The Web as Data

Jared Edgerton

Why the Web Matters for Social Data

Large volumes of social data are:

- Public
- Continuously updated
- Heterogeneous
- Poorly documented

The web is often the *first* place social behavior appears.

The Web Is Not a Database

Websites are built for **humans**, not analysis.

This leads to:

- Irregular structure
- Hidden assumptions
- Fragile access points

Scraping is the act of *recovering structure* from presentation.

Ways Data Appear on the Web

Common forms:

- Static HTML pages
- Embedded tables
- Linked documents (PDFs, CSVs)
- JavaScript-rendered content
- Archived snapshots

Most useful data are *incidental*, not intentional.

What HTML Is (Conceptually)

HTML is a markup language that:

- Defines document structure
- Organizes content hierarchically
- Uses nesting rather than labels

It describes *what exists*, not *what it means*.

HTML as a Tree

An HTML page forms a tree:

- Elements contain other elements
- Parent → child → sibling relationships
- Position conveys meaning

Scraping is tree traversal and pattern matching.

Core HTML Building Blocks

Key components:

- Tags: `div`, `p`, `a`, `table`
- Attributes: `class`, `id`, `href`
- Text nodes

Meaning is inferred from repetition and location.

Scraping as Pattern Recognition

Effective scraping relies on:

- Repeated structures
- Stable attributes
- Predictable nesting

You are reverse-engineering design decisions.

Static HTML Scraping (Python)

```python
import requests
from bs4 import BeautifulSoup

url = 'https://example.com'
resp = requests.get(url, headers={'User-Agent': 'research-bot'})
soup = BeautifulSoup(resp.text, 'html.parser')

titles = [h.text.strip() for h in soup.find_all('h2')]
```

Best when content loads directly in the page.

Static HTML Scraping (R)

```r
library(rvest)

url ← 'https://example.com'
page ← read_html(url)

titles ← page ▷
  html_elements('h2') ▷
  html_text(trim = TRUE)
```

Same logic, different syntax.

Embedded Tables

Many sites expose data via HTML tables.

```python
import pandas as pd

tables = pd.read_html('https://example.com/stats')
df = tables[0]
```

```r
library(rvest)

page <- read_html('https://example.com/stats')
df <- page |> html_table() |> .[[1]]
```

Often underused and very stable.

Pagination Without APIs

Data are often split across pages.

```python
for page in range(1, 6):
    url = f'https://example.com/page={page}'
    html = requests.get(url).text
```

```r
for (p in 1:5) {
  url <- paste0('https://example.com?page=', p)
  page <- read_html(url)
}
```

Look for predictable URL patterns.

Following Links

Many datasets require crawling links.

```python
links = [a['href'] for a in soup.find_all('a', href=True)]
```

```r
links <- page |>
  html_elements('a') |>
  html_attr('href')
```

Always normalize and deduplicate URLs.

Respectful Scraping

Good practice includes:

- Rate limiting
- User-agent headers
- Error handling
- Saving raw responses

Scraping is an interaction, not extraction.

Underused Sources of Web Data

Often overlooked:

- Local government websites
- Professional associations
- Regulatory filings
- Court postings
- Static reports

These are often more stable than platforms.

The Value of Boring Websites

Boring sites tend to have:

- Minimal JavaScript
- Clean HTML
- Few access restrictions

They are often the highest-quality sources.

Documentation Is Part of the Method

Every scraping workflow should record:

- Source URLs
- Access dates
- Parsing logic
- Known gaps or failures

Undocumented scraping is not reproducible.

What We Emphasize in Practice

- Understand structure before coding
- Start with the simplest approach
- Treat scraping as measurement
- Expect breakage and document it

Discussion

- Where does bias enter web-collected data?
- Which sites feel most trustworthy?
- What assumptions are hidden in page structure?