ECON 8320 Semester Project Write Up

Spring 2023 - NIL Project

Brett Thompson

For my ECON 8320 semester project I selected to collect NIL deal data. The first step in working on this project was to determine which website or websites I would collect the data from. On the project homepage there were two different websites that were given as potential sources of data. The first was nilcollegeathletes.com, which after reviewing the website seemed to have quite a few NIL deals that were at least a year old. The second was on3.com which had a page that was updated frequently with new NIL deals. For this reason, I decided to focus on scraping data from on3.com.

When opening the NIL deals page on on3.com it starts with giving the latest 25 deals to be added. By scrolling down to the bottom of the page a load more button can be pressed to load the next 25 most recent deals. To start, I decided to first focus on gathering the data from the most recent 25 deals and then later return to get more data.

In order to refresh on web scraping I returned to lesson 7 and reviewed the class notes and code. In the brick set homework we looped through each Lego brick set for a certain year and scraped various data before looping to the next brick set. I decided to do the same for this project. To do this I needed to import Beautiful Soup, Numpy, and Requests just like the brick set homework. I would later import Pandas as well to turn my data matrix into a Pandas data frame and remove Requests because I would not need it when using Selenium. I will explain this later when I talk about Selenium. As in the brick set homework I used requests.get with the on3.com url to get the html from the deals page. After that I passed the html into Beautiful Soup to be parsed.

After the html was parsed, I was able to get started on looping through the scraped data. To begin this process I needed to figure out which html tags were holding each deal's information. After using the inspect tool on my web browser I was able to determine that the 'li' tags held all of the deals. Knowing this information I could go through all of the deals on the page and extract information using a for loop and the .find_all method.

However, before extracting any data inside the for loop, an empty list called 'row' was created. This is just like in the web scraping homework because there needs to be an empty list where the data for each deal will get appended. Once the loop finishes scraping all of the desired data into the list 'row', it is appended into an empty list called 'nildata'. The for loop will then move on to the next deal with an empty 'row' and repeat the process. At the end of the for loop 'nildata' will be a matrix with all of the deals as rows and the deal information as columns.

The data that I decided to extract from the webpage using the for loop was the athlete's name, their on3 estimated NIL value, their year in school, their school, their position, the company/client who the deal is with, and the collective that the athlete had involvement with. In

order to do this I used the .find method to select the data by indicating the desired html tag and class (every instance except position). This data was then appended on to the 'row' list.

I also wanted to be able to get the athletes' respective sports into the data frame somehow. However, there was not a good way to scrape this data from each deal's html. This was one of the struggles I had to deal with on this project. In order to get the sport data I decided to determine sport based on the position data that I could scrape. The majority of the position abbreviations that I scraped were unique to their respective sport, so I decided to create lists for each sport with their respective position abbreviations in the list. I then added if and elif statements to my for loop that determined if the position scraped was in a certain sports list, that sport would be added to the 'row' list as the athlete's sport. For example, if an athlete's position was IOL, which corresponds to interior offensive lineman in football, that athlete's sport would be assigned as football.

However, there was one drawback to doing this which was that some of the position abbreviations that on3.com gives are the same across multiple sports. An example of this would be the position abbreviation 'S', which was shown on the website as the abbreviation for safety (football), sprinter (track & field), swimmer (swimming), and setter (volleyball). I was not able to come up with a novel solution for this issue, so I decided not to assign sports for these common abbreviations at all in order to not mislabel an athlete's sport.

After I had looped through the athlete deals and appended the information to my nildata matrix I was ready to convert it into a Pandas data frame. This was done in a similar fashion as we had done with all of the homework this semester. I used the pd.DataFrame method and added the desired column names. Once the data was in a data frame I could then write the data to a new .csv file by using the .to_csv method.

Once I had this small data frame completed I then started to think about how I could increase the size of my data frame and add the other deals on the webpage that are accessed by clicking the 'Load More' button at the bottom of the page. I went back to the web scraping notes to determine how to do this, but noticed one big problem. The 'Load More' button did not direct you to a new page, but rather kept the same page open and loaded 25 more athlete deals. After these 25 deals were loaded, another 'Load More' button appeared at the bottom of the page which allowed you to add 25 more deals. This 'Load More' button was able to be pressed until there was no more data available.

Since I would be loading more data onto the same page I could not follow the same direction as the brick set homework. This is where Selenium comes into the picture. Using Selenium I could run a driver which automates the clicking of the 'Load More' button however many times I wanted it to and then I could take the html from the web page, put that into Beautiful Soup, and then run the for loop to get the larger data frame.

In order to use Selenium, I needed to download the Chrome webdriver that would allow me to automate the clicking of the 'Load More' button. I downloaded this executable file to a local folder and pointed to that path. Once Selenium was imported and the Chrome webdriver downloaded, it was possible to set up the code to click the 'Load More' button.

This led to my next challenge which was determining how to select the ‘Load More’ button to be clicked. When I first tried to grab the button’s tag and class and use .click() to click the button I was being brought back to the on3 homepage. After some research I found that there were many buttons on the page with the same tag and class. I had been grabbing the first instance of this button and clicking it. Doing a little bit of trial and error I was able to determine that the ‘Load More’ button was actually the last instance of this button on the page and would continue to be as I clicked it. This allowed me to put all of the instances of this button into a list called click_button and to select the last by using click_button[-1] to grab the last instance.

After determining how to click the ‘Load More’ button using Selenium I could then use a for loop to continually click this button for however many times desired. However, I quickly found out that in order to do this clicking I needed to wait for the page to load before clicking again. This was done by importing the time module and adding a 5 second delay between the clicks.

After loading more results the html from the webpage, I stored the html into a variable using the .page_source method and then passed it into Beautiful Soup. Since this .page_source method was grabbing the html, I did not need to use the get.requests method that I talked about earlier.

This encapsulates the process behind creating this NIL data frame. The code should be able to be repeated on another computer as long as the user has the Chrome webdriver and the correct path to it called out. The user would also need to adjust their file path that they would like to print the .csv file to.

Looking back, there is one thing that I would do differently if I had the time. This would be to use Selenium to go into each deal individually to grab information. This would have allowed me to grab more information about the athlete. However, this would have also caused my code to take very long to run since I added a time delay in the Selenium code. I also would not have been able to do one loop through all the data at once, but would have had to append the data each time I would go into an individual deal.

Overall, this has been an enjoyable project and I have used tools learned in class and outside of class to complete it.