Flask API Documentation

Overview

This Flask application provides an API that allows users to interact with a pre-loaded dataset using OpenAI's GPT-4 model. The server loads data from a file (PDF, DOCX, CSV, or Excel) into memory when it starts. Users can then send questions to the API, and the GPT-4 model will analyze the pre-loaded data to provide answers.

Key Features

- Load and process data from various file formats (PDF, DOCX, CSV, Excel).
- Trim data to fit within the GPT-4 token limit if necessary.
- Respond to user questions based on the pre-loaded data.
- Return JSON responses indicating success or failure.

Code Explanation

1. Dependencies

The following libraries are used in the application:

- `Flask`: A micro web framework for Python.
- `openai`: OpenAI's Python client library to interact with GPT-4.
- `pandas`: A data manipulation library for CSV/Excel files.
- `docx`: A library to extract text from DOCX files.
- `pdfplumber`: A library to extract text from PDF files.
- `dotenv`: A library to load environment variables from a `.env` file.

2. Environment Configuration

The OpenAI API key is stored in an environment variable. The `.env` file should contain the following:

```
plaintext
Copy code
OPENAI_API_KEY=your_openai_api_key_here
```

3. Data Extraction Functions

- `extract_text_from_pdf(pdf_path)`: Extracts text from a PDF file.
- `extract_text_from_docx(docx_path)`: Extracts text from a DOCX file.
- `extract_data_from_csv_excel(file_path)`: Reads data from a CSV or Excel file, handling different encodings.
- `estimate_token_count(text)`: Estimates the number of tokens in the text.
- `trim_text_to_token_limit(text, max_tokens=9000)`: Trims text to a maximum of 10,000 tokens.

4. Data Loading

When the server starts, the data from the specified file path is loaded and processed. If the data exceeds the 10,000-token limit, it is trimmed to fit.

5. API Endpoint

- **`/ask` (POST)**: This endpoint accepts a JSON payload with a `question` field. The user's question is appended to the conversation history, and GPT-4 processes the data to generate a response.

Example request payload:

```
json
Copy code
{
  "question": "What is the average value of column X?"
}
```

Example response:

```
json
Copy code
{
  "status": "success",
  "answer": "The average value of column X is 42."
}
```

If the `question` field is missing or an error occurs, the API will return a failure response.

How to Run the Application

1. **Clone the repository** or place the code in a directory.
2. **Create a virtual environment** and install dependencies:

   ```
   bash
   Copy code
   python -m venv myenv
   source myenv/bin/activate  # On Windows use `myenv\Scripts\activate`
   pip install -r requirements.txt
   ```

3. **Set up the .env file**: Create a `.env` file in the same directory as your Flask application and add your OpenAI API key:

   ```
   plaintext
   Copy code
   OPENAI_API_KEY=your_openai_api_key_here
   ```

4. **Run the Flask server**:

   ```
   bash
   Copy code
   python your_flask_app.py
   ```

The server will start at `http://127.0.0.1:5000/`.

How to Test the API

You can test the API using tools like **Postman**, **cURL**, or even a simple Python script.

Using cURL

```
bash
Copy code
curl -X POST http://127.0.0.1:5000/ask -H "Content-Type: application/json" -d
'{"question": "What is the data about?"}'
```

Using Postman

1. Open Postman and create a new POST request.
2. Set the URL to `http://127.0.0.1:5000/ask`.
3. In the **Body** tab, select **raw** and set the content type to **JSON**.
4. Enter the following JSON:

   ```
   json
   Copy code
   {
     "question": "What is the data about?"
   }
   ```

5. Click **Send** to see the response.

Using Python

```
python
Copy code
import requests

url = "http://127.0.0.1:5000/ask"
payload = {"question": "What is the data about?"}
headers = {"Content-Type": "application/json"}

response = requests.post(url, json=payload, headers=headers)
print(response.json())
```

Expected Response

- **Success**:

  ```
  json
  Copy code
  {
    "status": "success",
    "answer": "The data contains information about..."
  }
  ```

- **Failure**:

  ```
  json
  ```

```
Copy code
{
  "status": "fail",
  "message": "Error message here"
}
```

Conclusion

This Flask API allows users to query pre-loaded data using GPT-4. The data is loaded and processed once when the server starts, and users interact with it via POST requests. The API handles various file formats and ensures that the data does not exceed GPT-4's token limits.

Hurdles

Text-Based Analysis Limitation:

- **Issue**: GPT-4 is primarily designed for text-based analysis and generation. After feeding data to the API, it cannot perform image-related tasks like generating bar charts or other visualizations directly.
- **Impact**: This limits the model's ability to provide comprehensive analysis when visual data representation is required.

Token Limit Constraint:

- **Issue**: GPT-4 has a token limit of 10,000 tokens per minute (TPM). This limit means that large datasets need to be trimmed or split, potentially leading to incomplete data being analyzed.
- **Impact**: Important data may be excluded from analysis if the token count exceeds the limit, reducing the accuracy or comprehensiveness of the model's output.

Currently I am looking into other ways to get visual response so that customer can get insights of the data.

One way could be :

Use Case: GPT-4 generates the necessary code or instructions, and our backend (e.g., Flask) executes this code to create the visualizations. The resulting image is then returned to the user as part of the API response.

- **Example**: The user asks for a bar chart of sales data, GPT-4 generates the code, and our server uses Matplotlib to generate and return the chart image.

Another way coulde be to find some other api’s that could help us in this situation(which return a meaningful data insight as image in response to a text prompt).