Capstone Project: Structured Data RAG and NLI for Insight Generation

---

1. Project Overview

This capstone project focuses on developing a **Retrieval-Augmented Generation (RAG)** system integrated with a **Natural Language Interface (NLI)** to extract insights from structured datasets such as CSV, SQL tables, or Excel files. The goal is to enable conversational analytics — allowing users to query structured enterprise data using natural language and receive contextually rich responses or summaries.

2. Objective

Develop an AI-driven system that combines **retrieval** and **generation** techniques to produce data-backed, human-readable insights from structured data sources.

3. Scenario-Based Problem Statement

A global retail company stores its sales, operations, and finance data across multiple structured data systems — SQL databases, CSV exports, and Excel files. Analysts often spend hours running manual SQL queries or writing Python scripts to derive insights such as sales growth, top-performing regions, or cost efficiency.

To streamline decision-making, the company wants to create an intelligent assistant that: - Connects to structured datasets (SQL, CSV, Excel). - Allows users to query using natural language. - Retrieves relevant data records and generates clear summaries or visual explanations. - Supports both fact-based lookups and generative insights (e.g., trend explanations, comparative analysis).

Example queries: - “Show me the top 5 performing regions by sales in Q3.” - “Explain the year-over-year growth trend for the finance domain.” - “Compare customer churn between 2022 and 2023 by region.”

4. Expected Outcomes

- **Data Ingestion and Indexing Module** for structured data.
- **RAG Pipeline** integrating retrieval from data sources and generation from LLMs.
- **Natural Language Interface (NLI)** for conversational querying.
- **Insight Summarization Layer** to produce business-friendly explanations.
- **Visualization Support** (charts, graphs, trend summaries).

5. System Architecture

Key Components: 1. **Data Ingestion Agent** – Connects to CSV, Excel, or SQL databases and extracts schema and data. 2. **Vectorization Layer** – Converts tabular and metadata context into vector embeddings using an embedding model. 3. **Retriever Module** – Fetches relevant data rows or records based on user query semantics. 4. **Generator Module (LLM)** – Synthesizes retrieved data into human-readable responses or explanations. 5. **Visualization Engine** – Generates supporting visuals using Matplotlib or Plotly. 6. **NLI Layer** – Provides a conversational query interface via chatbot or REST API.

6. Flow of Operations

1. User inputs a natural language query.
2. Query Understanding module classifies query intent (e.g., lookup, summary, comparison).
3. Retriever fetches structured data relevant to query.
4. Generator synthesizes retrieved content into a readable insight.
5. Output is displayed as text, table, or visual chart.

7. Tools and Frameworks

- **Libraries:** Pandas, SQLAlchemy, OpenPyXL, LangChain, Plotly, Matplotlib.
- **LLM APIs:** Gemini, GPT, Claude.
- **Vector Databases:** Pinecone, Chroma, FAISS.
- **Visualization/UI:** Streamlit, Dash, or Gradio.
- **Data Storage:** MySQL, PostgreSQL, or Google BigQuery.

8. Evaluation Metrics

- **Accuracy** of data retrieval.
- **Interpretability** of generated explanations.
- **Response Time** for queries.
- **User Satisfaction** in NLI usability.
- **Coverage** of data insights across datasets.

9. Supportive Implementation Guide (with Hints)

Hint 1: Structured Data Preprocessing
- Normalize all incoming structured data into consistent formats (CSV/SQL/Excel).
- Extract metadata (table names, column headers) for schema-level indexing.

Hint 2: Embedding Generation and Retrieval
- Represent column headers and key-value pairs as text strings for embedding.
- Use `text-embedding-3-small` or Gemini embeddings for vector representation.
- Store vectors in FAISS or Pinecone for fast semantic search.

Hint 3: Query Understanding
- Use prompt templates to classify query type — e.g., *aggregation*, *trend analysis*, *comparison*, or *summary*.
- For numeric or analytical queries, dynamically generate SQL queries or Pandas operations.

Hint 4: RAG Integration
- Combine structured data retrieval with generative summarization using LangChain's RAG chain or a custom retrieval pipeline.
- Ensure retrieved data is validated before feeding to the LLM.

Hint 5: Response Generation
- The generator should combine tabular facts with narrative summaries.
- Use prompts that explicitly instruct the LLM to include reasoning and context (e.g., "Based on the retrieved data, summarize the growth pattern.").

Hint 6: Visualization Layer
- Generate visual summaries using Plotly or Matplotlib.
- Embed charts directly into the NLI interface for enhanced insight comprehension.

Hint 7: User Interface and API
- Develop a conversational UI using Streamlit or Gradio for real-time query input.
- Provide API endpoints to allow other applications to use the NLI.

Hint 8: Evaluation and Tuning
- Evaluate retrieval precision and generation accuracy separately.
- Fine-tune prompts and retrieval configurations for domain-specific queries.

---

10. Deliverables

- Functional RAG-based structured data insight generator.
- Documentation on architecture and workflow.
- Demo interface showcasing conversational querying.
- Evaluation report with performance metrics and accuracy benchmarks.