Building an LLM-Powered Booking Analytics & QA System

Objective

Develop a system that processes hotel booking data, extracts insights, and enables retrieval-augmented question answering (RAG). The system should provide analytics as mentioned in below sections and answer user queries about the data.

Deliverables

1. Data Collection & Preprocessing

- Use a sample dataset of booking records (CSV, JSON, or database). You can use this dataset (Sample Dataset | Kaggle Dataset), or some other relevant dataset of your choice.
- Ensure that if you choose your own custom dataset, it has the fields that would be needed to generate the analytics or insights mentioned in below sections.
- Implement data cleaning (handle missing values, format inconsistencies, etc.), wherver necessary.
- Feel free to store data in a structured format.

2. Analytics & Reporting

- Implement the following analytics:
  - Revenue trends over time.
  - Cancellation rate as percentage of total bookings
  - Geographical distribution of users doing the bookings.
  - Booking Lead time distribution.
  - Additional Analytics if any
- Generate these insights using Python (You are free to use any of pandas, NumPy, Matplotlib, seaborn, SQL, etc.).

3. Retrieval-Augmented Question Answering (RAG)

- Use FAISS, ChromaDB, Weaviate, or similar vector databases to store vector embeddings of booking data.
- Implement natural language Q&A using an open-source LLM (Llama 2, Falcon, Mistral, GPT-Neo, or any other) with RAG.

- Example Questions:
    - "Show me total revenue for July 2017."
    - "Which locations had the highest booking cancellations?"
    - "What is the average price of a hotel booking?"

4. API Development

- Build a REST API using Django, FastAPI, or Flask.
- Required endpoints:
    - `POST /analytics` → Returns analytics reports.
    - `POST /ask` → Answers booking-related questions.

5. Performance Evaluation

- Evaluate the accuracy of Q&A responses.
- Measure API response time and optimize retrieval speed.

6. Deployment & Submission

- Package the solution with a README (setup instructions).
- Provide a GitHub repo with:
    - Codebase (LLM integration, analytics, API).
    - Sample test queries & their expected answers.
    - Short report explaining implementation choices & challenges.

Bonus (Optional)

- Implement real-time data updates using a database (e.g., SQLite, PostgreSQL). That is as and when new data or records are added to the database, system should be able to reflect those.
- Add query history tracking for questions that have been asked of the system.
- You can add a new API endpoint which can check the health of the system by internally checking the dependencies statuses. `API : GET /health` → Checks system status.

Good luck! Remember to approach the problem in parts, and reach out to us in case of any questions.