1. Assignment 3

2. Read

a. Foster ch. 2 (Scraping sections 2.2)

Chapter 2 of Big Data and Social Science focuses on how to collect and integrate data from the web using two main approaches: web scraping and APIs. Web scraping is the process of automating the extraction of information from websites by fetching HTML pages and parsing specific elements, often with tools like Python's requests and BeautifulSoup. While scraping can provide access to valuable data, it is fragile because even small changes to a website's structure can break the code, and there are also legal and ethical considerations since some sites forbid automated scraping. APIs, on the other hand, provide structured and more stable access to data. RESTful APIs in particular allow researchers to send HTTP requests to endpoints and receive responses in JSON format, which can be easily parsed into Python objects. Wrappers are often available to simplify API usage, though they may lag behind changes to the underlying API. Examples of useful APIs include Crossref for DOIs and publications, ORCID for researcher identifiers, and PubMed or Twitter for domain-specific data. Once data is retrieved, either by scraping or APIs, the challenge becomes integrating multiple sources, which requires matching identifiers such as DOIs or ORCID IDs, dealing with missing or inconsistent values, harmonizing formats, and handling different update cycles. The chapter emphasizes that APIs are generally preferable to scraping because they are cleaner and more stable, but both methods require careful attention to data quality, reproducibility, and compliance with ethical and legal standards. Ultimately, successful use of web data depends on planning ahead for integration and using consistent identifiers to connect datasets.

b. R4DS (2e) ch. 24

Chapter 24 explains how to use the rvest package in R to collect data from websites when an API is not available. It begins with the importance of considering ethics and legality: always check terms of service, avoid scraping personally identifiable information, respect copyright, and be polite to servers by limiting requests. The chapter then introduces the basics of HTML structure—elements, attributes, and nesting—which form the foundation for selecting data. Data is extracted by reading the HTML with read_html() and then using functions like html_elements() or html_element() with CSS selectors to target specific parts of a page. Text can be retrieved with html_text2() and attributes (such as links or image sources) with html_attr(). If data are in HTML tables, html_table() can directly convert them into tibbles. Since choosing the right selector is critical, the book recommends using tools like SelectorGadget or browser developer tools to identify precise CSS selectors. The chapter demonstrates these skills with examples, including scraping Star Wars movie data and IMDb's Top 250 list, which show how to handle real-world challenges like string cleaning and attribute extraction. It also highlights a limitation: many modern sites are dynamic, meaning the content is loaded via JavaScript and not present in the initial HTML; in these cases, more advanced tools like the chromote package may be required. The chapter concludes by stressing that web scraping is powerful but must be used responsibly, with equal attention to technical methods, legal boundaries, and ethical practices.

c. Additional/Optional: Aydin, Olgun. 2018. R Web Scraping Quick Start Guide: Techniques and tools to crawl and scrape data from websites. Packt Publishing Ltd,

3. Research

a. R packages for web scraping

Several R packages are commonly used for web scraping. The most widely used is rvest, which provides simple functions for downloading HTML, selecting elements with CSS or XPath selectors, and extracting text or attributes. Other packages include httr (for handling HTTP requests and responses, including headers, authentication, and rate limits), xml2 (for parsing XML and HTML documents), polite (which adds safeguards for scraping responsibly, e.g., respecting robots.txt and limiting request rates), and RSelenium (for interacting with dynamic, JavaScript-heavy websites by driving a browser session).

b. Choose two for comparison

rvest is ideal for static web pages where the content is already present in the HTML source. It is easy to learn, integrates well with the tidyverse, and allows direct extraction of tables, text, and attributes. However, it cannot handle sites where content loads dynamically through JavaScript.

RSelenium controls a browser (like Chrome or Firefox) through Selenium, allowing you to scrape dynamic websites, fill forms, click buttons, or scroll pages. It is much more powerful but also more complex to set up and slower to run compared to rvest. In short, rvest is lightweight and simple for static HTML, while RSelenium is heavyweight but necessary for complex, interactive sites.

c. What is robots.txt?

The robots.txt file is a standard used by websites to tell automated crawlers and bots which parts of the site they may or may not access. It is placed at the root of a domain (e.g., https://example.com/robots.txt). While it is not legally binding in most jurisdictions, it is considered best practice and ethical to follow its rules. The file typically specifies user-agent directives (which bots the rules apply to) and allow/disallow paths indicating which URLs can or cannot be crawled.

d. Find two websites' robots.txt and analyze

https://www.nytimes.com/robots.txt

The file disallows many areas such as /search/, /comments/, and /svc/, while allowing access to most news article content. This suggests that The New York Times allows bots to crawl its articles for indexing but restricts crawling of interactive features, APIs, and private services to protect performance and sensitive user content.

https://www.reddit.com/robots.txt

The file blocks crawling of many internal or API-related paths (e.g., /api/, /login, /r/*/comments/) but allows crawling of subreddit overview pages. This indicates Reddit wants search engines and scrapers to access subreddit-level content but not comment threads or backend APIs, likely to control server load and protect user-generated discussions.