Take-Home Test: Fraudulent Transaction Detection

Preface

This examination is designed to assess candidates across a broad spectrum of proficiency levels. We recognize that some sections may be challenging for candidates applying for interns or junior-level and certain topics may be unfamiliar, but please view this as an opportunity to deepen your understanding and gain insight into the practical application of your knowledge. Additionally, any concepts required to complete this assessment, that you may have not yet mastered, are essential for your future development, so this is a valuable chance to learn. We wish you the best of luck.

Background Context

You are tasked with building a machine learning solution for a renowned financial institution to **detect fraudulent transactions**. You will receive a **dataset in CSV format**. Your solution should not only build a performant classifier but also propose an actionable plan and show end-to-end deployment. Please document your work and organize your code and instructions for reproducibility.

Dataset Description

Rows: ~6M
Source: https://scbpocseasta001stdsbx.z23.web.core.windows.net/
Columns:

- time_ind: Simulation unit of time (step=1 is 1 hour; total 744 steps = 30 days)
- transac_type: Transaction type (CASH-IN, CASH-OUT, DEBIT, PAYMENT, TRANSFER)
- amount: Transaction amount (local currency)
- src_acc: Customer initiating the transaction
- src_bal: Initial balance (sender) before transaction
- src_new_bal: New balance (sender) after transaction
- dst_acc: Transaction recipient
- dst_bal: Initial balance (recipient) before transaction (missing for merchants)
- dst_new_bal: New balance (recipient) after transaction (missing for merchants)
- is_fraud: Transactions made by fraudulent agents (target)
- is_flagged_fraud: Transactions flagged for illegal attempts (e.g. transferring more than 200,000 in one transaction)

Disclaimer: The dataset is mocked and may contain some discrepancies.

Problem Statement & Deliverables

Please complete **ALL** the following steps:

1. Exploratory Data Analysis (EDA)

- Possible deliverable: notebook(s)
- Show your approach when you first receive new data. What checks/analyses do you perform?
- Uncover any data quality or integrity issues.
- Summarize the characteristics and important insights relevant to fraud detection.
- Use code, statistics, visuals, and markdown as needed.

2. Model Training

- Possible deliverable: notebook(s), python scripts
- Clean/process data as needed.
- Select features and train a model to classify whether a transaction is fraudulent (is_fraud).
- You may use any algorithms or techniques (justified by your findings).
- **Design your own train/test split** (don't use a test set derived from our internal holdout).
- Clearly document feature selection, algorithm choice, parameter tuning, and metric choice.
- Report and briefly interpret your model's performance (especially the implications of false positives/negatives in financial fraud).

3. Model Serving (API and Storage)

- Implement a REST API service in python (Utilizing Flask, FastAPI, or any framework is fine) that exposes at least two endpoints:
    - POST /predict: Accepts a transaction (in JSON or similar format, using the original features) and responds with a fraud prediction.
    - GET /frauds: Returns all transactions previously predicted as fraudulent (pulled from your own DB).
- **Important:**
  You must provide clear, detailed, and step-by-step instructions for running your code, from setting up the environment to executing data processing, model training, and evaluation. We will strictly follow your written instructions.

    - **If your instructions are unclear and we are unable to run your code, or if following your instructions as written results in failure, your score will be**

deducted accordingly.

- Use any DB technology (SQLite, Postgres, file system, etc.) to persist results. Ensure your service is runnable locally (README and instructions required).

4. System Architecture Scenario

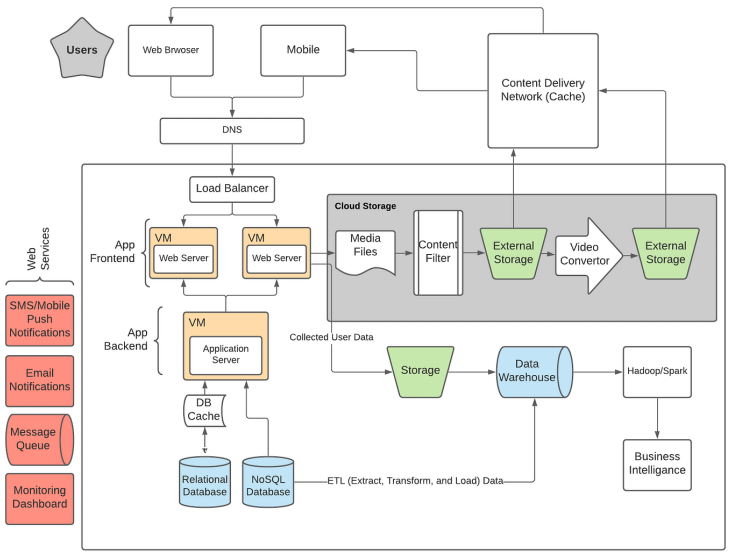
You are told:

- Transactions are streamed via **Kafka** (distributed streaming platform). Many financial services subscribe to transaction topics, including our fraud detection service.
- You are to design the **architecture** for the fraud detection system, which
    - Consumes transactions from Kafka
    - Predicts fraud and stores flagged transactions in a database
    - Provides a way for auditors to view flagged cases, investigate, and update their status

Deliverables:

- **Architecture Diagram** (.drawio/.png/.jpg/.pdf accepted), possibly showing:
    - Kafka integration
    - Fraud detection service (including model serving)
    - Database for fraudulent transactions
    - Application/interface for auditors to view/update cases
    - Any relevant considerations (latency, security, scaling, redundancy, etc.)
- You do not need to implement/code this part. An architecture diagram will suffice.

Example
Ref: Medium



Submission Instructions

1. You are **encouraged** to search the internet/books for reference, use publicly available libraries/docker images, or use AI tools (Claude, Gemini code assist, etc.) in this test. However, please do **not** clone other's solution to be used as yours. Cloned submission will be disqualified.
2. Organize all code (scripts, notebooks, service code, architecture diagram, and README) in a **public GitHub repository**.
3. Your README should:
    - Clearly explain dependencies, how to run the EDA, train your model, start the service, and test endpoints.
    - Include any architecture explanation or business recommendations.
        - **Important:** We will follow your README and instructions exactly as written. If they are unclear or if we are unable to run your work by following them, your score will be negatively affected.
4. Submit your GitHub repository link during HackerRank online test. Or email us if you've already done the HackerRank test.
5. If you are fresh college graduate, this test may overwhelm you. However, we encourage you to take your time, read tutorials, use AI tools, and learn how to complete the test. We don't expect a perfect score, but please submit your best. The content of this test reflects common work in our team.

Evaluation Criteria

Your solution will be evaluated on the following aspects:

Criteria	Weight
EDA & Data Understanding	10%
Modeling & Validation	10%
Code Quality & Documentation	30%
API & DB Implementation (runnable, correct)	30%
System Architecture Design (scaling, consistency, fault tolerance)	20%

Bonus:

Considerations for runtime/memory usage, security, deployment-readiness, and overall clarity of explanation.