Problem 05 — Application Logs ETL Pipeline

Client Scenario

A large software platform generates millions of log events daily across different microservices. The engineering analytics team wants to create an automated ETL (Extract-Transform-Load) pipeline that processes these logs, summarizes service performance, and detects anomalies in request latency and error patterns. Your task is to design a data engineering workflow that ingests, cleans, aggregates, and analyzes application logs.

Expected Outcome

You will build a pipeline that reads structured and semi-structured log data, cleans and normalizes timestamps, computes metrics per service, and develops a simple model to predict whether a service is at risk of latency degradation. The final outputs include transformed datasets, metric files, and a predictive model with performance reports.

Dataset Description

Files located under: datahack/p05/

- • logs.csv — contains synthetic log data from multiple microservices

logs.csv Columns:

- • service_name — name of the microservice
- • timestamp — time of the request event
- • latency_ms — time taken to process the request in milliseconds
- • status_code — HTTP status returned by the service
- • region — deployment region identifier

Artifacts and Folder Structure

All outputs must be stored in: artifacts_p05/

- Each step must generate both a data file and a metrics file.
- Step 1 → step1.parquet, step1_metrics.json
- Step 2 → step2.parquet, step2_metrics.json
- Step 3 → step3_model.joblib, step3_metrics.json

Step 1 — Easy: Log Cleaning and Daily Aggregation

Clean and normalize raw logs by removing null or malformed entries and converting timestamps into a standard datetime format. Aggregate logs by service and day, computing average latency, total requests, and error rate (percentage of 4xx and 5xx responses). Save the cleaned and aggregated dataset and record basic summary metrics.

Deliverables:

- • step1.parquet — cleaned and daily-aggregated logs
- • step1_metrics.json — contains {"services": count, "avg_latency": value}

Step 2 — Medium: Feature Engineering and Baseline Risk Model

Using the aggregated data, create features such as mean latency, error rate, and regional performance variability. Label services as 'high latency' if their mean latency exceeds the global median. Build a simple baseline classifier (such as Logistic Regression) to predict high-latency risk and evaluate using ROC AUC.

Deliverables:

- • step2.parquet — service-level feature table
- • step2_metrics.json — contains {"roc_auc": value}

Step 3 — Hard: Enhanced Model and Drift Validation

Train an improved model (e.g., Random Forest or Gradient Boosted Trees) to better classify high-latency services. Evaluate its accuracy using ROC AUC and analyze drift by comparing latency feature distributions between training and test sets using the KS statistic. Lower KS values indicate a more consistent model.

Deliverables:

- • step3_model.joblib — trained model file
- • step3_metrics.json — contains {"roc_auc": value, "ks_drift": value}

Team Collaboration and Submission

Teams must work collaboratively via shared Google Drive folders. Submit one Colab notebook (TeamName_Problem05.ipynb) along with the artifacts_p05 folder. The notebook should execute end-to-end without any manual adjustments or path edits.

AI Assistance Policy

Use of AI assistants (ChatGPT, Claude, Copilot, etc.) is permitted for explanation and code guidance. All AI assistance must be disclosed at the top of your notebook, specifying what was generated or suggested.

Presentation Requirement

Deliver a concise 5-minute presentation summarizing your ETL process, data transformations, and predictive model design. Discuss key metrics (latency, error rate, ROC AUC, KS drift) and their operational implications.

Organizer Testing Process

The judging script automatically validates file existence, correct naming, and valid JSON keys for metrics. Scores are assigned based on output correctness and performance.

Scoring Breakdown:

- • Step 1 — up to 30 points (log cleaning and metrics)
- • Step 2 — up to 35 points (feature design and ROC AUC)
- • Step 3 — up to 35 points (model accuracy and drift)

Environment Setup and Access Guide

Use Google Colab for all development. Mount Google Drive and ensure the same directory layout as provided:

- from google.colab import drive
- drive.mount('/content/drive')
- %cd /content/drive/MyDrive/Hackathon

Ensure DATA_DIR paths remain unchanged. Avoid absolute local paths or environment-specific variables.