Problem Statement 1

1. API Data Retrieval and Storage (Books API)

Assumptions

- API returns JSON in the format:
  `{ "books": [{ "title": "", "author": "", "year": 2020 }] }`

- API is publicly accessible and does not require authentication.

Approach

1. Use `requests` to fetch data from the REST API.
2. Parse the JSON response.
3. Create a SQLite database using `sqlite3`.
4. Create a `books` table with columns: `title`, `author`, `publication_year`.
5. Insert records using parameterized queries.
6. Fetch and display stored data using SQL `SELECT`.

Why this works

- SQLite is lightweight and sufficient for local storage.
- Parameterized queries prevent SQL injection.
- Clear separation of API handling and database logic.

2. Data Processing and Visualization (Student Scores)

Assumptions

- API returns JSON like:
  `{ "students": [{ "name": "A", "score": 85 }] }`

Approach

1. Fetch data from the API using `requests`.
2. Extract scores and compute the average using Python.
3. Use `matplotlib` to generate a bar chart:
    - X-axis: Student names
    - Y-axis: Scores
4. Display the average score as a reference line.

Reasoning

- Keeps computation logic simple and readable.
- Visualization helps quickly identify performance differences.
- Suitable for exploratory analysis.

3. CSV Data Import to SQLite Database

Assumptions

- CSV columns: `name,email`
- File size is manageable (fits in memory).

Approach

1. Read CSV using Python's `csv` module or `pandas`.
2. Validate required fields.
3. Create a `users` table in SQLite.
4. Insert rows using batch inserts for efficiency.

5. Commit changes and close the connection.

Why SQLite

- Easy setup.
- No external dependency.
- Ideal for scripting and testing.

Most Complex Python Code :
https://github.com/bhoomika1714/LlamaMeetsRag/blob/main/Long_document_summarizer.ipynb

Most Complex Database Code:
https://github.com/bhoomika1714/job-preference-matching-system/tree/main/backend/src/models

1. Self-Rating

Area	Rating
LLM	**B**
Deep Learning	**B**
Machine Learning	**B**
AI (overall)	**B**

Reasoning
I can implement models, pipelines, and experiments under guidance, understand architectures, and debug issues. I'm actively strengthening fundamentals to reach independent proficiency.

2. Key Architectural Components of an LLM-based Chatbot

High-Level Architecture

1. **User Interface** – Web or chat UI
2. **Backend API** – Handles requests and responses
3. **Prompt Engineering Layer** – Structures user input

4. **LLM Engine** – GPT / open-source LLM
5. **Context Management** – Maintains conversation history
6. **Vector Database (optional)** – For retrieval-augmented generation (RAG)
7. **Post-processing Layer** – Filters, formats output

Approach

- User input → backend
- Relevant context retrieved (if using RAG)
- Prompt sent to LLM
- Response generated and returned to user

Findout my Similar Kind of Project : https://github.com/bhoomika1714/LlamaMeetsRag

3. Vector Databases

What is a Vector Database?
A vector database stores embeddings (numerical representations of data) and allows fast similarity search using distance metrics like cosine similarity.

Why they matter

- Enable semantic search
- Core component of RAG-based systems
- Efficient for large unstructured datasets

Hypothetical Problem

Build a chatbot that answers questions from internal technical documentation.

Chosen Vector Database: `FAISS`

Why FAISS

- Fast and efficient for similarity search

- Easy to integrate with Python
- Good for local or small-scale systems
- Suitable for experimentation and prototyping

Alternatives (if scaling)

- Pinecone → managed, scalable
- Weaviate → schema + metadata support

These responses reflect my current hands-on understanding and learning trajectory. I focus on strong fundamentals, clean implementation, and practical system design rather than surface-level abstraction.

Please find my attached AIML Projects at : https://github.com/bhoomika1714

- **Flood-victim detection model using YOLO11** – handled data preprocessing, image annotations, model testing, and evaluating output accuracy.
- **Concept-aware summarization pipeline using LLaMA + RAG** – involved prompt engineering, dataset preparation, hallucination checks, and iterative model evaluation.
- **Medical imaging analysis and CV workflows** – worked with multi-modal datasets, validation routines, and error analysis loops.

Updated Resume Link

https://drive.google.com/file/d/1LVac32ZGZTu2iXj4Eaz-lemv6tYR-6FF/view?usp=sharing