Quant Developer Evaluation Assignment

Objective

Design and implement a small but complete analytical app that demonstrates your ability to work end-to-end from real-time data ingestion and storage to quantitative analytics and interactive visualization. You are provided with a sample HTML WebSocket tool that streams live tick data from Binance for selected symbols. Your goal is to build a system that ingests this stream, samples and processes it, computes key analytics, and presents the results through an interactive front-end.

Core Expectations

The project should include both a Backend (for real-time data ingestion, storage, and analytics) and a Frontend (for visualization, interactivity, and user control). Your architecture and design choices (databases, libraries, frameworks) are up to you, document your reasoning clearly.

Context

Helper analytics for traders, researchers at an MFT firm which trades across cash and derivatives segment primarily involved in statistical arbitrage, risk-premia harvesting, market-making, term-structure calls, micro-alphas across commodities, fixed income, energy and equities.

Workflow Overview

- Data Source: Use the provided HTML file connecting to Binance WebSocket streams emitting tick data {timestamp, symbol, price, size/qty}. You can refer to the official documentation to use more fields if necessary.
- Data Handling: Build an ingestion pipeline that reads tick data from the WebSocket and stores it (e.g., DB, Redis, SQLite, your design choice). Implement sampling for selectable timeframes (1s, 1m, 5m).
- Analytics: Compute price stats, hedge ratio via OLS regression, spread, z-score, ADF test, and rolling correlation. Extend with creative analytics. Plot the same using relevant charts. Show interactive visualizations (prices, spread, volume, stats etc)
- Live Analytics: Frontend should update key stats in near-real-time as WebSocket data streams. This isn't relevant for all analytics plots. For e.g. If plot shows resample in 5m, then that plot might only need to update every 5m but relevant tick-based analytics using the same plot, for example z-score may need to be updated live (say with a 500ms latency)
- Alerting: Allow users to define simple custom alerts (e.g., z-score > 2).
- Data Export: Provide download options for processed data and analytics outputs.

Frontend Requirements

Build an interactive dashboard using Streamlit, Flask + Plotly, Dash, HTML+JS, React or any framework of your choosing. We require you to use a python based framework on the backend. Display price charts, spread & z-score, correlation plots, and summary stats. Include controls for symbol selection, timeframe, rolling window, regression type, and ADF test trigger. Charts must support zoom, pan, and hover. Should be able to see analytics for multiple different products. Widget based designs are encouraged as well.

Suggested Advanced Extensions

- Dynamic hedge estimation via Kalman Filter
- Robust regression (Huber / Theil–Sen)
- Mini mean-reversion backtest (z>2 entry, z<0 exit)
- Liquidity filters and cross-correlation heatmaps
- Rule-based alerts and creative visual summaries
- Table displaying stats in time-series for different minutes along with the features at that point having a download button for csv export
- Any other creative analytics, features that you believe suits such an application

Deliverables

Your application, on run, should show the basic analytics it can and alert functionality possible with the real time data. Simultaneously, it should start aggregating and storing/resampling. As and when the required data points for different analytics are available, the corresponding functionality can be enabled on frontend. Do not use any analytics that require more than a day of data. Additionally include the functionality to upload OHLC data though it is mandatory that this works without any dummy upload.

- Runnable App - single-command local execution (e.g., python app.py).
- README.md - setup, dependencies, methodology, and analytics explanation.
- ChatGPT Usage Transparency - short note on how AI was used, prompts used.
- Architecture Diagram - created in draw.io or similar, showing ingestion, storage, analytics, APIs, frontend, and alert flows. Include .drawio source and an exported PNG/SVG.

Evaluation Criteria

- Frontend (30%) - Usability, interactivity, clarity, completeness, UI design, UX.
- Backend (30%) - Correct sampling, analytics accuracy, code quality, insight, modularity.
- Architecture & Design (40%) - Diagram clarity, trade-offs, extensibility, redundancies, logging.

Notes

You are encouraged to use ChatGPT or other LLMs for coding, debugging, or structuring your approach - just document usage briefly. You may freely choose tools, storage, and design, provided your rationale is sound. The purpose is to assess your analytical reasoning, design ability, and clarity of communication. This project is intended to take no more than a day of your time.

Design Philosophy

While your current implementation need not be production-scale, your architecture should reflect modularity and foresight.

Design the system such that:

- Components remain loosely coupled and clearly defined - data ingestion, analytics computation, storage, and visualization should interact through clean interfaces or APIs.
- Scaling does not require rewriting: imagine needing to plug in a different data feed (e.g., CME futures, REST API, or historical CSV) - this should involve minimal rework.
- Extensibility is deliberate: your design should make it straightforward to add new analytics, visual modules, or data sources without breaking existing logic.
- Clarity over complexity: simplicity and readability take precedence over premature optimization.

In essence, even though this project runs locally, think of it as a prototype that could grow into a larger, real-time analytics stack. We do not want these extended functionalities in the current assignment; however your current decisions should have been based on allowing and accounting for the same. The ability to recognize where scaling challenges *would* occur (and how you would isolate or mitigate them) will be valued as much as the code itself.

Something to remember

We encourage you to go beyond for brownie points and show some analytics, implement some features that you feel would perhaps be useful to a trader. The numerical/analytical robustness of the same will not be used as a judging criterion but the intuitiveness and rationality or intended impact of how it might make it easier to use along with your technical decisions will be the line of focus. The final intention is to test your development skill from a business problem-solving perspective rather than a code only perspective.