**Ema Intern Take-Home Challenge:**

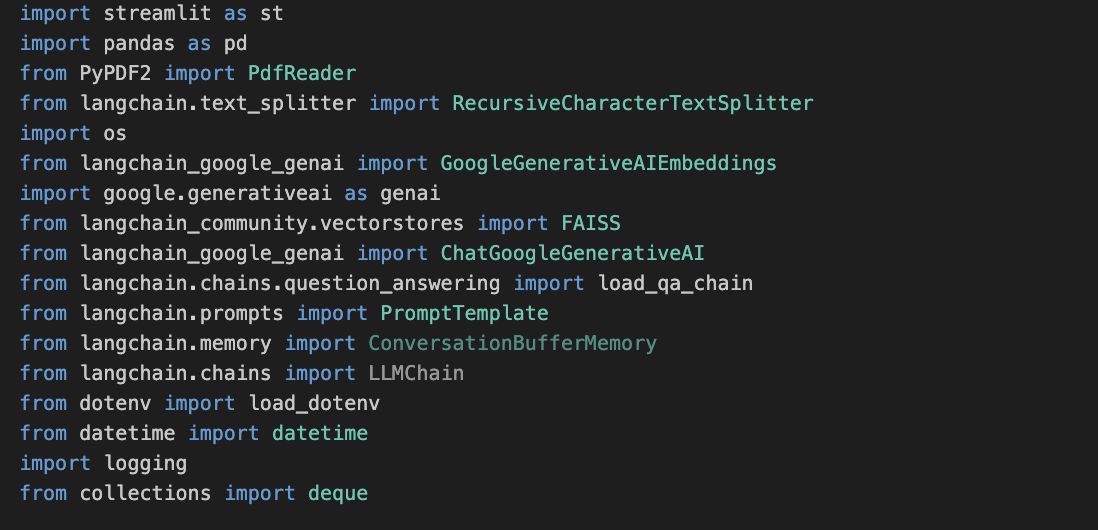
In this take-home challenge, we were to build a Natural Language Query Agent over some sample data. The purpose of this project is to demonstrate our ability to work with data, research techniques commonly used to solve such tasks and implement a trimmed down version of the approach to answer simple natural language queries.

**Here is my take on this Challenge:**

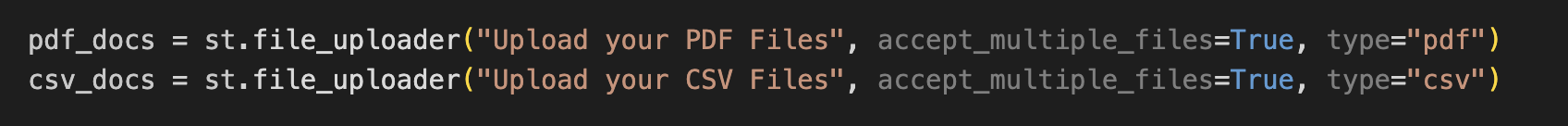
This documentation provides an overview of a Streamlit-based application designed to process PDF and CSV files, extract text, create a vector store using Google's Generative AI embeddings, and enable users to query the processed documents. The application leverages several cutting-edge technologies and frameworks to create a powerful tool for managing and querying document data.

1. **Key Features**
2. **Key Frameworks and libraries used:** Streamlit serves as the backbone for our user interface, handling elements such as file uploaders, text inputs, and buttons to create an interactive and user-friendly experience. We utilize Pandas to read and process CSV files, converting their contents into strings for easier manipulation and analysis. PyPDF2 is used to extract text from uploaded PDF files, enabling the app to handle various document formats seamlessly.

Langchain plays a crucial role by providing tools to work with large language models (LLMs), facilitating advanced text processing and AI interactions. Specifically, we use Langchain's text splitter to manage large texts by breaking them into smaller chunks, GoogleGenerativeAIEmbeddings for generating text embeddings, and FAISS for efficient storage and similarity searches of these embeddings. Additionally, ChatGoogleGenerativeAI interfaces with Google's generative AI to handle user queries and provide detailed responses. By combining these technologies, our app allows users to upload and query documents, leveraging the power of AI to extract meaningful insights and answers from their data.

****

1. **Upload and Process Multiple Files:** The application allows users to upload multiple PDF and CSV files simultaneously. This feature is crucial for handling large datasets and makes the tool versatile for various data formats. Upon upload, the application reads and extracts text from these files, preparing it for further processing.

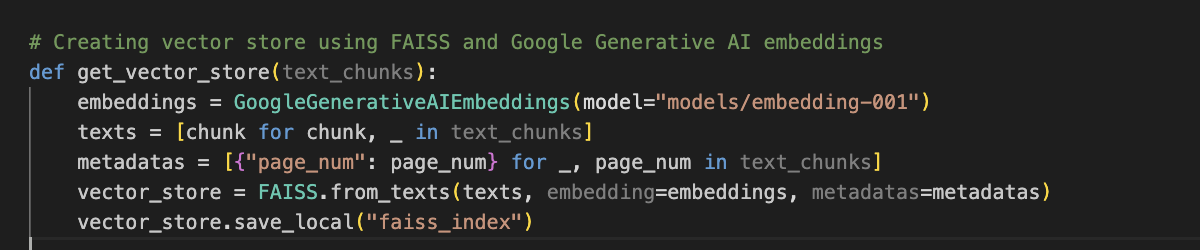
****

1. **Intermediary Representation and Data Transformation:**

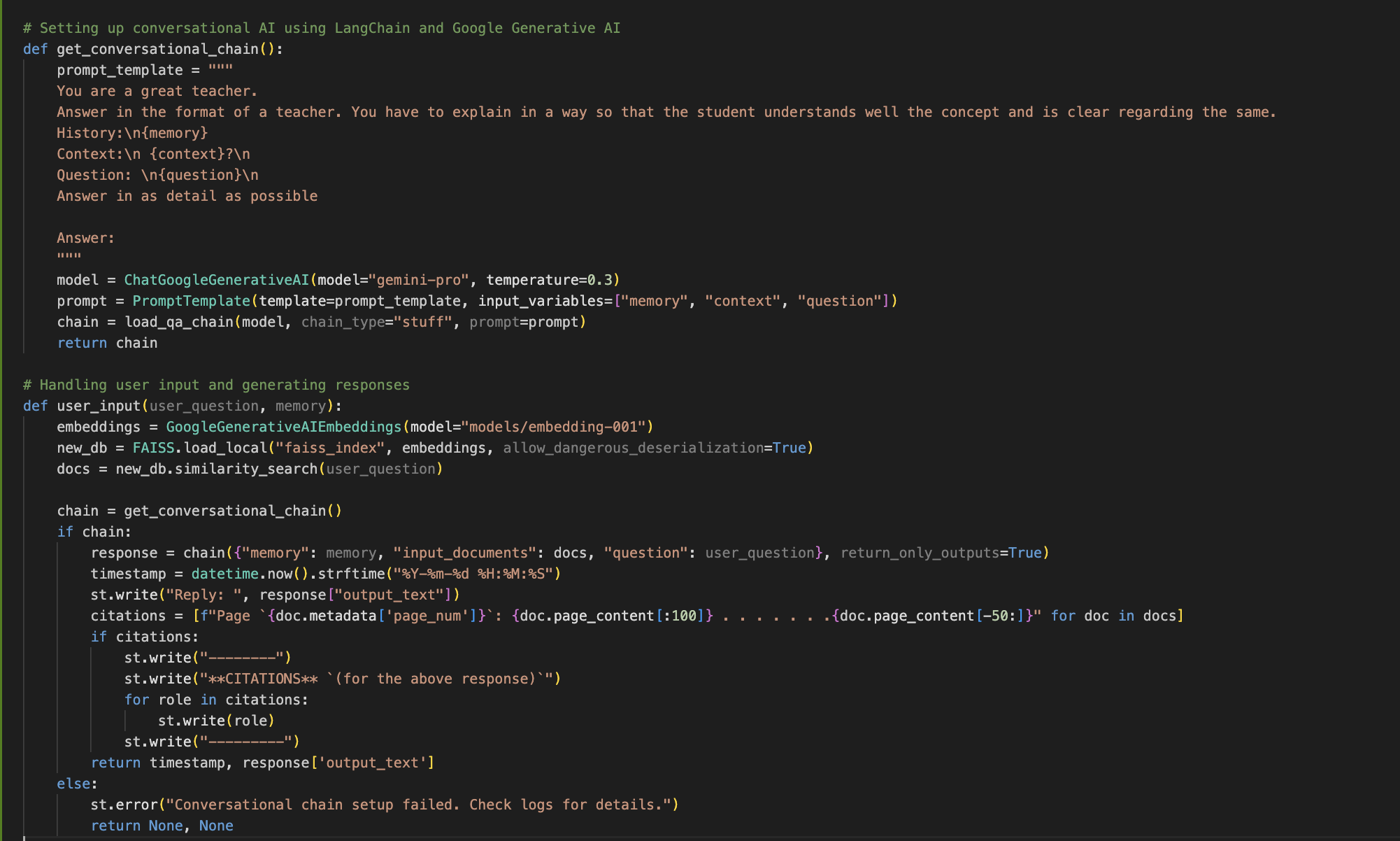
* **Text Extraction and Chunking:** Text extraction from PDFs and CSVs is accomplished using the PyPDF2 library for PDFs and pandas for CSVs. The extracted text is then split into manageable chunks using LangChain's RecursiveCharacterTextSplitter, which helps in handling large documents efficiently by breaking them into smaller pieces for easier processing.

****

* **Vector Store Creation:** Using Google Generative AI embeddings, the text chunks are transformed into embeddings and stored in a FAISS vector store. This step involves creating embeddings with the `GoogleGenerativeAIEmbeddings` model and saving the vector store locally. The vector store enables efficient similarity searches, allowing users to query the documents effectively.

****

* **Conversational AI for Querying Documents:** The application includes a conversational AI feature that uses LangChain's `load\_qa\_chain` to provide detailed answers to user queries. This feature leverages a custom prompt template designed to answer questions in an explanatory manner, suitable for teaching purposes. The conversational chain uses Google's Generative AI model to generate responses based on the context and memory of previous interactions.

****

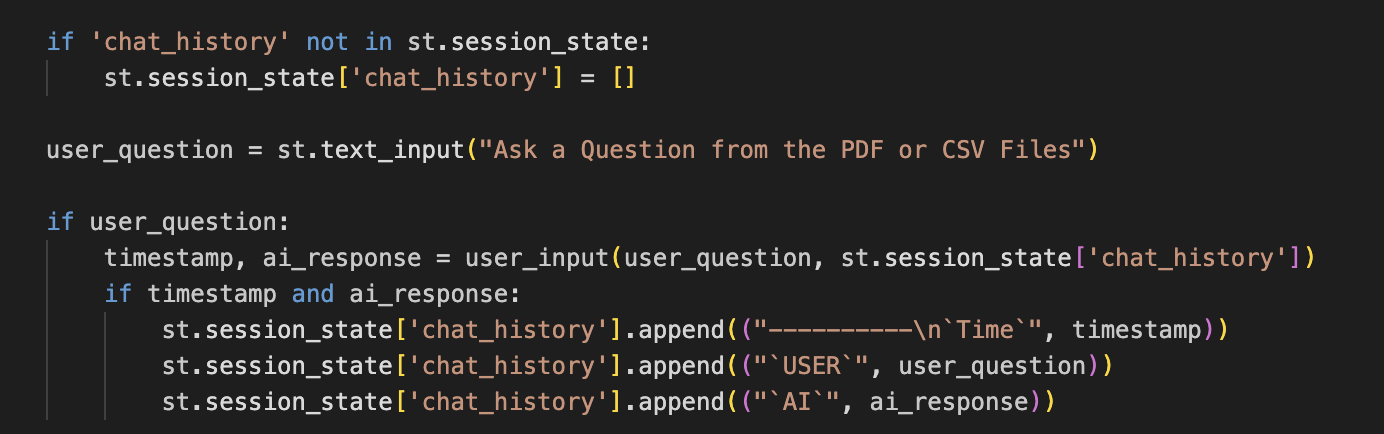
1. **Citing References:** To cite references and display sections from lecture notes used to compose the final conversational answer, we can modify the user\_input function to include citation details in the response.

* **Text Extraction and Chunks**: When processing the PDF or CSV files, text is extracted and split into chunks. Each chunk is associated with metadata such as the page number.
* **Storing Citations**: When the AI generates an answer, it also retrieves the most relevant document chunks. These chunks are then formatted to show citations from the original documents.
* **Displaying Citations**: The citations are displayed in the response to the user, providing context and references for the answer.

****

1. **Conversational Memory:** To enable the AI to maintain context across multiple interactions, allowing for follow-up questions that build on previous answers.

* **Session State for Memory**: The chat history is stored in the session state, which is retained across interactions within the same session.
* **Prompt Template**: The prompt template includes a section for "History" to pass previous interactions to the model, ensuring the context is maintained.
* **Appending History**: After each user interaction, the question and AI response are appended to the chat history.

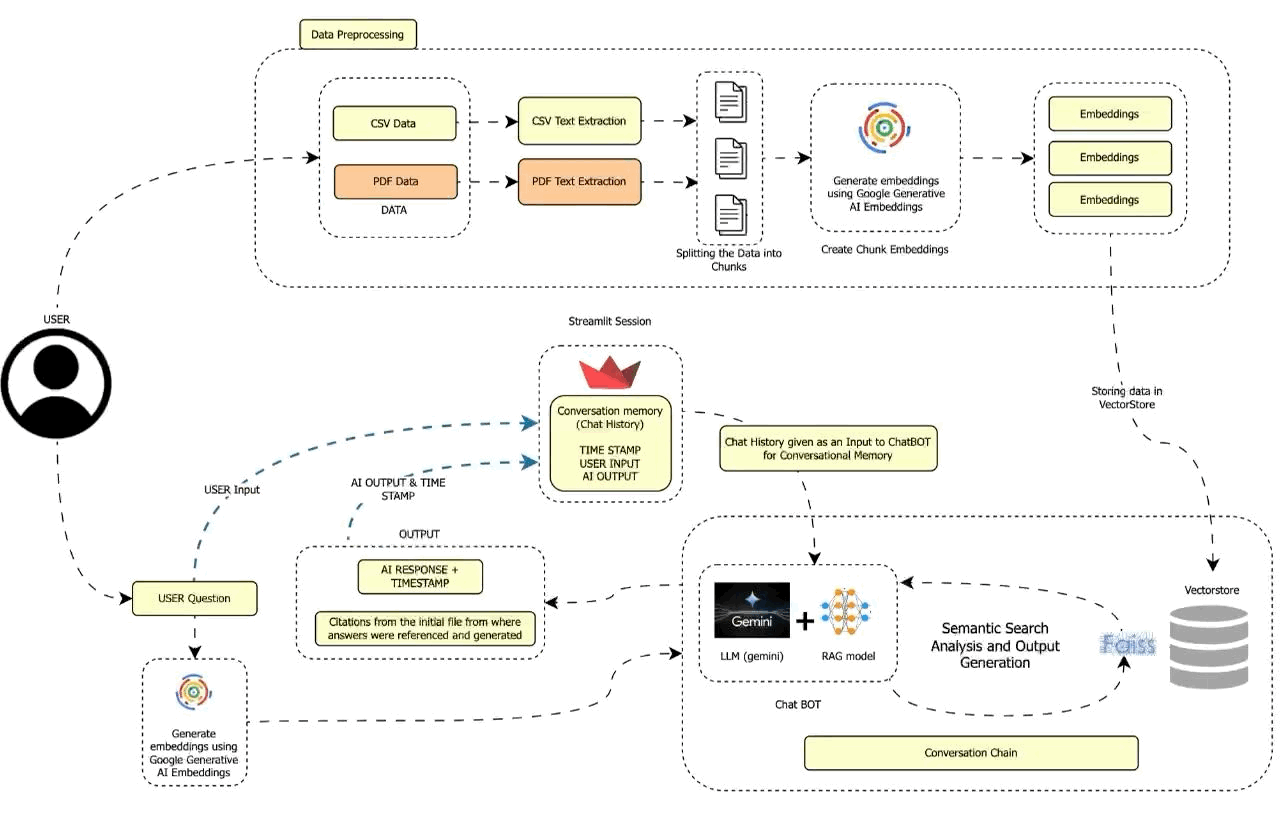
****

1. **Workflow and Approach used:**

**Pipeline Description:**

* *File Upload:* Users upload PDF and CSV files through the Streamlit interface.
* *Text Extraction:* Text is extracted from the uploaded files using PyPDF2 for PDFs and pandas for CSVs.
* *Text Chunking:* Extracted text is split into smaller chunks using LangChain's text splitter.
* *Vector Store Creation:* Text chunks are converted into embeddings using Google Generative AI and stored in a FAISS vector store.
* *Conversational AI:* A question-answering chain is set up using LangChain and Google's Generative AI to handle user queries and provide detailed responses.

**Workflow Flowchart:**

****

<https://github.com/AyonSOMADDAR/EMA_BOT/blob/main/ema.py>

1. **Areas of Improvement and Deployment Consideration**

* **Enhanced Error Handling**: Implementing more robust error handling mechanisms for various file formats and processing steps.
* **Optimized Text Chunking**: Fine-tuning the chunking process to handle edge cases and improve efficiency.
* **Scalability**: Ensuring the system can handle larger volumes of documents and more complex queries.
* **Cloud Deployment:** Deploy the containerized application on a cloud platform such as AWS, GCP, or Azure for scalability and reliability.
* **Database Integration:** Use cloud-based storage solutions for storing vector stores and embeddings to handle larger datasets and ensure persistence. We Use Elastic Search based vector management for High scalability and cloud deployments.

GitHub: <https://github.com/AyonSOMADDAR/EMA_BOT>