THE SHARD

Take Home Assessment:

Credit Risk Classification Model and API

Analytics & Insights:

Objective

You are tasked with building a small end-to-end system that predicts the likelihood of a borrower defaulting on a loan (i.e., a credit risk classification problem). This project has both:

⇒ **A data science component:** data cleaning, feature engineering, model training, and model evaluation.

⇒ **A DevOps/deployment component:** containerizing your model inference service and exposing it via an API, plus a simple front-end.

At the end of the project, please create a **public GitHub repository** containing all your code, documentation, and instructions. Send us the link when you are done.

Timeline

The assignment must be completed **by 23:59 PM on January 29, 2025,** and the GitHub repository link should be submitted by this time by responding to this email and cc jamesg.wanjiku@theshard.co.za and moseli.motsoehli@theshard.co.za. Dataset Suggestions

Below are a few open-source datasets you can choose from. Feel free to select one (or any other relevant, publicly available credit/loan dataset):

⇒ **Lending Club Loan Data (Kaggle):**
  - https://www.kaggle.com/datasets/wordsforthewise/lending-club

⇒ **German Credit Data (UCI Machine Learning Repository):**
  - https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)

⇒ **Credit Card Fraud Detection (Kaggle):**
  - https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Choose a dataset that focuses on credit/default risk. If you prefer a different dataset, ensure it is **related to credit, finance, or loan data** and is public/open source.

Assignment Requirements

Data Preparation & Model Training:

⇒ **Data Cleaning:** Handle missing values, outliers, incorrect data types, etc.

⇒ **Feature Engineering:** Create or transform features to improve model performance (e.g., debt-to-income ratios, credit-utilization rate).

⇒ **Model Selection:** Train a classification model (e.g., Logistic Regression, Random Forest, XGBoost).

⇒ **Evaluation:** Use at least one appropriate metric (e.g., AUC, precision, recall, F1- score) to measure performance. Justify your choice of metric(s).

⇒ **Model Selection Justification:** Explain the rationale behind selecting the model, highlighting its advantages and disadvantages. Justify your choice by discussing how the model aligns with the specific requirements of the task.

Containerized API (Back-End):

⇒ **Create an Inference Service:** Expose a REST API endpoint using a framework of your choice (Flask, FastAPI, etc.).

⇒ **API Endpoint:** When given input data (JSON), the endpoint should return the predicted probability of default.

⇒ **Containerization (Optional):** Package your service in a Docker container. The Docker image should:
  - Contain all dependencies for running your model inference (e.g., Python libraries, model artifact).
  - Provide clear instructions in a Dockerfile or docker-compose file to build and run the container.

Simple Front-End:

⇒ **User Input:** Create a basic form or interface (e.g., using React, Vue, or a simple HTML/JS page).

⇒ **Call the API:** On form submission, send the input data to the API endpoint, and display the resulting default probability.

⇒ **Dockerize (Optional, but Preferred):** If possible, place this front-end in a separate container or provide instructions to run it locally.

Documentation:

⇒ **README:** Provide a clear README.md in your repository describing:

- Overview of the project.
- Steps to build and run each container (back-end and front-end).
- How to test the endpoint with sample data (e.g., using curl or a browser).

⇒ **Code Structure:** Organize your code (data prep, training scripts, app code) logically.

⇒ **Environment Requirements:** List any Python/R/Node.js dependencies in requirements.txt, conda environment.yml, or similar.

Submission:

⇒ **GitHub Repository:** Make it public. Include all code, Dockerfiles, environment files, and the README.

⇒ **Link Delivery:** Send us the link to your GitHub repo (e.g. (github.com/username/reponame)

Evaluation Criteria

Data Science Workflow:

⇒ Quality of data cleaning and feature engineering.

⇒ Choice of model and relevant hyperparameter tuning (if any).

⇒ Appropriate evaluation metrics.

Software Engineering & DevOps:

⇒ Clarity and maintainability of your code.

⇒ Properly functioning Docker containers.

⇒ Well-documented API and front-end setup.

User Experience:

⇒ Simplicity and usability of your front-end interface.

⇒ How well the front-end and back-end communicate (error handling, etc.).

Final Thoughts and Appreciation

Good luck! We look forward to reviewing your submission.

Take this opportunity to showcase your strengths. We encourage you to focus on the areas where you feel you excel the most. What matters most is demonstrating your thought process and skills.

Feel free to reach out if you have any questions or need clarification.