Implementation Report – Buyogo AI/ML Internship Assignment

Project Title: LLM-Powered Hotel Booking Analytics & RAG-Based Question Answering System
Candidate: Vikas Srinivasa
Technology Stack: FastAPI, SQLAlchemy, Pandas, Weaviate, Ollama (LLaMA3), RAGAS, Langchain, OpenAI, PyPDF2

Objective

The objective of this project was to build a backend system capable of:

1. Analyzing hotel booking data and serving insights through a REST API.
2. Providing a natural language interface for querying booking analytics via Retrieval-Augmented Generation (RAG).
3. Evaluating the quality of generated responses using LLM-based faithfulness metrics.

Implementation Overview

1. Data Preprocessing

- Utilized the publicly available hotel_bookings.csv dataset from Kaggle.
- Performed data cleaning including handling missing values, formatting inconsistencies, and standardizing date fields.
- Loaded the cleaned dataset into a PostgreSQL database using SQLAlchemy for structured access and querying.

2. Analytics and Reporting

Developed multiple analytics endpoints grouped by topic:

- Revenue trends over time, by country, hotel type, and room type.
- Cancellation rates segmented by hotel, country, room type, lead time, market segment, etc.
- Geographical distribution of bookings by volume and revenue.
- Lead time distribution patterns.
- Additional metrics such as correlation between cancellations and deposit type or special requests.

These insights are computed through efficient SQL queries. The system uses query caching to recompute results only when new data is added to the database.

3. Query Caching

- All analytics endpoints use a custom AnalyticsCache class, which checks if the database has changed since the last computation. If not, cached results are returned.
- The /ask endpoint also uses caching. It stores previously asked queries and their responses (along with evaluation scores) in a query_history table. If a user repeats a query, the system returns the stored answer immediately.

4. Retrieval-Augmented Generation (RAG) using Weaviate

- Parsed structured insights from PDF documents using PyPDF2, extracting and indexing contextually relevant lines.
- Embedded the extracted text into Weaviate using nomic-embed-text, and connected it with a local LLM (llama3.2 via Ollama).
- Used Weaviate's near_text and generate methods to retrieve contextually similar answers and generate final responses.

5. Evaluation with RAGAS

- Integrated the faithfulness metric from RAGAS to assess alignment between generated answers and reference answers (if provided).
- Used Langchain wrappers for OpenAI LLM and embedding models to support evaluation.
- Evaluation scores are stored in the database for transparency and future refinement.

6. API Architecture

Implemented a clean REST API using FastAPI:

Endpoint	**Description**
POST /analytics/...	Returns structured analytics based on topic
POST /ask	Accepts natural language questions and generates contextual answers
(Optional) GET /health	*Not implemented*

Bonus Features

1. Real-Time Data Responsiveness

Implemented. The system detects updates to the database and only recomputes analytics when new data is added, ensuring near-real-time accuracy without unnecessary processing.

2. Query History Tracking

Implemented. All queries made to the /ask endpoint are stored along with their answers and evaluation scores in the database. This supports reuse, caching, and tracking of model performance over time.

3. Health Check API

Not implemented. The optional /health endpoint for system diagnostics was not included, but can be added if required.

Challenges and Resolutions

Challenge	Resolution
Setting up local infrastructure for LLM and vector search	Used host.docker.internal to bridge communication between Dockerized services and the host.
Preventing duplicate indexing in Weaviate	Checked collection size before inserting objects to avoid redundant embeddings.
Writing portable and optimized SQL queries	Used PostgreSQL-compatible functions such as DATE_TRUNC, CASE, ROUND, and aggregate filters.
Parsing and extracting structured data from PDFs	Used PyPDF2 to extract and clean up relevant content based on line formatting heuristics.

Example Use Cases

User Query	Response Summary
What was the revenue for July 2017?	Computed total revenue for that month from SQL data
Which countries had the highest cancellations?	Ranked list of countries by cancellation volume

User Query	Response Summary
What is the average lead time for bookings?	Average computed directly from DB records

---

Conclusion

The system meets all core and most bonus requirements of the assignment:

- Performs data preprocessing and cleaning
- Generates structured analytics across multiple booking dimensions
- Supports semantic question-answering over unstructured PDF data
- Uses a self-hosted RAG pipeline with LLM-generated responses
- Implements caching and evaluation for performance and quality