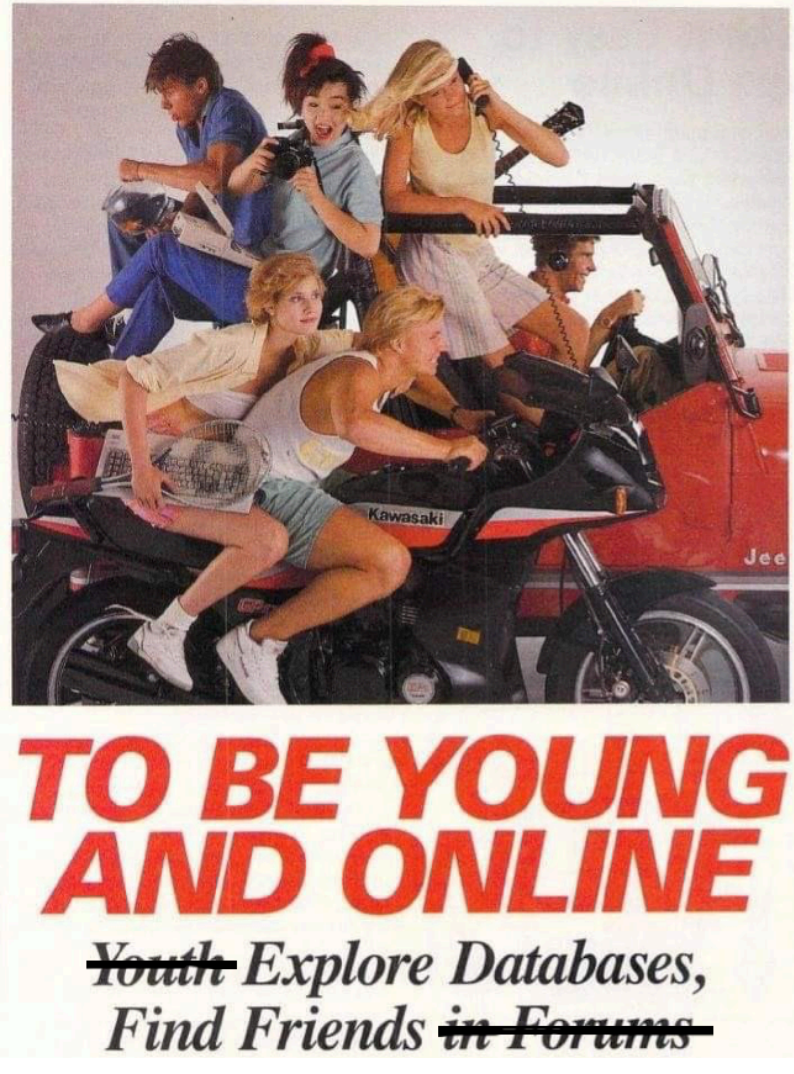
Forschungspraktikum CSS

Session 03 · Ethics & Dynamic Webpages

Dr. Sascha Grehl

2026-11-05



Before We Start:

interim.grehl@gmail.com

Scraping: Law and Ethics

Collection, Storage, and Publication

1. **Collection:** which pages may we access, and how much may we download?
2. **Storage and analysis:** which data may we retain, link, and process?
3. **Publication:** could a dataset, quotation, or figure expose someone?

- Legal requirements and ethical responsibilities apply throughout the project

Important: - I am not a lawyer - Ethical norms evolve (stay tuned!)

1. Collection

1. Collection

Rule #1 is that scraping a website is often **ILLEGAL**

Several issues with scraping:

- Infringement of database rights
- Copyright
- Terms of service
- Data protection (personal data)

Public visibility alone does not settle these questions

1. Collection

Rule #1 is that scraping a website is often **ILLEGAL**

Websites enforce these rules differently:

- Some will let you collect the information you are seeking
- Some will try to limit your impact
- Positively, by offering an API
- Negatively, by placing some constraints on your actions
- Some websites will actively go after you

1. Collection

Rule #1 is that scraping a website is often **ILLEGAL**

BUT

There are ‘research exceptions’ to these rules

In particular, there exists a European directive that specifies the perimeter of the exception for ‘text and data mining’

Directive (EU) 2019/790, Article 3 · German implementation: § 60d UrhG

1. Collection

Rule #1 is that scraping a website is often **ILLEGAL**

BUT

Some (vague) details about this exception

- EU copyright law includes a text and data mining exception for eligible research organisations
- It concerns material to which researchers have **lawful access**, including qualifying subscriptions
- Research status does not provide unrestricted access or permission to bypass controls

Research Exceptions

Rule #1 is that scraping a website is often **ILLEGAL**

BUT

Some (vague) details about this exception

Conditions:

- Access to the data was legal.
- No theft, no cheating

Data protection is key: take measures to store the data securely and prevent leaks or loss

robots.txt

- `robots.txt` communicates rules that crawlers are **asked to follow**
- It does not grant access permission or determine legality by itself

All crawlers are free to use everything:

```
User-agent: *
Allow: /
```

All crawlers are disallowed from accessing the following folders

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /tmp/
Disallow: /junk/
```

For more details, see: Robots Exclusion Protocol, RFC 9309

Ethics

Even if it is allowed, don’t forget to apply good practices:

- Check for a suitable data download or API first
- Identify the research client in the user agent where appropriate
- Respect rate limits and pause between requests
- Test on a small sample and cache responses to avoid repeat downloads
- Stop and reassess when the service blocks or restricts access

2. Storage and Analysis

GDPR

- We usually deal with human subjects
- This makes all actions fall under the remit of the GDPR (General Data Protection Regulation)
- GDPR is a vast, complex code (open to interpretations)

Any data that includes either identifying or sensitive information is subject to GDPR provisions

What Could Go Wrong – An Example

AOL released **657,000 users' search queries** associated with numerical **user IDs**



A Face Is Exposed for AOL Searcher No. 4417749

By Michael Barbaro and Tom Zeller Jr.

The full article

- Search histories contained **local places, names, and other personal clues**
- Reporters combined these clues to identify an individual

A numerical ID did not make the search history anonymous

Re-Identification: Netflix and IMDb

- The Netflix dataset contained movie ratings and rating dates
- Researchers compared rating patterns with public IMDb profiles
- Similar ratings and approximate dates enabled linkage across datasets
- Linked viewing histories could expose sensitive information or invite uncertain inferences

Several ordinary observations can form a distinctive pattern

> *"First, we can immediately find his **political orientation** based on his strong opinions about 'Power and Terror: Noam Chomsky in Our Times' and 'Fahrenheit 9/11.' Strong guesses about his **religious views** can be made based on his ratings on 'Jesus of Nazareth' and 'The Gospel of John.' He did not like 'Super Size Me' at all; perhaps this implies something about his **physical size**? Both items that we found with predominantly **gay themes**, 'Bent' and 'Queer as folk' were rated one star out of five. He is a cultish follower of 'Mystery Science Theater 3000.' This is far from all we found about this one person, but having made our point, we will spare the reader further lurid details."* - Narayanan & Shmatikov 2008: 16

Personal Data

Information about an identified or identifiable living person includes:

- Names, email addresses, and IP addresses
- Usernames or handles and the posts linked to them
- Identifiable faces and voice recordings
- Combinations of contextual details, such as occupation, workplace, and origin

Warning: Removing only the name may still leave the person identifiable (see the AOL example below)

GDPR, Article 4 and Recital 26

Legal Bases for Processing

Article 6 lists several possible bases:

- Consent
- A contract or a legal obligation
- Protection of vital interests
- A task in the public interest or exercise of official authority
- Legitimate interests, subject to conditions and limits

A research purpose alone does not establish a legal basis.

GDPR, Articles 6 and 89

Sensitive Personal Data

- Racial or ethnic origin
- Political opinions, religious or philosophical beliefs
- Trade union membership
- Health, sex life, or sexual orientation
- Genetic data and biometric data used to uniquely identify a person

Processing requires an applicable **Article 9 exception**, as well as an Article 6 legal basis

Most likely:

> “Paragraph 1 shall not apply if one of the following applies: … (e) processing relates to personal data which are manifestly made public by the data subject” – GDPR, Article 9, 2(e)

GDPR, Article 9

Pseudonymisation and Anonymisation

- Replacing names with IDs and keeping a lookup table is **pseudonymisation**
- Store that lookup table separately and restrict access
- Pseudonymised data remain personal data when re-identification is possible
- Anonymisation must account for indirect identifiers and linkage with other sources (see again AOL example)
- Exact quotations can make a post searchable even after removing its username

Secure Storage and Data Minimisation

- Collect only the fields needed for the research question
- Use institutionally approved storage with encryption and restricted access
- Keep identifiers separate from analysis data where feasible
- Define retention and deletion rules, including backups
- Check authorisation and safeguards before sharing data with third parties

Ethical Research Using AI

- Once you have collected data, you need to decide what to do with it
- The AI Research Act provides a framework for the responsible use of AI in research
- It sorts AI applications into risk categories and specifies requirements for each category
- Check if your research project falls under the scope of the AI Research Act!

3. Publication

Ethical Responsibilities Beyond Access

- What audience did people expect when they posted?
- Could collection or publication expose them to embarrassment, discrimination, or other harm?
- Does linking sources reveal more than each source reveals on its own?
- Could someone locate a person using our quotations or released data?

Public availability does not establish informed consent to every research use.

Salganik, *Bit by Bit*, Chapter 6: Ethics

Reproducibility and Responsible Sharing

- Share code, documentation, and collection procedures where possible
- Consider aggregated results or synthetic examples when individual records create risks
- Assess whether quotations or rare combinations allow re-identification
- Use controlled access when open release is inappropriate and sharing is permitted
- Document the reasons for restrictions so others understand what they can reproduce

Back to Scraping

Going further

Last Time

In ~70% of the cases, in order to scrape a site all you need to do is:

```
1 read_html("PAGE")
```

- But there are cases where this is not enough

User-Agent

Browsers send a `User-Agent` (see robots.txt) header with every request. The default for R is `NULL`

Honest

But we can set it to something that identifies our data collector, for example:

```
1 httr::user_agent("DataCollector/1.0 (contact: studi@studserv.uni-leipzig.de)")
```

Not So Honest

Sometimes, websites do not like our data collector and will block it. In these cases, we can pretend to be a browser:

```
1 httr::user_agent("Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:155.0) Gecko/20100101
   Firefox/155.0")
```

Tip: Use this website to find out what your browser sends

Cookies & Forms

Sometimes that won't be enough, because you'll need cookies or have to log in

In this case, you can use the `rvest` package to handle forms and cookies. For example, if you want to log in to a website, you can do something like this:

```
library(rvest)
library(httr)

url <- "https://quotes.toscrape.com/login"
s <- session(url)
forms <- html_form(s)
forms
```

```
[[1]]
<form> '<unnamed>' (POST https://quotes.toscrape.com/login)
  <field> (hidden) csrf_token: GOIMWjNVkFaUdZBSp...
  <field> (text) username:
  <field> (password) password:
  <field> (submit) : Login
```

Submitting a Form

Now we can fill in the form and submit it:

```
login_form <- forms[[1]]

login_form <- html_form_set(
  login_form,
  username = "demo",
  password = "demo"
)

s <- session_submit(s, login_form)
```

And verify that the login worked

```
s |>
  html_element("a[href='/logout']") |>
  html_text2()
```

```
[1] "Logout"
```

The Cookie

To see what cookie was set after logging in, you can use the `httr::cookies()` function:

```
1 httr::cookies(s)
```

```
                          domain  flag path secure expiration    name
1 #HttpOnly_quotes.toscrape.com FALSE    /  FALSE        Inf session

value
1
eyJjc3JmX3Rva2VuIjoiR09JTVdqTlZrRmFVZFpCU3BUcVB3dHZmZWxBbktzaENpeUp4WHJZZ2JETHpSRWN1SG1vUSIsI
nVzZXJuYW1lIjoiZGVtbyJ9.auxTiQ.B9L9jLffgxz5jyOfltgVgdNZBec
```

Going Further

```r
s <- session_jump_to(
  s,
  "https://quotes.toscrape.com/page/2/"
)
s |> read_html()
```

```
{html_document}
<html lang="en">
[1] <head>\n<meta charset="UTF-8">\n<title>Quotes to Scrape</title>\n<link re ...
[2] <body>\n    <div class="container">\n        <div class="row header-box"> ...
```

Scraping Dynamic Webpages

Dynamic Webpages

A growing number of websites are dynamic, meaning that the content is generated by JavaScript code running in the browser (e.g., scrolling, clicking, or other interactions)

These pages are often not fully rendered when you request them with `read_html()` or `httr::GET()`, because the JavaScript code has not yet run. As a result, you may not see the content you expect.

Example

```
read_html("https://quotes.toscrape.com/js") |>
  html_elements(".text")
```

```
{xml_nodeset (0)}
```

Headless Browser

- Since we need to run the JavaScript code, in order to scrape dynamic pages, we need a **headless browser**
- That is, a full web browser that operates without a graphical user interface (GUI)
- Therefore we first need to install a web browser (e.g., Firefox, Chrome, Edge) that can run in a headless mode
- Then we need to control it from R or Python (via a WebDriver)

Browser	WebDriver
Mozilla Firefox	GeckoDriver
Google Chrome	ChromeDriver
Microsoft Edge	EdgeDriver
Safari	SafariDriver

Headless Browser II

RSelenium vs Selenium

- In R, we can use the `RSelenium` package to control a headless browser
- In Python, we can use the `selenium` package to control a headless browser
- With `reticulate`, we can run Python code from R, and therefore use the `selenium` package from R

So why do we use `reticulate` to control `selenium` instead of using `RSelenium` directly?

- With RSelenium we have to run the browser in a separate process, which can be cumbersome and error-prone
- With `reticulate` we can run the browser in the same process as R, which is more convenient and less error-prone

Headless Browser - Setup

In R, this is often done using “Selenium”. To do so we will use the `reticulate` package to run Python code from R

```
1 library(reticulate)
```

Then we need to install `miniconda`, create a conda environment, and install the `selenium` package in it:

```
1 install_miniconda() # Installs Python and conda if not already installed
2 conda_create("toolbox_env") # Creates an environment called toolbox_env
3 conda_install("toolbox_env", "selenium") # Installs selenium in the environment
```

- Sometimes `Docker` is used to run Selenium in a container, but we will not cover that here. See this explanation by Chris Bail.
  - However: `Docker` will not work with the Scientific Computing Cluster at the University of Leipzig, because it requires root privileges

Running a Headless Browser - Theory

First, we need to load the `reticulate` package and activate the conda environment that contains Selenium:

```
use_condaenv("toolbox_env", required = TRUE)
```

Running a Headless Browser - Practice

My system (Linux) does not have a default temporary directory that is writable by the browser, so I need to create one and set the `TMPDIR` environment variable to point to it:

```
dir.create( path.expand("~/selenium-tmp"), showWarnings = FALSE )

Sys.setenv( TMPDIR = path.expand("~/selenium-tmp"))

use_condaenv("toolbox_env", required = TRUE)
```

I had to **use an LLM** to come up with this solution!

Do the same! Each operating system has its own quirks and you may need to adapt the solution to your system

Scraping with Python and Selenium - Theory

Now we run `python` code:

```
1 from selenium import webdriver # load package
2
3 driver = webdriver.Firefox() # Start Firefox
4
5 driver.get("https://quotes.toscrape.com/js/") # Navigate to website
```

Scraping with Python and Selenium - Practice

Now we run `python` code:

```python
from selenium import webdriver # load packages
from selenium.webdriver.firefox.service import Service
from selenium.webdriver.firefox.options import Options

# Use my installed geckodriver
service = Service("/home/sgrehl/miniconda3/envs/toolbox_env/bin/geckodriver")

# Tell the browser to run in headless mode (no GUI)
options = Options()
options.add_argument("-headless")

driver = webdriver.Firefox( # Start Firefox
    service=service,
    options=options
)

driver.get("https://quotes.toscrape.com/js/") # Navigate to website
```

Store the Rendered HTML

```
# Get the page source (HTML) after the JavaScript has run
html = driver.page_source

# save the HTML to a file
output_path = "data/raw/page_01.html"
with open(output_path, "w", encoding="utf-8") as f:
            f.write(html)
```

```
8925
```

Use the Cached HTML

```
# load the cached HTML file
html <- read_html(file.path("data","raw","page_01.html"))
html  |>
  html_elements(".text")
```

```
{xml_nodeset (10)}
 [1] <span class="text">“The world as we have created it is a process of our  ...
 [2] <span class="text">“It is our choices, Harry, that show what we truly ar ...
 [3] <span class="text">“There are only two ways to live your life. One is as ...
 [4] <span class="text">“The person, be it gentleman or lady, who has not ple ...
 [5] <span class="text">“Imperfection is beauty, madness is genius and it's b ...
 [6] <span class="text">“Try not to become a man of success. Rather become a  ...
 [7] <span class="text">“It is better to be hated for what you are than to be ...
 [8] <span class="text">“I have not failed. I've just found 10,000 ways that  ...
 [9] <span class="text">“A woman is like a tea bag; you never know how strong ...
[10] <span class="text">“A day without sunshine is like, you know, night.”</s ...
```

Navigating the Website

```
1 button = driver.find_element(By.LINK_TEXT, "next")
2 button = driver.find_element(By.CSS_SELECTOR, ".next a")
3
4 button.click()
```

More examples in the practical!

Full Looping Example

```python
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

page_nr = 1

while True:
    html = driver.page_source

    # Save the current page
    output_path = f"data/raw/page_{page_nr:02d}.html"
    with open(output_path, "w", encoding="utf-8") as f:
        f.write(html)

    # Find the Next button
    buttons = driver.find_elements(By.CSS_SELECTOR, ".next a")

    # Stop if there is no Next button
    if not buttons:
        break

    # Remember the current page element
    current_page = driver.find_element(By.TAG_NAME, "body")

    # Click Next
    buttons[0].click()

    # Wait until the previous page is unloaded
    WebDriverWait(driver, 10).until(
        EC.staleness_of(current_page)
    )

    page_nr += 1
```

```
8925
True
12854
True
8840
True
8320
True
8313
True
8716
```

Next Time

- OCR
- Whisper

Merci!

sascha.grehl@uni-leipzig.de

Universität Leipzig