USDC / USDT Pairs trading

Introduction

Our Trading system aims to retrieve data from Alpaca Markets and store it in our database. This then triggers a refresh in our trading strategy to update the latest signals, and send an order to Alpaca Markets if needed. We then Monitor the performance of our trading strategy throughout the period, to ensure that we are within risk limits. Backtesting has been done to ensure that we are using a more optimized approach with regards to our strategy.

Filename	Purpose
test.py	Combines the database, socket_streaming and api_calls into a single workflow
database.py	Provides an interface to store and manipulate OHLCV data in a SQLite database.
socket_streaming.py	Implements real-time data streaming using Alpaca WebSocket connections.
api_calls.py	Provides a custom interface for interacting with Alpaca's REST API for various trading and data retrieval purposes.
strategy_monitoring.py	Monitoring of various metrics to track our trading strategy's success
strategy.py	Pair trading strategy for USDC/ USDT
backtest.ipynb	Testing and Optimization
/data	Storage for historical data (Backtesting)
config.py	Storage of params

Market Data Retrieval

API Interaction (api_calls.py)

Purpose

Provides a custom interface for interacting with Alpaca's REST API for various trading and data retrieval purposes.

Key Classes and Methods

1. **CustomAlpaca**
    - Extends the default Alpaca API class to override test URLs for sandbox mode.
    - Overrides describe to customize API URLs.
2. **CustomAlpacaAPI**
    - Generalized class for interacting with the Alpaca API.
    - **Key Methods:**
        - fetch_ohlcv(symbol, timeframe): Fetches OHLCV data for a given symbol and timeframe.
        - fetch_order_book(symbol, limit): Retrieves the order book for the specified symbol.
        - fetch_balance(): Gets the account balance.
        - api_call(method, *args, **kwargs): General-purpose method for invoking Alpaca API calls dynamically.
        - close(): Cleans up resources.

Real-Time Streaming (socket_streaming.py)

Purpose

Implements real-time data streaming using Alpaca WebSocket connections.

Key Classes and Methods

1. **CustomAlpacaWs**
    - Extends Alpaca's WebSocket implementation to customize sandbox mode URLs.
2. **AlpacaWebSocket**
    - Manages WebSocket connections for streaming market data.
    - **Key Methods:**
        - initialize_exchange(): Initializes the WebSocket exchange connection.
        - stream_data(): Streams order book data or ohclv data in real-time.
        - run(): Starts the WebSocket loop.

Data Storage Strategy

Database Management (database.py)

Purpose

Provides an interface to store and manipulate OHLCV data in a SQLite database.

Key Methods

1. **create_table(query=None)**
    - Creates a database table to store data including ohlcv
    - If a custom query is provided, it executes the query.

2. **insert_data(ticker, data)**
    - Inserts ohlcv data into the database.
    - Converts timestamps to UTC format before storing.
3. **delete_data(ticker=None, before_date=None)**
    - Deletes data based on the ticker symbol or a date condition.
4. **close()**
    - Closes the database connection.

Trading Strategy Development

1. Defining Trading Goals

Long-Term Strategy with Short-Term Opportunities

- **Financial Goals**: The primary goal is to achieve consistent returns by exploiting inefficiencies in the cryptocurrency market. The strategy aims for long-term growth by repeatedly capturing small profits from temporary price divergences.
- **Risk Tolerance**: Moderate risk tolerance is adopted. While the strategy seeks to capitalize on volatility, it also incorporates hedging mechanisms to mitigate risk.
- **Objective**: Utilize statistical arbitrage in a mean-reverting framework to profit from the convergence of price spreads between two highly correlated assets.

---

2. Selection of Trading Instruments

Choosing USDC/USD and USDT/USD

- **Stablecoins as Instruments**:
    - **USDC (USD Coin)** and **USDT (Tether)** are both stablecoins pegged to the US Dollar.
    - They are designed to maintain a 1:1 ratio with USD but may experience minor price fluctuations due to market dynamics.
- **Reasons for Selection**:
    - **High Liquidity**: Both coins have substantial trading volumes, reducing slippage and ensuring efficient order execution.
    - **Volatility in Spread**: Despite being stablecoins, transient divergences occur due to supply and demand imbalances, providing arbitrage opportunities.
    - **24/7 Market Access**: Cryptocurrency markets operate continuously, allowing the strategy to function without interruption.
- **Higher Chance of Spread Divergence**:
    - The crypto market's inherent volatility increases the likelihood of significant spread deviations, even among stablecoins.
    - Regulatory news, market sentiment, or differences in issuance mechanisms can cause temporary dislocations.

---

3. Technical Indicators and Signals

Implementing a Mean Reversion Strategy

- **Price Spread Calculation**:
    - **Spread**: The difference between the price of USDC/USD and USDT/USD.
    - **Data Collection**: Utilizing 1-minute tick data over the past 30 days to calculate a rolling window of spreads.
- **Statistical Measures**:
    - **Mean Spread**: The average spread over the 30-day window.
    - **Standard Deviation**: Measures the spread's volatility around the mean.
- **Signal Generation**:
    - **Thresholds**:
        - **Upper Threshold**: Mean spread plus 1.5 times the standard deviation.
        - **Lower Threshold**: Mean spread minus 1.5 times the standard deviation.
    - **Trading Signals**:
        - **Sell USDC and Buy USDT**: Triggered when the current spread exceeds the upper threshold, indicating USDC is overvalued relative to USDT.
        - **Buy USDC and Sell USDT**: Triggered when the current spread falls below the lower threshold, indicating USDC is undervalued relative to USDT.
- **Assumption**:
    - The spread between USDC and USDT is mean-reverting; deviations from the mean are temporary and will correct over time.

---

4. Risk Management Rules

Mitigating Potential Losses

- **Hedged Positions**:
    - **Market Neutrality**: By simultaneously taking long and short positions in two correlated assets, the strategy hedges against overall market movements.
- **Position Limits**:
    - **Fixed Trade Size**: Limiting each trade to a predefined amount (e.g., $1,000) to control exposure.
- **Diversification**:
    - While focusing on a specific pair, the strategy benefits from the stability of stablecoins, reducing unsystematic risk.
- **Continuous Monitoring**:
    - **Real-Time Adjustments**: The algorithm monitors positions and market conditions to adjust or exit trades if necessary.

---

5. Position Sizing

Determining Trade Amounts

- **Consistent Trade Size**:
    - Each trade involves a fixed dollar amount to simplify execution and maintain consistent risk levels.
- **Scaling Strategy**:
    - **Adjustable Sizing**: Position sizes can be scaled proportionally with account growth or adjusted based on volatility measures.
- **Leverage Considerations**:
    - **Avoiding Excessive Leverage**: Using minimal or no leverage to reduce the risk of margin calls and amplify losses.

---

6. Implementation Details

Algorithm Development and Execution

- **Data Initialization**:
    - **Historical Data Fetching**: Collecting 30 days of historical 1-minute tick data for USDC/USD and USDT/USD to calculate initial statistical measures.
    - **Data Storage**: Using a deque (double-ended queue) to efficiently manage the rolling window of spread data.
- **Algorithm Logic**:
    - **Fetch Prices**: Continuously retrieving the latest prices for both assets.
    - **Update Spread**: Calculating the current spread and updating the spread history.
    - **Generate Signals**: Evaluating whether the current spread crosses predefined thresholds to trigger trades.
    - **Execute Trades**: Placing market orders via the Alpaca API based on generated signals.
- **Automation**:
    - **Continuous Operation**: The algorithm runs in an infinite loop, enabling it to respond promptly to market changes.
- **Use of Alpaca API**:
    - **Integration**: Leveraging Alpaca's trading platform for reliable order execution and market data access.
    - **Paper Trading Environment**: Testing the strategy in a simulated environment to validate performance before live deployment.

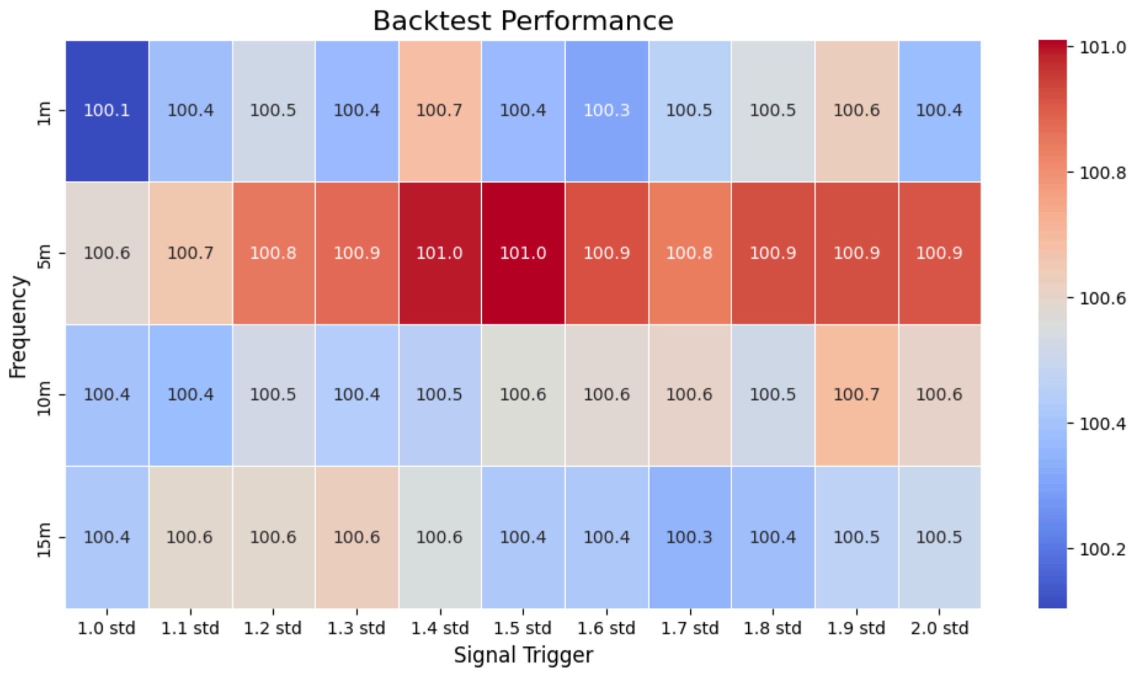
Testing and Optimization (backtest.py)

Purpose

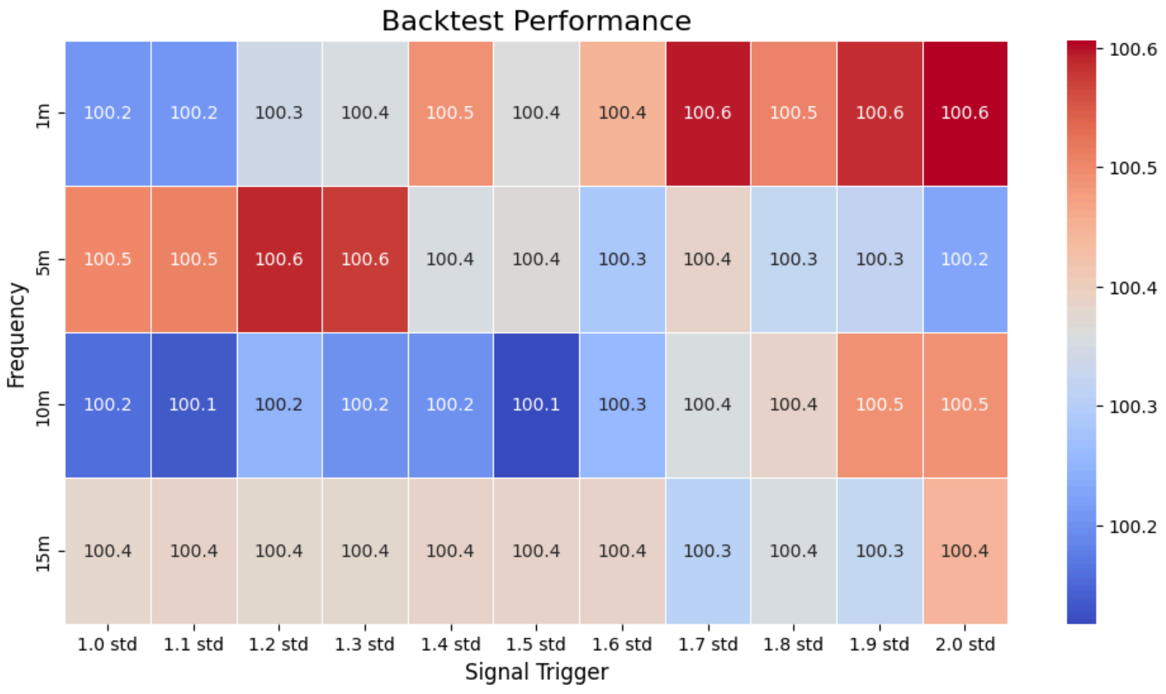
To test whether the trading strategy works. Also to test for the best parameters to use for the trading strategy.
We will be doing a gridsearch for 4 frequencies: 1m, 5m, 10m and 15m; as well as for 10 sets of standard deviations as the trigger levels (1std - 2std, with increments of 0.1 std).

Approach

1. Firstly, we obtain USDT/USD and USDC/USD historical OHLC data from Alpaca Markets, and save it to /data as a parquet file. We take data of 1, 5, 10, 15 min frequencies.
2. Generate the rolling mean and standard deviation.
3. Creates a signal for each combination of frequency, and standard deviation (as a trigger).
4. Run the backtest, where we initialize with the minimum rolling-window size and iterate through the November 2024 data points (walk-forward test from Nov 01 - Nov 27 2024).
5. After running the backtest for all the combinations, we take a look at their cumulative wealth. Note that 100% is the starting wealth.



6. We also tried to experiment with always holding a position, such that if we are long, we dont close it until we hit the trigger to go short, in which case we ‘flip’ our position.



7. Given the parameter optimization we have done, we will be conducting the strategy using **5 min** data, and using **1.5 std** as the trigger level.

Integration and Testing (test.py)

Purpose

Combines all components to demonstrate their functionalities in a cohesive workflow.

Workflow

This uses database.py file for automatically retrieving and inserting data and api_calls for market data retrieval and finally if needed, we can stack real-time data into database parsing the response

1. **Fetch OHLCV Data:**
   - Uses CustomAlpacaAPI to fetch OHLCV data for a specific symbol and timeframe.
   - Stores the data in the SQLite database using the Database class.
2. **Insert Data into Database:**
   - Inserts fetched data into the database and verifies successful insertion.
3. **Stream Real-Time Data:**
   - Streams real-time order book data using the AlpacaWebSocket class.

Performance Monitoring

Key Methods

get_positions(api_key, api_secret)

- Retrieves the current positions held in the Alpaca account.
- Sends a GET request to the Alpaca API endpoint.
- Returns details of all open positions, which are useful for assessing the portfolio's current exposure.

get_PnL(api_key, api_secret)

- Calculates the overall Profit and Loss (PnL) of the account.
- Queries the Alpaca API endpoint for historical portfolio data.
- Computes the total PnL by summing all profit and loss values and adding the starting capital.
- Helps measure the financial performance of the trading strategy.

successful_trades(api_key, api_secret)

- Counts the number of successful and unsuccessful trades.
- Extracts profit and loss data from the Alpaca portfolio history.
- Identifies positive PnL trades as successful and non-positive as unsuccessful.
- Provides a summary of trade success, aiding in strategy evaluation.

drawdowns(api_key, api_secret)

- Calculates the drawdowns over time.
- Uses portfolio data to compute drawdowns and the maximum drawdown percentage.
- Assesses the risk and volatility of the trading strategy.

Purpose

These methods collectively monitor key performance metrics of the trading strategy:

- **Portfolio exposure** (get_positions)
- **Overall profitability** (get_PnL)
- **Trade success rates** (successful_trades)
- **Risk metrics** (drawdowns)
- **Operational functionality** (test_PnL)

Conclusion

This trade is relatively simple to implement, and the convergence between USDT and USDC should hold true as long as both bear the same credit risk, and remain pegged to the USD. We have created a custom class to interact with the Alpaca API as well as have a socket-streaming connection to be able to pull data live - as we run our trading strategy. The data will be stored in a database. We also have strategy monitoring features available, to track our live performance. The testing of our strategy and optimization was done thoroughly in the jupyter notebook, however, we note that we can explore other frequencies, as well as implementing transaction costs to get a better understanding of our net PnL.