Documentation

Web Scraping and Storing Data

The task given was to scrape three websites for their data and store it in an appropriate manner using MongoDB or PostgreSQL.

The subtask of web scraping was performed using Python and its versatile libraries while the data collected was stored as comma separated vector files and uploaded to a database by using MongoDB Compass

Web Scraping

Task 1:

The data within the table displayed on the website was scraped using Selenium, a library for web browser automation. Selenium was chosen due its easy to use API, and since it was the first such library I encountered during my research for ways to perform this task. In retrospect, a more lightweight library such as Beautiful Soup would have the done the job just as well.

Dynamic content cannot be accessed directly through HTML tags and classes, and as such, I had to learn how to use 'XPATH's (or CSS selectors) to access dynamically rendered content.

This task proved to be an important step in my process of learning.

Task 2:

This task involved iterating through multiple pages to scrape the data from the table displayed on the website.

The requests_html library was used to perform this task as I was following a tutorial by John Watson Rooney. Requests-html also has an easier to use API than the vanilla requests library according to stackoverflow.com and is lighter than Selenium.

Initially, the CSS selectors were mentioned explicitly and were iterated through to access the data of each cell from the table, but this proved to be a syntactical nightmare as the scraped data is returned as a list and the '.text' field cannot not be used directly on a list. To extract the text from the scraped data, one must first access the '[0]th' element of the returned list, and then add a '.text' field at the end.

ChatGPT was then to optimise the code (which may not be the best method to do so, apologies) and was verified with sources from YouTube (most notably John Watson Rooney), which resulted in some better programming logic (function oriented) and the use of dictionaries to store the row data and map the pagination to the URL.

Parallel processing seemed a bit too complicated and was not implemented despite the relatively large amount of data. Nevertheless, execution happened under one minute, the reasons for which might be high speed internet and the advanced capabilities of my M2 chip.

Task 3:

This task involved scraping data by bypassing certain types of restrictions. Selenium was chosen to scrape the first link as it can operate a 'headless' web driver which means that there are no publicly available user agents that can be tracked and blocked.

However, the other two links could not be scraped as they require simulating logins and involve the usage of cookies, of which I have no knowledge.

Database:

A single database was created for the tasks called as 'Task' with separate collections for each dataset generated from the previous tasks.

Dependencies:

1) Selenium: pip install selenium
2) Pandas: conda install pandas (miniconda for Mac, anaconda for other systems)
3) Requests_html: pip install pipenv -> pipenv install requests-html -> pip install lxml-html-clean