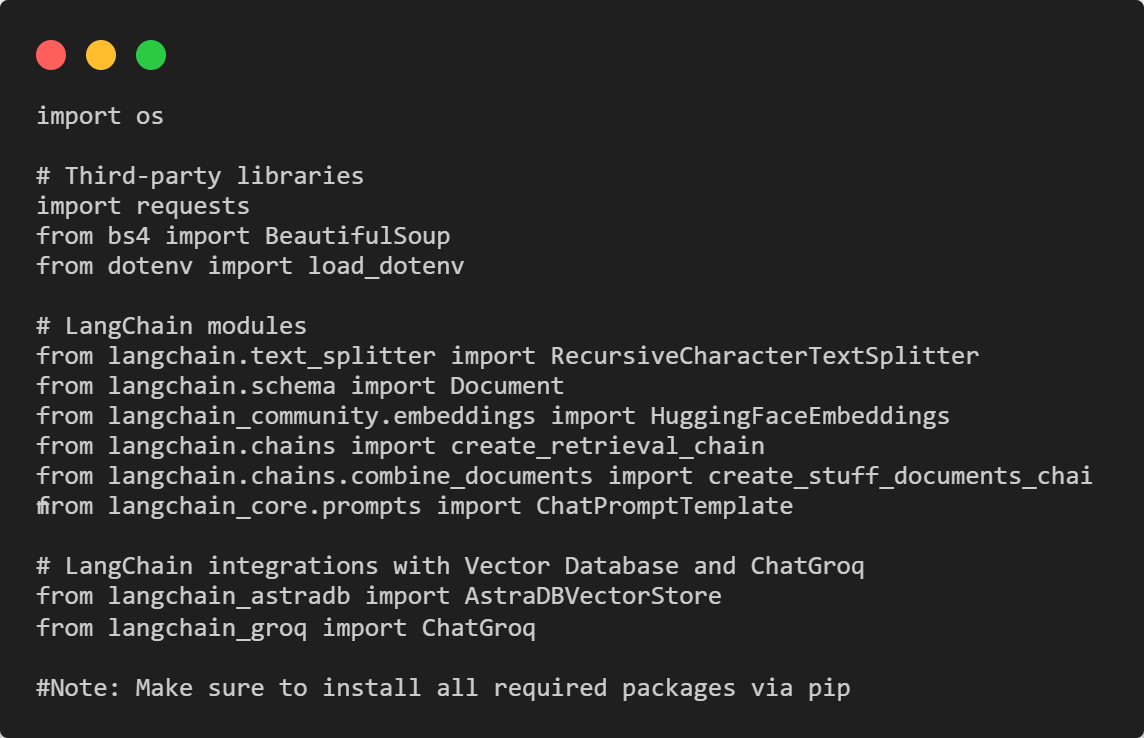
**Building RAG Pipeline using Lang Chain, Astra DB and Llama3-8b-8192**

Our main aim is to build a robust RAG pipeline with Groq (Model: Llama3-8b-8192) LLM, Lang Chain, and Astra DB which provides a powerful solution for enhancing information retrieval and generation. By integrating these technologies, we can develop an efficient and scalable system capable of processing and generating high-quality responses from diverse web data sources.

**Steps to Achieve our Goal**

* Importing the necessary libraries
* Web Scraping: Extract content from web pages using Beautiful Soup.
* Text Processing: Split the extracted content into chunks using RecursiveCharacterTextSplitter.
* Text Embedding: Convert text chunks into vector embeddings using HuggingFaceEmbeddings.
* Storage: Store vectors, content, and metadata in Astra DB.
* Building the RAG Pipeline: Integrate all components with Lang Chain to enable retrieval-augmented generation.

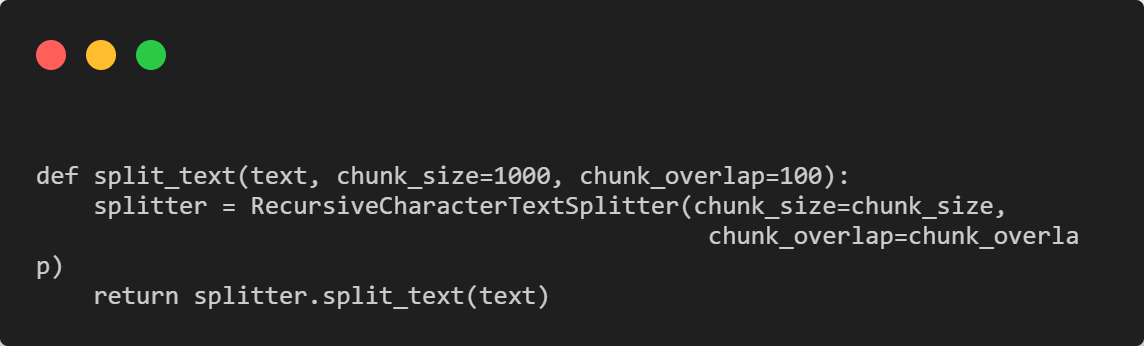
Let’s import necessary libraries for building the project as shown in the snippet below:



Perform the task of Web scrapping using BeautifulSoup. For this we will create a function that is **def fetch\_and\_parse\_url(url):** which will perform essential task and will produce clean text. Code snippet is as follows:



Let’s break a large text document into smaller, manageable chunks using Lang Chain class RecursiveCharacterTextSplitter. To achieve our task we will write a function split\_text mentioning chunk size of 1000 characters and chunk overlap of 100 characters as in the code snippet below:



Let’s mention the URL and call the function as mentioned below:

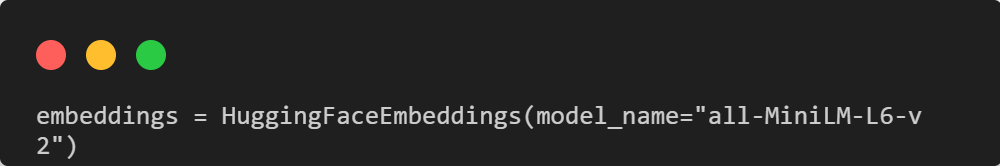
url = "https://edition.cnn.com/"

full\_text, metadata = fetch\_and\_parse\_url(url)

On successful execution of code we will retrieve clean text in chunks of 1000 characters and chunk overlap of 100 characters.

Embed the text extracted using HuggingFaceEmbeddings with model name: all-MiniLM-L6-v2

**Note**: We can use various embedding techniques, such as those provided by models from OpenAI or Ollama. However, inserting data into a Vector Store, especially when using Ollama embeddings with a RAM capacity of 8 GB, is time-consuming (It takes more than hours). To address this, I used **HuggingFaceEmbeddings**, which is quite efficient and allows data insertion to happen in seconds with a 384-dimensional vector.



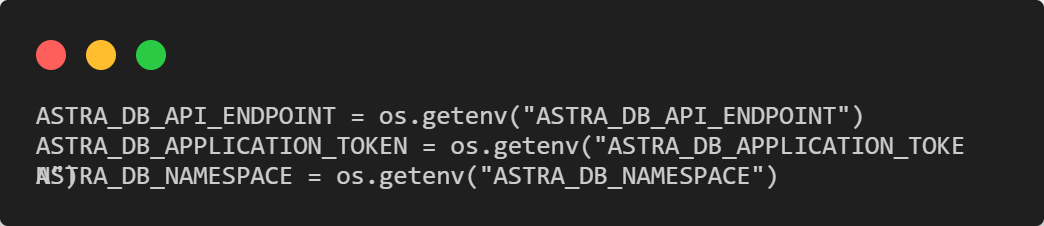
Let’s make connection to Astra DB. Before we make connection to Astra DB we need to login to Astra DB. We can login using our Google account or GitHub account and need to fetch important details as mentioned below:

ASTRA\_DB\_API\_ENDPOINT

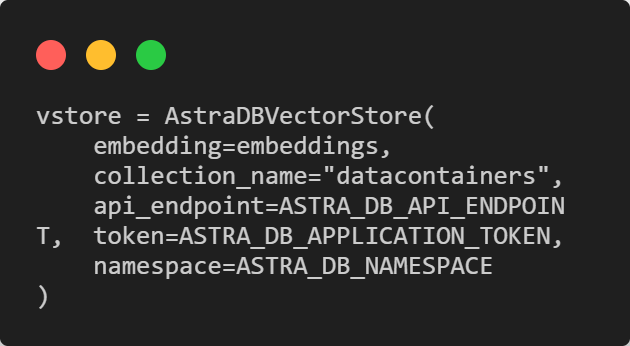
ASTRA\_DB\_APPLICATION\_TOKEN

ASTRA\_DB\_NAMESPACE

Store the details in .env file so that our secret keys are secured and access those keys from environment variables. Code snippet is as mentioned below:



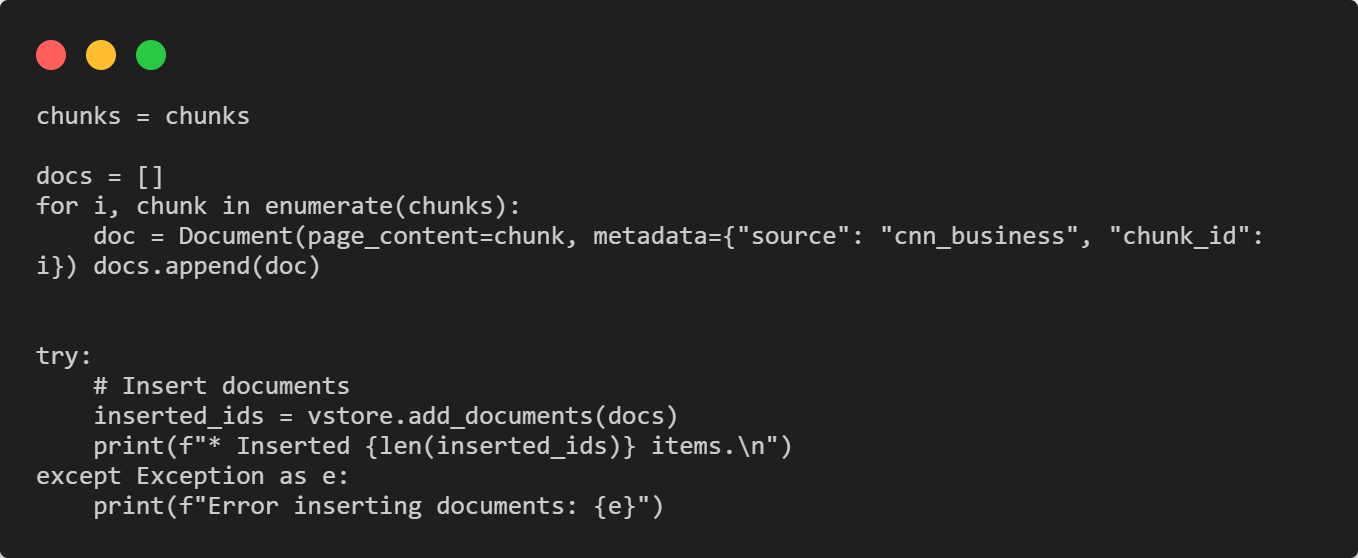
After successful Connection to the Vector Database (Astra DB) we need to create an instance of the AstraDBVectorStore class from the langchain\_astradb library. This instance represents a vector store that interacts with Astra DB. Code snippet is as mentioned below:



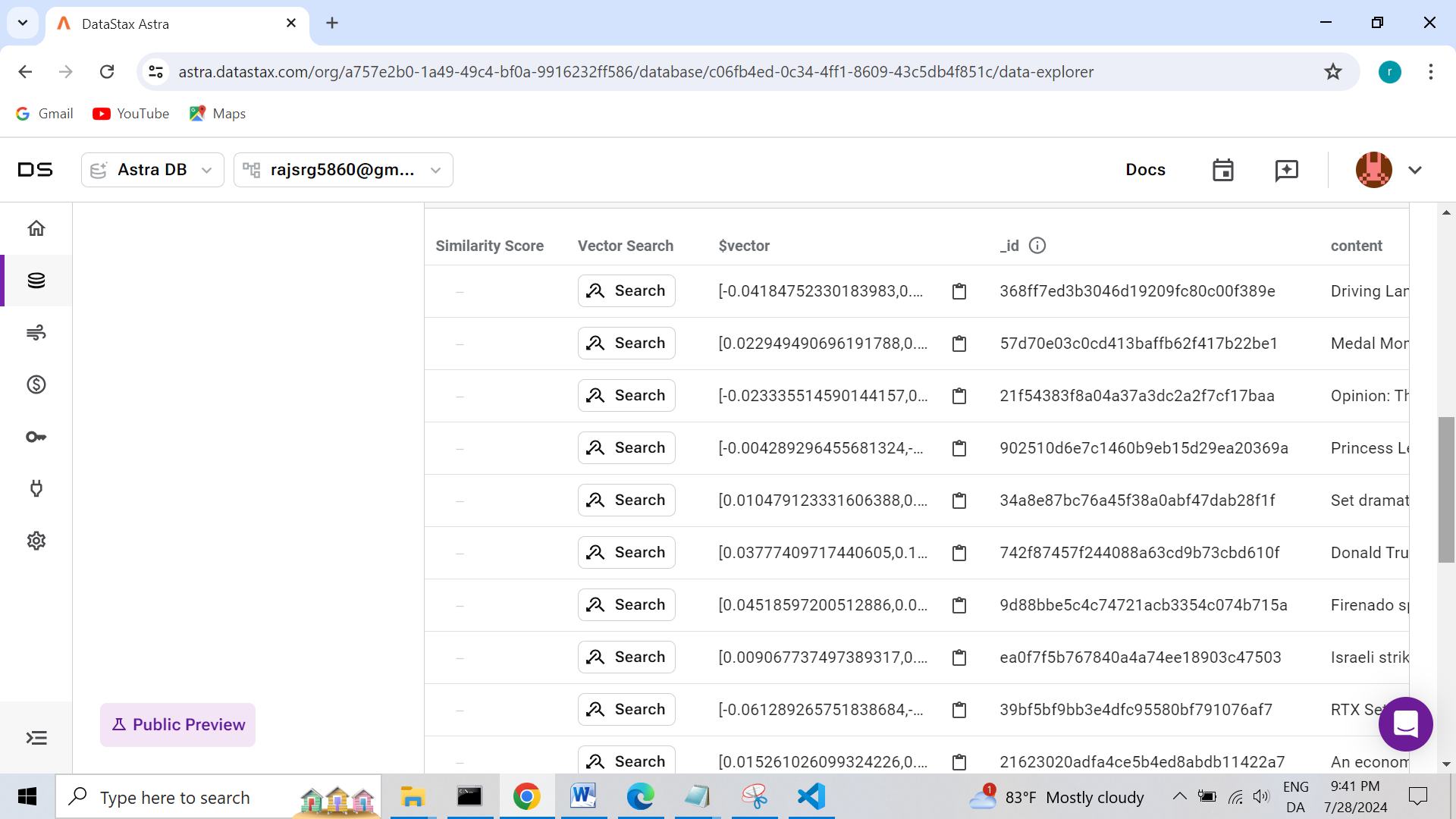
**Note**: In the above code snippet Collection\_name is the name of the table where the data is stored. I have mentioned “datacontainers”.

Prepare the data for data insertion to happen successfully into the Vector Store AstraDBVectorStore which stores vector embeddings in a database.

For all these things to happen successfully, we will initialize an empty list called docs where each chunk will be stored as a Document object. Convert Chunks to Documents and insert Documents into Vector Store. Code snippet is as mentioned below:



After successfully execution of the code we see data populated in Astra DB. It contains Vector, id, content, metadata as shown in the Screen shot below:

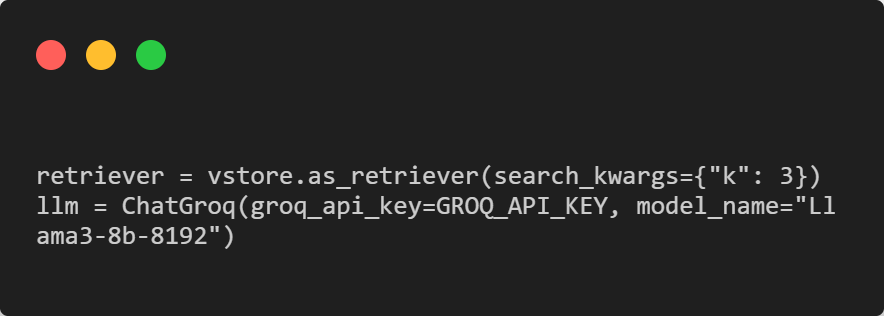


After successful insertion of data in Vector Store we will build up the RAG PIPE line so that we can build up effective Question Answer system for the inserted data into the Vector Store using LLM model Llama3-8b-8192(Latest language model released by Meta).

Load the GROQ\_API\_KEY from Environment Variables as mentioned below:



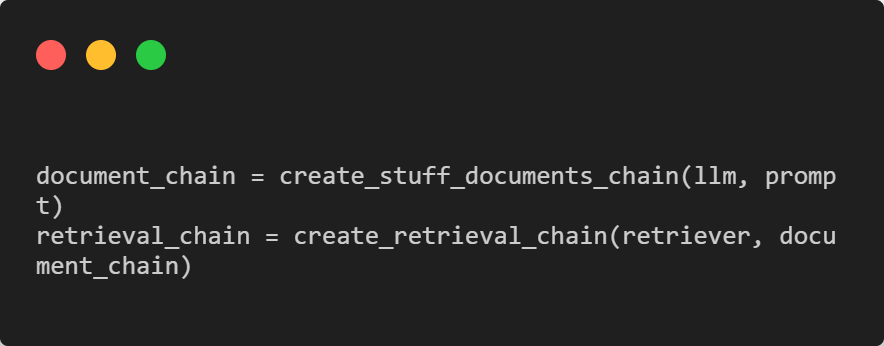
Set up document retriever with a search parameter to retrieve up to 3 relevant documents and initialize ChatGroq language model with the specified API key and model name (Llama3-8b-8192).



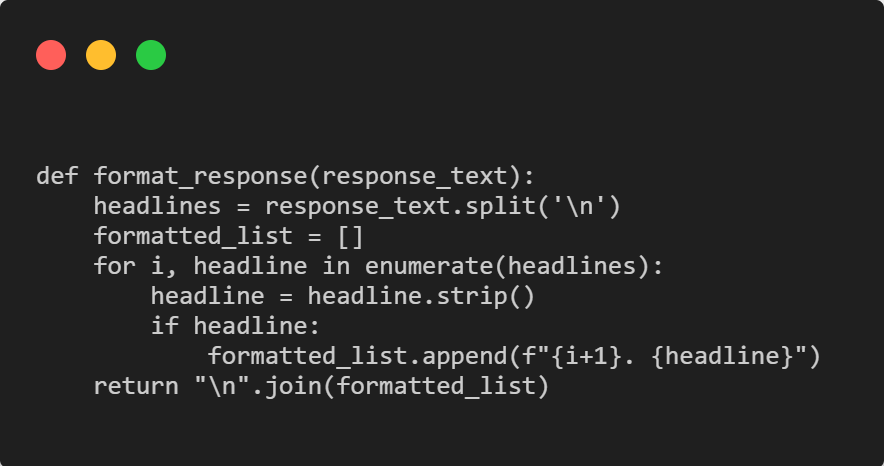
Create a prompt template for the language model. This template instructs the model to extract and format news headlines in a bullet-point list based on the provided context and question.



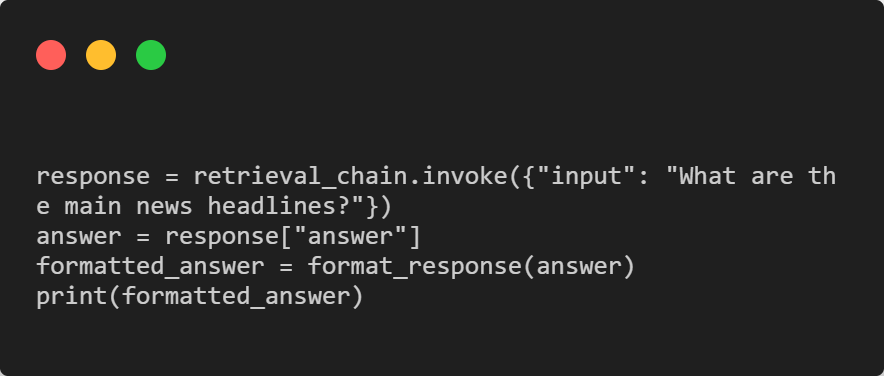
Create a chain that combines the language model with the prompt template to process documents, Combines the document retriever and the document chain to retrieve relevant documents and then process them with the language model.



Build a function to format the model’s response. It splits the response text into lines, trims whitespace, and formats each line into a numbered bullet-point list.



Execute the retrieval chain with the question "What are the main news headlines?" to get a response. It extracts the answer from the response, formats it using the format\_response function, and prints the result.



**We have successfully built RAG Pipeline using Lang Chain, Astra DB and Llama3-8b-8192**