**Creating a Sensemaking Data Pipeline**

**Part 1: Code Development**

A folder titled project-23 was created. The code visualization folder was placed inside the project-23 folder, and another folder titled airflow-docker was created. A new assignment.py file was created, which contained the coding for the project.

A screenshot of a computer

Description automatically generated

The assignment.py file was opened using VS Code. The necessary libraries were imported as follows:

* The DAG object from Airflow and datetime were imported.
* Operators from airflow.operators were imported.
* Task Functions such as urllib.request, time, glob, and os were imported. A screenshot was provided, displaying the imported DAG object, operators, and all the necessary task functions in the assignment.py file.

A screenshot of a computer program

Description automatically generated

The first task, a Python function named catalog(), was generated. It did not accept any arguments. Inside this function, two helper functions named pull(url) and store(data, file) were defined.

A screenshot of a computer program

Description automatically generated

The two helper functions were then utilized within the catalog() method. A list titled urls was created containing the working URLs in the 00\_urls.txt file. A for loop was written to iterate through the urls list, first calling the pull(url) function and then the store(data, file) function.

A screenshot of a computer

Description automatically generated

The second task in the Airflow pipeline, named combine(), was created within the assignment.py file. This task combined all the unstructured data files into one.

A screenshot of a computer program

Description automatically generated

The third task method, titles(), was created within the assignment.py file. This function utilized the BeautifulSoup4 library for web scraping. The titles() function took no arguments and imported the BeautifulSoup4 library. One helper function, store\_json(), was defined to store the resulting JSON file.

A screen shot of a computer

Description automatically generated

The next task method, clean(), was created in the assignment.py file. This method removed all punctuation, numbers, and one-character words from the titles.json file.

A screen shot of a computer

Description automatically generated

The final task method, count\_words(), was defined in the assignment.py file. A screenshot was provided, showing the completed count\_words() method with the code that called the store\_json(data, file) helper function.

A screen shot of a computer

Description automatically generated

The assignment.py file was continued, and an Airflow pipeline was designed. The DAG was first defined, and then each task from t0 to t5 was defined, resulting in a total of six tasks.

A screenshot of a computer

Description automatically generated

**Part 2: Code Execution**

The command was executed in a Terminal window to copy the assignment.py file into the airflow-docker/dags folder. Airflow pipeline was then initiated following the steps provided. A screenshot was provided, showing the initiation of the Airflow Docker container.

A screenshot of a computer

Description automatically generated

The Airflow session was accessed via <http://localhost:8080/>. The assignment.py DAG was selected, and tasks were run. A screenshot was provided, showing that the DAG had run successfully.

A screenshot of a computer

Description automatically generated

The generated files by the DAG were located on the worker\_1 airflow server. The server was accessed using a bash command.

A screenshot of a computer screen

Description automatically generated

The words.json file was transferred to the local machine, and its data was placed into the words.js file within the code\_visualization folder.

A screenshot of a computer screen

Description automatically generated

A screenshot of the words.js file with the data from the words.json file was provided.

A screen shot of a computer

Description automatically generated

The JavaScript visualization code was run by accessing the mitcourses\_graph.html file in a web browser.

A screenshot of a computer screen

Description automatically generated

The bubble chart was enhanced using the D3 library and the provided example code. The d3\_bubble\_chart\_example.html file was modified to display the words from the words.json file.

A screenshot of a computer screen

Description automatically generated