CSCI 0112 - Computing Foundations: Program Organization - Final Project

Ashley Chon

Goal

My goal for the final project was to do an extension of Project 3 by scraping data from Indeed, a website that is often used by students to find internships and job opportunities. Specifically, I wanted to make it easier for engineering students, like myself, so see internship listings in a more cohesive and clean way without all of the unnecessary, extra information.

Originally, I was planning on writing a few queries over my scraped data, similarly to Project 3. I planned for my scraper to be a function that takes in a dictionary where the keys are names of concentrations and the values are BeautifulSoup objects. The scraper would use BeautifulSoup methods to transform the data into a format that I would be able to query in my query functions. From each internship listing, I was planning on scraping the company name, location, position title, company rating, and the description. Listings that were missing any of those pieces of information would be skipped over. For the query functions, I was planning on writing functions to find the average company ratings across all of the concentrations, find the most commonly occurring "interesting" word in the listing for a given concentration (similar to Project 3), find the company with the most internship opportunities, and find locations that have the most internship opportunities.

After finishing up the internship recruiting process myself, I realized that the information the query functions I was originally going to write would provide as much help as I thought it would. For an engineering student looking for an internship, knowing things like the most commonly occurring "interesting" word in a listing would not be information that would be practical or helpful in the recruiting process. As a result, I changed my goal of the project to produce a scraper that would output a CSV file that when open in applications such as Numbers, Excel, and Google Sheets, can be easily read in a table format with streamlined information. I also still wrote query functions, but instead of the ones I was originally intending, I wrote functions that would find the company and location with the most listings for a given concentration. The last function I implemented found the concentration that had the most internship listings.

Implementation

In order to implement this project, I first had to import several libraries such as requests, csv, datetime, and BeautifulSoup. Overall, I implemented four different functions to accomplish my goal.

The first function, `indeed_get_url`, was used to generate a url from the input, which was a string representing the concentration that the user wanted to find internship listings for. I used the url template format used in Project 3, where `{}` was used to insert the proper value into the url based on the input. Before using .format to output the final url, I made sure to replace any spaces with `"%20"` and make all of the characters lowercase, using `.lower()`, based on patterns I found on the actual Indeed website. To make sure that the job type was an internship, the url included `"jt=internship"`.

The second function, `get_listing_data`, extracts the data from a single internship listing and returns a list that includes the job title, company name, job location, post date, current date, job summary, salary, and job url. To extract this data, I used various BeautifulSoup methods (`find, get`). To get the current date, I used the `datetime.today()` method.

The third function, `get_csv`, scrapes all of the data for a given concentration input and produces a CSV file. It uses requests and BeautifulSoup methods to get the listings for each concentration. It repeats for every subsequent page of results until there are no more remaining. Once it is determined that there are no more results remaining, the function writes a CSV file with the information that was extracted.

The fourth function, `scrape_data_into_dictionary`, scrapes all of the data for a list of concentrations and returns a dictionary with the keys as concentrations and values as a list of internship listings (which are also lists). Similar to the previous function, it uses requests and BeautifulSoup methods. It appends all of the listings to the list in the values of the dictionary.

The fifth function, `scrape_brown_concentrations`, a list of concentrations to produce CSV files for all of them.

The first query function, `company_with_most_listings`, takes in a dictionary of data and a string for the concentration to return the company that offers the most internships for a given concentration. I used a for loop to iterate through the lists inside of the list in the values of the dictionary of data and kept track of the count for the number of listings. I returned the concentration that had the highest count at the end.

The second query function, `state_with_most_listings`, takes in a dictionary of data and a string for the concentration to return the U.S. state that offers the most internships for a given concentration. I used a similar approach to the previous query function, but had to extract the state from the location information by calling the last two characters of the location string.

The last query function, `concentration_with_most_listings`, takes in a dictionary of data and returns the concentration that offers the most internship positions. To implement this function, I initialized a `max_len` and `max_key` value with a 0 and empty string, respectively. Then, I used a for loop to iterate through the dictionary. I set the length of the values of the key as a variable `cur_len` and used an if statement to see if `cur_len` was larger than `max_len`. If it was, then, the `max_key` would be set as that key and the `max_len` would be set equal to `cur_len`. Once the for loop was done iterating, the function returned the `max_key`.

Results

After running my Python file and calling `scrape_brown_concentrations` with `CONCENTRATIONS` (a list of different Brown engineering concentrations) as the input, I ended up with 7 different CSV files and exported them as Google Sheet files, both of which can be found in this folder. Since I couldn't test using pytest for the functions that returned CSV files, I went through all of the Google Sheet files and manually checked to see if the data that was extracted made logical sense. For example, the biomedical engineering internship results should be medicine/healthcare focused while most of the computer engineering internship listings should be software engineering intern roles. I also knew that the number of listings for concentrations such as computer engineering would be much higher than concentrations like chemical engineering. My results showed 947 results for computer science internships and 292 results for chemical engineering internships. When I looked through all of the files, the listings looked extremely reasonable and similar to what I saw when I searched on the actual website.

When I ran the first query function, `company_with_most_listings`, I found that for biomedical and chemical engineering, the company that offered the most internships was Thermo Fisher Scientific, for computer and electrical engineering it was Facebook, for environmental engineering it was Lehigh Hanson, for materials science and engineering it was Pacific Northwest National Laboratory, and for mechanical engineering it was Lockheed Martin Corporation.

When I ran the second query function, `state_with_most_listings`, I found that the state with the most listings for each concentration was California for all of the 7 different concentrations. This made sense, because most of the technology and engineering industry is located in Silicon Valley.

Lastly, when I ran the third query function, `concentration_with_most_listings`, I found that mechanical engineering and computer engineering had the most internship opportunities. I ran the function multiple times and every time, it would alternate between one of those two concentrations, depending on the live results that were scraped. For reference, for one of the times I ran the scraper, biomedical engineering had 40 listings, chemical engineering had 292, computer engineering had 947, electrical engineering had 414, environmental engineering had 396, materials science and engineering had 48, and mechanical engineering had 868. This also made sense because mechanical engineering is the most popular engineering concentration, even at Brown, and computer engineering allows you to also pursue software engineering internships, which there are a lot of.

To test these three query functions, I created a new file, `test_scraper.py`, and imported `scraper.py` and `pytest`. For the first query function, I tested two sample data sets, one with more normal data, and one with an empty dictionary. I tested a normal example, ties for companies, if a concentration doesn't have any internship listings, if the concentration isn't listed in the dictionary of data, and if the dictionary is empty. The tie returns the first listed company, the empty list as values returns a `ValueError`, the non-existent concentration returns a `KeyError`, and the empty dictionary returns a `KeyError` as well. For the second query function, I also tested two sample data sets, one with more normal data, and one with an empty dictionary. I tested a normal example, ties for states, if a concentration doesn't have any internship listings, if the concentration isn't listed in the dictionary of data, and if the dictionary is empty. The tie returns the first listed state, the empty list as values returns a `ValueError`, the non-existent concentration returns a `KeyError`, and the empty dictionary returns a `KeyError` as well. For the third query function, I tested four sample data sets, two with normal data, one empty dictionary, and a dictionary full of keys that had empty lists as values. I tested a normal example, ties, if a dictionary is empty, and if all of the values are empty. The tie returns the first listed concentration and both the empty value and dictionary return an empty string. These tests allowed me to catch small errors here and there. Once all of those were taken care of, the tests passed and I was able to know that the query functions worked as intended.

Challenges

Unlike Project 3, I wanted to include listings that were missing some of the information, so I had to figure out how to deal with those. After trial and error, I ended up using try and except blocks to set the values as empty strings if the data couldn't be extracted or found. I also had to figure out a way to scrape the data from all of the results and not just the first page of results that only contained 15 listings. With that came the issue of having to figure out how to tell the function when to stop scraping the data and when to start writing the CSV file. To tackle both of these issues, I ended up using a while loop (while True) along with a try and except block that would break if there were no more pages. Once it broke out of the while loop, it would continue to step through the function and write the CSV file with the data it had just scraped. Lastly, I was also not familiar with how to write csv files with the data that I had to scrape, so I had to do some research on what methods I should be using, such as `.writer`, `.writerow`, and `.writerows`.

For the query functions, I had to make sure that what I was returning from the scraping was using a data structure that would allow me to query over. Originally, when only scraping for the sake of writing CSV files, I was returning the listings as ('JobTitle', 'Company', 'Location', 'PostDate', 'ExtractDate', 'Summary', 'Salary', 'JobUrl'). I realized that I need to return the listings as lists in order to query over them.

Future Work

If I had as much time as I wanted to keep working on this project, I would want to figure out a way to generate the CSV files faster by optimizing my code. Currently, it takes quite a while to run my code, so I would want to work towards improving the run time. I would also want to implement functions that would allow the user to filter the internship listings by location, salary, post date, etc. I know this would be useful for many students in the recruiting process because there are many situations where you are only able to work in certain locations or for certain salaries.

Conclusion

Creating a web scraping implementation from scratch on a website that I was familiar with was incredibly eye opening. It allowed me to see the impact that web scraping could have on the industry and how helpful it could be. I learned more about using BeautifulSoup objects and web scraping in general, specifically implementing CSV file writing.