Ryan Gilmore, Lo-Badal Burch, and Aidan Sullivan

Group 15

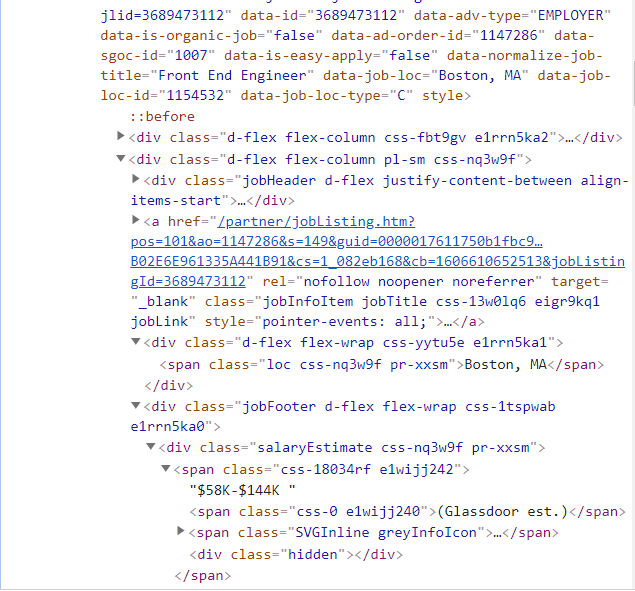
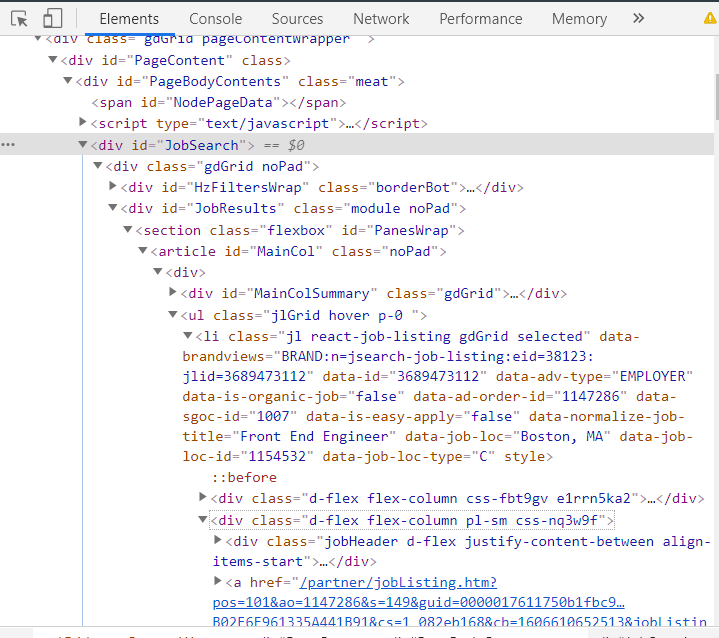
***Indeed.com Web Scraping***

**Introduction**

As the fall semester ends, students must begin searching for a summer Co-op. However, with Coronavirus negatively affecting the current job market, it is more important than ever for students to have as many job searches as possible for the best chance of obtaining a desirable Co-Op. Our team thought it would be a great idea to write a web scraping program for students to be able to search for current and local job postings. This paper will describe how our web scraping program was implemented and how it is able to scrape relevant data from Indeed.com.

What makes this project interesting is that every time a user requests a web page, the server responds by sending the relevant information back to the user. Python scripts can execute thousands of requests a second. If not coded correctly, a web scraping program could potentially overload the site and flood the hosts bandwidth. This could end up costing the site owner thousands of dollars and time to fix the site. Our program is small enough so we don’t have to worry about this issue, but it is important to recognize how powerful a web scraping program can be and what impacts it could have.

Our project is broken up into two main components: the flow of the HTTP request and response, and the parse of the resulting HTML file. The website we use is largely dependent on the websites protocols around bots and the html format. We decided to practice on a website hosting poker statistics with few bot restrictions. Originally, we were planning on scrapping Glassdoor for our project. However, we found that Glassdoor’s HTML is very difficult to read and to locate specific data required diving through a lot of different tags. When looking at Indeed’s HTML, it was much easier to follow.



Glassdoor Indeed

**Relevant Work**

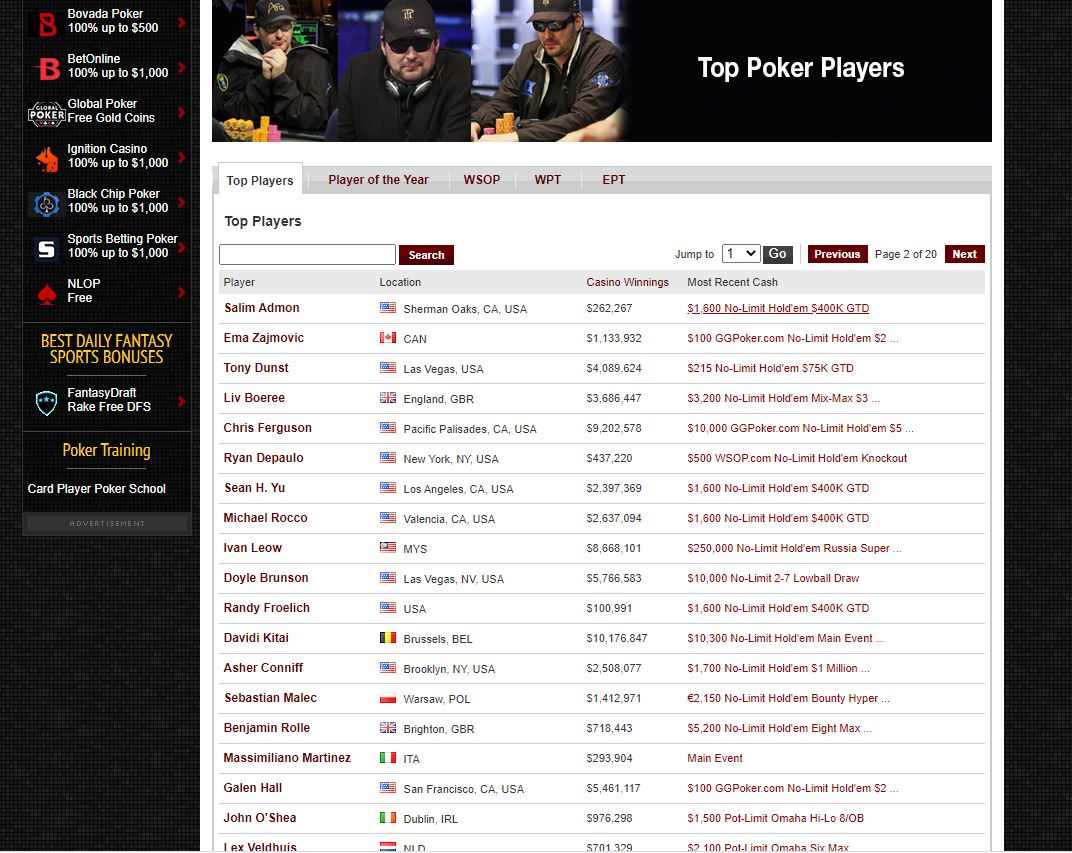
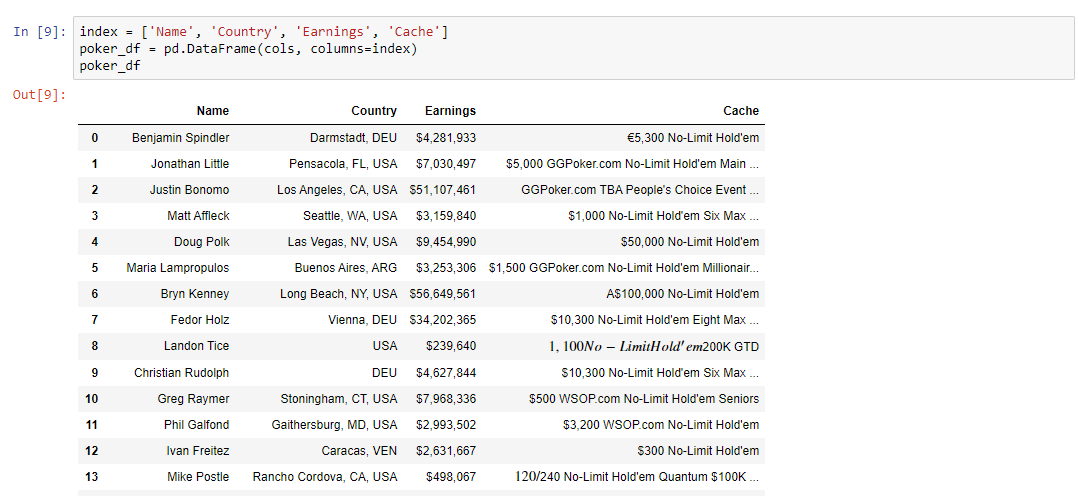
There is a plethora of sought-after data to be found across many sites all over the web. Oftentimes, the website does not have the best parameters to sort through this data, or better yet, order this data in a form that can be graphed or otherwise. Unless the website has a price tracker, or performance graph, etc. built into the website, you would need to copy the information manually into a spreadsheet. This would take any superhuman a lengthy amount of time to complete. Luckily web scraping in the form of applications such as Octoparse or Parsehub have been around for years now, built to solve this exact need.

Both of these tools do essentially the same thing: analyze the source code of a website and organize its relevant data into a chart. Octoparse, for example, is great at scraping Amazon for an item of your choosing. It then allows you to organize relevant information, such as prices, review counts, ratings, etc. This free reign over which site you scrape, as well as what parameter you search for was something we implemented early on in our final project. By allowing the user to edit the requested indeed.com link, they can receive data on any job position and any location. We organized our web scraped data in the form of a spreadsheet, as nearly all web scraping applications have done since their inception.

A relatively new alternative to html parsing is using machine learning and image descrimination. Parsing requires a fair bit of human intervention; html formatting is not consistent across all web pages so scraping programs need to be tailored to specific websites. This problem is addressed through a deep learning technique called text classification. While deep learning is outside the scope of our class, the basic principle is that the bot is trained on large text files with the target output of identifying certain words. With these bots, websites can be scraped without the need for python scripts.

**Description**

Our first implementation of a web scraper extracted poker winnings data from a table of top poker players (left). This is rather simple, as we are essentially scraping a copy of the table that already exists on the website (right); however, we will go about scraping indeed.com the same way, the only difference being that we will decide what data is relevant.



Our web scraping program will be using the BeautifulSoup Python library. BeautifulSoup is a Python package for parsing HTML and XML documents. Some features that make BeautifulSoup powerful include:

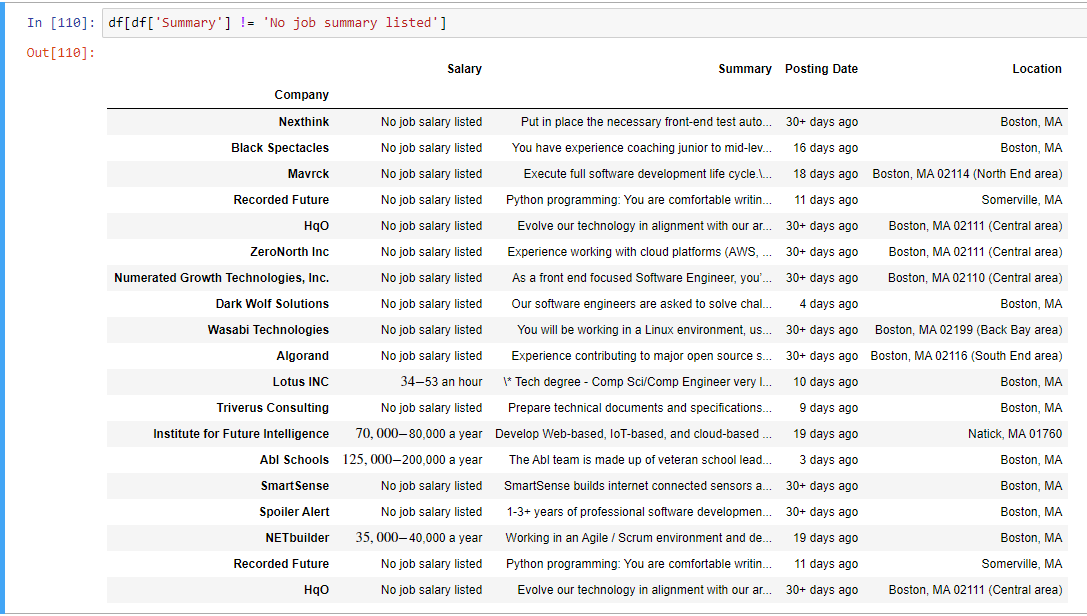
1. Provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for parsing exactly what you need.

2. BeautifulSoup automatically converts incoming documents to Unicode and outgoing documents to UTF-8.

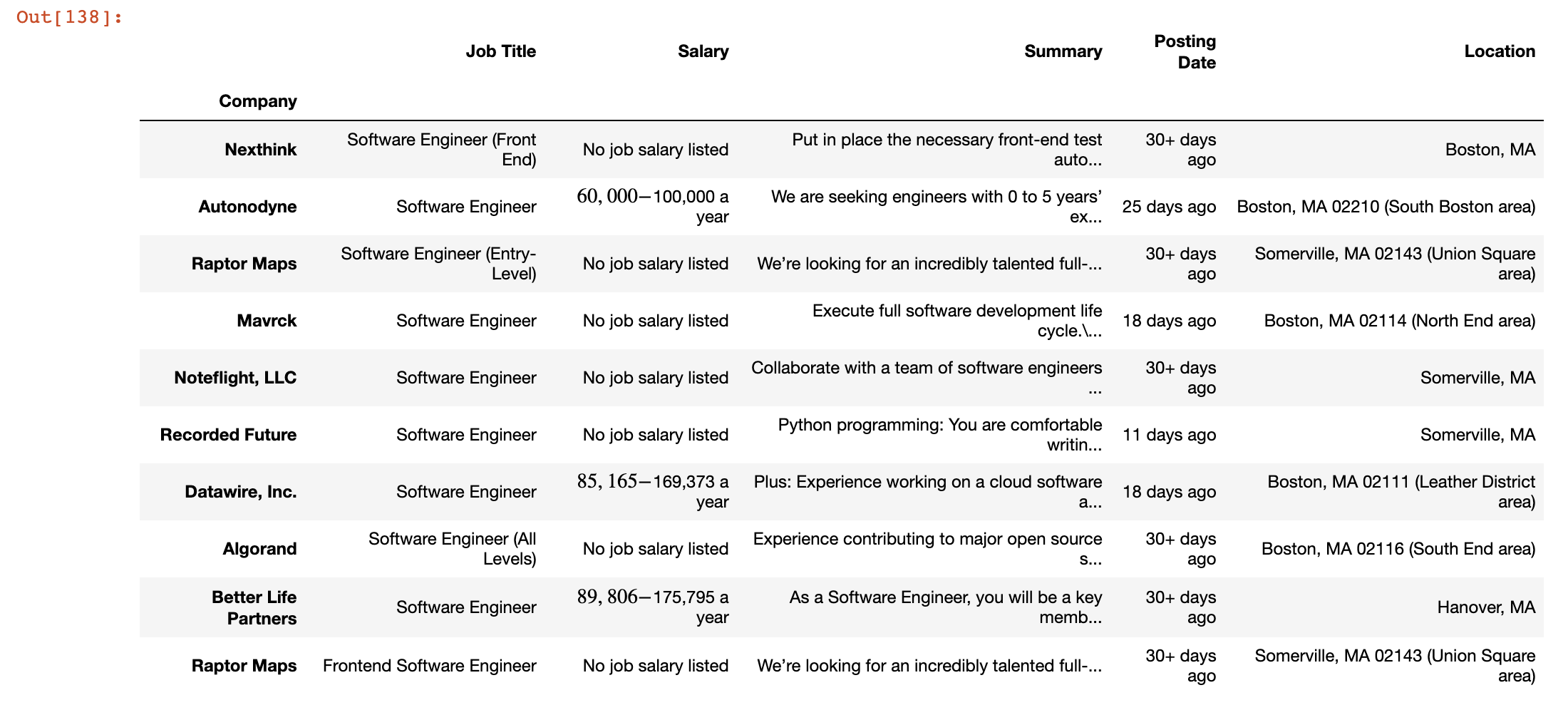
For this project we will utilize the package for parsing the HTML of Indeed.com. The goal of the program is to collect and process job listing information. Specifically, job title, company name, job description, postdate, location, and salary. The program will request the web server using our provided URL. The server will then respond with HTML files representing the user side of the website using hypertext transfer protocol. The program will then parse the HTML files, select relevant pieces of data, and write the data to a text file.

To start, we imported requests to get the get request from the server to acquire the HTML. Along with importing BeautifulSoup to parse the HTML and to extract the data we want. Then we took the URL and pasted it to the code as a string and assigned it to a template variable. We deleted the job and location text and assigned it with curly braces. This allows us to be able to create a string template that we can use to insert whatever job position or location we want. We created a function called get\_url with the arguments position and location. We used the new get\_url function to assign the return value to a URL variable. This allows our program the flexibility to search whatever job the user wants along with any location. Next we sent a get request to the website and extracted the raw HTML. To do this we used the get function of the request library, which returns a response object. We passed in the URL as the argument. response = requests.get(url) This will send a request to the site and a response will be sent back. We used the findAll method to find all div tags holding the value ‘jobsearch-Serp’. This will return a list of all job elements that meet the search criteria. We used this basis to scrape the job description, salary, location, and post date.

**Results and Evaluation**



*Filtered Results*



*Filtered Results w/ Job Title*

Judging from the results (above) our program is quite successful at scraping through indeed.com and returning the data asked of it. The data is from a few pages of indeed.com searching for software engineer in Boston MA. While our program does not have the extra features of the big web scrapes, like Octoparse or Parsehub, it returns the relevant information in spreadsheet form, and within a fraction of a second.

One large problem we faced when tackling indeed.com, was that a lot of job postings would fail to get scraped causing an AttributeError. We used a try except block to catch the Attribute errors, for when there were no listings and record the data as “not listed”. We did come across the issue of data not appearing even though the information was listed on the website. This may be due to the fact that indeed.com, like Glassdoor and any modern website, is more complicated to scrape then, for example, a database or the simple poker table from our first web scraper.

If we were to start again knowing this, we would most likely focus on another site to scrape. A good candidate for such a site would be one that has easily accessible data (no embedded urls, etc.), while still filling a need. Building out the poker web scraper to other sites in order to compile a large amount of data would be a possible avenue.

**Conclusion**

BeautifulSoup is packed with useful functionality for parsing HTML data. Its documentation is comprehensive and relatively user-friendly. We learned how to scrape data from the Web using Python, requests, and BeautifulSoup. This project also gained us insight in inspecting the HTML structure of a website using our browser’s developer tools. If we had more time, one thing we could have done is make a GUI interface. The interface would have the option to display more jobs relating to computer science and network programming. Not specifically just software engineering, as we did for our project for example. Overall, we were happy with the outcome of our project.

**References**

Author: Brendan Martin Founder of LearnDataSci, and Brendan Martin Founder of LearnDataSciAuthor and Editor at LearnDataSci. Python development and data science consultant. “Ultimate Guide to Web Scraping with Python Part 1: Requests and BeautifulSoup.” *Learn Data Science - Tutorials, Books, Courses, and More*, www.learndatasci.com/tutorials/ultimate-guide-web-scraping-w-python-requests-and-BeautifulSoup/.

“Beautiful Soup Documentation¶.” *Beautiful Soup Documentation - Beautiful Soup 4.9.0 Documentation*, www.crummy.com/software/BeautifulSoup/bs4/doc/.

“Technology.” *Software Engineer Jobs, Employment | Indeed.com*, www.indeed.com/jobs?q=software+engineer.