

Big Data and Economics

Lecture 6a: Web Data in Research

Kyle Coombs (he/him/his)

Bates College | [EC/DCS 368](#)

Table of contents

1. Prologue
2. Worldwide Web of Data
3. Examples of scraping in economics research
4. Access methods
 - Click and Download
 - Server-side scraping
 - Client-side scraping
5. Ethics of web scraping

Prologue

Check-in

- I graded Part IV of problem sets
 - Knit, knit, knit, knit, knit your R Markdown submission: ensures your code runs
 - New policy for PS submission: full points for pushing with GitHub, 10 points off if you upload a zip file
- Tip on debugging: just because R writes something in red text, it doesn't mean it's an error
 - Confirm a bug is a bug before you go down the rabbit hole
- GitHub message: "This branch is **2 commits ahead of, 2 commits behind** big-data-and-economics/ps1-employment-discrimination:main."
 - This is not (necessarily) a bug. Can anyone explain what it means
- Ask and answer questions on GitHub issues: part of your grade is using it
- Final project annotated summaries were due yesterday. Thoughts?
- Data description has been punted to be due after winter break (nothing is stopping you from doing it sooner though)

Prologue

- We've spent the first month of this class on learning:
 - empirical organization skills ("Clean Code"),
 - basics of R
 - basics of data wrangling and tidy data
- Now we're going to learn about a data acquisition skill: **scraping**
- Essentially, we're going to learn how to get data from the web
- These data are usually messy in one way or another, so it'll give you something to tidy

Plan for today

- What is scraping?
- Contrast Client-side and Server-side scraping
- Examples of scraping in economics research
- Ethical considerations
- Learn by doing with APIs (CSS will happen later -- potentially end of semester)

Attribution

- These slides take inspiration from the following sources:
 - [Nathan Schiff's web data lecture](#)
 - [Andrew MacDonald's slides](#)
 - [Jenny Bryan's textbook](#)
 - [Grant McDermott's notes on CSS and APIs](#)
 - [James Densmore's stance on ethics](#)

Worldwide Web of Data

Worldwide Web of Data

- Every website you visit is packed with data
- Every app on your phone is packed with data and taking data from you
- Guess what?
 - These data often measure hard to measure things
 - These data are often public (at some level of aggregation/anonymity)
 - These data are often not easily accessible and not **tidy**
 - Samples might be biased (have to navigate that)
 - This is legal (usually) and ethical (usually)
- Guess what? All this makes these data (and knowing how to access it) valuable
 - It also makes this a hard skill to pick up

Examples of scraping in economics research

What cool things can you do with web

- Can anyone think of examples of web data being used in economics research?

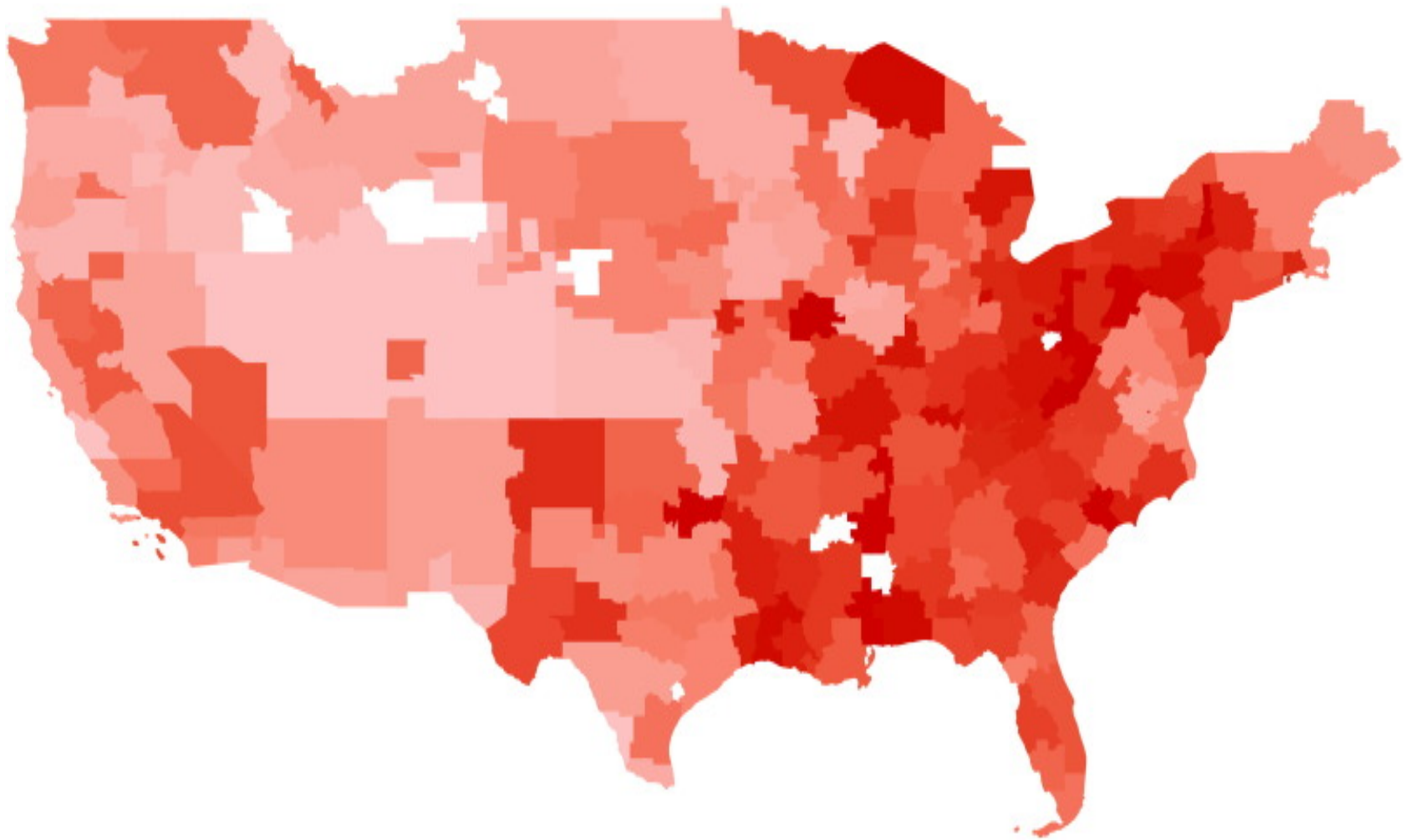
Measuring hard to measure things

- Imagine you survey a ton of people about their beliefs that a candidate is unfit to be president because of their race
- Due to social desirability bias, you get a lot of "I don't know" or "I don't think that"
- There are lots of creative survey methods to get at this, but is there some way to measure this without asking people?
- Say, why not find out the frequency that people search Google for racial epithets in connection to the candidate?
- Guess what? Stephens-Davidowitz (2014) did just that
 - Finds racial animus cost Barack Obama 4 percentage points in the 2008 election (equivalent of a home-state advantage)
 - Google search term data yield effects that are 1.5 to 3 times larger than survey estimates of racial animus

$$\text{Racially Charged Search Rate}_j = \left[\frac{\text{Google searches including the word "Word 1 (s)"} }{\text{Total Google searches}} \right]_{j,2004-2007}$$

for j geographical area (state, county, etc.)

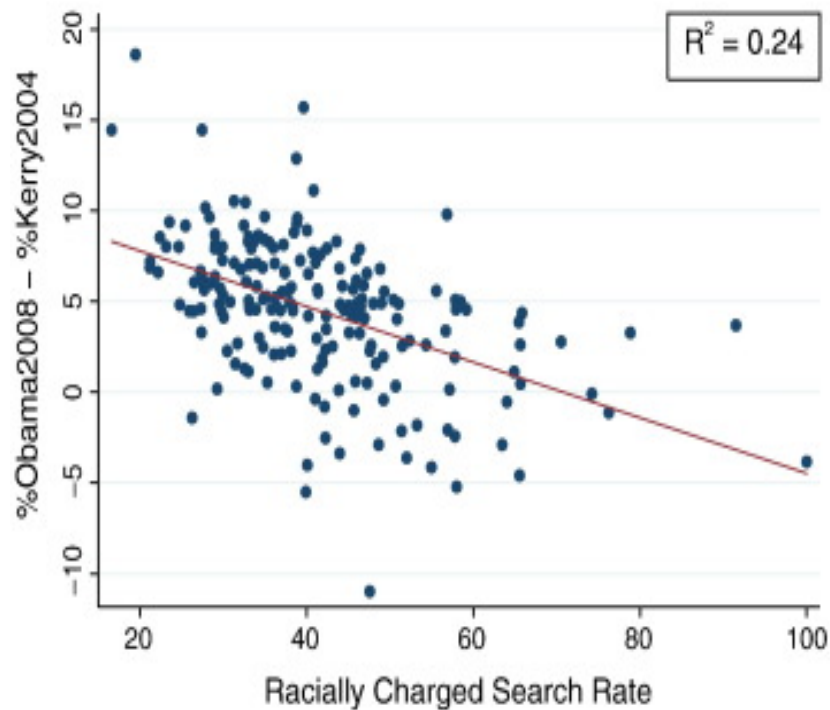
Racial Animus Map



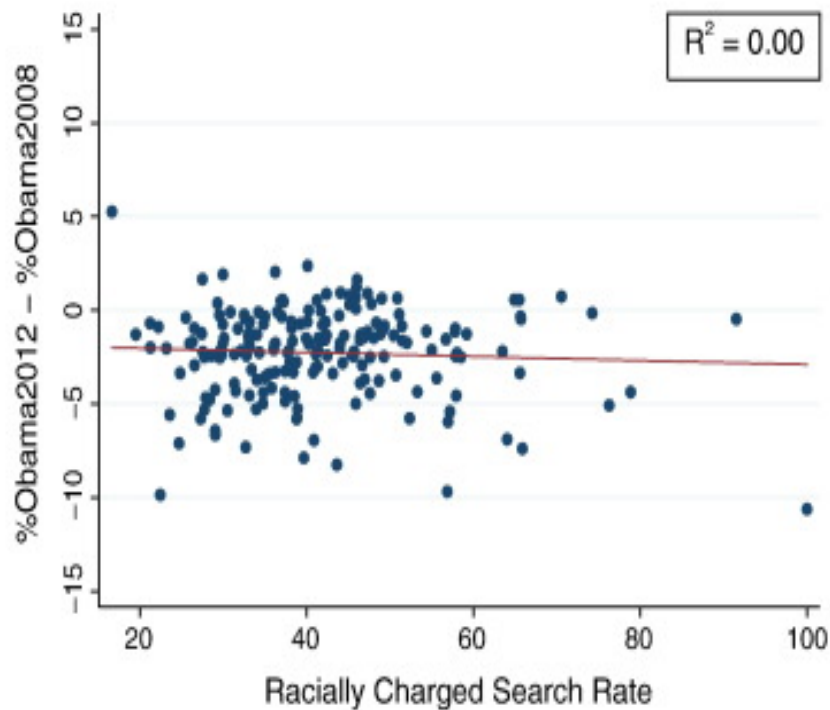
Map of media markets by racially charged search rate from 2004 to 2007. The darker red, the more racially charged.

Election performance

(a) Obama - Kerry



(b) Obama - Obama



Obama underperformed Kerry in areas with more racially charged search rates.

Other uses

- "Billion prices project" (Cavallo and Rigobon 2015) : collect prices from online retailers to look at macro price changes
- Davis and Dingell (2016): use Yelp to explore racial segregation in consumption
- Halket and Pginatti (2015): scrape Craigslist to look at housing markets
- Wu (2018): undergraduate hacked into online economics job market forum to look at toxic language and biases in the academic economics against women
- Glaeser (2018) uses Yelp data to quantify how neighborhood business activity changes as areas gentrify (**Student presentation**)
- Tons leverage eBay, Alibaba, etc. to look at all kinds of commercial activity
- Edelman B (2012) gives an overview of using internet data for economic research

Access methods

Access methods

There are three ways to data off the web:

1. **click-and-download** on the internet as a "flat" file, like a CSV or Excel file
 - What you're used to
 2. **Server-side** websites that sends HTML and JavaScript to your browser, which then renders the page
 - People often call this "scraping"
 - All the data is there, but not in a tidy format
 - **Key concepts:** CSS, Xpath, HTML
 3. **Client-side** *websites contain an empty template that _request* data from a server and then fills in the template with the data
 - The request is sent to an API (application programming interface) endpoint
 - Technically you can just source right from the API endpoint (if you can find it) and skip the website altogether
 - I consider this a form of scraping
 - **Key concepts:** APIs, API endpoints
- Key takeaway: if there's a structure to how the data is presented, you can exploit it to get the data

Click and Download

- You've all seen this approach before
- You go to a website, click a link, and download a file
- Sometimes you need to login first, but if not you can automate this with R's `download.file()` function
- Below will download the Occupational Employment and Wage Statistics (OEWS) data for Massachusetts in 2021 from the BLS

```
download.file("https://www.bls.gov/oes/special.requests/oesm21ma.zip", "oesm21ma.zip")
```

Client-side scraping

- The website contains an empty template of HTML and CSS.
 - E.g. It might contain a “skeleton” table without any values.
- However, when we actually visit the page URL, our browser sends a request to the host server.
- If everything is okay (e.g. our request is valid), then the server sends a response script, which our browser executes and uses to populate the HTML template with the specific information that we want.
- **Webscraping challenges:** Finding the “API endpoints” can be tricky, since these are sometimes hidden from view.
- **Key concepts:** APIs, API endpoints

APIs

- [Zapier](#) offers a fantastic intro to APIs
- APIs is a collection of rules/methods that allow one software application to interact with another
- Examples include:
 - Web servers and web browsers
 - R libraries and R clients
 - Databases and R clients
 - Git and GitHub and so on


Key API concepts

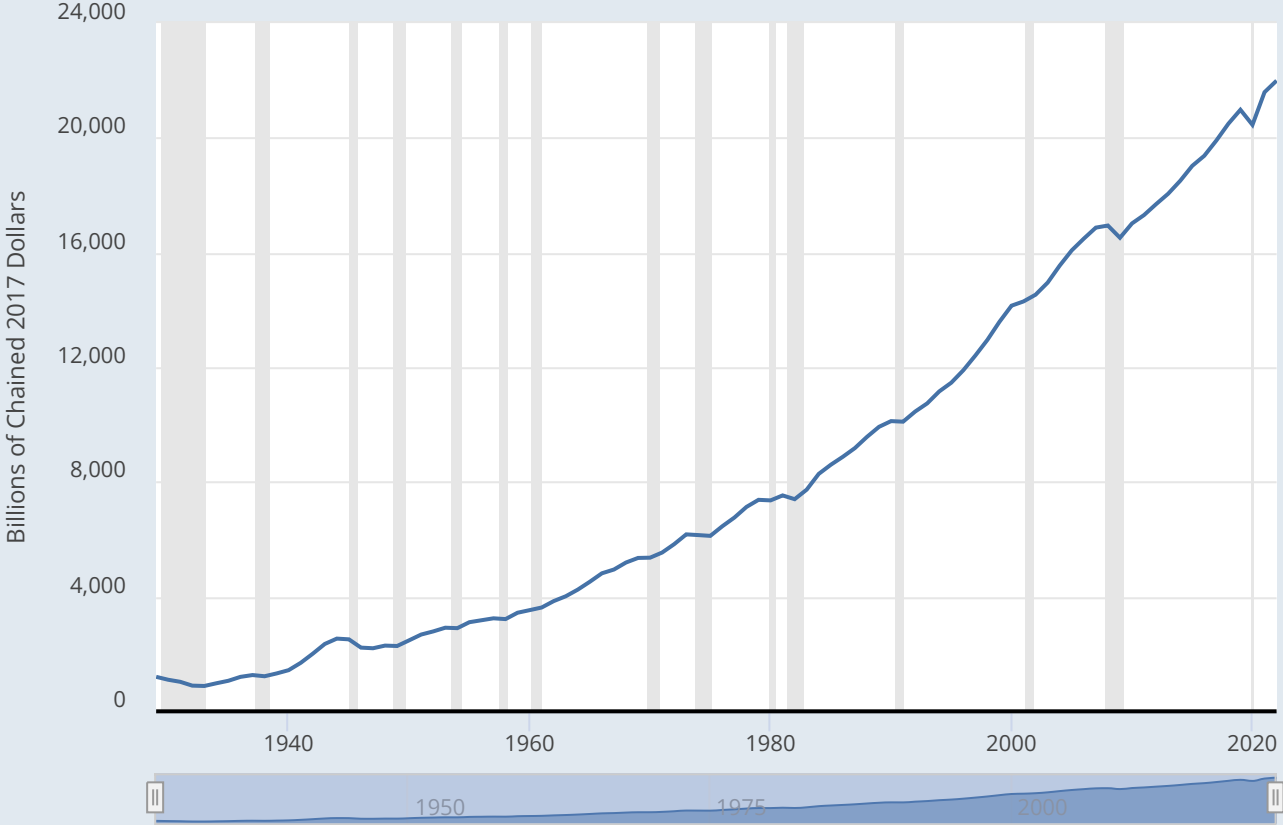
- **Server:** A powerful computer that runs an API.
- **Client:** A program that exchanges data with a server through an API.
- **Protocol:** The “etiquette” underlying how computers talk to each other (e.g. HTTP).
- **Methods:** The “verbs” that clients use to talk with a server. The main one that we’ll be using is GET (i.e. ask a server to retrieve information), but other common methods are POST, PUT and DELETE.
- **Requests:** What the client asks of the server (see Methods above).
- **Response:** The server’s response. This includes a Status Code (e.g. “404” if not found, or “200” if successful), a Header (i.e. meta-information about the response), and a Body (i.e. the actual content that we’re interested in).

API Endpoints

- Web APIs have a URL called an **API Endpoint** that you can use to access view the data in your web browser
- Except instead of rendering a beautifully-formatted webpage, the server sends back a ton of messy text!
 - Either a JSON (JavaScript object notation) or XML (eXtensible Markup Language) file
- It'd be pretty overwhelming to learn how to navigate these new language syntaxes
- Guess what? R has packages to help you with that
 - `jsonlite` for JSON
 - `xml2` for XML
- Today we're going to work through a few of these
- That means the hardest parts are:
 - Finding the API endpoint
 - Understanding the rules
 - Identify the words you need to use to get the data you want
- To be clear, that's all still tricky!

You've likely used FRED before

FRED  — Real Gross National Product



Source: U.S. Bureau of Economic Analysis

[Customize](#) | [Download Data](#) | [FRED - Economic Data from the St. Louis Fed](#)

Underneath is an API!

- The endpoint is https://api.stlouisfed.org/fred/series/observations?series_id=GNPCA&api_key=YOUR_API_KEY&file_type=json
- Just sub in your API key and you're good to go

```
{"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "observation_start": "1600-01-01", "observation_end": "9999-12-31", "units": "1"}, [{"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1929-01-01", "value": "1202.659"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1930-01-01", "value": "1100.67"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1931-01-01", "value": "1029.038"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1932-01-01", "value": "895.802"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1933-01-01", "value": "883.847"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1934-01-01", "value": "978.188"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1935-01-01", "value": "1065.716"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1936-01-01", "value": "1201.443"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1937-01-01", "value": "1264.393"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1938-01-01", "value": "1222.966"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1939-01-01", "value": "1320.924"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1940-01-01", "value": "1435.656"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1941-01-01", "value": "1690.844"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1942-01-01", "value": "2008.853"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1943-01-01", "value": "2349.125"}, {"realtime_start": "2024-02-03", "realtime_end": "2024-02-03", "date": "1944-01-01", "value": "2535.744"}]
```


What did I need to know?

- The base URL: <https://api.stlouisfed.org/>
- The API endpoint (fred/series/observations/)
- The parameters:
 - series_id="GNPCA"
 - api_key=YOUR_API_KEY
 - file_type=json
- What's an API Key? It is a unique identifier that is used to authenticate a user, developer, or calling program to an API.
 - It's like a password, but it's not a password
 - It tracks who is using the API and how much they're using it
 - Example: `asdfjaw523a3523414at43sad`

Hide your API Key

- In general, you don't want to share your API key with anyone
- Instead, you can make it an environment variable either for a single session or permanently

```
Sys.setenv(FRED_API_KEY_TEST="abcdefghijklmnopqrstuvwxy0123456789")  
FRED_API_KEY_TEST = Sys.getenv("FRED_API_KEY_TEST")  
FRED_API_KEY_TEST
```

```
## [1] "abcdefghijklmnopqrstuvwxy0123456789"
```

- You can also permanently add it to your `.Renviron` file, by running the `edit_r_environ()` function from the **usethis** package

```
usethis::edit_r_environ()
```

- Then just type in `FRED_API_KEY_TEST=abcdefghijklmnopqrstuvwxy0123456789` and save the file and re-read using

```
readRenviron("~/Renviron")
```

- Any time you need it, you can just call `Sys.getenv("FRED_API_KEY_TEST")`

Popular APIs

- Many popular APIs are free to use and have a lot of documentation
- Sometimes the documentation gets a bit cumbersome though
- So kind souls have developed R packages to help you "abstract" these details (**Clean Code**)
- For example, the `tidycensus` package is a wrapper for the US Census API
 - You'll use it on your problem set
- Others include: `fredr`, `blsAPI`, `rgithub`, `googlesheets4`, `googledrive`, `wikipediR`, etc.
- Here's a curated list: <https://github.com/RomanTsegelskyi/r-api-wrappers>

Hidden APIs

- Sometimes the API endpoint is hidden from view
- But you can find it by using the "Inspect" tool in your browser
- It will require some detective work!
- But if you pull it off, you can get data that no one else has

Server-side scraping

- The scripts that “build” the website are not run on our computer, but rather on a host server that sends down all of the HTML code.
 - E.g. Wikipedia tables are already populated with all of the information — numbers, dates, etc. — that we see in our browser.
- In other words, the information that we see in our browser has already been processed by the host server.
- You can think of this information being embedded directly in the webpage’s HTML.
 - So if we can get our hands on the HTML, we can get our hands on the data.
 - We just have to figure out how to strip off the HTML and get the data into a tidy format.
- **Webscraping challenges:** Finding the correct CSS (or Xpath) “selectors”. Iterating through dynamic webpages (e.g. “Next page” and “Show More” tabs).
- **Key concepts:** CSS, Xpath, HTML
- **R package:** `rvest` has a suite of functions to help convert HTML to a tidy format

Underneath Wikipedia

W List of Olympic records in athlet x +

en.wikipedia.org/wiki/List_of_Olympic_records_in_athletics

Google Voice - Inbo... us.megabus.com/ab... Gmail YouTube Maps Random Econ Ideas DND Columbia Stuff Life Jazz All Bookmarks

Beamon's compatriot, [Mike Powell](#), jumped farther in the [1991 World Championships in Athletics in Tokyo](#).^[1]


Note, only those events currently competed for and recognised by the IOC as Summer Olympic events are listed.^[8]

Men's records [edit]

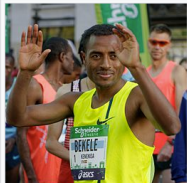
♦ denotes a performance that is also a current [world record](#). Statistics are correct as of August 3, 2021.

List of men's Olympic records in athletics

Event	Record	Athlete(s)	Nation	Games	Date	Ref(s)
100 metres	9.63	Usain Bolt	 Jamaica (JAM)	2012 London	August 5, 2012	[9]
200 metres	19.30	Usain Bolt	 Jamaica (JAM)	2008 Beijing	August 20, 2008	[10]
400 metres	♦43.03	Wayde van Niekerk	 South Africa (RSA)	2016 Rio de Janeiro	August 14, 2016	[11]
800 metres	♦1:40.91	David Rudisha	 Kenya (KEN)	2012 London	August 9, 2012	[12]
1,500 metres	3:28.32	Jakob Ingebrigtsen	 Norway (NOR)	2020 Tokyo	August 7, 2021	[13]
5,000 metres	12:57.82	Kenenisa Bekele	 Ethiopia (ETH)	2008 Beijing	August 23, 2008	[14]
10,000 metres	27:01.17	Kenenisa Bekele	 Ethiopia (ETH)	2008 Beijing	August 17, 2008	[15]
Marathon	2:06:32	Samuel Wanjiru	 Kenya (KEN)	2008 Beijing	August 24, 2008	[16]
110 metres	12.91	Liu Xiang	 China (CHN)	2004 ...	August 27, 2004	[17]



[Usain Bolt](#) currently holds three Olympics records, two individually and one with the [Jamaican 4 × 100m relay team](#).



The HTML source

- If we can just cut out all the HTML and get the data into a tidy format, we're golden
- Better yet, we can use some of the HTML to help us find **harvest** the data we want

```
<caption>List of men's Olympic records in athletics
</caption>
<tbody><tr>
<th scope="col" width="12%">Event
</th>
<th class="unsortable" width="5%">Record
</th>
<th scope="col" width="10%">Athlete(s)
</th>
<th scope="col" width="15%">Nation
</th>
<th scope="col" width="10%">Games
</th>
<th scope="col" width="5%">Date
</th>
<th scope="col" class="unsortable" width="3%">Ref(s)
</th></tr>
<tr>
<th scope="row"><span data-sort-value="001006#160;! "><a href="/wiki/100_metres" title="100 metres">100 metres</a></span>
</th>
<td align="right">9.636#160;
</td>
<td><span data-sort-value="Bolt, Usain"><span class="vcard"><span class="fn"><a href="/wiki/Usain_Bolt" title="Usain Bolt">Usain Bolt
</td>
<td><span class="mw-image-border" typeof="mw:File"><span><a href="/wiki/Athletics_at_the_2012_Summer_Olympics_%E2%80%93_Men%27s_100_metres" title="Ath
</td>
<td><span data-sort-value="000000002012-08-05-0000" style="white-space:nowrap">August 5, 2012</span>
</td>
<td align="center"><sup id="cite_ref-9" class="reference"><a href="#cite_note-9">9#91;96#93;</a></sup>
</td></tr>
```

Stability and CSS scraping

- Websites change over time
- That can break your scraping code
- This makes scraping as much of an "art" as it is a science

Ethics of web scraping

Legality of web scraping

- All of today is about how to get data off the web
- If you can see it in a browser window and work out its structure, you can scrape it
- And the legal restrictions are pretty obscure, fuzzy, and ripe for reform
 - hiQ Labs vs LinkedIn court ruling defended hiQ's right to scrape, then the Supreme Court vacated the ruling, and the final decision was against HiQ Labs
 - The Computer Fraud and Abuse Act (CFAA) protects the scraping of publicly available data
 - Legality gets messy around personal data and intellectual property (for good reason, but again reform is needed)

Ethics of web scraping

- Technically, web scraping just automates what you (or a team of **well**-compensated RAs) could do manually
 - It's just a lot faster and more efficient (no offense)
- Webscraping is an integral tool to modern investigative journalism
 - Sometimes companies hide things in their HTML that they don't want the public to see
 - Pro Publica has developed a tool called [Upton](#) to make it more accessible
- So I stand firmly on the pro-scraping side with a few ethical caveats
 - Just because you can scrape it, doesn't mean you should
 - It's pretty easy to write up a function or program that can overwhelm a host server or application through the sheer weight of requests
 - Or, just as likely, the host server has built-in safeguards that will block you in case of a suspected malicious Denial-of-serve (DoS) attack

Be nice

- Once you get over the initial hurdles, scraping is fairly easy to do (cleaning can be trickier)
- There's plenty of digital ink spilled on the [ethics of web scraping](#)
- The key takeaway is to be nice
 - If a public API exists, use it instead of scraping
 - Only take the data that is necessary
 - Have good reason to take data that is not intentionally public
 - Do not repeatedly swarm a server with requests (use `sys.sleep()` to space out requests)
 - Scrape to add value to the data, not to take value from the host server
 - Properly cite any scraped content and respect the terms of service of the website
 - Document the steps taken to scrape the data

polite package and robots.txt

- Sites often have a "robot.txt," which is a file that tells you what you can and cannot scrape
- A "web crawler" should be written to start with the robots.txt and then follow the rules
- The `polite` package is a tool to help you be nice
- It explicitly checks for permissions and goes to the robots.txt of any site you visit
- As you get better at scraping and start trying to scrape at scale, you should use this

Conclusion

- Web content can be rendered either 1) server-side or 2) client-side.
- Client-side content is often rendered using an API endpoint, which is a URL that you can use to access the data directly.
 - APIs are a set of rules/methods that allow one software application to interact with another they often require an access token
 - You can use R packages (**httr**, **xml2** **jsonlite**) to access these endpoints and tidy the data.
 - Popular APIs have packages in R or other software that streamline access
- Server-side content is often rendered using HTML and CSS.
 - Use the **rvest** package to read the HTML document into R and then parse the relevant nodes.
 - A typical workflow is: `read_html(URL) %>% html_elements(CSS_SELECTORS) %>% html_table()`.
 - You might need other functions depending on the content type (e.g. `html_text`).
- Just because you can scrape something doesn't mean you should (i.e. ethical and possibly legal considerations).
- Webscraping involves as much art as it does science. Be prepared to do a lot of experimenting and data cleaning.

Next: Onto scraping and API activities!
