Configurar la aplicación Python

Vamos a crear nuestra carpeta de proyecto, la llamaremos :`chroma-langchain-demo`

```
mkdir chroma-langchain-demo
```

Entremos en el nuevo directorio y creemos nuestro archivo principal:`cd.py`

```
cd chroma-langchain-demo
touch main.py
```

(Opcional) Ahora, crearemos y activaremos nuestro entorno virtual:

```
python -m venv venv
source venv/bin/activate
```

Instalar el SDK de Python de OpenAI

Genial, con la configuración anterior, instalemos el SDK de OpenAI usando:`pip`

```
pip install openai
```

Paso 2: Instalar Chroma y LangChain

Instalación de Chroma

Necesitaremos instalar usando . En la ventana de su terminal, escriba lo siguiente y presione retorno:`chromadbpip`

```
pip install chromadb
```

Instalar LangChain, PyPDF y tiktoken

Hagamos lo mismo para , (necesario para más abajo), y que es un cargador de PDF para LangChain. También utilizaremos :`langchaintiktokenOpenAIEmbeddingsPyPDFpip`

```
pip install langchain pypdf tiktoken
```

Paso 3: Crear un almacén de vectores a partir de fragmentos

Descargue y coloque un archivo en una carpeta en la raíz del directorio de su proyecto:`/data`

Esta debería ser la ruta de su archivo PDF: .`chroma-langchain-demo/data/document.pdf`

En nuestro archivo, vamos a escribir código que cargue los datos de nuestro archivo PDF y los convierta en incrustaciones vectoriales utilizando el modelo OpenAI Embeddings.`main.py`

Utilice el cargador de PDF de LangChain `PyPDFLoader`

```
from langchain.document_loaders import PyPDFLoader
from langchain.embeddings.openai import OpenAIEmbeddings


loader = PyPDFLoader("data/document.pdf")
docs = loader.load_and_split()
```

Aquí usamos el para ingerir nuestro archivo PDF. carga el contenido del PDF y luego lo divide en trozos usando el método.`PyPDFLoaderPyPDFLoaderload_and_split()`

Pasar los fragmentos al modelo de incrustaciones de OpenAI

Bien, a continuación tenemos que definir qué LLM y modelo de incrustación vamos a utilizar. Para hacer esto, definamos nuestras variables y como se muestra a continuación:`llmembeddings`

```
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0.8)


embeddings = OpenAIEmbeddings()
```

Usando el , nuestros fragmentos se pasarán al modelo y luego se devolverán y persistirán en el directorio de la colección, como se muestra a continuación:`Chroma.from_documentsdocsembeddingsdatalc_chroma_demo`

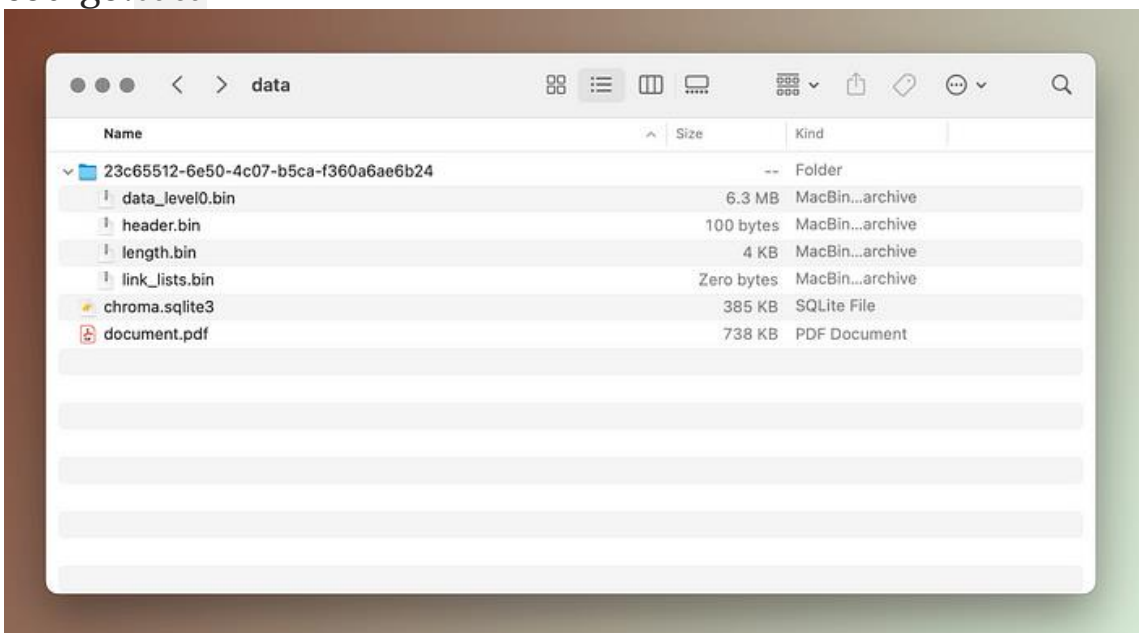
```
chroma_db = Chroma.from_documents(
    documents=docs,
    embedding=embeddings,
    persist_directory="data",
```

```
    collection_name="lc_chroma_demo"
)
```

Veamos los parámetros:

- `embedding`: permite a LangChain saber qué modelo de incrustación utilizar. En nuestro caso, estamos usando `.OpenAIEmbeddings()`
- `persist_directory`: Directorio donde queremos almacenar nuestra colección.
- `collection_name`: Un nombre amigable para nuestra colección.

Así es como se ve nuestro directorio después de ejecutar este código:`data`



Paso 4: Realizar una búsqueda de similitud localmente

¡Eso fue fácil! ¿Verdad? Una de las ventajas de Chroma es su eficiencia a la hora de manejar grandes cantidades de datos vectoriales. Para este ejemplo, estamos usando un PDF pequeño, pero en su aplicación del mundo real, Chroma no tendrá problemas para realizar estas tareas en muchas más incrustaciones.

Vamos a realizar una búsqueda de similitud. Esto simplemente significa que, dada una consulta, la base de datos encontrará información similar de las incrustaciones vectoriales almacenadas. Veamos cómo se hace esto:

```
query = "What is this document about?"
```

A continuación, podemos utilizar el método:`similarity_search`

```
docs = chroma_db.similarity_search(query)
```

Otro método útil es , que también devuelve la puntuación de similitud representada como un decimal entre 0 y 1. (1 es una coincidencia perfecta).`similarity_search_with_score`

Paso 5: Consultar el modelo

Tenemos nuestra consulta y documentos similares en la mano. Vamos a enviarlos al modelo de lenguaje grande que definimos anteriormente (en el paso 3) como .`llm`

Vamos a usar la cadena de LangChain y pasar algunos parámetros como se muestra a continuación: `RetrievalQA`

```
chain = RetrievalQA.from_chain_type(llm=llm,
                                    chain_type="stuff",
retriever=chroma_db.as_retriever())

response = chain(query)
```

Lo que esto hace es crear una cadena de tipos, usar nuestro definido , y nuestro almacén de vectores Chroma como un recuperador.`stuffllm`