MATH 300 Final: Web Scrapping Job Listings

Justin Prachar

Due: August 2nd, 2020

1 Background

The python script "FindMeJobs.py" is a web scrapper designed to function with multiple job searching websites. Web scrapping is a powerful tool allowing for any website to be turned into a database. Furthermore, this tool automated as a service, giving its user a competitive advantage in job search due to increased information.

> We are going through the process where software will automate software, automation will automate automation. - Mark Cuban

Web scrapping is done by using the URL of a website as a query. There are parameters listed inside search type URL's. For example:

> https://www.monster.com/jobs/search/?q=Process-Engineerwhere=pullman
> https://www.monster.com/jobs/search/?q=@Job-Titlewhere=@Location

By sending a request for this query a list of record sets (job postings) will be returned. Inside of each of those record sets individual pieces of data can be extracted (job title, company, location, application URL). Useful analysis can then be run on these data sets. In the example to be discussed our analysis covers how likely a job search is to return the actual job the user has requested. The basic tutorial used for information on extracting data from the HTML page came from Real Python[3]:

2 Tutorial of Application Setup

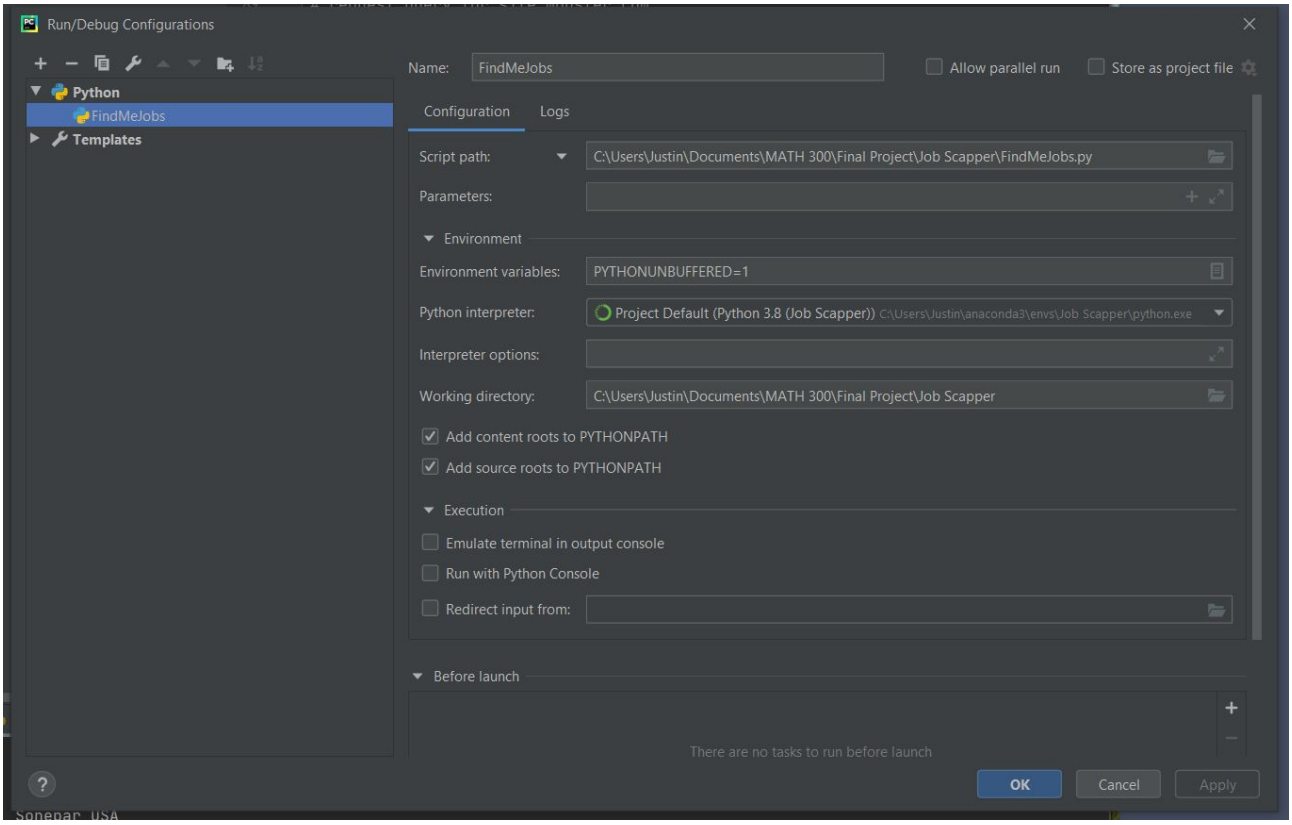
2.1 Python File User Setup

Inside the .py file, after the imports of needed modules: requests[4], bs4 (BeautifulSoup)[5], csv[1], and pandas[6], there will be the opening and creating of the comma-separated-values (CSV) file. The fields will be created as myFieldNames; This should remain static. The job search title and location should then be set by the user and the query function is called with a parameter of a job search website (currently only 'monster' is supported, ZipRecruiter is blocked by Captcha). Upon function call, the program will come to a selection block and the key results used by BeautifulSoup will be set. Each website uses slightly different key names and locations for attributes of their search results. The programmer must update these variables based on the HTML source code to have added functionality of new websites. Outside of these variables, the function should remain untouched. For data analysis the user can then input the main and supporting substrings they would like to find from all the title results. These sub strings should be a two-word string with the remaining two being the different single word compositions of the two-word string.

2.2 Python File Algorithm and Data Analysis

The Python file is complied using Andaconda[2]. Configuration below:

The CSV is created with the header row. This is done in write mode so all previously stored data will be overwritten. This is designed because the jobs list should be kept updated. The search keys and locations are then set and a query request is made. The resulting page is then stored into a soup variable. The keys



set up previously are then used to extract the 4 data values aforementioned. This loop of extracting data values is done for as many resulting elements are returned from the initial URL query. This number can be set to a user defined value by limiting the loop (decrease number of elements), or by editing the URL so n pages are shown (increase number of elements; monster allows for larger page numbers than actual). The data is then written to the CSV "jobResults.csv" and the file is closed.

Next, the data analysis is performed. As described in the previous subsection the user should have defined three sub strings to found in the job titles list. The total number of job listings is compared (as a percent) to the number of job listings that contained the entire substring, and parts of the substring (compositions). This data is then printed to the console.

2.3 Application Automation

The task automatically finding and storing job search results has been accomplished by the Python file. However, as Mark Cuban informed us in the introduction if we don't automate the automation, then we're getting replaced. In order for this task to be truly automated, it will need to be able to execute itself with zero integration. At this point the application's interaction count is at 1 (click run), but that is far too many. To have this task run every day at 5AM providing the user with all the information needed to apply to a massive amount of job listings, the task should be set up as a service. There are several free ways this can be done, in this case the in-house Windows Task Scheduler administrative tool is used. Open Windows Task Scheduler and create a new task. Set a trigger for the program to launch; time of day repeated daily is good because we will also tell the computer to wake at that time. Next, the action should be set. Program should be the file path string for the python interpreter; mine is located in anaconda. The argument should be the path to the .py file. Remember to check the box next to "wake the computer to run this task". This step of automating the automation means a brand new csv file will be created housing the newest job listings everyday before even waking.

3 Conclusion

When the test parameters used we location of Seattle, and the job title of process engineer were passed into the query function with using monster.com's first 3 pages 76 results were returned. Of those 76 only 44 (57.89%) of them included the jobs returned included the title "Process Engineer", an additional 20 (26.32%) jobs had only "Engineer" in the title, an additional 6 (7.89%) jobs had only "Process" in the title. This left 6 (7.89%) jobs containing none of the substrings. From this data we can conclude that monster.com is adequate at searching for specific jobs. They will return far more generic type named jobs like engineer than they will from specific titles like process. From this data set 8% of all jobs returned has no relevancy to the query from the title alone. This is probably done because it is better for a search engineer to return too many rather than too little amount of results. Otherwise, I think the more value able portion of this project is in the constantly updating links with no effort from the user. The next script written should be one to fill out application forms.

References

[1] *CSV File Reading and Writing*, 2020.

[2] Inc Anaconda. *Anaconda Individual Edition*, 2020.

[3] Martin Breuss. Beautiful soup: Build a web scraper with python, 2019. `https://realpython.com/beautiful-soup-web-scraper-python`.

[4] Kenneth Reitz. *Requests: HTTP for Humans*, 2020.

[5] Leonard Richardson. *Beautiful Soup Documentation*, 2020.

[6] the pandas development team. *pandas documentation*, 2020.