Individual Programming Assignment: Detecting Crypto Volatility in Real-time

Start Assignment

- Due Friday by 11:59pm
- Points 100
- Submitting a text entry box, a website url, a media recording, or a file upload

(Individual) Programming Assignment: Detecting Crypto Volatility in Real Time

Executive Overview

You will build a real-time data pipeline that connects to Coinbase’s Advanced Trade WebSocket API, collects live market data, and streams it to Kafka. You’ll process this data into features, train models to detect short-term volatility spikes, log your experiments to MLflow, and generate an Evidently report to monitor data quality and drift. You’ll deliver three milestones over three weeks.

What You’ll Do

You will build a real-time data pipeline that connects to Coinbase’s Advanced Trade WebSocket API, collects live market data, and streams it to Kafka. You’ll process this data into features, train models to detect short-term volatility spikes, log your experiments to MLflow, and generate an Evidently report to monitor data quality and drift.

Learning Goals

- **Streaming & packaging**: Stream data via Kafka and run services with Docker Compose.
- **Features & analysis**: Build a reliable, reproducible feature pipeline and explore the data.
- **Modeling & tracking**: Train and log baseline and ML models using MLflow; monitor drift with Evidently.

Tools You Must Use

- **Language**: Python 3.10+
- **Streaming**: Kafka (KRaft) via Docker Compose
- **Tracking**: MLflow (local SQLite backend or containerized)
- **Drift & Quality Reporting**: Evidently (HTML or JSON reports)
- **Data Storage**: NDJSON, Parquet, or SQLite
- **Secrets**: Never commit secrets. Use .env files or environment variables.
- **Note**: This assignment uses public market data only. Do not place trades.

Milestone Breakdown

Milestone 1: Streaming Setup & Scoping

Goal: Get Kafka and MLflow running. Ingest data and define your problem.

Tasks: - Launch Kafka and MLflow using Docker Compose. - Ingest Coinbase WebSocket ticker data for 1–2 trading pairs (e.g., BTC-USD). - Ensure reconnect/resubscribe and heartbeats are implemented. - Publish incoming ticks to the Kafka topic ticks.raw. Optionally mirror to data/raw/. - Write a Kafka consumer to validate your stream. - Write a one-page Scoping Brief: use case, 60-second prediction goal, success metric, and risk assumptions. - Containerize your ingestor with Dockerfile.ingestor.

Deliverables: - docker/compose.yaml, docker/Dockerfile.ingestor - scripts/ws_ingest.py and scripts/kafka_consume_check.py - docs/scoping_brief.pdf - config.yaml (if used)

What to test: - docker compose ps shows all services running. - Running ws_ingest.py for 15 minutes yields messages in ticks.raw. - Container builds and runs successfully.

Milestone 2: Feature Engineering, EDA & Evidently

Goal: Build a Kafka consumer that computes features and generate your first data report.

Tasks: - Build features/featurizer.py (Kafka consumer) to compute windowed features like: - midprice returns, bid-ask spread, trade intensity (optionally order-book imbalance). - Output to Kafka topic ticks.features and also save to Parquet. - Build a replay script: take saved raw data, regenerate features identically. - Conduct EDA in a notebook. Use percentile plots to set a spike threshold. - Create your first Evidently report comparing early and late windows of data.

Deliverables: - features/featurizer.py, scripts/replay.py - data/processed/features.parquet, notebooks/eda.ipynb - docs/feature_spec.md including:

Target horizon: 60s
Volatility proxy: rolling std of midprice returns over the next horizon
Label definition: 1 if σ_future >= τ; else 0
Chosen threshold τ: <value> (justify via plots)

- reports/evidently/ with HTML/JSON report

What to test: - Replay and live consumer should yield identical features. - Evidently report includes drift and data quality.

Milestone 3: Modeling, Tracking, Evaluation

Goal: Train a model, log everything, and compare to a baseline.

Tasks: - Train one **baseline model** (e.g., z-score rule) and one **ML model** (e.g., Logistic Regression or XGBoost). - Use time-based train → validation → test splits. - Log your parameters, metrics, and model artifacts to MLflow. - Metrics must include: PR-AUC (required); optionally F1@threshold. - Write a Model Card v1. - Generate a fresh Evidently report comparing test vs training distribution.

Deliverables: - models/train.py, models/infer.py, and models/artifacts/ - reports/model_eval.pdf, refreshed Evidently report - docs/model_card_v1.md, docs/genai_appendix.md

What to test: - MLflow UI shows at least 2 runs (baseline and ML). - infer.py scores in < 2x real-time for your windows. - Evaluation report includes PR-AUC.

Quick Commands (Example)

```
# 0) Start Kafka + MLflow
$ docker compose up -d
```

```
# 1) Ingest 15 minutes of ticks
$ python scripts/ws_ingest.py --pair BTC-USD --minutes 15

# 2) Check messages in Kafka
$ python scripts/kafka_consume_check.py --topic ticks.raw --min 100

# 3) Build features
$ python features/featurizer.py --topic_in ticks.raw --topic_out ticks.features

# 4) Replay raw to verify feature consistency
$ python scripts/replay.py --raw data/raw/*.ndjson --out data/processed/features.parquet

# 5) Train and evaluate
$ python models/train.py --features data/processed/features.parquet
$ python models/infer.py  --features data/processed/features_test.parquet
```

Handoff to Team Project

Create a /handoff/ folder. Your team will either: - Use **your model** as the base, OR - Create a **composite** from the best parts of all team members.

You must include: - docker/compose.yaml, Dockerfile.ingestor, .env.example - docs/feature_spec.md, docs/model_card_v1.md - models/artifacts/, requirements.txt - A 10-minute raw slice + its features - reports/model_eval.pdf, Evidently report, and one predictions file - A short note: “Selected-base” OR “Composite” with exact steps

Repository Layout (expected)

```
/data/raw/                 Captured raw data
/data/processed/           Final features
/features/                 Feature job
/models/                   Training, inference, artifacts
/notebooks/                EDA
/reports/                  Evaluation + drift reports
/scripts/                  Ingest, replay, Kafka sanity check
/docker/                   Compose + Dockerfile
/docs/                     Briefs, specs, model card, GenAI log
/handoff/                  Files passed to team
config.yaml  requirements.txt  README.md
mlruns/                     MLflow store (if using file backend)
```

Use of Generative AI

You may use GenAI tools (e.g., ChatGPT or GitHub Copilot) for code or writing. You must: - Document how you used it in docs/genai_appendix.md - Use this format:

Prompt (summary): "Generate docstring for featurizer"
Used in: features/featurizer.py
Verification: I reviewed and edited the response

Grading Rubric

Area	Points	What We’re Looking For
Streaming & Packaging	25	Kafka working, producer/consumer run, image builds, replay OK
Feature Engineering & Drift	25	Features clean, EDA done, drift report complete, replay match
Modeling & Evaluation	30	MLflow logs, PR-AUC reported, model card + artifacts
Docs & Professionalism	20	Clear structure, model card, GenAI appendix, team handoff ready

Timeline & Lecture Sync

- **Milestone 1**: Scoping + Kafka Setup → Lectures on Value Scoping & Data Foundations
- **Milestone 2**: Feature Engineering + Drift → Lectures on Data Patterns & Quality
- **Milestone 3**: Modeling + MLflow + Evaluation → Lectures on Modeling & Evaluation

References and Professional Resources

- Shapira, Gwen, Todd Palino, Rajini Sivaram, and Krit Petty. ***Kafka: The Definitive Guide*** **(https://learning.oreilly.com/library/view/kafka-the-definitive/9781492043072/)**. 2nd Edition. O'Reilly Media, Inc. November 2021. *Comprehensive technical reference for Apache Kafka architecture and implementation patterns.*
- Apache Kafka Documentation: **https://kafka.apache.org/documentation/** **(https://kafka.apache.org/documentation/)** *Official technical documentation and configuration reference.*