Project Documentation: Mistral API Integration for Medical Records

# Introduction

This project integrates the **Mistral API** through **Groq API** to build a system that efficiently processes medical records stored in **Snowflake** and generates insights via a chatbot and a dashboard. The project enhances data retrieval and presentation for hospital patient data, enabling users to interact with large datasets and gain meaningful insights.

# Objectives

The main objectives of this project are:

1. **Chatbot**: Build a chatbot capable of answering questions related to the patient records stored in the **FLATTENED\_MEDICAL\_RECORDS** table.
2. **Dashboard Generator**: Develop an automated dashboard generator that can create dashboards based on the data in the **FLATTENED\_MEDICAL\_RECORDS** table by submitting a user prompt.

The **FLATTENED\_MEDICAL\_RECORDS** table is located within the **MISTRALHEALTHDB.MEDICALRECORDDATAMART** schema in **Snowflake** and contains detailed patient data, illnesses, and associated metadata.

# Project Architecture

The project is broken down into two major sections, Medical data is entered into the Snowflake database via a dedicated application. This system allows medical personnel to perform CRUD operations (Create, Read, Update, and Delete) by efficiently receiving and managing patient information.

The Mistral AI model is integrated with the Snowflake database to provide actionable insights from the data. This integration supports informed decision-making within the organization. It provides medical professionals with the ability to interact with the database and get needed information.

A screenshot of a computer

Description automatically generated

# Producer Script

The producer sends data to Snowflake Data Warehouse where the Mistral Model will build a RAG model upon.

**👉🏾** [**Producers**](https://github.com/kiddojazz/Snowflake-Mistral-Project-RAG/blob/main/RaG_N_ROLL/producer.py)

## Mistral API Integration

The heart of this project is powered by the **Mistral API** via **Groq API**. The **Groq API version** of Mistral was selected due to its superior execution speed, which is essential for processing large datasets like the medical records efficiently.

### Snowflake Data Storage

Data for this project is stored in **Snowflake** under the **FLATTENED\_MEDICAL\_RECORDS** table. This table includes:

* Patient Demographic Information
* Patient Body Stats Information
* Illness Details
* Physician Information
* Insurance Details

# Deliverables

To make the documentation more digestible, the project has been broken down into three main categories based on the deliverables:

* **Chatbot Modules**: Modules used to build the chatbot functionality.
* **Dashboard Modules**: Modules used to build the automatic dashboard generation.
* **Generic Modules**: Common modules used by both the chatbot and the dashboard functionality.

Each section below outlines the modules and their roles in the project.

# Modules Overview

## Generic Modules:

**Pages\_markdowns.py:**

This module defines the **markdown pages** used for each page in the **Streamlit** application. It provides a structure for presenting content in markdown format within the web interface.

## Chatbot Modules:

**Mixtral\_chat.py:**

This module defines the core chatbot functionality through the **MixtralAgents** class, which houses multiple agents that work together using the **Mistral-Groq API**. Each agent serves a specific purpose for processing user prompts and returning the appropriate responses:

* **Sql\_query\_agent**: Generates SQL queries corresponding to user prompts. This agent helps retrieve data from the **Snowflake** database based on user questions.
* **Prompt\_agent**: Optimizes user prompts by generating more refined SQL queries. This agent enhances the quality of the user's input to get more accurate results.
* **Streamlit\_agent**: This agent is responsible for answering user queries related to a dataset displayed on the **Streamlit** interface. It enables interaction with data in a visualized format.
* **Get\_mixtral\_response**: A helper agent used by other modules to interact with the **Mistral-Groq API** and retrieve responses from the LLM.

Additionally, the **Mixtral\_chat.py** module includes the **ResponseCache** class, which tracks and stores the most recent conversation history to maintain context during the interaction.

**Mixtral\_utilities.mixtral\_context.py**

This module stores all the **contexts** and **prompts** used to guide the **Mistral LLM** in generating answers to user queries related to the **Snowflake** table. These contexts define how the LLM should interpret and respond to different prompts about the dataset.

**Mixtral\_utilities.mixtral\_tools.py**

Currently, this module contains a function for querying the **Snowflake DB**. It serves as a utility to interact with the data storage, enabling other components to fetch relevant data based on the user's requests.

## Chat Conversation Module

**Chat\_conversation.py:**

This module orchestrates the **Streamlit** chat interface, which is the core interaction point for answering questions about the patient data in the **FLATTENED\_MEDICAL\_RECORDS** table. It provides two classes for different modes of operation within the chat interface:

* **ChatInterface**: This class represents the main chat interface, where users send queries or prompts. It returns a **DataFrame** based on the prompt, providing the relevant data extracted from the **Snowflake** table.
* **SideChatInterface**: This class manages the **side chat functionality** within the same page. After a **DataFrame** is generated from the **ChatInterface**, users can interact further with the chatbot through the side chat. This mode allows for follow-up conversations or deeper exploration of the data that was previously retrieved.

A screenshot of a computer

Description automatically generated

Figure : Full View

A screenshot of a chat

Description automatically generated

Figure : Main Chat

A screenshot of a phone

Description automatically generated

Figure : Side Chat

## Dashboard Module

A screenshot of a computer

Description automatically generated

**Pages\_utilities.streamlit\_plots.py**

This module contains all the possible **plotting functions** that could be implemented on the **Streamlit Dashboard** page. It provides various charting utilities, each designed to work with different data types and visualization needs. These functions can generate a wide range of visualizations, including but not limited to **bar charts**, **line charts**, **pie charts**, and **histograms**.

**Pages\_create\_streamlit\_chart.py:**

This module defines functions that utilize the **streamlit\_plots** module to generate the charts required for the dashboard. It takes the information about the charts (e.g., chart type, data, title) and uses the plot utilities to display them on the **Streamlit** page.

**Pages/dashboard\_generation.py**:

This module is the central component for generating the **Streamlit Dashboard**. It follows a series of steps to process the user's request and create dynamic, data-driven reports:

1. **Accept User Prompt**: The module first accepts a user’s prompt and determines the **number of reports** that can be generated based on the prompt. Each report will have its own **description** and scope, helping define the type of visualizations and data needed.
2. **Generate SQL Queries**: For each report description, **SQL queries** are generated using the **Mistral API**. These queries are tailored to pull the necessary data from the **FLATTENED\_MEDICAL\_RECORDS** table in **Snowflake**, based on the user's requested report topics.
3. **Query Snowflake**: Using the generated SQL queries, the **Snowflake** table is queried to retrieve the relevant data. The queries return **DataFrames** containing the necessary information to populate the dashboard.
4. **Generate Dashboard Charts**: The data from the **DataFrames** is used to generate detailed information about the charts. This chart information is then fed into the **create\_streamlit\_chart.py** module, which creates the visualizations on the Streamlit dashboard. The charts are generated dynamically, reflecting the data queried from the Snowflake database.

**LLM\_messaging.context\_functions.py**

This module contains functions that are used to **perform each step** described in the operation of **dashboard\_generation.py**. The functions in this module act as the intermediary between the **dashboard generation process** and the **Mistral API**, ensuring smooth execution of the operations required to generate reports and dashboards. These functions handle tasks such as **query formation**, **response handling**, and **data extraction**, all critical to the flow of generating dynamic content based on user prompts.

**LLM\_messaging.llm\_context\_and\_format.py**

Similar to **mixtral\_context.py**, this module defines **contexts** and **prompts** that are used to instruct the **Mistral LLM** for generating the **Streamlit charts**. It contains the logic that feeds structured inputs to the Mistral API, guiding it to provide responses that match the desired output format for visualizing the data in **Streamlit**. This module ensures that all the chart-related contexts are appropriately formatted and that the LLM responds with the correct metadata, chart details, and visual components.

# Conclusion

This project integrates various modules and technologies to create an intelligent, interactive system that leverages the **Mistral API** for dynamic question-answering and dashboard generation. By combining **Snowflake** as the data source, **Streamlit** for visualization, and **Mistral-Groq API** for natural language processing, we’ve developed a solution that allows users to explore and interact with hospital patient data through both **chatbot** and **dashboard** interfaces.

The **Chatbot Modules** enable users to query the medical records seamlessly, while the **Dashboard Modules** automatically generate insightful reports and charts based on user prompts. The **Generic Modules** provide foundational utilities that support both deliverables.

With this architecture, the system provides an intuitive, flexible approach to medical data exploration, empowering users to gain insights without needing advanced technical knowledge. Moving forward, the framework can be extended to other types of datasets and visualizations, offering a scalable solution for various business and research applications.