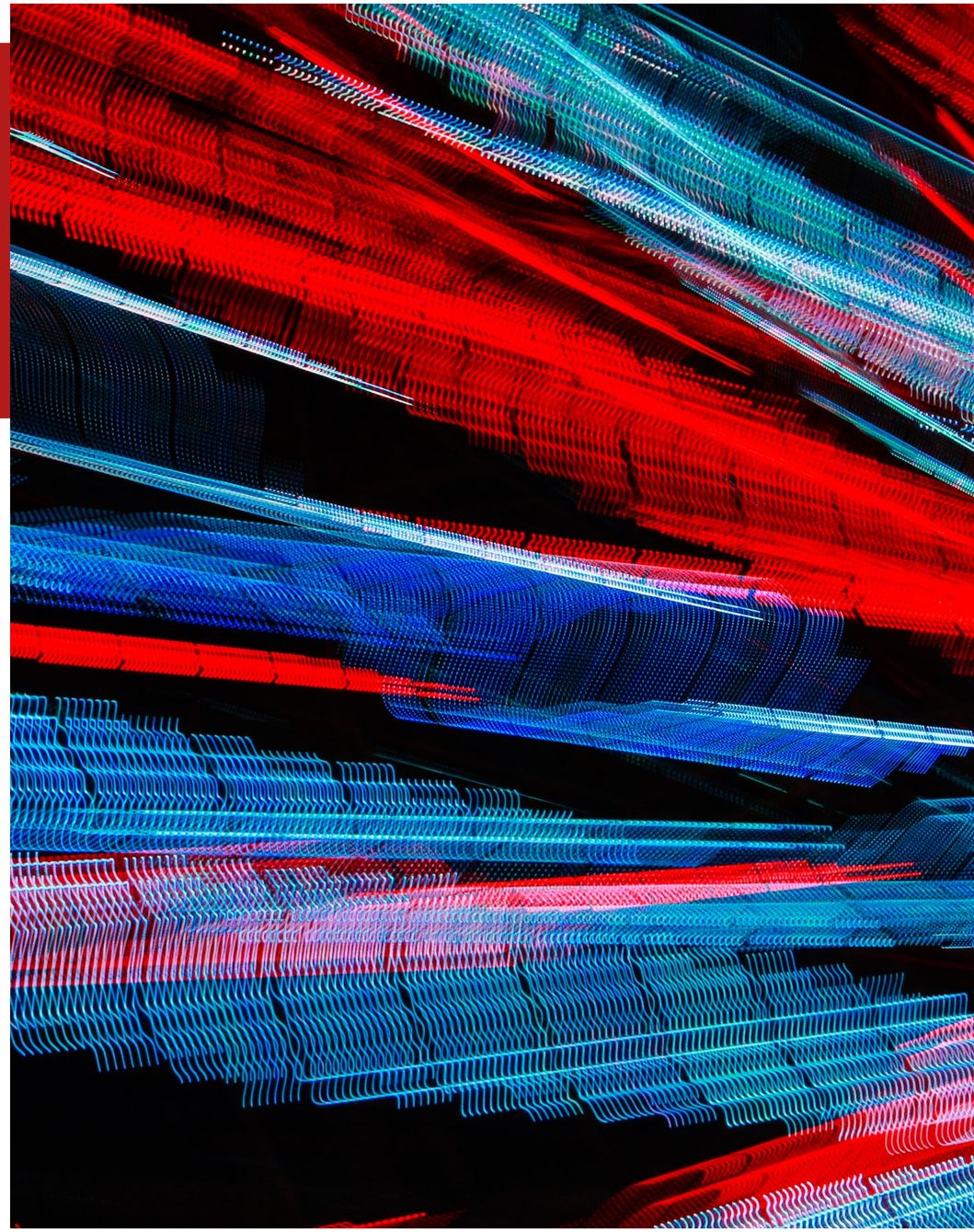
CORNELL CENTER for
SOCIAL SCIENCES

Web Scraping in Python

Remy Stewart

CCSS Data Science Fellow

November 10th, 2-4 PM



CCSS Code of Conduct

The Cornell Center for Social Sciences provides a welcoming environment for everyone embracing all backgrounds or identities. We encourage the following behaviors in our workshops:

- Respect differing viewpoints and ideas
- Share your own perspectives and ask any questions
- Accept constructive criticism
- Use welcoming and inclusive language
- Show courtesy and respect for all instructors and attendees

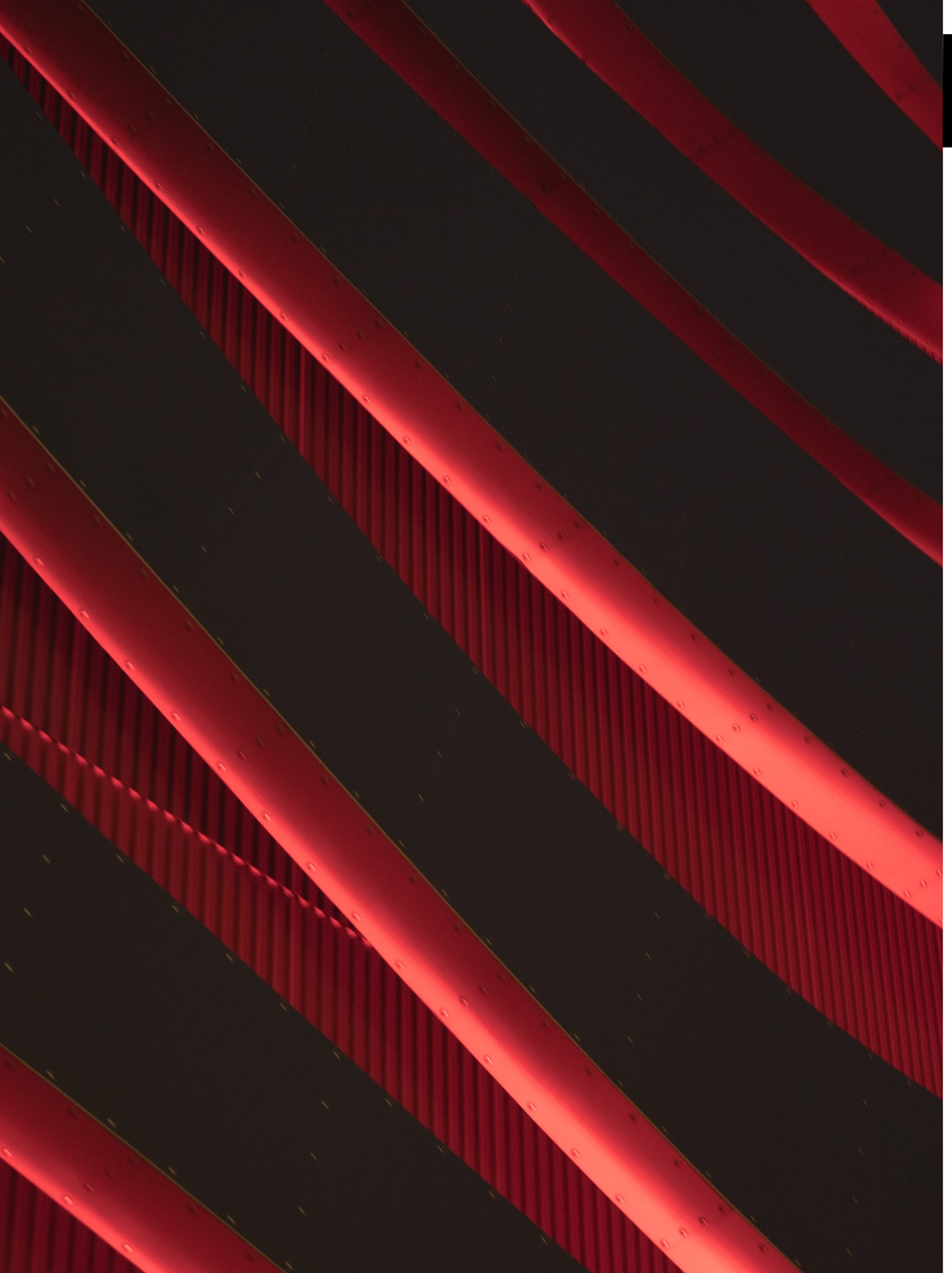
If you believe that an instructor or attendee has violated the code of conduct, please report the violation to CCSS-ResearchSupport@cornell.edu. We take all reported incidents seriously.

Land Acknowledgement

Cornell University is located on the traditional homelands of the Gayogo̱hó꞉nǫ' (the Cayuga Nation). The Gayogo̱hó꞉nǫ' are members of the Haudenosaunee Confederacy, an alliance of six sovereign Nations with a historic and contemporary presence on this land. The Confederacy precedes the establishment of Cornell University, New York state, and the United States of America. We acknowledge the painful history of Gayogo̱hó꞉nǫ' dispossession, and honor the ongoing connection of Gayogo̱hó꞉nǫ' people, past and present, to these lands and waters.

Here are additional links for more on the history of Cornell's violent, colonial formation, the movement to return native lands, and about the AIISP program at Cornell.

Consider donating to the Gayogo̱hó꞉nǫ' sovereignty initiative here.



Outline

- Introduction to Web Scraping
- Code Demo 1- Fundamentals with Beautiful Soup
- Break
- From Static to Dynamic Scraping
- Code Demo 2- Dynamic Sites with Selenium
- Conclusion

Web Scraping

Automated collection of data from websites spearheaded by the researcher

Web Scraping VS. **Application Programming Interfaces (APIs)**

- Data collection via individually developed programs
- Greater flexibility around acquired data
- More complex custom programs
- Challenges around access & use terms

- Querying pre-established databases
- Limited but often cleaner data
- Streamlined collection process
- Available data contracted by API provider

Why Web Scrape?

- Often the only means to get the data you're interested in
- Significantly faster than manual collection
- Automate website interactions

Why Not Web Scrape?

- Requires programming knowledge rare among social scientists
  (Hence our workshop today!)
- Will almost always encounter unexpected problems
- Associated risks around legality & ethics

Recommendation:

Prioritize alternative methods but learn the fundamentals for well-suited data opportunities

Legality & Ethics

- Ambiguous & actively debated topic within research communities
- Web scraping usually violates platforms' Terms of Service (ToS)
- Concern of legal action against individual web scrapers
- Easy to accidentally burden websites & other users while learning scraping
- Expectations around data use from platform users themselves

Interacting with Websites

Website browsers load content produced through programming languages including Hypertext Markup Language (**HTML**) and **Javascript**

```
<!DOCTYPE html>
<html>

    <head>
        <title>My First Webpage</title>
    </head>

    <body>
        <h1>
            My First Webpage
        </h1>
        <p>This is a paragraph...</p>
    </body>

</html>
```

Static content

- Embedded within the site's HTML
- Can simply download & parse specific elements
- Consists of text and links to files (JPG, PDF, etc.)

Dynamic content

- Implemented with Javascript & requires site interaction
- Clicking buttons, scrolling, logging in, inputing text, etc.
- Data often does not appear until we've automated these behaviors

Python Libraries for Web Scraping

- Most user friendly but slowest & least flexible
- Not designed for dynamic sites
- Best for small cross-sectional projects

- Moderate complexity but greater flexibility
- Navigates Javascript interactivity
- Best for larger cross-sectional projects

- Greatest flexibility & least user friendly
- Combine with Selenium to navigate multiple websites at once
- Best for large onging data collection initiatives

Code Demo 1- Fundamentals with Beautiful Soup

https://colab.research.google.com/drive/1VwyM37dnepd2LaERcFU1kLcID9J487IJ?usp=sharing

From Static to Dynamic Scraping

- Javascript within websites has become so widespread that most scrapers will have to account for dynamic interactions
- Selenium will automate this process building from the limitations of Beautiful Soup
- Use the website’s Inspect feature to identify what elements need to be manipulated

Common Selenium Scraping Workflow

1.) Access URL by using a launched browser **web driver**

2.) Interact with the page through the driver to have the site correctly display data by clicking, submitting, etc.

3.) Implement **waits** to ensure that the site has time to update based on interactions

4.) Download the content retrieved from the page

5.) Manipulate & clean the data

Web Drivers

- Use a running instance of a browser to facilitate website interactions
- External program that needs to be downloaded and referred to within the scraper
- Different drivers for Chrome, Firefox, Microsoft Edge
  - We'll be using ChromeDriver in our code demo
- **Headless Drivers** are ran in the background without launching a visible browser

Waiting Times (Waits)

Two primary reasons for implementing waits:

1.) Provides site time to update after interacting with elements

2.) Minimizes site request burdens & reduces likelihood of Internet Protocol (IP) address blockage

Implicit Waits

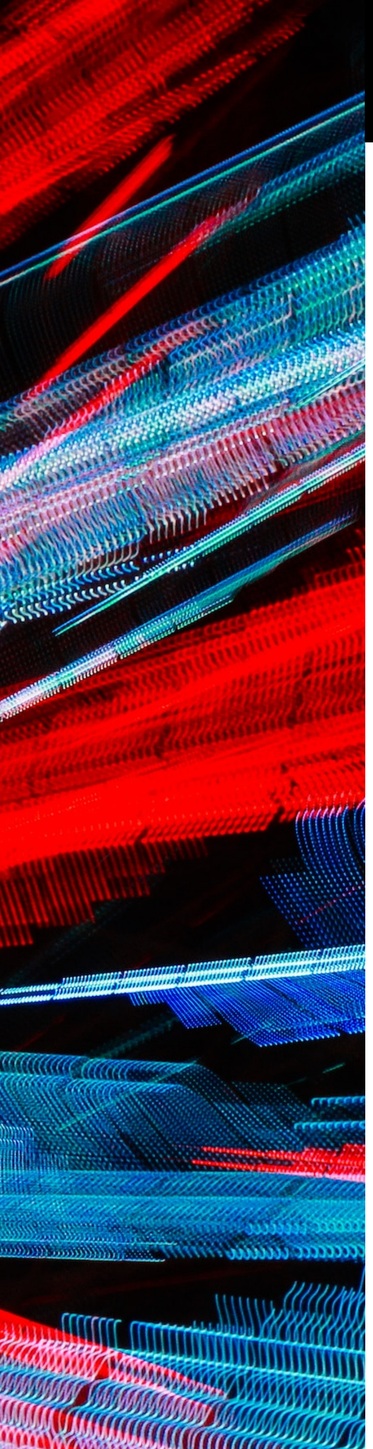
- Pauses program for designated amount of seconds

Explicit Waits

- Set a condition to be met before proceeding such as an element being clickable

Code Demo 2- Dynamic Sites with Selenium

https://colab.research.google.com/drive/1VwyM37dnepd2LaERcFU1kLcID9J487IJ?usp=sharing



Additional Topics

Iterating on Scrapers

- Websites often change in ways that breaks scrapers
- Scraper development as an iterative process that requires long-term planning

Data Storage

- Scraping projects for large and/or continuously collected data are usually paired with a database backend
- Primarily SQL-based such as SQLite, PostgreSQL and/or cloud platform providers (AWS S3/Athena)

Data Sharing

- Navigate the tension of open data access with legal reprecussions
- Also applies to public programs and Github repos

I got blocked! What should I do?

1.) Try again later

2.) Revise your code to be less demanding on the website

3.) Run your script during non-peak times

4.) Investigate the specific error that’s being raised. Is it caused by another underlying issue?

5.) Consider IP rotation via proxy servers or using a Virtual Private Network (VPN)

Conclusion

- Web scraping is a powerful tool that with care & management can provide unparalleled data access
- Ongoing process of navigating sites, revising scrapers, and adjusting to dynamic changes
- Weight challenges with social beneficence & scientific advancement

“How can I ground my project towards social good?”

Thank You! Questions?

Remy Stewart

CCSS Data Science Fellow

CCSS-ResearchSupport@cornell.edu