Webscrapping and Shiny application

Barbier–Darnal Joseph, Lacoste Victor, Judic Erwan and Komla Djodji Adayeke

Webscrapping

Context

For this project, we decided to scrape 4 sites: LinkedIn, JobTeaser, HelloWork and WelcomeToTheJungle. The sites have quite different page architectures, which led us to create a scraper (Python class) for each site, contained in the `src` directory.

All source code is available on our *GitHub repository*.

Tools

All the work is done using `pandas==2.2.0`, `selenium==4.17.2`, `requests==2.31.0` and `bs4==4.12.3`. Those are listed in the `requirements.txt` file.

Scraping

LinkedIn

The `LinkedIn()` class is designed for scraping job listings from LinkedIn using Selenium for web automation. It encapsulates functionality for initializing a web driver, managing browser interactions such as logging in, accepting cookies, entering search criteria (keywords and location), navigating through job listings, and extracting job details.

Difficulty

- The main difficulty in scrapping these jobs is the fact that the pages are dynamically generated and therefore require the simulation of user behavior via the driver to access the job.
- Also, multiple elements are not consistent between job offers, which makes it difficult to extract the data, such as salary, contract type, etc.
- Another problem is that, for some unknown reason, the date (or rather the element in which it is found) on which the job offer was published is sometimes found.
- After login, LinkedIn can put a captcha to verify that the user is not a robot. Fortunaltey, this behavior seems more of an exception than a rule.
- Finally, the number of jobs found is highly variable from one search to another. In the best scenario, we can find more than 1000 jobs, but in average we are close to between 300 and 600 jobs.

JobTeaser

The `JobTeaser()` class is crafted for automating the process of scraping job listings from the JobTeaser website using Selenium. This class initializes with a Chrome webdriver, targets a specific URL for job offers, and sets a limit on the number of pages to visit for scraping. It encompasses methods to handle cookie acceptance, job search by keyword, dynamic page navigation to load job listings, and extraction of job details such as titles, companies, contract types, locations, descriptions, and posting dates from individual job links.

Login to the site

Access the Jobteaser 'offers' page with our selenium driver, then after accepting cookies, enter the keyword 'Data Science' in the search bar.

Retrieving job links

We store the links of all the offers appearing on the first page, then make the driver go to the next page, iterating until we've reached the last accessible page (the 50th). Difficulty encountered for this step: To change page, the driver clicks on a button at the bottom of the page, which must appear on the screen to be clickable. The driver therefore had to scroll to access the button.

Retrieving offer information

Open a new driver for each offer, accessing the page from the link retrieved in step 2. We then retrieve all the information on the page and close the driver.

Difficulty encountered with this step: When we tried to use the same driver to open all the pages containing the offers, even with long waiting times, or by opening new tabs, we were immediately blocked by the site. So we thought we'd have to log in to an account to be less restricted in our scraping, but that didn't prevent the site from blocking us. This is why we open a new driver for each offer, which wastes a lot of time.

Data formatting

This step is actually included in step 3, but as far as data formatting is concerned, just to mention the information badly inconsistent with the other sites we scrapped:

- Publication date: Conversion from 'Published January 24' to '2024/24/01' format
- Type of contract time: Conversion from a format 'Internship - 4 to 6 months' to 'Internship' for contract type and '4 to 6 months' for contract time

Welcome To The Jungle

The `JungleScraper()` class is designed to automate the process of scraping job offers from the Welcome to the Jungle website. It utilizes Selenium with a webdriver to navigate the site, accept cookies, conduct job searches based on specific keywords, and collect job offer URLs across multiple pages. Key functionalities include searching for jobs using keywords, navigating through job offer pages, and scraping detailed information from each job offer's URL.

Implemented strategy

In order to efficiently retrieve the information contained in each of the job offers, a first function retrieves the URLs of these offers, and another function retrieves the information present on these pages. The majority of the information retrieved comes from the tags present at the beginning of the page, which are rather practical because they are all stored in the same classes.

Challenges

A major difficulty was to navigate efficiently between all the results pages, the code had to be able to move forward in the pages in order to retrieve new job offers, so the driver had to find the "next page" button and click on it. However, the "next page" and "previous page" buttons are stored in the same object class, except for the first page where the “previous page” button doesn’t exist. It took some time to realize this, so the solution chosen was to click on the first button for the first page, then click on the second for all the other pages.

HelloWork

The `HelloWork()` class is a comprehensive tool designed for scraping job listings from the HelloWork platform. It leverages the Selenium WebDriver to navigate the website, perform job searches based on keywords, and systematically collect job listing URLs for further processing. The class incorporates a variety of methods to manage browser interactions, such as loading pages, accepting cookies, searching for jobs, scrolling through pages, and retrieving job links from search results.

- Although the site is dynamic, it offers the possibility of accessing a search page by modifying the entry URL. This makes it possible to control pagination by making the URL mobile and including search terms in it. However, on first connection, the Hellowork asks whether to accept or reject cookies, which returns null when using static scraping packages such as Beautiful Soup. To get around this problem, we Selenium, which reproduces the movements of a browser.
- Initially, we retrieved the information from the search page. However, we found that all the information we were looking for could be found within each ad, accessible via a URL. The solution was to retrieve the ad URLs from each page, then extract all the desired information from each ad.
- The ad URLs are displayed with an HTML extension, this time allowing the use of the Beautiful Soup package. However, we did encounter a few minor challenges, notably with regard to job type and location, both of which have the same selection `soup.find('span')` with spans with the same name. Within their respective div, we had to retrieve all the spans, then divide them divide them using Python word processing. A similar process was also required to find the date.
- Finally, we had to deal with missing information, such as our HTML parser not finding object in the specified location. In fact, it’s possible that an element such as the type of job or the salary. Our solution was simply to check with if the parser couldn’t find an object, it returned `None`, and the same for other information.

Shiny web application