is this a practical use case?

use a combination of a python app front end and open ai api to create a solution that:

1) allows for the upload of any number of market response documents (eg tenders, rfq, eoi, etc)

2) allows for the upload of an evaluation document that has specific criteria and weightings

3) performs the evaluation

4) outputs the results in a report that justifies the evaluation

Yes, this is a highly practical use case! Using a combination of a Python app front end and OpenAI’s API can effectively create a solution to evaluate market response documents against specified criteria and weightings. Here's how this use case can be structured:

**Proposed Solution Design**

**1. Python App Front End**

The Python app provides the user interface for:

* **Uploading Documents**:
  + Market response documents (e.g., tenders, RFQs, EOIs).
  + Evaluation documents with criteria and weightings.
* **Configuring the Evaluation**:
  + Allow users to verify or adjust criteria/weightings.
* **Viewing and Exporting Results**:
  + Present evaluation results in an interactive report and allow downloads.

**2. OpenAI API for Evaluation**

Leverage the OpenAI API to:

* **Analyze Market Response Documents**:
  + Summarize or extract key sections of each document.
  + Score each response against evaluation criteria.
* **Automate Justifications**:
  + Use the generative capabilities of OpenAI to create natural language justifications for each score based on extracted content.

**3. Evaluation Workflow**

1. **Upload Files**:
   * Market response documents are uploaded as .pdf, .docx, or .txt.
   * Evaluation criteria are uploaded in a structured format (e.g., JSON or Excel).
2. **Parse Documents**:
   * Use libraries like **PyPDF2** or **python-docx** to extract text from the uploaded files.
   * Store content for processing.
3. **Weighting Application**:
   * Criteria weightings from the evaluation document are applied to ensure the scoring reflects organizational priorities.
4. **Generate Scores**:
   * For each document, use OpenAI’s API to compare the content with evaluation criteria and generate scores.
   * Example prompt:
   * Evaluate the following document against this criterion: "Company experience with similar projects (Weight: 20%)". Provide a score out of 10 and justify your score:
   * Document: <Extracted Text>
5. **Generate Justifications**:
   * OpenAI’s API can create human-readable explanations for each score based on the text of the documents and evaluation criteria.
6. **Output Results**:
   * Combine the scores and justifications into a structured report.
   * Use libraries like **pandas** for data handling and **Matplotlib** or **Plotly** for visualization.

**4. Reporting**

* **Summary Report**:
  + A high-level overview of the evaluation, including total scores and rankings.
* **Detailed Report**:
  + Justifications for scores, mapped to the specific criteria and weightings.
  + Export as .pdf, .docx, or .xlsx.

**Benefits**

* **Efficiency**:
  + Automates manual evaluations, saving significant time.
* **Consistency**:
  + Ensures evaluation criteria are applied uniformly across all submissions.
* **Transparency**:
  + Provides clear, detailed justifications for each score.
* **Scalability**:
  + Handles a large number of documents and criteria without significant manual effort.

**Technical Components**

1. **Front-End Frameworks**:
   * **Flask** or **FastAPI** for the Python app interface.
   * **Streamlit** for a simple, interactive UI.
2. **File Parsing Libraries**:
   * **PyPDF2**, **pdfminer**, or **PyMuPDF** for PDFs.
   * **python-docx** for .docx files.
3. **OpenAI API**:
   * Use GPT models for evaluation and justification.
4. **Data Handling**:
   * **pandas** for processing criteria and results.
   * **NumPy** for weighting calculations.
5. **Visualization and Reporting**:
   * **Matplotlib**, **Plotly**, or **Seaborn** for score visualization.
   * **FPDF** or **python-docx** for generating reports.

**Potential Challenges**

* **Data Privacy**:
  + Sensitive information in market response documents might require compliance with data protection regulations.
  + Mitigation: Use OpenAI’s API with strict controls or host models locally.
* **Prompt Engineering**:
  + Crafting effective prompts to ensure accurate and fair evaluation.
* **Complex Criteria**:
  + Some criteria might be highly subjective, requiring careful human validation.

**Extensions**

* Add support for real-time feedback, allowing users to tweak criteria and regenerate evaluations.
* Integrate with existing procurement or document management systems.
* Use vector databases (e.g., Pinecone or FAISS) for document embeddings and faster retrieval of relevant sections.

This solution is feasible and could significantly improve the efficiency and transparency of document evaluation processes. Let me know if you’d like further technical details or help designing a prototype!

# Jan 29

## Prompt

docker run -d -p 5001:5000 --env-file .env TSSVR23/voice-app:latest

how to run a docker container so that it automatically restarts when the docker engine starts

## response

docker run -d --restart unless-stopped -p 5001:5000 --env-file .env TSSVR23/voice-app:latest

**Explanation of the --restart Options**

* **no** (default) → The container does **not** restart automatically.
* **always** → The container **always restarts** whenever Docker starts, even if manually stopped.
* **unless-stopped** → The container restarts unless it was manually stopped (recommended).
* **on-failure** → The container restarts only if it **exits with an error**.

### Establish the directory structure

1. Copy the last *chatbot* directory and paste it into the *development* directory
2. Rename it to *chatbot<next number in the sequence>*
3. Delete the *venv* directory

### Create and activate the environment

1. Open a command prompt (cmd)
2. Go to the newly created development directory
3. Create the environment with python -m venv venv
4. deactivate
5. Activate the environment with *venv\Scripts\activate*

## Establish the directory of the python interpreter

In VSCode

1. **View** -> **Command Palette**
2. *Python: Select Interpreter*
3. **Enter interpreter path**
4. **Browse your file system to find a Python interpreter**
5. Browse to: *C:\Users\andrew.dilley\development\chatbot3\venv\Scripts\python.exe*

## Install Python dependencies

pip install --no-cache-dir -r requirements.txt

## new version of pip

python.exe -m pip install --upgrade pip

## Run the application

## ModuleNotFoundError: No module named 'PyPDF2'

Requirements.txt updated to include:

PyPDF2

## Re-install Python dependencies

pip install --no-cache-dir -r requirements.txt

Downloading pypdf2-3.0.1-py3-none-any.whl (232 kB)

Installing collected packages: PyPDF2

Successfully installed PyPDF2-3.0.1

## Run the application

## ModuleNotFoundError: No module named 'pandas'

Requirements.txt updated to include:

pandas

## Re-install Python dependencies

pip install --no-cache-dir -r requirements.txt

Successfully installed numpy-2.2.2 pandas-2.2.3 python-dateutil-2.9.0.post0 pytz-2024.2 six-1.17.0 tzdata-2025.1

## Run the application

## Running on http://127.0.0.1:5000

Change the port to avoid any clashes

Change to app.py

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, port=5002)

## CSS not working

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

#### Stage All Changes:

git add .

### *check the status of the local repository*

git status

### *Committing Changes*

git commit -m "Redaction of PII and hiding copy button"

### *Push Changes to Remote:*

git push origin master

# Jan 30