**DAMG 7245 - Assignment 4 - Team 5**

| **Summary** | This project focuses on building an end-to-end data pipeline for extracting, validating, and storing PDF data into Snowflake using Airflow, Fast API, and Streamlit. Learners will gain expertise in workflow automation, API development, and user interface design, along with containerization using Docker for seamless deployment. |
| --- | --- |
| **URL** | <https://codelabs-preview.appspot.com/?file_id=1RqHOoUqMIQfOulQol3h0uNZgvZv2XoVlEHEZQiwK2no#9> |
| **Category** | Databases |
| **Environment** | Python |
| **Status** | Completed |
| **Github** | <https://github.com/BigDataIA-Spring2024-Sec1-Team5/Assignment-3> |
| **Authors** | Aditya Kanala, Shikhar Patel, Shubh Patel |

**Table of Contents**

[Architecture and Workflow](#_kq2jkt5kr1uc)

Overview

Workflow

[Streamlit](#_5a0ngon9cken)

[S3](#_lbvbvpnygws7)

[FastAPI](#_aewacfknq23m)

[Airflow](#_chiy56izyext)

[PyPDF](#_1y0tywcsjqqy)

[Grobid](#_vxidejhkg2lr)

Pydantic & Pytest

[Snowflake](#_18rjsos03hss)

[Streamlit Output](#_9jf9o2l6mjrc)

[References](#_xbxvb98sy4e0)

**PART - I**

# **Architecture and Workflow**

The workflow depicted in the image outlines a data processing and validation pipeline, with each node representing a service or a step in the process. Here's a brief description of the steps and the functionality of each service:

1. **Streamlit:**

Functionality: Acts as the starting point of the workflow, possibly representing the legacy system or the initial data sources.

Steps: Data from the Previous Architecture is exported in two formats:

CSV files, which are web-scraped CSV files.

XML files, which are generated by Grobid, a tool for extracting, parsing, and restructuring scholarly documents.

2. **S3:**

Functionality: Contains structured data scraped from web pages.

Steps: The CSV data is passed to the URL\_Class for schema validation and processing.

3. **Fast API:**

Functionality: XML files with metadata and content extracted from PDF documents using Grobid.

Steps: The XML data is split into two paths:

One leads to the Content\_Class for content extraction.

The other leads to an S3 bucket for metadata storage.

4. **Airflow:**

Functionality: These are Python classes that define the schema for the respective data types. They use Pydantic for data validation and Pytest for testing the data against the schema.

Steps: After validation and testing, the URL\_Class and Content\_Class output clean CSV files.

5. **PyPdf:**

Functionality: Represents the validated and cleaned data ready for further processing.

Steps: The Clean CSV files are loaded into Snowflake.

6. **Grobid:**

Functionality: A storage service provided by Amazon Web Services (AWS) for object storage.

Steps: Stores the metadata from the XML files.

7. **Pydantic & Pytest:**

Functionality: A cloud-based data warehousing service that allows for data storage, processing, and analytic solutions.

Steps: Receives the clean CSV data for storage and further transformations.

**8. Snowflake:**

Functionality: A command-line tool that enables data analysts and engineers to transform data in their warehouse more effectively.

Steps: dbt is used to transform the data within Snowflake, applying business logic, creating summary tables, and preparing the data for analysis.

**Workflow** A diagram of a diagram of a company

Description automatically generated with medium confidence

Architecture Diagram Code:

A screenshot of a computer

Description automatically generated

# **Streamlit**

**Overview**:

Streamlit provides a simple and intuitive way to create web applications directly from Python scripts. It offers various components for creating interactive elements such as sliders, buttons, text inputs, and plots. You can seamlessly integrate your data processing and visualization code with Streamlit, enabling rapid prototyping and deployment of web applications without needing to write HTML, CSS, or JavaScript code.

**Dependencies:**

Streamlit itself has minimal dependencies. However, depending on the functionality you want to include in your application, you may need additional libraries. For your assignment, you might need dependencies for handling file uploads, interfacing with Snowflake, and possibly other data processing libraries. Ensure to install these dependencies along with Streamlit.

**Setup:**

Install Streamlit: You can install Streamlit using pip, preferably within a virtual environment.

pip install streamlit

Create a Streamlit Script: Write your application code using Streamlit components. This script will define the layout and functionality of your web application.

**Usage:**

To run the script, simply execute it in a Python environment. The script performs the following steps:

1. Streamlit facilitates rapid development of interactive web applications directly from Python scripts, simplifying user interface creation.

2. Integration with data processing enables seamless incorporation of visualization, analysis, and machine learning models into web apps.

3. Interactive components like sliders and buttons enhance user engagement and customization within the application interface.

4. Flexible deployment options allow for local testing, cloud hosting, and easy sharing of applications online.

5. Streamlit's agile development approach enables quick prototyping and iteration, promoting efficient development cycles.

**Steps:**

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

**1. Imports and Environment Setup:**

- Imports necessary libraries and modules such as Streamlit, requests, os, boto3, pandas.

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



**2. Function to Send Query to Fast API Endpoint:**

- 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`.

A computer screen shot of text

Description automatically generated

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

- Defines a function `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.

- Displays success or error messages using Streamlit's `st.success` and `st.error`.

A computer screen shot of text

Description automatically generated

**4. Function to Execute Query in Snowflake:**

- Defines a function `execute\_query` to connect to Snowflake and execute a SQL query.

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

- Executes the query, fetches results, and returns them as a Pandas DataFrame.

- Handles errors and closes the connection in a finally block.

A computer screen shot of a program code

Description automatically generated

**5. Main Streamlit Application Function:**

- Defines the main function `main` of the Streamlit application.

- Sets up a sidebar with two options ('Upload PDF' and 'Query Response').

- Depending on the user's choice, initiates either file upload to S3 or displays query responses from Snowflake.

A screen shot of a computer program

Description automatically generated

**6. Conditional Branches for User Choice:**

- In the `main` function, based on the user's choice, it either remains unchanged or proceeds to handle file upload or query responses.

These points provide a structured overview of the code, outlining its key components and functionalities.

A black rectangle with white text

Description automatically generated

**Output:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

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 Name of the topic, Year, Level, Introduction, Learning Outcomes, Summary, Link to the Summary Page and Link to the PDF file.

# **S3**

**Introduction**

Amazon Simple Storage Service (S3) is a scalable object storage service offered by AWS. It allows you to store and retrieve any amount of data from anywhere on the web. S3 provides high availability, durability, and security for your data. It's commonly used for storing various types of data, including files, images, videos, and more.

**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.

**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”.

# 

# **FastAPI**

**Introduction**:

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints. It is built on top of Starlette for the web parts and Pydantic for the data parts, providing automatic validation, serialization, and documentation of API endpoints. FastAPI is known for its simplicity, performance, and automatic generation of interactive API documentation (Swagger UI and ReDoc).

**Installation and Setup:**

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

* Installation: Install FastAPI and Uvicorn (ASGI server) using pip.
* Create an API Script: Write your API endpoints in a Python script using FastAPI decorators.
* Run the API: Start the FastAPI application using Uvicorn.

**Steps:**

1. Imports and Environment Setup:

A screenshot of a computer program

Description automatically generated

1. Set AWS Credentials and Bucket Name:

A screen shot of a computer

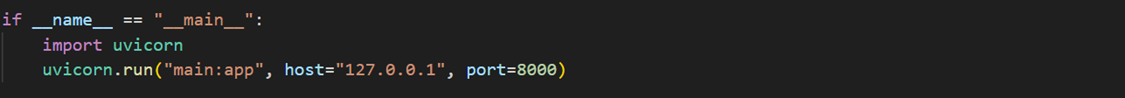
Description automatically generated

1. Define Airflow DAG URL and Fast API Endpoint to trigger DAG:

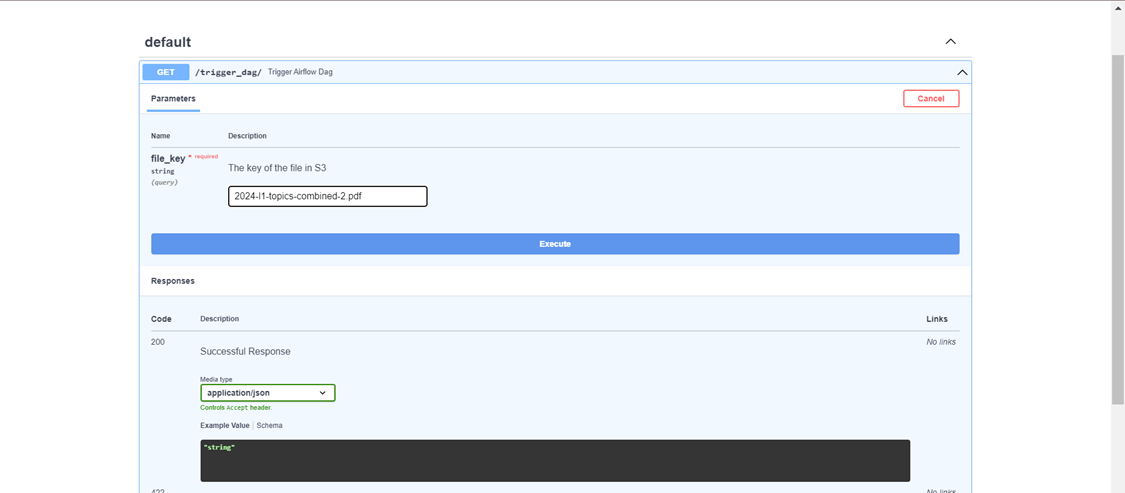
A screen shot of a computer program

Description automatically generated

1. Running the FastAPI Application:



Output:



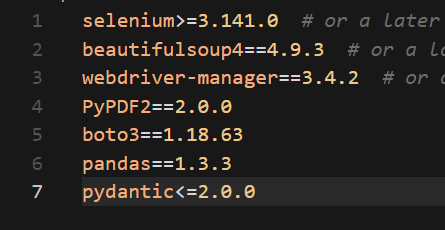
# **Airflow**

Apache Airflow is an open-source platform to programmatically author, schedule, and monitor workflows. It allows you to define workflows as directed acyclic graphs (DAGs) of tasks, where each task represents a specific action to be performed. Airflow provides a rich set of features including task dependencies, scheduling, retrying, monitoring, and logging, making it suitable for orchestrating complex data pipelines and workflows.

For executing the Airflow dags, firstly we would need 3 basic files,

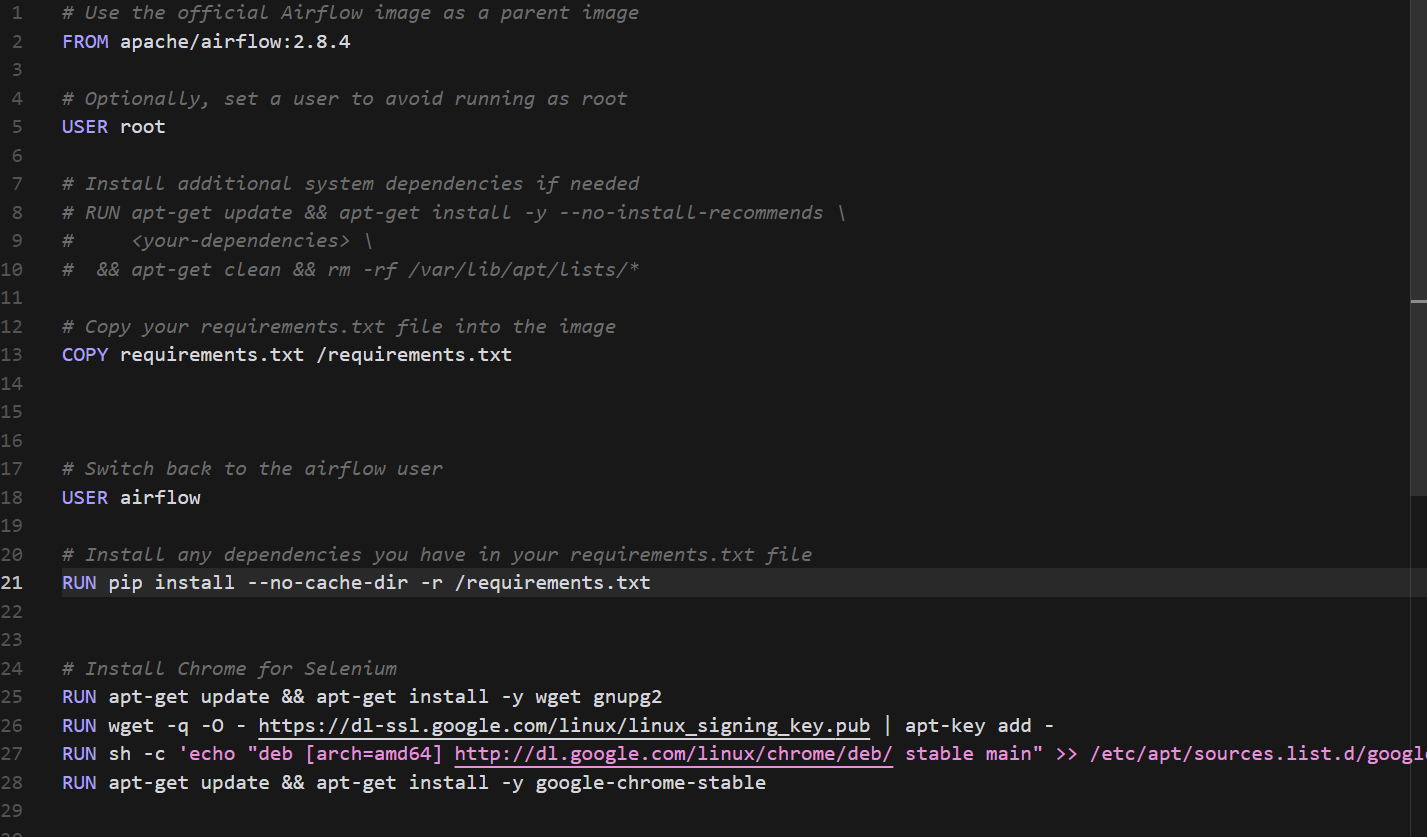
Requirements.txt

Consists of the required packages to run the workflows.



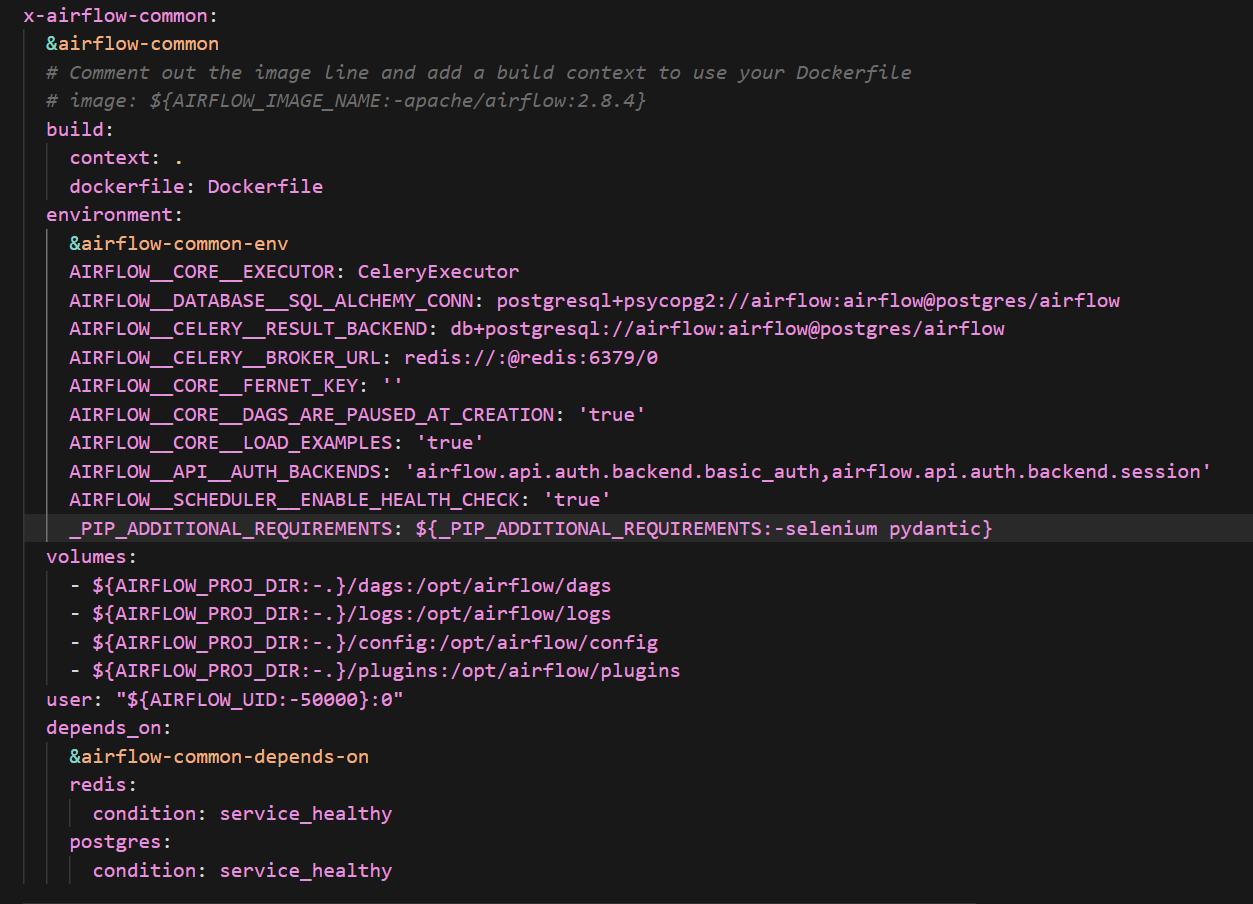
Dockerfile

This contains the commands for running the requirements file and other packages needed.



Docker-compose.yaml

This contains the containers required to run the dag workflows.



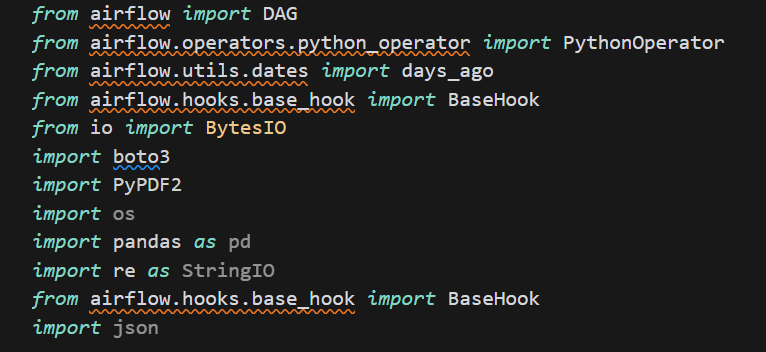
# **PyPDF**

**Overview**

This Apache Airflow DAG (pypdf\_extraction.py) is designed to automate the process of extracting text from PDF files stored in an AWS S3 bucket, and then uploading the extracted text back to a different S3 bucket. The process involves several steps, each encapsulated in a Python function. Below is an overview of each component and its functionality.

Dependencies:

The script imports necessary modules for its operation, including Airflow components, boto3 for interacting with AWS S3, PyPDF2 for PDF processing, and other standard Python libraries.



**Functions:**

**clean\_text(text)**

Cleans the extracted text by removing excessive newline characters and leading/trailing whitespace.

* **Parameters**: text (str) - The text to be cleaned.
* **Returns**: Cleaned text (str).

**download\_pdf\_from\_s3(bucket\_name, s3\_key, local\_path, aws\_conn\_id)**

Downloads a PDF file from an S3 bucket to a local path.

* **Parameters**:
* bucket\_name (str) - The name of the S3 bucket.
* s3\_key (str) - The S3 key where the PDF is stored.
* local\_path (str) - The local file path to save the downloaded PDF.
* aws\_conn\_id (str) - The Airflow connection ID for AWS credentials.

**extract\_text\_pypdf(file\_path)**

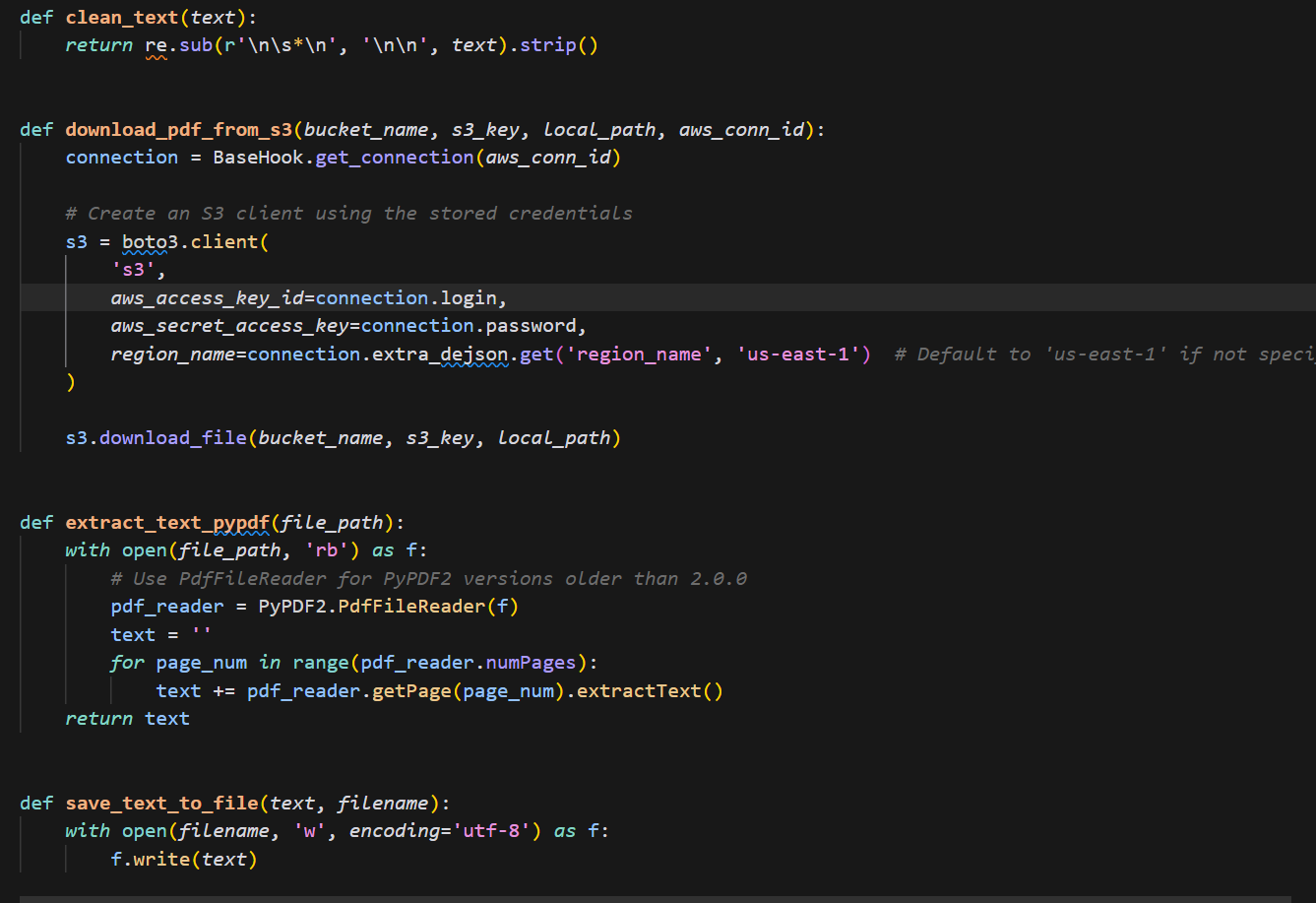
Extracts text from a PDF file using PyPDF2.

* **Parameters**: file\_path (str) - The local path to the PDF file.
* **Returns**: Extracted text (str).

**save\_text\_to\_file(text, filename)**

Saves the extracted text to a local file.

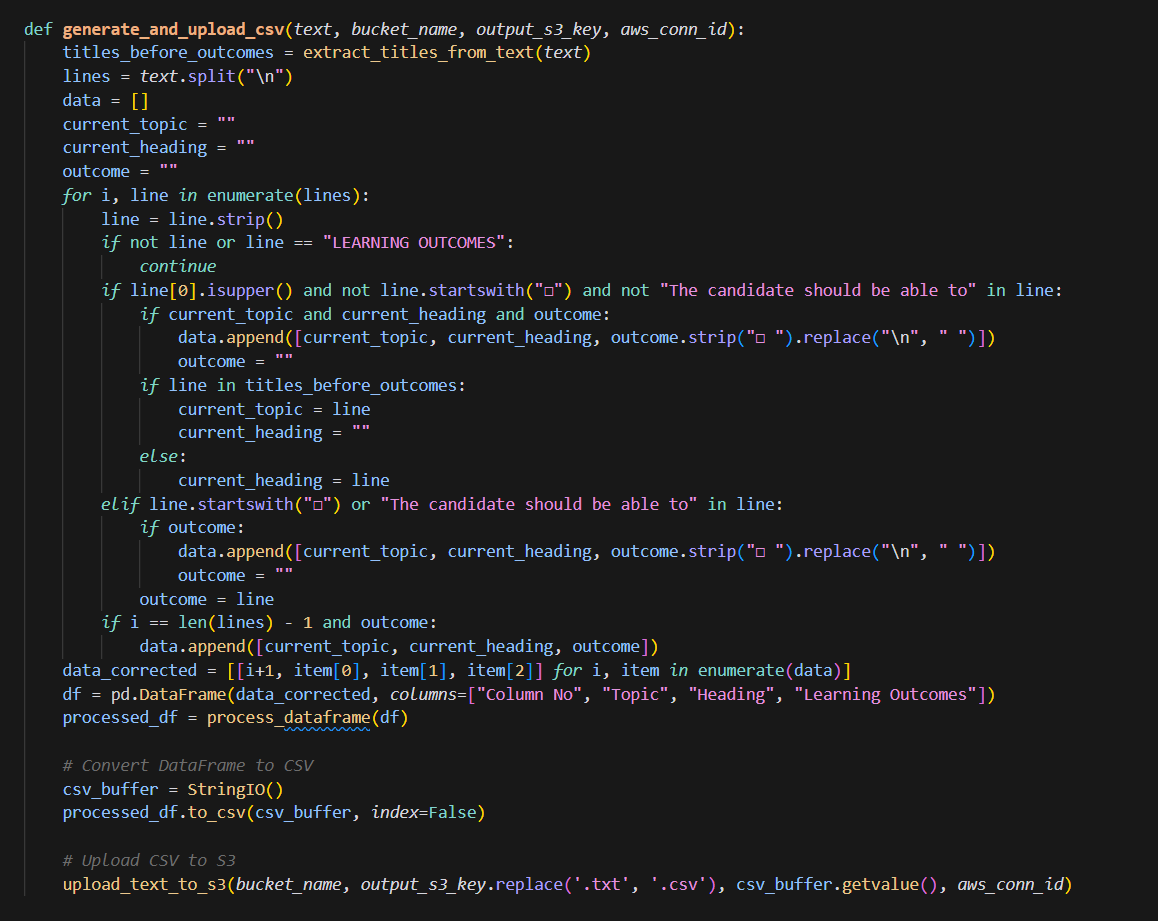
* **Parameters**:
* text (str) - The text to save.
* filename (str) - The name of the file to save the text in.



**generate\_and\_upload\_csv(text,bucket\_name, s3\_key,aws\_conn\_id)**

Generates and uploads the csv file from txt file to the s3..

* **Parameters**:
* bucket\_name (str) - The name of the S3 bucket.
* s3\_key (str) - The S3 key to upload the text to.
* text (str) - The text to upload.

aws\_conn\_id (str) - The Airflow connection ID for AWS credentials. 

**upload\_text\_to\_s3(bucket\_name, s3\_key, text, aws\_conn\_id)**

Uploads the extracted text to an S3 bucket.

* **Parameters**:
* bucket\_name (str) - The name of the S3 bucket.
* s3\_key (str) - The S3 key to upload the text to.
* text (str) - The text to upload.
* aws\_conn\_id (str) - The Airflow connection ID for AWS credentials.

**process\_pdf(bucket\_name, s3\_keys, output\_s3\_bucket, aws\_conn\_id)**

The main function that orchestrates the download of PDFs from S3, extracts text from them, and uploads the text back to a different S3 bucket.

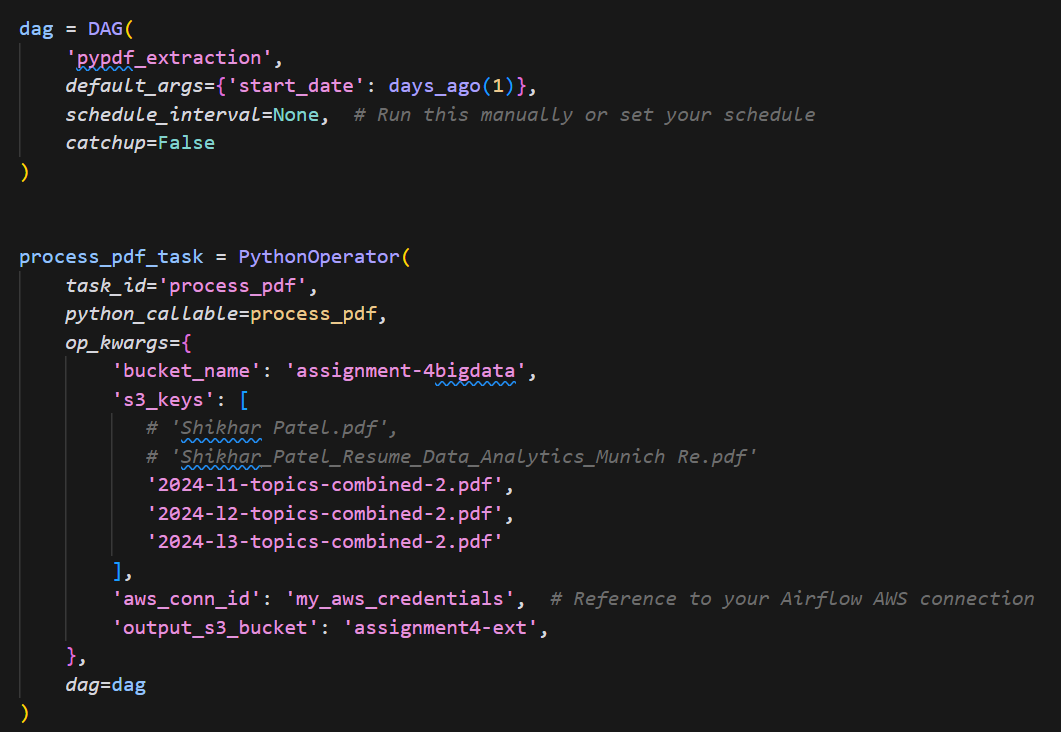
* **Parameters**:
* bucket\_name (str) - The name of the S3 bucket containing the PDFs.
* s3\_keys (list) - A list of S3 keys for the PDFs to process.
* output\_s3\_bucket (str) - The name of the S3 bucket to upload the extracted text to.
* aws\_conn\_id (str) - The Airflow connection ID for AWS credentials.



Dag and Tasks:

The DAG is defined with the ID pypdf\_extraction, set to start a day ago, with no specific schedule (intended to be triggered manually or on a custom schedule), and with catchup turned off.

A single task, process\_pdf\_task, is defined using the PythonOperator. It is configured to call the process\_pdf function with specific arguments for the S3 bucket names, keys, and AWS connection ID.



# **Grobid**

**Overview**

This Apache Airflow Directed Acyclic Graph (DAG), named grobid\_extraction, is designed to automate the extraction of text from PDF files using Grobid, a machine learning library for extracting and processing PDF content. The extracted text is then structured, validated, and uploaded as CSV files to an AWS S3 bucket.

## **Dependencies**

Airflow for workflow management.

Boto3 for AWS S3 interactions.

Pydantic for data validation.

lxml for XML parsing.

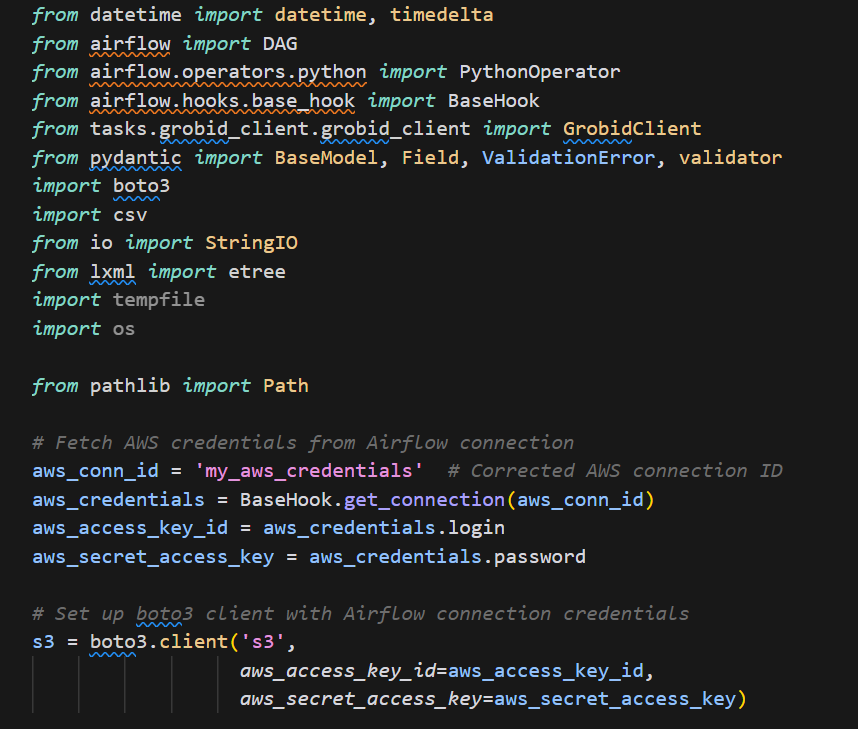
GrobidClient for interacting with the Grobid service.

## **Configuration and Setup**

AWS credentials are fetched from an Airflow connection 1.

A Boto3 client is configured for S3 interactions using these credentials.

The GrobidClient is initialized with a configuration file path.



## **DAG Configuration**

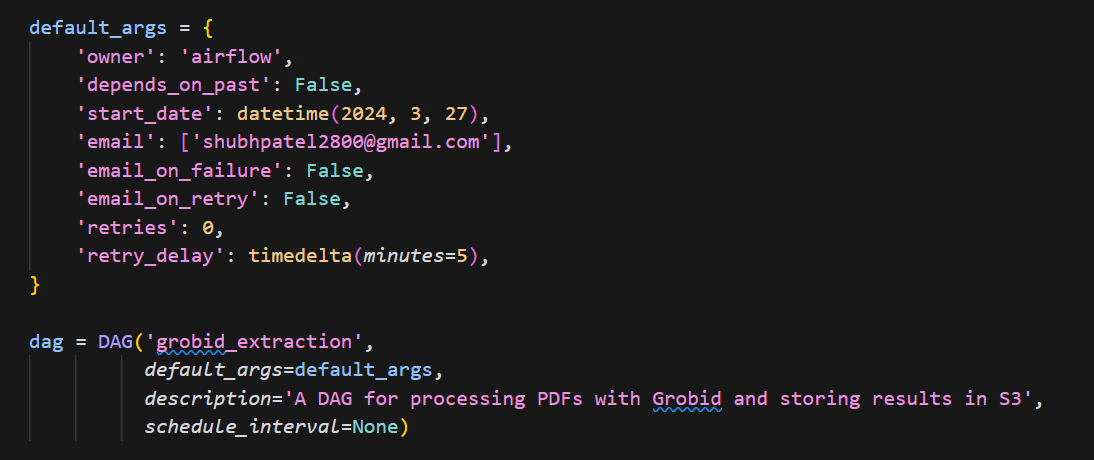
**Owner**: airflow

**Start Date**: March 27, 2024

**Email**: Specified for notifications.

**Retries**: None

**Schedule**: Manual trigger (no schedule).



**Tasks**

### **extract\_text\_from\_pdf**

**Description**: Downloads a PDF from a specified S3 bucket and file key, processes it using Grobid to extract text, and saves the result locally.

**Input**: Bucket name and file key for the PDF.

**Output**: Processed PDF content saved locally.

### **validate\_and\_structure\_data**

**Description**: Validates and structures the extracted text into a predefined format using Pydantic models.

**Input**: XML data from the Grobid extraction.

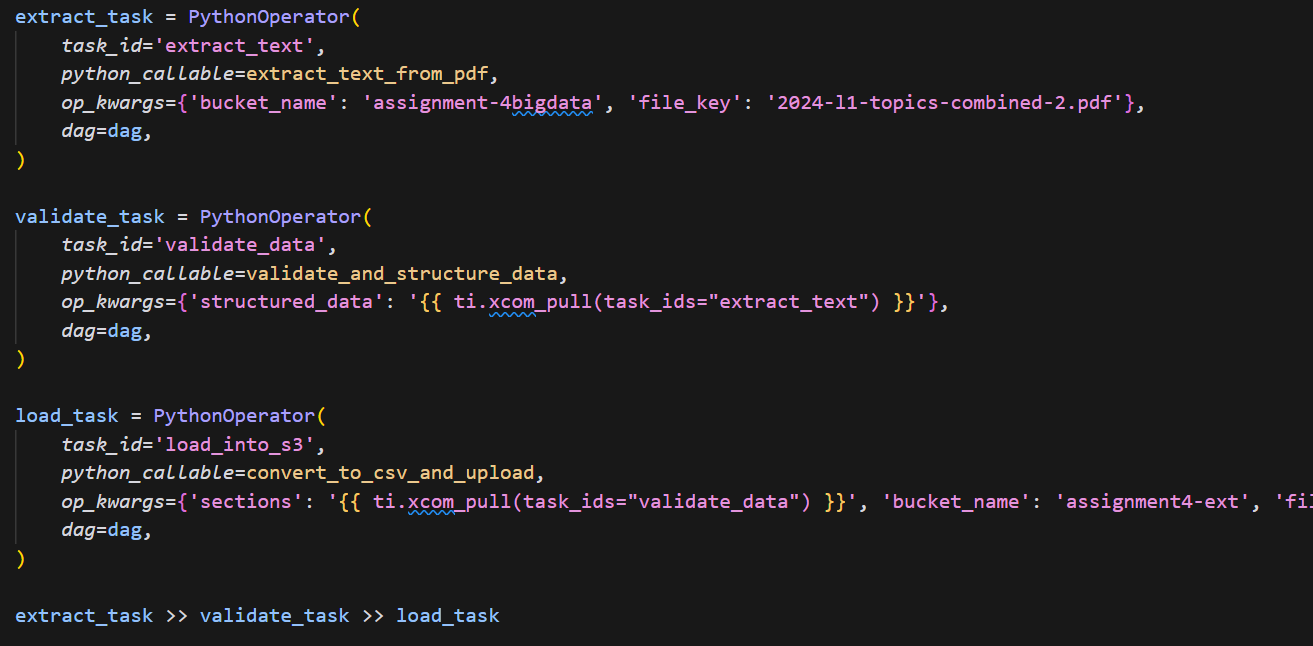
**Output**: A list of validated and structured document sections.

### **convert\_to\_csv\_and\_upload**

**Description**: Converts the structured document sections into CSV format and uploads the CSV file to an S3 bucket.

**Input**: Structured document sections.

**Output**: CSV file uploaded to S3.



## **Workflow**

1. **Extract**: The extract\_text\_from\_pdf task downloads a PDF from S3 and processes it with Grobid.

2. **Validate and Structure**: The validate\_and\_structure\_data task takes the Grobid output, validates, and structures it.

3. **Load**: The convert\_to\_csv\_and\_upload task converts the structured data into CSV format and uploads it to S3.

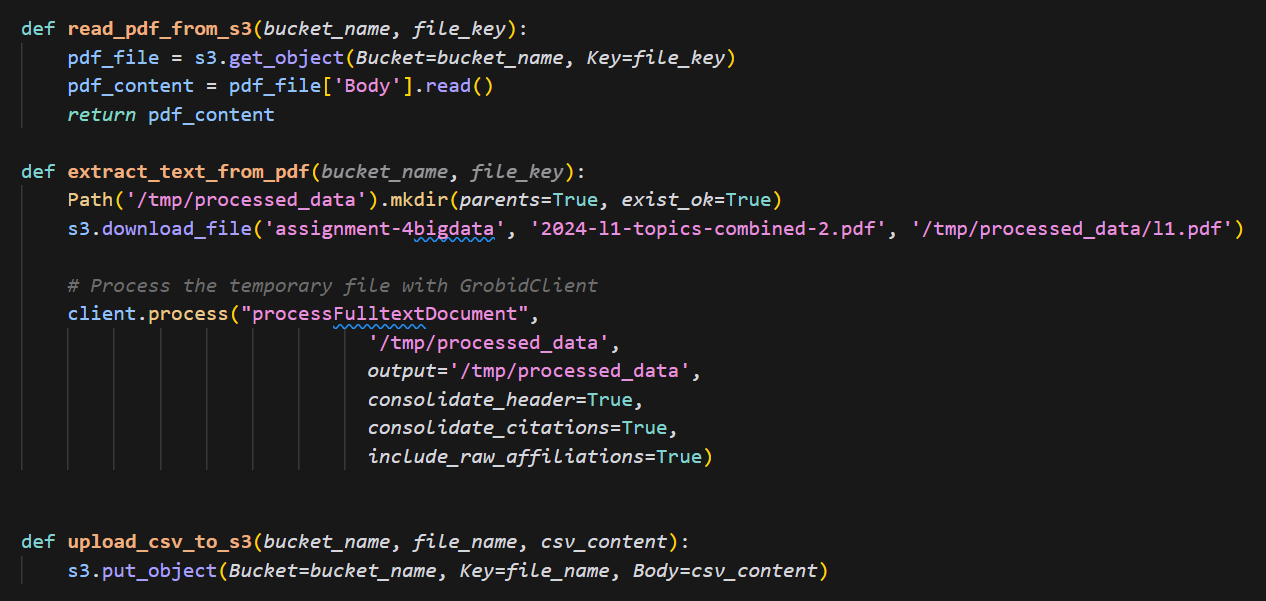
## **Functions**

### **read\_pdf\_from\_s3**

Fetches a PDF file from S3.

### **extract\_text\_from\_pdf**

Processes a PDF with Grobid and saves the output locally.

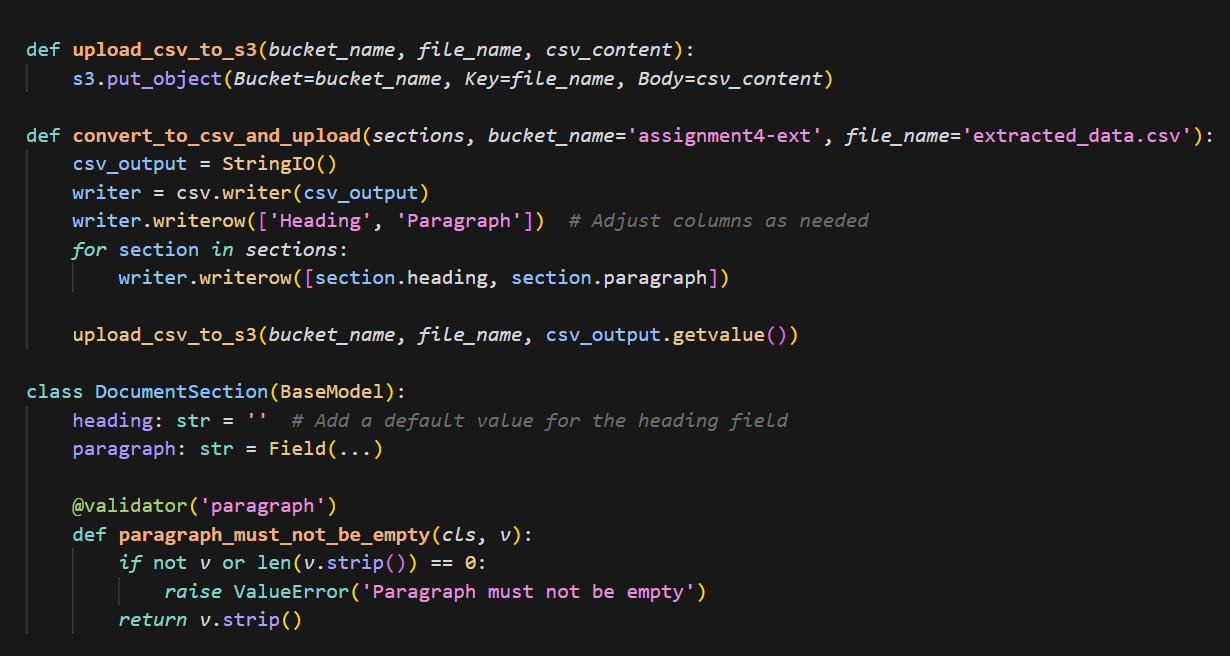


### **upload\_csv\_to\_s3**

Uploads a given CSV content to an S3 bucket.

### **convert\_to\_csv\_and\_upload**

Converts structured data to CSV and uploads it to S3.

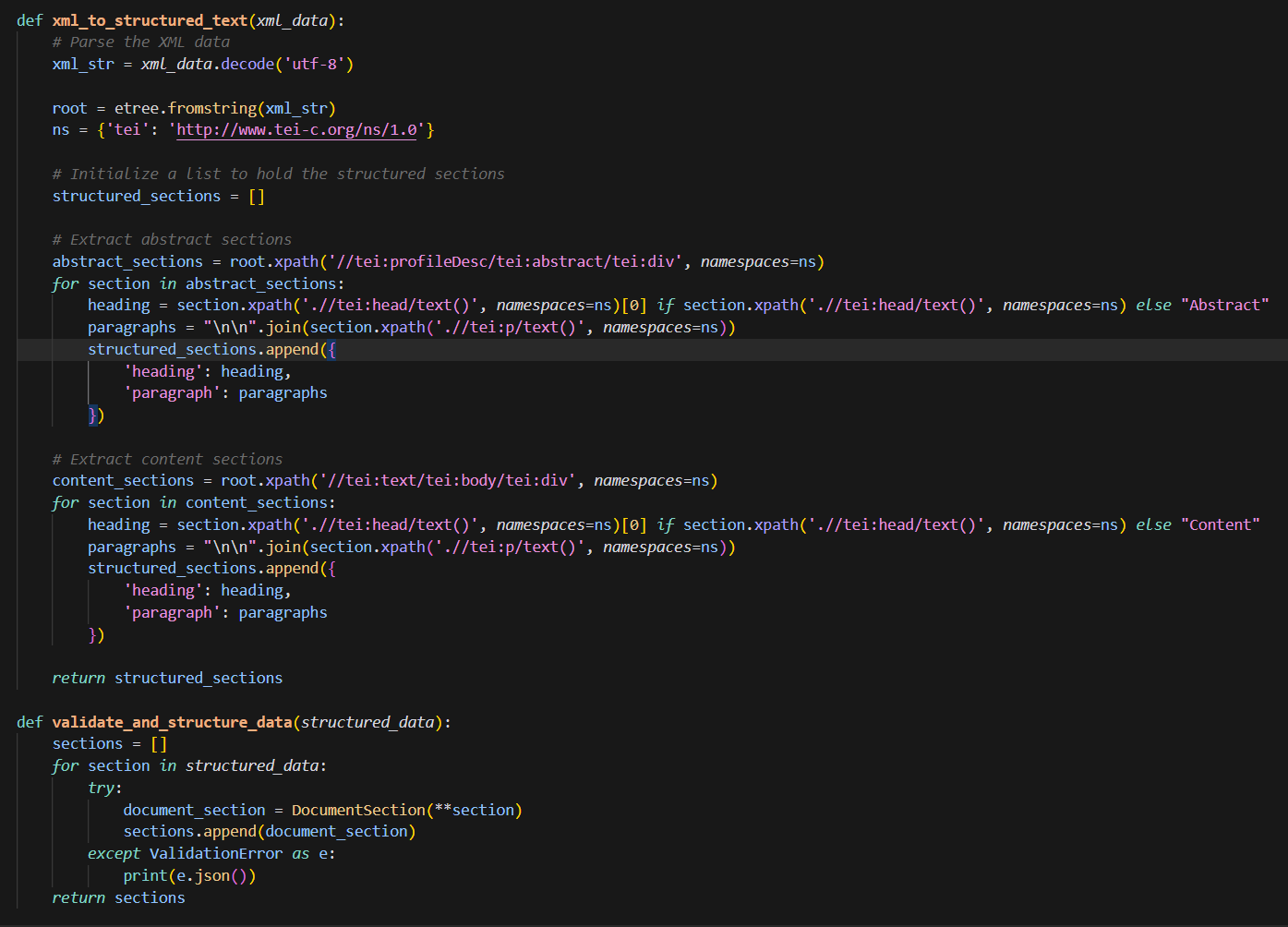


### **xml\_to\_structured\_text**

Parses XML data from Grobid into structured text sections.

### **validate\_and\_structure\_data**

Validates and structures raw data into DocumentSection objects



**WebScrapping**

This Apache Airflow Directed Acyclic Graph (DAG), named cfa\_scraper, is designed to automate the process of scraping reading materials from the CFA Institute's website and uploading the extracted data to an AWS S3 bucket. The process involves scraping links to reading materials, extracting relevant information from each reading, saving the data into a CSV file, and then uploading this file to S3.

## **Dependencies**

Airflow for workflow management.

Selenium WebDriver for web scraping.

BeautifulSoup for parsing HTML content.

Boto3 for AWS S3 interactions.

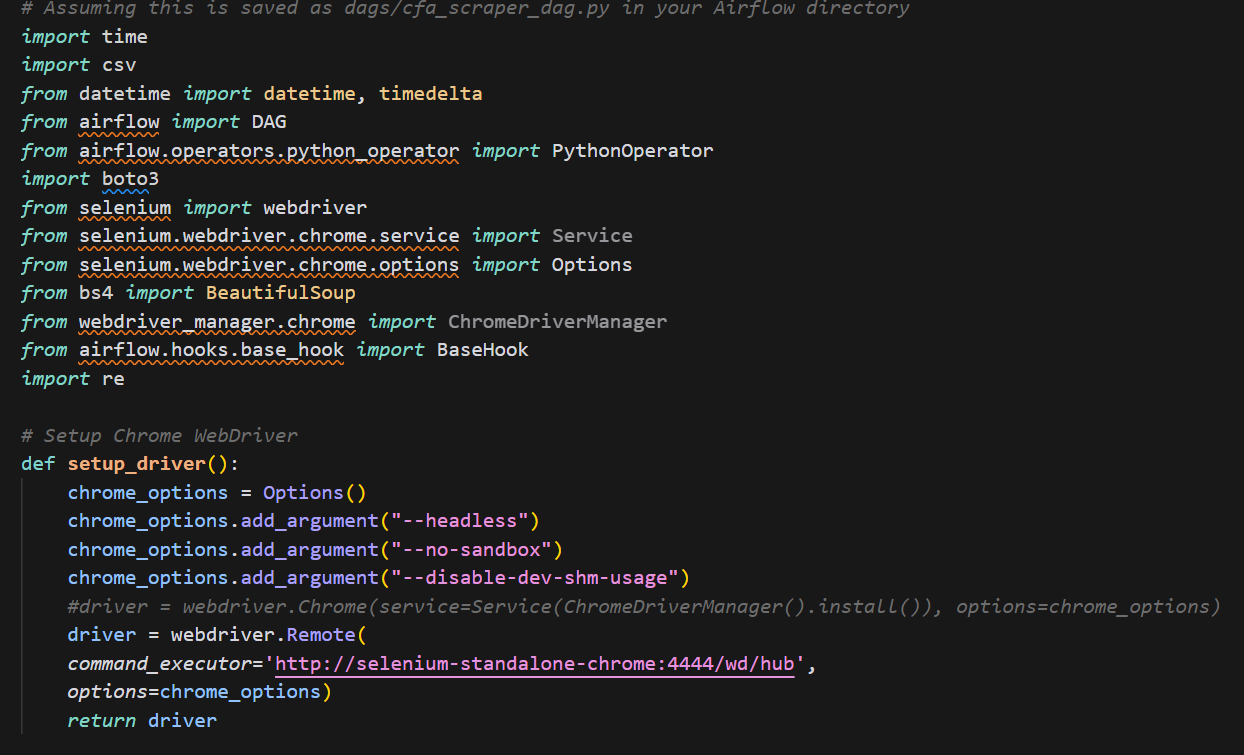
CSV for data serialization.

Regular expressions (re) for text processing.

## **Configuration and Setup**

AWS credentials are fetched from an Airflow connection 1.

Chrome WebDriver is set up in headless mode for Selenium to interact with web pages without a GUI.

****

## **Tasks**

### **scrape\_and\_upload**

Description: Orchestrates the scraping, data processing, and uploading tasks.

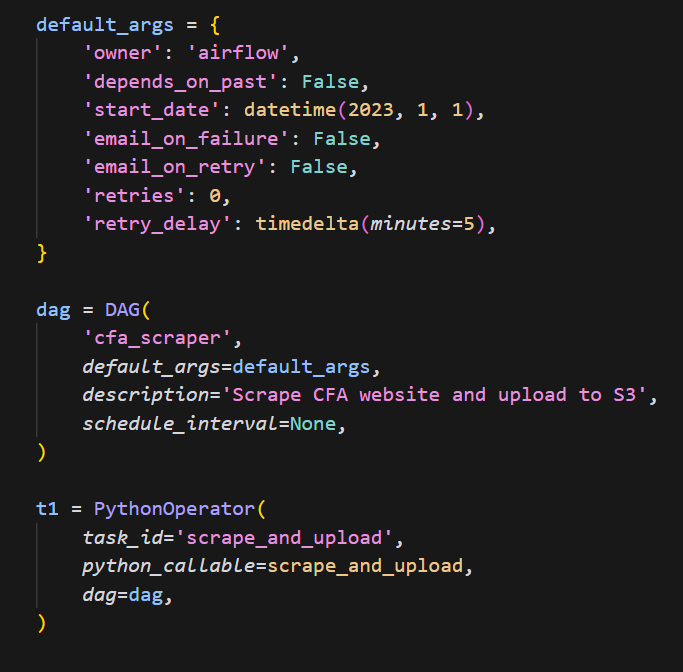
Functionality:

1. Scrape Links: Fetches links to reading materials from specified URLs.

2. Extract Data: Scrapes title, topic, year, level, introduction, learning outcomes, summary, and PDF link for each reading.

3. Save to CSV: Serializes the extracted data into a CSV file.

4. Upload to S3: Uploads the CSV file to an AWS S3 bucket.



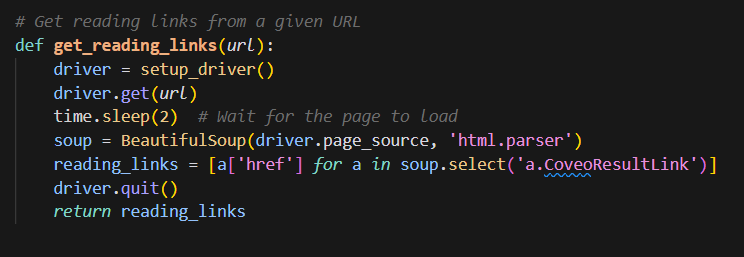
**Functions**

### **setup\_driver()**

Initializes and returns a Selenium WebDriver configured for Chrome in headless mode.

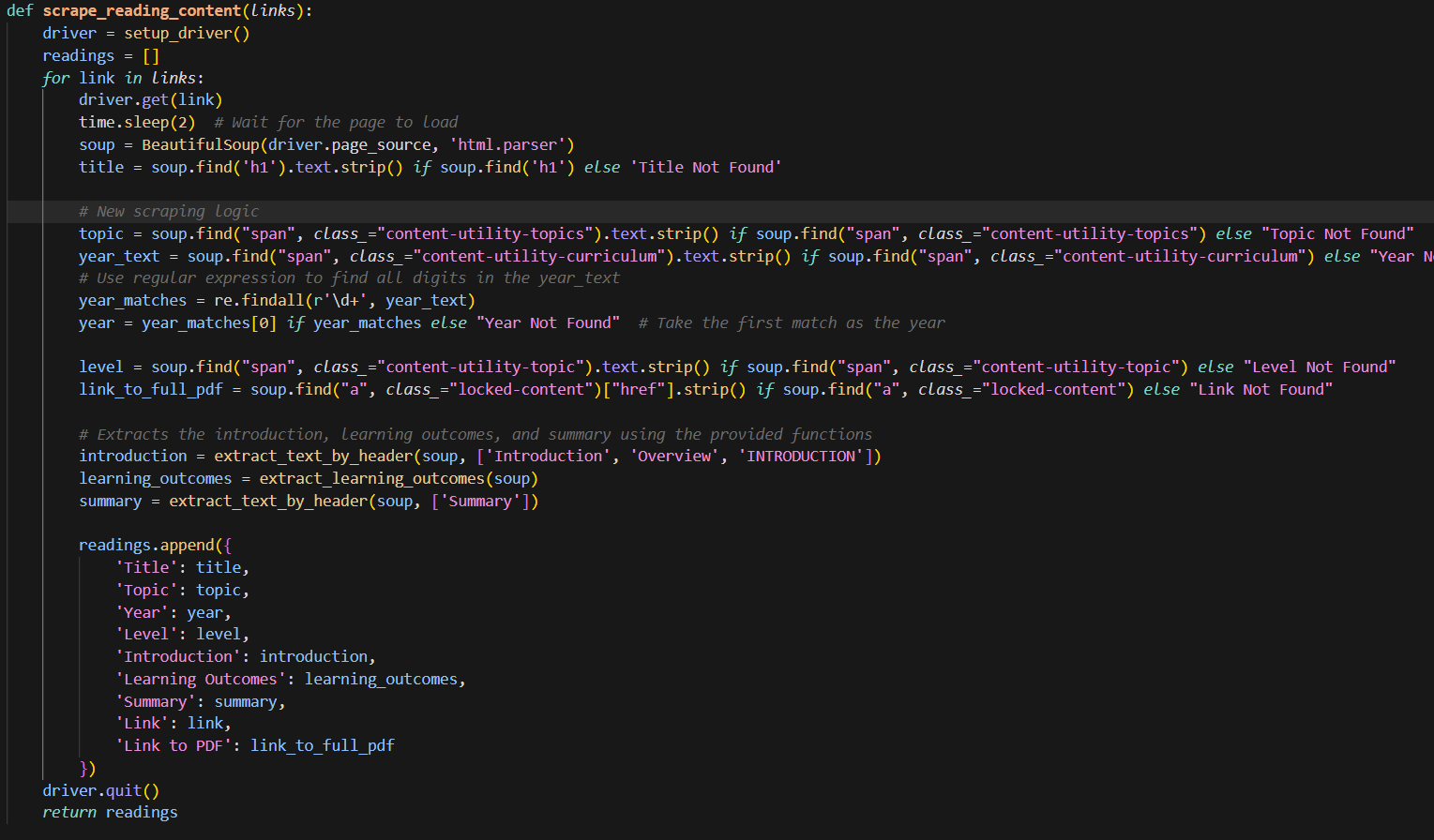
### **get\_reading\_links(url)**

Fetches and returns a list of links to reading materials from a given URL.



### **scrape\_reading\_content(links)**

Extracts relevant information from each reading link and returns a list of dictionaries containing the data.



### **extract\_text\_by\_header(soup, header\_texts)**

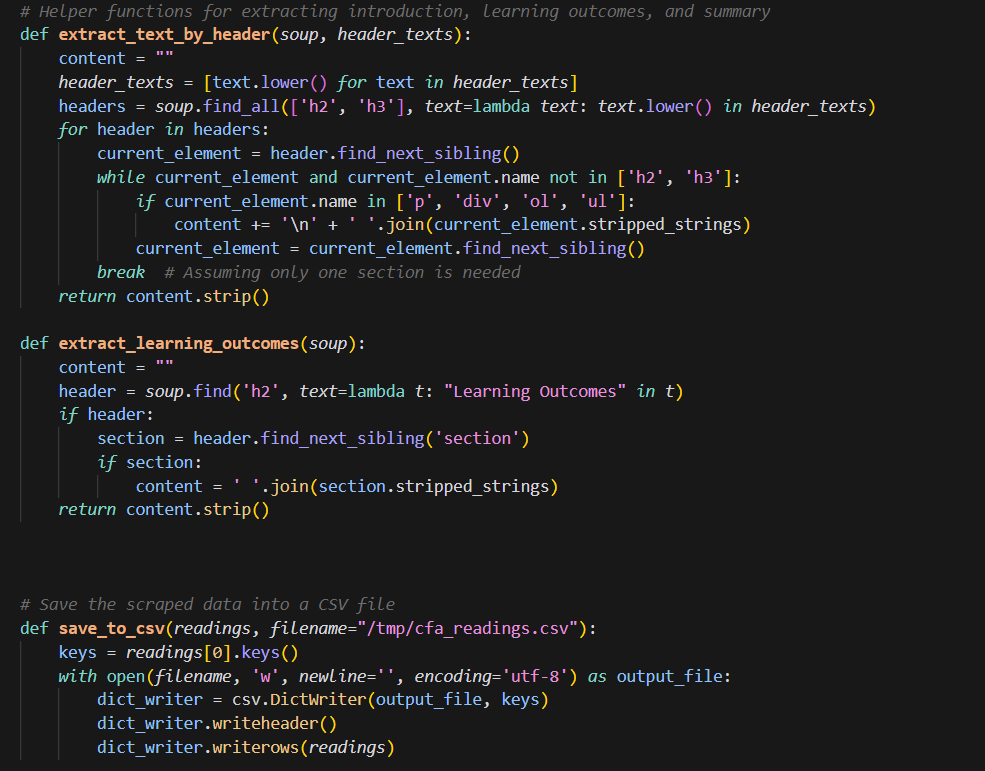
Extracts and returns text content based on specified headers from a BeautifulSoup object.

### **extract\_learning\_outcomes(soup)**

Extracts and returns the learning outcomes section from a BeautifulSoup object.

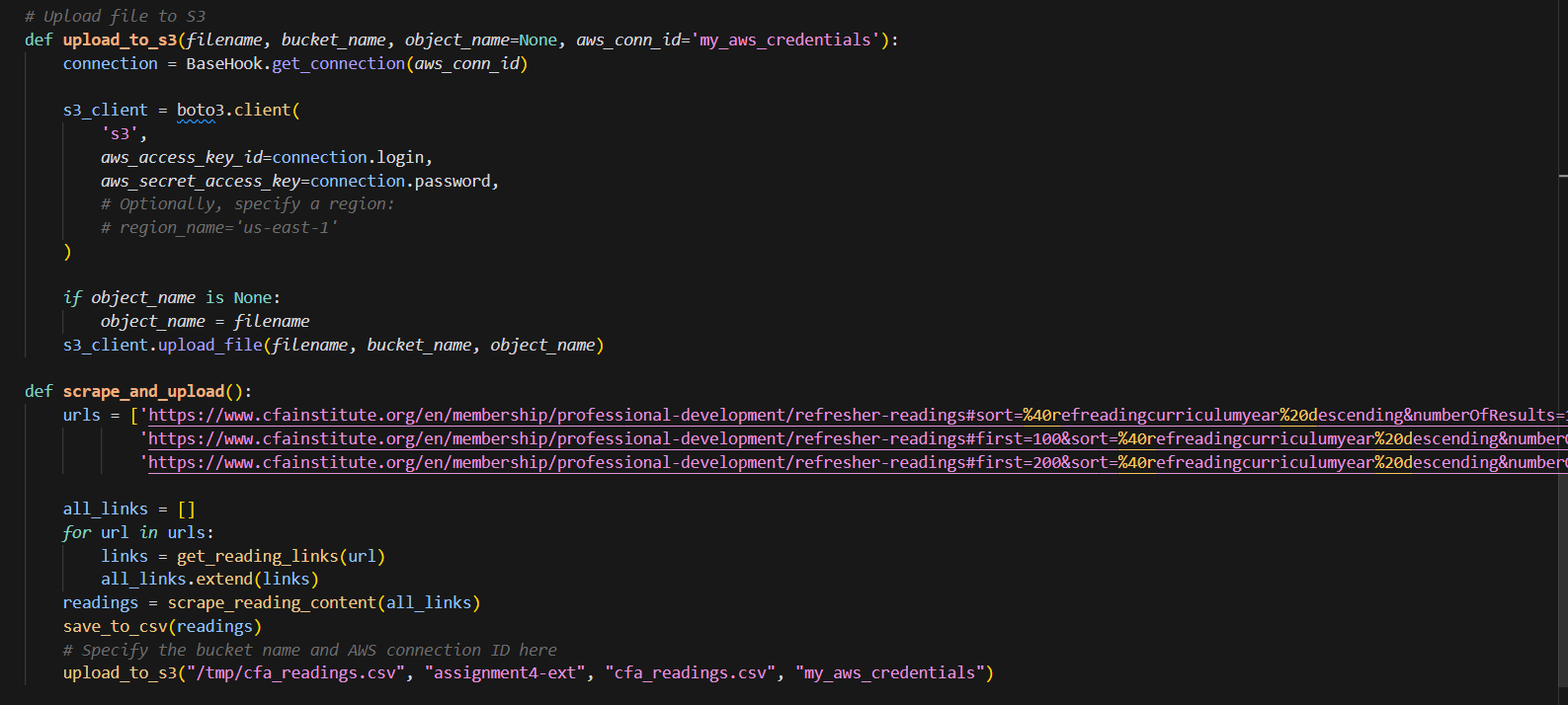
### **save\_to\_csv(readings, filename)**

Serializes the list of reading dictionaries into a CSV file.



### **upload\_to\_s3(filename, bucket\_name, object\_name, aws\_conn\_id)**

Uploads a file to an AWS S3 bucket using Boto3 and Airflow's AWS connection.



# **Snowflake**

# 

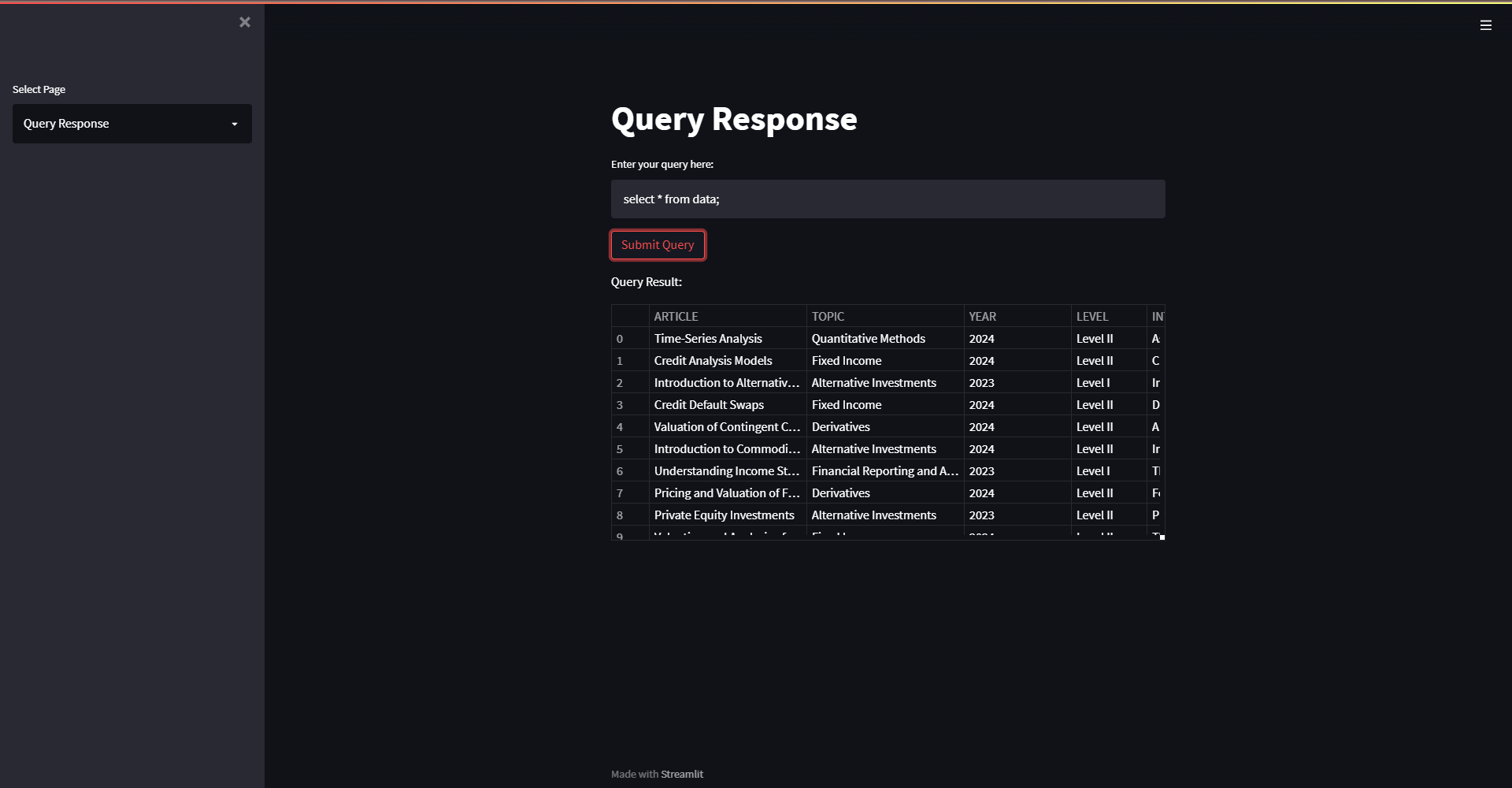
Snowflake is a cloud-based data warehousing platform that allows you to store and analyze large volumes of structured and semi-structured data. It offers a fully managed service with features such as automatic scaling, built-in security, and support for diverse data types. Snowflake uses a multi-cluster architecture to separate compute and storage, enabling elastic scaling and efficient query processing.

Output:

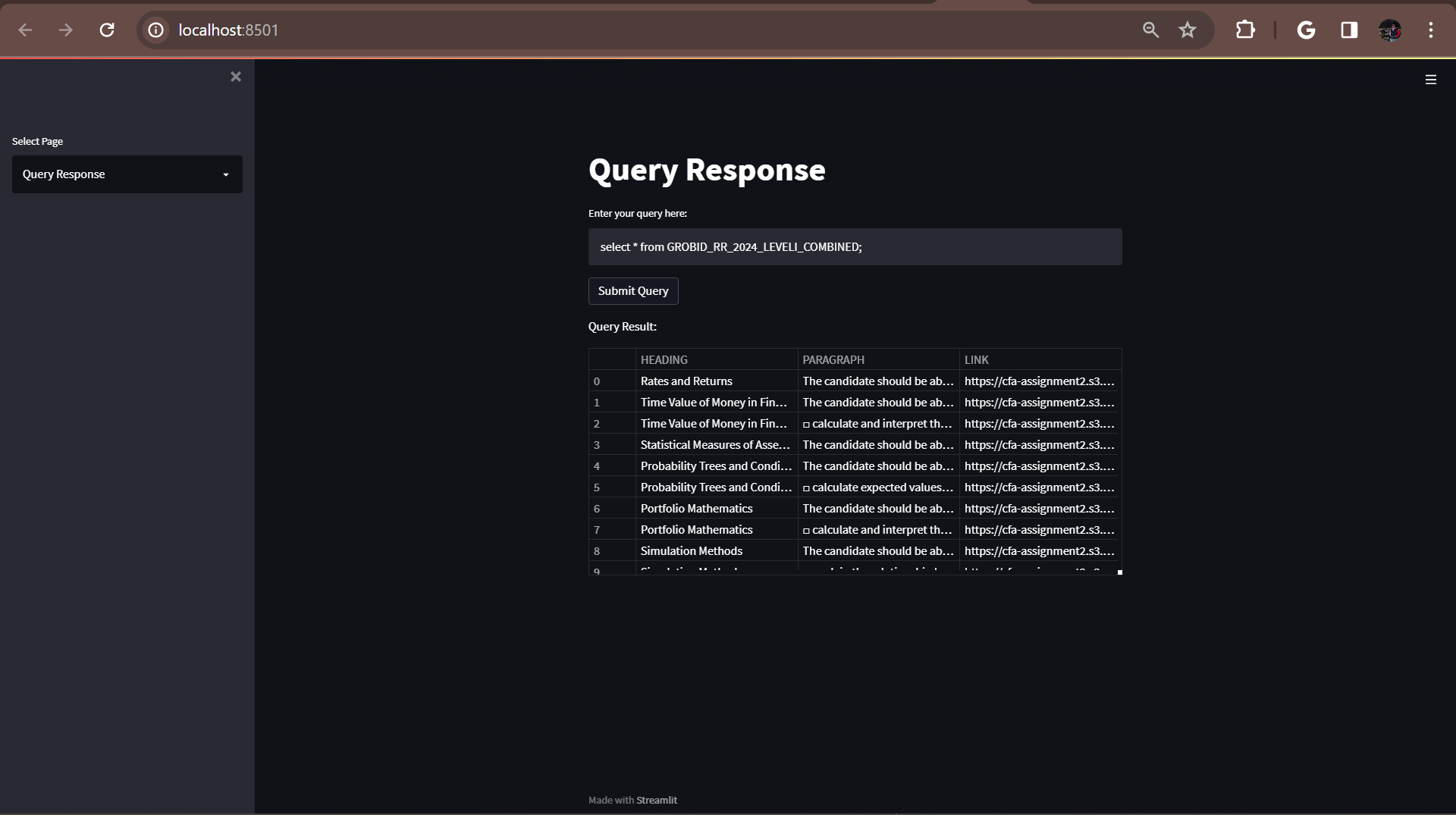
A screenshot of a computer

Description automatically generated

# **Streamlit Output:** **PART - II**

Query Response from the Web Scraped data:

Query Response from the PDF data:



# **References**

[https://airflow.apache.org/docs/apache-airflow/stable/index.html](https://www.getdbt.com/)

[https://docs.docker.com/](https://docs.getdbt.com/guides/snowflake?step=1)

<https://docs.snowflake.com/en/developer-guide/python-connector/sqlalchemy>

<https://docs.pytest.org/en/8.0.x/>

<https://docs.pydantic.dev/latest/>

[https://grobid.readthedocs.io/en/latest/Introduction/](https://docs.pydantic.dev/latest/)

<https://learn.microsoft.com/en-us/azure/architecture/ai-ml/>