AI Engineer – Round 2 Assignment

(Two Mini-Projects)

This round contains **two independent ai integrated with backend mini-projects**. Students must implement **both** and submit a single GitHub repository or individual repositories with clear README.

Common Guidelines

- **Language/Stack**: Python 3.10+, FastAPI, Pydantic, Uvicorn.
- **Code Quality**: Modular structure, type hints, docstrings, comments.
- **Config**: Use .env with API Keys; do not hardcode secrets.
- **Notebooks**: Keep EDA/training in Jupyter Notebooks (.ipynb). Convert final logic into importable modules used by APIs.
- **Docs**: Clear README.md with setup, run, API usage, and how to retrain/regenerate artifacts.
- **Requirements**: Clear library requirements & versions in requirements.txt.

Submission

Provide a GitHub link(s) in the README add:

- Local run instructions
- Screenshots/plots in reports

Project A — AI-Powered Loan Eligibility & Risk Scoring System

Dataset Link - Here

Objective

Build an end-to-end system that trains and serves a model predicting **loan default risk** and exposes predictions via **FastAPI**.

Features to Implement

Data Science & ML

- Data preprocessing & cleaning
- Feature engineering (derived & selected features)
- Model training to predict default probability (risk score)
- Model evaluation and selection; persist best model

FastAPI Backend

- REST endpoints to:
  - Predict risk score from borrower features (e.g., income, loan amount, credit score)
  - Return model performance summary and feature importance
- Input validation with Pydantic; robust error handling
- Production-ready structure and documentation

Data Visualization (documented)

- Generate plots for feature distributions, correlations, class imbalance, risk segmentation, and model performance
- Summarize key dataset trends and high-impact variables

Evaluation Criteria

- Quality/robustness of preprocessing & feature engineering
- Model choice, training rigor, and evaluation depth
- API design, validation, and error handling quality
- Clarity/insight of visualizations and narrative
- Code quality and repo hygiene

Deliverables (GitHub)

- **Model File** – Serialized trained model (e.g., model.pkl or model.joblib)
- **FastAPI Backend** – Prediction and model-insights endpoints with validation & errors handled
- **Data & Modeling Notebook/Script** – Preprocessing, feature engineering, training, evaluation, visualization generation
- **Documentation & Visualizations** – README.md (setup, API usage, retraining/regen steps, dataset insights) + a charts / plots for patterns/trends/performance

Project B — AI-Powered PDF Context Retrieval Chatbot (RAG)

Objective

Build a backend that ingests **any PDF**, indexes it for **semantic search**, and answers queries via a **RAG pipeline** using FastAPI.

Features to Implement

Data Processing & Retrieval

- Accept a PDF upload, extract text, split into segments/chunks
- Generate embeddings per segment and store them in a **vector database** for fast semantic search

RAG Pipeline

- Implement a retrieval module to fetch the most relevant text from the given PDF for the given query
- Integrate with a Large Language Model (LLM) to generate context-aware answers.
- Support both direct answers and source context references in the output.

FastAPI Backend

- REST API endpoints to:
    - Upload PDF documents and index their content.
    - Accept a user query and return a context-based answer with references.
- Input validation and error handling with Pydantic models.
- Clean, modular, production-ready code with documentation.

Tech Stack

- Vector DB: **FAISS**, **Chroma**, or **Pinecone**
- **FastAPI** for REST API
- **LLM Integration** via provider API (e.g., OpenAI, Gemini, or compatible)

Evaluation Criteria

- Accuracy & relevance of retrieved context
- Efficiency and correctness of ingestion & indexing
- API design quality, validation, and error handling
- Code cleanliness, structure, and documentation

Deliverables (GitHub)

- **Serialized Embedding Index** – e.g., FAISS files or Chroma/Pinecone persistence
- **FastAPI Backend Source** – PDF ingestion and query endpoints
- **README.md** – Setup, run, API usage (upload & query), model/config notes