Barry Crosse CQF Final Project Algorithmic Trading for Reversion and Trend-Following

Introduction

Algorithmic trading has seen significant advancements with the integration of quantitative models to predict and respond to financial market movements. It is now more accessible than ever for traders to create fully automated algorithmic trading strategies without the need for substantial set up costs or IT expertise.

This project aims to explore two prominent types of trading strategies: mean reversion and trend following. These strategies represent contrasting approaches to market prediction. While mean reversion assumes that prices tend to revert to long-term averages after extreme movements, trend following strategies attempt to capitalise on the continuation of price momentum in a given direction, up or down.

This project focuses on applying these strategies to cryptocurrency markets, an asset class characterised by high volatility and liquidity, making it an ideal asset class for testing these strategies. The analysis involves the use of technical indicators such as exponential moving averages (EMAs), Bollinger Bands, Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), and Average Directional Index (ADX). These indicators are used to design, backtest and optimise the trading strategies using historical data.

This report considers the theoretical underpinnings of the strategies, the statistical methods used to test their feasibility (e.g. stationarity tests for mean reversion), and the practical implementation of these models in live trading systems. Through the use of systematic backtesting and optimisation, the report evaluates the efficacy of these strategies using key performance metrics such as returns, drawdowns, Sharpe ratio, Value at Risk and the return-to-drawdown ratio.

Data

The chosen broker, Alpaca, offers historical bar (OHLC) data through its API. This was used to get data in 5 minute intervals for the selected BCHUSD, ETHUSD and USDTUSD for the periods 01/07/2023 to 31/12/2023 and for the full 2024 year. The process of pulling and handling this data can be seen in P1AGetData.ipynb. The CSV files have been submitted also as BCHUSD_2023.csv, ETHUSD_2023.csv, USDTUSD_2023.csv BCHUSD_2024.csv, ETHUSD_2024.csv and USDTUSD_2024.csv

For the trades analysis completed in part three of the project, a proxy for the risk-free rate was needed. 3-month US Treasury Bills (t-bill) data was sourced from the Federal Reserve Bank of St. Louis (FRED) to use as a proxy for the risk-free rate. A 3-month t-bill is considered one of the safest investments as it is backed by the US government, making it essentially risk-free.

EURUSD prices were sourced from Yahoo Finance to graph some of the technical indicators. This data was not used in backtesting or strategy development.

Part 1: Generic Strategies Made Proprietary

Mean Reversion

Mean reversion is a theory that suggests asset prices tend towards a long term average. The greater the deviation from the mean, the higher the probability the asset's price will move toward the long term mean in the future. Mean reversion trading strategies try to capitalise on extreme price changes, assuming the price will return to the average. Technical analysis tools such as moving averages, relative strength index (RSI) and bollinger bands are commonly used in mean reversion strategies.

Stationarity

A time series is stationary if it has finite and constant mean, standard deviation and autocorrelation function. In general, stocks tend to increase in price, and are non-stationary. A stationary series does not go far from its mean in general.

Testing for the stationarity of a time series X_t requires a linear regression to find the coefficients $a,\ b\ and\ c$ in

$$X_t = aX_{t-1} + b + ct$$

If it is found that $|a| > 1$ then the series is unstable. If $-1 <= a < 1$ then the series is stationary. If $a = 1$ then the series is non-stationary. The value of a is accurate to a certain level of confidence. To decide if the series is stationary or non-stationary, the Dickey-Fuller statistic is used to estimate the degree of confidence in the result (Wilmott, 2006).

The Augmented Dickey Fuller Test is a unit root test, used for testing the stationarity of a time series. A unit root is a characteristic of a time series that makes it non-stationary. A unit root is said to exist in a time series if the value of $\alpha = 1$ in the below equation:

$$Y_t = \alpha Y_{t-1} + \beta X_e + \epsilon$$

where Y_t is the value of the series at time t, and X_e is another separate explanatory variable. The presence of a unit root means the time series is non-stationary.

Dickey-Fuller Test

The Dickey-Fuller test is a unit root test that tests the null hypothesis that $\alpha = 1$ in the following equation:

$$y_t = c + \beta t + \alpha y_{t-1} + \phi \Delta Y_{t-1} + e_t$$

Null hypothesis (H_0): $\alpha = 1$

where

- α is the coefficient of the first lag on Y
- y_{t-1} = lag 1 of the time series
- ΔY_{t-1} = the first difference of the series at time $t - 1$

The Dickey-Fuller test has a similar hypothesis as the unit root test, that the coefficient of Y_{t-1} is 1, implying the presence of a unit root. If not rejected, the series is taken to be non-stationary.

Augmented Dickey-Fuller Test (ADF)

The ADF is an “augmented” version of the Dickey-Fuller test. It expands the Dickey-Fuller test to include high order regressive process in the model:

$$y_t = c + \beta t + \alpha y_{t-1} + \phi_1 \Delta Y_{t-1} + \phi_2 \Delta Y_{t-2} + \ldots + \phi_p \Delta Y_{t-p} + e_t$$

There are more differencing terms added to the equation. The null hypothesis remains the same as the Dickey-Fuller test, that $\alpha = 1$, the series is non-stationary.

In order to say the series is stationary the p-value obtained should be less than the significance level (generally 0.05), in order to reject the null hypothesis.

Mean Reversion - Ornstein-Uhlenbeck Process (OU)

The Ornstein-Uhlenbeck process is a continuous-time stochastic process that models mean-reverting behaviour. Unlike a standard random walk / Brownian motion which can drift indefinitely, the OU process reverts to a long term mean over time. The OU process is a solution to the below stochastic differential equation that governs the mean-reverting behaviour:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t$$

where

- X_t represents the stochastic process at time t
- μ is the long-term mean
- θ is the rate of mean reversion
- σ is the volatility
- dW_t is a Wiener process / standard Brownian motion

The OU process is useful in quantitative finance for modelling assets that do not exhibit random walk behaviour, those that fluctuate around a long-term mean, such as interest rates and exchange rates.

Solving the Ornstein-Uhlenbeck SDE

To solve the OU SDE the integrating factor method can be used:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t$$

Multiply both sides by the integrating factor $e^{\theta t}$:

$$e^{\theta t}dX_t = \theta\mu e^{\theta t}dt - \theta X_t e^{\theta t}\ dt + \sigma e^{\theta t}dW_t$$

By adding $\theta X_t e^{\theta t}\ dt$ to both sides, the LHS can be rewritten as the differential of a product:

$$d(e^{\theta t}X_t) = \theta\mu e^{\theta t}dt + \sigma e^{\theta t}dW_t$$

Integrating both sides from 0 to t we get:

$$X_t = X_0 e^{-\theta t} + \mu(1 - e^{-\theta t}) + \sigma\int_0^t e^{-\theta(t-s)}dW_s$$

- The first term, $X_0 e^{-\theta t}$, represents the initial value decaying over time
- The second term, $\mu(1 - e^{-\theta t})$, shows that the prices reverse to its mean μ over time
- The third term, $\sigma\int_0^t e^{-\theta(t-s)}dW_s$, introduces stochasticity with the integral involving a Wiener process accounting for the random fluctuations.

The solution combines the mean-reverting behaviour and the stochastic component driven by Brownian motion.

Mean Reversion Data Testing

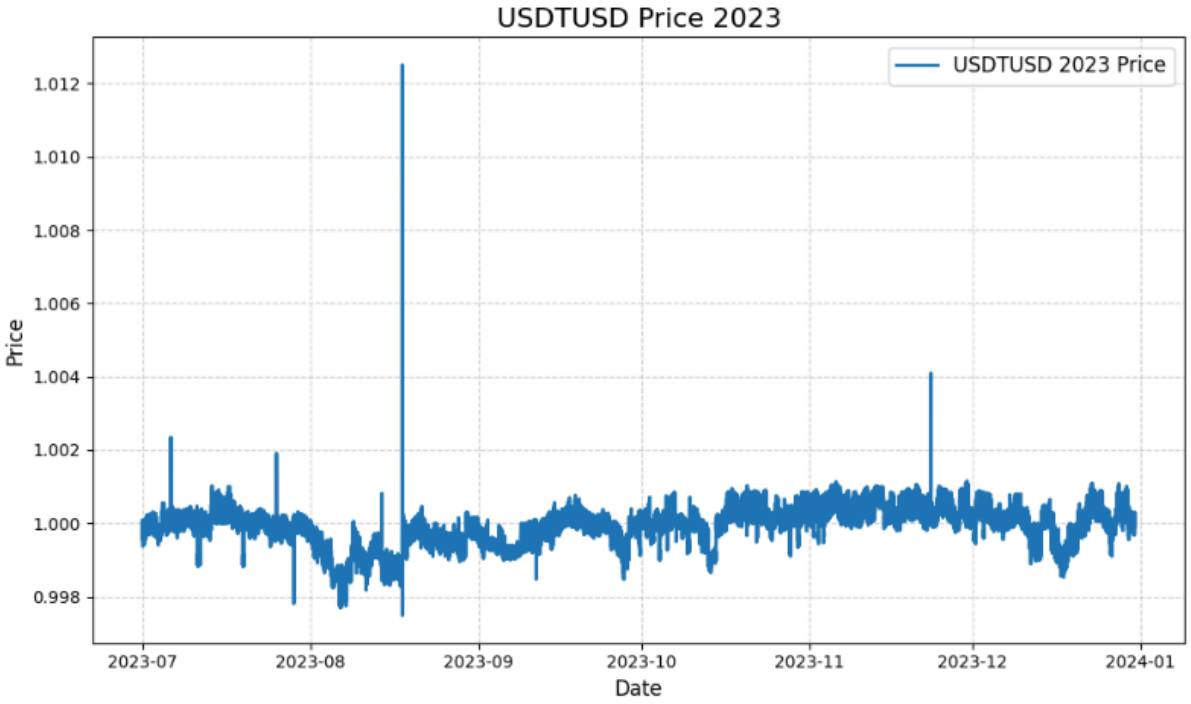
The BCHUSD, ETHUSD & USDTUSD prices can be tested for stationarity in python using the Augmented Dickey Fuller (ADF) Test. If a prices series is stationary it may be possible to use this in a mean reversion trading strategy. If prices stray far from the long term average, a trader can buy/sell the asset with some degree of confidence that the asset price will return towards the mean. As mentioned previously, the ADF Test is a unit root test, used for testing the stationarity of a time series. The most significant results came from testing USDTUSD prices:

USDTUSD

[7]:
```
prices = all_prices_df["USDTUSD"]
log_prices = np.log(prices)

# Check for stationarity using ADF test
adf_result = adfuller(log_prices)
print("Prices")
print("ADF Statistic:", adf_result[0])
print("p-value:", adf_result[1])
print("Num lags used:", adf_result[2])
print("Num obsvs used:", adf_result[3])
print("ADF Test Critical Values:", adf_result[4])
print("\n")

# check on returns on log prices
returns = log_prices.diff().dropna()
print("Returns")

# Check for stationarity using ADF test
adf_result = adfuller(returns)
print("ADF Statistic:", adf_result[0])
print("p-value:", adf_result[1])
print("Num lags used:", adf_result[2])
print("Num obsvs used:", adf_result[3])
print("ADF Test Critical Values:", adf_result[4])
```

```
Prices
ADF Statistic: -6.93225313933133
p-value: 1.0766239922195141e-09
Num lags used: 34
Num obsvs used: 52670
ADF Test Critical Values: {'1%': -3.4304741621175237, '5%': -2.861594877167301, '10%': -2.566799209290527}


Returns
ADF Statistic: -39.71881662976936
p-value: 0.0
Num lags used: 52
Num obsvs used: 52651
ADF Test Critical Values: {'1%': -3.4304742069256995, '5%': -2.8615948969712024, '10%': -2.566799219831557}
```

For USDTUSD prices, the ADF statistic is below the critical values and p-value is extremely low (1.08 * 10^-9), meaning we can reject the null hypothesis of non-stationarity at all critical level values (1%, 5% and 10%). Log prices often show trends and are typically non-stationary, as can be seen in the test results for ETHUSD log prices. However this is an expected result as USDTUSD (Tether) is a stablecoin that is pegged to the US dollar.



For USDTUSD returns, the extremely low ADF statistics and p-value confirm the prices are stationary.

Trend Following

Trend following is a style of trading that attempts to profit from an asset's momentum in an upward/downward direction. A trend following trader will enter into a long position when an asset is in an uptrend, and a short position when an asset is in a downward trend. This involves the assumption that the price will continue to move in the direction that it is currently heading. Stop-loss and take-profit provisions are commonly used in order to realise profits and avoid large losses if a trend reverses.

An uptrend can be defined as a series of higher swing highs and higher swing lows. Traders want to see the price rise above recent highs, but when the price falls it should stay above previous swing lows in order for the overall trajectory of the asset to be trending up. The opposite is true for downtrends, where they are defined as a series of lower swing highs and lower swing lows.

Technical Analysis Indicators

Simple Moving Average (SMA)

A SMA is a technical indicator commonly used in mean reversion and trend following trading strategies. The SMA calculates the average price of an asset during a specified period of time. Short-term SMAs react more quickly to changes in the price of an asset than a longer-term SMA.

The formula for calculating SMA is:

$$SMA = \frac{A_1 + A_2 + \ldots + A_n}{n}$$

where;

- A_n = the price of an asset at period n
- n = the total number of periods

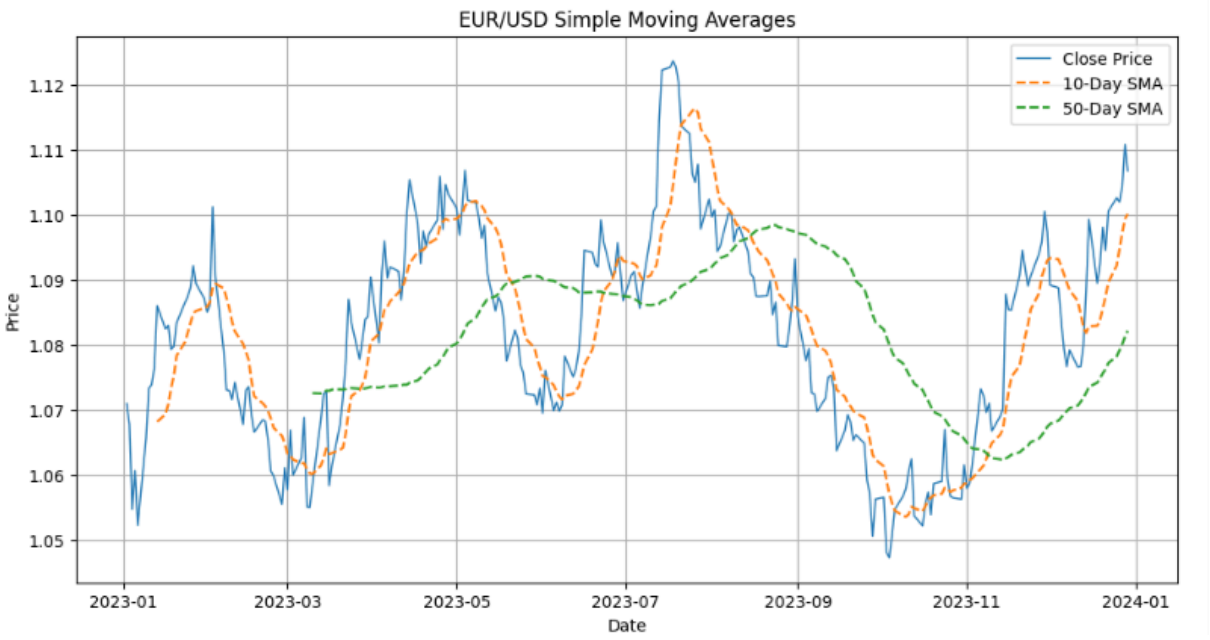
A SMA smooths the volatility of an asset, enabling a trader to more easily view the price trend of an asset. The longer the time frame of the SMA, the smoother it will be. A shorter-term SMA is more volatile, but it is reacting to more recent price changes.

The SMA gives equal weighting to all prices used, the 10th or 200th day price will have the same impact as the first or second day's price. Many traders believe that more emphasis should be placed on more recent data as it will better reflect current averages / trends.

Pandas has a built-in ".rolling()" method for calculating SMA but it can also be calculated as below:

[5]:
```python
def calculate_sma(prices, window):
    """
    Calculate Simple Moving Average (SMA).

    Parameters:
        prices (list or np.ndarray): List of prices.
        window (int): Moving average window size.

    Returns:
        list: SMA values (with None for indices where SMA cannot be calculated).
    """
    # List to store SMA values
    sma = []

    for i in range(len(prices)):
        # Ensure enough data to calc SMA
        if i >= window - 1:
            # Compute SMA as the mean of the last `window` prices
            sma.append(sum(prices[i - window + 1:i + 1]) / window)
        else:
            # Not enough data for SMA, append None
            sma.append(None)
    return sma
```

10 & 50 day SMAs can be seen on the plot below for EURUSD close prices for 2024:



The 10 day SMA is more sensitive to recent price changes than the 50 day SMA.

Exponential Moving Average (EMA)

An EMA is a moving average that places a greater weighting on more recent prices. The formula for an EMA is:

$$EMA_t = K * (Price_t) + (1 - K) * EMA_{t-1}$$

where;

- K is the weighting factor for the EMA. $K = \frac{2}{n+1}$, where n is the selected time period (window length)
- t is the current time period, $t - 1$ is the previous time period
- To initially calculate the EMA, the previous EMA value (EMA_{t-1}) is simply an average of all prices over the n number of periods (a SMA); $\frac{sum\ of\ 'n'\ prices}{'n'\ number\ of\ periods}$

The weighting factor is dependent on the length of the EMA specified. EMAs calculated over a shorter time period place a greater weighting on more recent observations, e.g.

Weighting factor for 21 Day EMA:

$$K = \frac{2}{21+1} = 0.09 = 9\%$$

vs

Weighting factor for 50 Day EMA:

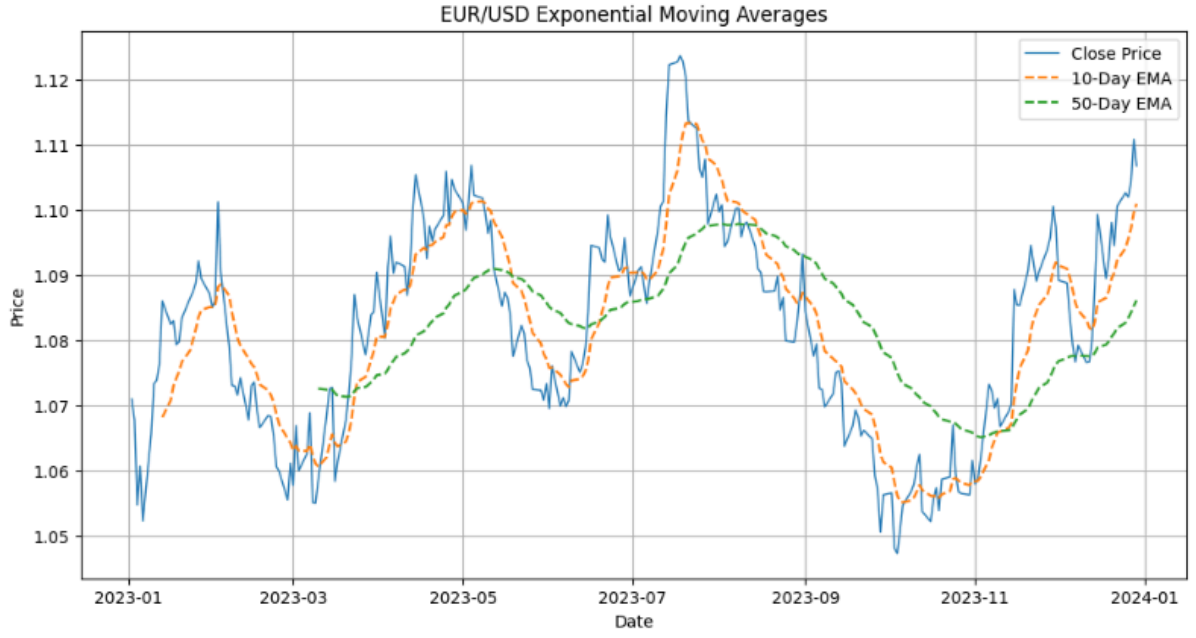
$$K = \frac{2}{50+1} = 0.039 = 3.9\%$$

The 21 day EMA places a 9% weighting on the most recent price, whereas the 50 day EMA places a 3.9% on the most recent price. Therefore EMAs calculated over shorter time periods are more sensitive to price changes.

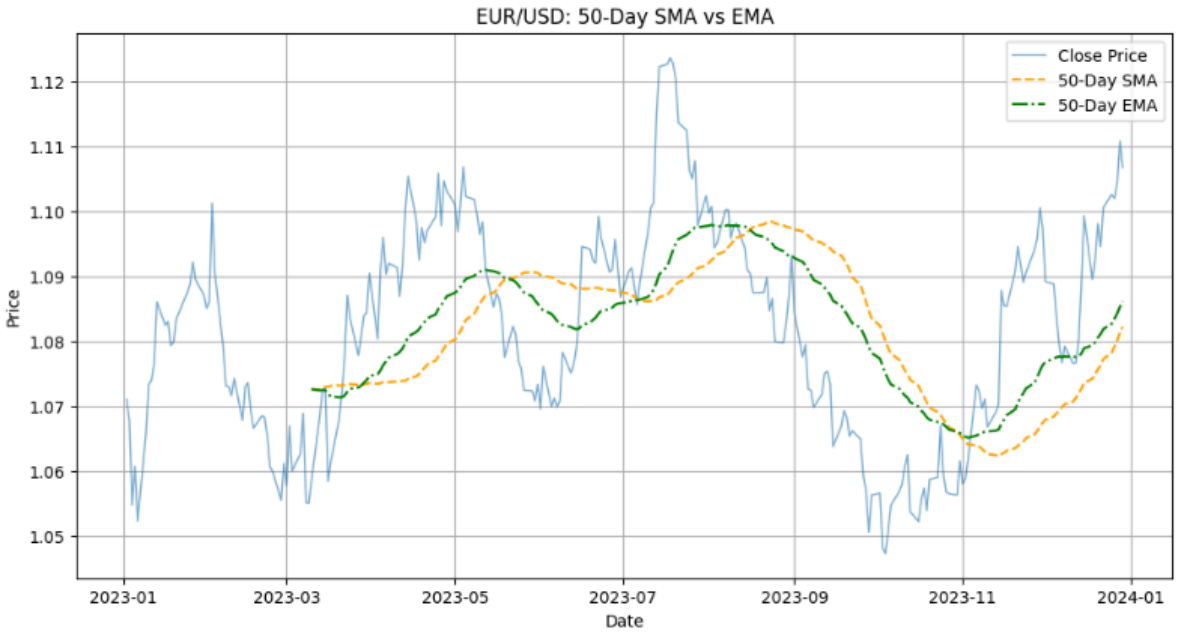
The greater the smoothing factor chosen, the greater the weight placed on the more recent prices on the EMA.

Similarly, Pandas also has a built-in ".ewm()" method to calculate EMA values. It can also be calculated as described above:

[7]:
```python
def calculate_ema(prices, window):
    """
    Calculate Exponential Moving Average (EMA).

    Parameters:
        prices (list or np.ndarray): List of prices.
        window (int): Moving average window size.

    Returns:
        list: EMA values.
    """
    # List to store EMA values
    ema = []
    # calc the smoothing factor
    alpha = 2 / (window + 1)

    # Calc the first EMA value as the SMA of the first `window` prices
    if len(prices) >= window:
        sma = sum(prices[:window]) / window  # Calculate SMA
        ema.append(sma)  # Use SMA as the first EMA value
    else:
        raise ValueError("Not enough data points to calc the initial SMA for EMA.")

    # Calc subsequent EMA values
    for i in range(window, len(prices)):
        ema.append(alpha * prices[i] + (1 - alpha) * ema[-1])

    # Add None for the first (window - 1) entries, as EMA is undefined for those
    ema = [None] * (window - 1) + ema

    return ema
```



The EMA gives a higher weighting to more recent price changes, whereas the SMA assigns an equal weighting to all values. Both of the moving averages are used by traders to smooth out price fluctuations. EMAs react more timely to new information, which is why they are more commonly used in algorithmic trading systems than SMAs.


EUR/USD: 50-Day SMA vs EMA
Close Price
50-Day SMA
50-Day EMA
Price
1.12
1.11
1.10
1.09
1.08
1.07
1.06
1.05
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01
Date

Bollinger Bands

Bollinger Bands are used to gauge the volatility of an asset and to determine if an asset is over/undervalued. Bollinger bands are shown on an asset price chart as three lines that move with the price. The centre line is typically the asset's 20 day SMA, and the upper/lower bands are generally two standard deviations above and below the middle line.

The bands widen when asset prices become more volatile and narrow when volatility decreases. The idea is that the asset becomes overbought as the price reaches the upper band, and oversold when the price approaches the lower band.

$$\begin{aligned} Middle\ Band &= SMA(P, n) \\ Upper\ Band &= Middle\ Band + k * SD(P, n) \\ Lower\ Band &= Middle\ Band - k * SD(P, n) \end{aligned}$$

where;

- P is the price
- n is the number of periods
- k is the number of standard deviations
- SD is the standard deviation

Standard Deviation can be calculated as follows:

$$SD_t = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(P_i - Mean_t)^2}$$

where;

- n is the rolling window size
- P_i is the price at time i in the window
- $Mean_t$ is the mean of the prices in the window

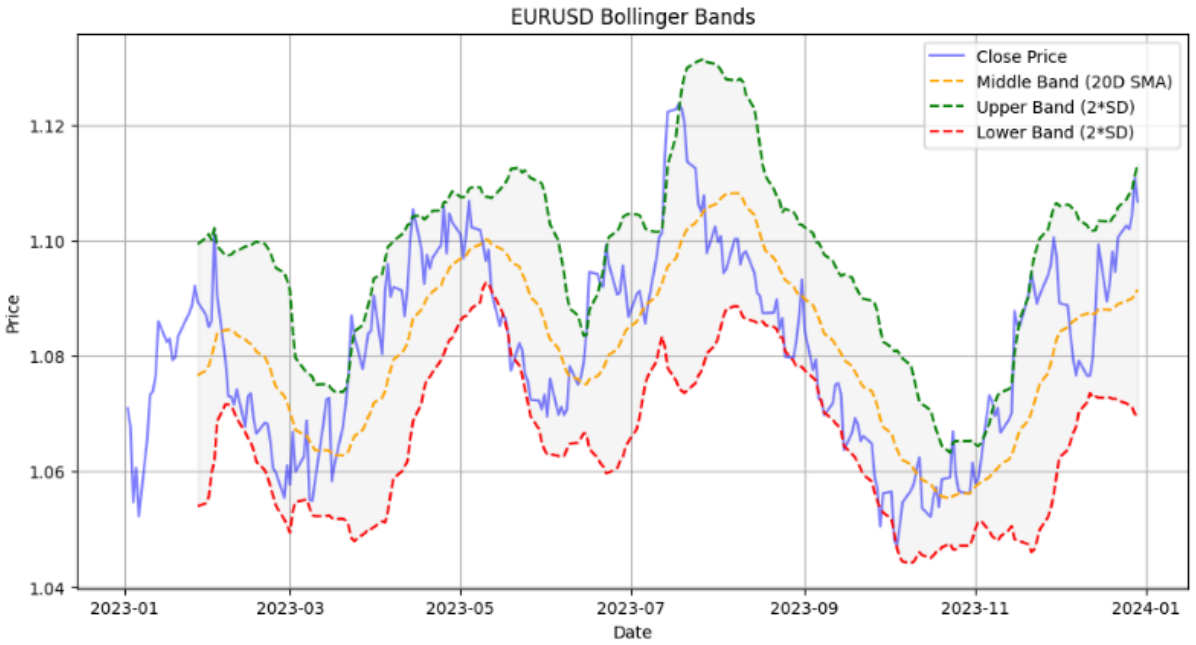
The use of two standard deviations to construct the upper and lower bands make use of the statistical property that creates a range in which 95% of prices are expected to be within. However this assumption relies on the dataset being normally distributed, but asset prices are known to have fat-tailed non-normal distributions.

Another function is needed to calculate the rolling standard deviation:

[9]:
```python
def calculate_rolling_std(prices, window):
    """
    Calculate rolling standard deviation.

    Parameters:
        prices (list or np.ndarray): List of prices.
        window (int): Rolling window size.

    Returns:
        list: Rolling standard deviation values (None for periods with insufficient data).
    """
    # List to store stand dev values
    std_values = []

    for i in range(len(prices)):
        if i >= window - 1:
            # Extract the rolling window of prices
            window_prices = prices[i - window + 1:i + 1]
            # Calculate the mean of the window
            mean = sum(window_prices) / window
            # Calculate the variance
            variance = sum((p - mean) ** 2 for p in window_prices) / window
            # Stand dev is the square root of variance
            std_values.append(variance ** 0.5)
        else:
            # Not enough data to calculate stand dev
            std_values.append(None)

    return std_values
```

This can then (along with the SMA function) to create upper and lower bollinger bands for an asset's price:

```python
# BB calcs
data["BB_SMA_20"] = calculate_sma(prices=data["Close"], window=20)
data["BB_STD"] = calculate_rolling_std(prices=data["Close"], window=20)
data["UPPER_BB"] = data["BB_SMA_20"] + 2 * data["BB_STD"]
data["LOWER_BB"] = data["BB_SMA_20"] - 2 * data["BB_STD"]
```



Relative Strength Index (RSI)

RSI is a momentum indicator used to measure the speed and magnitude of an asset's recent price changes to detect overbought/oversold conditions. The RSI is displayed as an oscillator on a scale of 0 to 100. Generally an RSI value of greater than 70 indicates an asset is overbought, and a value of less than 30 indicates it is oversold.

As a momentum indicator the RSI compares an asset's strength on days when prices go up to its strength on days when prices go down. RSI is calculated in two steps:

Step 1: $$RSI_{Step\ 1} = 100 - [\frac{100}{1 + \frac{Avg\ gain}{Avg\ loss}}]$$

The $Avg\ gain$ and $Avg\ loss$ is the average percentage gain/loss during a specified lookback period. Positive values are used for the average loss. Periods with price losses are counted as zero in the calculation for average gain, and periods with price gains are calculated as zero when calculating average loss.

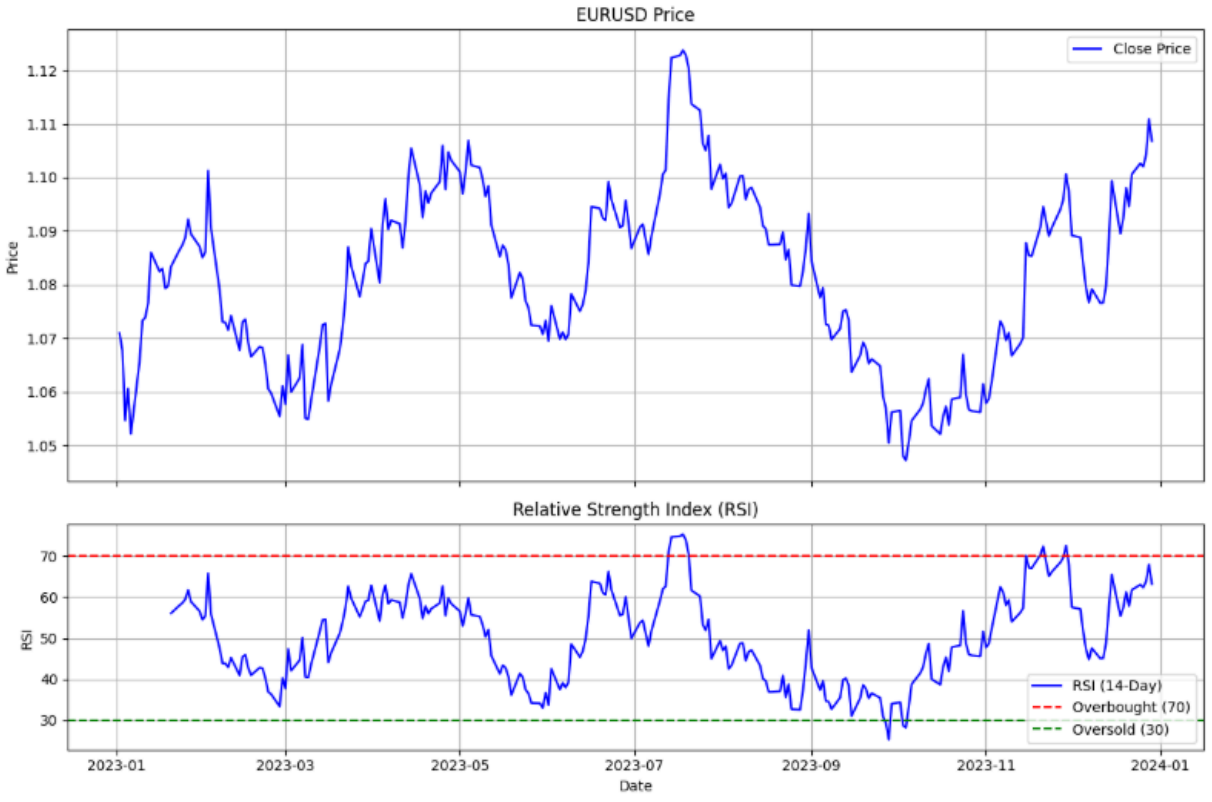
The standard number of periods used to calculate the initial RSI value is 14 ('n' below). Once there are 14 periods of data available, the next calculation can be done. This smooths the RSI so that it will only approach 100 or 0 in a strongly trending market.

Step 2: $$RSI_{Step\ 2} = 100 - [\frac{100}{1 + \frac{(Prev\ Avg\ gain * (n-1)) + Current\ Gain}{(Prev\ Avg\ loss * (n-1)) + Current\ Loss}}]$$

RSI is typically plotted below an asset's price chart. The RSI will rise as the size and number of "up days" increases, and fall as the number and size of "down days" increases.
RSI values can be calculated for a dataset using the below function:

[15]:

```python
def calculate_rsi_smoothed(prices: pd.Series, window: int = 14) -> pd.Series:
    """
    Calculate the Relative Strength Index (RSI) using the smoothed formula.

    Parameters:
        prices (pd.Series): Series of prices.
        window (int): Lookback period for RSI (default is 14).

    Returns:
        pd.Series: RSI values.
    """
    # Calculate daily price changes
    delta = prices.diff()

    # Separate gains and losses
    gain = delta.where(delta > 0, 0)  # Positive price changes
    loss = -delta.where(delta < 0, 0)  # Absolute value of negative price changes

    # Initialise the first average gain and loss
    avg_gain = gain.rolling(window=window, min_periods=window).mean()
    avg_loss = loss.rolling(window=window, min_periods=window).mean()

    # Create Series to store RSI values
    rsi = pd.Series(index=prices.index, dtype="float64")

    # Calculate the first RSI value after the initial window
    for i in range(window, len(prices)):
        if i == window:
            # Initial Avg Gain and Avg Loss
            avg_gain.iloc[i] = gain.iloc[:window].mean()
            avg_loss.iloc[i] = loss.iloc[:window].mean()
        else:
            # Smooth Avg Gain and Avg Loss
            avg_gain.iloc[i] = ((avg_gain.iloc[i - 1] * (window - 1)) + gain.iloc[i]) / window
            avg_loss.iloc[i] = ((avg_loss.iloc[i - 1] * (window - 1)) + loss.iloc[i]) / window

        # Compute RSI
        rs = avg_gain.iloc[i] / avg_loss.iloc[i] if avg_loss.iloc[i] != 0 else np.inf
        rsi.iloc[i] = 100 - (100 / (1 + rs))

    return rsi
```



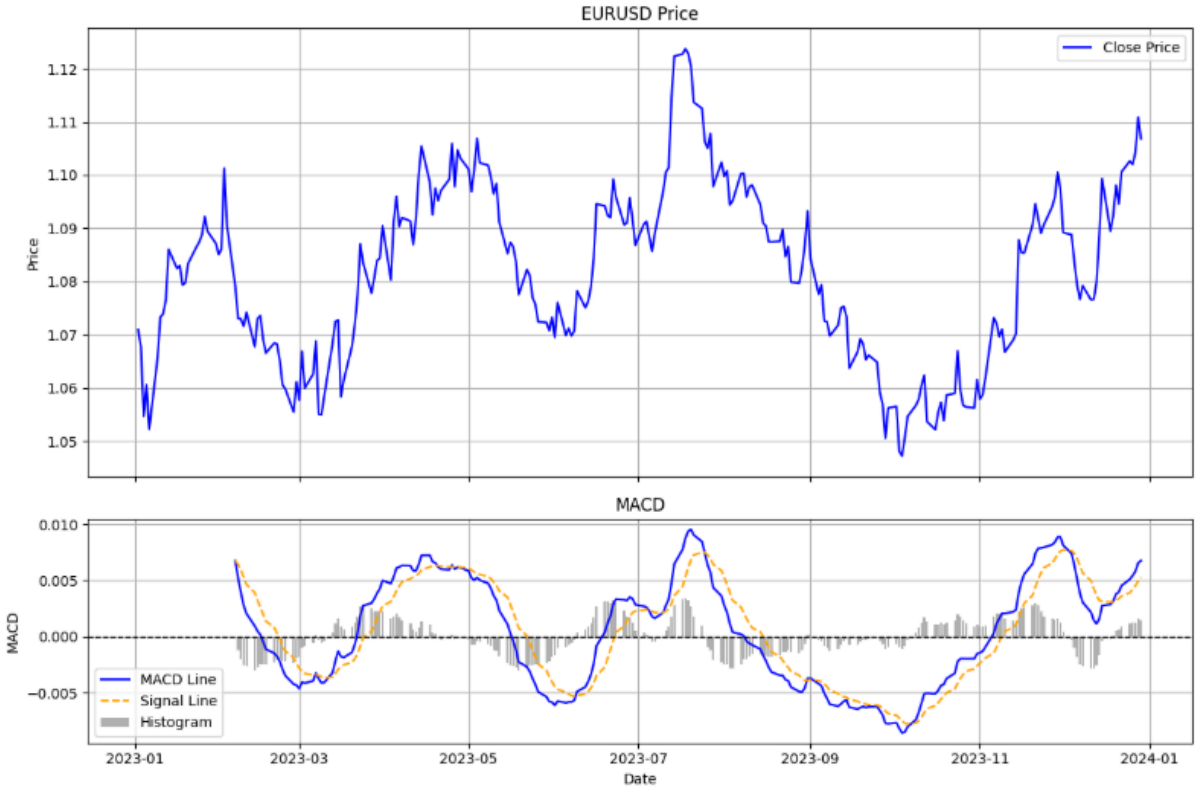
Moving Average Convergence / Divergence (MACD)

MACD is a technical indicator that traders use to identify price trends, measure trend momentum, and to identify entry points for buying/selling an asset. The MACD line is calculated by subtracting a longer-term EMA from a short-term EMA, typically the 26 and 12 day EMA values, creating the MACD line. Then, a nine day EMA of the MACD line is called the signal line, which is used as an indicator for buy/sell signals. A trader would then buy the asset when the MACD line crosses above the signal line, and sell the asset when the MACD crosses below the signal line.

MACD is commonly used together with the RSI. MACD measures the relationship between two EMAs to identify momentum and potential trade reversals, whereas the RSI aims to identify overbought/oversold conditions by evaluating recent price action.

MACD can be calculated by performing the below steps on a Pandas dataframe:

```
# Calc MACD
data["EMA_12"] = calculate_ema(prices=data["Close"], window=12)
data["EMA_26"] = calculate_ema(prices=data["Close"], window=26)
data["MACD"] = data["EMA_12"] - data["EMA_26"]
data["MACD_SIGNAL"] = data["MACD"].ewm(span=9, adjust=False).mean()
data["MACD_HISTO"] = data["MACD"] - data["MACD_SIGNAL"]
```



MACD is known to also produce false positive signals. These occur when the price of an asset is neither trending up or down, but moving sideways. As such, the Average Directional Index (ADX) can be used in conjunction with the MACD. The ADX indicates whether a trend

is in place, with a reading above 25 showing a trend is in place, and a reading of less than 20 showing no trend is in place. Traders using MACD crossovers in trading strategies can use the ADX before actioning the MACD signal. For example if the MACD shows a bullish crossover and the ADX is below 20 (no trend), this may signal that a low has been hit and the trader would be more willing to take the bullish trade.

Average Directional Index (ADX)

As mentioned, the ADX is used by traders to determine the strength of a trend. This trend can be up or down, and is shown by two accompanying indicators, the negative directional indicator (-DI) and the positive directional indicator (+DI). The trend has strength when the ADX is greater than 25 and the trend is weak / there is no trend when ADX is less than 20. A signal below 20 does not necessarily mean there is no trend but it may be that the price is too volatile for a clear direction to be observed. Fourteen periods are typically used to calculate the ADX.

Calculating the ADX:

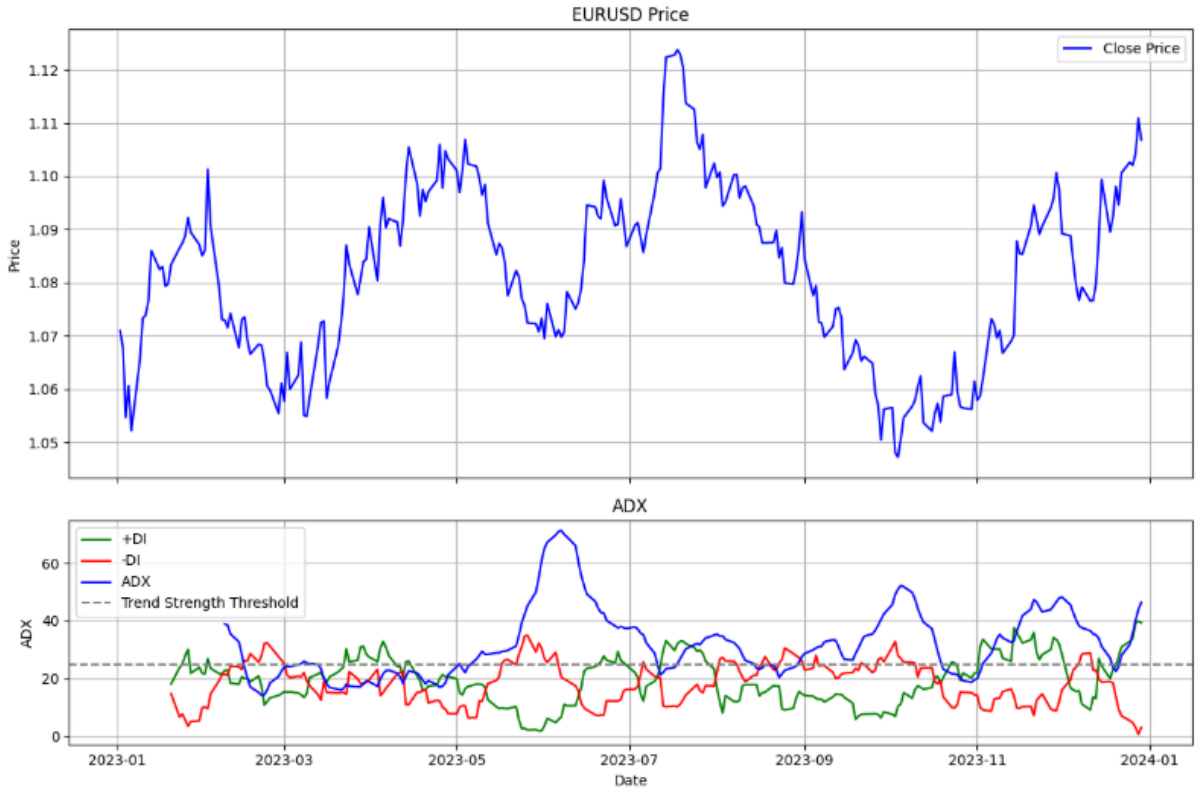
- $True\ Range\ (TR)$ is the max of
  - $Current\ High\ -\ Current\ Low$
  - $|Current\ High\ -\ Prev\ Close|$
  - $|Current\ Low\ -\ Prev\ Close|$
- $Directional\ Movement\ (+\ DM\ and\ -\ DM)$
  - $+\ DM\ =\ Current\ High\ -\ Prev\ High$
    - If > Current Low - Prev Low and positive, otherwise 0
  - $-\ DM\ =\ Prev\ Low\ -\ Current\ Low$
    - If > Prev High - Current High and positive, otherwise 0
- $Smoothed\ Averages$
  - Smoothed $TR,\ +\ DI,\ -\ DI$ are calculated using EMA
- $Directional\ Index\ (DX)\ =\ \frac{|+DI--DI|}{+DI+-DI}\ *\ 100$
- $Average\ Directional\ Index\ (ADX)$ is a moving average of the DX over a specified period (14 periods).

A strong trend is identified when ADX is greater than 25, and a weak/no trend when ADX is less than 20. Crossovers of the -DI and +DI lines can also be used to generate trade signals. For example if the +DI line crosses above the -DI line and the ADX is above 25, this is a potential signal to buy.

ADX can be calculated using the following function:

[24]:

```python
def calculate_adx(data: pd.DataFrame, window: int = 14) -> pd.DataFrame:
    """
    Calculate the Average Directional Index (ADX) along with +DI and -DI.

    Parameters:
        data (pd.DataFrame): DataFrame with columns 'High', 'Low', 'Close'.
        window (int): Lookback period for ADX (default is 14).

    Returns:
        pd.DataFrame: Original DataFrame with +DI, -DI, and ADX columns added.
    """
    # Calculate True Range (TR)
    data["TR"] = np.maximum(data["High"] - data["Low"],
                            np.maximum(abs(data["High"] - data["Close"].shift(1)),
                                       abs(data["Low"] - data["Close"].shift(1))))

    # Calculate +DM and -DM
    data["+DM"] = np.where((data["High"] - data["High"].shift(1)) > (data["Low"].shift(1) - data["Low"]),
                           np.maximum(data["High"] - data["High"].shift(1), 0), 0)
    data["-DM"] = np.where((data["Low"].shift(1) - data["Low"]) > (data["High"] - data["High"].shift(1)),
                           np.maximum(data["Low"].shift(1) - data["Low"], 0), 0)

    # Smooth the TR, +DM, and -DM using a moving average
    data["TR_smooth"] = data["TR"].rolling(window=window).mean()
    data["+DM_smooth"] = data["+DM"].rolling(window=window).mean()
    data["-DM_smooth"] = data["-DM"].rolling(window=window).mean()

    # Calculate +DI and -DI
    data["+DI"] = 100 * (data["+DM_smooth"] / data["TR_smooth"])
    data["-DI"] = 100 * (data["-DM_smooth"] / data["TR_smooth"])

    # Calculate DX
    data["DX"] = 100 * abs(data["+DI"] - data["-DI"]) / (data["+DI"] + data["-DI"])

    # Calculate ADX as a rolling average of DX
    data["ADX"] = data["DX"].rolling(window=window).mean()

    return data
```



Trading Strategies

Mean Reversion

1. EMA & RSI

The first mean reversion trading strategy combines Exponential Moving Averages (EMAs) and the Relative Strength Index (RSI) to identify buying opportunities. It is a long-only trading strategy. The long-only strategy is appropriate for two reasons; 1. Cryptocurrency markets are extremely volatile and though these large swings in asset prices provide market opportunities, a trader can experience serious losses with short positions in cryptocurrencies, and 2. The chosen broker (Alpaca) does not allow for short positions to be opened on cryptocurrencies. A mean reversion trading strategy that uses EMA and RSI values aims to identify trading opportunities to buy/sell when prices deviate from their average and are likely to revert to that average. The EMA is used to calculate the “average” price of the asset, and the RSI is used to confirm oversold/overbought conditions.

The strategy identifies opportunities to open a long position when both:

1. The current price is below the EMA - indicating the market is oversold and deviating from its average trend.

and

2. RSI is low (below an optimised value) - indicating the market is oversold and likely to revert to the mean.

This entry signal combines RSI with deviations from the average price (price vs EMA) to identify oversold conditions that are likely to revert. For risk management purposes, stop losses are set at 5% below the entry price, and take profit set at 10% above the entry price.These are predefined risk-levels to ensure the trades are managed systematically. Positions are managed by exiting trades when stop loss/ take profit levels are hit, or when the RSI goes above an upper threshold.

Parameters to be set / optimised:

- **lower_rsi_band** - A buy signal is triggered when the RSI is below this value, indicating oversold market conditions.
- **upper_rsi_band** - If there is an open long position a sell signal is triggered to close the position if the RSI exceeds this value, as it suggests the asset is overbought and has likely reverted to the mean.
- **rsi_window** - Determines the lookback period for calculating the RSI. A shorter window, e.g. 8 periods, makes the RSI more sensitive to price changes, which will provide more signals for short-term trading.
- **ema_window** - Defines the length of the EMA used to smooth the price data. A 50 period EMA is common. Shorter EMAs (e.g. 20) react more quickly to but may generate more noise, whereas longer EMAs (e.g. 70) are more reliable for identifying major trend reversals but may not capture short-term opportunities.

As with all mean reversion trading strategies, this approach will have limited / poor performance in trending markets. In markets that are trending downward strongly it may generate false buying signals.

The basic outline of this strategy can be seen below. This code is used in the backtesting, but will be adapted to be used in the live trading systems with broker integration:

```
class EmaRsiMeanReversion(Strategy):
    upper_rsi_band = 80
    lower_rsi_band = 30
    rsi_window = 10
    ema_window = 70
    name = "EmaRsiMeanReversion"

    def init(self):
        self.ema = self.I(calculate_ema, self.data[CLOSE],
self.ema_window)
        self.rsi = self.I(calculate_rsi_smoothed, self.data[CLOSE],
self.rsi_window)

    def next(self):
        # Buy Signal: Price crosses below EMA & RSI < 30
        if (crossover(self.ema, self.data[CLOSE]) and
self.data[CLOSE][-1] < self.ema[-1] and
                self.rsi[-1] < self.lower_rsi_band and not
self.position.is_long):
            self.buy(size=EQUITY_FRACTION_TO_TRADE)

            # Set stop loss and take profit level for long
            self.stop_loss = self.data[CLOSE][-1] * 0.95  # Stop loss at
5% below entry price
            self.take_profit = self.data[CLOSE][-1] * 1.10 # Take profit
at 10% above entry price

        # Close long position
        # if stop loss/ take profit triggered or based on trend
        if self.position.is_long:
            if (self.data[CLOSE][-1] <= self.stop_loss or
self.data[CLOSE][-1] >= self.take_profit
                    or self.rsi[-1] > self.upper_rsi_band):
                self.position.close()

        elif self.position.is_short:
            # Should only have long position so exit if short
            self.position.close()
```

2. Bollinger Bands and RSI

The second mean reversion strategy combines the use of Bollinger Bands (BBs) and the Relative Strength Index (RSI) to identify mean reversion opportunities. The strategy aims to identify overbought and oversold conditions using both BBs and RSI, and identify mean-reversion buying opportunities using BBs that show when a price has deviated significantly from the mean.

The strategy identifies opportunities to open a long position when both:

1. The price is below the lower bollinger band - indicating the market is oversold.

and

2. RSI is below a threshold - indicating strong bearish momentum and oversold conditions.

If a long position is open, it will be closed if the RSI exceeds the upper band or the stop loss/ take profit levels are hit (which are set at 5% and 10% respectively again).

Parameters to be set / optimised:

- **bb_window** - Defines the number of periods used to calculate the BBs
    - The middle band is the SMA of the prices over this window
    - The upper and lower bands are equal to SMA +/- 2 * standard deviation over the window
- **rsi_window** - Determines the lookback period for calculating the RSI
- **lower_rsi_band** - A buy signal is triggered when the RSI is below this value, indicating oversold market conditions
- **upper_rsi_band** - If there is an open long position a sell signal is triggered to close the position if the RSI exceeds this value, as it suggests the asset is overbought and has likely reverted to the mean

Again, this strategy will be limited in markets that are trending strongly with no reliable long-term average. It uses Bollinger Bands and RSI to identify mean-reversion opportunities in oversold markets.

This strategy will be compared against the first mean reversion strategy with one being chosen as the “optimal” mean reversion strategy to create a live trading system for.

The basic outline of this strategy can be seen below. This code is used in the backtesting, but will be adapted to be used in the live trading systems with broker integration:

```
class BbRsiMeanReversion(Strategy):
    bb_window = 20
    rsi_window = 14
    lower_rsi_band = 30
    upper_rsi_band = 70
    name = "BbRsiMeanReversion"

    def init(self):
```

```
        # Calc BBs
        self.bb_sma, self.upper_bb, self.lower_bb = self.I(lambda
prices:

calculate_bollinger_bands(prices, self.bb_window),

self.data[CLOSE])
        # Calculate RSI
        self.rsi = self.I(calculate_rsi_smoothed, self.data[CLOSE],
self.rsi_window)

    def next(self):
        # Buy Signal: Price < Lower BB and RSI < 30
        if (crossover(self.data[CLOSE], self.lower_bb) and
self.data[CLOSE][-1] < self.lower_bb[-1] and
            self.rsi[-1] < self.lower_rsi_band and not
self.position.is_long):
            self.buy(size=EQUITY_FRACTION_TO_TRADE)

            # Set stop loss and take profit level for long
            self.stop_loss = self.data[CLOSE][-1] * 0.95  # Stop loss at
5% below entry price
            self.take_profit = self.data[CLOSE][-1] * 1.10  # Take
profit at 10% above entry price


        # Close long position
        # if stop loss/ take profit triggered or based on trend
        if self.position.is_long:
            if (self.rsi[-1] > self.upper_rsi_band or
self.data[CLOSE][-1] <= self.stop_loss or
                    self.data[CLOSE][-1] >= self.take_profit):
                self.position.close()

        elif self.position.is_short:
            # Should only have long position so exit if short
            self.position.close()
```

Trend Following

1. MACD & ADX

The first trend following trading strategy is one that combines Moving Average Convergence Divergence (MACD) to identify trend direction and momentum with Average Directional Index (ADX) to measure the strength of a trend and confirm its validity. It utilises the MACD crossover signal to identify new trends and confirms the strength of the trend using the ADX indicator.

The strategy identifies opportunities to open a long position when all three have occurred:

1. MACD line crosses above the signal line - Identifies a bullish trend
2. +DI > -DI - Confirming the positive directional movement, that the price is trending upward
3. ADX > ADX Threshold - Confirming the trend has sufficient strength to signal a buying opportunity

If a long position is open, it will be closed if the ADX falls below the exit threshold (ADX exit threshold is different to ADX threshold above) as this indicates the trend is weakening, or if the stop loss / take profit levels are hit.

Parameters to be set / optimised:

- **macd_short_window** - The short EMA window used to calculate the MACD line
- **macd_long_window** - The long EMA window used to calculate the MACD line
- **macd_signal_window** - The window used to calculate the signal line. The signal line is an ‘n’ period EMA of the MACD line, referring to this value as ‘n’
- **adx_window** - The lookback period used to calculate the ADX and directional indices (+DI and -DI)
- **adx_threshold** - The minimum value of ADX to confirm a strong trend
- **adx_exit_threshold** - If ADX drops below this value and a long position is open it will be closed, the trend is considered too weak to hold the position

The strategy uses MACD to identify trend direction and momentum, with the ADX ensuring only strong trends are identified as buy signals. The caveat to this, and all trend following strategies, is that they will perform poorly in markets that are not trending but merely trending sideways.

The basic outline of this strategy can be seen below. This code is used in the backtesting, but will be adapted to be used in the live trading systems with broker integration:

```
class MacdAdxTrendFollowing(Strategy):
    adx_window = 13
    adx_threshold = 25
    adx_exit_threshold = 20
    macd_short_window = 14
    macd_long_window = 28
    macd_signal_window = 9
```

```
    name = "MacdAdxTrendFollowing"

    def init(self):
        # Calculate MACD components
        self.macd, self.signal, _ = self.I(calculate_macd,
                                           self.data[CLOSE],
                                           self.macd_short_window,
                                           self.macd_long_window,
                                           self.macd_signal_window)

        # Calculate ADX and directional indices
        adx_data = calculate_adx(self.data.df.copy(), self.adx_window)
        self.adx = self.I(lambda x: adx_data["ADX"].values,
self.data[CLOSE])
        self.plus_di = self.I(lambda x: adx_data["+DI"].values,
self.data[CLOSE])
        self.minus_di = self.I(lambda x: adx_data["-DI"].values,
self.data[CLOSE])

    def next(self):
        # Buy Signal: MACD line crosses above Signal line, +DI > -DI,
ADX > threshold
        if (crossover(self.macd, self.signal) and
                self.plus_di[-1] > self.minus_di[-1] and
                self.adx[-1] > self.adx_threshold and not
self.position.is_long):
            self.buy(size=EQUITY_FRACTION_TO_TRADE)

            # Set stop loss and take profit level for long
            self.stop_loss = self.data[CLOSE][-1] * 0.95  # Stop loss at
5% below entry price
            self.take_profit = self.data[CLOSE][-1] * 1.10  # Take
profit at 10% above entry price


        # Exit Condition: ADX falls below exit threshold or stop loss/
take profit triggered
        if self.position.is_long:
            if (self.adx[-1] < self.adx_exit_threshold or
self.data[CLOSE][-1] <= self.stop_loss
                    or self.data[CLOSE][-1] >= self.take_profit):
                self.position.close()

        elif self.position.is_short:
            # Should only have long position so exit if short
            self.position.close()
```

2. EMA & ADX

The second trend following strategy combines Exponential Moving Averages (EMAs) to identify trend direction using EMA crossovers, and the Average Directional Index (ADX) to measure the strength of these trends. Bullish momentum can be identified by the short-term EMA crossing above the longer-term EMA, and the strength of this trend will be measured by the ADX.

The strategy identifies opportunities to open a long position when all three have occurred:

1. The short-term EMA crosses above the longer-term EMA - Indicating a bullish trend
2. +DI > -DI - Confirming the positive directional movement, that the price is trending upward
3. ADX > ADX Threshold - Confirming the trend has sufficient strength to signal a buying opportunity

If a long position is open, it will be closed if the ADX falls below the exit threshold (ADX exit threshold is different to ADX threshold above) as this indicates the trend is weakening, or if the stop loss / take profit levels are hit.

Parameters to be set / optimised:

- **ema_short_window** - Reacting faster to price changes, capturing short-term trends
- **ema_long_window** - Smooths the price over a longer period, capturing the overall trend
- **adx_window** - The lookback period used to calculate the ADX and directional indices (+DI and -DI)
- **adx_threshold** - The minimum value of ADX to confirm a strong trend
- **adx_exit_threshold** - If ADX drops below this value and a long position is open it will be closed, the trend is considered too weak to hold the position

Again, the caveat with this strategy is that it will perform poorly in markets that are not trending but merely trending sideways. This strategy will be compared against the first trend following strategy with one being chosen as the “optimal” trend following strategy to create a live trading system for.

The basic outline of this strategy can be seen below. This code is used in the backtesting, but will be adapted to be used in the live trading systems with broker integration:

```
class EmaAdxTrendFollowing(Strategy):
    adx_window = 14
    adx_threshold = 25
    adx_exit_threshold = 20
    ema_short_window = 12
    ema_long_window = 26
    name = "EmaAdxTrendFollowing"

    def init(self):
        # Calculate EMAs
```

```
        self.short_ema = self.I(lambda prices:
pd.Series(prices).ewm(span=self.ema_short_window, adjust=False).mean(),
                                self.data[CLOSE])
        self.long_ema = self.I(lambda prices:
pd.Series(prices).ewm(span=self.ema_long_window, adjust=False).mean(),
                               self.data[CLOSE])

        # Calculate ADX and directional indices
        adx_data = calculate_adx(self.data.df.copy(), self.adx_window)
        self.adx = self.I(lambda x: adx_data["ADX"].values,
self.data[CLOSE])
        self.plus_di = self.I(lambda x: adx_data["+DI"].values,
self.data[CLOSE])
        self.minus_di = self.I(lambda x: adx_data["-DI"].values,
self.data[CLOSE])


    def next(self):
        if not self.position:
        # Buy Signal: Short EMA crosses above Long EMA, +DI > -DI, ADX >
threshold
            if (crossover(self.short_ema, self.long_ema) and
                    self.plus_di[-1] > self.minus_di[-1] and
                    self.adx[-1] > self.adx_threshold):
                self.buy(size=EQUITY_FRACTION_TO_TRADE)

                # Set stop loss and take profit level for long
                self.stop_loss = self.data[CLOSE][-1] * 0.95  # Stop
loss at 5% below entry price
                self.take_profit = self.data[CLOSE][-1] * 1.10  # Take
profit at 10% above entry price


            # Exit Condition: ADX falls below exit threshold or stop
loss/ take profit triggered
            if self.position.is_long:
                if (self.adx[-1] < self.adx_exit_threshold or
self.data[CLOSE][-1] <= self.stop_loss
                        or self.data[CLOSE][-1] >= self.take_profit):
                    self.position.close()
            elif self.position.is_short:
                # Should only have long position so exit if short
                self.position.close()
```

Backtesting & Parameter Optimisation

As the (non parameterised) trading strategies have been created, the next step is to backtest the strategies on different datasets, optimise the parameters for each strategy and decide what asset(s) to trade and on what frequency. The “Backtesting.py” package was used to perform the backtesting and optimisations.

This testing has been carried out on the following assets

- BCHUSD (Bitcoin Cash) - BCHUSD trades at a fraction of the price of a Bitcoin. It has been chosen as the backtesting package did not allow for fractional trading.
- ETHUSD (Ethereum) - One of the most actively traded cryptocurrencies globally.
- USDTUSD - A stablecoin that is pegged to the US dollar.

Each backtest was carried out assuming a $1,000,000 portfolio. The Backtesting.py documentation recommends setting commission for forex trading to be a 0.2% spread, this has been assumed to carry over to trading these highly liquid cryptocurrencies.

The backtesting and optimisations were performed on six months of data from 01/07/2023 to 31/12/2023. The data was resampled into periods of 5, 15, 30 and 60 minute intervals, with each time frequency optimised and tested.

The strategies were optimised to maximise the return to drawdown ratio. The return to drawdown ratio evaluates the risk-adjusted returns of a trading strategy. It measures the relationship between the returns achieved and the maximum drawdown experienced during the period. The ratio provides an analysis on the efficiency of a strategy’s risk.

$$Return\ to\ Drawdown\ Ratio\ =\ ABS(\frac{Return\ \%}{Max\ Drawdown\ \%})$$

- $Return\ \%$ is the total return of the strategy over the period, expressed as a percentage.
- $Max\ Drawdown\ \%$ is the largest peak to trough decline in portfolio value experience over the period, expressed as a percentage.

The return to drawdown ratio is an effective tool as it balances returns with the risk of significant losses. In real trading systems, if a trading strategy has large drawdowns it will be pulled from production and will never realise potentially large returns that would have occurred post drawdown. In order for the strategy to survive it must not have great drawdowns. A high return to drawdown ratio indicates a strategy generates strong returns relative to its drawdowns, which signals efficient risk-taking. The return to drawdown ratio is a suitable metric for strategies where the goal is the preservation of capital, as is the case here. As such, the maximum trade size has been set to 2% of overall capital for a single trade, which is $20,000 at the start of the strategy. This value updates dynamically as the portfolio equity changes in value and the max trade size remains fixed at 2%.

The strategies were then tested as “live” strategies against one year’s data from 01/01/2024 to 31/12/2024 to ensure adequate time for the selected trading strategies to be evaluated.

As discussed in the Trading Strategies section, each strategy had different parameters that could be optimised. These optimisation ranges can be found below:

Mean Reversion Strategy 1: Exponential Moving Averages & Relative Strength Index

- **lower_rsi_band** - Optimised value between 10 and 50 in steps of 10 to be selected
- **upper_rsi_band** - Optimised value between 50 and 100 in steps of 10 to be selected
    - With the constraint that upper RSI band > lower RSI band
- **rsi_window** - Optimised value between 8 and 18 in steps of 2 to be selected
- **ema_window** - Optimised value between 30 and 80 in steps of 10 to be selected

These ranges have been selected so that the “industry standard” values are within the possible values.

Mean Reversion Strategy 2: Bollinger Bands & Relative Strength Index

- **bb_window** - Optimised value between 16 and 24 in steps of 2 to be selected
- **rsi_window** - Optimised value between 8 and 18 in steps of 2 to be selected
- **lower_rsi_band** - Optimised value between 10 and 50 in steps of 10 to be selected
- **upper_rsi_band** - Optimised value between 50 and 100 in steps of 10 to be selected
    - With the constraint that upper RSI band > lower RSI band

Trend Following Strategy 1: Moving Average Convergence Divergence & Average Directional Index

- **macd_short_window** - Optimised value between 8 and 16 in steps of 2 to be selected
- **macd_long_window** - Optimised value between 24 and 30 in steps of 2 to be selected
- **macd_signal_window** - Hardcoded to the value of 9 time periods.
- **adx_window** - Optimised value between 7 and 16 in steps of 2 to be selected
- **adx_threshold** - Hardcoded to the value of 25, the industry standard strong trend threshold
- **adx_exit_threshold** - Hardcoded to the value of 20, the industry standard weak / weakening trend threshold

Trend Following Strategy 2: Exponential Moving Averages & Average Directional Index

- **ema_short_window** - Optimised value between 5 and 21 in steps of 1 to be selected
- **ema_long_window** - Optimised value between 50 and 110 in steps of 10 to be selected
- **adx_window** - Optimised value between 7 and 16 in steps of 2 to be selected
- **adx_threshold** - Hardcoded to the value of 25, the industry standard strong trend threshold
- **adx_exit_threshold** - Hardcoded to the value of 20, the industry standard weak / weakening trend threshold

All optimisations have been specified to optimise the return to drawdown ratio. The optimisation is completed using a grid search, which performs an exhaustive search over the cartesian product of parameter combinations.

Unnamed: 0	Return %	Ann. Ret %	Ann. Vol %	Max Drawdown %	R2D Ratio	Win Rate %	Num Trades	Optimal Params
EmaRsiMeanReversion_BCHUSD_30min	0.7988	1.5908	0.4535	-0.184	4.3406	90	10	['rsi_window: 10', 'lower_rsi_band: 30', 'upper_rsi_band: 80', 'ema_window: 70']
EmaRsiMeanReversion_ETHUSD_60min	0.2912	0.5784	0.2607	-0.0948	3.0713	71.6667	60	['rsi_window: 8', 'lower_rsi_band: 40', 'upper_rsi_band: 50', 'ema_window: 50']
EmaRsiMeanReversion_BCHUSD_5min	0.2165	0.4299	0.1438	-0.0391	5.5408	85.7143	14	['rsi_window: 8', 'lower_rsi_band: 20', 'upper_rsi_band: 70', 'ema_window: 70']
EmaRsiMeanReversion_BCHUSD_60min	0.1727	0.3429	0.1067	-0.0209	8.2771	100	6	['rsi_window: 8', 'lower_rsi_band: 30', 'upper_rsi_band: 50', 'ema_window: 30']
EmaRsiMeanReversion_ETHUSD_15min	0.1528	0.3034	0.2795	-0.1126	1.3574	70.5882	17	['rsi_window: 16', 'lower_rsi_band: 40', 'upper_rsi_band: 70', 'ema_window: 30']
EmaRsiMeanReversion_BCHUSD_15min	0.103	0.2045	0.1229	-0.0395	2.611	100	1	['rsi_window: 10', 'lower_rsi_band: 20', 'upper_rsi_band: 70', 'ema_window: 50']
EmaRsiMeanReversion_ETHUSD_30min	0.0558	0.1107	0.0505	-0.0097	5.7474	100	4	['rsi_window: 10', 'lower_rsi_band: 30', 'upper_rsi_band: 50', 'ema_window: 30']
EmaRsiMeanReversion_USDTUSD_15min	0.0277	0.0549	0.0424	-0.0051	5.4183	100	1	['rsi_window: 8', 'lower_rsi_band: 20', 'upper_rsi_band: 90', 'ema_window: 30']
EmaRsiMeanReversion_USDTUSD_5min	0.0237	0.0471	0.0442	-0.0111	2.1304	50	2	['rsi_window: 14', 'lower_rsi_band: 40', 'upper_rsi_band: 90', 'ema_window: 40']
EmaRsiMeanReversion_ETHUSD_5min	0.0184	0.0366	0.026	-0.0012	14.9112	100	1	['rsi_window: 10', 'lower_rsi_band: 20', 'upper_rsi_band: 50', 'ema_window: 30']
BbRsiMeanReversion_USDTUSD_60min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_USDTUSD_30min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_USDTUSD_15min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_USDTUSD_5min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_ETHUSD_60min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_ETHUSD_30min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_ETHUSD_15min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_ETHUSD_5min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_BCHUSD_60min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_BCHUSD_30min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_BCHUSD_15min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
BbRsiMeanReversion_BCHUSD_5min	0	0	0	0	*NaN*	*NaN*	0	['bb_window: 16', 'rsi_window: 8', 'lower_rsi_band: 10', 'upper_rsi_band: 50']
EmaRsiMeanReversion_USDTUSD_60min	-0.0091	-0.018	0.0067	-0.0091	1	0	4	['rsi_window: 8', 'lower_rsi_band: 30', 'upper_rsi_band: 60', 'ema_window: 30']
EmaRsiMeanReversion_USDTUSD_30min	-0.0326	-0.0646	0.0148	-0.0326	1	0	11	['rsi_window: 14', 'lower_rsi_band: 40', 'upper_rsi_band: 60', 'ema_window: 60']

The results for the mean reversion backtesting can be seen below:

From analysing the optimised results the decision has been made to select the EMA RSI Mean Reversion for BCHUSD using a 30 minute frequency. This strategy performed well in terms of the return to drawdown ratio (4.34), in comparison to the rest of the strategies, and it had a stronger return than the rest of the time horizons. It had an impressive 90% win rate over the period. The Bollinger Band and RSI strategies did not find any trading opportunities, with no clear inefficiencies signals found over the six month period. For this backtesting, mean reversion of the asset has been assumed, although it was shown in the previous section that the BCHUSD price series was non-stationary. So the lack of mean reversion trading opportunities found by this strategy is not unreasonable. As such, the optimal parameters have been selected as:

<u>EMA RSI Mean Reversion</u>

- **lower_rsi_band** - 30
- **upper_rsi_band** - 80
- **rsi_window** - 10
- **ema_window** - 70
- **Data frequency** - 30 minutes

Similarly, the results for the trend following backtests can be seen below:

Unnamed: 0	Return %	Ann. Ret %	Ann. Vol %	Max Drawdown %	R2D Ratio	Win Rate %	Num Trades	Optimal Params
MacdAdxTrendFollowing_BCHUSD_30min	0.6139	1.2901	1.0558	-0.6269	0.9793	34.1463	82	['adx_window: 13', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 14', 'macd_long_window: 28', 'macd_si
MacdAdxTrendFollowing_BCHUSD_60min	0.4447	0.9083	1.1323	-0.6362	0.699	45.098	51	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 14', 'macd_long_window: 28', 'macd_sig
EmaAdxTrendFollowing_ETHUSD_30min	0.3662	0.7277	0.8283	-0.4971	0.7366	100	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 5', 'ema_long_window: 60']
EmaAdxTrendFollowing_ETHUSD_15min	0.366	0.7311	0.8283	-0.6407	0.5713	100	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 8', 'ema_long_window: 50']
EmaAdxTrendFollowing_ETHUSD_60min	0.3529	0.7013	0.828	-0.4943	0.714	100	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 9', 'ema_long_window: 100']
EmaAdxTrendFollowing_ETHUSD_5min	0.3487	0.7312	0.8285	-0.6408	0.5442	100	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 5', 'ema_long_window: 50']
MacdAdxTrendFollowing_ETHUSD_60min	0.2916	0.5792	0.479	-0.2336	1.2483	39.3939	33	['adx_window: 15', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 12', 'macd_long_window: 28', 'macd_si
EmaAdxTrendFollowing_USDTUSD_5min	-0.0034	0.0006	0.0134	-0.03	0.1141	0	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 5', 'ema_long_window: 50']
EmaAdxTrendFollowing_USDTUSD_15min	-0.0038	0.0006	0.0134	-0.028	0.1371	0	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 5', 'ema_long_window: 50']
EmaAdxTrendFollowing_USDTUSD_60min	-0.0039	0.0006	0.0134	-0.0109	0.3589	0	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 5', 'ema_long_window: 60']
EmaAdxTrendFollowing_USDTUSD_30min	-0.0044	-0.0087	0.0146	-0.0109	0.3996	0	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 9', 'ema_long_window: 80']
MacdAdxTrendFollowing_USDTUSD_60min	-0.1333	-0.2643	0.0323	-0.1338	0.9966	0	33	['adx_window: 13', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 12', 'macd_long_window: 24', 'macd_si
MacdAdxTrendFollowing_USDTUSD_30min	-0.1631	-0.3233	0.0365	-0.1631	1	0	41	['adx_window: 13', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 10', 'macd_long_window: 28', 'macd_si
EmaAdxTrendFollowing_BCHUSD_5min	-0.1763	-0.2783	0.9867	-0.9153	0.1926	0	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 10', 'ema_long_window: 100']
EmaAdxTrendFollowing_BCHUSD_60min	-0.1956	-0.3877	0.9682	-0.7962	0.2457	0	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 5', 'ema_long_window: 80']
EmaAdxTrendFollowing_BCHUSD_15min	-0.1984	-0.2742	0.9722	-0.8963	0.2214	0	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 5', 'ema_long_window: 50']
EmaAdxTrendFollowing_BCHUSD_30min	-0.2167	-0.4294	0.9542	-0.8117	0.267	0	1	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'ema_short_window: 6', 'ema_long_window: 100']
MacdAdxTrendFollowing_ETHUSD_30min	-0.5021	-0.9879	0.6313	-0.6779	0.7406	28.4314	102	['adx_window: 9', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 8', 'macd_long_window: 26', 'macd_signl
MacdAdxTrendFollowing_USDTUSD_15min	-0.5349	-1.0407	0.0588	-0.5349	1	0	131	['adx_window: 11', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 12', 'macd_long_window: 24', 'macd_si
MacdAdxTrendFollowing_BCHUSD_5min	-1.0073	-1.7673	0.7246	-1.0652	0.9456	29.8734	395	['adx_window: 15', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 12', 'macd_long_window: 28', 'macd_si
MacdAdxTrendFollowing_ETHUSD_15min	-1.2121	-2.3728	0.5944	-1.2692	0.955	26.8156	179	['adx_window: 9', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 14', 'macd_long_window: 28', 'macd_sig
MacdAdxTrendFollowing_BCHUSD_15min	-1.356	-2.6496	1.1541	-1.6818	0.8063	32.9545	176	['adx_window: 7', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 8', 'macd_long_window: 24', 'macd_signl
MacdAdxTrendFollowing_ETHUSD_5min	-2.0473	-3.9407	0.5532	-2.0591	0.9943	24.4165	557	['adx_window: 11', 'adx_threshold: 25', 'adx_exit_threshold: 20', 'macd_short_window: 8', 'macd_long_window: 26', 'macd_sig

The return to drawdown ratios, annualised returns and win rate percentages were lower for the trend following strategies than for the mean reversion strategies as a whole. The chosen strategy here is MACD ADX trend following for BCHUSD, also using the 30 minute frequency of data. Its return to drawdown ratio of 0.98 and return of 0.61% make it arguably the best performing strategy relative to the competing strategies. The win rate of 34% however is one that is not promising in a backtest. The optimal parameters to be implemented in the live trading system are:

<u>MACD ADX Trend Following</u>

- **macd_short_window** - 14
- **macd_long_window** - 28
- **macd_signal_window** - 9
- **adx_window** - 13
- **adx_threshold** - 25
- **adx_exit_threshold** - 20
- **Data frequency** - 30 minutes

Part 2: Broker API

Alpaca is the chosen broker to implement the mean reversion and trend following strategies algorithmically. Alpaca offers market data and trading for stocks, options and cryptocurrencies through its API. It offers commission free cryptocurrency trading, with traders paying the spread plus other account fees.

The code for each strategy can be seen in the below code blocks, and is also attached in the python files P2Amean_reversion_live_trading.py & P2Btrend_follow_live_trading.py. The trading strategies are fully automated trading systems that pull recent historical data, request new OHLC bars on a specified frequency, calculate technical indicators and check for buy signals, position management and API management such as error handling. The scripts are set to run for fifteen minutes at a time to avoid a build up of cache, as well as any restarts needed for broken connections. The scripts can be run as often as required via a cron job or using Windows Task Scheduler.

Order Types

Alpaca offers multiple order types through its API, each of which will be discussed. For cryptocurrency trading specifically, Alpaca offers Good Till Cancelled (GTC) and Immediate or Cancel (IOC), with additional order types for equity traders.

Day Order

A day order remains active only on the trading day on which it was placed. If it is not executed by the end of the trading session it is cancelled automatically. It is often used by day traders to avoid carrying positions overnight, and to avoid overnight exposure to risks such as earnings announcements or other geopolitical events.

Good Till Cancelled (GTC)

A good till cancelled order remains open until it is either executed or cancelled by the user that placed the trade. Unlike the day order, a GTC order can last for many trading sessions. This type of order is useful for capturing opportunities at specific target prices. Less frequent monitoring is needed with a GTC order.

On the Open (OPG)

An OPG order is executed only during the market opening auction. If not executed at the open, it is automatically cancelled. It is often used to capture opportunities during the opening auction where significant price discovery occurs.

On the Close (CLS)

A CLS order is executed during the market's closing auction. If not executed during this time period, it is cancelled automatically. It is useful to ensure fair execution at a market's official closing price.

Immediate or Cancel (IOC)

An IOC order requires that any part of the order that can be executed immediately is filled, with the remaining portion cancelled. It is often used by high frequency trading strategies to ensure rapid execution without part of the order remaining unfilled. It captures available liquidity instantly, without risking partial fills at worse prices later. It avoids stale orders by cancelling unexecuted portions immediately. However it may fail to execute significant quantities in illiquid markets.

Fill or Kill (FOK)

A fill or kill order requires the entire order to be executed immediately. If the full order cannot be filled at the specified price, the entire order is cancelled. It is often used by institutional traders to execute large orders without partial fills. It has a greater likelihood of not being filled in illiquid markets, as it relies on immediate liquidity, which may not always be available.

Selected Order Type - GTC

For cryptocurrency trading, Alpaca offers IOC and GTC order types. It has been decided to use GTC orders for both the mean reversion and trend following trading strategies.

With a mean reversion strategy the objective is to buy low and sell high by exploiting price deviations from a long-term average price. Trades are required to be executed at specific price levels that are deemed far from the long-term average. GTC orders ensure the price of the asset deviates enough to trigger the trade and it avoids missing out on trade as the market does not need to be continuously monitored

With a trend following strategy the objective is to take advantage of sustained price trends in the direction of that trend. GTC orders allow trend following traders to place orders at breakout points and wait for the market to trigger them. Trends can take some time to form so a GTC order allows the trader to ensure the trade is executed and not cancelled before the signal materialises.

See below sections for full code and comments that explain the systems step by step.

Mean Reversion Strategy Live Trading Code - EMA & RSI

```
"""Script to implement live trading of strategy using EMA Crossovers and Relative Strength Index with
Alpaca's API"""

import time
from datetime import timedelta, datetime
from logging.handlers import RotatingFileHandler

import pandas as pd
from alpaca.data import CryptoHistoricalDataClient, CryptoBarsRequest, TimeFrame, TimeFrameUnit
from alpaca.trading import TradingClient, OrderSide, MarketOrderRequest, TimeInForce
from decouple import config
import sys
import logging

# Configure the logger
LOG_FILENAME = "mean_reversion_logs.log"
logger = logging.getLogger("MeanReversionLogger")
logger.setLevel(logging.DEBUG)
# File handler
file_handler = RotatingFileHandler(LOG_FILENAME, maxBytes=5 * 1024 * 1024, backupCount=2)  # 5MB per log
file
file_handler.setLevel(logging.DEBUG)
# Console handler
console_handler = logging.StreamHandler()
console_handler.setLevel(logging.DEBUG)
# Formatter
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
file_handler.setFormatter(formatter)
console_handler.setFormatter(formatter)
# Add handlers to the logger
logger.addHandler(file_handler)
logger.addHandler(console_handler)

# Get API keys
alpaca_api_key = config("ALPACA_MEAN_KEY")
alpaca_secret = config("ALPACA_MEAN_SECRET")

# Instantiate Trading Client
api = TradingClient(api_key=alpaca_api_key, secret_key=alpaca_secret, raw_data=True)

# set some global variables
SYMBOL: str = "symbol"
OPEN: str = "Open"
HIGH: str = "High"
LOW: str = "Low"
CLOSE: str = "Close"
VOLUME: str = "Volume"
TRADE_COUNT: str = "Trade_count"
VWAP: str = "Vwap"
EMA: str= "Ema"
RSI: str = "Rsi"
EQUITY: str = "equity"
LONG: str= "long"

class EmaRsiMeanReversionLiveTradingApp:
   def __init__(self):
       self.symbol = "BCH/USD"
       self.minute_interval = 30
       self.rsi_window = 10
       self.ema_window = 70
       self.upper_rsi_band = 80
       self.lower_rsi_band = 30
       self.client = CryptoHistoricalDataClient(api_key=alpaca_api_key, secret_key=alpaca_secret)
       self.data = None
       self.stop_loss = None
```

```
        self.take_profit = None
        self.current_position = None
        self.position_open = None

    def fetch_historical_data(self):
        logger.info("Fetching historical data.")
        data_request = CryptoBarsRequest(
            symbol_or_symbols=self.symbol,
            start=datetime.now() - timedelta(days=5),
            end=datetime.now(),
            timeframe=TimeFrame(self.minute_interval, TimeFrameUnit.Minute),
        )
        self.data = self.client.get_crypto_bars(data_request).df
        # Reset index to isolate 'timestamp' level as it is a multiindex
        self.data = self.data.reset_index(level=SYMBOL, drop=True)
        self.data.index = pd.to_datetime(self.data.index)
        logger.info("Historical data fetched.")

    def fetch_latest_data(self):
        try:
            # Ensure there is existing data to get the last timestamp
            if self.data is not None and not self.data.empty:
                last_timestamp = self.data.index[-1]
                self.data.columns = self.data.columns.str.capitalize()
            else:
                raise ValueError("No existing data found. Fetch historical data first.")

            logger.info("Fetching new data from the last timestamp onward.")
            data_request = CryptoBarsRequest(
                symbol_or_symbols=self.symbol,
                start=last_timestamp + timedelta(seconds=1),  # Avoid duplicate data
                end=datetime.now(),
                timeframe=TimeFrame(self.minute_interval, TimeFrameUnit.Minute),
            )
            new_data = self.client.get_crypto_bars(data_request).df

            if not new_data.empty:
                logger.info("New data fetched, adding to existing dataset.")
                new_data = new_data.reset_index(level=SYMBOL, drop=True)
                new_data.index = pd.to_datetime(new_data.index)
                new_data.columns = new_data.columns.str.capitalize()

                # Ensure new data has only original columns
                original_columns = [OPEN, HIGH, LOW, CLOSE, VOLUME, TRADE_COUNT, VWAP]
                new_data = new_data[original_columns]

                # Add missing columns to new_data
                for col in self.data.columns:
                    if col not in new_data.columns:
                        new_data[col] = None

                # Combine the historical and new data
                combined_data = pd.concat([self.data, new_data]).drop_duplicates()
                self.data = combined_data.sort_index()
                logger.info("Latest data added.")
            else:
                logger.info("No new data available.")
        except Exception as e:
            logger.error(f"Error fetching latest data: {e}")

    def calculate_indicators(self):
        logger.info("Calculating EMA.")
        self.data[EMA] = self.data[CLOSE].ewm(span=self.ema_window, adjust=False).mean()

        logger.info("Calculating RSI.")
        delta = self.data[CLOSE].diff()
        gain = (delta.where(delta > 0, 0)).rolling(window=self.rsi_window).mean()
        loss = (-delta.where(delta < 0, 0)).rolling(window=self.rsi_window).mean()
```

```
        rs = gain / loss
        self.data[RSI] = 100 - (100 / (1 + rs))
        logger.info("Indicators Calculated.")

    def calculate_position_size(self):
        logger.info("Calculating position size to trade.")
        account = api.get_account()
        equity = float(account[EQUITY])  # Get the current account equity
        risk_amount = equity * 0.02  # 2% of equity
        current_price = self.data[CLOSE].iloc[-1]

        # Ensure current price is valid and greater than zero
        if current_price <= 0:
            raise ValueError("Invalid current price for position size calculation.")

        position_size = risk_amount / current_price  # Number of units to trade
        return position_size

    def execute_trade(self, side: OrderSide, size: float):
        try:
            logger.info(f"Executing {side} order of size {size}")
            order = MarketOrderRequest(
                symbol=self.symbol.replace("/", ""),
                qty=size,
                side=side,
                time_in_force=TimeInForce.GTC
            )
            api.submit_order(order)
            logger.info(f"Submitted {side} order with size {size}.")
            logger.info("Verifying if order has been executed.")
            if api.get_all_positions():
                logger.info("Open position found.")
                self.position_open = True
                # can only go long
                self.current_position = LONG
                logger.info("Setting stop loss and take profit levels.")
                self.stop_loss = self.data[CLOSE].iloc[-1] * 0.95
                self.take_profit = self.data[CLOSE].iloc[-1] * 1.10
            else:
                logger.info("No open position found, resetting position tracker, stop loss and take profit
levels.")
                self.position_open = False
                self.current_position = None
                self.stop_loss = None
                self.take_profit = None

        except Exception as e:
            logger.info("Error trying to execute trade, handling...")
            logger.error(f"Error executing trade: {e}")
            logger.info("Verifying if there is an open position.")
            if api.get_all_positions():
                self.position_open = True
            else:
                self.position_open = False

    def manage_position(self):
        logger.info("Checking if signal detected or stop loss / take profit hit.")
        if self.current_position == LONG:
            if (self.data[CLOSE].iloc[-1] <= self.stop_loss or self.data[CLOSE].iloc[-1] >=
self.take_profit or
                    self.data[RSI].iloc[-1] > self.upper_rsi_band):
                logger.info("Closing long position.")
                self.close_position()


    def close_position(self):
        logger.info("Closing the current position.")
        self.execute_trade(OrderSide.SELL if self.current_position == LONG else OrderSide.BUY,
```

```
                        size=self.calculate_position_size())

    def run(self):
        start_time = datetime.now()
        # script will only run for the length of the max_duration
        # it should be scheduled via cron job or Windows Task Scheduler to run multiple times per day to
avoid
        # cache building up
        max_duration = timedelta(minutes=15)
        self.fetch_historical_data()

        while True:
            if datetime.now() - start_time > max_duration:
                logger.info(f"Reached maximum runtime of {max_duration}. Exiting.")
                sys.exit()
            logger.info("Fetching latest data...")
            self.fetch_latest_data()
            self.calculate_indicators()

            # Check for open positions
            if self.position_open:
                logger.info("Managing existing position.")
                self.manage_position()
            else:
                logger.info("Checking for buy signals.")
                position_size = self.calculate_position_size()

                if (
                        self.data[CLOSE].iloc[-2] > self.data[EMA].iloc[-2]  # Previous close above EMA
                        and self.data[CLOSE].iloc[-1] < self.data[EMA].iloc[
                    -1]  # Current close below EMA (crossover)
                        and self.data[RSI].iloc[-1] < self.lower_rsi_band  # RSI below lower band
                ):
                    logger.info("Buy signal detected.")
                    logger.info("Attempting to execute trade.")
                    self.execute_trade(OrderSide.BUY, size=position_size)

            # set script to sleep to avoid making too many calls to API
            # 200 calls per minute is the maximum allowed
            # this script makes much fewer calls than the permitted maximum
            sleep_amount = 60
            logger.info(f"Sleeping for {sleep_amount} seconds")
            time.sleep(sleep_amount)

if __name__ == "__main__":
    try:
        app = EmaRsiMeanReversionLiveTradingApp()
        app.run()
    except KeyboardInterrupt:
        print("Program interrupted by user. Exiting.")
        logger.info("Program interrupted by user. Exiting.")
```

Trend Following Strategy Live Trading Code - MACD & ADX

```
"""Script to implement live trading of strategy using MACD Crossovers and Average Directional Index with
Alpaca's API"""
import time
from datetime import timedelta, datetime
from logging.handlers import RotatingFileHandler

import numpy as np
import pandas as pd
from alpaca.data import CryptoHistoricalDataClient, CryptoBarsRequest, TimeFrame, TimeFrameUnit
from alpaca.trading import TradingClient, OrderSide, MarketOrderRequest, TimeInForce, ClosePositionRequest
from decouple import config
import sys
import logging


# Configure the logger
LOG_FILENAME = "trend_follow_logs.log"
logger = logging.getLogger("MeanReversionLogger")
logger.setLevel(logging.DEBUG)
# File handler
file_handler = RotatingFileHandler(LOG_FILENAME, maxBytes=5 * 1024 * 1024, backupCount=2)
file_handler.setLevel(logging.DEBUG)
# Console handler
console_handler = logging.StreamHandler()
console_handler.setLevel(logging.DEBUG)
# Formatter
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
file_handler.setFormatter(formatter)
console_handler.setFormatter(formatter)
# Add handlers to the logger
logger.addHandler(file_handler)
logger.addHandler(console_handler)

# Get API keys
# change for specific trend follow keys
alpaca_api_key = config("ALPACA_TREND_KEY")
alpaca_secret = config("ALPACA_TREND_SECRET")

# Instantiate Trading Client
api = TradingClient(api_key=alpaca_api_key, secret_key=alpaca_secret, raw_data=True)

# set some global variables
SYMBOL: str = "symbol"
OPEN: str = "Open"
HIGH: str = "High"
LOW: str = "Low"
CLOSE: str = "Close"
VOLUME: str = "Volume"
TRADE_COUNT: str = "Trade_count"
VWAP: str = "Vwap"
EQUITY: str = "equity"
LONG: str= "long"
MACD: str = "Macd"
MACD_SIGNAL: str = "Macd_signal"
TR: str = "Tr"
PLUS_DM: str = "+dm"
MINUS_DM: str = "-dm"
TR_SMOOTH: str = "Tr_smooth"
PLUS_DM_SMOOTH: str = "+dm_smooth"
MINUS_DM_SMOOTH: str = "-dm_smooth"
PLUS_DI: str = "+di"
MINUS_DI: str = "-di"
DX: str = "Dx"
ADX: str = "Adx"
```

```python
class MacdAdxLiveTradingApp:
    def __init__(self):
        self.symbol = "BCH/USD"
        self.minute_interval = 30
        self.adx_window = 13
        self.adx_threshold = 25
        self.adx_exit_threshold = 20
        self.macd_short_window = 14
        self.macd_long_window = 28
        self.macd_signal_window = 9
        self.client = CryptoHistoricalDataClient(api_key=alpaca_api_key, secret_key=alpaca_secret)
        self.data = None
        self.stop_loss = None
        self.take_profit = None
        self.current_position = None
        self.position_open = None

    def fetch_historical_data(self):
        """Get 5 days of data to calculate indicators form the start."""
        logger.info("Fetching historical data.")
        data_request = CryptoBarsRequest(
            symbol_or_symbols=self.symbol,
            start=datetime.now() - timedelta(days=5),
            end=datetime.now(),
            timeframe=TimeFrame(self.minute_interval, TimeFrameUnit.Minute),
        )
        self.data = self.client.get_crypto_bars(data_request).df
        # Reset index to isolate 'timestamp' level
        self.data = self.data.reset_index(level=SYMBOL, drop=True)
        self.data.index = pd.to_datetime(self.data.index)
        logger.info("Historical data fetched.")

    def fetch_latest_data(self):
        try:
            # Ensure there's existing data to get the last timestamp
            if self.data is not None and not self.data.empty:
                last_timestamp = self.data.index[-1]
                self.data.columns = self.data.columns.str.capitalize()
            else:
                raise ValueError("No existing data found. Fetch historical data first.")

            logger.info("Fetching new data from the last timestamp onward.")
            data_request = CryptoBarsRequest(
                symbol_or_symbols=self.symbol,
                start=last_timestamp + timedelta(seconds=1),  # Avoid duplicate data
                end=datetime.now(),
                timeframe=TimeFrame(self.minute_interval, TimeFrameUnit.Minute),
            )
            new_data = self.client.get_crypto_bars(data_request).df

            if not new_data.empty:
                logger.info("New data fetched, adding to existing dataset.")
                new_data = new_data.reset_index(level=SYMBOL, drop=True)
                new_data.index = pd.to_datetime(new_data.index)
                new_data.columns = new_data.columns.str.capitalize()

                # Ensure new data has only original columns
                original_columns = [OPEN, HIGH, LOW, CLOSE, VOLUME, TRADE_COUNT, VWAP]
                new_data = new_data[original_columns]

                # Add missing columns to new_data
                for col in self.data.columns:
                    if col not in new_data.columns:
                        new_data[col] = None

                # Combine the historical and new data
                combined_data = pd.concat([self.data, new_data]).drop_duplicates()
```

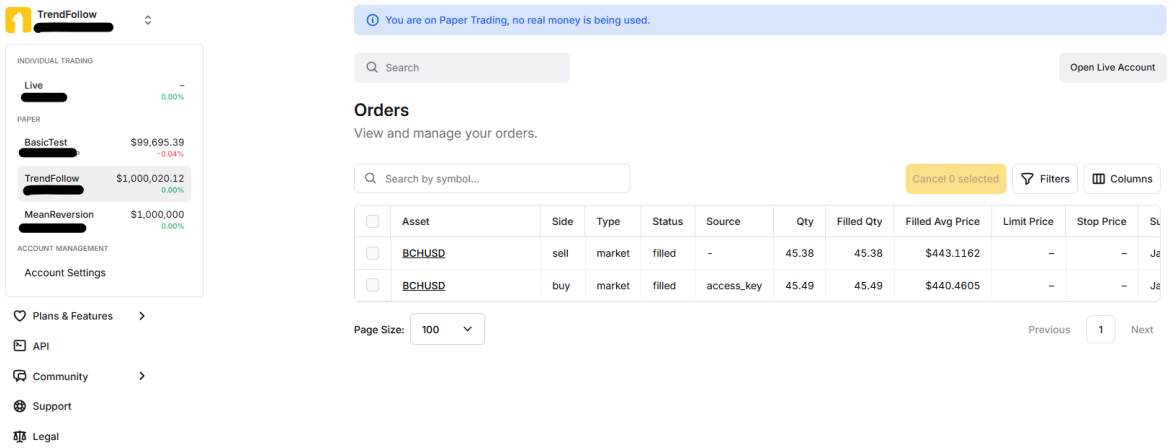
```
                self.data = combined_data.sort_index()
                logger.info("Latest data added.")
            else:
                logger.info("No new data available.")
        except Exception as e:
            logger.error(f"Error fetching latest data: {e}")

    def calculate_indicators(self):
        logger.info("Calculating MACD.")
        short_ema = self.data[CLOSE].ewm(span=self.macd_short_window, adjust=False).mean()
        long_ema = self.data[CLOSE].ewm(span=self.macd_long_window, adjust=False).mean()
        self.data[MACD] = short_ema - long_ema
        self.data[MACD_SIGNAL] = self.data[MACD].ewm(span=self.macd_signal_window, adjust=False).mean()

        logger.info("Calculating ADX.")
        self.data[TR] = np.maximum(
            self.data[HIGH] - self.data[LOW],
            np.maximum(abs(self.data[HIGH] - self.data[CLOSE].shift(1)),
                       abs(self.data[LOW] - self.data[CLOSE].shift(1))),
        )
        self.data[PLUS_DM] = np.where(
            (self.data[HIGH] - self.data[HIGH].shift(1)) > (self.data[LOW].shift(1) - self.data[LOW]),
            np.maximum(self.data[HIGH] - self.data[HIGH].shift(1), 0), 0
        )
        self.data[MINUS_DM] = np.where(
            (self.data[LOW].shift(1) - self.data[LOW]) > (self.data[HIGH] - self.data[HIGH].shift(1)),
            np.maximum(self.data[LOW].shift(1) - self.data[LOW], 0), 0
        )

        self.data[TR_SMOOTH] = self.data[TR].ewm(span=self.adx_window, adjust=False).mean()
        self.data[PLUS_DM_SMOOTH] = self.data[PLUS_DM].ewm(span=self.adx_window, adjust=False).mean()
        self.data[MINUS_DM_SMOOTH] = self.data[MINUS_DM].ewm(span=self.adx_window, adjust=False).mean()

        self.data[PLUS_DI] = 100 * (self.data[PLUS_DM_SMOOTH] / self.data[TR_SMOOTH])
        self.data[MINUS_DI] = 100 * (self.data[MINUS_DM_SMOOTH] / self.data[TR_SMOOTH])
        self.data[DX] = 100 * abs(self.data[PLUS_DI] - self.data[MINUS_DI]) / (self.data[PLUS_DI] +
self.data[MINUS_DI])
        self.data[ADX] = self.data[DX].ewm(span=self.adx_window, adjust=False).mean()
        logger.info("Indicators Calculated.")

    def calculate_position_size(self):
        logger.info("Calculating position size to trade.")
        account = api.get_account()
        equity = float(account[EQUITY])
        risk_amount = equity * 0.02
        current_price = self.data[CLOSE].iloc[-1]

        # Ensure current price is valid and greater than zero
        if current_price <= 0:
            raise ValueError("Invalid current price for position size calculation.")

        position_size = risk_amount / current_price  # Number of units to trade
        return position_size

    def execute_trade(self, side: OrderSide, size: float):
        try:
            logger.info(f"Executing {side} order of size {size}.")
            order = MarketOrderRequest(
                symbol=self.symbol.replace("/", ""),
                qty=size,
                side=side,
                time_in_force=TimeInForce.GTC
            )
            api.submit_order(order)
            logger.info(f"Submitted {side} order with size {size}.")
            logger.info("Verifying if order has been executed.")
            if api.get_all_positions():
                logger.info("Open position found.")
```

```
                self.position_open = True
                # can only go long
                self.current_position = LONG
                logger.info("Setting stop loss and take profit levels.")
                self.stop_loss = self.data[CLOSE].iloc[-1] * 0.95
                self.take_profit = self.data[CLOSE].iloc[-1] * 1.10
            else:
                logger.info("No open position found, resetting position tracker, stop loss and take profit
levels.")
                self.position_open = False
                self.current_position = None
                self.stop_loss = None
                self.take_profit = None

        except Exception as e:
            logger.info("Error trying to execute trade, handling...")
            logger.error(f"Error executing trade: {e}")
            logger.info("Verifying if there is an open position.")
            if api.get_all_positions():
                self.position_open = True
            else:
                self.position_open = False

    def manage_position(self):
        logger.info("Checking if ADX is below the exit threshold.")
        if self.data[ADX].iloc[-1] < self.adx_exit_threshold:
            logger.info("ADX below exit threshold. Closing position.")
            self.close_position()
            return

        logger.info("Checking if stop loss or take profit levels have been hit.")
        if self.current_position == LONG:
            if self.data[CLOSE].iloc[-1] <= self.stop_loss or self.data[CLOSE].iloc[-1] >=
self.take_profit:
                logger.info("Stop-loss or take-profit hit for long position. Closing position.")
                self.close_position()

    def close_position(self):
        logger.info("Closing the current position.")
        self.execute_trade(OrderSide.SELL if self.current_position == LONG else OrderSide.BUY,
                           size=self.calculate_position_size())

    def run(self):

        start_time = datetime.now()
        # script will only run for the length of the max_duration
        # it should be scheduled via cron job or Windows Task Scheduler to run multiple times per day to
avoid
        # cache building up
        max_duration = timedelta(minutes=15)
        self.fetch_historical_data()

        while True:
            if datetime.now() - start_time > max_duration:
                logger.info(f"Reached maximum runtime of {max_duration}. Exiting.")
                sys.exit()
            logger.info("Fetching latest data...")
            self.fetch_latest_data()
            self.calculate_indicators()

            # Check for open positions
            if self.position_open:
                logger.info("Managing existing position.")
                self.manage_position()
            else:
                logger.info("Checking for buy signals.")
                position_size = self.calculate_position_size()
```

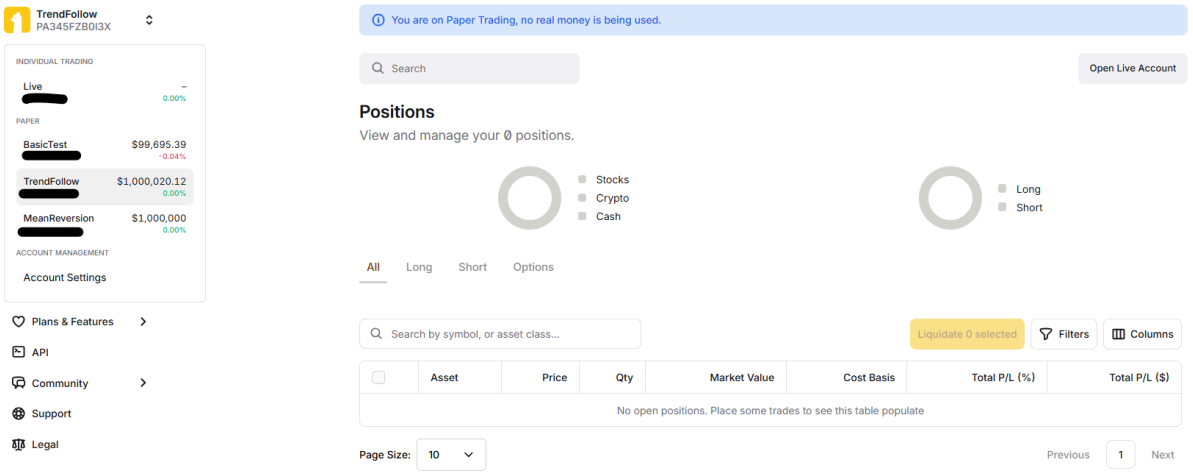
```
            if (
                    self.data[MACD].iloc[-1] > self.data[MACD_SIGNAL].iloc[-1]
                    and self.data[PLUS_DI].iloc[-1] > self.data[MINUS_DI].iloc[-1]
                    and self.data[ADX].iloc[-1] > self.adx_threshold
            ):
                logger.info("Buy signal detected.")
                logger.info("Attempting to execute trade.")
                self.execute_trade(OrderSide.BUY, size=position_size)

        # set script to sleep to avoid making too many calls to API
        # 200 calls per minute is the maximum allowed
        # this script makes much fewer calls than the permitted maximum
        sleep_amount = 60
        logger.info(f"Sleeping for {sleep_amount} seconds.")
        time.sleep(sleep_amount)


if __name__ == "__main__":
    try:
        app = MacdAdxLiveTradingApp()
        app.run()
    except KeyboardInterrupt:
        logger.info("Program interrupted by user. Exiting.")
```

These strategies were run for four hours from approximately 11/01/2025 17:30 to 11/01/2025 21:30. The full logs for each strategy can be found in the attached files P2Dtrend_follow_logs.log & P2Cmean_reversion_logs.log. Throughout this time period all events and requests were handled successfully. The trend following strategy identified one buy signal and executed it algorithmically. At the end of the manually specified four hours, the position was still open and being managed (stop loss, take profit or sell signals were not hit) so I liquidated the position as I am not running the strategies live currently.

Both orders can be seen below:



There are no positions held after liquidation:


TrendFollow
PA345FZB0I3X
INDIVIDUAL TRADING
Live
–
0.00%
PAPER
BasicTest
$99,695.39
-0.04%
TrendFollow
$1,000,020.12
0.00%
MeanReversion
$1,000,000
0.00%
ACCOUNT MANAGEMENT
Account Settings
Plans & Features
API
Community
Support
Legal
You are on Paper Trading, no real money is being used.
Search
Open Live Account
Positions
View and manage your 0 positions.
Stocks
Crypto
Cash
Long
Short
All
Long
Short
Options
Search by symbol, or asset class...
Liquidate 0 selected
Filters
Columns
Asset
Price
Qty
Market Value
Cost Basis
Total P/L (%)
Total P/L ($)
No open positions. Place some trades to see this table populate
Page Size:
10
Previous
1
Next

Part 3: Systematic Backtesting

While event handling, position management and market data validation are all handled in the second section of the report, systematic backtesting can be completed using the Backtesting.py package on the chosen strategies. The strategy optimisation was completed using six months of data from 01/07/2023 to 31/12/2023. For the systematic backtesting, the strategies have been tested using data for all of 2024, with a starting equity of $1,000,000

Mean Reversion Strategy: EMA & RSI

The strategy identifies opportunities to open a long position when both:

1. The current price is below the EMA - indicating the market is oversold and deviating from its average trend.

and

2. RSI is below 30 - indicating the market is oversold and likely to revert to the mean.

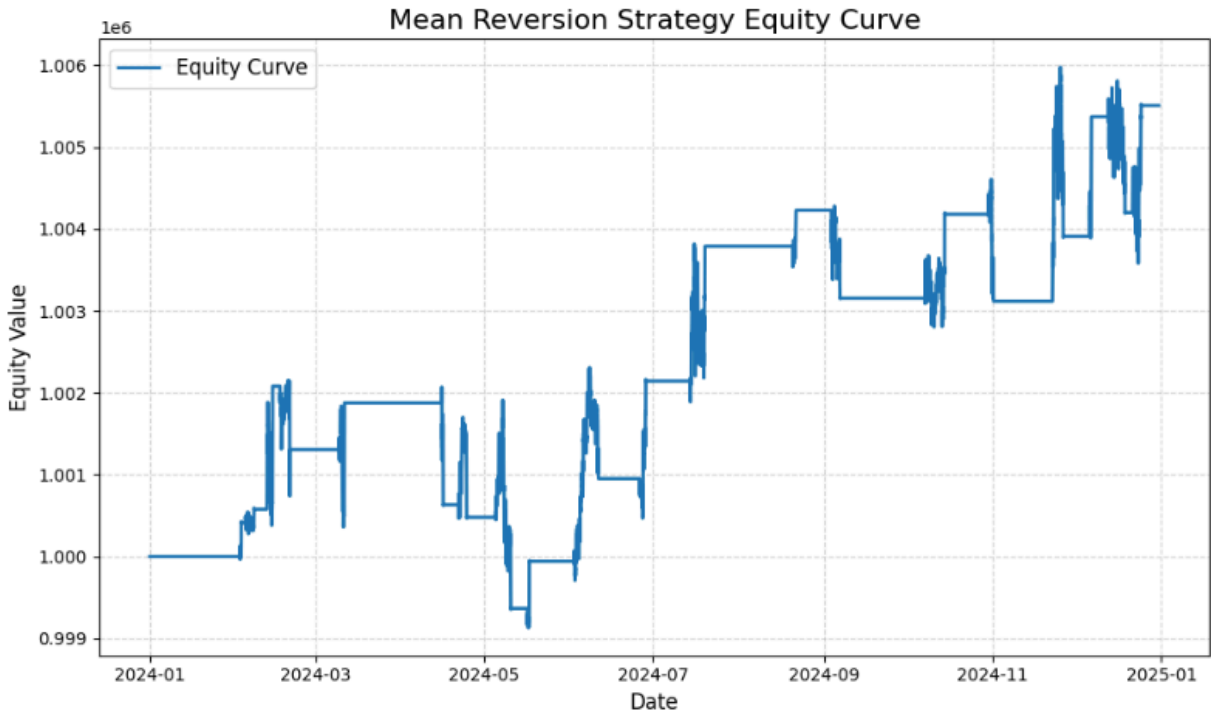
<u>EMA RSI Mean Reversion Parameters</u>

- **lower_rsi_band** - 30
- **upper_rsi_band** - 80
- **rsi_window** - 10
- **ema_window** - 70
- **Data frequency** - 30 minutes

Some summary statistics can be seen below:

key	Results
Start	2024-01-01 00:00:00+00:00
End	2024-12-31 00:00:00+00:00
Duration	365 days 00:00:00
Exposure Time [%]	17.428049337597077
Equity Final [$]	1005508.830615958
Equity Peak [$]	1005977.350898958
Return [%]	0.5508830615957966
Buy & Hold Return [%]	70.89457764377315
Return (Ann.) [%]	0.5493737897401374
Volatility (Ann.) [%]	0.5451411090298006
Sharpe Ratio	1.0077643763059252
Sortino Ratio	1.655240523019148
Calmar Ratio	1.8241217568358903
Max. Drawdown [%]	-0.3011716666836306
Avg. Drawdown [%]	-0.04995080745210068
Max. Drawdown Duration	108 days 19:00:00
Avg. Drawdown Duration	7 days 08:36:00
# Trades	23
Win Rate [%]	60.86956521739131
Best Trade [%]	10.515826012709084
Worst Trade [%]	-6.564812275989629
Avg. Trade [%]	1.0385696864688043
Max. Trade Duration	7 days 03:00:00
Avg. Trade Duration	2 days 17:51:00
Profit Factor	1.559633196217651
Expectancy [%]	1.211041732045951
SQN	0.9597693303479203
_strategy	EmaRsiMeanReversion

The strategy achieved an annual return of 0.55%, a Sharpe ratio of 1 and a return to drawdown ratio of 1.83 (calculated as $\left|\frac{0.55}{-0.3}\right|$). This was completed in a total of 23 trades throughout the year. The starting equity of $1,000,000 peaked at $1,005,977 and finished at $1,005,509. The equity curve can be seen below.



The strategy performed well at preserving capital, but did not provide great returns. This strategy will not be analysed in depth due to its low number of trades. The trend following strategy will be discussed in detail as it resulted in a large number of trades.

Trend Following Strategy: MACD & ADX

The strategy identifies opportunities to open a long position when all three have occurred:

1. MACD line crosses above the signal line - Identifies a bullish trend
2. +DI > -DI - Confirming the positive directional movement, that the price is trending upward
3. ADX > 25 - Confirming the trend has sufficient strength to signal a buying opportunity

MACD ADX Trend Following

- **macd_short_window** - 14
- **macd_long_window** - 28
- **macd_signal_window** - 9
- **adx_window** - 13
- **adx_threshold** - 25
- **adx_exit_threshold** - 20
- **Data frequency** - 30 minutes

Some summary statistics can be seen below:

key	Results
Start	2024-01-01 00:00:00+00:00
End	2024-12-31 00:00:00+00:00
Duration	365 days 00:00:00
Exposure Time [%]	40.360895386021014
Equity Final [$]	998447.1723965514
Equity Peak [$]	1009649.424981
Return [%]	-0.1552827603448648
Buy & Hold Return [%]	70.89457764377315
Return (Ann.) [%]	-0.12677857827151362
Volatility (Ann.) [%]	1.107481514329727
Sharpe Ratio	0.0
Sortino Ratio	0.0
Calmar Ratio	0.0
Max. Drawdown [%]	-1.603663328459326
Avg. Drawdown [%]	-0.12314600958181708
Max. Drawdown Duration	270 days 17:00:00
Avg. Drawdown Duration	12 days 02:43:00
# Trades	178
Win Rate [%]	35.95505617977528
Best Trade [%]	14.867644884272613
Worst Trade [%]	-6.440858558167928
Avg. Trade [%]	-0.13139932065400206
Max. Trade Duration	3 days 14:00:00
Avg. Trade Duration	0 days 19:22:00
Profit Factor	0.9712802226309125
Expectancy [%]	-0.042322577287928514
SQN	-0.13627147762892391
_strategy	MacdAdxTrendFollowing

The trend following strategy using MACD and ADX had an annualised return of -0.13% and a return to drawdown ratio of 0.08. Its win rate of 35% approximately matches the 34% win rate the strategy achieved in the backtesting using six months of 2023 data. The starting

equity of $1,000,000 peaked at $1,009,649 and finished at $998,447. The equity curve can be seen below.

An equity curve is a graphical representation of the value of a trading account over time. It tracks the cumulative changes in the account's equity, including the realised profits/losses, the unrealised profits/losses, and any other changes to the equity account balance.

How an Equity Curve is Calculated

1. **Initial Equity** - the starting balance of the account is denoted as E_0
2. **Realised PnL** - the profit or loss (P_n) from each trade is calculated as:

$$P_n = Q * (P_{sell} - P_{buy}) - C$$

Where:

- Q: Quantity of the asset traded
- P_{buy} and P_{sell}: Buy and sell prices
- C: Transaction costs, e.g. spread

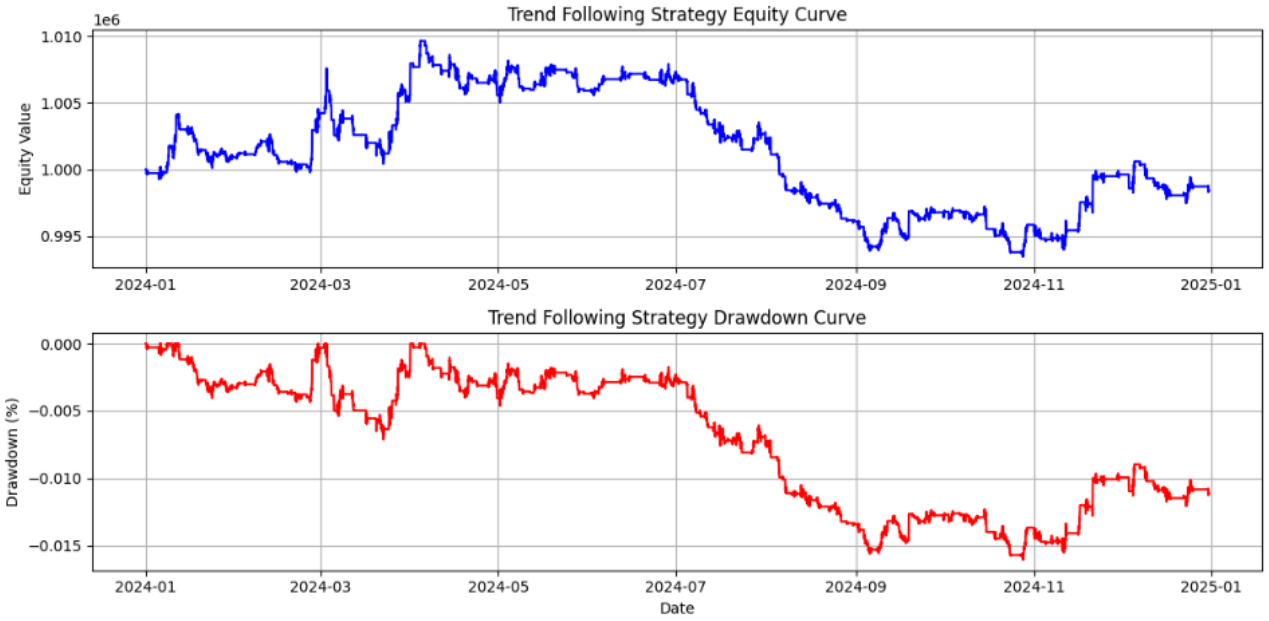
3. After each trade the equity is updated:

$$E_{n+1} = E_n + P_n$$

4. Unrealised PnL (the mark-to-market value of open positions) is added to the equity value

Equity curves are useful as they provide an intuitive visualisation of the changes to the equity value throughout the trading period, and it also highlights drawdowns and volatility clearly. The strategy had a profitable period for the majority of the first four months of the year, with equity declining from approximately halfway through the year onward.

A drawdown refers to how much the trading account is down from a peak before it recovers back to that peak, typically quoted as a percentage. They are a useful tool to measure the downside volatility of a trading strategy. The time it takes to recover from a drawdown is also important to consider. The time taken to recover from a loss, the drawdown duration, is key when analysing the robustness of a trading strategy.



The average drawdown duration was 12 days, but the maximum drawdown duration was 270 days, a potentially catastrophic result for a strategy that was only active for 365 days. There were a total of 178 trades placed, with each trade open for an average of 19 hours and 22 minutes, with the longest trade open 3 days and 14 hours.

Sharpe Ratio

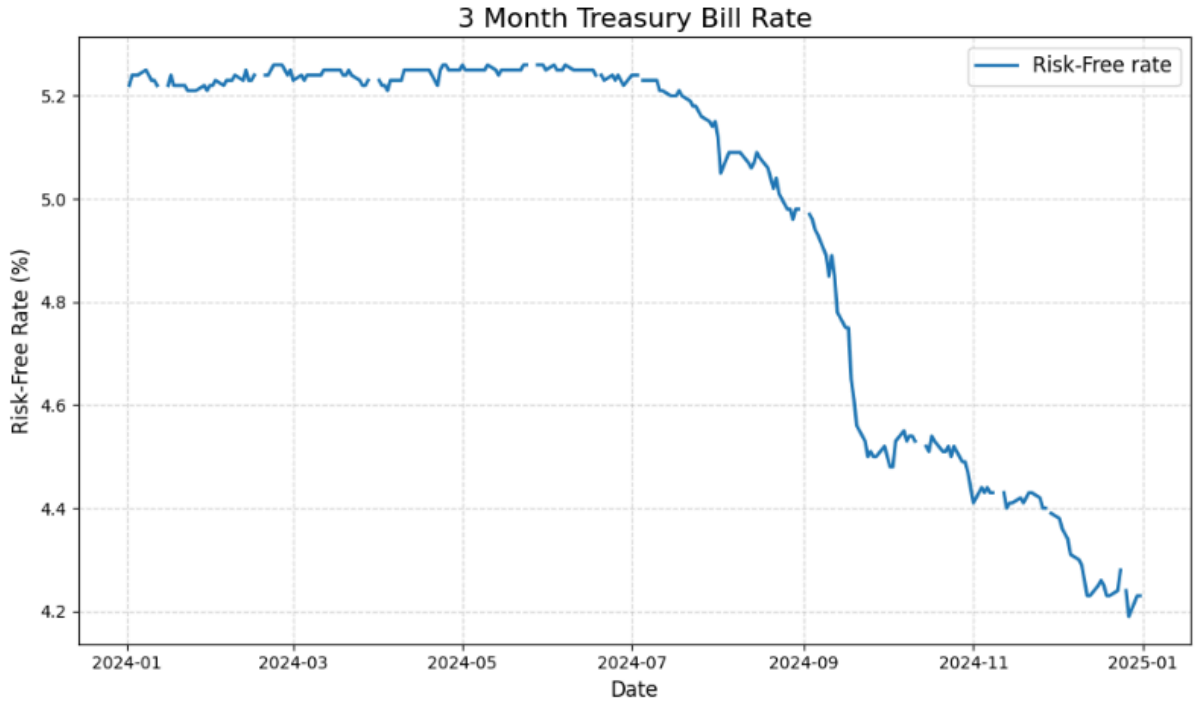
The Sharpe ratio measures the risk-adjusted return of a trading strategy or investment portfolio. It can be calculated as:

$$Sharpe\ Ratio\ = \frac{R_p - R_f}{\sigma_p}$$

Where:

- R_p is the return of the portfolio
- R_f is the return of the risk-free rate
- σ_p is the standard deviation of the portfolio returns

A common proxy for the risk-free rate are rates of 3-month US Treasury bills, as can be seen below for 2024 using data sourced from the Federal Reserve Bank of St. Louis (FRED):



The average rate for 2024 was 4.97%. The strategy returned -0.13% and had an annualised volatility of 1.11%. The Sharpe ratio can be seen below:

$$\frac{-0.0013 - .047}{.011} = -4.39$$

A Sharpe ratio of -4.39 shows that the return of the strategy is significantly less than the return on the risk-free rate. The strategy is not offering adequate returns to compensate for the risk taken. This would indicate that the trend following strategy is not worth the risk, it would need to be addressed before making this strategy live in a non paper trading account.

The rolling Sharpe ratio is another useful metric for analysing risk-adjusted returns, showing how the Sharpe Ratio evolves over time. The formula is the same as above, but is only applied to a rolling window of returns. The 30 day rolling Sharpe of the trend following strategy for 2024 can be seen below:

The Sharpe ratio was negative for the majority of the rolling windows, reaching its lowest value of approximately -15 in September.

Value at Risk (VaR)

VaR is a metric that quantifies the potential loss in the value of a portfolio over a specified time period, at a given level of confidence. It provides a number to the question of what is the maximum loss that can be expected to to occur, with a specified level of confidence.

Two common methods of calculating VaR are via historical simulation, and by the parametric (variance-covariance) method that assumes returns follow a normal distribution.

1. <u>VaR Historical Simulation</u>

- Sort the returns series in ascending order
- Find the return at the desired confidence level, e.g. at a 95% confidence level find the 5th percentile
- Calculate VaR by multiplying the value by the value of the equity in the portfolio

2. <u>Parametric (Variance-Covariance) VaR</u>

- Calculate the mean (μ) and the standard deviation (σ) of historical returns
- Determine the z-score corresponding to the level of confidence
    - z = -1.645 for 95% confidence
    - z = -2.33 for 99% confidence
- Calculate VaR:
    - $VaR = Portfolio\ Equity\ *\ (\mu\ +\ z\ *\ \sigma)$

To scale VaR from a one-time period (e.g. 1-day) metric to a 10-day metric as is commonly done, the single time period VaR must be multiplied by $\sqrt{10}$.

As the trading strategy is using 30-minute data, scaling is required to achieve a 1-day VaR figure. Returns must be scaled to match the desired time horizon (1-day). The scaling is based on the square root of time as volatility grows proportionately to the square root of the time interval. There are 48 30-minute time periods in one day, resulting in:

$$\sigma_{1Day} = \sigma_{30Minute} * \sqrt{48}$$

VaR is a risk metric that provides an easy to interpret measure of potential losses for a portfolio. It should be used as a stand alone metric, but one that along with other metrics can provide valuable insight into the risk of a trading strategy.

The below functions can be used to calculate 1-Day VaR:

```python
# Historical Simulation Method
def historical_var(returns, portfolio_value, confidence_level):
    """
    Calculate 1-Day VaR for 30 minute data using Historical Simulation.
    """
    # Aggregate returns over daily intervals
    daily_returns = [
        np.sum(returns[i:i + 48])  # Aggregate 48 intervals for daily returns
        for i in range(0, len(returns) - 48, 48)
    ]
    sorted_returns = np.sort(returns)  # Sort returns in ascending order
    percentile_index = int((1 - confidence_level) * len(sorted_returns))
    var_percentile = sorted_returns[percentile_index]
    var_value = portfolio_value * abs(var_percentile)
    return var_value, var_percentile

# Parametric (Variance-Covariance) Method
def parametric_var(returns, portfolio_value, confidence_level):
    """
    Calculate 1-Day VaR for 30 minute data using Parametric (Variance-Covariance) Method.
    """
    mean_return = np.mean(returns)
    scaling_factor = np.sqrt(48) # There are 48 30-minute periods in a day
    std_dev = np.std(returns) * scaling_factor
    z_score = norm.ppf(1 - confidence_level)  # Z-score for the confidence level
    var_percentile = mean_return + z_score * std_dev
    var_value = portfolio_value * abs(var_percentile)
    return var_value, var_percentile
```

```python
# Calculate VaR
oned_var_historical_95, oned_percentile_historical_95 = historical_var(returns=tf_equity_curve["Return"].dropna(), portfolio_value=1000000, confidence_level=0.95)
oned_var_historical_99, oned_percentile_historical_99 = historical_var(returns=tf_equity_curve["Return"].dropna(), portfolio_value=1000000, confidence_level=0.99)

oned_var_parametric_95, oned_percentile_parametric_95 = parametric_var(returns=tf_equity_curve["Return"].dropna(), portfolio_value=1000000, confidence_level=0.95)
oned_var_parametric_99, oned_percentile_parametric_99 = parametric_var(returns=tf_equity_curve["Return"].dropna(), portfolio_value=1000000, confidence_level=0.99)

print(f"Historical 1-day VaR (95% confidence): ${oned_var_historical_95:,.2f}, Percentile: {oned_percentile_historical_95:.4f}")
print(f"Parametric 1-day VaR (95% confidence): ${oned_var_parametric_95:,.2f}, Percentile: {oned_percentile_parametric_95:.4f}")
print("\n")
print(f"Historical 1-day VaR (99% confidence): ${oned_var_historical_99:,.2f}, Percentile: {oned_percentile_historical_99:.4f}")
print(f"Parametric 1-day VaR (99% confidence): ${oned_var_parametric_99:,.2f}, Percentile: {oned_percentile_parametric_99:.4f}")
```

```
Historical 1-day VaR (95% confidence): $112.20, Percentile: -0.0001
Parametric 1-day VaR (95% confidence): $940.97, Percentile: -0.0009


Historical 1-day VaR (99% confidence): $252.73, Percentile: -0.0003
Parametric 1-day VaR (99% confidence): $1,330.80, Percentile: -0.0013
```

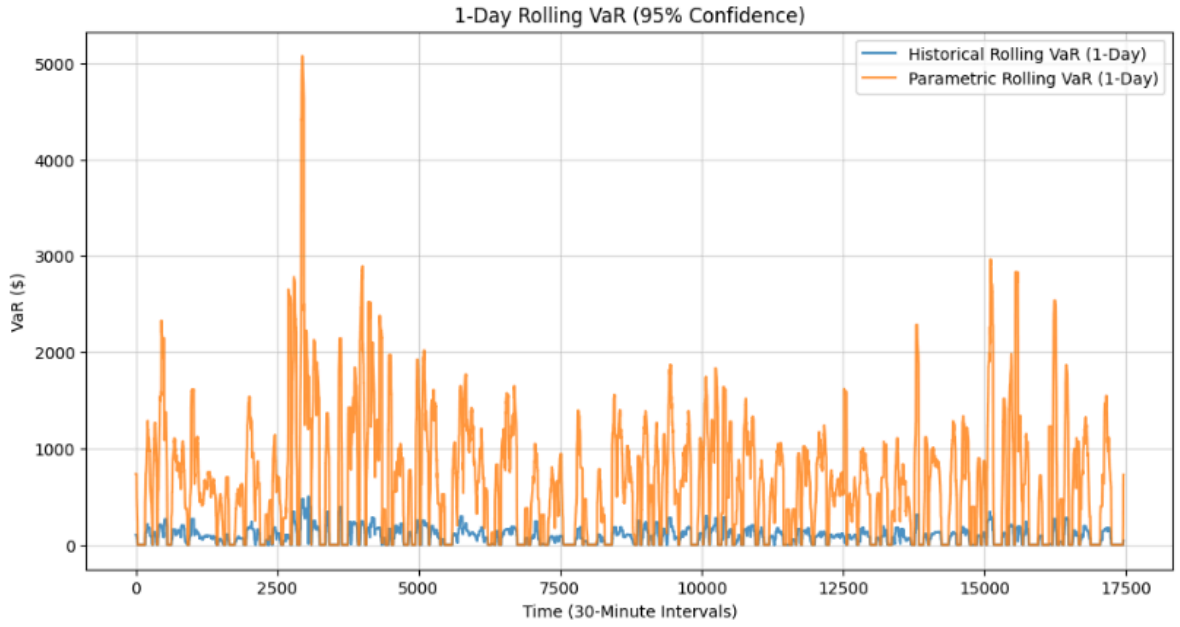
Using parametric VaR, the maximum loss that can be expected to occur over one day, to a 95% confidence level is $940.97 and $1,330.80 to a 99% level. As discussed, this can be scaled from one day to ten days:

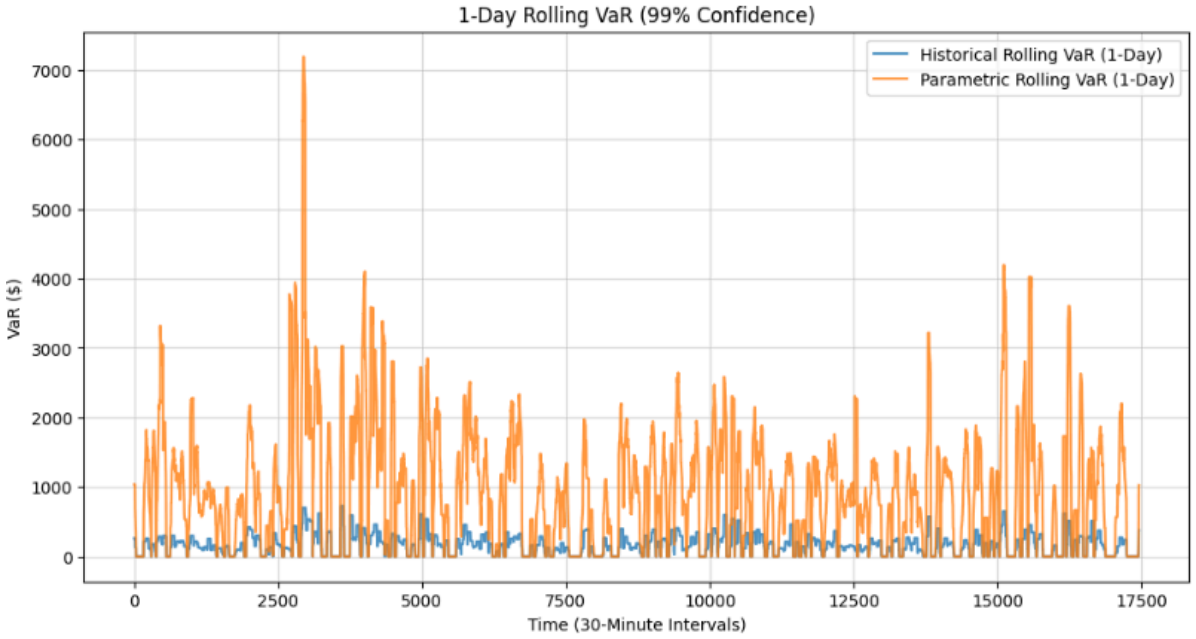
```
# scale for 10 days
tend_var_historical_95 = oned_var_historical_95 * np.sqrt(10)
tend_var_parametric_95 = oned_var_parametric_95 * np.sqrt(10)

tend_var_historical_99 = oned_var_historical_99 * np.sqrt(10)
tend_var_parametric_99 = oned_var_parametric_99 * np.sqrt(10)

print(f"Historical 10-day VaR (95% confidence): ${tend_var_historical_95:,.2f}")
print(f"Parametric 10-day VaR (95% confidence): ${tend_var_parametric_95:,.2f}")
print("\n")
print(f"Historical 10-day VaR (99% confidence): ${tend_var_historical_99:,.2f}")
print(f"Parametric 10-day VaR (99% confidence): ${tend_var_parametric_99:,.2f}")
```

```
Historical 10-day VaR (95% confidence): $354.81
Parametric 10-day VaR (95% confidence): $2,975.62


Historical 10-day VaR (99% confidence): $799.21
Parametric 10-day VaR (99% confidence): $4,208.36
```

At 10 days these figures increase to $2,2975.62 and $4.208.36 respectively. These calculations are based on an initial equity value of $1,000,000.

Rolling VaR is a dynamic calculation of VaR over a moving time window. Rolling VaR evaluates risk continuously across a time series, allowing a trader to monitor how risk is evolving over time. A rolling 1-day VaR can be calculated using the previous 48 30-minute intervals to calculate a 1-day VaR figure.The rolling 1-day parametric and historical VaR can be seen below for a $1,000,000 portfolio:





The parametric VaR can be seen to be much greater than the historical VaR figure throughout the year.

Conclusion

The exploration of mean reversion and trend following strategies in cryptocurrency markets reveals valuable insights into the performances, limitations and potential for improvement. The mean reversion strategy using EMAs and RSI performed well in markets with low or no trend and the use of stop loss and take profit levels ensure disciplined trade exits. The strategy is simple, and straightforward to implement, relying on well-established and documented technical indicators. However, as expected the mean reversion strategy had poor performance in trending markets. Markets exhibiting strong directional trends led to false signals being generated. The mean reversion strategy also had strict trade entry levels. The reliance on oversold/overbought market conditions resulted in fewer trading opportunities, and much less trades overall compared to the trend following strategy.

The trend following strategy, using MACD and ADX indicators, offered potential for significant gains during trending periods. The use of ADX filtered out weak trends, enhancing the reliability of trade signals and reducing false positives. This trend following strategy has potential to be applied across a range of markets. However, the strategy performed poorly in range-bound markets, where whipsaws led to frequent stop-outs and reduced overall profitability. Despite identifying profitable trends, the overall win rate was low. This reflects the challenges of trend following in volatile markets such as cryptocurrencies.

There are multiple limitations when performing historical analysis such as this. Data quality and frequency is a major limitation faced by so-called “retail traders”. 30-minute interval data is sufficient for most backtesting, but it may not capture microstructure dynamics critical for short-term strategies. Tick data, used by high frequency trading firms, is extremely expensive to purchase, as well as requiring great computational power to be able to extract insights from. Time series tools such as KDB+ or OneTick would be required to handle such data instead of python alone. The parameter optimisation was used to enhance historical performance, but it may lead to overfitting and an overall limitation to the strategy’s robustness in unseen data. A better approach at splitting data into training and test data should be implemented to improve the trading strategies.

A major assumption made as required for the project is the assumptions of mean reversion for the asset BCHUSD. It was clearly shown that the BCHUSD prices for 2023 were non-stationary, they were not mean reverting. However, for the purpose of the assignment, stationarity was assumed. Transaction costs and slippage could also have been better estimated. The impact of trading feeds, slippage and liquidity constraints was not fully implemented in the backtesting, potentially overestimating the real-world performance.

In order to improve the trading strategies, one could look at diversification across assets. Applying the strategies of a diversified portfolio of cryptocurrencies or other asset classes has the potential to improve risk adjusted returns and reduce strategy-specific vulnerabilities. Additionally, machine learning techniques could be used to dynamically adjust the strategy parameters based on changing market conditions. The parameters (e.g. EMA window length) were static from the beginning of the 2024 bactest until the end.

In conclusion, while both mean reversion and trend following strategies demonstrated unique strengths and limitations, their complementary nature suggests potential for integration. With continued refinement and adaptation to market conditions, these strategies can form the foundation for robust and scalable trading systems in volatile and dynamic markets like cryptocurrencies.

References

Wilmott, P., 2006. Paul Wilmott on quantitative finance. Vol. 1. 2nd ed. Chichester: John Wiley & Sons.

Methods Implemented

Below is a table listing all methods implemented in python, and in which submitted file they can be found:

Method	**File(s)**	**Comment**
Simple Moving Average (SMA)	P1BTechIndicatorsPlots.ipynb, P1Ftechinical_indicators.py	
Exponential Moving Average (EMA)	P1BTechIndicatorsPlots.ipynb, P1Ftechinical_indicators.py	
Rolling Standard Deviation	P1BTechIndicatorsPlots.ipynb, P1Ftechinical_indicators.py	
Relative Strength Index (RSI)	P1BTechIndicatorsPlots.ipynb, P1Ftechinical_indicators.py	
Average Directional Index (ADX)	P1BTechIndicatorsPlots.ipynb, P1Ftechinical_indicators.py	
Moving Average Convergence Divergence (MACD)	P1BTechIndicatorsPlots.ipynb, P1Ftechinical_indicators.py	
Augmented Dickey Fuller Test	P1CMeanReversion.ipynb	Judged to be too involved to write in python, adfuller method used from statsmodels.tsa.stattools
Sharpe Ratio	P3TradesAnalysis.ipynb	
Value at Risk (VaR)	P3TradesAnalysis.ipynb	

Appendix

Multiple files have been submitted with the final report. Below provides an explanation for what each file contains:

- P1AGetData.ipynb
  - The python notebook used to query data from Alpaca's API, format the data and save into CSV files
- P1BTechIndicatorsPlots.ipynb
  - Python notebook used to showcase examples of the technical indicators used in the project
- P1CMeanReversion.ipynb
  - Python notebook used to test for stationarity in the cryptocurrency prices
- P1DAnalyseOptimisedBacktests.ipynb
  - Python notebook used to analyse the optimised backtests results
- P1EStrategies.py
  - Python module containing the strategies used in the back testing performed using the Backtesting-py package
- P1E2backtesting_strats.py
  - Script used to perform the backtesting and optimisations
- P1Ftechnical_indicators.py
  - Python module used to store functions used to calculate the technical indicators
- BCHUSD_2023.csv
  - OHLC bar data in 5-minute intervals from 01/07/2023 -> 31/12/2023 for BCHUSD, sourced from Alpaca's API
- ETHUSD_2023.csv
  - OHLC bar data in 5-minute intervals from 01/07/2023 -> 31/12/2023 for ETHUSD, sourced from Alpaca's API
- USDTUSD_2023.csv
  - OHLC bar data in 5-minute intervals from 01/07/2023 -> 31/12/2023 for USDTUSD, sourced from Alpaca's API
- BHCUSD_2024.csv
  - OHLC bar data in 5-minute intervals for the full year of 2024, sourced from Alpaca's API
- ETHUSD_2024.csv
  - OHLC bar data in 5-minute intervals for the full year of 2024,sourced from Alpaca's API
- USDTUSD_2024.csv
  - OHLC bar data in 5-minute intervals for the full year of 2024, sourced from Alpaca's API

- DTB3.csv
    - Daily 3-month US Treasury Bill rates for 2024, sourced from FRED
- opt_results_2023-7-1_2023-12-31.csv
    - CSV file containing the results from the backtesting optimisations
- P2Amean_reversion_live_trading.py
    - Script to implement live trading of strategy using EMA Crossovers and Relative Strength Index with Alpaca's API
- P2Btrend_follow_live_trading.py
    - Script to implement live trading of strategy using MACD Crossovers and Average Directional Index with Alpaca's API
- P2Cmean_reversion_logs.log
    - Log files for the four hour period of live trading for the mean reversion strategy
- P2Dtrend_follow_logs.log
    - Log files for the four hour period of live trading for the trend following strategy
- P3TradesAnalysis.ipynb
    - Python notebook used to analyse the performance of the trading strategies on unseen 2024 data