**Project Statement :Enhancing Search Engine Relevance for Video Subtitles**

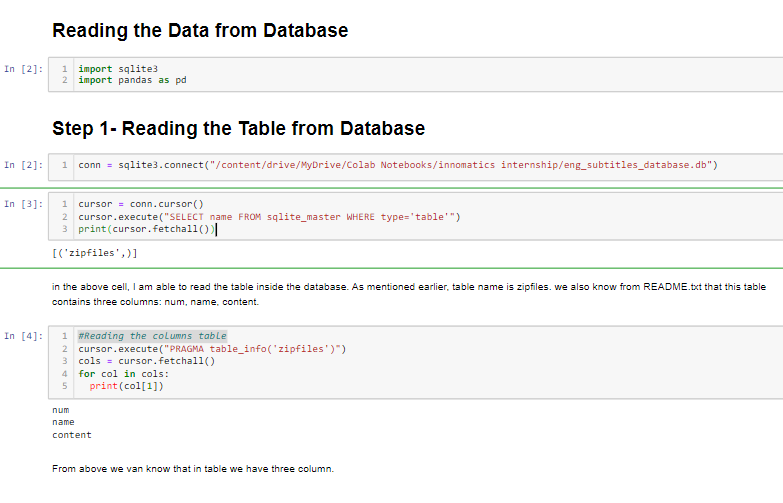
**Objective:**

Develop an advanced search engine algorithm that efficiently retrieves subtitles based on user queries, with a specific emphasis on subtitle content. The primary goal is to leverage natural language processing and machine learning techniques to enhance the relevance and accuracy of search results.

**Introduction:**

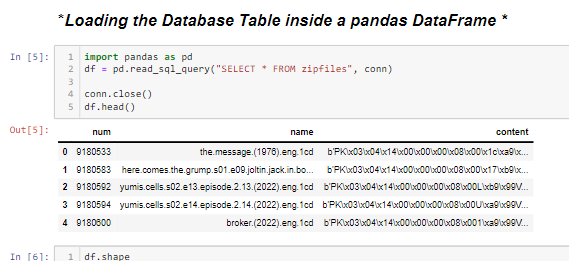
A search engine is a crucial software tool that facilitates users in findinginformation across the vast expanse of the World Wide Web. In the vast realm of data, our endeavor encompasses a diverse array of information. Presently, we embark on a journey where we import data from databases, transforming them into embedding vectors, and securely storing them in our chroma database. Subsequently, we engage with user input, seamlessly converting it into embedding vectors. Leveraging the power of cosine similarity, we navigate through this rich tapestry of information to curate personalized movie titles that resonate with relevance and allure.

# HERE STEPS : ***Step 1- Reading the Table and columns from Database***



It establishes a connection to an SQLite database located at "/content/drive/MyDrive/Colab Notebooks/innomatics internship/eng\_subtitles\_database.db".It creates a cursor object associated with the database connection.It executes a SQL query to retrieve the names of all tables present in the database and prints the result using **fetchall()**.Then, it executes another SQL query to fetch information about the columns in the "zipfiles" table using the **PRAGMA table\_info()** command.It iterates over the fetched column information and prints the name of each column.

***Loading the Database Table inside a pandas DataFrame***



This code snippet employs the pandas library to interface with an SQLite database. Initially, it imports the pandas library for data manipulation, assigning it the alias "pd". Subsequently, it utilizes the **read\_sql\_query()** function from pandas to execute an SQL query against the specified SQLite database connection (**conn**). The query selects all columns (**\***) from the "zipfiles" table. The retrieved data is then stored in a pandas DataFrame labeled **df**. Afterward, the code closes the database connection to free up system resources. Finally, it displays the first few rows of the DataFrame using the **head()** method, providing a concise preview of the retrieved data from the "zipfiles" table.

From the content, it appears to start with the bytes "PK\x03\......", which suggests that it might be a ZIP archive file. How do I know it? Experience! I have worked with something similar earlier.

## *****Unzipping the content of 385th row and decoding using******latin-1*

## 

## ****Applying the above Function on the Entire Data:****

## This Python code defines a function named clean\_data(data) designed for text preprocessing. It incorporates various cleaning techniques such as removing timestamps, dialogue indices, escape sequences, HTML tags, and certain types of links from the input text. Additionally, it converts the text to lowercase for consistency. The purpose of this function is to prepare textual data, often extracted from subtitles, for subsequent analysis or processing by eliminating irrelevant elements and standardizing the text format.

## 

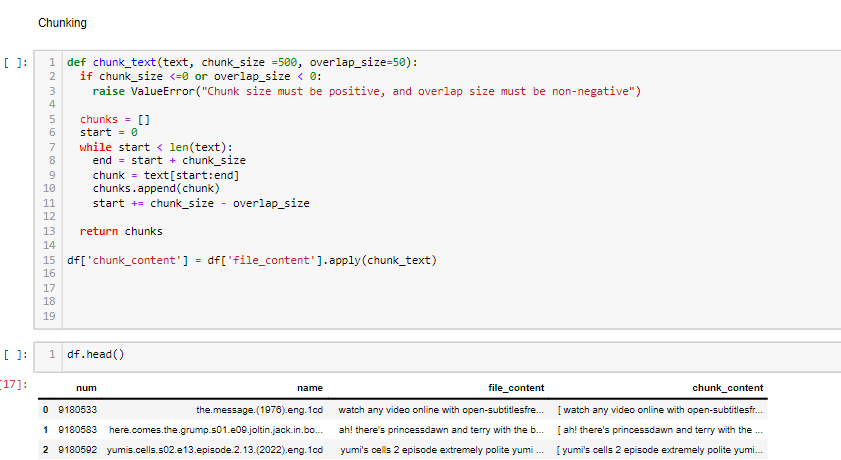
## *Saving to Dataframe csv format:*

## 

## 

## By breaking down large datasets into smaller, more manageable chunks, AI systems can process information more seamlessly, leading to improved performance and scalability. This approach not only facilitates smoother training processes but also enables models to generalize better and generate more coherent outputs. Through data chunking, AI systems can better capture the underlying patterns and structures within the data, ultimately enhancing their ability to generate meaningful and contextually relevant outputs.

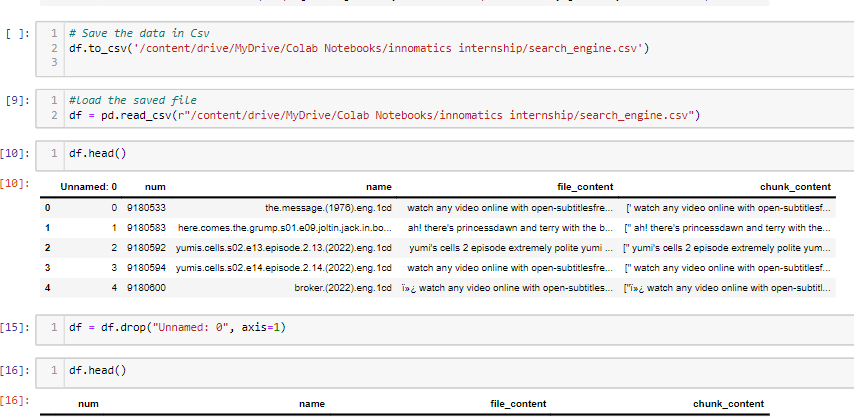
## 



function named **chunk\_text()** tailored for text chunking, often utilized in Natural Language Processing (NLP) tasks. It's designed to segment lengthy texts into smaller, more manageable chunks to facilitate analysis or processing. The function accepts three parameters: the **text** to be chunked, the **chunk\_size** defining the length of each chunk (default set to 500 characters), and the **overlap\_size** indicating the overlap between adjacent chunks (default set to 50 characters).

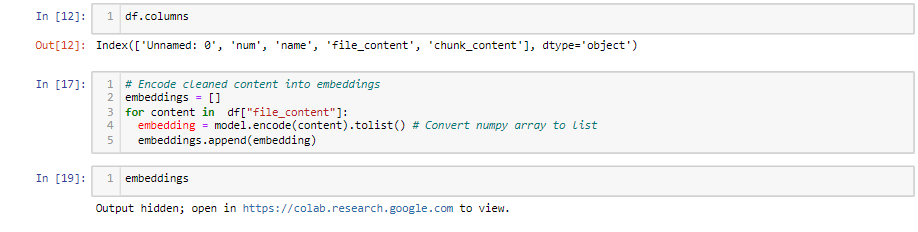
Within the function, error handling ensures that both the **chunk\_size** is positive and the **overlap\_size** is non-negative. Then, it initializes an empty list **chunks** to store the segmented text. Using a while loop, the function iterates over the input text, slicing it into chunks of the specified size while accounting for the overlap. Each chunk is appended to the **chunks** list.

Finally, the function returns the list of chunks. This approach enables efficient processing of large textual datasets, allowing AI systems to analyze or generate insights on smaller segments at a time. The code can be seamlessly integrated into NLP pipelines, enhancing the scalability and performance of text-based applications.



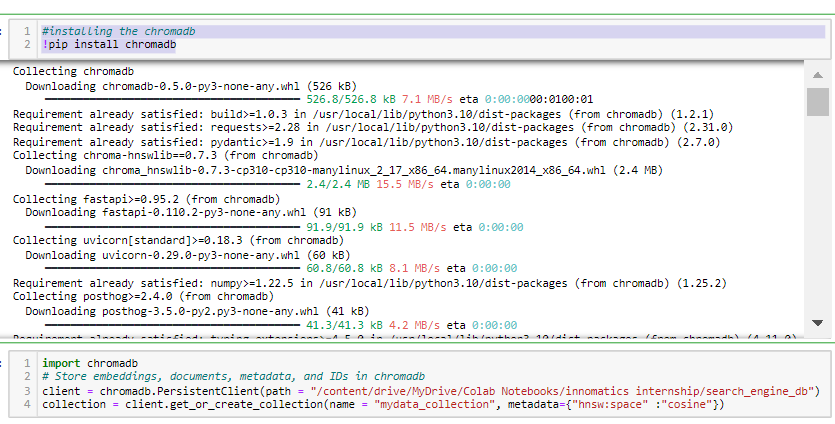
Transformers library is utilized to incorporate advanced sentence embedding capabilities into the Python environment. The first line imports the necessary module from the library, enabling access to its functionalities. Following this, a pre-trained embedding model named 'paraphrase-MiniLM-L3-v2' is loaded using the **SentenceTransformer()** function, and assigned to the variable **model**.

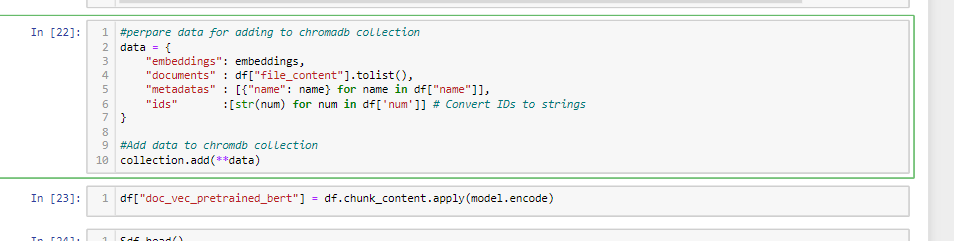
model, 'paraphrase-MiniLM-L3-v2', is renowned for its ability to generate dense representations of sentences, capturing semantic similarities with high precision. By leveraging pre-trained models like this, developers can expedite the process of embedding sentences into high-dimensional vector spaces without the need for extensive training on large datasets. These embeddings serve as potent tools for various natural language processing tasks, including semantic search, text classification, and information retrieval. Overall, this code snippet lays the foundation for harnessing the power of advanced sentence embeddings within Python applications, enabling the development of sophisticated NLP solutions.



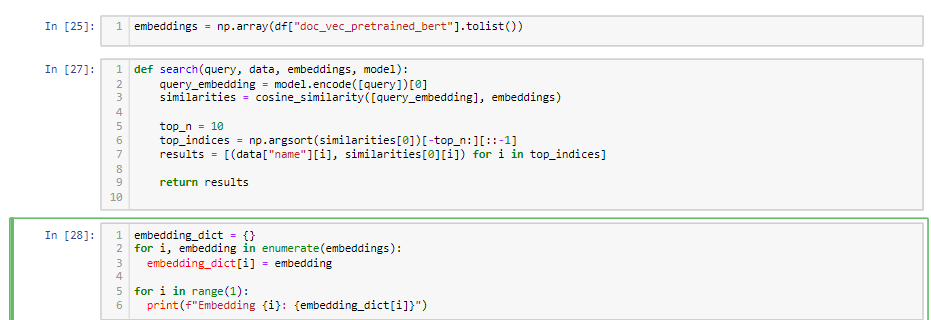
the cleaned content from the DataFrame is encoded into embeddings using the pre-trained Sentence Transformer model. Initially, an empty list named **embeddings** is initialized to store the generated embeddings. Then, for each **content** in the "file\_content" column of the DataFrame, the model's **encode()** function is applied to transform the text into embeddings. These embeddings are converted from NumPy arrays to lists and appended to the **embeddings** list. This iterative process ensures that each piece of content is accurately represented as an embedding, facilitating subsequent analysis or processing tasks. Overall, this code segment demonstrates the integration of pre-trained models to generate meaningful embeddings from textual data efficiently.

***Saving to chromaDB:***

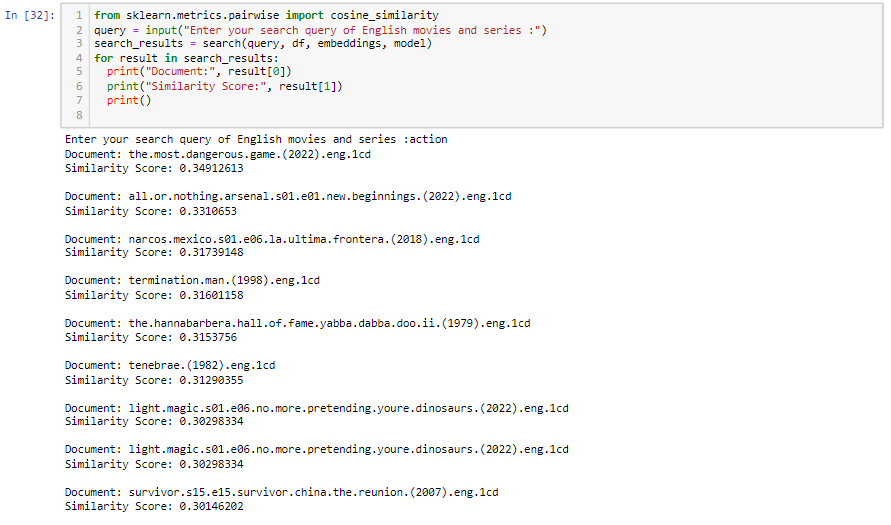
******

This code snippet seems to be setting up a connection to a database using ChromaDB, a library for managing embeddings, documents, metadata, and IDs. It starts by importing the ChromaDB module. Then, it initializes a **PersistentClient** object to connect to a ChromaDB database stored at the specified path, likely in Google Drive given the **/content/drive/MyDrive/Colab Notebooks/** prefix. The **get\_or\_create\_collection** method is called on the client to either retrieve an existing collection named "mydata\_collection" or create a new one if it doesn't exist. Metadata is provided to specify the space as cosine for similarity search, potentially indicating that this database is being used for similarity-based retrieval tasks. Overall, this code sets up a connection to a database and prepares a collection for storing and querying data, likely for a search engine or similar application. 

It’s a process to prepare and add data to a ChromaDB collection. It first organizes the data into a dictionary with keys representing different aspects such as embeddings, documents, metadata, and IDs. The embeddings likely represent numerical representations of the documents, while the documents themselves are strings of text content fetched from a DataFrame. Metadata, possibly containing additional information about the documents, is structured as a list of dictionaries. Finally, unique identifiers for each document are converted into strings to form the IDs.



to define a search function for retrieving the most similar documents to a given query within a dataset. It starts by calculating the query embedding using a pre-trained model, presumably a BERT-based model, and then computes the cosine similarity between this query embedding and the embeddings of the documents in the dataset. The top N most similar documents are then identified based on these cosine similarity scores, and their names along with their similarity scores are stored in the results.



enables users to input a search query related to English movies and series. Upon receiving the query, the code utilizes a pre-defined search function, likely employing a BERT-based model, to find the most similar documents within a dataset based on embeddings. The search results, comprising document names and their corresponding similarity scores to the query, are then displayed to the user. This functionality enables users to quickly retrieve relevant documents from the dataset based on their query, facilitating efficient exploration and analysis of English movies and series data. Overall, the code streamlines the process of searching for and presenting relevant information, enhancing user experience and productivity when interacting with the dataset.

***Implementation in vs code using streamlit application:***

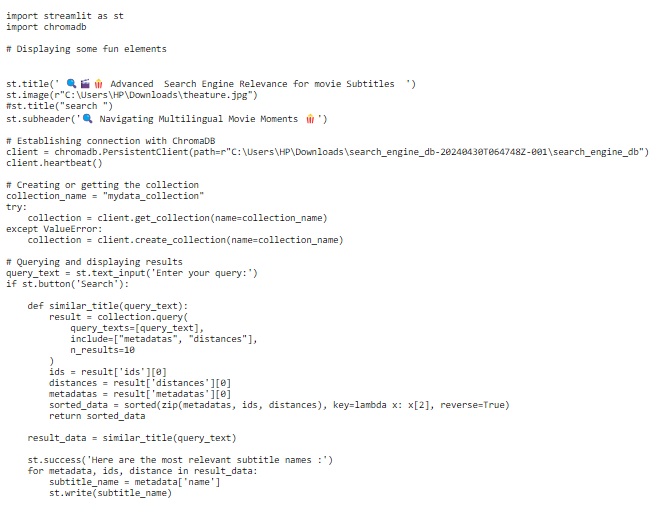
Streamlit code constructs an interactive web application for a sophisticated movie subtitle search engine. It starts by presenting a visually appealing interface with a title, an image, and a subheader that collectively convey the purpose of the tool, which is to facilitate the exploration of multilingual movie moments through subtitles,

Core functionality lies in its integration with ChromaDB, a database management library. The application establishes a connection with ChromaDB and either creates or retrieves a collection named "mydata\_collection", where the movie subtitle data is presumed to be stored,

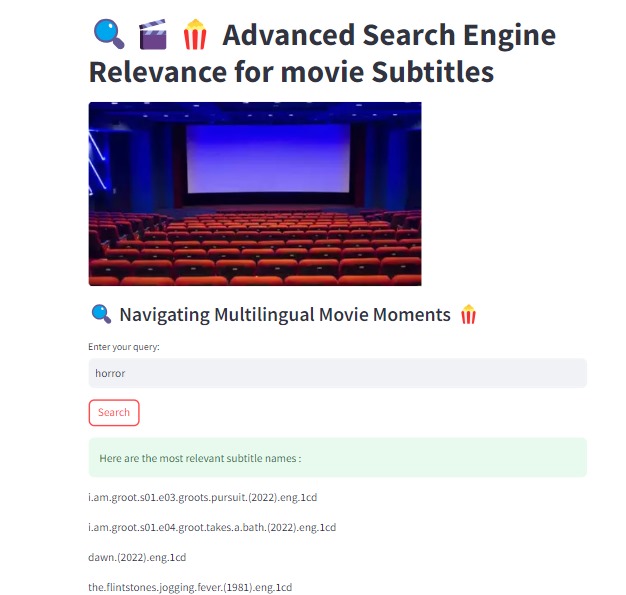
Users are prompted to input their search queries related to movie subtitles. Upon submitting the query, the application executes the **similar\_title** function to find similar subtitles based on the user-provided query. This function retrieves the most relevant subtitle names from the dataset, considering the query's relevance. The results are then displayed to the user within a success message, presenting the identified subtitle names in a straightforward manner

this Streamlit application offers an intuitive and user-friendly platform for exploring and discovering movie subtitles, leveraging advanced search capabilities provided by ChromaDB. Users can seamlessly navigate through multilingual movie moments, making their movie-watching experience more enjoyable and informative.

**Code:**

****

Localhost output looks like:



=============================thanku==============================