**NOME: Marco Antonio Gomes Caldeira Junior, Pedro Boro Guerra  
  
Documentação do Código**

Este código implementa uma aplicação de chatbot com suporte a upload de documentos PDF e interações baseadas em IA, utilizando Gradio para a interface, Groq para o LLM (Modelo de Linguagem), e ChromaDB para o armazenamento vetorial. A aplicação é focada em responder a perguntas sobre cafés de uma loja fictícia chamada *Serenatto*.

Aqui está o passo a passo detalhado do código:

**1. Importação de Módulos**

* **os** e **shutil** são usados para manipulação de arquivos e diretórios.
* **gradio** para criação da interface do usuário.
* **llama\_index** para a indexação de documentos e uso do modelo de IA.
* **chromadb** para interação com o banco de dados vetorial ChromaDB.
* **PyPDF2** para leitura de arquivos PDF.
* **TemporaryDirectory** da biblioteca **tempfile** para criação de diretórios temporários.

**2. Classe ChromaEmbeddingWrapper**

* Esta classe é um "wrapper" para a classe HuggingFaceEmbedding usada em ChromaDB, permitindo o uso de embeddings para documentos em modelos de IA.
* **Função \_\_call\_\_**: Aceita uma lista de strings e retorna a representação vetorial dos documentos utilizando o modelo de embedding configurado.

**3. Inicialização do Modelo de Embedding**

* O modelo de embeddings utilizado é o intfloat/multilingual-e5-large, inicializado tanto para o uso direto (embed\_model) quanto para o uso com ChromaDB (embed\_model\_chroma).

**4. Inicialização do ChromaDB**

* **ChromaDB** é inicializado com um cliente persistente em um diretório local (./chroma\_db) para armazenar os dados vetoriais.
* A coleção documentos\_serenatto é criada ou recuperada.
* **VectorStoreIndex** é configurado para ser usado com a coleção ChromaDB para armazenar vetores.

**5. Inicialização do LLM (Groq)**

* A API do Groq é configurada utilizando a chave de API fornecida no ambiente (GROQ\_API\_KEY).
* O modelo llama3-70b-8192 é carregado para usar com o chatbot.

**6. Função process\_pdf**

* **Objetivo**: Processar o PDF enviado pelo usuário e preparar os dados para o chatbot.
* **Passos**:
  1. O PDF é copiado para um diretório temporário.
  2. O texto é extraído de todas as páginas do PDF utilizando PdfReader.
  3. O texto extraído é salvo em um arquivo temporário de texto.
  4. O **SimpleDirectoryReader** carrega os dados do diretório temporário e o **SentenceSplitter** divide o conteúdo em blocos menores (nós) de até 1200 caracteres.
  5. A indexação dos documentos é realizada, criando um **VectorStoreIndex** com o uso do modelo de embedding configurado.
  6. A memória do chatbot é configurada com o **ChatSummaryMemoryBuffer** para limitar o histórico de mensagens.
  7. Um chatbot é criado com base no modelo LLM e no contexto dos dados carregados.
  8. A função retorna uma mensagem de sucesso ou erro.

**7. Função converse\_com\_bot**

* **Objetivo**: Gerenciar a conversa entre o usuário e o chatbot, mantendo o histórico de mensagens.
* **Passos**:
  1. Se o chatbot não estiver carregado, solicita que o usuário envie um PDF primeiro.
  2. Se o chatbot estiver disponível, o texto da mensagem do usuário é enviado ao **chat\_engine** para gerar uma resposta.
  3. O histórico de mensagens é atualizado com a entrada do usuário e a resposta do assistente.

**8. Função resetar\_chat**

* **Objetivo**: Limpar o histórico de chat e resetar o estado do chatbot.
* **Passos**:
  1. A função limpa o estado do chatbot, reiniciando a conversa.
  2. Retorna um histórico de chat vazio.

**9. Interface Gradio**

* **Estrutura**:
  + Um título é exibido no topo da interface.
  + Um campo de upload permite que o usuário envie o arquivo PDF.
  + O botão **"Carregar PDF"** chama a função process\_pdf para processar o arquivo PDF.
  + Um **textbox de status** exibe a mensagem de status após o upload do PDF.
  + **Chatbot**: Um componente de chat interativo onde o usuário pode enviar mensagens.
  + **Campo de entrada de texto** para digitar as mensagens.
  + **Botão "Limpar"** limpa o histórico da conversa.
* **Funções Gradio**:
  + **upload\_button.click**: Quando o botão de upload é clicado, ele chama a função process\_pdf e limpa o chat.
  + **msg.submit**: Quando o usuário envia uma mensagem, a função converse\_com\_bot é chamada para processar a mensagem e atualizar o histórico de chat.
  + **limpar.click**: Chama a função resetar\_chat para limpar o histórico de chat.

**10. Lançamento da Interface**

* O código finaliza com o lançamento da interface usando app.launch(debug=True), que permite a interação com o usuário de forma simples e intuitiva.

**Resumo**

Este código cria um chatbot interativo, onde os usuários podem carregar arquivos PDF que contêm informações sobre café da loja Serenatto. O chatbot utiliza modelos de linguagem e embeddings para processar o conteúdo e responder de maneira contextualizada e personalizada, com a interface criada por Gradio.