網路成品參考：<https://github.com/vladyslav-honcharuk/sensemaking-data-pipeline/tree/main>

In this project, you will use the Python urllib *library* to *pull* data from MIT’s course catalog. The data being *pulled* from the web will be unstructured, so you need to clean it first to extract the course names. Once the course names are extracted and the data is structured, you will perform some data analysis to learn how many times each word occurs throughout all of the course names. This analysis will be saved as a JSON file, which will then be referenced by your D3 web application. Lastly, the web application will generate a visual analysis of the JSON data that you just collected.

To streamline the data analysis process, you will use software tools that you have learned about in previous modules, such as Docker and Airflow. You will be responsible for instantiating an Airflow web *server* within a Docker *container* to handle each Python *task* that you define. These *tasks* will divide the project into smaller, more manageable pieces. In the end, you will experiment with different D3 *libraries* to customize your project.

As you write the code for this project, it is strongly recommended that you run each *method* independently as it is created so that you can verify that each *task* works as expected.

This project is worth a total of 100 points.

**To complete this project, follow these steps:**

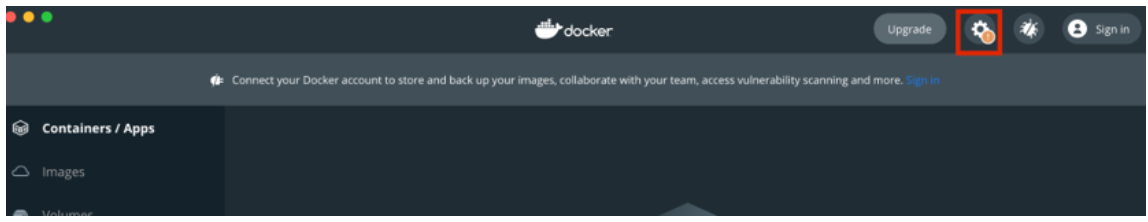
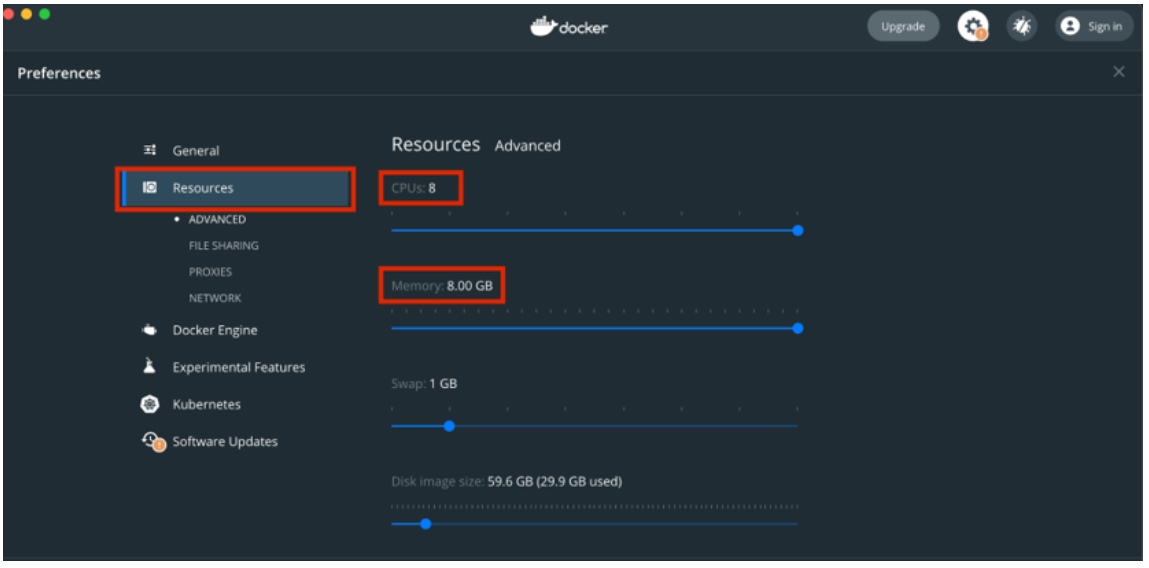
**Part 1: Code Development**

1. Create a folder titled project-23. Place the [code visualization](https://classroom.emeritus.org/courses/10605/files/3006859/download) folder within the project-23 folder, and create another empty folder titled airflow-docker. Then, create a new assignment.py file, which will contain your coding throughout the project. Provide a screenshot of the project-23 folder with the code visualization folder, airflow-docker folder, and assignment.py file within it.
2. Open the assignment.py file using VS Code. First, you need to import all of the *libraries* you will use. Import the following *libraries*:  
   The DAG object (needed to instantiate a DAG)
   * 1. From airflow, import DAG.
     2. From datetime, import timedelta.
3. *Operators* (needed to operate)
   * 1. From airflow.operators.bash, import BashOperator.
     2. From airflow.utils.dates, import days\_ago.
     3. From airflow.operators.python, import PythonOperator.
4. *Task Functions* (used to facilitate *tasks*)
   * 1. Import urllib.request.
     2. Import time.
     3. Import glob, os.
     4. Import json.
5. Provide a screenshot to show that you have imported the DAG object, the *operators*, and all of the necessary *task functions* into the assignment.py file.
6. Generate your first *task*. This *task* will be a Python *function* named catalog()that accepts no *arguments*. Inside the catalog()*function*, define two helper *functions* named pull(url)and store(data, file).  
   The pull(url)*function* accepts, as an *argument*, a URL of type *string*. This *function* will execute the code provided in lines 6 and 7 in the [code\_starter/01\_pull.py](https://classroom.emeritus.org/courses/10605/files/3006880/download) file and will return the data variable.  
   The store(data, file)*function* accepts two *arguments*: a filename of type *string* and the data returned by the pull()*function*. The store()*method* will execute the following:
   * Create and open a file named after the URL where the data came from (example: m1a.html).
   * Write the data to the file.
   * Close the file.
7. Below, you are provided with the pseudocode to define the catalog()*function* and thepull(url)and store(data, file)helper *functions.* Note: This is just pseudocode; you will need to complete it with the correct syntax:  
   # pull course catalog pages  
   def catalog():  
    #define pull(url) helper function  
    def pull(url):  
      
    return data  
      
    #define store(data,file) helper function  
    def store(data,file):  
      
    print('wrote file: ' + file)  
   Provide a screenshot of the pull(url)and store(data, file)helper *functions* defined inside of the catalog()*task*.
8. Once you have defined these two helper *functions*, you need to write some code within the catalog()*method* to utilize them. Create a *list* titled urls that contains the working URLs in the [00\_urls.txt](https://classroom.emeritus.org/courses/10605/files/3006862/download) file as *strings*. Use a for *loop* to iterate through the urls *list.* First, the *loop* willcall the pull(url)*function* to obtain the data. Then, the *loop* will call the store(data, file)*function* to store the data. Use the pseudocode provided below to define the for *loop*. Note: This is just pseudocode; you will need to complete it with the correct syntax:  
   index = url.rfind('/') + 1  
   #call pull function  
   file = url[index:]  
   #call store function  
   print('pulled: ' + file)  
   print('--- waiting ---')  
   time.sleep(15)  
   Provide a screenshot of the entire catalog()*function*, including the urls *list* and the for *loop* you just implemented.
9. Continue coding within the assignment.py file. Create the second *task* in the Airflow pipeline called combine(). The combine()*task* will combine all of the unstructured data files into one large file. Use the pseudocode below to combine the files. Note: This is just pseudocode; you will need to complete it with the correct syntax:  
   open('combo.txt') as outfile:  
    for file in glob.glob("\*.html"):  
    open(file) as infile:  
    outfile.write(infile.read())  
   Provide a screenshot of the combine()*method* with the correct code to combine the files.
10. Continue coding within the assignment.py file. Create the third *task* *method*, called titles(). This *function* will utilize the [BeautifulSoup4](https://pypi.org/project/beautifulsoup4/)
11. [Links to an external site.](https://pypi.org/project/beautifulsoup4/)
12. *library*,which enables you to *scrape* web pages.  
    The titles()*function* will take no *arguments.* This *function* imports the BeautifulSoup4 *library* and contains one helper *function,* store\_json(), to store the resulting JSON file. Copy the code below under the definition of the titles()*function* to import the BeautifulSoup4 *library* and define the store\_json()*function.*from bs4 import BeautifulSoup  
    def store\_json(data,file):  
     with open(file, 'w', encoding='utf-8') as f:  
     json.dump(data, f, ensure\_ascii=False, indent=4)  
     print('wrote file: ' + file)  
    Next, you will use the BeautifulSoup4 *library* to inspect the HTML page and gather the information within the h3 element of the page where the course titles are found. You will then place the result in a JSON file. Complete the pseudocode for the titles()*function* below to open and read the HTML file generated by the combine()*function.*def titles():  
     from bs4 import BeautifulSoup  
     def store\_json(data,file):  
     with open(file, 'w', encoding='utf-8') as f:  
     json.dump(data, f, ensure\_ascii=False, indent=4)  
     print('wrote file: ' + file)  
     #Open and read the large html file generated by combine()  
      
     #the following replaces new line and carriage return char  
     html = html.replace('\n', ' ').replace('\r', '')  
     #the following create an html parser  
     soup = BeautifulSoup(html, "html.parser")  
     results = soup.find\_all('h3')  
     titles = []  
     # tag inner text  
     for item in results:  
     titles.append(item.text)  
     store\_json(titles, 'titles.json')  
    Provide a screenshot of the completed titles()*method* with the correct code to open and read the HTML file generated by the combine()*function.*
13. Continue coding within the assignment.py file. The next *task method* will remove all punctuation, numbers, and one-character words from the titles.json file. This *task method*, clean(), takes no *arguments* and uses the same helper *function,* store\_json(data,file), that you used in the title()*function* defined in the previous step. Below is the pseudocode for the clean()*method*. Implement any needed commands to complete the pseudocode:  
    def clean():  
     #complete helper function definition below  
     def store\_json(data,file):  
     ..   
     with open(titles.json) as file:  
     titles = json.load(file)  
     # remove punctuation/numbers  
     for index, title in enumerate(titles):  
     punctuation= '''!()-[]{};:'"\,<>./?@#$%^&\*\_~1234567890'''  
     translationTable= str.maketrans("","",punctuation)  
     clean = title.translate(translationTable)  
     titles[index] = clean  
     # remove one character words  
     for index, title in enumerate(titles):  
     clean = ' '.join( [word for word in title.split() if len(word)>1] )  
     titles[index] = clean  
     store\_json(titles, 'titles\_clean.json')  
    Provide a screenshot of the fully implemented clean()*method* with the correct code to remove all punctuation, numbers, and one-character words from the titles.json file.
14. Continue coding within the assignment.py file. The final *task method* you will define is count\_words(). The count\_words()*method* accepts no *arguments* and uses the helper *function* store\_json(data,file)that was used in the previous two *tasks* to save the resulting JSON file. The pseudocode for the count\_words()*method* is below. Implement any needed commands to complete the pseudocode:  
    def count\_words():  
     from collections import Counter  
     def store\_json(data,file):  
     ..  
     with open(titles\_clean.json) as file:  
     titles = json.load(file)  
     words = []  
     # extract words and flatten  
     for title in titles:  
     words.extend(title.split())  
     # count word frequency  
     counts = Counter(words)  
     store\_json(counts, 'words.json')  
    Provide a screenshot of the completed count\_words()*method* with the correct code to call the store\_json(data,file)helper *function*.
15. Continue coding within the assignment.py file. Because this Python file will be run using Airflow, you need to design an Airflow pipeline as the final part to this file. The first step is to define the DAG. Then, define each *task,* from t0 to t5*,* for a total of six *tasks*. The first *task,* t0, will be a bash command to install the BeautifulSoup4 *library* within the Docker environment. The next five *tasks* will each call one Python *function* you have defined in the previous steps.  
    Below, you are provided with some code to get started. The t0 *task* is already defined for you. The t1 *task* is also already defined for you, and it executes the catalog()*function* in your DAG. Complete the code below to define the remaining four *tasks*:  
    with DAG(  
     "assignment",  
     start\_date=days\_ago(1),  
     schedule\_interval="@daily",catchup=False,  
    ) as dag:  
    # INSTALL BS4 BY HAND THEN CALL FUNCTION  
     # ts are tasks  
     t0 = BashOperator(  
     task\_id='task\_zero',  
     bash\_command='pip install beautifulsoup4',  
     retries=2  
     )  
     t1 = PythonOperator(  
     task\_id='task\_one',  
     depends\_on\_past=False,  
     python\_callable=catalog  
     )  
     #define tasks from t2 to t5 below  
      
     t0>>t1>>t2>>t3>>t4>>t5  
    Provide a screenshot of the DAG declaration with all six *tasks,* from t0 to t5, correctly defined.

**Part 2: Code Execution**

1. Execute the command below in a Terminal window to copy the assignment.py file inside the airflow-docker/dags folder so that Airflow can load the Python *tasks* as a DAG.

| cp assignment.py airflow-docker/dags |
| --- |

1. Next, make sure that your Docker application is open. To speed up your computations, select the gear icon on the top right corner of your Docker UI.  
   Select the Resources option in the menu on the left and increase the CPU and the Memory to the maximum values available.  
   Select Apply and Restart to finalize your changes.  
   You are now ready to run the Airflow pipeline. In a Terminal window, navigate to the airflow-docker folder you created within the project-23 folder. Follow the first four steps in [Mini-Lesson 19.2](https://classroom.emeritus.org/courses/10605/pages/mini-lesson-19-dot-2-installing-airflow-in-a-docker-container-30-00) to install Airflow on your machine and initialize the *containers*.  
   Provide a screenshot of your Docker application that shows that your Airflow Docker *container* has been initiated.
2. Navigate to http://localhost:8080/ to see your Airflow session. Access Airflow by entering “airflow” for both the username and password. Select Sign In. Select the assignment.py DAG and switch to the Graph View. Select each *task* and then select Run. Once the DAG has finished running, provide a screenshot of the *task* boxes to show that the DAG ran successfully.
3. The files generated by your DAG that you will need for the final steps of the assignment are located on the worker\_1 airflow *server*. To verify this, in a new Terminal window, use the following bash command to enter the *server*:

| docker exec -it airflow\_docker\_airflow-worker\_1 /bin/bash |
| --- |

1. Use the ls bash command to list the files inside your *container*. If your DAG was successful, you will see a words.jsonfile, which you will use to visualize word frequency.  
   In a new local Terminal window (not the *server*), run the command below to transfer the words.json file to your local machine. You will have to edit this command to include the path to your project-23/code\_visualization folder.

| docker cp airflow\_docker\_airflow-worker\_1:/opt/airflow/words.json YOUR\_PATH\_HERE |
| --- |

1. There is a small difference between the .json file you created and the .js file needed to visualize the code. Place the data from the words.json file into the words.js file within the code\_visualization folder.  
   Provide a screenshot of the words.js file with the data from the words.json file. You may not be able to fit all the data in one screenshot. If so, the first part of the file is sufficient.
2. Next, run the JavaScript visualization code by placing the direct path to the mitcourses\_graph.html file in your web browser.  
   Ex: Navigate to file:///YOUR\_PATH\_HERE/project-23/code\_visualization/mitcourses\_graph.html in your web browser. Provide a screenshot of the visualization produced with the mitcourses\_graph.html file in your web browser.
3. Enhance the bubble chart produced in the previous step using the D3 *library* and the example code provided to you in [Mini-Lesson 23.5](https://classroom.emeritus.org/courses/10605/pages/mini-lesson-23-dot-5-the-data-driven-documents-d3-library-30-00). Modify the code in the d3\_bubble\_chart\_example.html file to display the words saved in the words.json file. Provide a screenshot of your enhanced visualization created with the D3 *library* and the modified example code.

**Submission Instructions:**

Your submission for this project should be a Word document that includes the following screenshots, each labeled for the step that the screenshot represents:

**Part 1: Code Development**

1. Provide a screenshot of the project-23 folder with the code visualization folder, airflow-docker folder, and assignment.py file within it.
2. Provide a screenshot to show that you have imported the DAG object, the *operators*, and all the necessary *task functions* into the assignment.py file.
3. Provide a screenshot of the pull(url)and store(data, file)helper *functions* defined inside of the catalog()*task*.
4. Provide a screenshot of the entire catalog()*function*, including the urls *list* and the for *loop* you just implemented.
5. Provide a screenshot of the combine()*method* with the correct code to combine the files.
6. Provide a screenshot of the completed titles()*method* with the correct code to open and read the HTML file generated by the combine()*function*.
7. Provide a screenshot of the fully implemented clean()*method* with the correct code to remove all punctuation, numbers, and one-character words from the titles.json file.
8. Provide a screenshot of the completed count\_words()*method* with the correct code to call the store\_json(data,file)helper *function*.
9. Provide a screenshot of the DAG declaration with all six *tasks,* from t0 to t5, correctly defined.

**Part 2: Code Execution**

1. Provide a screenshot of your Docker application that shows that your Airflow Docker *container* has been initiated.
2. Provide a screenshot of the *task* boxes to show that the DAG ran successfully.
3. Provide a screenshot of the words.js file with the data from the words.json file. You may not be able to fit all the data in one screenshot. If so, the first part of the file is sufficient.
4. Provide a screenshot of the visualization produced with the mitcourses\_graph.html file in your web browser.
5. Provide a screenshot of your enhanced visualization created with the D3 *library* and the modified example code.