Chat2Chart

By
Anirudh
Abhijeet

Idea

- **Goal**
  - Build an interface where users can interact with their data using natural language processing.
- **Inspiration**:
  - The concept is similar to Jupyter Notebook but with a focus on natural language instead of manually loading data into Pandas for analysis.
- **Core Features**:
  - Users can perform data analysis by conversing in natural language, eliminating the need for coding skills.
  - AI agent interprets user queries, breaks them down into simple steps, and executes the necessary Python code for data analysis.
- **Visualization Options**:
  - Users can request specific chart types (line, bar, scatter, etc.) to visualize their data.
- **Additional Functionality**:
  - Users can ask for explanations about their data, and the AI assistant will provide clear insights.Planned feature: The ability to publish dashboards directly from the platform.

Project Approach & Technologies Used

- **Project Approach**:
  - Started from scratch, avoiding LangChain and Llama-Index.
  - Goal: Build a custom pandas DataFrame agent for interacting with tabular data.
- **Technologies Used**:
  - **Streamlit**: For building the user interface and interacting with the model.
  - **OpenAI (for LLM)**: To process natural language queries and generate insights.
  - **Pandas**: For managing and analyzing tabular data within the project.
  - **Cursor & Claude Sonnet**: Used for assistance in implementing the DataFrame agent and refining the process.

Implementation

- **Creating the Pandas DataFrame Agent**:
  - Used **Claude Sonnet** in **Cursor** to build the DataFrame agent from scratch.
  - After multiple iterations, successfully created a working agent tailored for our use case.
- **Connecting to the LLM**:
  - Initially wrote rough code with the **autocompletion feature** for prompts.
  - Connected the agent to the **OpenAI LLM**, enabling natural language queries on data.
- **Handling Image Output**:
  - Encountered difficulties in generating visual outputs after the agent execution.
  - Use chat feature in **Cursor**, which helped finalize the image output implementation.
- **Outcome**:
  - The implementation works, though still not perfect.
  - **Cursor** and **Claude Sonnet** played crucial roles in speeding up the development process and guiding through challenges.

Current Challenges

- **Conversation Memory**:
  - Difficulty in remembering previous conversations or passing context to the LLM during session.
- **Complex Queries**:
  - Challenges with handling multi-step queries that require generating multiple lines of Python code.
- **Expanding Features**:
  - Developing the ability to publish a dashboard directly from the application.
  - Attempting to integrate **Streamlit bar charts** and other visual features into the interface.

Next Steps

- **Transitioning from Streamlit to Next.js**:
    - Moving from a **Streamlit demo** to a more scalable, production-ready platform using **Next.js**.
    - Aiming for better performance, UI/UX, and flexibility with a Next.js-based architecture.
- **Future Improvements**:
    - Optimizing the LLM’s handling of complex, multi-line code queries.
    - Enhancing the user interface with more robust data visualization tools.
    - Integrating dashboard publishing and other advanced features into the Next.js application.
- **Expand support** for more data file formats (e.g., JSON, XML, SQL databases).