,398 - INFO - Signal: BUY for NVDA at price 148.02
2024-11-26 22:39:06,652 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:39:06,900 - ERROR - Order with ID 31f0cd6a-dc5c-4eb6-aef9-b86060d7e91a not found.
2024-11-26 22:39:06,901 - WARNING - Order not filled. Current status: None
2024-11-26 22:39:12,148 - INFO - Signal: SELL for NVDA at price 148.4646
2024-11-26 22:39:12,412 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:39:12,693 - ERROR - Order with ID 72bc62c9-8115-416d-93ce-48ad48fc7753 not found.
2024-11-26 22:39:12,694 - WARNING - Order not filled. Current status: None
2024-11-26 22:39:12,695 - INFO - Trade PnL: $0.00
2024-11-26 22:39:17,942 - INFO - No trade executed. Trial: 75
2024-11-26 22:39:23,186 - INFO - No trade executed. Trial: 76
2024-11-26 22:39:28,483 - INFO - No trade executed. Trial: 77
2024-11-26 22:39:33,729 - INFO - No trade executed. Trial: 78
2024-11-26 22:39:38,977 - INFO - No trade executed. Trial: 79
2024-11-26 22:39:44,260 - INFO - No trade executed. Trial: 80
2024-11-26 22:39:49,507 - INFO - Signal: BUY for NVDA at price 147.27
2024-11-26 22:39:49,775 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:39:50,034 - INFO - Order status: OrderStatus.NEW
2024-11-26 22:39:50,035 - WARNING - Order not filled. Current status: OrderStatus.NEW
2024-11-26 22:39:55,283 - INFO - Signal: SELL for NVDA at price 148.14
2024-11-26 22:39:55,575 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:39:55,821 - ERROR - Order with ID f01e8a41-e8d8-4932-a11e-4ea7b0f8fbdd not found.
2024-11-26 22:39:55,821 - WARNING - Order not filled. Current status: None
2024-11-26 22:39:55,823 - INFO - Trade PnL: $0.00
2024-11-26 22:40:01,079 - INFO - No trade executed. Trial: 83
2024-11-26 22:40:06,321 - INFO - Signal: BUY for NVDA at price 148.5001
2024-11-26 22:40:06,568 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:40:06,864 - INFO - Order status: OrderStatus.NEW
2024-11-26 22:40:06,865 - WARNING - Order not filled. Current status: OrderStatus.NEW
2024-11-26 22:40:12,131 - INFO - No trade executed. Trial: 85
2024-11-26 22:40:17,392 - INFO - No trade executed. Trial: 86
2024-11-26 22:40:22,637 - INFO - No trade executed. Trial: 87
2024-11-26 22:40:27,890 - INFO - No trade executed. Trial: 88
2024-11-26 22:40:33,135 - INFO - No trade executed. Trial: 89
2024-11-26 22:40:38,380 - INFO - No trade executed. Trial: 90
2024-11-26 22:40:43,624 - INFO - Signal: SELL for NVDA at price 147.5
2024-11-26 22:40:43,873 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:40:44,116 - INFO - Order status: OrderStatus.NEW
2024-11-26 22:40:44,117 - WARNING - Order not filled. Current status: OrderStatus.NEW
2024-11-26 22:40:44,117 - INFO - Trade PnL: $0.00
```

```
2024-11-26 22:40:49,368 - INFO - No trade executed. Trial: 92
2024-11-26 22:40:54,612 - INFO - No trade executed. Trial: 93
2024-11-26 22:40:59,873 - INFO - No trade executed. Trial: 94
2024-11-26 22:41:05,119 - INFO - No trade executed. Trial: 95
2024-11-26 22:41:10,365 - INFO - Signal: BUY for NVDA at price 147.12
2024-11-26 22:41:10,607 - INFO - Order placed: OrderSide.BUY 1 shares of NVDA
2024-11-26 22:41:10,861 - ERROR - Order with ID 1a0bf8ba-bc06-4589-be8f-295cef847685 not found.
2024-11-26 22:41:10,861 - WARNING - Order not filled. Current status: None
2024-11-26 22:41:16,115 - INFO - No trade executed. Trial: 97
2024-11-26 22:41:21,366 - INFO - Signal: SELL for NVDA at price 147.34
2024-11-26 22:41:21,965 - INFO - Order placed: OrderSide.SELL 1.0 shares of NVDA
2024-11-26 22:41:22,230 - ERROR - Order with ID e2720d32-a7db-4c93-8bb4-8f5c9dedc226 not found.
2024-11-26 22:41:22,231 - WARNING - Order not filled. Current status: None
2024-11-26 22:41:22,231 - INFO - Trade PnL: $0.00
2024-11-26 22:41:27,479 - INFO - No trade executed. Trial: 99
2024-11-26 22:41:27,480 - INFO - Maximum order limit (10) reached. Exiting trading loop.
```

Performance and Risk Reporting

Adapting Risk Measures to Trading Systems to Conserve Capital

This code demonstrates how to integrate **risk management** and **performance evaluation** into a trading system to ensure capital preservation. It focuses on calculating key risk measures like Value at Risk (VaR) and performance metrics to assess the viability of trading strategies.

Key Features

1. **Rolling Value at Risk (VaR)**

- **Purpose**: Quantifies the potential loss at a specified confidence level over a rolling window.
- **Calculation**:
  - Rolling VaR is computed for the strategy's daily returns using a specified confidence level (default: 95%).
  - Formula: $\text{VaR} = -\text{Percentile}(X, 100 \times (1 - \text{Confidence Level}))$.

2. **Performance Metrics**

- **Purpose**: Measures the effectiveness and risk-adjusted performance of the trading strategy.
- **Metrics**:
  - **Sharpe Ratio**: Measures risk-adjusted returns.
    - $\text{Sharpe Ratio} = \frac{\text{Mean Returns}}{\text{Standard Deviation}} \times \sqrt{252}$
  - **Profit Factor**: Ratio of total profits to total losses.
    - $\text{Profit Factor} = \frac{\text{Total Profits}}{\text{Total Losses}}$
  - **Max Drawdown**: Largest peak-to-trough decline in cumulative returns.
    - $\text{Max Drawdown} = \min(\text{Cumulative Returns}/\text{Cumulative Max} - 1)$
  - **Turnover Costs**: Measures trading frequency and associated costs.
    - $\text{Turnover Costs} = \sum |\text{Position Changes}|$.

3. **Risk and Performance Evaluation**

- Integrates:
  - Rolling VaR for ongoing risk measurement.
  - Performance metrics to assess trading effectiveness.
- Returns both **VaR** and a dictionary of performance metrics.

Example Integration

1. **Simulated Backtesting Data**:

   - **Strategy_Returns**: Randomized daily returns to simulate strategy performance.
   - **Cumulative_Returns**: Simulated cumulative returns.
   - **Position**: Randomized position changes (-1, 0, 1) to reflect trading activity.
2. **Workflow**:

   - Rolling VaR is calculated for risk assessment.
   - Performance metrics are computed to evaluate strategy effectiveness.
   - Results are logged and displayed for analysis.

Results

- **Rolling VaR**: Shows the risk of potential losses over time.
- **Performance Metrics**:
  - Sharpe Ratio: Indicates risk-adjusted returns.
  - Profit Factor: Measures profitability relative to losses.
  - Max Drawdown: Highlights the largest potential loss.
  - Turnover Costs: Reflects the cost of frequent trading.

In [18]:

```python
# Rolling Value at Risk (VaR)
def calculate_var(data, confidence_level=0.95, window=20):
    """
    Calculate rolling Value at Risk (VaR).
    """
    return data["Strategy_Returns"].rolling(window).apply(
        lambda x: -np.percentile(x, 100 * (1 - confidence_level)), raw=True
    )

# Performance Metrics
def calculate_performance_metrics(data):
    """
    Calculate key performance metrics for the strategy.
    """
    returns = data["Strategy_Returns"]
    cumulative_returns = data["Cumulative_Returns"]

    # Sharpe Ratio
    sharpe_ratio = returns.mean() / returns.std() * np.sqrt(252)

    # Profit Factor
    total_profit = returns[returns > 0].sum()
    total_loss = abs(returns[returns < 0].sum())
    profit_factor = total_profit / total_loss if total_loss != 0 else np.nan

    # Max Drawdown
    cumulative_max = cumulative_returns.cummax()
    drawdowns = cumulative_returns / cumulative_max - 1
    max_drawdown = drawdowns.min()

    # Turnover Costs
    turnover = data["Position"].diff().abs().sum()

    return {
        "Sharpe Ratio": sharpe_ratio,
        "Profit Factor": profit_factor,
        "Max Drawdown": max_drawdown,
        "Turnover Costs": turnover,
    }

# Integrate evaluation
def risk_and_performance_evaluation(data):
    """
    Perform risk and performance evaluation on the strategy.
    """
    # Calculate Rolling VaR
    data["VaR"] = calculate_var(data)

    # Calculate Performance Metrics
    performance_metrics = calculate_performance_metrics(data)

    return {
        "Rolling VaR": data["VaR"],
        "Performance Metrics": performance_metrics,
    }

# Example Integration
def main():
    # Example backtesting data
    data = pd.DataFrame({
        "Strategy_Returns": np.random.normal(0, 0.01, 252),  # Simulated daily returns
        "Cumulative_Returns": (1 + np.random.normal(0, 0.01, 252)).cumprod(),
        "Position": np.random.choice([-1, 0, 1], 252)  # Simulated position changes
    })

    # Risk and Performance Evaluation
    results = risk_and_performance_evaluation(data)

    # Print Performance Metrics
    print("Performance Metrics:")
    for key, value in results["Performance Metrics"].items():
        print(f"{key}: {value:.4f}")

if __name__ == "__main__":
    main()
```

```
Performance Metrics:
Sharpe Ratio: 0.4300
Profit Factor: 1.0716
Max Drawdown: -0.2621
Turnover Costs: 209.0000
```

Pros and Cons of the Strategies

1. **Trend-Following**

- **Pros**:
  - Captures prolonged market trends effectively.
  - EMA, MACD and ADX provide robust signal generation in trending markets.
- **Cons**:
  - Generates false signals in sideways markets.
  - Suffers during periods of rapid trend reversals.

2. **Mean-Reversion**

- **Pros**:
  - Effective in identifying price overextensions in range-bound markets.
  - Z-scores and OU processes are mathematically robust.
- **Cons**:
  - Struggles in trending or volatile markets.
  - Requires precise calibration of reversion thresholds.

Suggested Improvements

1. **Dynamic Strategy Switching**:

   - Incorporate market condition detectors (e.g., trend/volatility classifiers) to switch between trend-following and mean-reversion strategies dynamically.
2. **Parameter Optimization**:

   - Use Bayesian optimization or genetic algorithms to fine-tune thresholds and indicator parameters.
3. **Hybrid Approaches**:

   - Combine mean-reversion with trend-following (e.g., entering a trend-following strategy post mean-reversion signal).
4. **Risk Management Enhancements**:

   - Include trailing stops and adaptive position sizing based on market volatility.
5. **Advanced Features**:

   - Integrate machine learning models for predictive insights and signal validation.