Technical Documentation for Streamlit Energy Dashboard Application

Overview

This Streamlit application serves as an energy management dashboard tailored for two types of users: consumers and prosumers. It provides functionalities for visualizing energy consumption/production data, forecasting future consumption/production using the Prophet model, and managing Energy Resource Communities (ERCs).

To Run this Script

Step 1: Run the command streamlit run f.py

Step 2: Press Enter Press Enter to execute the command. Streamlit will launch a web server and open a web browser to display your app.

Step 3: View your app Open a web browser and navigate to http://localhost:8501 (or the port number you specified) to view your Streamlit app.

User Authentication

Login Functionality

- **Username and Password:** The application starts with a login screen where users must enter their credentials. The default credentials are:
  - Username: `manager`
  - Password: `password`
- **User Type Selection:** Users can select their type as either 'consumer' or 'prosumer'.
- **Session Management:** Upon successful login, the session state is updated to reflect that the user is logged in and their selected user type. If the credentials are incorrect, an error message is displayed.

Sample Data Generation

Data Generation

- **Purpose:** To generate sample energy consumption/production data for demonstration purposes.
- **Data Structure:** The data consists of DateTime, Consumption (kWh), Cost ($), and CO2 Emissions (kg) columns. Each day has 24 random entries representing hourly data.
- **Sorting:** The generated data is sorted by DateTime to ensure chronological order.

Dashboard for Consumers and Prosumers

Shared Features

Both dashboards share some common features and functionalities:

- **Date Range Filter:** Users can filter data based on a selected date range. If the start date is after the end date, an error message is shown.
- **Visualizations:** The application provides various types of visualizations:
  - **Line Chart:** Shows hourly energy consumption/production.
  - **Bar Chart:** Displays hourly cost/revenue.
  - **Scatter Plot:** Illustrates hourly CO2 emissions/savings.
- **Future Prediction:** Uses the Prophet model to forecast energy consumption/production for the next 7 days.

Consumer Dashboard

- **Title:** "Consumer Dashboard"
- **Data Display:** Shows filtered data in a tabular format.
- **Specific Visualizations:**
  - **Hourly Energy Consumption:** Line chart of consumption over time.
  - **Hourly Energy Cost:** Bar chart of cost over time.
  - **Hourly CO2 Emissions:** Scatter plot of CO2 emissions over time.
- **Interactive Heatmap:** Visualizes consumption data against cost using a heatmap.

Prosumer Dashboard

- **Title:** "Prosumer Dashboard"
- **Data Display:** Shows filtered data in a tabular format.
- **Specific Visualizations:**
  - **Hourly Energy Production:** Line chart of production over time.
  - **Hourly Revenue:** Bar chart of revenue over time.
  - **Hourly CO2 Savings:** Scatter plot of CO2 savings over time.
- **Interactive Histogram:** Shows the distribution of energy production.

ERC Management

ERC Management Functionality

- **Sample ERC Data:** Initially, two sample ERCs are created.
- **Display Existing ERCs:** Lists all existing ERCs along with their names, users, and types.
- **Create New ERC:** Allows users to create a new ERC by providing a name and selecting the type (consumer or prosumer).
- **Add User to ERC:** Users can add a new user to an existing ERC.
- **Change User Type:** Users can change the type of an existing user in an ERC.

Sidebar Chatbot

Chatbot Functionality

- **Initialization:** The sidebar contains a chat interface for users to interact with a chatbot.
- **Message History:** Displays previous chat messages.
- **User Input:** Users can type and send messages through a form.
- **Hardcoded Responses:** The bot can provide hardcoded responses to specific queries.

- **OpenAI API Integration:** For other queries, the application uses the OpenAI API to generate responses.

Utility Functions

Customization of Plots

- **Function:** `customize_plot(fig)`
- **Purpose:** Customizes the appearance of Plotly figures to have a dark background and light gridlines for better visual contrast.

Session State Management

Key Session Variables

- **`logged_in`:** Boolean indicating if the user is logged in.
- **`user_type`:** The type of user (consumer or prosumer).
- **`messages`:** List of chat messages between the user and the chatbot.
- **`ercs`:** List of existing ERCs and their details.

Dependencies

Libraries Used

- **Streamlit:** For the web application framework.
- **Pandas:** For data manipulation and analysis.
- **Numpy:** For numerical operations.
- **Plotly:** For creating interactive visualizations.
- **Prophet:** For time series forecasting.
- **OpenAI:** For chatbot functionality.

Environment Variables

- **OpenAI API Key:** Required for integrating the OpenAI chatbot functionality.

Conclusion

This documentation covers the key functionalities and components of the Streamlit Energy Dashboard application, detailing the purpose and usage of each feature. The application is designed to provide an intuitive and interactive experience for managing and visualizing energy data for both consumers and prosumers.

Additional information which can be helpful for the Bot understanding.

To implement Retrieval-Augmented Generation (RAG):

1. **Data Ingestion**: Collect and organize our documents or relevant data sources that will be used for retrieval. This involves preparing a document corpus that can be processed by the system.
2. **Embedding Creation**: Tokenize the text from your documents and convert these tokens into vector representations (embeddings). This step is crucial as it allows the system to understand and retrieve relevant information based on user queries.
3. **Indexing**: Store the created embeddings in a vector database. This enables efficient similarity searching, allowing the system to quickly find relevant document chunks when a query is made.
4. **Query Processing**: When a user submits a query, convert it into an embedding using the same model used for the document embeddings. This ensures consistency and allows for accurate similarity comparisons.
5. **Response Generation**: Retrieve the most relevant document chunks based on the query embedding and feed them, along with the original user query, into a language model (LLM). The LLM will generate a coherent and contextually relevant response based on the retrieved information