Published in MLearning.ai

Ahmed Negm Follow
Aug 9, 2021 · 5 min read

How to build a Web Scraper for LinkedIn

Photo by Greg Bulla on Upslash

Building Machine Learning Algorithms to fetch and analyze Linkedin data is a popular idea among ML Enthusiasts. But one snag everyone comes across is the lack of data considering how tedious data collection from Linkedin is and the legality behind it all.

The U.S. Ninth Circuit Court of Appeals ruled that the CFAA does not prohibit a company from scraping data that is publicly accessible on the internet, despite LinkedIn's claims that this violates user privacy.

San Francisco-based start-up hiQ Labs harvests user profiles from LinkedIn and uses them to analyze workforce data, for example by predicting when employees are likely to leave their jobs, or where skills shortages may emerge.

After LinkedIn took steps to block hiQ from doing this, hiQ won an injunction two years ago forcing the Microsoft-owned company to remove the block. That injunction has now been upheld by the 9th US Circuit Court of Appeals in a 3–0 decision.

In this article, we will be going over how you can build and automate your very own Web Scraping tool to extract data from any Linkedin Profile using just Selenium and Beautiful Soup. This is a step-by-step guide complete with code snippets at each step. I've added the GitHub repository link at the end of the article for those who would want the complete code.

Requirements

- python (Obviously. 3+ recommended)
- Beautiful Soup (Beautiful Soup is a library that makes it easy to scrape information from web pages.)
- Additionally, you'll also need the pandas, time, and regex libraries.
- A Code editor, I used Jupyter Notebook, you may use Vscode/Atom/Sublime or any of your choice.

> Use `pip install selenium` to install the Selenium library.
>
> Use `pip install beautifulsoup4` to install the Beautiful Soup library.
>
> You can download the Chrome WebDriver from here:
>
> https://chromedriver.chromium.org/

Overview

Here's a complete list of all the topics covered in this article.

- How to Automate Linkedin using Selenium.

- How to use BeautifulSoup to extract posts and their authors from the given Linkedin profiles.
- Writing the data to a .csv or .xlsx file to be made available for later use.
- How to automate the process to get posts of multiple authors in one go.

With that said, Let's Get Started

- We'll start by importing everything we'll need.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium import webdriver
from bs4 import BeautifulSoup as bs
import re as re
import time
import pandas as pd
```

- In addition to selenium and beautiful soup, we'll also be using the regex library to get the author of the posts, the time library to use time functionalities such as sleep and pandas to handle large-scale data, and for writing into spreadsheets. More on that coming right up!
- Selenium needs a few things to start the automation process. The location of the web driver in your system, a username, and a password to log in with. So let's start by getting those and storing them in variables PATH, USERNAME, and PASSWORD respectively.

```
PATH = input("Enter the Webdriver path: ")
USERNAME = input("Enter the username: ")
PASSWORD = input("Enter the password: ")
print(PATH)
print(USERNAME)
print(PASSWORD)
```

- Now we'll initialize our web driver in a variable that Selenium will use to carry out all the operations. Let's call it driver. We tell it the location of the web driver, i.e., PATH

```
driver = webdriver.Chrome(PATH)
```

- Next, we'll tell our driver the link it should fetch. In our case, it's Linkedin's home page.

```
driver.get("https://www.linkedin.com/uas/login")
time.sleep(3)
```

> You'll notice, I've used the sleep function in the above code snippet. You'll find it used a lot in this article. The sleep function basically halts any process (in our case, the automation process) for the specified number of number of seconds. You can freely use it to pause the process anywhere you need to like, in cases where you have to bypass a captcha verification.

- We'll now tell the driver to login with the credentials we've provided.

```
email=driver.find_element_by_id("username")
email.send_keys(USERNAME)
password=driver.find_element_by_id("password")
password.send_keys(PASSWORD)
time.sleep(3)
password.send_keys(Keys.RETURN)
```

- Now let's create a few lists to store data such as the profile links, the post content, and the author of each post. We'll call them post_links, post_texts, and post_names respectively.
- Once that's done, we'll start the actual web scraping process. Let's declare a function so we can use our web scraping code to fetch posts from multiple accounts in recursion. We'll call it Scrape_func.

Okay, that’s quite a long function. Worry not! I’ll explain it step by step. Let’s first go over what our function does.

- It takes 3 arguments, i.e., post_links, post_texts, and post_names as a, b, and c respectively.
- Now we’ll go into the internal working of the function. It first takes the profile link, and slices off the profile name.
- Now we use the driver to fetch the “posts” section of the user’s profile. The driver scrolls through the posts collecting the posts’ data using beautiful soup and storing them in “containers”.

- Line 17 of the above-mentioned code governs how long the driver gets to collect posts. In our case, it's 20 seconds, but you may change it to suit your data needs.

```
if round(end-start)>20:
    breakexcept:
    pass
```

- We also get the number of posts the user wants from each account and store it in a variable 'nos'.
- Finally, we iterate through each "container", fetching the post data stored in it and appending it to our post_texts list along with post_names. We break the loop when the desired number of posts is reached.

> You'll notice, we've enclosed the container iteration loop in a try-catch block. This is done as a safety measure against possible exceptions.

- hat's all about our function! Now, Time to put our function to use! We get a list of the profiles from the user and send it to the function in recursion to repeat the data collection process for all accounts.
- The function returns two lists: post_texts, containing all the posts' data, and post_names, containing all the corresponding authors of the posts.
- Now we've reached the most important part of our automation: Saving the data!

```
data = {
    "Name": post_names,
    "Content": post_texts,
}df = pd.DataFrame(data)
df.to_csv("gtesting2.csv", encoding='utf-8', index=False)writer =
pd.ExcelWriter("gtesting2.xlsx", engine='xlsxwriter')
df.to_excel(writer, index =False)
writer.save()
```

- We create a dictionary with the lists returned by the function and save it to a variable 'df' using pandas.
- You could either choose to save the collected data as a .csv file or a .xlsx file.

For csv:

```
df.to_csv("test1.csv", encoding='utf-8', index=False)
```

For xlsx:

```
writer = pd.ExcelWriter("test1.xlsx", engine='xlsxwriter')
df.to_excel(writer, index =False)
writer.save()
```

> In the above code snippets, I've given a sample file name, 'text1'. You may give any file name of your choice!

Conclusion:

Phew! That was long, but we've managed to successfully create a fully automated web scraper that'll get you any Linkedin post's data in a jiffy! Hope that was helpful! Do keep an eye out for more such articles in the future! Here's the link to the complete code on GitHub: https://github.com/AhmedKNegm/Scrape_func

Thanks for stopping by! Happy Learning!

Selenium Beautifulsoup Python3 Automation Machine Learning