Maram Mounish Reddy

Quant Developer Evaluation Project

Real-Time Analytics Dashboard

November 3, 2025

Contents

1 Project Objective

The primary objective of this project is to design, implement, and document a complete end-to-end analytical application. The system demonstrates the ability to ingest real-time tick data from a live WebSocket feed (Binance), store this data efficiently, perform complex quantitative analytics, and present the findings through an interactive web-based front-end.

The project is designed as a modular prototype for a professional trading analytics stack, emphasizing scalability, extensibility, and clarity, as per the assignment's design philosophy.

2 System Architecture

The application is built using a modern, multi-service (or microservice-style) architecture. This design, orchestrated by `docker-compose`, ensures that each component of the system is loosely coupled, independently maintainable, and resilient.

2.1 Service Overview

The system is comprised of five distinct services that run in isolated Docker containers:

`db (PostgreSQL + TimescaleDB)` The core time-series database. It stores all raw tick data and the pre-computed 1-minute and 5-minute OHLC bars.

`ingestor (Python)` A standalone WebSocket client. Its sole responsibility is to connect to the Binance stream, receive trade ticks, and insert them into the database.

`aggregator (Python)` A simple but crucial service that runs in a loop. It periodically tells the TimescaleDB to refresh its continuous aggregates (the 1m and 5m bars), keeping them up-to-date.

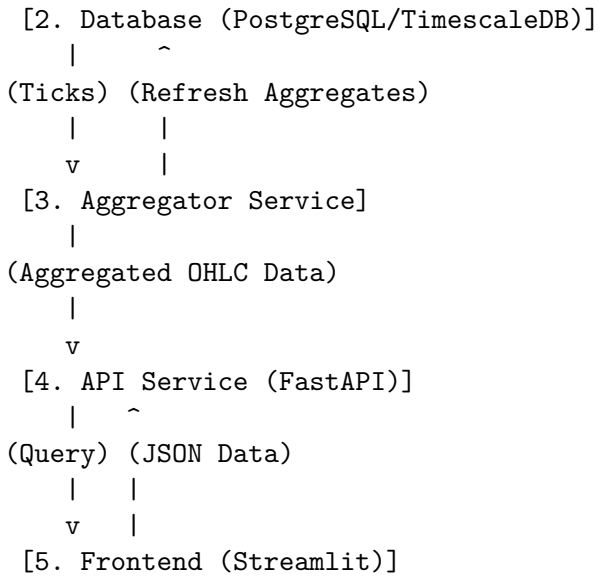
`api (Python/FastAPI)` A high-performance, asynchronous backend API. It queries the database, performs all statistical calculations, and serves clean JSON data to the frontend.

`frontend (Python/Streamlit)` The interactive web dashboard. It is a pure client of the API and handles all user interaction, data visualization, and alerting.

2.2 Data Flow Diagram

The flow of data is unidirectional and logically separated:

```
 [Binance WebSocket]
        |
 (Real-time Ticks)
        |
        v
 [1. Ingestor Service]
        |
    (SQL Insert)
        |
        v
```



3 Core & Advanced Features

The dashboard fulfills all core assignment requirements and includes several advanced extensions.

3.1 Core Features

- **Real-Time Ingestion:** Connects to the Binance Futures WebSocket and ingests live trade data for multiple symbols.
- **Data Sampling:** Automatically aggregates ticks into 1-second, 1-minute, and 5-minute OHLCV bars.
- **Dual Data Modes:** The app can run analysis on either the live data feed or on a user-uploaded historical OHLCV CSV file.
- **Interactive Charts:** All charts, built with Plotly, support full zoom, pan, and hover capabilities, with a persistent UI state.
- **Live Stats:** Real-time metric boxes display the latest price, volume, z-score, and spread.
- **Alerting:** A simple on-screen `st.toast` notification fires when the pair's z-score breaches a ± 2.0 threshold.
- **Data Export:** Users can download all processed data from the dashboard as a `.csv` file.

3.2 Advanced Analytics (Extensions)

To demonstrate extensibility, the pair trading module was enhanced with an advanced regression suite, allowing the user to select their desired model:

- **OLS (Static):** Standard Ordinary Least Squares regression.
- **Huber (Robust):** A robust model that is less sensitive to outliers in the data.
- **Theil-Sen (Robust):** A non-parametric, highly robust model that is resilient to significant outliers.

- **Kalman Filter (Dynamic):** A sophisticated model that calculates a *dynamic* hedge ratio, which adapts with each new data point. This is plotted as a separate time-series chart.

4 Technology Stack & Justification

The technology for each service was chosen to meet specific design goals.

Docker Compose
Why: Fulfills the ”single-command execution” and ”modular architecture” requirements. It allows the five services to run in isolated, linked containers, making the system resilient and easy to deploy.

PostgreSQL + TimescaleDB
Why: This is a time-series data problem. TimescaleDB is a PostgreSQL extension that provides superior performance for this use case. Its **Continuous Aggregates** feature is the most efficient solution for the 1m/5m sampling requirement, performing the aggregation inside the database itself.

FastAPI
Why: A modern, high-performance Python backend. Its asynchronous nature is perfect for an I/O-bound application that frequently queries a database. It is significantly faster than Flask and provides automatic data validation.

Streamlit
Why: The fastest way to build an interactive data dashboard in Python. It natively integrates with Plotly and provides all required widgets (sidebar, file uploader, radio buttons) out of the box, fulfilling the ”widget-based design” encouragement.

Plotly
Why: Explicitly meets the ”zoom, pan, and hover” chart interactivity requirement.

Statsmodels & Scikit-learn
Why: The industry-standard Python libraries for statistical analysis. They provide the exact functions needed for OLS, ADF, Huber, and Theil-Sen regressions.

PyKalman
Why: The standard library for implementing the Kalman Filter advanced extension.

5 How to Run the Application

5.1 Prerequisites

- Docker Desktop must be installed and running.

5.2 Single-Command Launch

The entire 5-service stack is launched with a single command.

```
# 1. Navigate to the project's root directory
cd /path/to/assignment

# 2. Build and run all services
docker-compose up --build
```

Listing 1: Terminal command to launch the project

After the services start, the dashboard is accessible in a web browser at:

`http://localhost:8501`

Note: Please allow 1-2 minutes after launch for the database to populate with 1-minute aggregates.

6 Project Dependencies

The project's dependencies are managed in two separate `requirements.txt` files.

Table 1: Python Library Dependencies

Service	Libraries
`backend`	websockets, psycopg2-binary, fastapi, uvicorn, pandas, statsmodels, numpy, pykalman, scikit-learn
`frontend`	streamlit, requests, plotly, pandas, numpy, statsmodels, pykalman, scikit-learn