Wine Business Chatbot Documentation

Overall Approach

The goal was to develop a chatbot that can answer user queries based on a predefined corpus and sample Q&A data for a wine business. The approach involved several key steps:

1. **Text Extraction from PDF**: Using PyMuPDF, the text content was extracted from a provided PDF file containing the corpus.
2. **Text Preprocessing**: The extracted text was split into manageable chunks for easier embedding and retrieval.
3. **Text Embedding**: SentenceTransformers was used to create embeddings of the text chunks, which allows for efficient similarity searches.
4. **Q&A Matching**: A predefined set of sample questions and answers (SAQ) was loaded from a JSON file to provide direct matches for common queries.
5. **Context Management**: Streamlit's session state was utilized to maintain conversation history and context, ensuring that follow-up questions were handled appropriately.
6. **Response Generation**: OpenAI's GPT-4 model was used to generate responses for queries, especially when a direct answer from the corpus or SAQ was not available.

Frameworks/Libraries/Tools Used

1. **Streamlit**:
    - Used to build the web interface for user interaction.
    - Components: st.text_input, st.write, st.title.
2. **OpenAI**:
    - Used for generating conversational responses using the GPT-4 model.
    - Key Function: openai.ChatCompletion.create.
3. **PyMuPDF**:
    - Used for extracting text from the provided PDF corpus.

- Key Function: fitz.open.

4. SentenceTransformers:
    - Used for creating embeddings of the text chunks.
    - Model: paraphrase-MiniLM-L6-v2.
5. Torch:
    - Used for tensor operations and calculating cosine similarity scores.
    - Functions: torch.topk, util.pytorch_cos_sim.

Problems Faced and Solutions

1. Context Management:
    - Problem: The chatbot initially failed to maintain the context of the conversation.
    - Solution: Implemented the use of Streamlit's session state to track and maintain conversation history, enabling context-aware responses.
2. Redundant Responses:
    - Problem: The chatbot sometimes provided the same response repeatedly.
    - Solution: Implemented a caching mechanism to store and reuse responses for repeated queries.
3. Answer Retrieval Accuracy:
    - Problem: The retrieval mechanism sometimes failed to find relevant answers in the corpus.
    - Solution: Fine-tuned the threshold for cosine similarity and improved text preprocessing to ensure better matching.
4. API Key Management:
    - Problem: Incorrect or invalid API keys led to failures in generating responses.
    - Solution: Ensured the API key was correctly set and valid. Added error handling to notify users of issues with the API key.

Future Scope

1. **Enhanced UI/UX**:
    - Improve the chatbot interface with richer interactions, such as buttons for common queries, and better formatting for responses.
2. **Backend Integration**:
    - Connect to a backend database to allow dynamic updates to the corpus and SAQ data without needing to modify the codebase.
3. **Multilingual Support**:
    - Extend the chatbot's capabilities to handle multiple languages, making it accessible to a broader audience.
4. **Voice Interaction**:
    - Add support for voice-based interactions, allowing users to speak their queries instead of typing them.
5. **Personalization**:
    - Implement user profiles and preferences to provide personalized recommendations and answers.
6. **Analytics Dashboard**:
    - Develop an analytics dashboard for the business to track user interactions, common queries, and chatbot performance metrics.

Conversion Tools

- **Google Docs**: Copy the markdown content, paste it into a Google Docs document, and download it as a PDF.
- **Markdown to PDF**: Use online tools like Dillinger (dillinger.io) to convert markdown to PDF.

Submitting the Repository

1. Repository is public.
2. Included the README.md and documentation.pdf in the root directory.
3. Tested all instructions to ensure they are accurate and the code runs without issues.