This project is a **conversational chatbot** that enables users to generate data insights in the form of **charts and graphs**, mimicking a dashboard experience. The bot supports natural language input and leverages:

* **Llama models** (via Groq)
* **PandasAI** for data-aware chart generation
* **Streamlit (or Flask)** as the interface (TBD after full file inspection)

**🔹 Key Features**

* Accepts natural language queries
* Converts queries into data visualizations
* Offers multiple chart suggestions
* Supports different Excel schemas
* Fast response generation via **Groq-hosted LLMs**

**🔹 Tech Stack**

| **Component** | **Description** |
| --- | --- |
| LLM | LLaMA models via Groq |
| Charts | PandasAI, Matplotlib, Seaborn |
| Data Input | CSV/Excel-based |
| Backend | Python (Streamlit) |
| NLP Layer | Custom nlp\_response.py + LLM |

**🔹 Folder Structure**

app.py # App entry point

config.py # API keys and config

db\_utils.py # Utility to read/store data

excel2.py # Excel processing logic

excelll.py # Excel interface variant

excel\_3.py # Another Excel logic variant

nlp\_response.py # Handles user queries and chart types

new\_schema\_2.csv # Sample input data

llm\_groq # Groq integration and prompt logic

**🔹 Setup Instructions**

1. **Clone the Repository / Extract the ZIP**

unzip new\_prototype.zip

cd new\_prototype

1. **Install dependencies**

pip install -r requirements.txt

1. **Use config.py for API keys**

GROQ\_API\_KEY=your\_key

PANDASAI\_API\_KEY=your\_key

1. **Run the App**

Streamlit run app.py

**🔹 File Responsibilities**

| **File** | **Description** |
| --- | --- |
| app.py | Central app logic, routes, execution of chatbot response |
| config.py | Stores keys and configuration values |
| db\_utils.py | Reads and formats Excel/CSV input |
| nlp\_response.py | Contains logic to match prompt to graph/chart type |
| excel2.py, excel\_3.py, excelll.py | Variants for parsing and formatting Excel datasets |

**🔹 How it Works**

1. **User Input** → User types a query like *“Show sales by category over time.”*
2. **LLM Prompting** → The prompt is passed to LLaMA via Groq.
3. **NLP Logic** → nlp\_response.py identifies keywords like "bar", "stacked", "line".
4. **Excel Processing** → One of the excelX.py files loads and parses the dataset.
5. **Graph Generation** → PandasAI creates a visual representation based on LLM + data context.

**🔹 Example Usage**

**User Prompt:**

"Give me a 100% stacked bar chart of transactions over the month."

**Generated Response:**

Chart rendered using PandasAI showing % of 'Transaction Success', 'Timeout', etc., grouped by day.

**File-by-File Explanation**

**1. app.py**

* **Role:** Orchestrator of the application.
* **What it does:** Acts as the main entry point. It reads user queries, triggers appropriate logic (data loading, NLP, LLM call), and returns a chart.
* **Flow:**
  + User query → NLP → Data Load → Chart Generator → Return plot

**2. config.py**

* **Role:** Stores all configuration variables such as:
  + API keys (e.g., Groq)
  + File paths
  + Environment variables

Keeping this file separate ensures better security and maintainability.

**3. nlp\_response.py**

* **Role:** Analyzes the user's query to determine the chart type or logic needed.
* **How:** It searches for keywords (like “bar chart”, “line graph”, “stacked”) to decide the direction of processing.
* **Output:** A clear instruction about the desired visualization.

**4. db\_utils.py**

* **Role:** Responsible for loading data from the source files (CSV, Excel).
* **Functions:**
  + Reads files
  + Converts them into structured Pandas DataFrames
  + Makes data ready for chart generation

**5. excel2.py, excel\_3.py, excelll.py**

* **Role:** Handle preprocessing of Excel files with slightly different structures.
* **Functionality:**
  + Standardize column names
  + Handle date/time formatting issues
  + Remove inconsistencies or junk values
* **Why multiple files?** Each corresponds to a different dataset format/schema used in the application.

**6. query\_generator.py**

* **Role:** Handles interaction with the **Groq LLaMA API**.
* **What it does:**
  + Sends the user's question as a prompt
  + Receives back code or chart logic
  + This result is sent to PandasAI for execution

**7. schema\_utils.py**

* **Role:** Helps in validating or loading structured CSV schema.
* **Functionality:** Ensures that the CSV being processed fits the required column names and format.

**8. visualization.py**

* **Role:** Contains the actual plotting logic.
* **Key Functions:**
  + plot\_100\_percent\_stacked\_bar\_chart(data): Generates a 100% stacked bar chart, breaking down each category by its percentage on a given date
  + generate\_dashboard(...): Automatically chooses chart types and creates plots using the PandasAI engine

This is the file where visualizations are assembled before being displayed to the user.

**9. new\_schema\_2.csv**

* **Role:** A sample dataset used to test the application.
* **Structure:** Contains typical business KPIs or transactional data like status, dates, and categories.