Algorithmic Trading for Reversion and Trend-Following v2025 Algorithmic trading in HFT space is concerned with how to optimally run the core strategies focusing on execution. The issues of order types as allowed by broker/exchange, slippage, incorrect messages from a broker, and own high-frequency data storage (15 Min and less) – can disturb any strategy and you address them in Part III (last but not the least). However, this is a quantitative study. With attention to maths and application detail (eg, sample period/time window experiment) implement at least two kinds of (a) trend-following strategy with several indicators of your choice AND (b) reversion strategy with an indicator of your choice. This is Part I and you can do it as a historic backtest. In Parts II and III extend the strategies from backtesting to some degree of live-testing to test simple scripts with Broker API of your choice. However, if specific Broker API/language learning curve is steep, you can limit the scope – Part II covers more detail. Real-time trading relies on loops which check for buy/sell signal, and to improve the performance more optimized languages than Python are used in the industry, eg GoLang. A. The trading code must handle exceptions: failures during the order execution and inconsistent/incorrect information received back from the broker (eg, order filled when it wasn’t, mismatch in order type). B. Trading overall should consider liquidity, market impact, and sudden market events – though you are not likely to be in position to affect the market but consider, 1. changes in price and volume (market impact, liquidity regime); 2. order-specific events, subject to information about execution from API. Trend-Following Strategy Trend-following strategies aim to capitalize on sustained movements in the market. The indicators such as the Moving Averages, Exponential Moving Averages (EMA) and Average Directional Index (ADX) to confirm trends. The ADX measures the strength of a trend, helping to filter out weak trends that may not be profitable. You can combine moving averages with ADX, in order to generate better quality trading signals. Experiment with resampling interval and averaging period to assess the effectiveness of your specific trend-following approach. Discuss or better backtest the impact of various market-specific events/liquidity conditions/regime in asset price.

Part I: Generic Strategies Made Proprietary Write code that implements testing on your core strategies, that can be done Python and on historical data (backtesting). Understanding mathematical nuance of indicator, and its backtesting has different purposes, as compared to live-testing which would focus on order slippage, for example. Here, you are not limited to Python, and can use a specialized language + Broker API for Part I as well. There are Backtrader, Zipline, PyAlgoTrade packages but Python does’t have a ready one solution for our Part I purposes. 1. For a core trend-following strategy type, common choices are Exponential Moving Average (EMA) and Average Directional Index (ADX), which is a kind of oscillator. Other approach is a convergence/divergence indicator, Moving Average Convergence/Divergence (MACD). For each strategy, you need to decide on: your own indicator, how it is computed, and what constitutes a trading signal (eg, crossover of 20D EMA with another). 2. Simple but practical trend indicator primarily used in FX is of the following design: • Step 1: resample the prices at regular intervals (eg, 30 seconds) for price level or average; Can use DataFrame.resample; • Step 2: calculate an average price over the longer period (eg, 5-minute intervals). • Step 3: compute the ratio of short-term price (or its average) to long-term average price: near 1 signals ’no trend’ as short-term prices ≈ the long-term prices. Uptrend is signaled by the ratio above 1, and downtrend by less than 1. Give several calibrations.

Full mathematical description of indicators chosen and your calibration: experiment with the ratios over different timeframes, frequency. 3. For a mean-reversion strategy type, • Z ∗ deviation from the price can be used as a simple signal. Or think about distance measures from machine learning. • Formal modeling of mean-reversion with OU process can be invoked expecting the price to mean-revert over short-time. • OPTIONAL To generate a stable P&L from reversion strats, it’s very likely some kind of filtering needs to be applied to the price (see Topic TS). 4. Discussion. Consider the behaviour of the non-stationary price (regime): for example, would the upward trend with more jumps and volatility produce better/worse returns for a mean-reversion? Part II: Broker API and Input data 1. Treat this project as more professional, eg, even for a historical backtesting fetch data from OpenBB/brokerage (vs Yahoo!Finance) and write a couple of routines checking data quality. You are encourage to work with 15-minute and higher-frequency data. 2. The common API choice is REST (Representational State Transfer). • Alpaca, Interactive Brokers Web, and Oanda all have their own versions. In particular, Alpaca REST API is free and includes asynchronous events handling based on WebSocket and Server Side Events (SSE). • REST API is referred to as HTTP API because it utilizes HTTP methods, eg GET (retrieve data), POST (create new data), PUT (update data). Useable for non-timecritical operations such as retrieving historical data, account information, placing orders, and getting order status. 3. The more industrial strength and lower latency API choice is FIX (Financial Information eXchange). A messaging protocol designed for the real-time exchange of securities transactions. It supports submission and cancellation of various order types, trade execution

reports, and market data dissemination – all for high frequency. FIX is used by large institutions, funds, and broker/dealers. 4. Interactive Brokers offer TWS API with connection to their client application and possibility to use C++, C#, Java, Python. A useful comparison at www.interactivebrokers. com/en/index.php?f=5041. Describe order types suitable to your tickers and strategies, and attempt code for order loops and order handling using an API. Set price parameters far from the market to avoid execution. Though, it is absolutely recommended that you do not run any actual live-trading. Part III: Evaluate Risk and Test Thrice An opened market position is exposed to various types of risk. While the market risk can be managed with quant methods like rolling VaR, order-handling risk is more important and reliant on API choices (Part II) as well as dev environment and stack of libraries choices. 1. Containerise and consider docker and crone-style scheduling (think how different parts of your code can be scheduled to run as an independent scripts). Consider implications for running the image of your code data collection, backtest, and trading order loops on a virtual server. 2. Positions Tracking. Code can request account updates (eg, in loop) to provide a secondary confirmation layer. • Code must have a verification of server responses. From the broker side orders may be partially filled and/or returned with an incorrect fill information (eg, ticker not bought when it was actually bought). • Catch the stale websockets and connection issues.

3. Performance and Risk Reporting. Of less utility to algotrading are Pyfolio-style systematic backtesting, scorecards with concentration in asset vs portfolio, Beta-to-SPY. However, do report on P&L performance, turnover, trading costs, and strategy-specific drawdowns. • One recipe but not mandatory is to present a risk dashboard: run a strategy alike to live for N trading days; save fills and P&L in TimescaleDB (your own database); display drawdown, Sharpe, intraday VaR. You can try Streamlit.

4. Market Data. Introduce simple checks to catch inconsistencies in market data (downloaded for a backtest or received from the broker live). For example, futures price can’t be below spot price, implied quantities can’t be negative