# CQF Final Project

Viranca Balsingh

In partial fulfilment of the Certificate in Quantitative Finance
As offered by Fitch Learning under the supervision of Dr. Paul Wilmott
and guidance of Kannan Singaravelu, CQF

January 14, 2025

# Contents

# 1 Introduction

Algorithmic trading has revolutionized financial markets, enabling rapid decision-making and trade execution based on data-driven strategies. As markets become increasingly complex, the ability to leverage quantitative methods for identifying and capitalizing on market opportunities is more critical than ever. This project delves into the design, implementation, and evaluation of two widely recognized trading strategies: *trend-following* and *mean-reversion*.

The *trend-following strategy* seeks to capitalize on sustained price movements by identifying and acting on bullish or bearish trends using tools such as Exponential Moving Averages (EMAs) and the Average Directional Index (ADX). Conversely, the *mean-reversion strategy* exploits statistical deviations from historical norms, leveraging rolling z-scores to identify potential opportunities for price correction.

Driven by a personal passion for algorithmic trading, this project not only serves as a foundation for developing and testing systematic strategies but also aligns with a long-term goal of creating a dedicated gym environment for training reinforcement learning (RL) agents. Such an environment would provide a controlled yet flexible platform for exploring dynamic decision-making, optimization techniques, and adaptive trading strategies in realistic market conditions.

This study emphasizes robust data handling, accurate signal generation, and real-time trading execution. A TimescaleDB database and Python-based scripts integrate with Alpaca's broker API to simulate live trading scenarios. Rigorous backtesting forms the foundation of this project, ensuring the strategies' viability while highlighting areas for improvement.

The datasets employed include minute-level and daily-level stock data sourced from Yahoo Finance, Alpaca, and Interactive Brokers. By combining insights from these diverse data sources, the project ensures data quality and comprehensive market coverage.

To facilitate reproducibility and further development, all code is thoroughly documented with detailed instructions on functionality and execution. This project aims to bridge the gap between theoretical quantitative finance and practical trading applications, providing a robust framework for future enhancements, including the development of a gym environment for RL training.

# 2 Part one: Generic Strategies Made Proprietary

This chapter describes part one of the project. In part one, the foundation is built towards a full fletched algorihmic trading system. The core components described in this chapter are the mathematical and numerical methods, the implementation and some preliminary results.

Table 2.1: Summary of Numerical Methods Used

| **Numerical Method** | **Description and Implementation Details** |
|---|---|
| **Exponential Moving Average (EMA)** | Calculated using the formula: $$\text{EMA}_t = \alpha \cdot x_t + (1 - \alpha) \cdot \text{EMA}_{t-1}$$ where $\alpha = \frac{2}{N+1}$. EMA smooths price data and detects trends, implemented in Python using a rolling window. |
| **Average Directional Index (ADX)** | Measures trend strength by combining True Range (TR) and Directional Movement (DM), smoothed over a rolling window. ADX helps filter out weak trends and is implemented in Python with Pandas. |
| **Rolling Z-Score** | Calculates the standard deviation-adjusted deviation of data from the mean: $$Z_t = \frac{x_t - \mu_t}{\sigma_t}$$ Used to generate mean-reversion signals. |
| **Backtesting Framework** | Simulates historical trades based on signals, tracking portfolio value, drawdowns, and risk-adjusted returns. Implemented in Python using Pandas. |
| **Signal Generation Logic** | Generates trading signals based on Z-scores ($\pm 2$) for mean-reversion and EMA crossovers with ADX thresholds for trend-following. |
| **Exception Handling** | Manages errors during API calls and data inconsistencies. Includes retry logic and logging for debugging. Implemented in Python. |

## 2.1 Mathematical Models and Numerical Methods

### 2.1.1 Exponential Moving Average (EMA)

The Exponential Moving Average (EMA) is a weighted moving average that gives more significance to recent data points, making it more responsive to price changes. For a rolling ticker window, it dynamically updates as new data enters the window.

$$\text{EMA}_t = \alpha \cdot x_t + (1 - \alpha) \cdot \text{EMA}_{t-1}$$

where:

- $x_t$: Current data point
- $\text{EMA}_{t-1}$: Previous EMA
- $\alpha = \frac{2}{N+1}$: Smoothing factor ($N$ is the window size)
- **Weighted Average**: The Exponential Moving Average (EMA) assigns more importance to recent data points compared to older ones, ensuring that the calculation reflects the most current trends.
- **Dynamic Adjustment**: As new data points are added to the rolling window, the EMA recalculates, continuously updating its value to account for the latest information.
- **Smoothing Effect**: The EMA reduces the impact of short-term fluctuations or noise in the data, allowing users to focus on the underlying trends and patterns over time.

### 2.1.2 Average Directional Index (ADX) on a Rolling Ticker Window

The Average Directional Index (ADX) is a technical indicator designed to measure the strength of a trend in a financial market. It is particularly useful in identifying whether a market is trending strongly or moving sideways. The ADX is calculated using a series of steps that involve the True Range (TR), Directional Movement (DM), and smoothed averages over a rolling window. Below is a detailed explanation of the key components and formulas:

#### True Range (TR)

The True Range represents the range of price movement within a specific period. It accounts for price gaps and is computed as the maximum of the following:

$$\text{TR}_t = \max\left(\text{High}_t - \text{Low}_t,\ \left|\text{High}_t - \text{Close}_{t-1}\right|,\ \left|\text{Low}_t - \text{Close}_{t-1}\right|\right)$$

This ensures the most extreme price movement is captured for the period.

### Directional Movement (DM)

Directional Movement quantifies the directional price movement between periods:

$$+DM_t = \begin{cases} \text{High}_t - \text{High}_{t-1}, & \text{if } (\text{High}_t - \text{High}_{t-1}) > (\text{Low}_{t-1} - \text{Low}_t) \text{ and } (\text{High}_t - \text{High}_{t-1}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$-DM_t = \begin{cases} \text{Low}_{t-1} - \text{Low}_t, & \text{if } (\text{Low}_{t-1} - \text{Low}_t) > (\text{High}_t - \text{High}_{t-1}) \text{ and } (\text{Low}_{t-1} - \text{Low}_t) > 0 \\ 0, & \text{otherwise} \end{cases}$$

These formulas help isolate whether upward or downward price movement dominates during the period.

### Smoothed Values

To reduce noise and capture trends more effectively, the True Range and Directional Movements are smoothed over a rolling window:

$$\text{Smoothed TR}_t = \frac{\sum_{i=t-n+1}^{t} \text{TR}_i}{n}$$

$$\text{Smoothed } +DM_t = \frac{\sum_{i=t-n+1}^{t} +DM_i}{n}, \quad \text{Smoothed } -DM_t = \frac{\sum_{i=t-n+1}^{t} -DM_i}{n}$$

### Directional Indicators (DI)

The Positive and Negative Directional Indicators ($+DI$ and $-DI$) are calculated as:

$$+DI_t = \frac{\text{Smoothed } +DM_t}{\text{Smoothed TR}_t} \times 100$$

$$-DI_t = \frac{\text{Smoothed } -DM_t}{\text{Smoothed TR}_t} \times 100$$

These indicators are expressed as percentages and measure the relative strength of upward or downward movement.

### Directional Movement Index (DX)

The DX measures the absolute difference between $+DI$ and $-DI$, normalized by their sum:

$$\text{DX}_t = \frac{|+DI_t - -DI_t|}{|+DI_t + -DI_t|} \times 100$$

This provides a non-biased view of the magnitude of directional movement.

### Average Directional Index (ADX)

Finally, the ADX is computed as the smoothed average of the DX values over the rolling window:

$$\mathrm{ADX}_t = \frac{\sum_{i=t-n+1}^{t} \mathrm{DX}_i}{n}$$

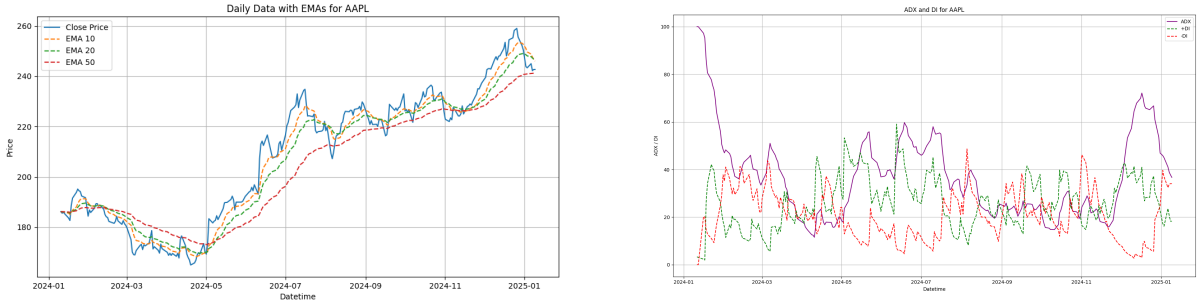
The ADX provides a single value to represent the overall trend strength, without indicating its direction.

### Interpretation of ADX

The ADX values can be interpreted as follows:

| | |
|---|---|
| $\mathrm{ADX}_t > 25$ | Indicates a strong trend is present. |
| $\mathrm{ADX}_t < 20$ | Suggests a weak trend or a non-trending market. |

Values between 20 and 25 may indicate the early stages of a trend formation or a transition from a strong trend to a weaker phase. By combining the ADX with other technical indicators or price action analysis, traders can identify optimal entry and exit points, adapt their strategies to market conditions, and manage risk effectively.




(a) Exponential Moving Average (EMA) (b) Average Directional Index (ADX)

Figure 2.1: EMA and ADX charts

### 2.1.3 Z-Score on a Rolling Ticker Window

The **z-score on a rolling ticker window** quantifies the position of a data point relative to the mean and standard deviation of a rolling subset of data. This measure is widely used in time series analysis, particularly in financial applications.
The z-score in the context of a financial ticker's closing price requires the dimensions:

- **Ticker:** The financial instrument or stock being analyzed.
- **Rolling Window:** A moving subset of time-series data over which statistics (mean and standard deviation) are calculated. The window advances with each new data point.
- **Mean and Standard Deviation:**

$$\mu_t = \text{Mean of data in the rolling window at time } t,$$
$$\sigma_t = \text{Standard deviation of data in the rolling window at time } t.$$

The z-score for a data point $x_t$ at time $t$ is given by:

$$Z_t = \frac{x_t - \mu_t}{\sigma_t}$$

where:

$$x_t = \text{Current value of the ticker at time } t,$$
$$\mu_t = \text{Rolling mean at time } t,$$
$$\sigma_t = \text{Rolling standard deviation at time } t.$$

A **positive z-score** indicates that the value is above the rolling mean, suggesting a relatively high data point compared to recent observations. A **negative z-score** indicates that the value is below the rolling mean, suggesting a relatively low data point compared to recent observations.The **magnitude** of the z-score shows how extreme the value is compared to its recent historical behavior.

## 2.2 Implementation

The codebase is designed for financial research, backtesting, and trading automation. It includes components for data initialization, strategy development, and trading execution, focusing on algorithmic trading strategies.

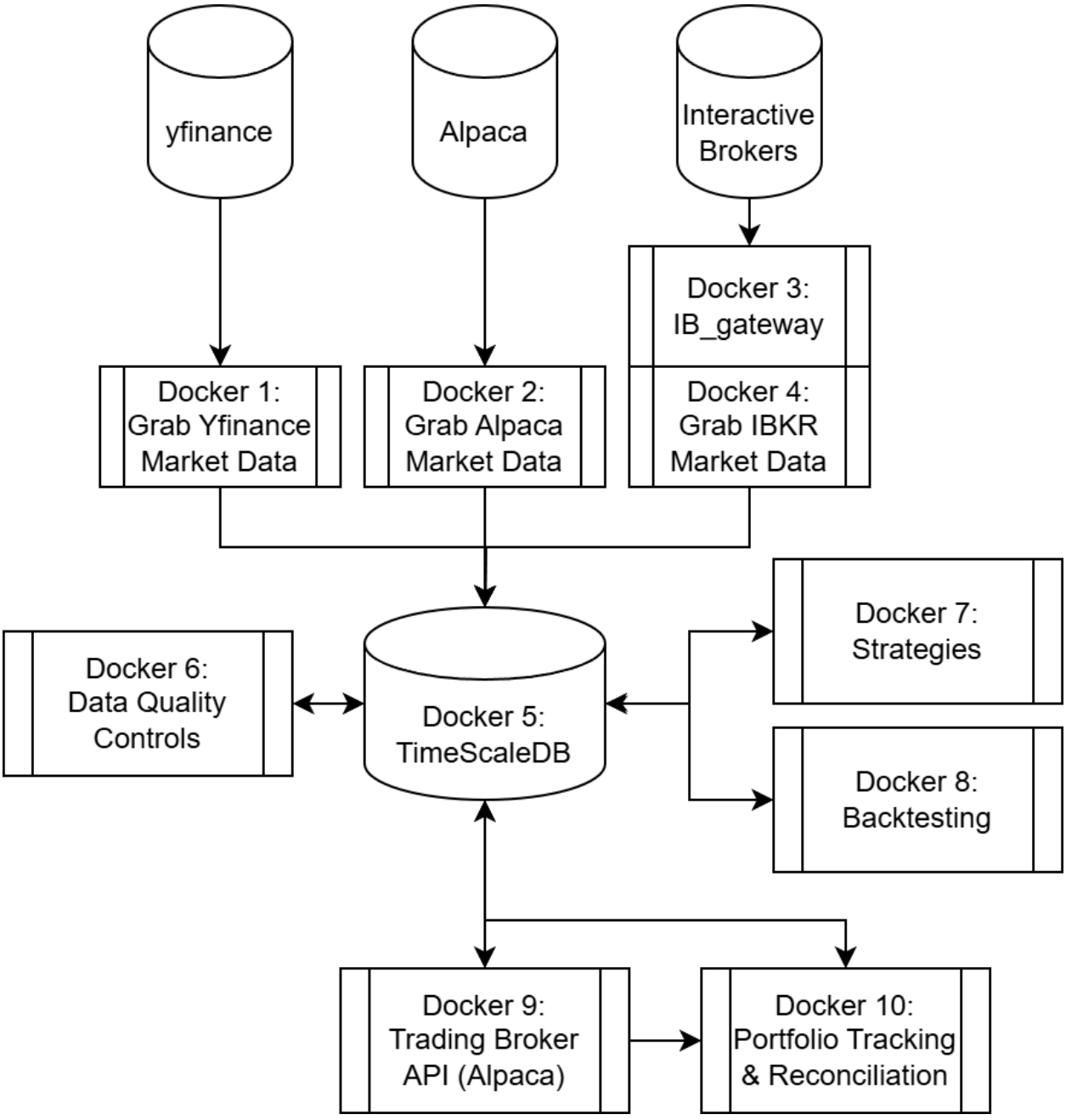



Figure 2.2: Overview of Data Flow and Processing

Table 2.2: Docker Components and their Names

| **Docker No.** | **Name** | **Run Frequency** |
|---|---|---|
| Docker 1 | Grab Yfinance Market Data | every minute |
| Docker 2 | Grab Alpaca Market Data | every minute |
| Docker 3 | IB_gateway | continuous |
| Docker 4 | Grab IBKR Market Data | every minute |
| Docker 5 | TimeScaleDB | continuous |
| Docker 6 | Data Quality Controls | every minute |
| Docker 7 | Strategies | every minute |
| Docker 8 | Backtesting | manual |
| Docker 9 | Trading Broker API (Alpaca) | every minute |
| Docker 10 | Portfolio Tracking & Reconciliation | every minute |

The TimeScale DB consists of the following tables:

Table 2.3: TimescaleDB Tables

| **Docker No.** | **Name** | **Update Frequency** |
|---|---|---|
| 1 | $yfinance\_minute$ | every minute |
| 2 | $yfinance\_daily$ | every day |
| 3 | $ibkr\_minute$ | every minute |
| 4 | $ibkr\_daily$ | every day |
| 5 | $alpaca\_minute$ | every minute |
| 6 | $alpaca\_daily$ | every day |
| 7 | $data\_quality$ | every minute |
| 8 | $ticker\_minute\_indicators$ | every minute |
| 9 | $ticker\_daily\_indicators$ | every day |
| 10 | $ticker\_minute\_signals$ | every minute |
| 11 | $ticker\_daily\_signals$ | every day |
| 12 | $backtest\_performance$ | manual |
| 13 | $Trading\ Broker\ API\ (Alpaca)$ | every minute |
| 14 | $portfolio\_orders$ | every minute |

The codebase consists of three main folders:

1. **Research**
   The research folder is a comprehensive repository containing scripts and tools designed for financial data analysis and strategy development. Data initialization scripts facilitate the retrieval of financial data from multiple APIs, including Alpaca, Interactive Brokers (IBKR) and Yahoo Finance (yfinance). Each data source has its own pros and cons. Multiple are used and compared to ensure high quality data. The strategies section contains implementations of trading methodologies, including Mean Reversion, which identifies and exploits statistical price deviations using rolling z scores, and Trend Following, which monitors directional price movements using tools such as the Average

Directional Index (ADX) and Exponential Moving Averages (EMA). Data quality is maintained using scripts designed to detect, log and rectify anomalies such as missing data points, outliers, and inconsistencies in time series data, ensuring the integrity and reliability of the dataset. The database management tools provide seamless interaction with Timescale Db, which is a PostgreSQL evolution optimized for time series.

2. **Trading**
   The trading folder contains scripts specifically designed for live or paper trade execution, focusing on the precise placement and management of trades at a minute-level granularity. These scripts facilitate real-time interaction with brokerage platforms, enabling the system to monitor the portfolio continuously. This includes tracking open positions, managing order executions, and handling events such as trade confirmations, updates, and error responses. The functionality ensures that trades are executed efficiently and accurately, while also providing the ability for position tracking and making sure orders are executed as expected.

3. **test_scripts** Finally, this folder contains test scripts for validating:

   - Data pipelines for Alpaca, IBKR, and Yahoo Finance.
   - Trading signals and indicators.
   - Strategies and execution logic.

## 2.3 Signal Generation

### 2.3.1 Mean Reversion Signal Generation

The function `generate_mean_reversion_signals` generates trading signals based on the mean reversion principle using the rolling z-score of the ticker time series. This function identifies extreme deviations from the mean to suggest buy or sell actions. The algorithm can be summarized as follows:

Listing 2.1: Mean Reversion Signal Generation

```
def generate_mean_reversion_signals(df, window=10):
    """
    Generate mean reversion signals based on rolling z-score.
    """
    df['signal'] = 'neutral'

    # Convert 'close' to float if it's of type Decimal
    if df['close'].dtype == 'object' or isinstance(df['close'].iloc[0], Decimal):
        df['close'] = df['close'].astype(float)

    # Calculate rolling z-score using the imported function
    df = calculate_rolling_z_score(df, window=window)
```

```
    # Signal conditions
    buy_signal = df['z_score'] < -2
    sell_signal = df['z_score'] > 2

    df.loc[buy_signal, 'signal'] = 'buy'
    df.loc[sell_signal, 'signal'] = 'sell'

    return df
```

## Steps Involved

1. **Initialization:** A new column `signal` is initialized to 'neutral' to represent no trading action.
2. **Data Type Conversion:** If the `close` column contains data as strings or `Decimal` objects, it is converted to float to ensure proper numerical calculations.
3. **Rolling Z-Score Calculation:** The rolling z-score is computed over a specified window size (default is 10). This uses the `calculate_rolling_z_score` function.
4. **Signal Conditions:**
   - A `buy` signal is triggered when the z-score is less than $-2$, indicating the price is significantly below the mean.
   - A `sell` signal is triggered when the z-score is greater than 2, indicating the price is significantly above the mean.
5. **Signal Assignment:** The `signal` column is updated based on these conditions.

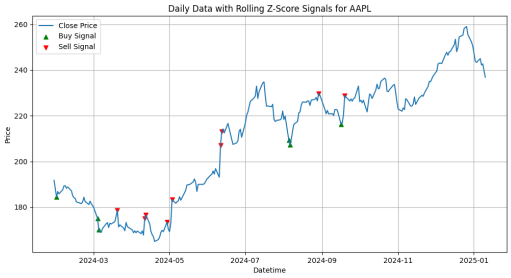
The choice of 2 and $-2$ as thresholds in the z-score-based mean reversion strategy is grounded in statistical principles and practical considerations. The z-score measures the number of standard deviations a data point deviates from the mean, and in a normal distribution, approximately 95.4% of data lies within $\pm 2$ standard deviations. Values beyond $\pm 2$ are relatively rare, falling within the extreme 4.6% of cases, making them statistically significant outliers. By selecting these thresholds, the strategy focuses on identifying substantial deviations from the mean, where the likelihood of mean reversion is higher. This ensures the signals generated are meaningful and not influenced by minor fluctuations or noise.
In financial markets, mean-reverting behavior is often observed, where prices that deviate significantly from their average tend to revert over time. The thresholds of 2 and $-2$ help capture such behavior by targeting scenarios where the deviation is strong enough to suggest a potential correction. At the same time, these thresholds balance the trade-off between signal frequency and reliability. Larger thresholds, such as $\pm 3$, would generate fewer signals, targeting only the most extreme cases, but could miss profitable opportunities. On the other hand, smaller thresholds, such as $\pm 1$, would generate more frequent signals, increasing the likelihood of false positives caused by noise. The chosen thresholds strike a balance by providing a reasonable frequency of signals while maintaining high reliability.

## Output

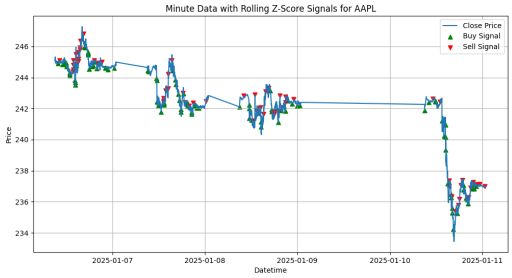
The function returns the input DataFrame `df` with the updated `signal` column, which can have the values:

- `neutral`: No significant deviation from the mean.
- `buy`: Price is significantly below the mean, suggesting a buying opportunity.
- `sell`: Price is significantly above the mean, suggesting a selling opportunity.

The below charts illustrate price data and mean-reversion trading signals for AAPL stock across different timeframes. Figure 2.3a shows daily data for 2024, with buy signals represented as green triangles and sell signals as red triangles. These signals are based on rolling Z-scores, suggesting potential undervaluation or overvaluation for mean reversion. Figure 2.3b uses minute-level data, highlighting frequent buy and sell signals reflecting a short-term trading strategy. Figures 2.4a and 2.4b zoom into specific time windows of minute-level data, offering detailed views of signal behavior during varying market conditions. These figures collectively demonstrate the applicability of a mean-reversion strategy across daily and intraday timeframes. The performance of these signals is further elaborated in a later Chapter.

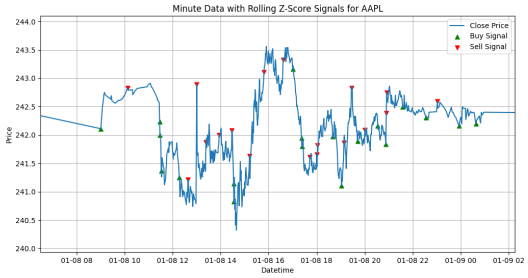



(a) AAPL and Z-Score Signal for Daily ticker data

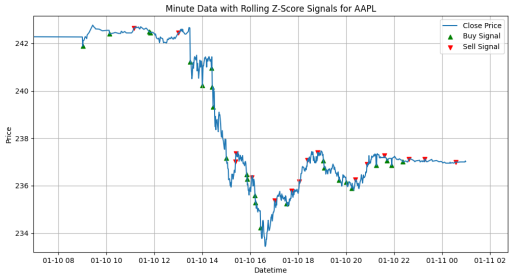



(b) AAPL and Z-Score Signal for Minute ticker data (4 days)

Figure 2.3: AAPL Z-Score Signals for Daily and Minute Ticker Data.




(a) AAPL and Z-Score Signal for Minute ticker data (1 day)




(b) AAPL and Z-Score Signal for Minute ticker data (1 day, zoomed)

Figure 2.4: AAPL Z-Score Signals for Minute Ticker Data (1 Day and Zoomed View).

# Trend Following Signal Generation

The function `generate_trend_signals` is designed to identify trend-following signals in financial time-series data. It evaluates conditions based on Exponential Moving Averages (EMAs) and the Average Directional Index (ADX) to classify the trend into one of three categories: **Uptrend**, indicating strong bullish momentum; **Downtrend**, indicating strong bearish momentum; or **Neutral**, indicating no significant trend.

Listing 2.2: Trend Following Signal Generation

```
def generate_trend_signals(df):
    df['trend'] = 'neutral'
    adx_threshold = 25

    uptrend = (df['ema_10'] > df['ema_20']) & \
              (df['ema_20'] > df['ema_50']) & \
              (df['close'] > df['ema_10']) & \
              (df['adx'] > adx_threshold)

    downtrend = (df['ema_10'] < df['ema_20']) & \
                (df['ema_20'] < df['ema_50']) & \
                (df['close'] < df['ema_10']) & \
                (df['adx'] > adx_threshold)

    df.loc[uptrend, 'trend'] = 'uptrend'
    df.loc[downtrend, 'trend'] = 'downtrend'

    return df
```

# Steps Involved

The logic and implementation follow these steps:

1. **Initialize the Trend Column:** Add a new column, `trend`, to the dataset and set its default value to `'neutral'`.

2. **Define the ADX Threshold:** Set the ADX threshold for a strong trend as:

$$\texttt{adx\_threshold} = 25$$

3. **Evaluate Uptrend Conditions:** A row is classified as an `'uptrend'` if:

$$\texttt{ema\_10} > \texttt{ema\_20}, \quad \texttt{ema\_20} > \texttt{ema\_50}, \quad \texttt{close} > \texttt{ema\_10}, \quad \texttt{adx} > \texttt{adx\_threshold}$$

4. **Evaluate Downtrend Conditions:** A row is classified as a `'downtrend'` if:

$$\texttt{ema\_10} < \texttt{ema\_20}, \quad \texttt{ema\_20} < \texttt{ema\_50}, \quad \texttt{close} < \texttt{ema\_10}, \quad \texttt{adx} > \texttt{adx\_threshold}$$

5. **Update the Trend Column:** Assign `'uptrend'` or `'downtrend'` based on the conditions, leaving rows that meet neither condition as `'neutral'`.

The choice of the EMA windows, specifically 10, 20, and 50, is motivated by the need to capture market trends across different time horizons. The 10-period EMA is designed to reflect short-term price movements and is highly responsive to recent changes. This responsiveness makes it ideal for identifying the initial signs of trend reversals or continuations. In contrast, the 20-period EMA provides a balanced view by smoothing out more noise than the 10-period EMA, while still being responsive enough to detect emerging trends. It acts as an intermediary between short-term and long-term perspectives. The 50-period EMA, on the other hand, is used to capture long-term trends. By focusing on a larger timeframe, it reduces the impact of short-term fluctuations and provides a clear indication of the primary trend direction. Combining these three EMAs allows for the analysis of market behavior across multiple timeframes, with the relationships between them—such as the faster EMAs crossing above or below the slower ones—serving as strong indicators of bullish or bearish trends.

The ADX, or Average Directional Index, is used to measure the strength of a trend irrespective of its direction. The threshold of 25 is a standard value in technical analysis to differentiate between strong and weak trends. When the ADX exceeds 25, it signals a strong trend, making the market conditions suitable for trend-following strategies. Conversely, when the ADX is below 25, the market is considered to be in a weak or range-bound state, where such strategies are less effective. The choice of 25 strikes a balance between avoiding false signals, which are more likely if the threshold is set too low, and minimizing missed opportunities, which can occur if the threshold is set too high.

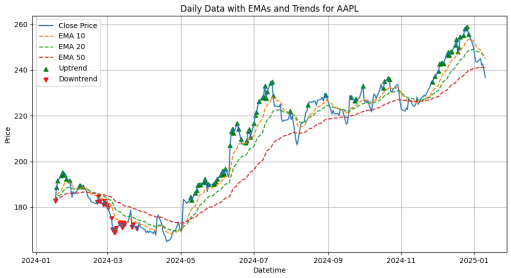
Together, the EMA windows and the ADX threshold create a robust framework for identifying and acting on strong market trends. The EMA crossovers provide insight into the direction of the trend, while the ADX ensures that only trends with significant strength are considered, thus reducing the likelihood of false signals in choppy or range-bound markets.

# Output

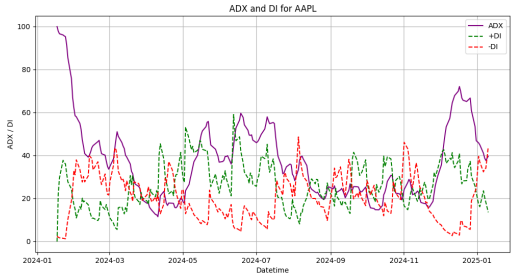
The function outputs the modified dataset with an additional column, `trend`, containing one of the following values for each row: `'uptrend'`, indicating a strong bullish trend; `'downtrend'`, indicating a strong bearish trend; or `'neutral'`, indicating no significant trend.

The below charts illustrate trend-following signals for AAPL, both on daily and minute-level data. For daily data, one set of charts presents the close price along with exponential moving averages (EMAs) of 10, 20, and 50 periods, highlighting uptrends and downtrends (Figure 2.5a). Another set of charts uses the Average Directional Index (ADX) to showcase the strength of these trends (Figure 2.5b). Similarly, minute-level data for a 4-day (Figures 2.6a and 2.7a) and 1-day window (Figures 2.6b and 2.7b) is analyzed. The EMA

indicators and trend markers provide insights into price dynamics, while the ADX charts highlight trend strength.

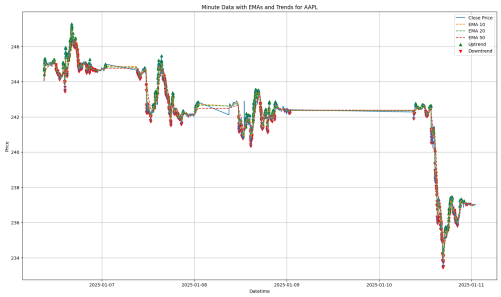



(a) AAPL and EMA Signal for Daily ticker data

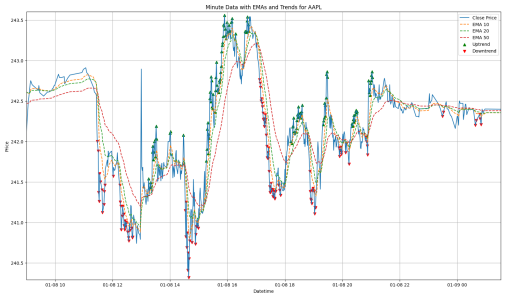



(b) AAPL and ADX Signal for Daily ticker data

Figure 2.5: AAPL EMA and ADX Signals for Daily Ticker Data.

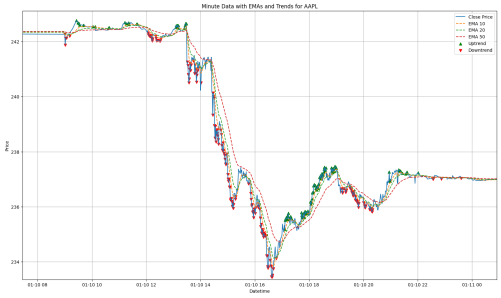


(a) AAPL and EMA Signal for minute ticker data (4 days)

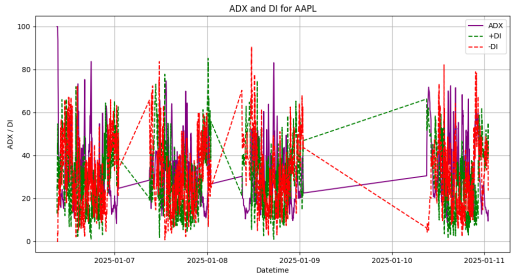


(b) AAPL and EMA Signal for minute ticker data (1 day)

Figure 2.6: AAPL EMA Signals for Minute Ticker Data (4 Days and 1 Day).



(a) AAPL and ADX Signal for minute ticker data (4 days)




(b) AAPL and ADX Signal for minute ticker data (1 day)

Figure 2.7: AAPL ADX Signals for Minute Ticker Data (4 Days and 1 Day).

# 3 Part Two: Broker API

## 3.1 System Configuration

The live trading system is configured with a PostgreSQL database that stores trading signals and portfolio updates. Signals are fetched from a dedicated table called `ticker_minute_signals`, which contains trading instructions such as `buy` or `sell`, associated stock symbols, and trends. The portfolio updates are logged in another table called `portfolio_orders`, which records details of all executed orders.
For executing trades, the system leverages the Alpaca API. A paper account is used for testing purposes, with the API endpoint set to `https://paper-api.alpaca.markets`. This allows testing without risking real capital while ensuring realistic simulation of market conditions.

- **Database**: PostgreSQL is used for signal storage and portfolio updates.
- **Broker API**: The Alpaca API is utilized for real-time trading. A paper account (`https://paper-api.alpaca.markets`) is used for testing purposes.
- **Signal Table**: `ticker_minute_signals` table stores trading signals (`buy`/`sell`) along with associated stock symbols and trends.
- **Portfolio Table**: `portfolio_orders` logs all executed orders for auditing and performance analysis.

## 3.2 Description of Python Scripts

The system is built using Python scripts that integrate the database and broker API to automate the trading workflow. Below is a description of each script:

### 3.2.1 `fetch_minute_signal_table`

This script connects to the PostgreSQL database to retrieve trading signals from the `ticker_minute_signals` table. The signals, such as `buy`, `sell`, and market trends (e.g., `uptrend`, `downtrend`), are extracted and converted into a Pandas DataFrame for further analysis. The script uses `psycopg2` to connect to the database, ensures proper handling of database errors and resource cleanup, and structures the raw query results into a format suitable for downstream processing.

### 3.2.2 `place_orders`

This script processes the trading signals fetched earlier to place orders through the Alpaca API. Depending on the signal type and trend, it submits either `buy` or `sell` market orders and records the transaction in the `portfolio_orders` table. It leverages the Alpaca REST

API to execute trades, logs all executed orders into the database for traceability, implements error handling for API and database interactions to ensure reliability, and supports dynamic signal evaluation, such as combining signals with market trends.

### 3.2.3 `validate_positions`

This script validates the alignment between open positions in the Alpaca account and recorded portfolio orders in the PostgreSQL database. It aggregates portfolio orders to calculate net quantities for each symbol and compares these values with actual positions retrieved from the Alpaca API. The script queries the `portfolio_orders` table to compute net quantities for each symbol, fetches open positions using the Alpaca API, reports any discrepancies between the database and live positions for manual investigation, and ensures data integrity across the trading system.

## 3.3 Real-Time Trading Challenges

Real-time trading introduces several challenges that must be addressed to ensure accurate and reliable operation. One significant challenge is latency. Market conditions can change rapidly, and delays in fetching signals, processing data, and executing orders can lead to missed opportunities or outdated trades. Another critical issue is slippage, which occurs when there is a difference between the expected price based on signals and the actual price at the time of trade execution. Slippage can significantly impact the profitability of trading strategies.

The Alpaca API enforces rate limits, which presents another challenge. If the system exceeds these limits, order placement may be delayed, potentially missing key trading opportunities. Additionally, error handling becomes critical in a real-time environment. Failures in API calls or database interactions must be managed carefully to prevent data corruption or missed trades. Lastly, network reliability is crucial, as disruptions during trade execution could result in inconsistent portfolio states or unexecuted trades.

## 3.4 Example Live-Trade Outputs

To illustrate live trading outputs, consider the following examples. A successful order placement might result in the following output:

**Successful Order:**

```
Placed buy order for AAPL at 145.30
Inserted record into portfolio table: AAPL, buy, 145.30
```

On the other hand, an API error could result in the following message:
**API Error:**

```
Failed to place sell order for TSLA: Order rejected due to insufficient buying
    power
```

Similarly, a database error while fetching signals might produce this output:

**Database Error:**

```
Error fetching minute signal table from PostgreSQL database: connection timeout
```

## 3.5 Exception Handling Mechanisms

Robust exception handling mechanisms are implemented to address potential issues during live trading. For database operations, all queries are wrapped in `try-except` blocks to log errors and allow fallback actions. Connections and cursors are properly closed in `finally` blocks to ensure no resources are left open.
For order placement, API exceptions, such as `tradeapi.rest.APIError`, are caught and logged. This allows capturing issues like rejected orders or invalid inputs. To handle transient failures such as network issues or temporary API unavailability, retry logic is implemented. This ensures that minor disruptions do not halt the trading process.
To address API rate limits, delays are introduced between API calls, or orders are batched when the usage approaches the limit. Input validation is also performed to ensure that signals and trends are valid before executing trades. This reduces the likelihood of placing erroneous orders due to incorrect or incomplete data.

# 4 Part Three: Evaluate Risk and Test Thrice Trend-Following Strategy

The trend-following strategy is designed to capture sustained price movements by generating buy or sell signals based on market trends. A buy signal is triggered during an uptrend, while a sell signal is activated during a downtrend. The strategy aims to exploit directional trends while minimizing exposure during flat or choppy markets.
In the backtest, the trend-following strategy achieved a final portfolio value of $9785.20, corresponding to a total return of -2.15%. The maximum drawdown was 0.05%, indicating limited risk exposure. A total of 195 trades were executed, with a win rate of 41.54%. This higher win rate compared to typical trend-following strategies suggests better signal accuracy but may also indicate suboptimal profit-per-trade, as the overall return remained negative. The Sharpe ratio hovered near zero, highlighting inadequate risk-adjusted returns. The strategy maintained minimal drawdowns, suggesting effective risk management but insufficient performance during favorable market conditions.

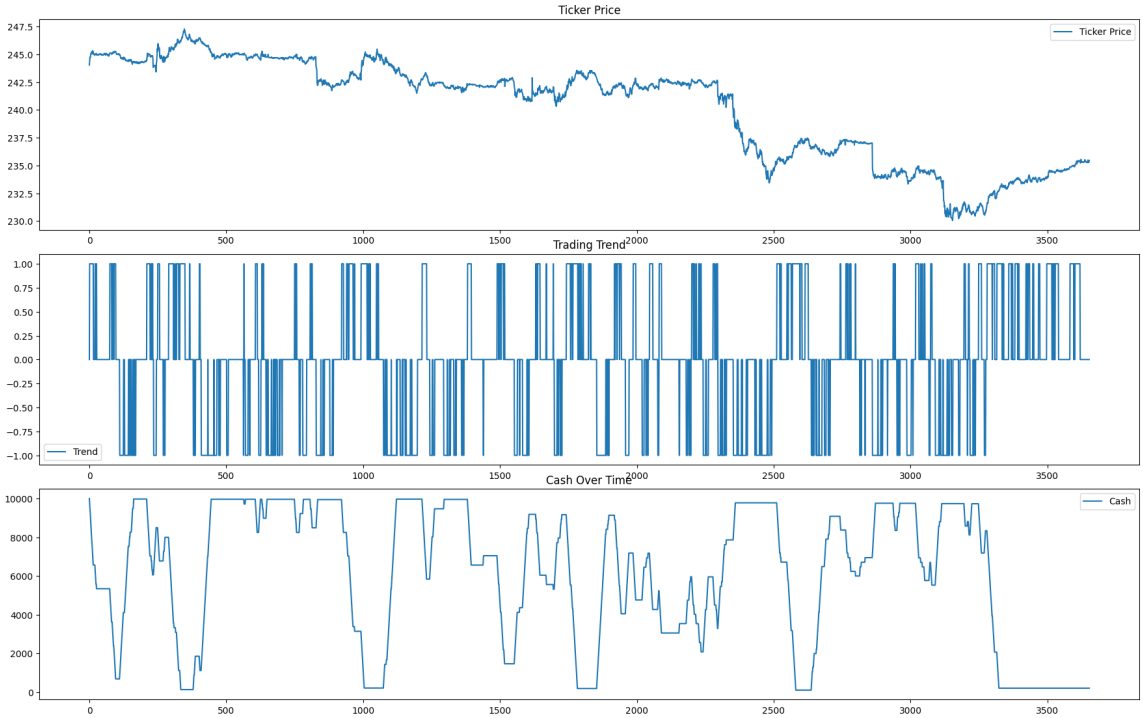



Figure 4.1: Trend-Following: Price, Signals, and Cash Over Time.

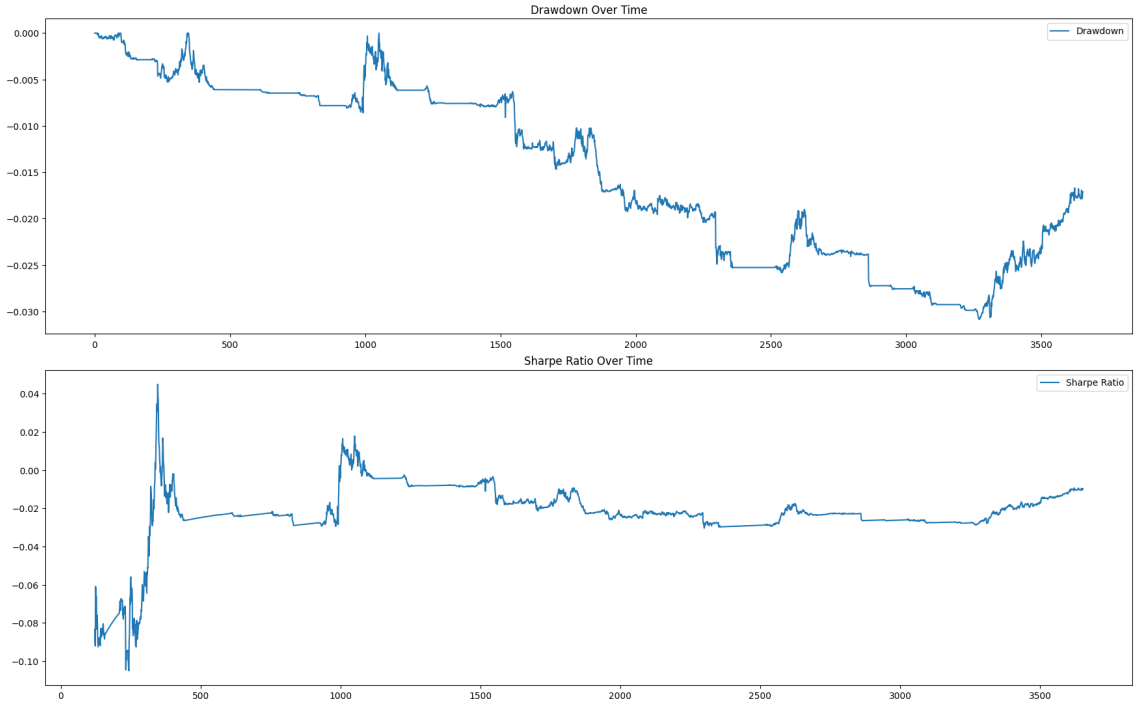



Figure 4.2: Trend-Following: Drawdown and Sharpe Ratio Over Time.

Figures 4.1 and 4.2 illustrate the price movements, trading signals, cash balance, drawdowns, and Sharpe ratio over time.

# Mean-Reversion Strategy

The mean-reversion strategy is based on the assumption that prices tend to revert to their historical mean. This strategy generates a buy signal when prices deviate significantly below the mean and a sell signal when prices exceed it. The goal is to exploit short-term price anomalies while maintaining market neutrality over the long term.

The backtest results indicate a final portfolio value of $9785.20, with a total return of -1.38%. The strategy experienced a maximum drawdown of 0.03% and executed 925 trades. The win rate was 15.46%, which is significantly lower than trend-following. This low win rate suggests the strategy frequently reacted to noise rather than genuine price anomalies.

While the mean-reversion strategy managed to contain drawdowns better than trend-following, the higher trade frequency led to increased transaction costs, which negatively impacted profitability. The Sharpe ratio was consistently negative, indicating poor risk-adjusted returns.

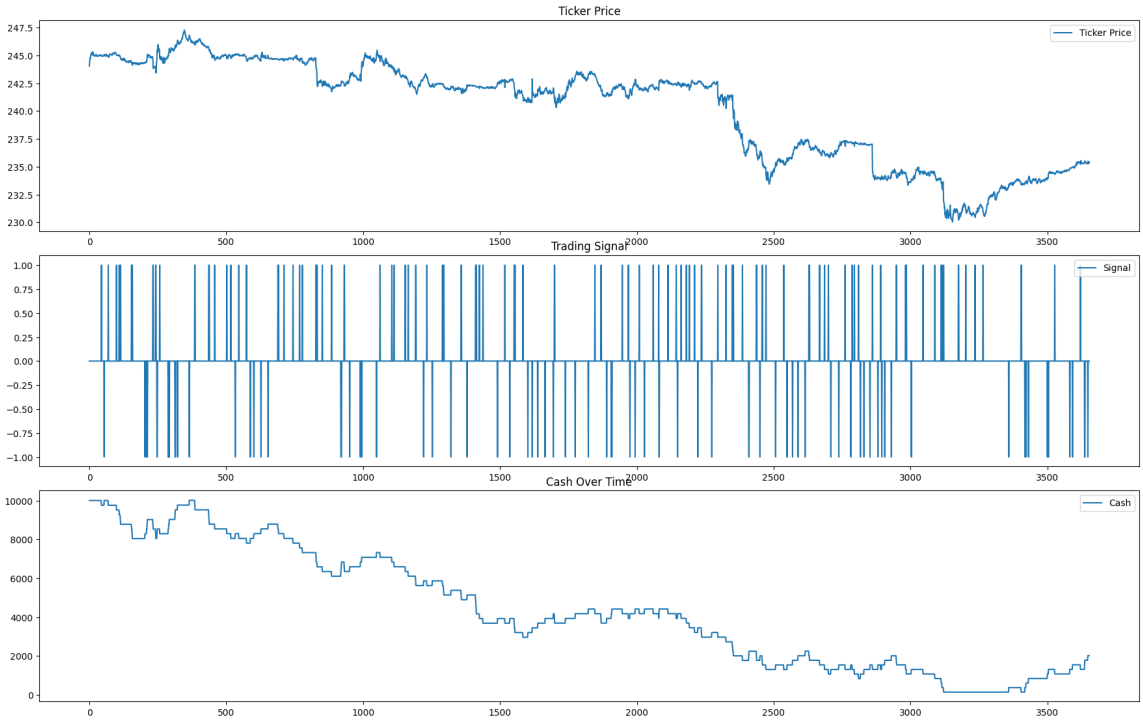



Figure 4.3: Mean-Reversion: Price, Signals, and Cash Over Time.

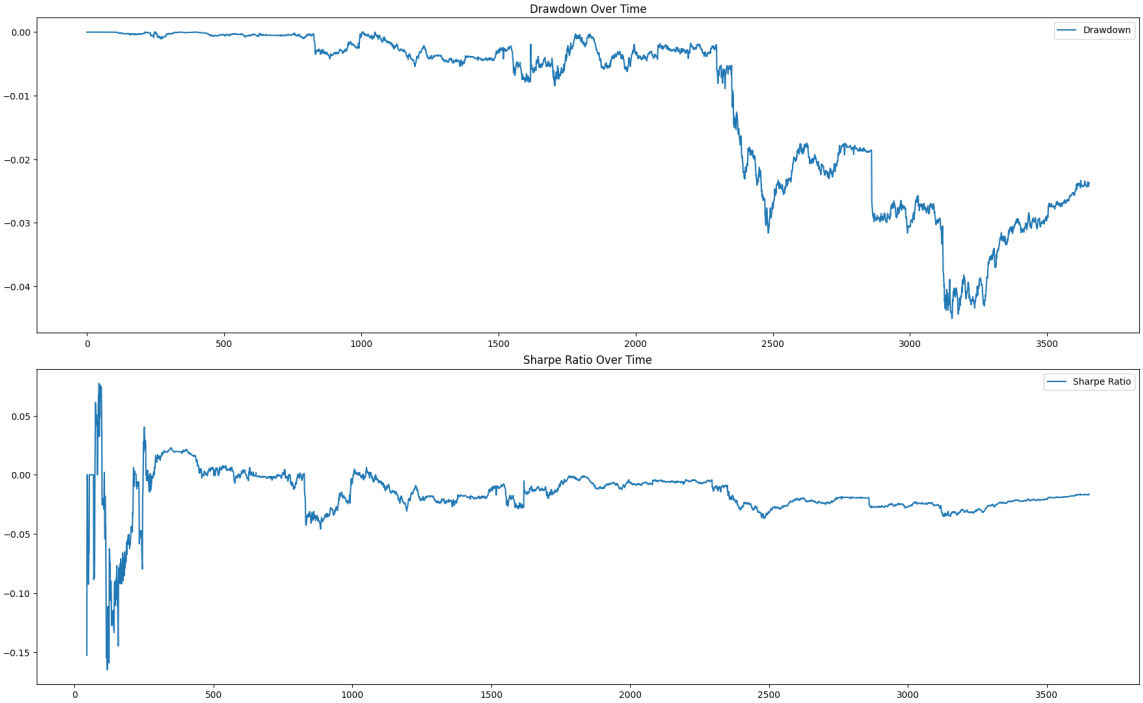



Figure 4.4: Mean-Reversion: Drawdown and Sharpe Ratio Over Time.

Figures 4.3 and 4.4 show the frequent trading signals, cash fluctuations, drawdowns, and risk-adjusted performance of the mean-reversion strategy.

# Comparison Between Strategies

A comparison of the two strategies reveals distinct differences in trade frequency, signal accuracy, and risk-adjusted performance. Both strategies finalized with similar portfolio values: $9785 vs $9862. However, the trend-following strategy incurred a slightly larger loss (-2.15% vs. -1.38%) but executed significantly fewer trades (195 vs. 925). The win rate for trend-following (41.54%) was substantially higher than mean-reversion (15.46%), suggesting better signal quality.

The maximum drawdowns were similarly low for both strategies, with mean-reversion performing slightly better (0.03% vs. 0.05%). However, the frequent trades in mean-reversion will likely lead to higher transaction costs, which offset its performance advantage.

| **Metric** | **Trend-Following** | **Mean-Reversion** |
|---|---|---|
| Final Portfolio Value | $9785.20 | $9862.40 |
| Total Return | -2.15% | -1.38% |
| Maximum Drawdown | 0.05% | 0.03% |
| Number of Trades | 195 | 925 |
| Win Rate | 41.54% | 15.46% |
| Sharpe Ratio | Near Zero | Negative |

Table 4.1: Comparison of Performance Metrics.

Table 4.1 summarizes the key performance metrics for both strategies.

# Key Insights and Recommendations

The trend-following strategy showed better signal accuracy and managed drawdowns effectively, but it struggled to generate positive returns due to low profit-per-trade. The mean-reversion strategy contained drawdowns slightly better but suffered from frequent trades and poor signal accuracy.

To improve performance, mean-reversion could incorporate filters like Bollinger Bands or stricter z-scores to reduce unnecessary trades. Transaction costs should also be accounted for in backtests. For trend-following, momentum indicators such as RSI or MACD could enhance signal quality, while stop-loss mechanisms could mitigate losses during trend reversals. Both strategies could benefit from dynamic position sizing to optimize risk-reward ratios. A hybrid approach combining elements of both strategies might better adapt to varying market conditions.

# 5 Discussion and Conclusion

The development and testing of the two trading strategies, mean-reversion and trend-following, provided valuable insights into their performance and limitations. This section reflects on the outcomes of the backtesting process, the challenges encountered during the implementation, and opportunities for future improvements.

## 5.1 Challenges Encountered

One of the most significant challenges in this project was the integration of live trading systems. The original plan was to use the IB_Gateway within a Dockerized environment. However, technical difficulties arose due to the need for special privileges and additional configuration to run the gateway within Docker. These issues consumed a significant amount of time and ultimately necessitated a switch to the Alpaca API for both data sourcing and brokerage services. While Alpaca offered a straightforward and user-friendly API, the last-minute change limited the scope for live trading experiments and validation of the strategies under real market conditions.

Additionally, experiments with KDB were conducted as part of an effort to leverage its capabilities for high-frequency data handling and analysis. Similar to the IB_Gateway, KDB required elevated privileges and specific configurations to function within a Dockerized environment. These constraints prevented its seamless integration into the project and highlighted the importance of having flexible and well-documented tools for real-time trading applications.

## 5.2 Strategy Performance and Improvements

The backtesting results showed that both strategies were able to limit drawdowns effectively, with the mean-reversion strategy achieving a slightly better maximum drawdown (0.03%) compared to trend-following (0.05%). However, both strategies struggled to generate positive returns, with final portfolio values of $9862.40 and $9785.20, corresponding to total returns of -1.38% and -2.15%, respectively.

The mean-reversion strategy exhibited high trade frequency (925 trades) and a low win rate (15.46%), indicating that it was overly sensitive to market noise. Potential improvements include the use of stricter Z-score thresholds or additional filters, such as Bollinger Bands, to reduce unnecessary trades. Incorporating transaction cost modeling into the backtests could also provide a more realistic assessment of profitability.

The trend-following strategy demonstrated better signal accuracy with a win rate of 41.54%, but its returns were hindered by low profit-per-trade. Enhancements such as adding momentum indicators like the Relative Strength Index (RSI) or Moving Average Convergence

Divergence (MACD) could improve signal quality. Implementing stop-loss mechanisms might also help mitigate losses during trend reversals.

Both strategies could benefit from dynamic position sizing to optimize risk-reward ratios and adapt to changing market conditions. A hybrid approach, combining elements of both strategies, could further enhance robustness and adaptability across varying market regimes.

## 5.3 Lessons Learned and Future Directions

This project underscored the importance of robust infrastructure and reliable data sources in algorithmic trading. The technical challenges encountered with IB_Gateway and KDB highlight the need for careful planning and contingency measures when selecting tools and platforms. Future work could involve the development of more generalized and modular solutions to integrate with different APIs and data sources seamlessly.

Further improvements to the trading strategies could focus on exploring alternative indicators, such as the Average True Range (ATR) or volume-weighted metrics, to enhance signal generation. Additionally, more advanced optimization techniques, such as reinforcement learning, could be applied to adapt the strategies dynamically to market conditions.

Lastly, transitioning from paper trading to live trading with real capital will require rigorous testing, including stress testing under volatile market conditions, to ensure the strategies' robustness and reliability in production environments.

## 5.4 Conclusion

Despite the challenges and limitations, this project provided a comprehensive framework for developing and evaluating algorithmic trading strategies. By addressing the identified gaps and building on the lessons learned, the strategies can be further refined to achieve better performance and scalability in real-world applications.