1. Problem Statement

Our business teams have access to critical data spread across multiple CSV files. Currently, analyzing this data requires manual effort from data analysts who write SQL queries to answer business questions. This process is slow and creates a bottleneck.

We want you to build a proof-of-concept (PoC) **Text-to-SQL system**. This system will take a natural language question from a user, interpret it, generate the appropriate SQL query to run against our data, execute it, and provide a clear, natural language response.

2. The Dataset

You will be provided with a set of mock CSV files representing our business data. These files contain two primary categories of information: **Financials** and **Operations**.

- **Financial Data:** Includes standard quarterly statements reflecting the company's performance (e.g., revenue, net income) and financial position (e.g., assets, liabilities). It also contains key annual performance metrics.
- **Operational Data:** Contains information on cargo volumes handled at various international ports over time.

Your first step will be to explore these files, understand their structure and relationships, and design a logical database schema. You must then load the data into a local **SQLite** database based on the schema you design. The clarity and logic of your schema will be a key part of the evaluation.

3. Core Requirements

Your solution must be an end-to-end system that accomplishes the following four steps:

1. **Data Ingestion & Schema Design:** Explore the provided CSVs, design a relational schema in a local **SQLite** database, and write a script to populate it.
2. **Natural Language to SQL (NL-to-SQL):** This is the core AI component. The system must accept a natural language question as a string input and convert it into a valid SQL query that can be executed on your database.
3. **Query Execution:** The system must execute the generated SQL query against the database to fetch the results.

4. **Result to Natural Language (SQL-to-NL):** The system should not just return the raw data (e.g., a table or list of tuples). It must synthesize the results into a concise, human-readable, natural language sentence or paragraph.

4. Expected System Capabilities

Your system should be robust enough to handle a variety of business questions. While we are not providing a list of test questions, you should design for the following capabilities:

- **Simple Data Retrieval:** Answering questions about specific data points.
- **Aggregations & Grouping:** Calculating sums, averages, and counts, often grouped by categories like time or location.
- **Cross-Domain Analysis:** Answering questions that require combining financial and operational data (e.g., calculating financial metrics per unit of operational output).
- **Comparative Analysis:** Comparing figures across different time periods (e.g., quarter-over-quarter or year-over-year).
- **Graceful Failure:** The system should recognize questions that are outside the scope of the provided data and respond appropriately (e.g., "I'm sorry, I can only answer questions about company finance and cargo operations.").

5. Technical Stack & Approach

- **Language: Python 3.x** is required.
- **Database: SQLite**.
- **AI/ML Approach:** You have freedom here. We encourage you to use modern Large Language Model (LLM) techniques.
    - **Recommended:** Using an LLM framework like **LangChain** or **LlamaIndex** is a great choice, as they are well-suited for this type of application.
    - **Alternative:** You could directly use an API from a model provider like **OpenAI**, **Cohere**, or use an open-source model via a library like **Hugging Face Transformers**.
    - Clearly document your choice of model and prompting strategy. You are not required to fine-tune a model, but your prompt engineering will be a key point of evaluation.
- **Code Structure:** We recommend wrapping your logic in a simple API using a web framework like **FastAPI** or **Flask**, but a well-structured command-line interface (CLI) is also acceptable.

6. Evaluation Criteria

We will evaluate your submission based on the following criteria:

1. **Correctness & Functionality:** Does the system work? Does it consistently generate accurate SQL queries and coherent natural language responses for plausible business questions?
2. **Database Schema Design:** The logic, clarity, and efficiency of the database schema you design to represent the relationships in the raw data.
3. **Code Quality:** Is the code clean, modular, well-commented, and easy to understand? Does it follow standard Python conventions (PEP 8)?
4. **Design & Approach:** Your reasoning behind the chosen architecture, model, and prompting techniques. Why did you make certain design trade-offs?
5. **Robustness:** How well does the system handle edge cases, errors, and questions outside its scope?
6. **Documentation:** The quality and clarity of your `README.md` file is critical.

7. Deliverables

Please provide the following in a single submission:

1. **Source Code:** A link to a Git repository (e.g., on GitHub or GitLab).
2. **`README.md` file:** This is crucial. Your README should include:
    a. A brief overview of your project and approach.
    b. **Clear setup and running instructions** (e.g., `pip install -r requirements.txt`, `python main.py`).
    c. A list of dependencies (`requirements.txt`).
    d. A section on your **Database Schema**, explaining the tables you created and the relationships between them.
    e. A section on your **Design Choices**, explaining the model you used, your prompting strategy, and why you chose your architecture.
    f. Any **Limitations** or known issues with your implementation.

Please do not hesitate to reach out to us.

We are excited to see what you build. Good luck