LumenAI: A Conversational Analytics Assistant for Intelligent Database Analysis

Milestone 2 Report | MVP with LLM & RAG

Mohamed Kassem | mok206@g.harvard.edu, **Erich Gunsenheimer** | erg480@g.harvard.edu

Overview

This milestone focused on developing a minimum viable product (MVP) that leverages Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) to create an interactive text-to-SQL application. The system loads sample data into a SQLite database, allowing users to ask natural language questions about the data. The application then uses an LLM to generate corresponding SQL queries, executes these queries on the database, and finally produces an analysis of the results. The process creates a chatbot-like loop for follow-up queries.

Key Components

1. Data Processing and SQLite Database Creation

- Sample Data Source: We used sample data from Kaggle's AdventureWorks dataset. This dataset, originally in CSV format, was used for testing the application.
- Data Loader Pipeline: A dedicated data_loader application (containerized separately) is responsible for reading CSV files from a specified folder, processing them, and constructing a SQLite database. This database serves as the backend on which SQL queries are executed.
- Data Versioning: Since the application is designed for users to upload their own CSV files at runtime, no formal data versioning (e.g., using DVC) was applied. The uploaded data is transient and intended solely for interactive querying.

2. LLM Integration and RAG Workflow

- LLM for Query Generation: Two LLMs were evaluated during development. Initially, a locally downloaded Deepseek model was used. However, it generated only basic SQL queries and most of the time generated errors. Subsequently, we switched to using the GPT-based "o3-mini" model (via OpenAI) which proved to be significantly more robust in both generating SQL queries and providing comprehensive analyses.
- Prompt Engineering and Fine-Tuning: The application employs prompt fine-tuning to guide the model. A developer message and example are provided in the prompt to ensure that the model generates correct SQL syntax based on the database schema.

- Three-Level RAG Framework: The RAG workflow is implemented using the LlamaIndex framework. The framework operates at three levels:
- Basic Schema Generation: The application extracts the database schema from SQLite and creates a Document for each table.
- Schema with Simple Column Summaries: For each table, simple data summaries (e.g., minimum, maximum, counts) for each column are generated and appended to the schema. This provides additional context to the LLM for query generation.
- Advanced Column Summaries via LLM: In the most computationally expensive level, the LLM is also used to generate detailed summaries for each column. These summaries (which describe typical values, purpose, etc.) are fed into the RAG pipeline to ensure that only the most relevant pieces of information are passed to the final query generation prompt.

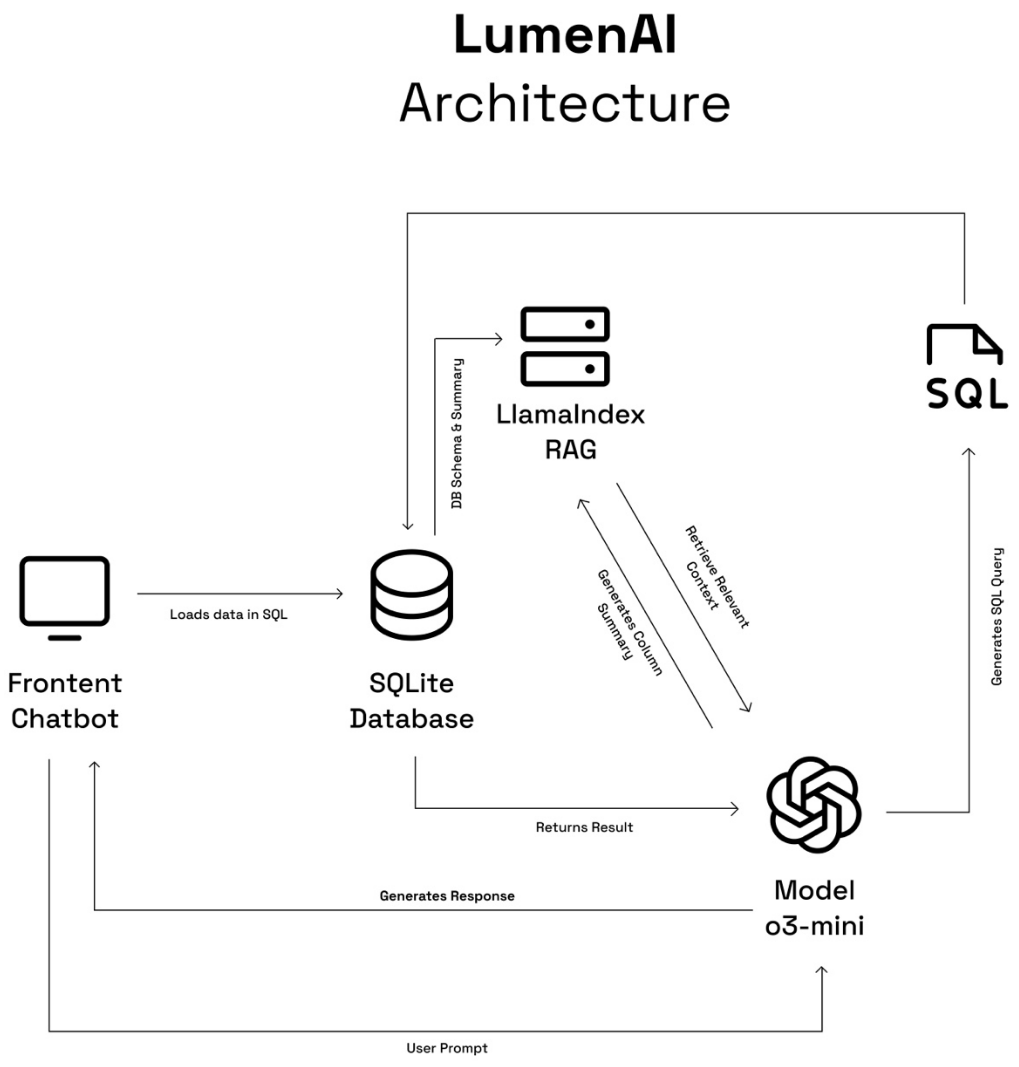
The RAG framework effectively chunks and retrieves the most pertinent information from the combined schema and column summaries before generating the final SQL query.

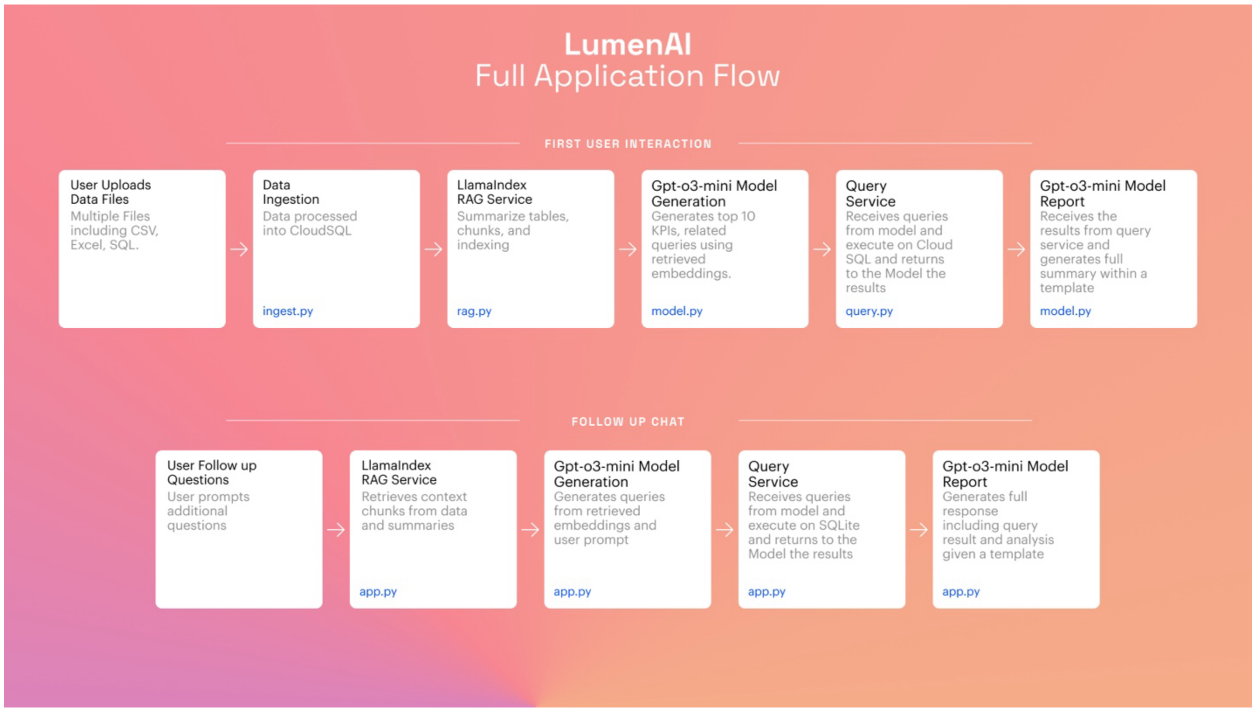
3. Containerization

- Atomic Containers: The entire application is containerized using Docker. We used a Dockerfile to build the main application image and a shell script (dockershell.sh) to run the container interactively.
- Docker Setup: Containerization ensures that the environment is reproducible and isolated. All dependencies are managed via Pipenv, with Docker handling the packaging and execution of the application.

4. Application Flow

- User Interaction: Users interact with the application via a command-line interface (CLI). They enter natural language questions about the data, such as “What is the count of records in AdventureWorks_Sales_2015?”.
- Query Generation and Execution: The LLM processes the prompt and generates a SQL query. This query is executed on the SQLite database, and the result set is returned.
- Analysis and Chatbot Loop: The output of the SQL query is fed back to the LLM along with the original user query and schema details. The model then generates an analysis that explains the output in context. This design supports a conversational loop where users can ask follow-up questions.


LumenAI
Architecture
Frontent Chatbot
Loads data in SQL
SQLite Database
DB Schema & Summary
LlamaIndex RAG
Retrieve Relevant Context
Generates Column Summary
SQL
Generates SQL Query
Returns Result
Model o3-mini
Generates Response
User Prompt



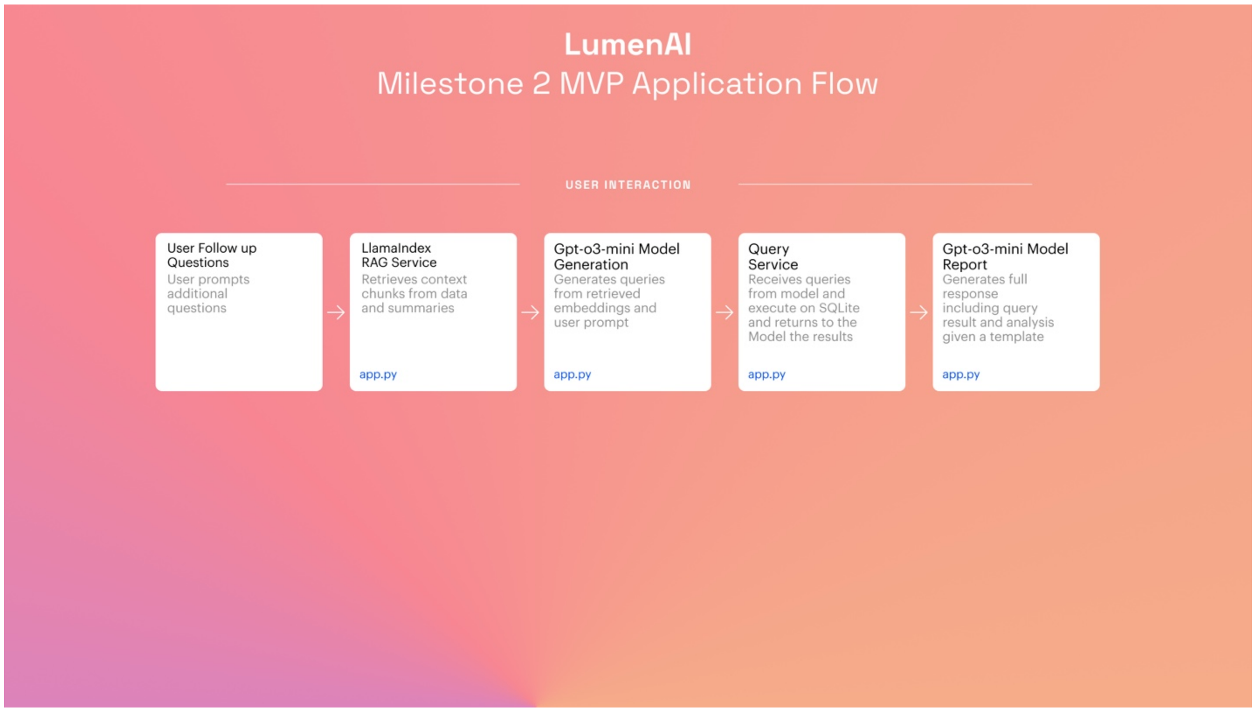
LumenAI
Full Application Flow

FIRST USER INTERACTION

User Uploads Data Files Multiple Files including CSV, Excel, SQL.
→ **Data Ingestion** Data processed into CloudSQL — ingest.py
→ **LlamaIndex RAG Service** Summarize tables, chunks, and indexing — rag.py
→ **Gpt-o3-mini Model Generation** Generates top 10 KPIs, related queries using retrieved embeddings. — model.py
→ **Query Service** Receives queries from model and execute on Cloud SQL and returns to the Model the results — query.py
→ **Gpt-o3-mini Model Report** Receives the results from query service and generates full summary within a template — model.py

FOLLOW UP CHAT

User Follow up Questions User prompts additional questions
→ **LlamaIndex RAG Service** Retrieves context chunks from data and summaries — app.py
→ **Gpt-o3-mini Model Generation** Generates queries from retrieved embeddings and user prompt — app.py
→ **Query Service** Receives queries from model and execute on SQLite and returns to the Model the results — app.py
→ **Gpt-o3-mini Model Report** Generates full response including query result and analysis given a template — app.py



LumenAI
Milestone 2 MVP Application Flow

USER INTERACTION

User Follow up Questions User prompts additional questions
→ **LlamaIndex RAG Service** Retrieves context chunks from data and summaries — app.py
→ **Gpt-o3-mini Model Generation** Generates queries from retrieved embeddings and user prompt — app.py
→ **Query Service** Receives queries from model and execute on SQLite and returns to the Model the results — app.py
→ **Gpt-o3-mini Model Report** Generates full response including query result and analysis given a template — app.py

Evaluation and Next Steps

Evaluation

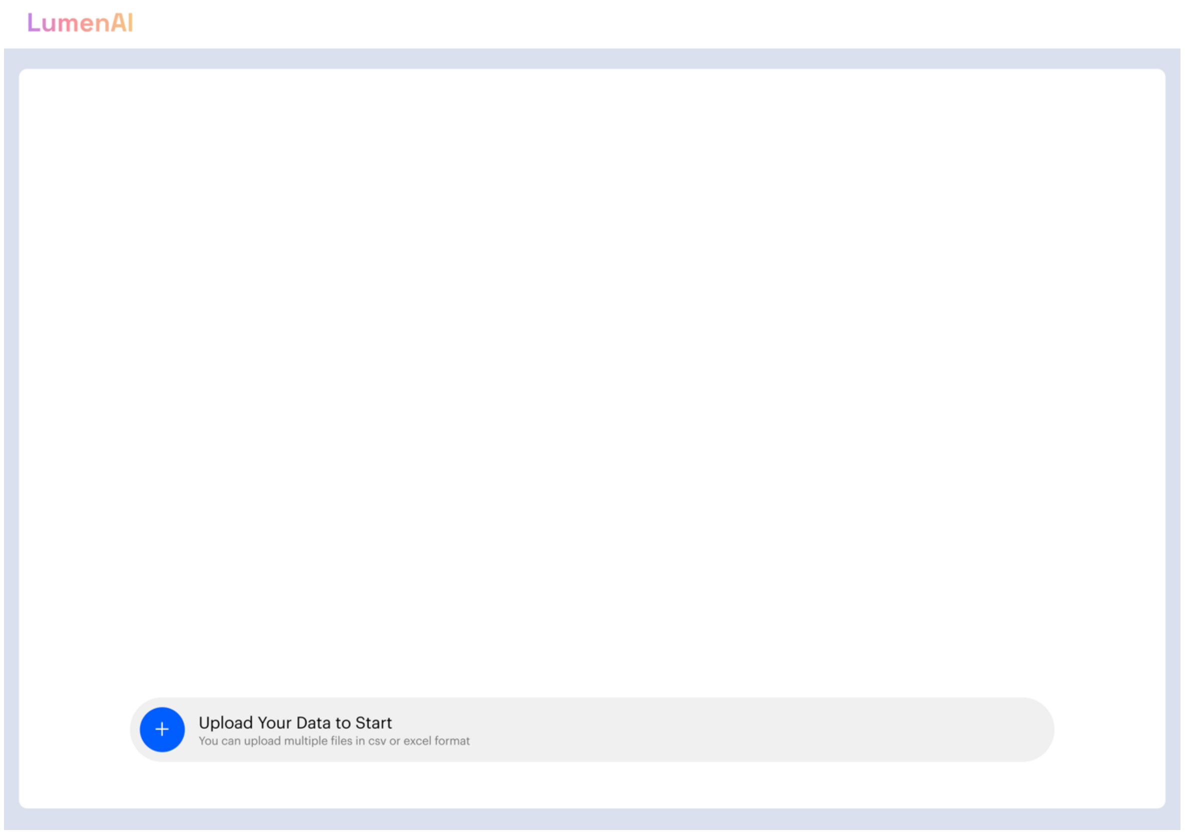
- Model Performance: The initial Deepseek model yielded weak results with basic SQL queries. Transitioning to the GPT-based “o3-mini” model improved both query generation and analysis, producing robust results.
- Prompt Engineering: Fine-tuned prompts, which include developer instructions and example queries, significantly improved output quality.
- RAG Efficacy: The three-level RAG framework (schema generation, simple column summaries, and advanced column summaries) has allowed the model to select the most relevant information dynamically, thereby enhancing query accuracy.

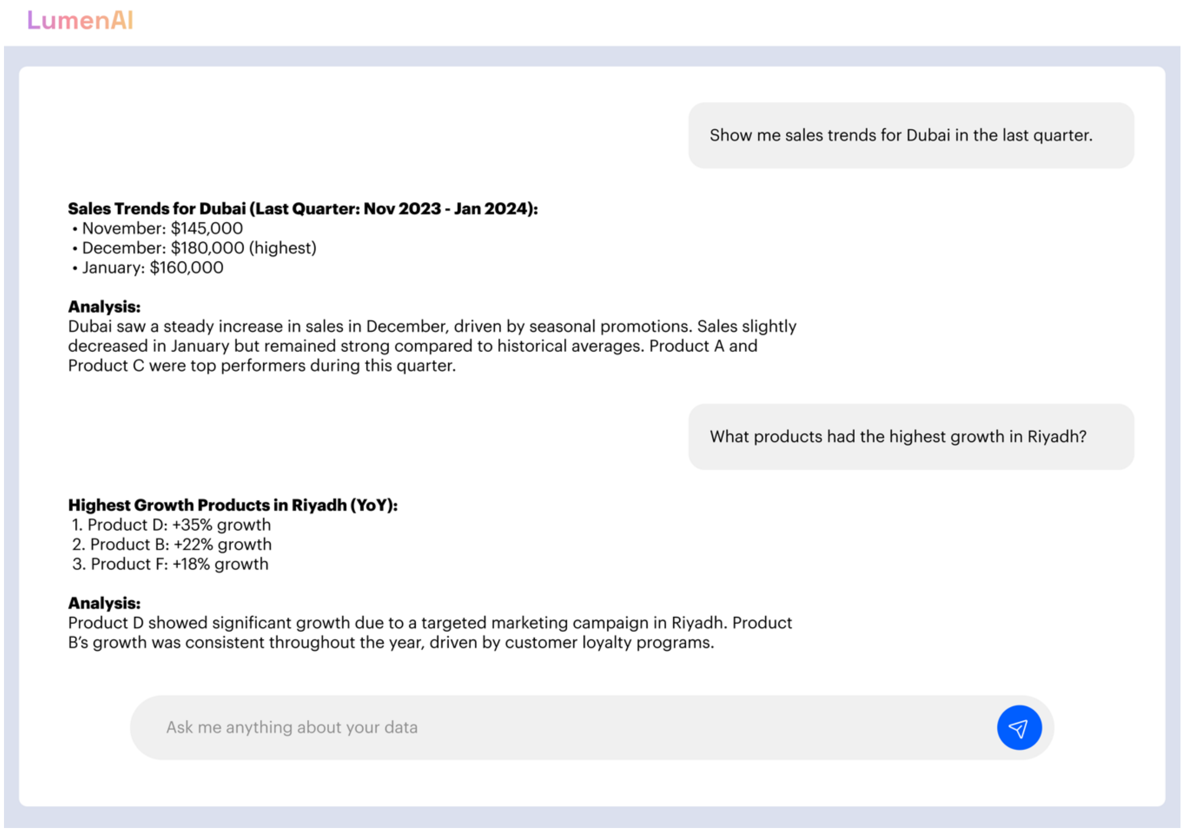
Model Comparison and Computational Costs

This submission includes four models to illustrate the development and optimization process:
1. **Basic Locally-Run Deepseek Model:**
- Performed basic SQL queries but was prone to generating errors.
2. **Each Model with RAG Framework Steps:**
- Implemented three levels of RAG framework:
  - **LumenAI** | Basic Schema Generation: Extracted database schema from SQLite.
  - **LumenAI Mini** | Basic Summary: Generated simple data summaries (minimum, maximum, counts) for each column.
  - **LumenAI Advanced** | LLM Summary: Used the LLM to generate detailed summaries for each column, describing typical values, purpose, etc.
3. **Optimal Model Comparison:**
- The optimal model was selected by comparing the basic schema and simple summaries to the computational cost of generating detailed LLM summaries. This comparison was crucial in balancing accuracy with efficiency.

User Interface Concept

The user interface for our chatbot is designed for simplicity, allowing users to upload CSV files and interact through a chat interface to generate summaries and insights from their data. The UI automatically builds the database schema via a quick questionnaire, generates various types of summaries, and integrates visualizations for enhanced reporting.


LumenAI
Upload Your Data to Start
You can upload multiple files in csv or excel format



LumenAI

Show me sales trends for Dubai in the last quarter.

Sales Trends for Dubai (Last Quarter: Nov 2023 - Jan 2024):
- November: $145,000
- December: $180,000 (highest)
- January: $160,000

Analysis:
Dubai saw a steady increase in sales in December, driven by seasonal promotions. Sales slightly decreased in January but remained strong compared to historical averages. Product A and Product C were top performers during this quarter.

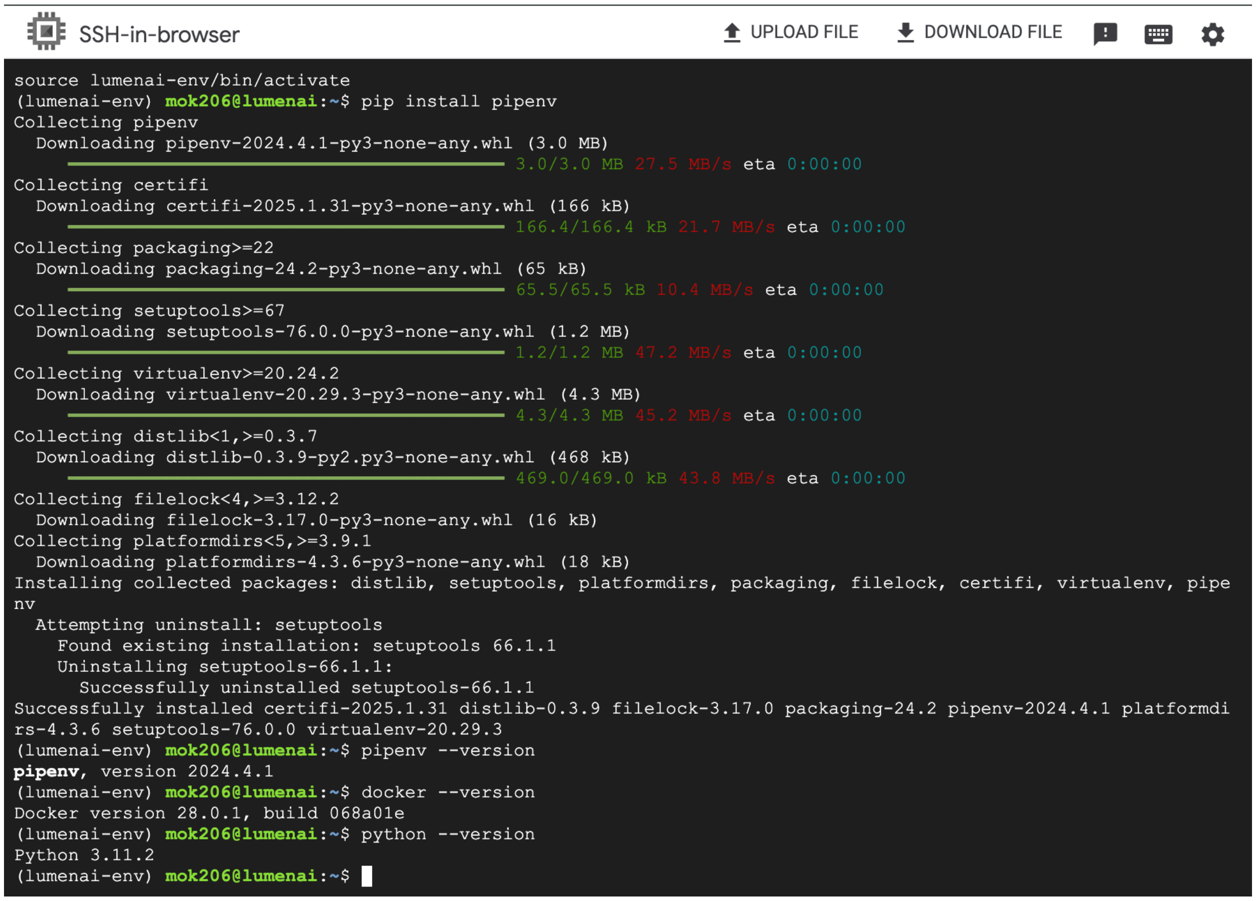
What products had the highest growth in Riyadh?

Highest Growth Products in Riyadh (YoY):
1. Product D: +35% growth
2. Product B: +22% growth
3. Product F: +18% growth

Analysis:
Product D showed significant growth due to a targeted marketing campaign in Riyadh. Product B's growth was consistent throughout the year, driven by customer loyalty programs.

Virtual Machine on GCP

Setting up a Virtual Machine on GCP was essential for our infrastructure. By utilizing Google Cloud Platform, we ensured scalable and secure deployment. The process involves creating an instance, choosing the appropriate machine type, configuring network settings, and deploying the necessary software such as Docker and Python to facilitate seamless operations.



```
source lumenai-env/bin/activate
(lumenai-env) mok206@lumenai:~$ pip install pipenv
Collecting pipenv
  Downloading pipenv-2024.4.1-py3-none-any.whl (3.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 27.5 MB/s eta 0:00:00
Collecting certifi
  Downloading certifi-2025.1.31-py3-none-any.whl (166 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 166.4/166.4 kB 21.7 MB/s eta 0:00:00
Collecting packaging>=22
  Downloading packaging-24.2-py3-none-any.whl (65 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.5/65.5 kB 10.4 MB/s eta 0:00:00
Collecting setuptools>=67
  Downloading setuptools-76.0.0-py3-none-any.whl (1.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 47.2 MB/s eta 0:00:00
Collecting virtualenv>=20.24.2
  Downloading virtualenv-20.29.3-py3-none-any.whl (4.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.3/4.3 MB 45.2 MB/s eta 0:00:00
Collecting distlib<1,>=0.3.7
  Downloading distlib-0.3.9-py2.py3-none-any.whl (468 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 469.0/469.0 kB 43.8 MB/s eta 0:00:00
Collecting filelock<4,>=3.12.2
  Downloading filelock-3.17.0-py3-none-any.whl (16 kB)
Collecting platformdirs<5,>=3.9.1
  Downloading platformdirs-4.3.6-py3-none-any.whl (18 kB)
Installing collected packages: distlib, setuptools, platformdirs, packaging, filelock, certifi, virtualenv, pipe
nv
  Attempting uninstall: setuptools
    Found existing installation: setuptools 66.1.1
    Uninstalling setuptools-66.1.1:
      Successfully uninstalled setuptools-66.1.1
Successfully installed certifi-2025.1.31 distlib-0.3.9 filelock-3.17.0 packaging-24.2 pipenv-2024.4.1 platformdi
rs-4.3.6 setuptools-76.0.0 virtualenv-20.29.3
(lumenai-env) mok206@lumenai:~$ pipenv --version
pipenv, version 2024.4.1
(lumenai-env) mok206@lumenai:~$ docker --version
Docker version 28.0.1, build 068a01e
(lumenai-env) mok206@lumenai:~$ python --version
Python 3.11.2
(lumenai-env) mok206@lumenai:~$ 
```

Next Steps

- Model Enhancements: Expand on the capabilities of the “o3-mini” model to further improve query generation and output analysis. Experiment with additional prompt fine-tuning and potentially integrate feedback loops for self-correction.
- User Data Input: Modify the application to allow users to upload their own CSV files. Develop a quick questionnaire to capture context and automatically build the database and its schema.
- Enhanced Reporting: Improve the analysis reporting, possibly by integrating visualizations and more detailed KPI suggestions, to provide richer insights from the data.
- Scalability and Integration: Continue to refine the containerized components and ensure they can scale. Explore integration with versioned data pipelines if future requirements demand it.

Conclusion

This milestone successfully established the core infrastructure for our text-to-SQL application using LLMs and a multi-level RAG framework. We developed an MVP that:

- Loads CSV data into a SQLite database.
- Extracts detailed schema information and column summaries.
- Uses an LLM to generate and execute SQL queries.
- Provides an interactive chatbot-like loop for further user inquiries.

This foundation sets the stage for further enhancements, including better model performance, improved user data ingestion, and richer data analysis capabilities. The next steps will focus on refining the LLM integration, enhancing the reporting, and expanding the system to handle user-uploaded data.