Assignment 7

Goals

The goal of this assignment is to work with the file system, concurrency, and basic data processing in Python.

Instructions

You will be doing your work in Python for this assignment. You may choose to work on this assignment on a hosted environment (e.g. [tiger](https://tiger.cs.niu.edu/jupyter/)) or on your own local installation of Jupyter and Python. You should use Python 3.8 or higher for your work. To use tiger, use the credentials you received. If you work remotely, make sure to download the .py files to turn in. If you choose to work locally, [Anaconda](https://www.anaconda.com/download) is the easiest way to install and manage Python. If you work locally, you may launch Jupyter Lab either from the Navigator application or via the command-line as jupyter-lab. You may need to install some packages for this assignment: aiohttp (or requests) and pandas. Use the Navigator application or the command line conda install pandas requests to install them.

In this assignment, we will be working with files and threads. We will be using energy usage data from New York state, available on the [Utility Energy Registry](https://utilityregistry.org/). I have downloaded a subset of the data, and made it available on the course web site. You will download three zip files from there, extract the archives, load files from the archives, improve the specification of missing data, and store it by year. You will use threading to download and process the data.

Due Date

The assignment is due at 11:59pm on Monday, November 22.

Submission

You should submit the completed notebook file required for this assignment on [Blackboard](https://webcourses.niu.edu/). The filename of the notebook should be a7.ipynb.

Details

Please make sure to follow instructions to receive full credit. Because you will be writing classes and adding to them, you do not need to separate each part of the assignment. Please document any shortcomings with your code. You may put the code for each part into one or more cells. Note that CS 503 Students must use asyncio which is optional for CS 490 students.

0. Name & Z-ID (5 pts)

The first cell of your notebook should be a markdown cell with a line for your name and a line for your Z-ID. If you wish to add other information (the assignment name, a description of the assignment), you may do so after these two lines.

1. Download & Extract Files

There are three zip files posted on the [course web site](http://faculty.cs.niu.edu/~dakoop/cs503-2021fa/a7/) that you will download using python. Their filenames are fg.zip, hij.zip, and tu.zip, signifying the first letters of the counties contained in the files. CSCI 503 students will use [aiohttp](https://docs.aiohttp.org/en/stable/) for this task while CSCI 490 students may use the [requests](https://python-requests.org/) library. (If you have trouble with this part and wish to continue with other parts of the assignment, download the files manually.)

1a. [CSCI 490] Download & Extract Files (20 pts)

To download the files, use the [requests](https://python-requests.org/) library. (This library is installed on tiger, but if you work locally, you may need to install it; conda install requests should work.) Note that each download is a two-step process: first “get”-ing the file and then writing the response to a file. Consult the [documentation](https://docs.python-requests.org/en/master/). After downloading the archives, you should extract all three zip files into the local directory. Remember the DRY principle here. Also, once you download and extract the files into the working directory, rerunning the code to test it may not work as expected because they will already be there. To check for this situation, write code to check (1) if an archive has already been downloaded before downloading it, and (2) if the extracted directory already exists before extracting it. You may also write code to delete those files once to check that your code works, but **be careful** that you do not delete other files in the process!

Hints

* Note that all three archives expand into a directory named data.
* When writing the archive, you will probably want to open the file in binary write mode.
* Use the exists method with os.path or pathlib.Path to check for path existence
* Either [zipfile](https://docs.python.org/3/library/zipfile.html) or [shutil](https://docs.python.org/3/library/shutil.html) will be useful in extracting the archive.
* shutil’s rmtree can be used to delete a directory.

1b. [CSCI 530] Download & Extract Files (30 pts)

To download the files, use the [aiohttp](https://docs.aiohttp.org/en/stable/) library and asyncio to download all the files. (This library is installed on tiger, but if you work locally, you may need to install it; conda install aiohttp should work. You may also need to install nest\_asyncio via pip.) While you can refer to the [example](https://realpython.com/python-concurrency/#asyncio-version) we showed in class, remember that you need to **save** each file locally after downloading it. This should be handled as a separate async coroutine, but note that the code in the method will be synchronous because operating systems generally do not support asynchronous file I/O. After downloading the archives, you should extract all three zip files into the local directory. This may be done **synchronously**! Remember the DRY principle here. Also, once you download and extract the files into the working directory, rerunning the code to test it may not work as expected because they will already be there. To check for this situation, write code to check (1) if an archive has already been downloaded before downloading it, and (2) if the extracted directory already exists before extracting it. You may also write code to delete those files once to check that your code works, but **be careful** that you do not delete other files in the process!

Hints

* You will probably need to run import nest\_asyncio; nest\_asyncio.apply() in the notebook
* Note that all three archives expand into a directory named data.
* When writing the archive, you will probably want to open the file in binary write mode.
* Use the exists method with os.path or pathlib.Path to check for path existence
* Either [zipfile](https://docs.python.org/3/library/zipfile.html) or [shutil](https://docs.python.org/3/library/shutil.html) will be useful in extracting the archive.
* shutil’s rmtree can be used to delete a directory.

2. Find Matching Files (10 pts)

Now, write code to use the unzipped directory to find all files that end with the file extension .csv. Note that some files may reside in subdirectories (or sub-subdirectories). The list of file names should be paths that start with 'data', and there are ten of them.

Hints

* There is more than one way to accomplish this, depending on which libraries you use.
* Remember that your code needs to check subdirectories.

3. Use Threads to Process Files (30 pts)

Finally, we are going to process all the files using threads via [concurrent.futures](https://docs.python.org/3/library/concurrent.futures.html). Use [pandas](https://pandas.pydata.org/) for the data processing steps. (Again, pandas is installed on tiger, but if you work locally, you may need to install it; conda install pandas should work.) Besides reading the files, we will replace -999.0 values with NaN (the missing indicator in pandas) and filter only those records related to electricity. Each comma-separated values (csv) file has the columns county\_name, com\_name, year, month, data\_class, data\_field\_display\_name, unit, value, and number\_of\_accounts. Note that each file also has a header with these column names. For each file, extract those rows with the data\_class electricity, and replace the -999.0 values in the value and number\_of\_accounts columns with NaN. Use a [ThreadPoolExecutor](https://docs.python.org/3/library/concurrent.futures.html" \l "concurrent.futures.ThreadPoolExecutor) to process (read, filter, and replace -999 values) each matching file from Part 2. Take the results from each run, and concatenate them together.

From the single concatenated dataframe, create two dataframes, one for 2019 and one for 2020, and write two files (2019.csv.gz and 2020.csv.gz) which contain only the records from the specified year. The .gz extension means that the file will be compressed using the gzip algorithm to make the output smaller (pandas will do this automatically if you specify that file extension).

While this does not guarantee correct answers, the following code should produce the corresponding results:

df2019 = pd.read\_csv('2019.csv.gz')

df2019.groupby('county\_name').size()

county\_name

Franklin 1134

Fulton 630

Genesee 798

Greene 840

Hamilton 504

Herkimer 1302

Jefferson 1680

Tioga 588

Tompkins 630

Ulster 1050

df2020 = pd.read\_csv('2020.csv.gz')

df2020.groupby('county\_name').size()

county\_name

Franklin 3724

Fulton 2100

Genesee 2660

Greene 2884

Hamilton 1680

Herkimer 4340

Jefferson 5600

Tioga 1960

Tompkins 2100

Ulster 3500

Hints

* Consider writing the processing function first, then add threading.
* [pandas](https://pandas.pydata.org/) has methods to read and write csv files. The index argument can be useful if you don’t want to write the index.
* [pandas](https://pandas.pydata.org/) offers selection via boolean indexing similar to numpy.
* [pandas](https://pandas.pydata.org/) has a replace method that should be useful.
* Use concurrent.futures to run each thread, and consider the [map](https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.Executor.map) function to wait for all results to complete