TekFriday - Development Role - Assignment Summary (Github)

Part A: Chatbot for Loan Terms

This module implements a **conversational loan assistant chatbot** using **Streamlit**. The goal was to build a simple, console-based or UI-based bot that can respond to user questions about basic loan-related terms and concepts. To enhance functionality and realism, we went beyond the minimum requirements and developed a fully interactive Streamlit-based chatbot UI.

Objective:

- Greets the user
- Accepts a question in natural language (e.g., *"What is EMI?"*)
- Returns a predefined answer based on a knowledge base
- Responds correctly even if the input varies slightly in casing or spacing

Key Features Implemented

- **Streamlit-Based UI**:
    - We created a web-based chatbot interface using Streamlit, styled to look like a modern chat application.
    - Chat messages are stored in `st.session_state`, maintaining the full chat history during interaction.
- **50+ Predefined Responses**:
    - A dictionary of over **50 loan-related FAQs** was created, covering everything from *"What is EMI?"* to *"What is CIBIL score?"*, *"What is a top-up loan?"*, and *"What is foreclosure?"*
    - These answers were generated and validated using **ChatGPT**, ensuring coverage of relevant financial concepts.
- **Input Normalization**:
    - All user inputs are processed through a `clean_input()` function that:
        - Converts text to lowercase
        - Removes punctuation
        - Normalizes multiple spaces
    - This ensures that inputs like `"  WHAT is emi??  "` are matched correctly to `"what is emi"`.
- **User-Friendly Chat Interface**:
    - Input box placed at the bottom of the screen (like WhatsApp)

    - Chat bubbles for both user and bot messages with color distinction
    - Auto-updating message thread with scrollable history

- **Greeting Mechanism**:
    - On app start, the bot opens with a greeting message like:
      "Hello! I'm your Loan Assistant. You can ask me anything about loan terms."

Tech Stack :

- **Language**: Python
- **Frontend**: Streamlit
- **Logic**: Dictionary-based response system
- **Message Memory**: `st.session_state` for preserving full chat context

Files Submitted

- `part_A.py`: Streamlit chatbot with 50+ FAQs
- (Optional) `chatbot_console.py`: Command-line version (not required, but implemented for completeness)

Value Addition

This chatbot simulates a **real-world banking assistant**, and could be extended to:

- Use NLP models for dynamic responses
- Log unanswered queries for future learning
- Integrate with a database of terms or backend API
- Realistic chatbot UI with session-based message history

Part B: Loan Risk Calculator

This module implements a loan risk scoring mechanism that classifies loans into **LOW**, **MEDIUM**, or **HIGH** risk categories based on the borrower's repayment behavior and loan details.

Objective

To develop a Python-based solution that:

- Calculates a **numerical risk score** using borrower and loan data
- Classifies each loan into a risk category:
  - **LOW**: Score < 15
  - **MEDIUM**: 15 ≤ Score ≤ 25
  - **HIGH**: Score > 25

The goal is to simulate how banks or NBFCs might assess loan default probability based on simple heuristics.

Risk Score Formula Used

```
risk_score = (missed_repayments * 2) + (loan_amount / collateral_value) + (interest / 2)
```

This formula combines behavioral and financial risk indicators:

- `missed_repayments`: Higher missed payments indicate unreliability
- `loan_amount / collateral_value`: Represents exposure risk
- `interest`: Higher interest can increase EMI burden and default chances

Enhancements Made

- **Data Scaling for Realism**:
  - The original values of `missed_repayments` were very large (often in lakhs), making all scores unrealistically high.
  - To align with the provided classification thresholds (15, 25), we **scaled `missed_repayments` by 1000** for meaningful scoring.
- **Separation of Logic**:
  - Score calculation was done using a function: `calculate_risk_score(row)`
  - Classification was applied using a separate function: `classify_loan_risk(score)`
  - This modular approach makes the code clean and reusable.
- **New Columns Added**:

    - `risk_score`: Numeric score based on the formula
    - `risk_level`: Category assigned based on score

<u>Files Submitted</u>

- `part_B.ipynb`: Fully annotated notebook
    - Loads `main_loan_base.csv`
    - Applies risk scoring and tagging
    - Displays sample of 10 randomly selected loans for output variety

Part C: EMI Risk Tagging

This module automates the process of tagging loan records with a risk category by applying a custom risk score formula across the entire dataset using Pandas' `.apply()` function.

Objective

To build a function `classify_risk(row)` that:

- Calculates a risk score using specific loan-related fields
- Classifies each loan into **LOW**, **MEDIUM**, or **HIGH** risk
- Adds a new column `risk_level` to the full dataset

This part essentially scales up the logic from Part B to a full dataset, demonstrating how business logic can be embedded directly into data processing pipelines.

Risk Score Formula

The same formula from Part B is used:

risk_score = (missed_repayments * 2) + (loan_amount / collateral_value) + (interest / 2)

But with the critical adjustment:

- `missed_repayments` is **divided by 1000** to make the risk score compatible with classification thresholds.

Implementation Highlights

- **Two-Step Function Pipeline**:
  - Step 1: `calculate_risk_score(row)` — computes the risk score
  - Step 2: `classify_risk(score)` — assigns a category based on the score
- **Applied at Scale**:
  - The logic is applied to every row in `main_loan_base.csv` using `.apply()` for risk score
  - Followed by `.apply()` on the score column to determine `risk_level`
- **New Columns Added**:
  - `risk_score`: A numeric score for each loan
  - `risk_level`: One of `LOW`, `MEDIUM`, or `HIGH`
- **Output Variety**:

    - A `.sample(10)` was used to display random rows and ensure diverse classification results, making the output visually meaningful

Tech Stack

- Language: Python
- Tools: Pandas
- Output Format: Jupyter Notebook (`part_C.ipynb`)

Files Submitted

- `part_C.ipynb`: Contains:
    - Data loading
    - Score and risk tagging logic
    - Randomized sample output with `risk_score` and `risk_level` columns

Repayment Behavior Analysis

In this analysis, `repayment_base.csv` was joined with `main_loan_base.csv` using `loan_acc_num`. We calculated:

- **Total repaid amount**
- **Repayment ratio** = total repaid / loan amount
- A binary flag **`is_partially_repaid`** for loans where repayment was less than 75%

This provides a quick view of borrower repayment behavior and identifies under-recovered loans.

Assumptions Made

• Only `main_loan_base.csv` was used for Part B and C unless otherwise stated.
• `missed_repayments` were too large for the given thresholds, so they were scaled down by 1000.
• Test datasets were not used for training or evaluation, assuming they are for internal validation.
• Additional datasets like `monthly_balance_base.csv` and `repayment_base.csv` were explored optionally.
• AI-generated responses for the chatbot were generated using OpenAI's ChatGPT.