**DAMG 7245 - Assignment 5 - Team 5**

| **Summary** | The assignment involves experimenting with Models as a Service APIs, specifically OpenAI's GPT and Pinecone, to build intelligent applications for knowledge retrieval and question-answering tasks. The goal is to create knowledge summaries, generate a Q/A knowledge base, use a vector database for finding and answering questions, and utilize knowledge summaries to answer questions. |
| --- | --- |
| **URL** | <https://codelabs-preview.appspot.com/?file_id=11RAbeC36bhyxd02Jn-SgXHzEH83_TUOGNOXyD155i3I#1> |
| **Category** | OpenAI |
| **Environment** | Python |
| **Status** | Completed |
| **Github** | <https://github.com/BigDataIA-Spring2024-Sec1-Team5/Assignment5> |
| **Authors** | Aditya Kanala, Shikhar Patel, Shubh Patel |

**Table of Contents**

[Architecture and Workflow](#_1kfrldy4e1mg)

Overview

Workflow

[Part - 0: Scraping](#_igkpmf35l2af)

[Part - 1: Knowledge Summaries](#_3tyslooiz2c4)

[PART 2 - Knowledge Base](#_lbvbvpnygws7)

[PART - 3: RAGS](#_aewacfknq23m)

[PART - 4 : Metrics](#_chiy56izyext)

[References](#_xbxvb98sy4e0)

# **Architecture and Workflow**

This workflow visualizes the interaction between various components in a data processing and retrieval system. It incorporates services for web scraping, data storage, query processing, and user interaction. The system leverages a combination of custom components, AWS S3 for storage, and external APIs for data processing and retrieval

1. **Streamlit**: A custom application for displaying data and interacting with the user. It serves as the primary interface for data presentation and query input.
2. **AWS S3 (bucket)**: Utilized for storing scraped data and the data generated from user prompts. Acts as a central repository for both input and output data.
3. **Chrome**: Represents a web scraping tool or service that collects data from the web and stores it in the AWS S3 bucket.
4. **OpenAI**: An external API used for processing user queries and generating responses based on the data retrieved from Pinecone.
5. **Pinecone**: A vector database used for storing and retrieving data based on nearest vector search, facilitating efficient query processing by OpenAI.
6. **User**: Denotes the end-user who interacts with the Streamlit application, providing prompts for data retrieval and viewing the presented data.

**Workflow**

1. **Data Collection**: The Chrome component scrapes data from the web and stores it in the AWS S3 bucket.

2. **Data Storage and Retrieval**:

The scraped data in the AWS S3 bucket is utilized by Pinecone for indexing and retrieval based on vector search.

Streamlit displays data required from the bucket and stores data generated from user prompts back into the bucket.

3. **Query Processing**:

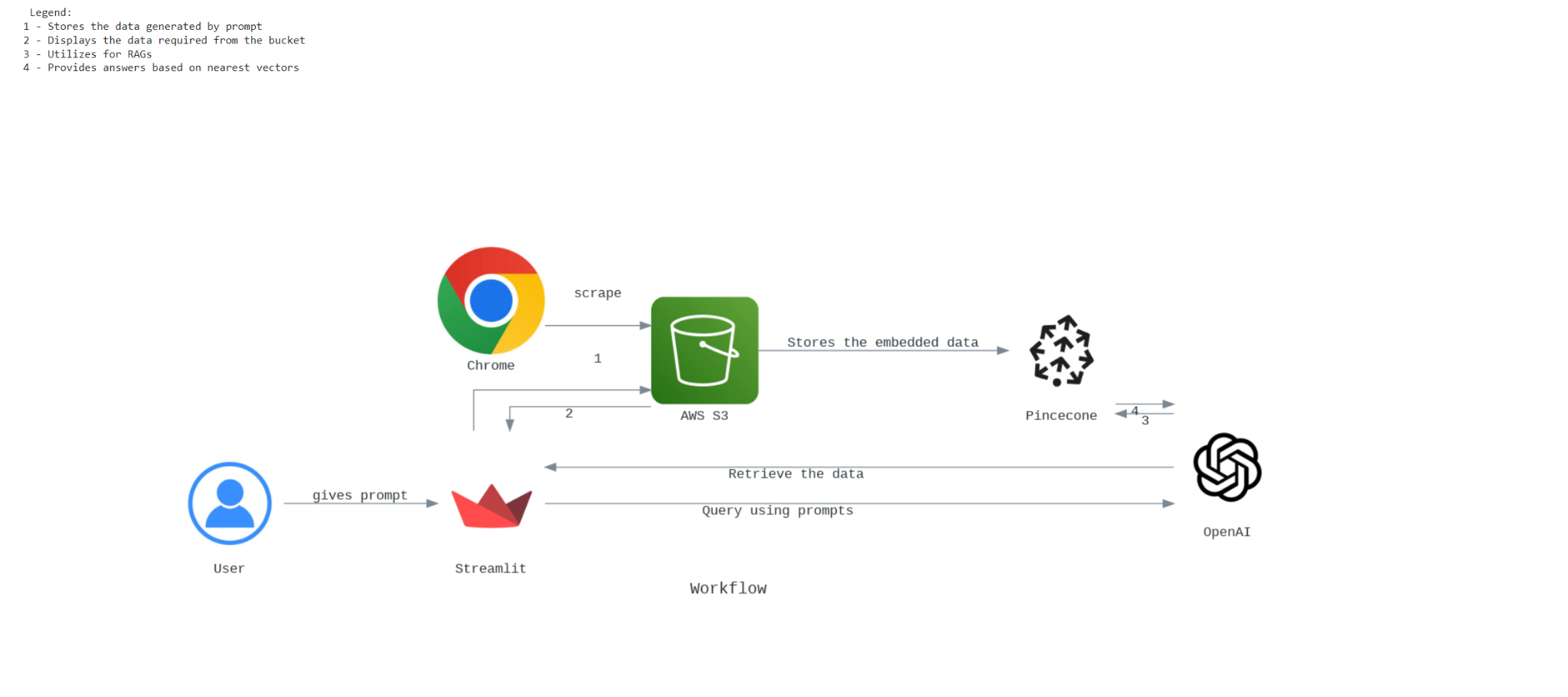
The user provides prompts to the Streamlit application.

Streamlit forwards these prompts to OpenAI for processing.

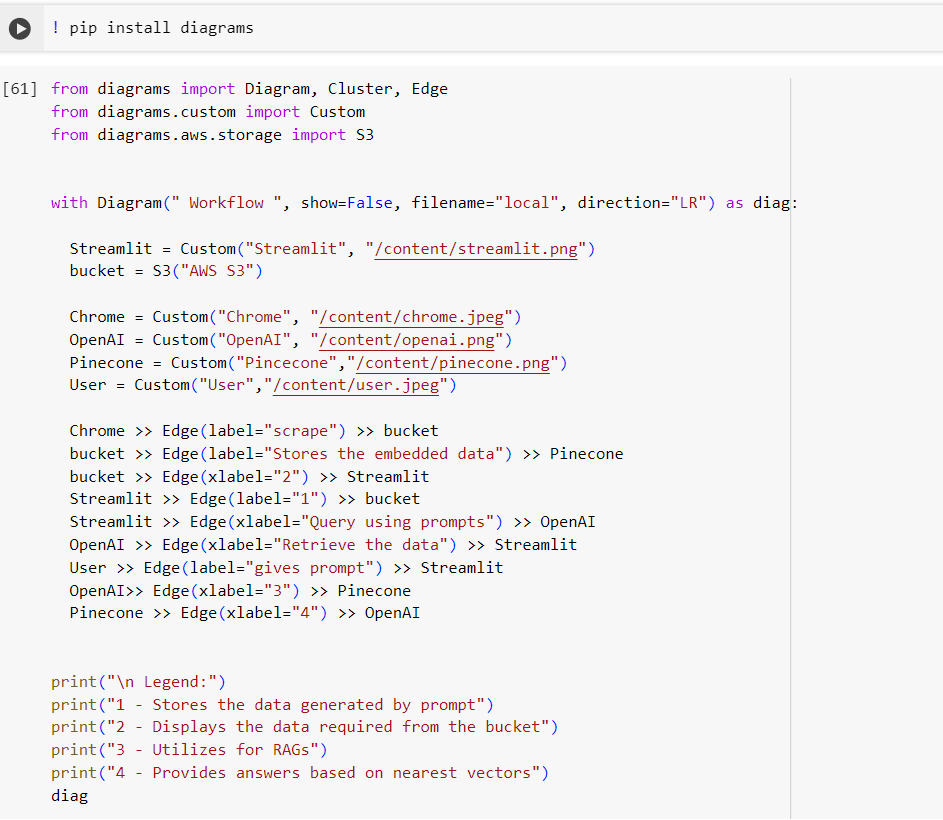
OpenAI utilizes Pinecone for Retrieval-Augmented Generation (RAG), leveraging the indexed data for generating responses.

Pinecone provides answers based on nearest vectors, which are then sent back to OpenAI.

OpenAI retrieves the data and sends the processed information back to Streamlit for display to the user.



Architecture Diagram Code:



# **PART - 0 : Scrape the CFA Website**

# 

**Overview**:

The provided Python script, automates the process of scraping educational content from specific URLs of the CFA Institute’s website. It extracts information such as article title, topic, publication year, level, introduction, learning outcomes (LOS), summary, and links to full PDFs. This data is then saved into a CSV file and optionally uploaded to an AWS S3 bucket.

**Dependencies:**

* **Selenium**: For automating web browser interaction to scrape content.
* **BeautifulSoup (bs4)**: For parsing HTML and extracting data.
* **webdriver-manager**: For managing the browser driver (ChromeDriver in this case) automatically.
* **boto3**: For interacting with AWS services, specifically S3 for file storage.
* **python-dotenv**: For loading environment variables from a .env file, which can store sensitive information like AWS credentials.
* **csv**: For writing the scraped data into a CSV file format.
* **re (Regular Expressions)**: For extracting specific patterns from text, such as the publication year.

**Setup:**

1. **Python Environment**: Ensure Python is installed on your system.

2. **Install Dependencies**: Install the required Python packages using pip for the requirements.txt which has all the below mentioned packages:

Selenium,webdriver\_manager,beautifulsoup4,pandas,boto3,botocore,six, python-dotenv

3. **Selenium WebDriver**: The script uses ChromeDriver managed by webdriver-manager, so ensure Google Chrome is installed.

4. **AWS Credentials**: If uploading to S3, configure your AWS credentials. This can be done by setting up an AWS credentials file or using environment variables. The .env file should contain your AWS access key and secret key.

5. **Environment Variables**: Use a .env file to store sensitive information such as AWS credentials securely.

**Usage:**

1. **Configuration**: Modify the script to include the correct URLs in the urls list for the documents you need to scrape. Also, update the bucket\_name in the save\_to\_csv\_and\_upload\_to\_s3 function to match your AWS S3 bucket name.

2. **Running the Script**: Execute the script from the command line:

python scraping.py

This will start the scraping process for each URL listed, extract the required information, save it into a CSV file named Team05.csv, and optionally upload this file to the specified S3 bucket.

3. **Output**: The script outputs a CSV file with columns for Article, Topic, Year, Level, Introduction, Learning Outcomes, Summary, Link to the Summary Page, and Link to the PDF File. If configured, the file is also uploaded to an AWS S3 bucket.

4. **Error Handling**: The script includes basic error handling for missing credentials and elements not found during the scraping process. Ensure proper error handling is in place for a smooth operation, especially when dealing with dynamic web content.

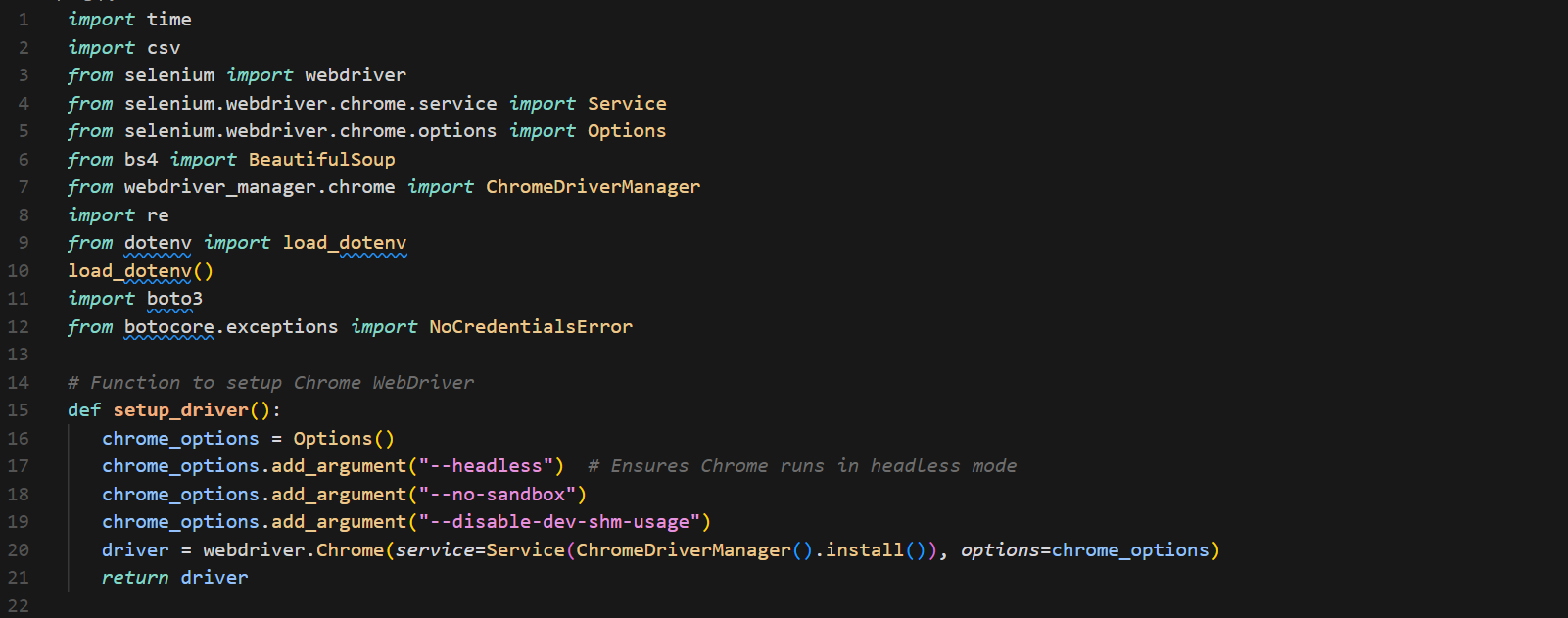
**Steps:**

Sure, here's a breakdown of the provided code in points:

**1. Imports and Environment Setup:**

- Imports necessary libraries and modules such as Selenium, re, os, boto3

- Loads environment variables from a .env file using `dotenv`.

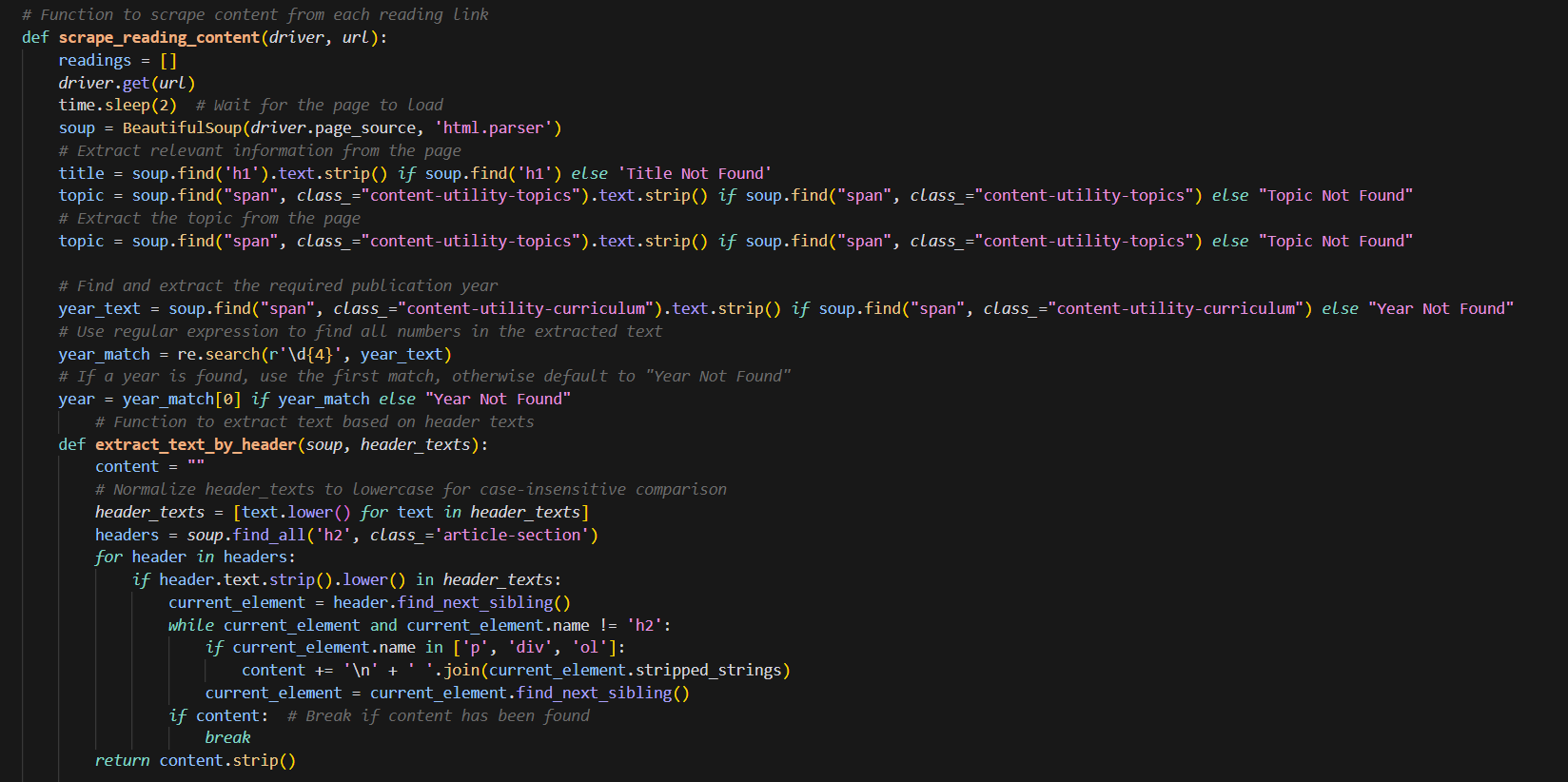


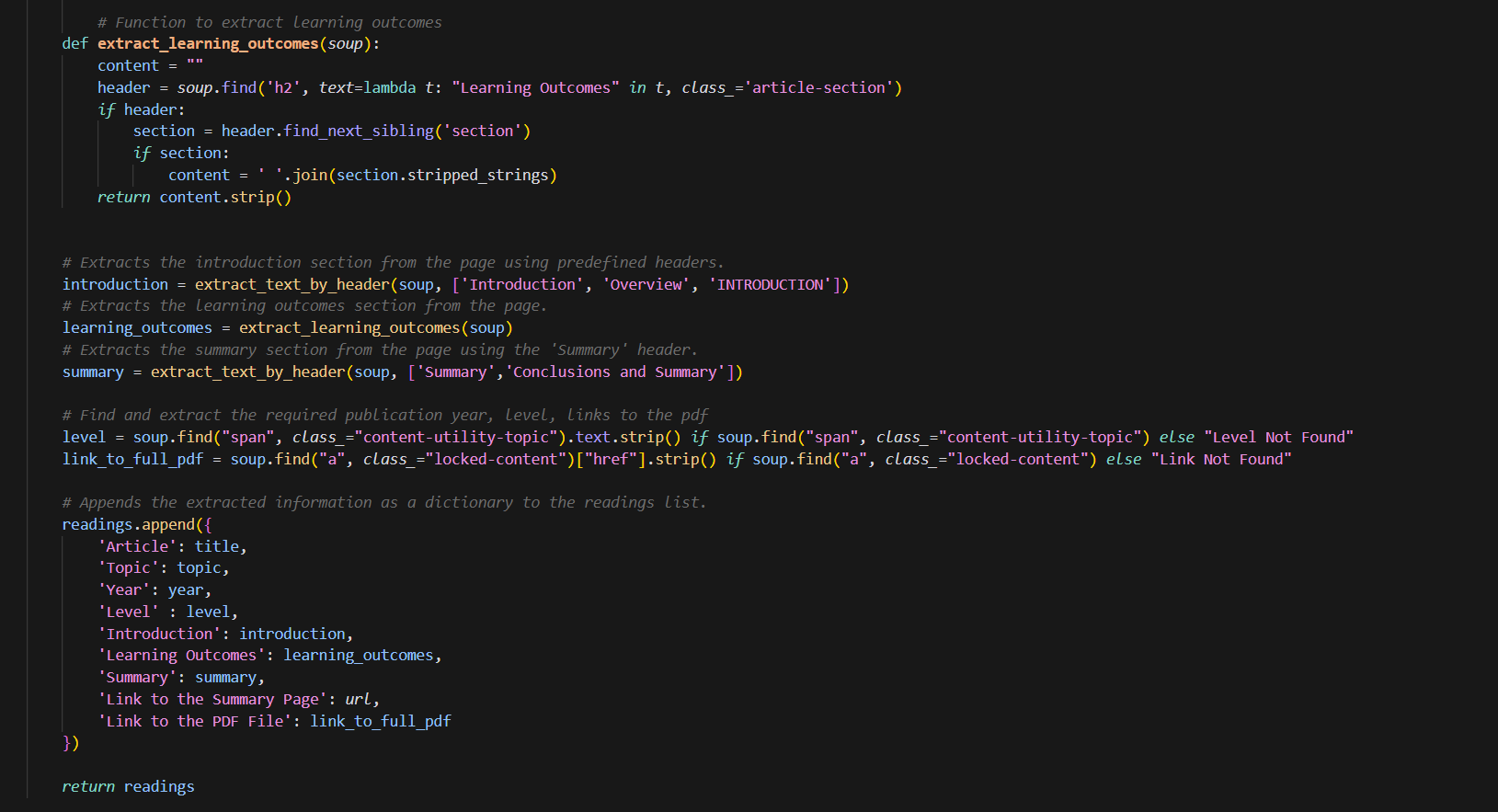
**2. Function to scrape content from the required links:**

- Defines a function `send\_query\_to\_fastapi` to send a query to a Fast API endpoint with a file key parameter.

- Uses the requests library to make an HTTP GET request to the specified endpoint.

- Handles errors and displays them using Streamlit's `st.error`.

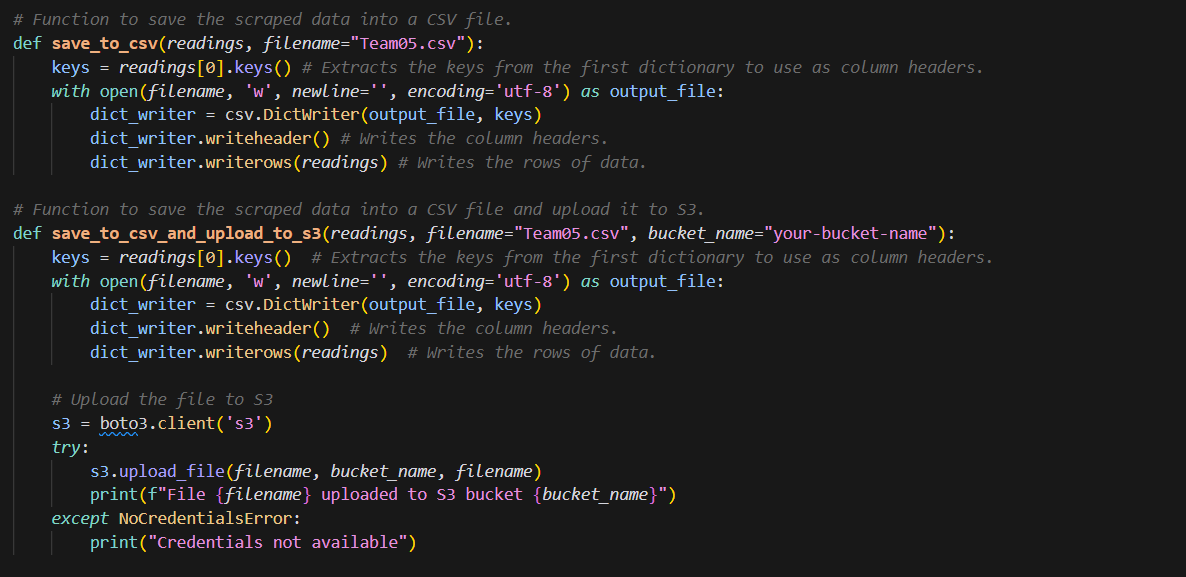




**3. Function to Save and Upload File to S3:**

- Defines a function `save\_to\_csv\_and\_upload\_to\_s3` to upload a file to an S3 bucket.

- Uses the Boto3 library to interact with AWS S3, using credentials loaded from environment variables.



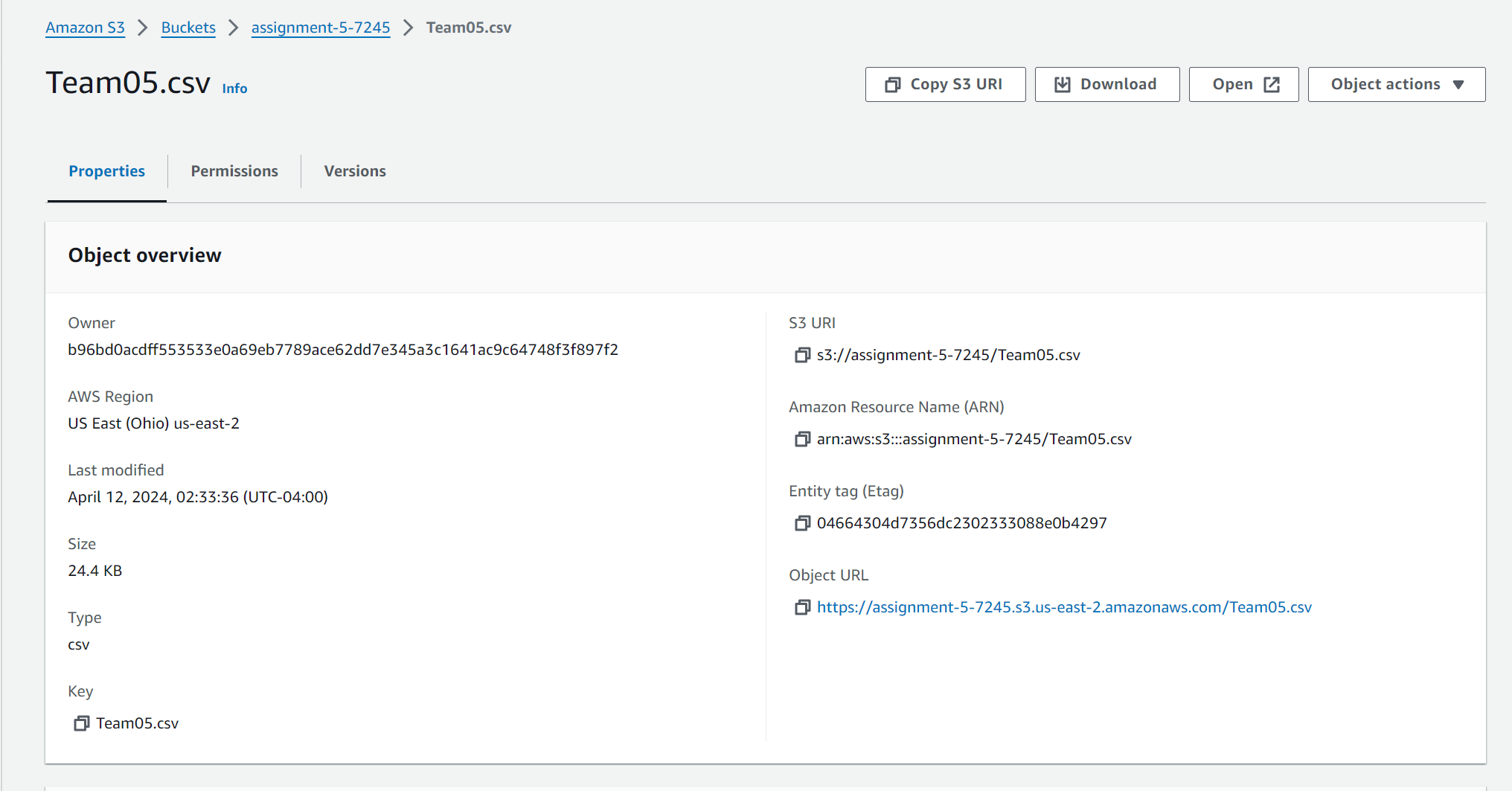
**4. Main Scraping Application Function:**

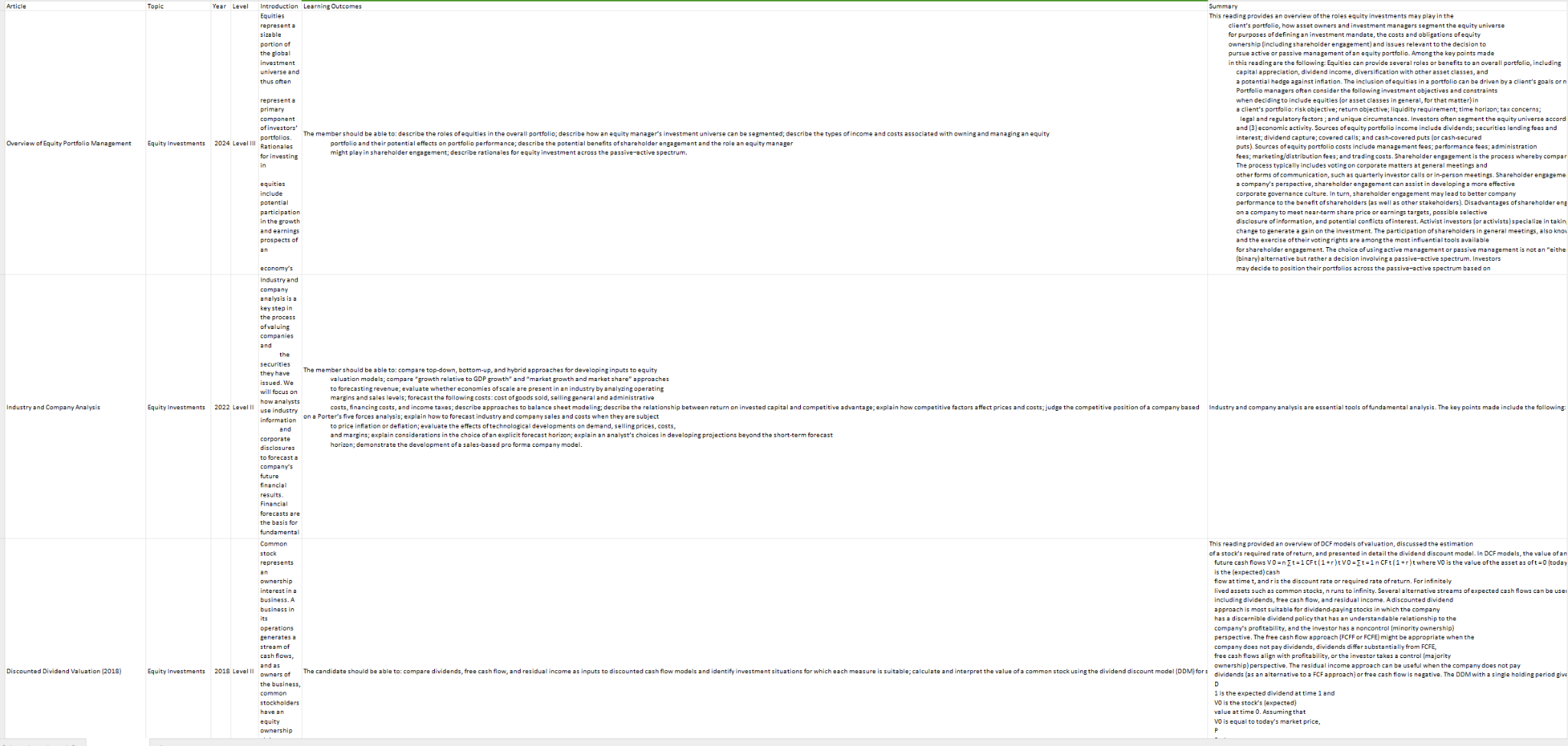
- Defines the main function `main` of the Scraping script.

- Contains the urls assigned to each member of the team.



**Output:**





The output of the whole scrapping is stored in csv as requested in the assignment. The above image is just an example of how the output is being stored. The columns that have been generated are Article, Name of the topic, Year, Level, Introduction, Learning Outcomes, Summary, Link to the Summary Page and Link to the PDF file.

# **PART - 1 : Creating Knowledge Summaries using OpenAI’s GPT**

# 

**Overview**:

The project aims to create a knowledge base using OpenAI's GPT for generating knowledge summaries based on Learning Outcome Statements (LOS) from financial documents. It involves creating technical notes for each LOS, consolidating these notes into a markdown document, chunking the content, and storing it in Pinecone, a vector database, for easy retrieval and question answering.

**Dependencies:**

**OpenAI GPT**: For generating knowledge summaries and technical notes.

**Pinecone**: For storing and retrieving vectorized chunks of the generated notes.

**Python Libraries**:

* boto3: For AWS S3 interactions, such as reading and uploading files.
* csv: For reading CSV files containing LOS and other relevant information.
* dotenv: For loading environment variables (e.g., API keys).
* io: For handling in-memory file operations.
* openai: For accessing OpenAI's API.
* langchain: Specifically, RecursiveCharacterTextSplitter for chunking large texts.

**Setup:**

1. **Environment Variables**: Set up environment variables for OpenAI and Pinecone API keys, and AWS credentials by using a .env file or directly in your environment.

2. **Pinecone Setup**: Initialize a Pinecone environment by creating an index to store the vectorized chunks.

3. **AWS S3 Setup**: Ensure you have an S3 bucket set up for storing the CSV files and the generated markdown documents.

**Usage:**

1. **Data Preparation**:

Store the LOS and related information in a CSV file in an S3 bucket.

Use boto3 to read this CSV file from S3.

2. **Generating Knowledge Summaries**:

Implement a function create\_technical\_note\_md that takes LOS, summaries, and optionally tables, figures, and equations to generate a markdown-formatted technical note.

Use OpenAI's GPT to generate content where necessary.

3. **Consolidating and Chunking**:

Consolidate all generated markdown notes into a single document.

Use RecursiveCharacterTextSplitter from langchain to chunk the consolidated document into smaller pieces suitable for vectorization and storage in Pinecone.

4. **Storing in Pinecone**:

Vectorize the chunks using OpenAI's embedding API.

Store the vectorized chunks in Pinecone for later retrieval.

5. **Uploading Documents**:

Upload the consolidated markdown document back to S3 for storage or sharing.

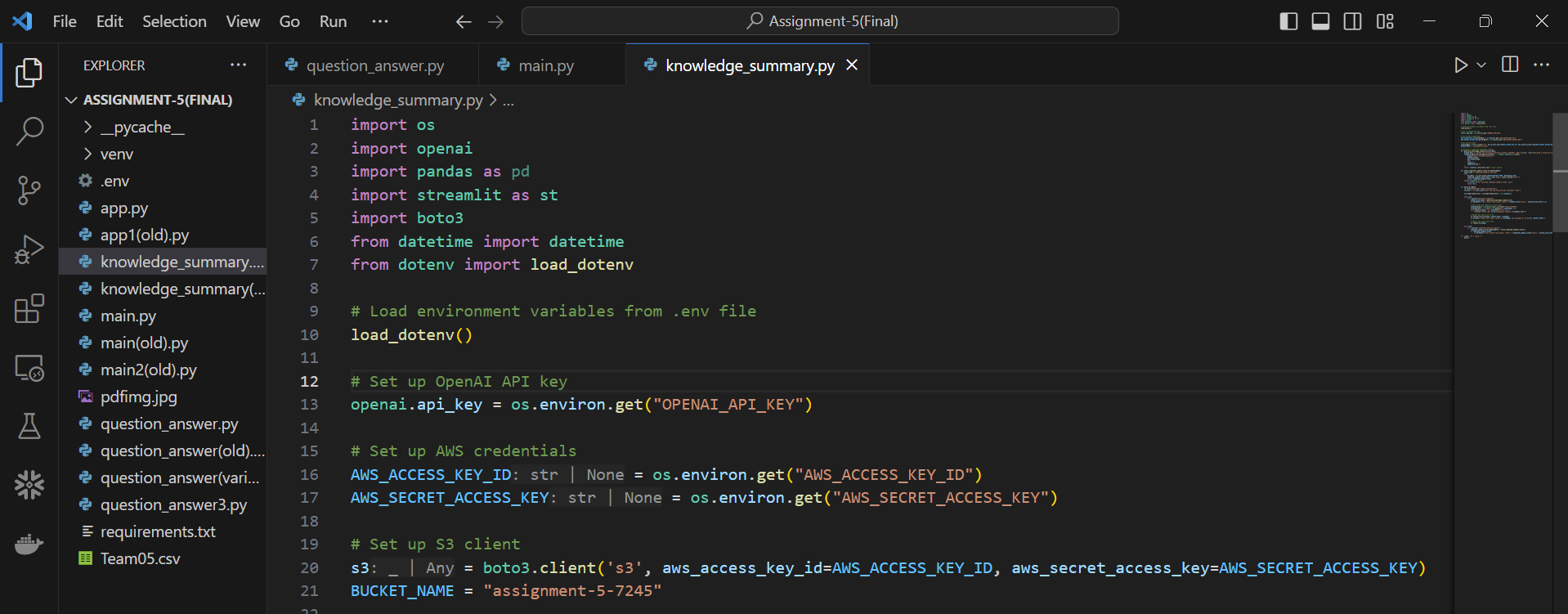
Ensure the document is accessible for future reference or updates.

**Steps:**

Sure, here's a breakdown of the provided code in points:

**1. Set up environment variables:**

Make sure you have an .env file in your project directory with the following environment variables:

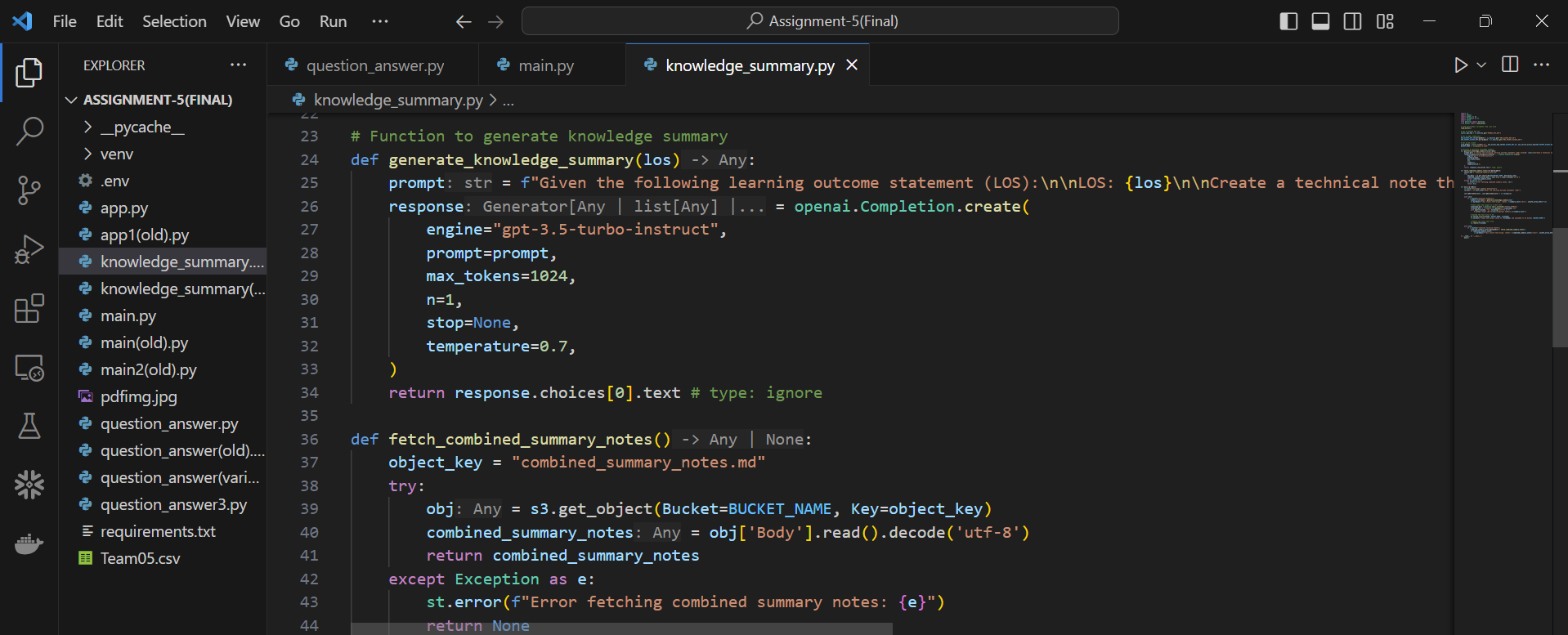


**2.Run the Streamlit app:**

Save the provided code in a file named knowledge\_summary.py.

Open a terminal and navigate to the directory containing knowledge\_summary.py.

Run the Streamlit app using the following command:



**3. Use the application:**

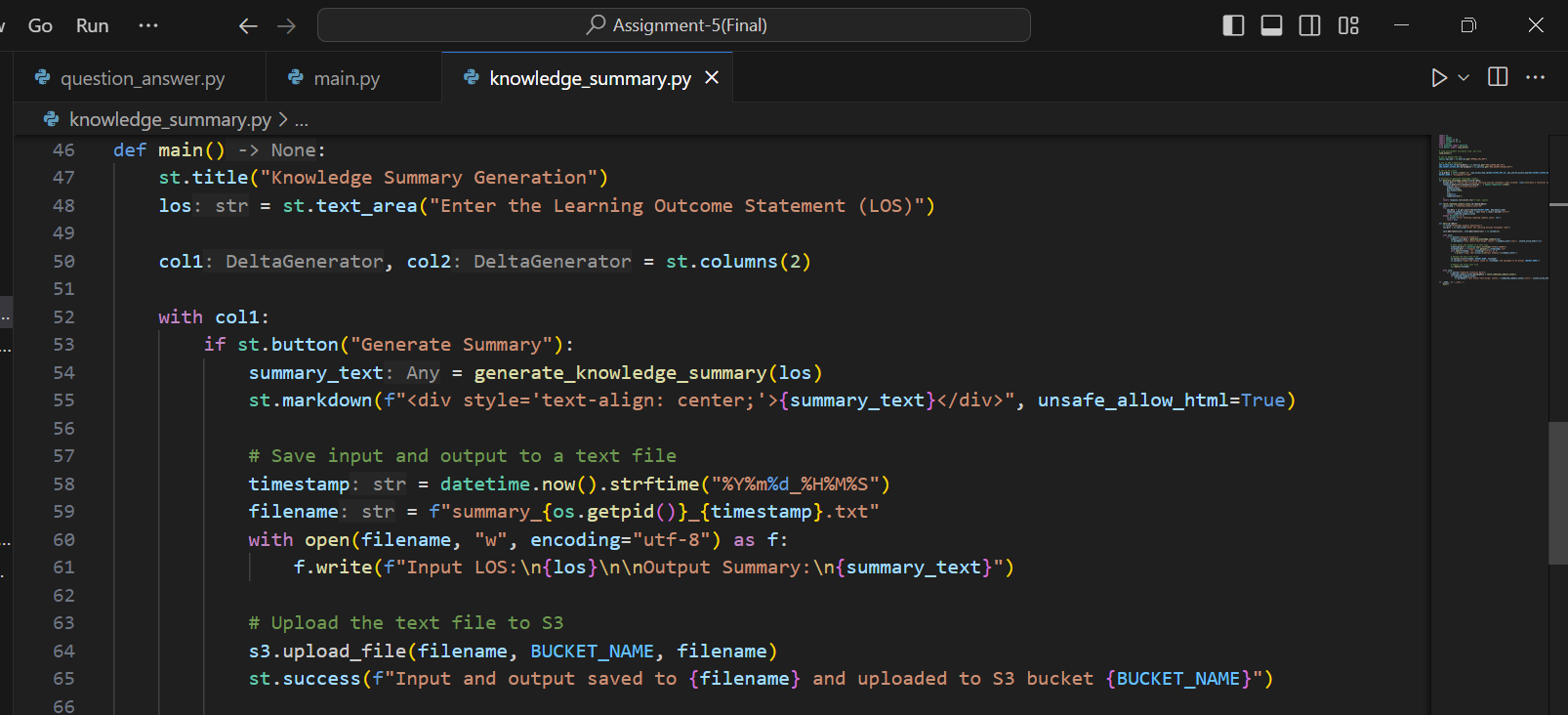
Once the Streamlit app is running, you will see an input text area labeled "Enter the Learning Outcome Statement (LOS)".

Enter the learning outcome statement (LOS) for which you want to generate a knowledge summary.

Click the "Generate Summary" button to generate the summary.

The generated summary will be displayed in the right column.

If you click the "Combined Technical Note" button, it will attempt to fetch and display a combined summary note stored in an S3 bucket (assuming it exists).

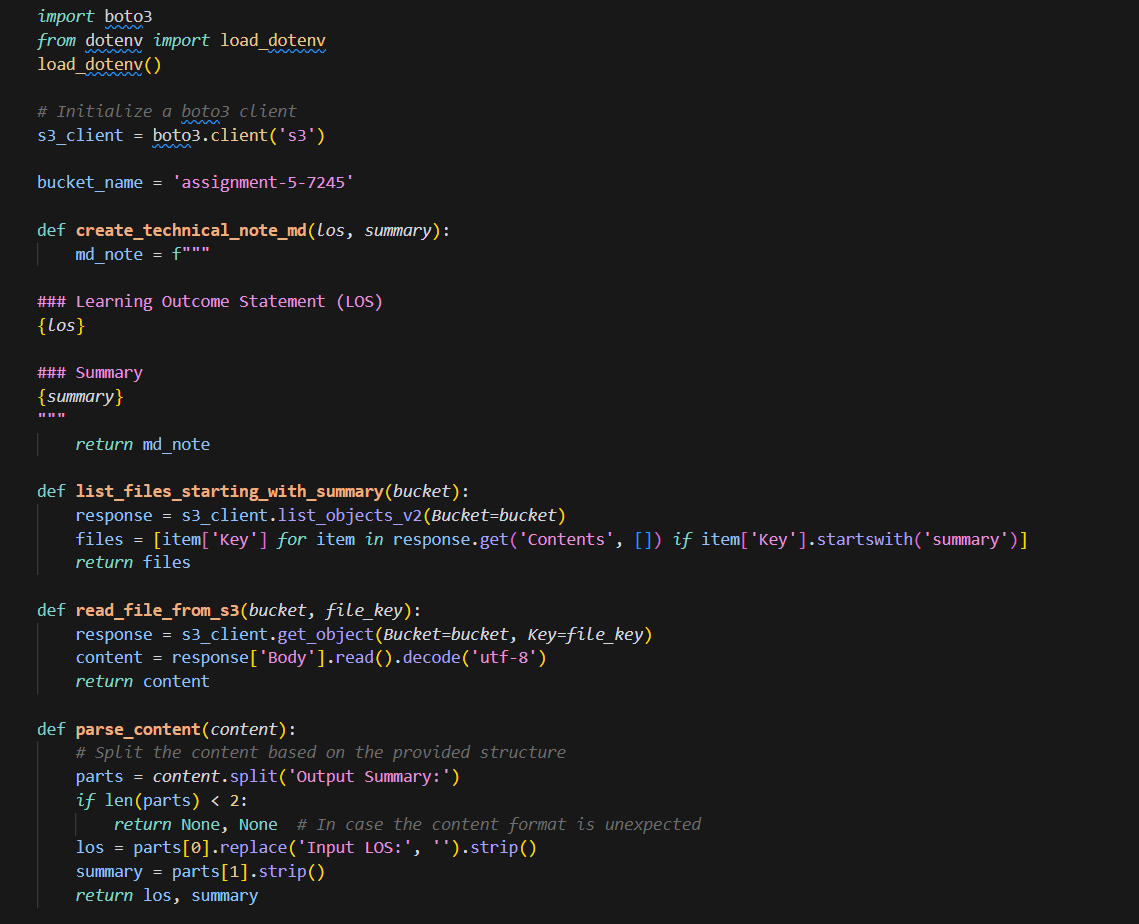


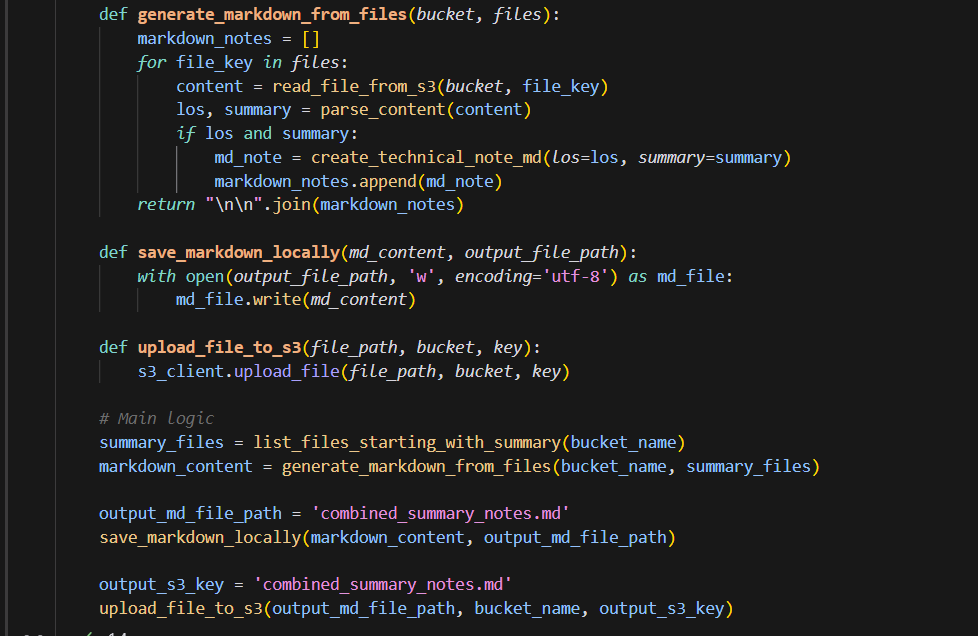
**4. Consolidating into a Markdown Document:**

- Defines functions to load the 3 summary texts from AWS S3 and combine them to one single text file and upload it back to S3..

- Uses credentials loaded from environment variables to establish a connection to S3.

- Defines another function to convert the combined summaries text file to a markdown document and upload it back to S3.

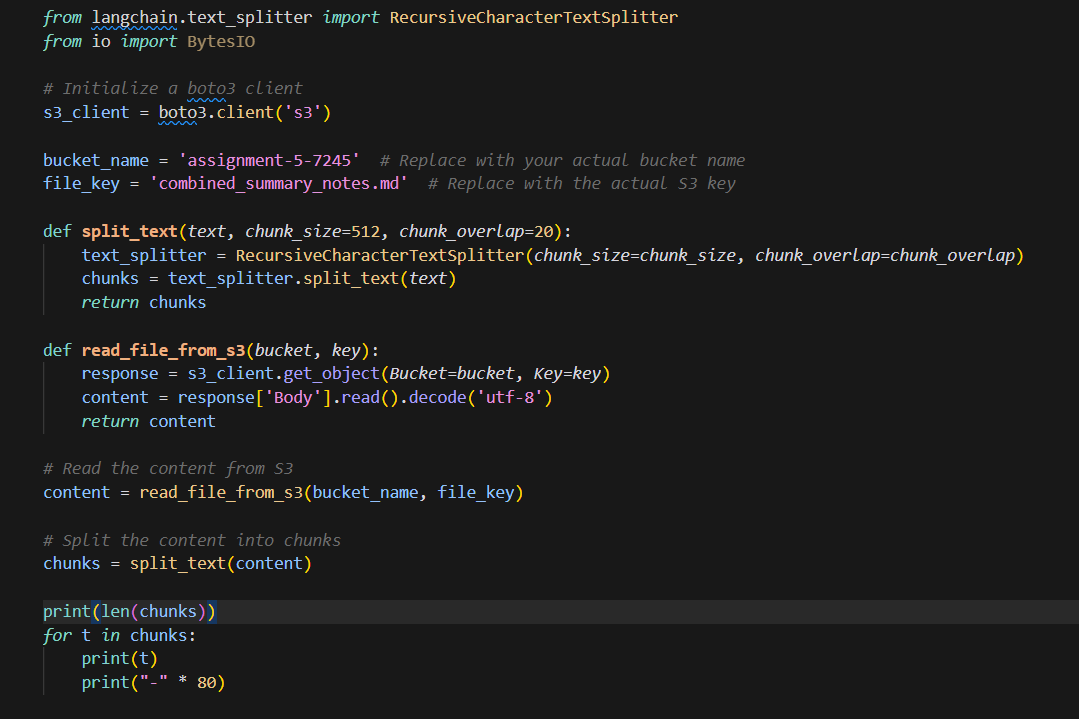




**5. Chunking the Text File:**

- Defines a couple of functions to split the text and read the file from S3.

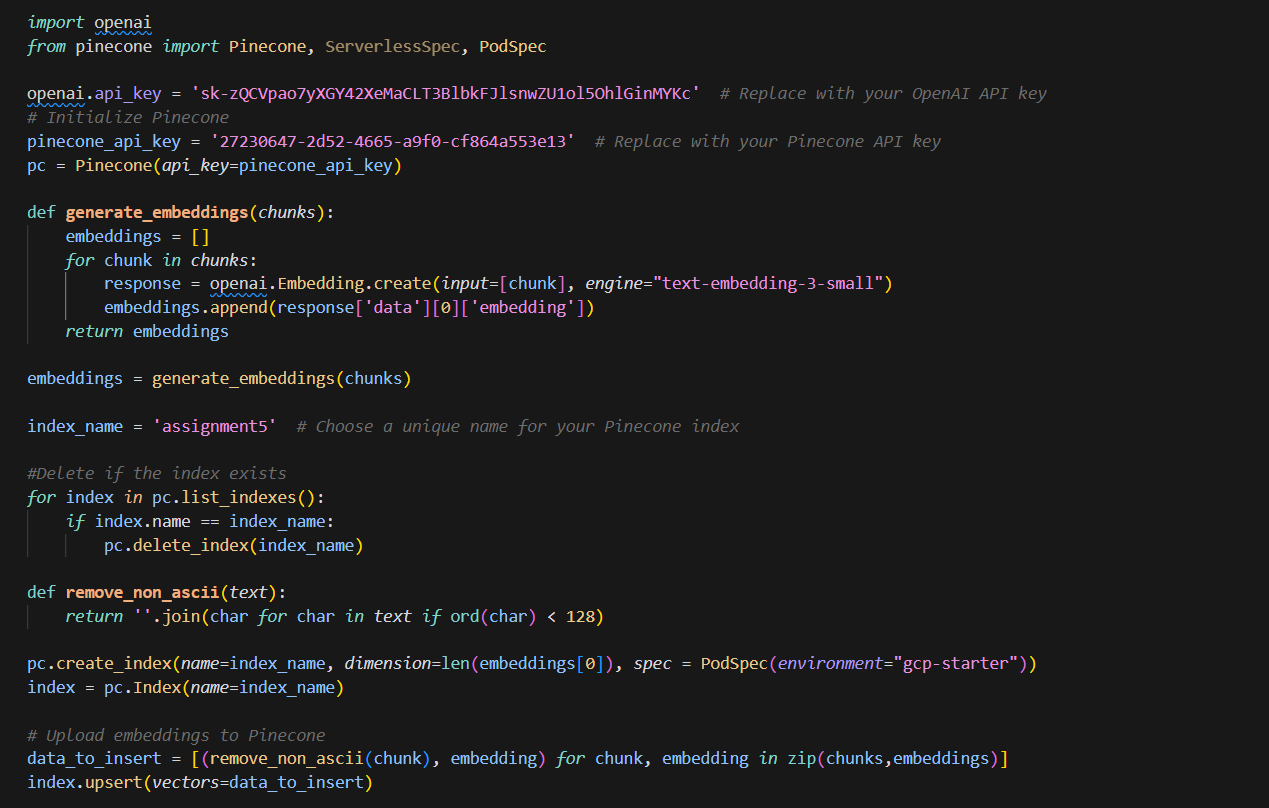
- The split function works using a library from langchain .



**6. Pushing the Embedded Code to Pinecone:**

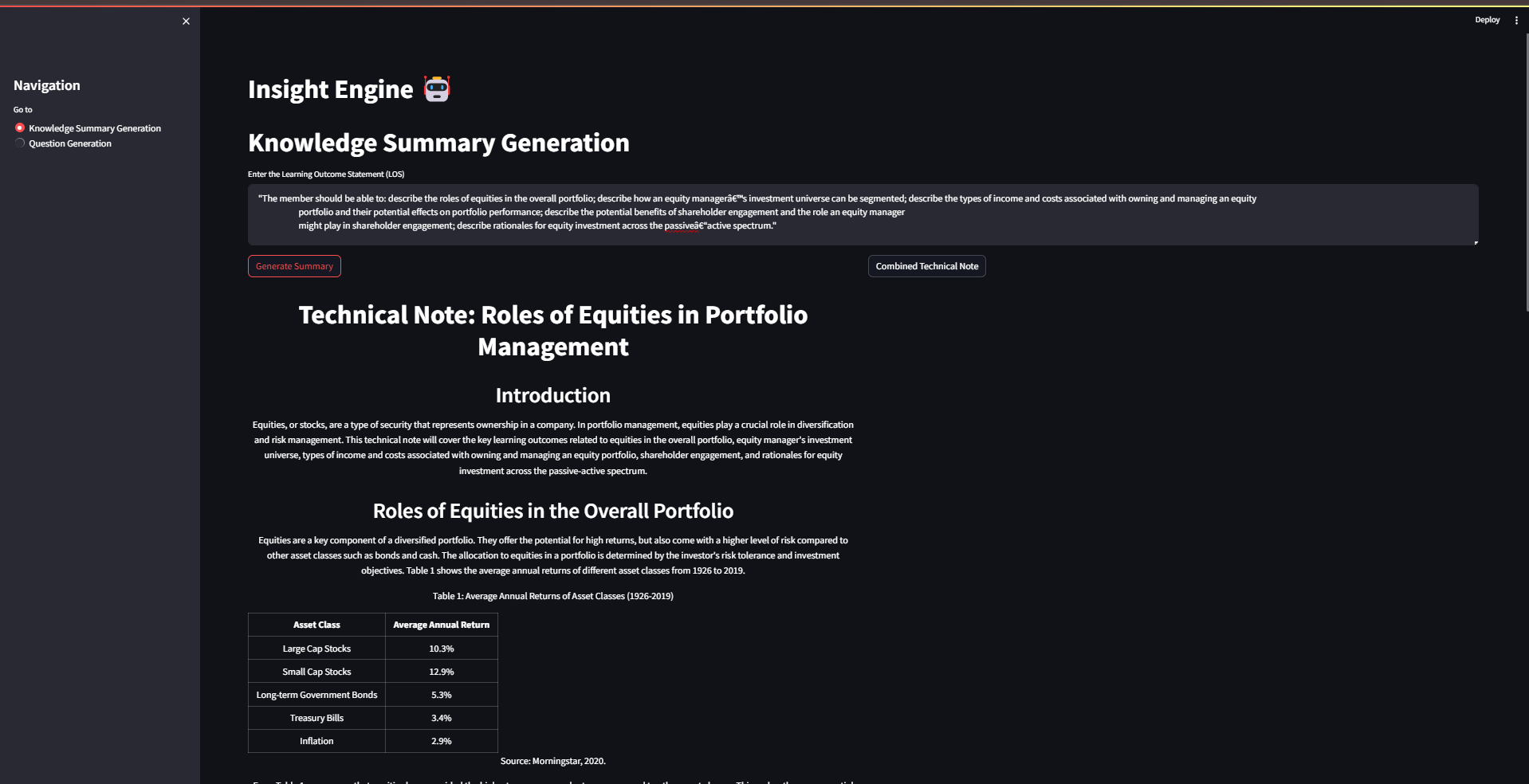
- In the generate embeddings function, the chunked data is assigned embeddings using the openai API of embedding.

- An index is created in Pinecone to store the embedded data in a vector format. A condition to delete and create an index if exists also is implemented.

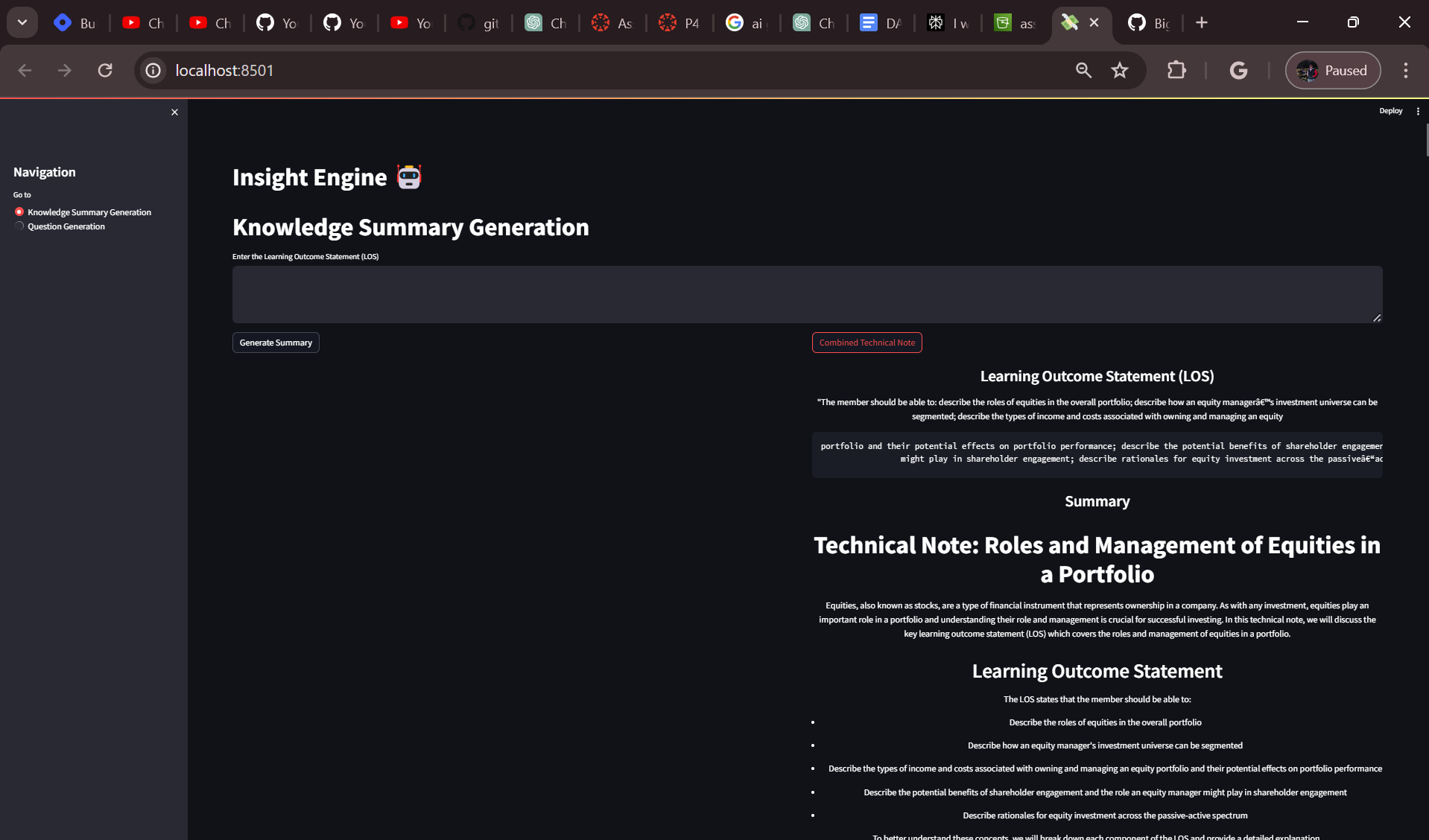


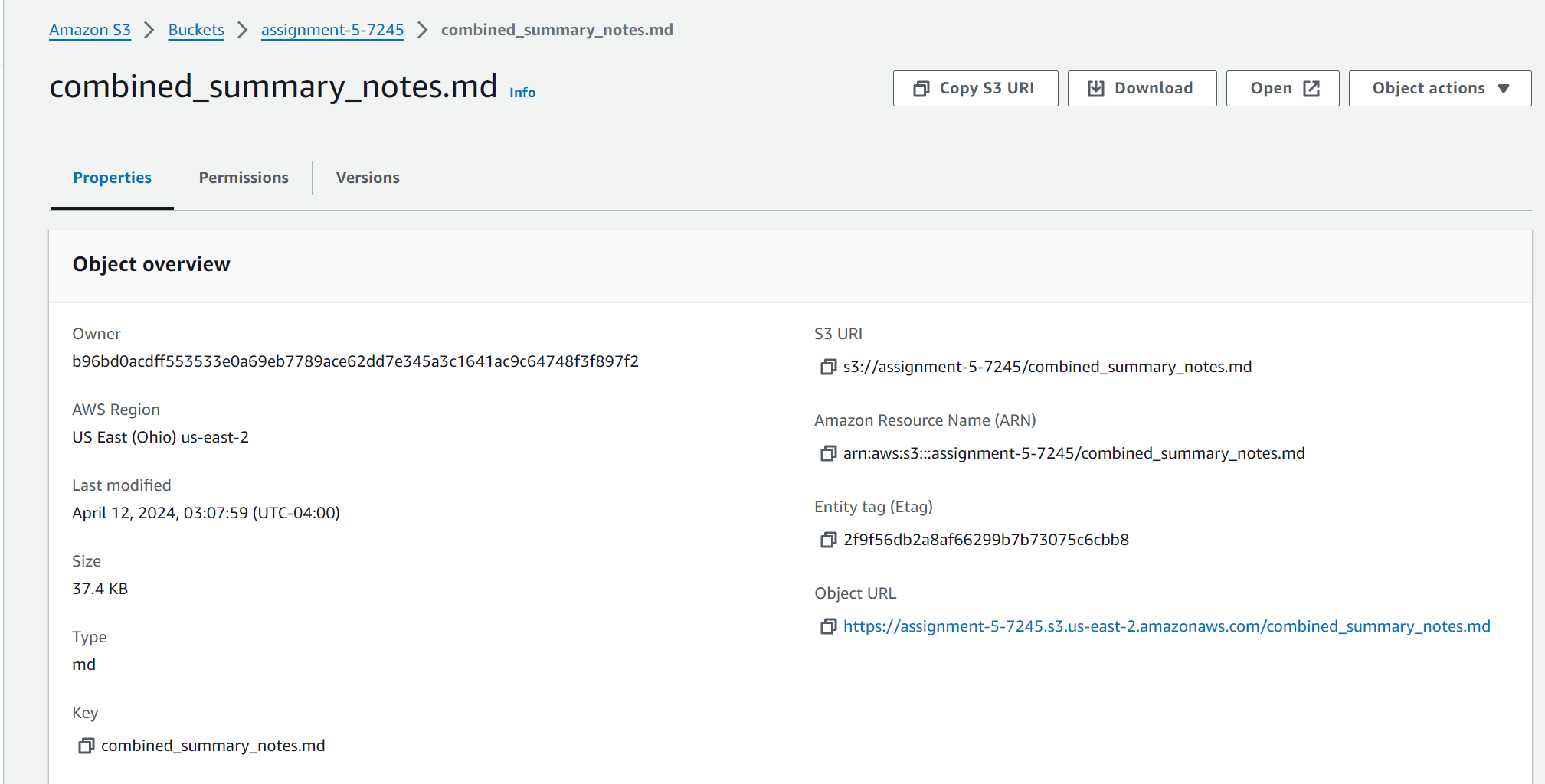
**Output:**

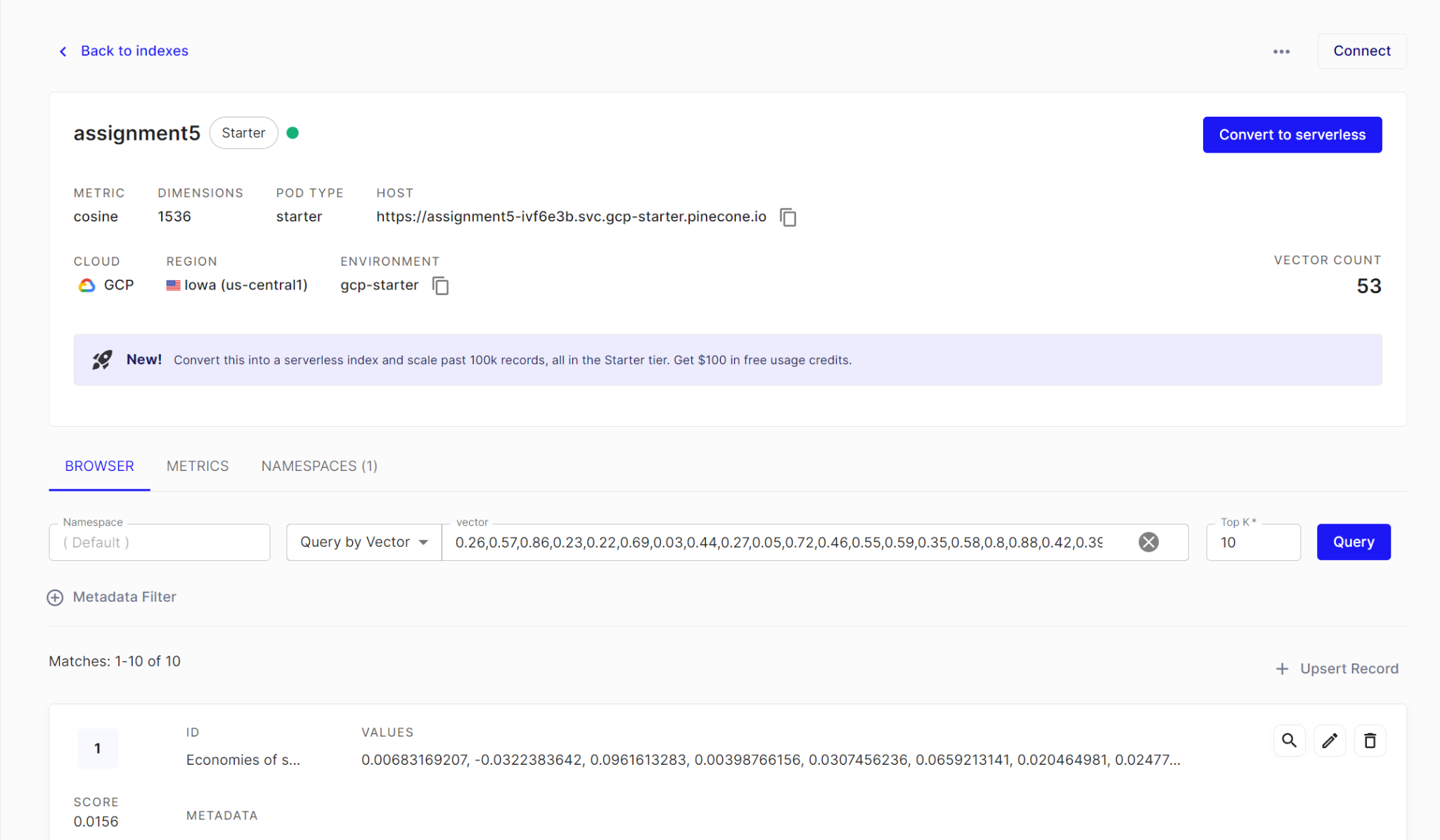
The output when we press the button Generate Summary for a specific LOS.

****

Output when we press the button Combined Technical Note.







The output of the whole embedded text is stored in pinecone as requested in the assignment. The above image is just an example of how the output is being stored. The columns that have been generated are ID which contains the chunk and the values which contain the vectors.

# **PART - 2: Generating a Knowledge base (Q/A) providing context**

**Overview**

The task involves generating a question bank of 50 questions (Set A) based on the "Summary" section of each assigned topic. These questions, along with four options including one correct answer, aim to reinforce learning for financial analysts with an MBA. The questions should match the complexity and type found in the sample CFA exam questions provided by the CFA Institute. After generating Set A, the process is repeated to create another set of 50 questions (Set B), which is kept aside for future use. Set A's questions and answers are then stored in Pinecone with separate namespaces for easy retrieval and organization.

**Installation and Setup**

To set up S3 for your assignment, follow these steps:

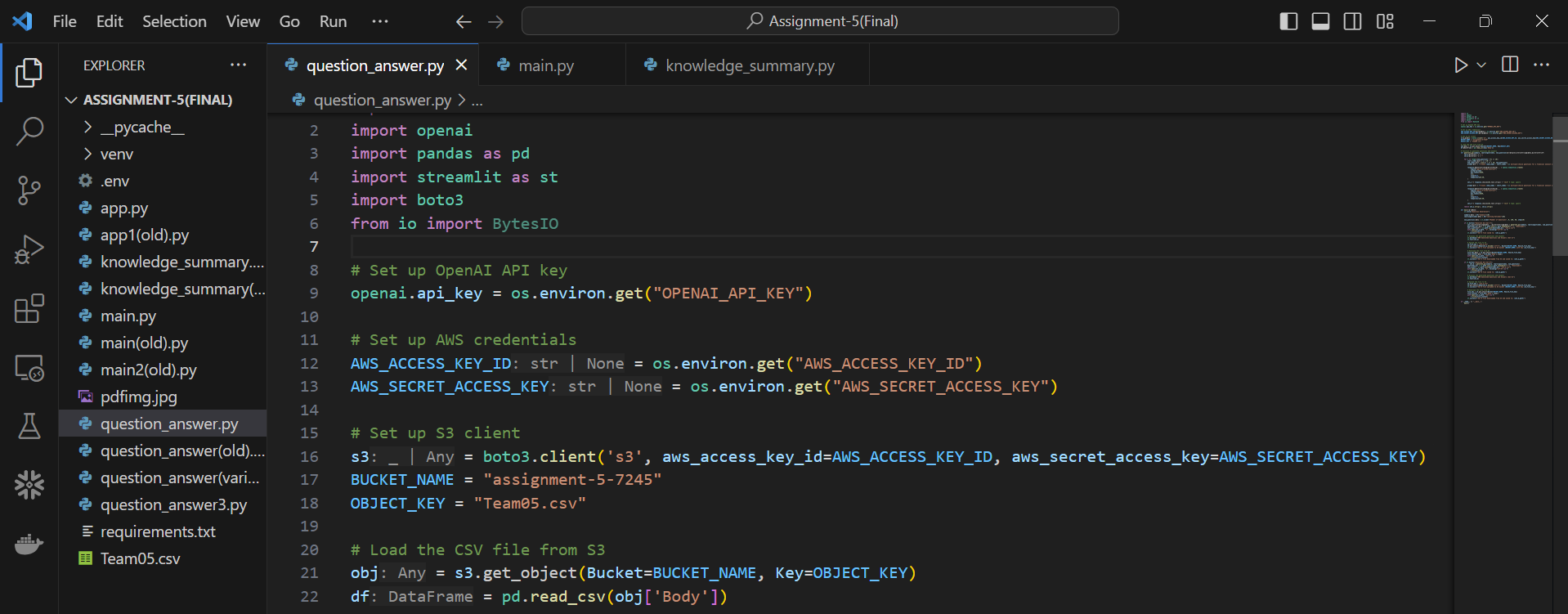
Create an S3 Bucket: Log in to the AWS Management Console, navigate to the S3 service, and create a new bucket. Choose a unique name for your bucket and configure settings such as region and access permissions.

Set Permissions: Configure bucket policies and access control lists (ACLs) to define who can access your bucket and what actions they can perform.

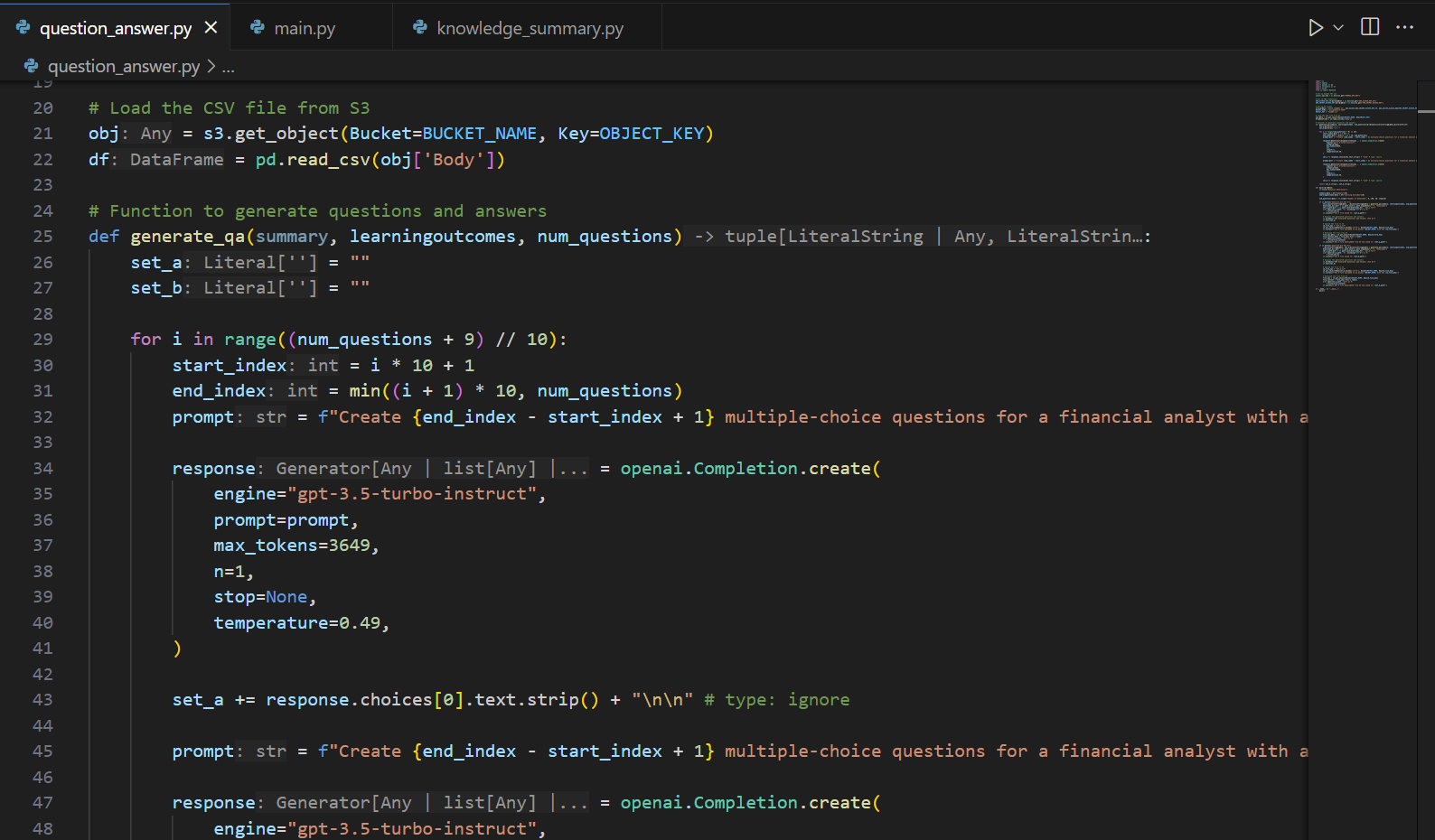
Access Keys: Obtain AWS access keys (Access Key ID and Secret Access Key) from the IAM (Identity and Access Management) console. These credentials will be used to interact with S3 programmatically.

**Steps:**

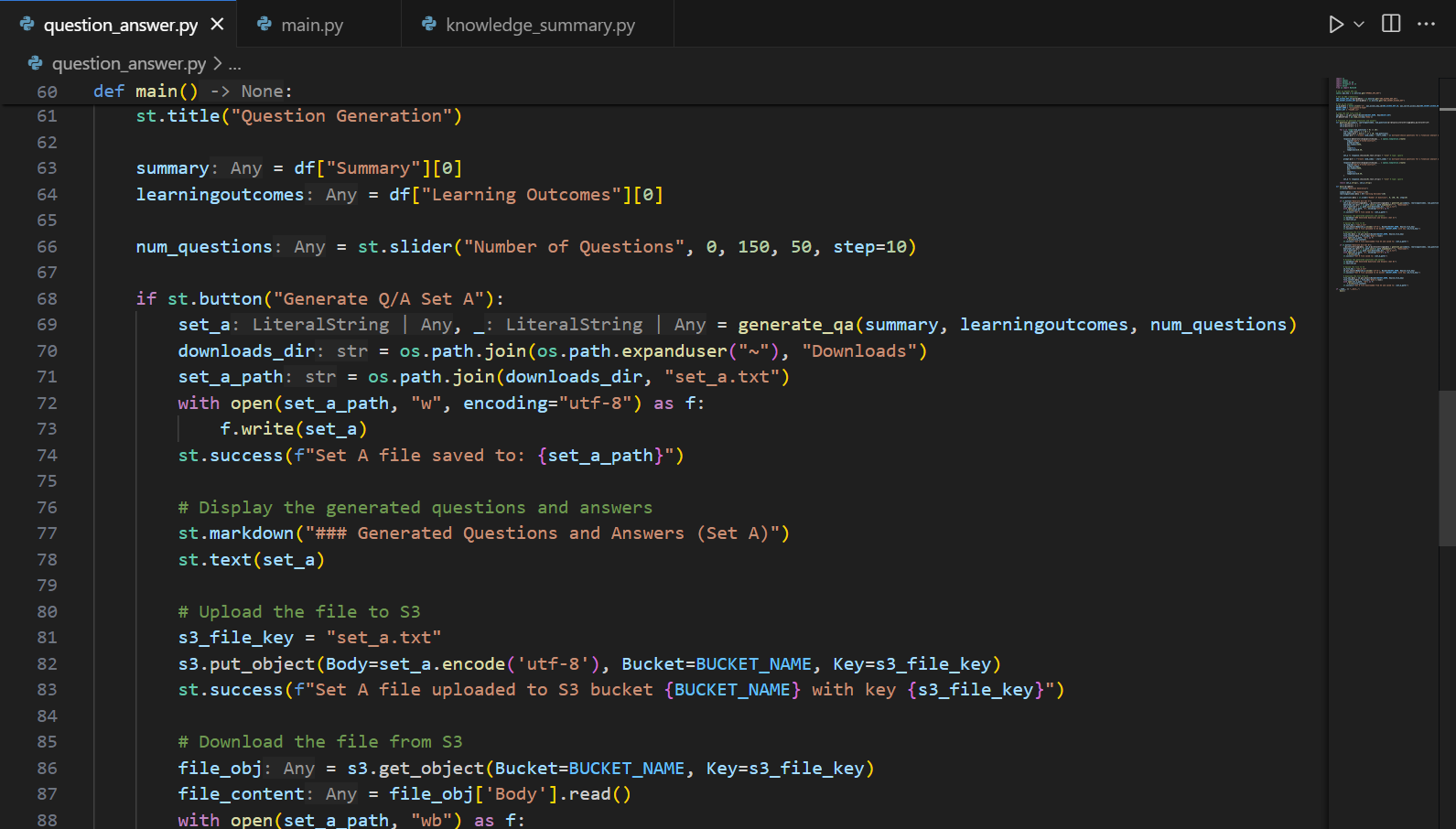
1. Import Libraries: Import necessary libraries and setting up API keys for OpenAI and AWS credentials using environment variables.



1. Load CSV File from S3: Retrieve the CSV file from the specified S3 bucket and load it into a Pandas DataFrame.



1. Get Summary and Learning Outcomes: Extract summary and learning outcomes from the loaded DataFrame and use a streamlit slider to get user input for the number of questions to generate.



**Output:**

A screenshot of a computer

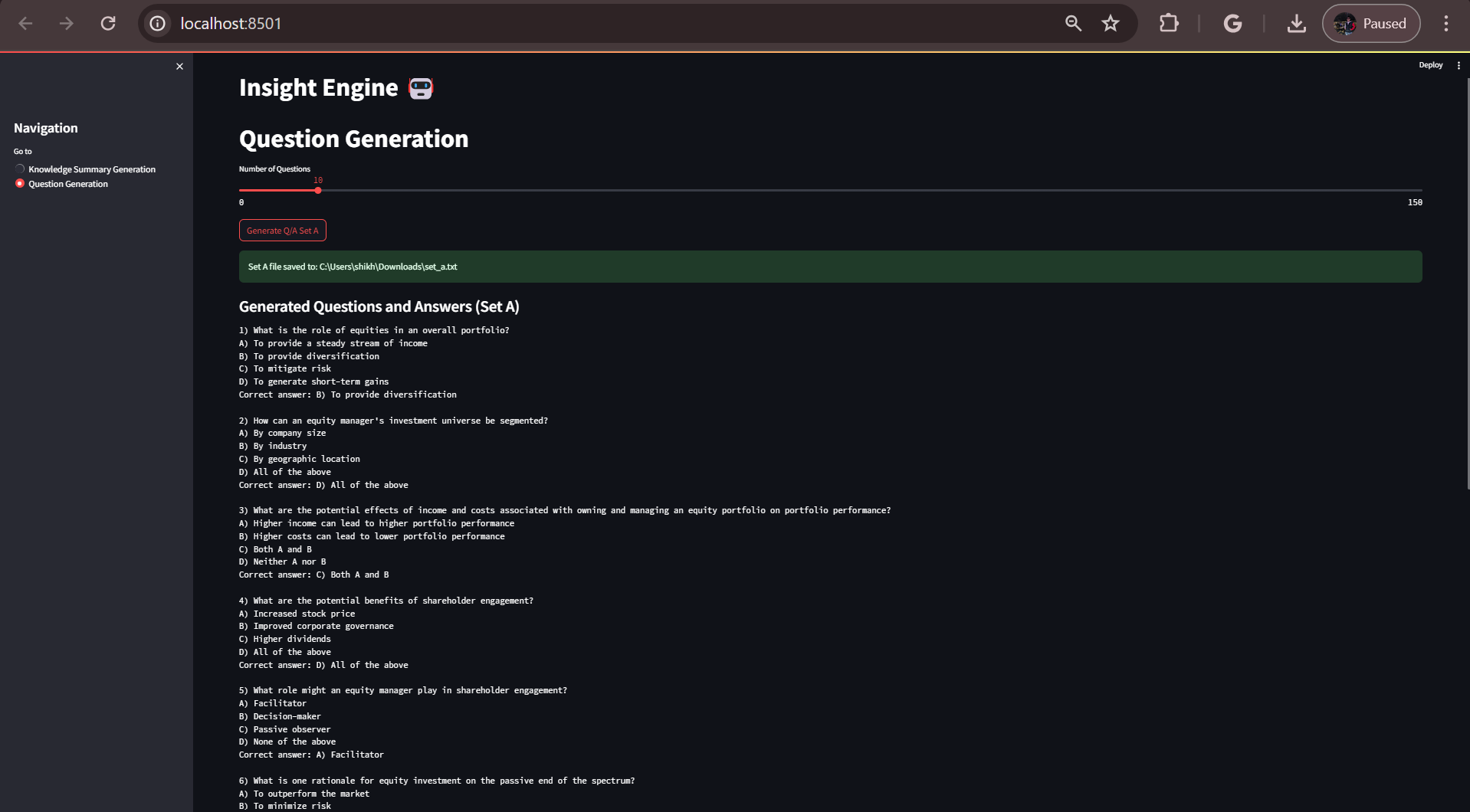
Description automatically generated

All the inputs files are stored in an S3 bucket named “assignment-4bigdata” and the outputs

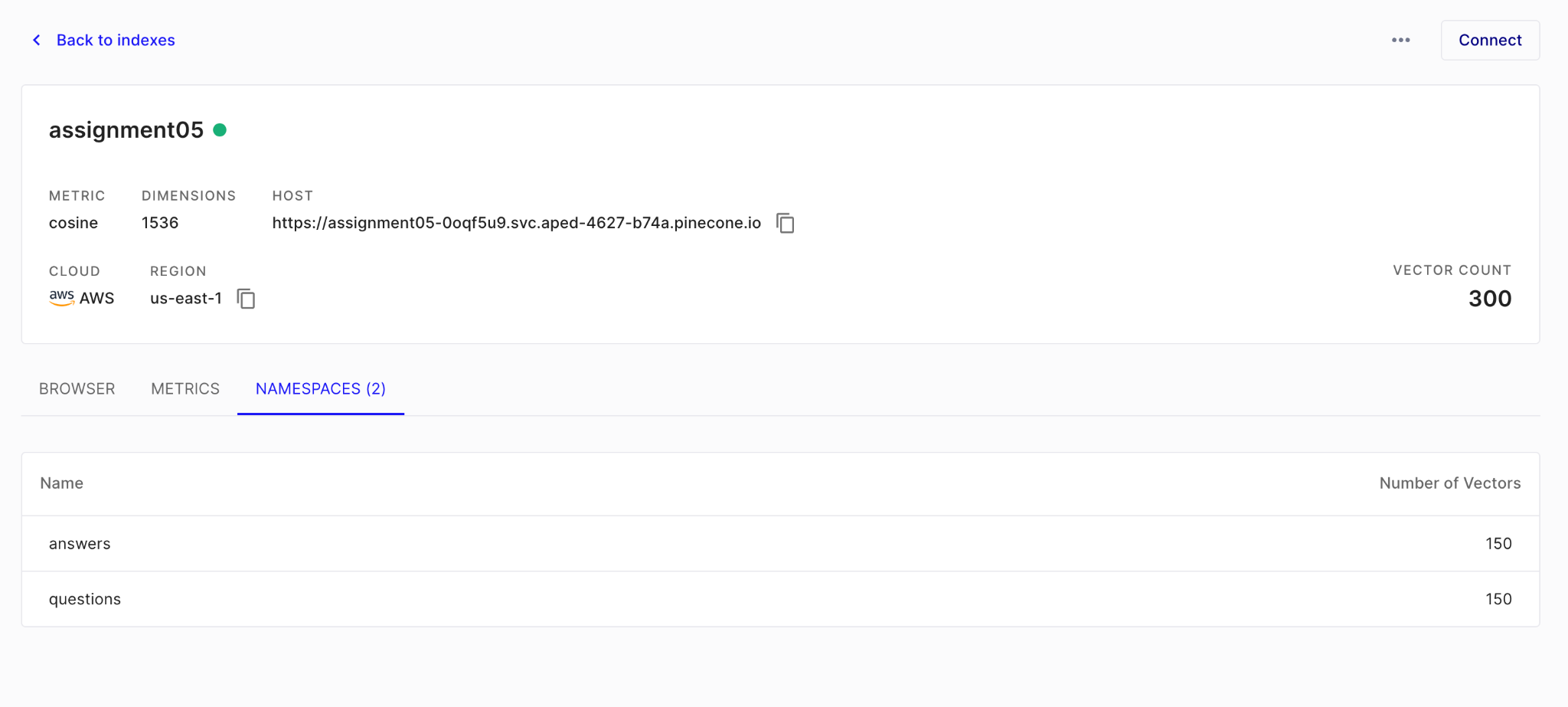
are stored in another S3 bucket names “assignment4-ext”.

# 

**Streamlit UI:**

****

**Storing set A in Pinecone with separate namespaces for question and answers**

****

# **PART - 3: Using a Vector Database to Find the Answers**

# **PART - 4: Use the Knowledge Summaries to Get the Metrics**

# **References**

<https://www.cfainstitute.org/membership/professional-development/refresher-readings/overview-equity-portfolio-management>

<https://www.cfainstitute.org/membership/professional-development/refresher-readings/industry-company-analysis>

<https://www.cfainstitute.org/membership/professional-development/refresher-readings/2018/discounted-dividend-valuation>

<https://medium.com/muthoni-wanyoike/implementing-text-summarization-using-openais-gpt-3-api-dcd6be4f6933>

<https://us-east-2.console.aws.amazon.com/s3/get-started?region=us-east-2&bucketType=general&region=us-east-2>

<https://github.com/openai/openai-cookbook/tree/main/examples/vector_databases>