Assignment 1

Assumptions Made

1. Public APIs were used to fetch sample data for demonstration purposes.
2. The APIs return data in JSON format.
3. For student score visualization, actual test scores were not available, so sample scores were assumed.
4. The CSV file used for data import is available locally and contains valid user information.
5. SQLite was used as the database since it is lightweight and suitable for small-scale applications and assignments.

Problem 1: API Data Retrieval and Storage

In this task, I fetched book details from a public API using Python. The API returns data in JSON format. From the response, I extracted the book title, author, and publication year and stored them in a local SQLite database. After storing the data, I retrieved it from the database and displayed it.

Problem 2: Data Processing and Visualization

In this task, student data was fetched from a public API. Since test scores were not available, I assumed scores for demonstration purposes. I calculated the average score using Python and visualized the student scores using a bar chart.

Problem 3: CSV Data Import

In this task, I read user data from a CSV file and inserted it into a SQLite database. The data was then retrieved and displayed to verify successful insertion.

Link of GitHub repository - https://github.com/gaurav5268/api.git

4. link for python code - https://github.com/gaurav5268/Medical_Assistance

Assignment 2

1. Self-Assessment

I would rate myself as

Artificial Intelligence	B
Machine Learning	B
Deep Learning	B
Large Language Models	B

I have a good understanding of core concepts and have applied them in hands-on projects such as chatbots, machine learning models, and API-based systems. I am comfortable implementing solutions with guidance or reference material.

2. High-Level Architecture of an LLM-Based Chatbot

An LLM-based chatbot is designed to understand user queries and generate meaningful responses using a large language model.

At a high level, the chatbot consists of several interconnected components that work together to process user input and produce accurate answers.

User Interface

v

Backend / API Layer

v

Prompts

v

Retrieval System (RAG – Vector Database)

v

Large Language Model (LLM)

v

Response Generation

v

User Interface (Response Displayed)

The process starts with a **user interface**, where the user types a query.
Then query is sent to the **backend system**, backend prepares the user input by formatting it into a **prompt**. This prompt may include system instructions, conversation history, or rules that guide the model's behaviour.
If required, the system retrieves related information from a vector database using a Retrieval-Augmented Generation (**RAG)** approach.
The prepared prompt is then sent to the Large Language Model (**LLM**), which generates a response.
Finally, the response is displayed back to the user.

3. Vector Databases and Their Use in AI Systems

A vector database is used to store embeddings, which are numerical representations of data such as text or documents. It helps in finding similar information based on meaning rather than exact words.

For example, in a customer support chatbot that answers questions from company documents and FAQs, all documents are converted into embeddings and stored in a vector database.

For this use case, I would choose **FAISS** because it is open-source, fast, and easy to use with Python. It is suitable for small to medium-scale applications.