Computational Political Science

Session 9

David Broska
Zeppelin University

April 13, 2021

Looking ahead

What we did

So far we focused on how methods for quantitative text analysis work.

We only briefly looked at applications of those methods.

The road ahead

However, we need to have a *deeper* understanding of how QTA helps us answer *substantive research questions*

In the next sessions, we will therefore evaluate how useful QTA methods are (as opposed to understanding how they work)

Motivation

While methodological innovations are quite common, articles rarely showcase strong explanatory power of QTA for substantive research questions

E.g. an application challenged a widely established belief or revealed new insights for a specific research area in political science/sociology

Session 10

Text	Presenter
How Censorship in China Allows Government Criticism but Silences Collective Expression (King et al 2014)	T.O.
Rhetorics of Radicalism (Karell and Freedman 2019)	A.V., M.M.
Racialized Discourse in Seattle Rental Ad Texts (Kennedy et al 2020)	N.R.
Whose Ideas Are Worth Spreading? The Representation of Women and Ethnic Groups in TED Talks (Schwemmer and Jungkunz 2019)	V.O.
The Geometry of Culture: Analyzing the Meanings of Class through Word Embeddings (Kozlowski et al 2019)	D.B.

Note:

These are good examples of articles that addresses substantial social science questions

However, you may also present a different article if it better fits your research interest

You find the literature on ILIAS

Research Design

A research design typically has the following structure and answers implicit questions

1. Background/literature review: What do we know already?
2. Research question(s): What are you going to try to learn?
3. Data collection strategies: What kind of evidence are you going to collect and how will you collect it?
4. Data analysis strategies: How does that evidence enable us to draw conclusions?
5. Potential impact and relevance of the study: What might those conclusions be and why do they matter?
6. Limitations and further research: What are the limitations of what you are going to do? What have you done to mitigate these limitations? What more could be done with extra time and/or resources?
7. References / bibliography

You can use this structure for the presentation of the reading (session 10) and your own intended research (session 11)

Outline for today (session 9)

1. **How to retrieve data from the web?**

2. **Features of the internet**

3. **HTML and CSS**

4. **XPath and CSS Selectors**

5. **APIs**

6. **Bias in social media data**

7. **Coding example**

    - Scraping the course website on GitHub with CSS selectors
    - How to use the Twitter API and classify (dis)approval tweets

8. **Coding exercise**

    - Scrape an online bookshop

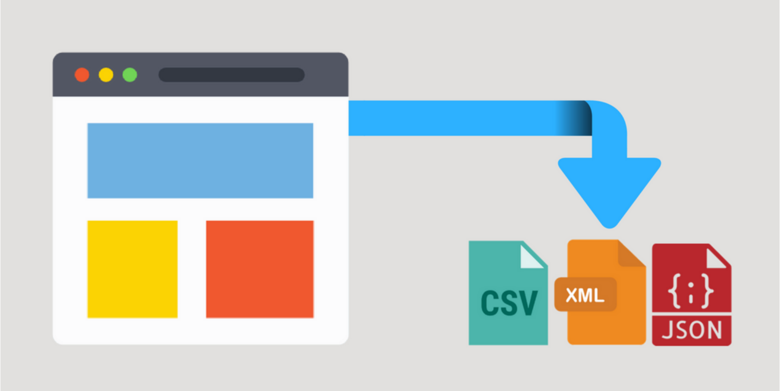
Course schedule

Session	Date	Topic	Assignment	Due date
1	Feb 02	Overview and key concepts	-	-
2	Feb 09	Preprocessing and descriptive statistics	Formative	Feb 22 23:59:59
3	Feb 16	Dictionary methods	-	-
4	Feb 23	Machine learning for texts: Classification I	Summative 1	Mar 08 23:59:59
5	Mar 02	Machine learning for texts: Classification II	-	-
6	Mar 09	Supervised and unsupervised scaling	Summative 2	Mar 22 23:59:59
7	Mar 16	Similarity and clustering	-	-
8	Mar 23	Topic models	Summative 3	Apr 12 23:59:59
-	-	*Break*	-	-
9	Apr 13	*Retrieving data from the web*	-	-
10	Apr 20	Published applications	-	-
11	Apr 27	Project Presentations	-	-

What is webscraping?

An increasing amount of data is available on the web

- Speeches, biographical information …
- Social media data, press releases …
- Geographic information, conflict data…



However, these data are often provided in an *unstructured format*

Web scraping is the process of automatically extracting content from the web and transforming it into a structured dataset

How to get data from the internet with R

1. Screen scraping

Extract data from source code of website with HTML parser and/or regular expressions

- `rvest` package in R for screen scraping

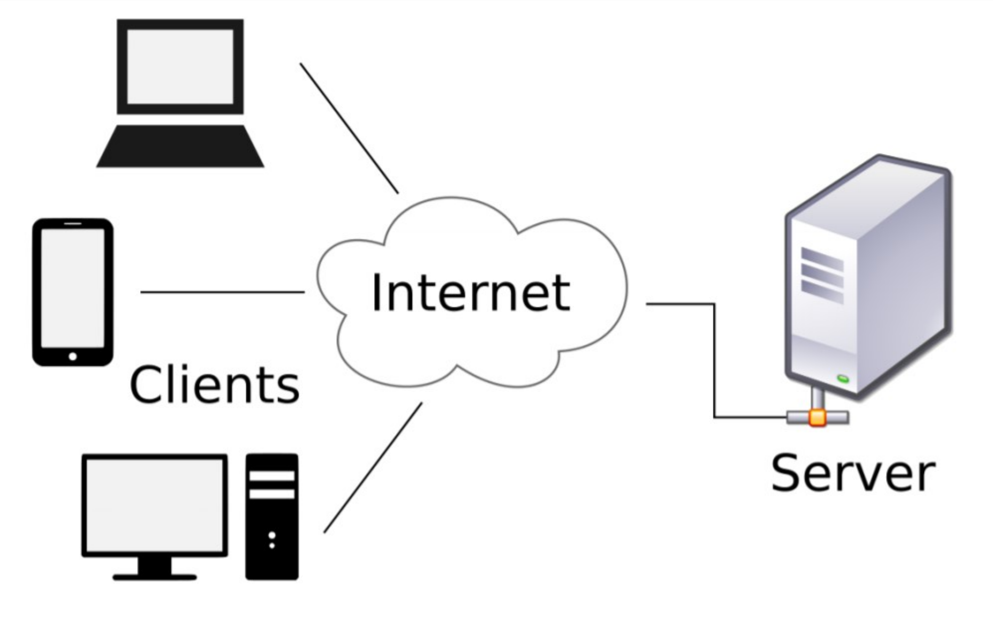
2. Web APIs

A set of structured http requests that return JSON or XML data

- `httr` package to construct API requests
- Packages specific to each API

Key features of the internet

Client-server model



1. User computer tablet, phone, etc. make request to server. Depending on what you want to get, the request might be

- HTTP(S): Hypertext Transfer Protocol
- SMTP: Simple Mail Transfer Protocol
- FTP: File Transfer Protocol

2. Server returns response

HTTP request and response

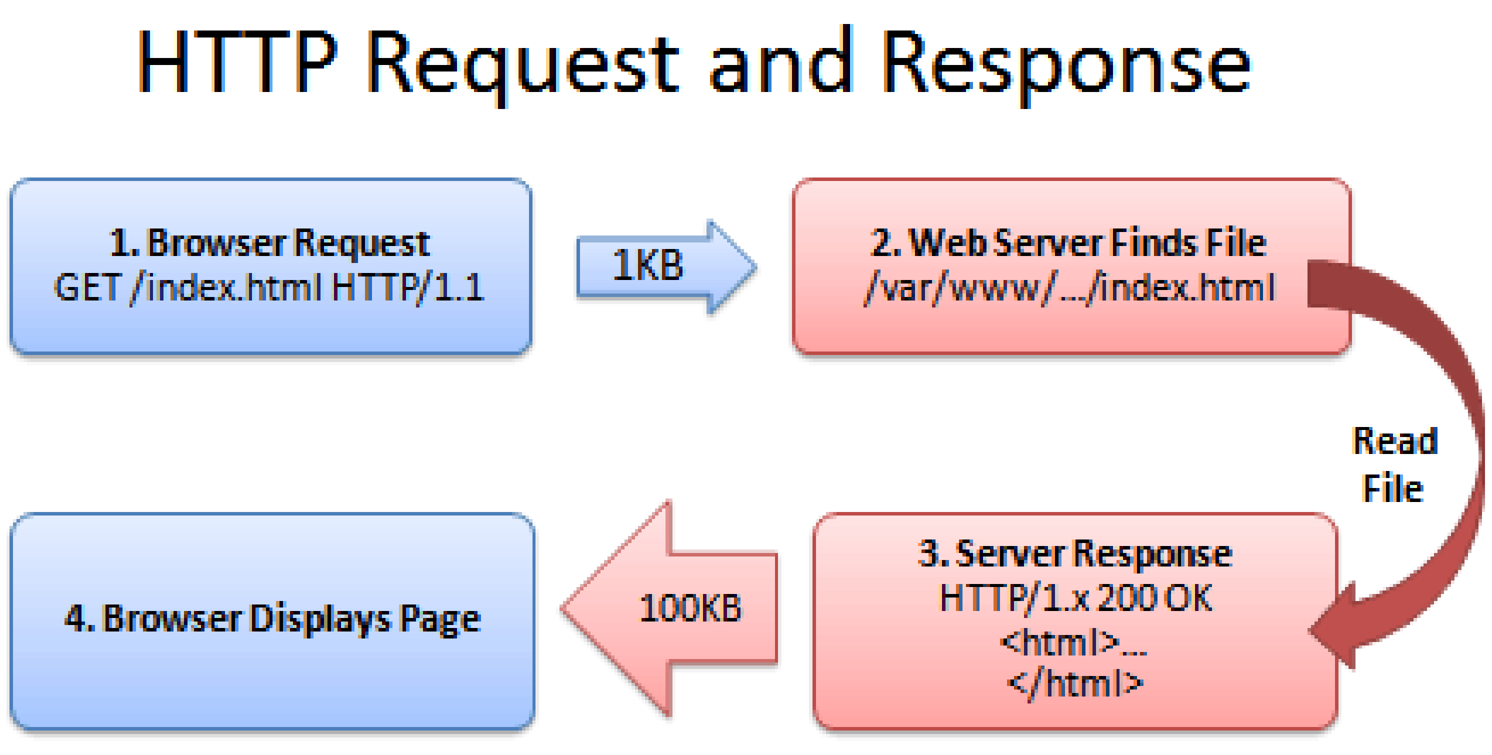
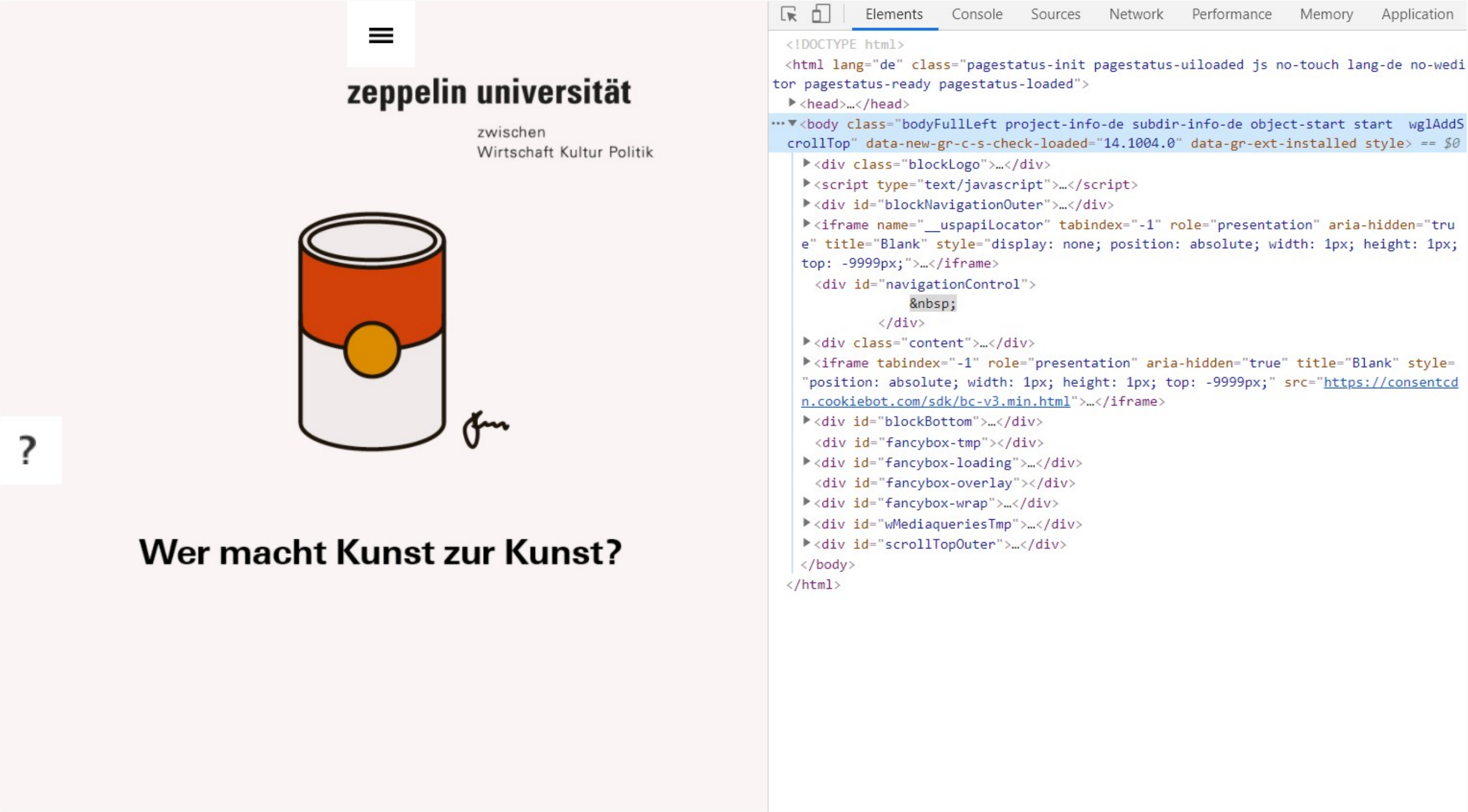


Figure from StackOverflow

Example: zu.de

Press `Ctrl+Shift+I` in your Google Chrome browser to see the source code of a website

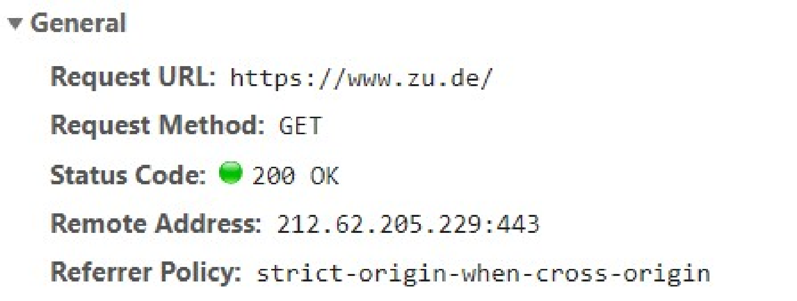


```
<!DOCTYPE html>
<html lang="de" class="pagestatus-init pagestatus-uiloaded js no-touch lang-de no-editor pagestatus-ready pagestatus-loaded">
 ▶<head>…</head>
…▼<body class="bodyFullLeft project-info-de subdir-info-de object-start start  wglAddScrollTop" data-new-gr-c-s-check-loaded="14.1004.0" data-gr-ext-installed style> == $0
   ▶<div class="blockLogo">…</div>
   ▶<script type="text/javascript">…</script>
   ▶<div id="blockNavigationOuter">…</div>
   ▶<iframe name="__uspapiLocator" tabindex="-1" role="presentation" aria-hidden="true" title="Blank" style="display: none; position: absolute; width: 1px; height: 1px; top: -9999px;">…</iframe>
    <div id="navigationControl">
            &nbsp;
        </div>
   ▶<div class="content">…</div>
   ▶<iframe tabindex="-1" role="presentation" aria-hidden="true" title="Blank" style="position: absolute; width: 1px; height: 1px; top: -9999px;" src="https://consentcdn.cookiebot.com/sdk/bc-v3.min.html">…</iframe>
   ▶<div id="blockBottom">…</div>
    <div id="fancybox-tmp"></div>
   ▶<div id="fancybox-loading">…</div>
    <div id="fancybox-overlay"></div>
   ▶<div id="fancybox-wrap">…</div>
   ▶<div id="wMediaqueriesTmp">…</div>
   ▶<div id="scrollTopOuter">…</div>
  </body>
</html>
```

We can use the `rvest` package to parse data from the website's source code!

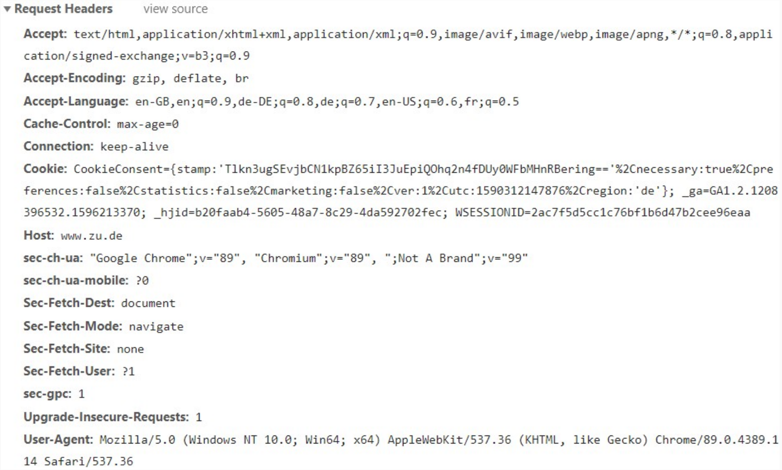
Example: zu.de

General header



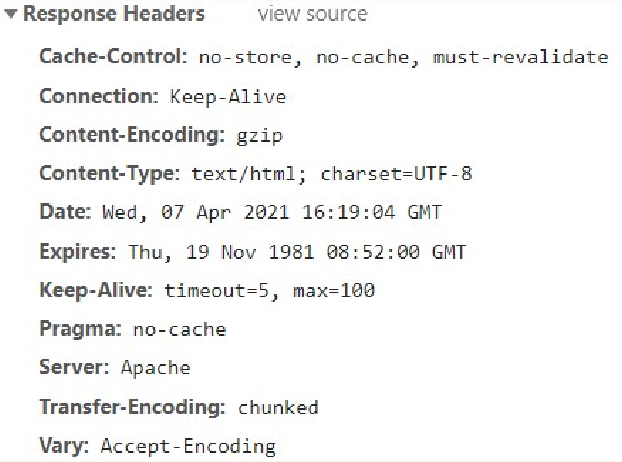
```
▾ General
    Request URL: https://www.zu.de/
    Request Method: GET
    Status Code: ● 200 OK
    Remote Address: 212.62.205.229:443
    Referrer Policy: strict-origin-when-cross-origin
```

Request header



```
▾ Request Headers     view source
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appli
    cation/signed-exchange;v=b3;q=0.9
    Accept-Encoding: gzip, deflate, br
    Accept-Language: en-GB,en;q=0.9,de-DE;q=0.8,de;q=0.7,en-US;q=0.6,fr;q=0.5
    Cache-Control: max-age=0
    Connection: keep-alive
    Cookie: CookieConsent={stamp:'Tlkn3ugSEvjbCN1kpBZ65iI3JuEpiQOhq2n4fDUy0WFbMHnRBering=='%2Cnecessary:true%2Cpre
    ferences:false%2Cstatistics:false%2Cmarketing:false%2Cver:1%2Cutc:1590312147876%2Cregion:'de'}; _ga=GA1.2.1208
    396532.1596213370; _hjid=b20faab4-5605-48a7-8c29-4da592702fec; WSESSIONID=2ac7f5d5cc1c76bf1b6d47b2cee96eaa
    Host: www.zu.de
    sec-ch-ua: "Google Chrome";v="89", "Chromium";v="89", ";Not A Brand";v="99"
    sec-ch-ua-mobile: ?0
    Sec-Fetch-Dest: document
    Sec-Fetch-Mode: navigate
    Sec-Fetch-Site: none
    Sec-Fetch-User: ?1
    sec-gpc: 1
    Upgrade-Insecure-Requests: 1
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.1
    14 Safari/537.36
```

Response header



```
▾ Response Headers     view source
    Cache-Control: no-store, no-cache, must-revalidate
    Connection: Keep-Alive
    Content-Encoding: gzip
    Content-Type: text/html; charset=UTF-8
    Date: Wed, 07 Apr 2021 16:19:04 GMT
    Expires: Thu, 19 Nov 1981 08:52:00 GMT
    Keep-Alive: timeout=5, max=100
    Pragma: no-cache
    Server: Apache
    Transfer-Encoding: chunked
    Vary: Accept-Encoding
```

Please refer to this guide on viewing HTTP headers

HTML and CSS

HTML and beyond

Hypertext Markup Language (HTML)

- HTML displays mostly static content
- Many contents of dynamic webpages cannot be found in HTML, e.g. Google Maps

⇒ Understanding what is static and dynamic in a webpage is a crucial first step

Cascading Style Sheets (CSS)

- Style sheet language which describes formatting of HTML components
- CSS is useful for webscraping because there are CSS-based selectors for HTML elements

Javascript (JS)

- Adds functionalities to websites, e.g. change content/structure after website has been loaded
- Javascript on websites usually makes webscraping more difficult

A simple HTML file

Let's create a simple HTML page

```html
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <h3>My First Heading</h1>
    <p>My first paragraph.</p>
  </body>
</html>
```

It will look like this:

My first heading

My first paragraph.

Slightly more features

```html
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <h1>Heading of the first division</h1>
    <p>A first paragraph.</p>
    <p>A second paragraph with some <b>formatted</b> text.</p>
    <p>A third paragraph with a <a href="http://www.zu.de">hyperlink</a>.</p>
  </body>
</html>
```

It will look like this:

A first paragraph.

A second paragraph with some **formatted** text.

A third paragraph with a hyperlink.

With some content divisions

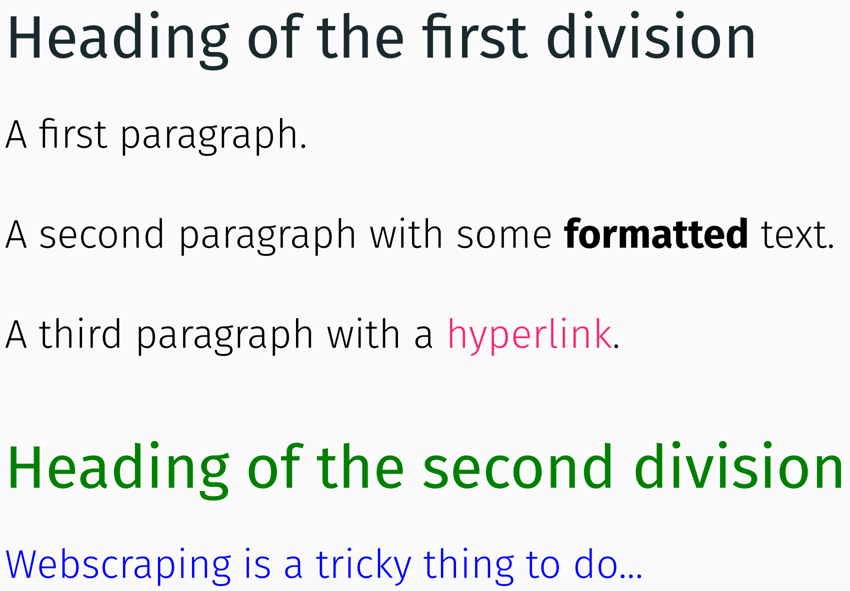
```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <div>
      <h3>Heading of the first division</h3>
      <p>A first paragraph.</p>
      <p>A second paragraph with some <b>formatted</b> text.</p>
      <p>A third paragraph with a <a href="http://www.zu.de">hyperlink</a>.</p>
    </div>
    <div>
      <h3>Heading of the second division</h3>
      <p>Another paragraph with some text.</p>
    </div>
  </body>
</html>
```

Adding some simple CSS

```html
<!DOCTYPE html>
<html>
  <head>
    <style>
      .text-about-web-scraping {color: blue;}
      .division-two h3 {color: green;}
    </style>
  </head>
  <body>
    <div>
      <h3>Heading of the first division</h3>
      <p>A first paragraph.</p>
      <p>A second paragraph with some <b>formatted</b> text.</p>
      <p>A third paragraph with a <a href="http://www.zu.de">hyperlink</a>.</p>
    </div>
    <div class="division-two">
      <h3>Heading of the second division</h3>
      <p class="text-about-web-scraping">Webscraping is a tricky thing...</p>
    </div>
  </body>
</html>
```

Et voilà ...

This is how our HTML page looks like with formatting tags and CSS styles:



Heading of the first division

A first paragraph.

A second paragraph with some **formatted** text.

A third paragraph with a hyperlink.

Heading of the second division

Webscraping is a tricky thing to do...

Identifying elements with CSS and XPath

Identifying elements via CSS selector

Selecting by tag-name

Exemplary html code: `<h3>Some text</h3>`

Selector: `h3`

Selecting by class

Exemplary html code: `<div class = 'itemdisplay'>Some text</div>`

Selector: `.itemdisplay`

Selecting by id

Exemplary html code: `<div id = 'maintitle'>Some text</div>`

Selector: `#maintitle`

Identifying elements via CSS selector

Selecting by tag-name

Exemplary html code: `<h3>Some text</h3>`

Selector: `h3`

Selecting by class

Exemplary html code: `<div class = 'itemdisplay'>Some text</div>`

Selector: `.itemdisplay`

Selecting by id

Exemplary html code: `<div id = 'maintitle'>Some text</div>`

Selector: `#maintitle`

XPath basic syntax

`/` selects from the root node, e.g. `/html/body/div[2]/p[1]`

`//` selects specific nodes from the document, e.g. `//div[2]/p[1]`

`//div/*` Selects all nodes which are immediate children of a div node

`//div/p[last()]` selects the last paragraph nodes which are children of all div nodes

`//div[@*]` selects all division nodes which have any attribute

`//div[@class]` selects all division nodes which have a class attribute

`//div[@class='division-two']` selects all division nodes which have a class attribute with name "division-two"

`//*[@class='division-two']` selects any node with a class attribute with name "division-two"

See w3schools.com for reference and full details

XPath vs CSS selector

Selector type	CSS Selector	XPath
By tag	`"h1"`, `"p"`	`"//h1"`, `"//p"`
By class	`".division-two"`	`"//*[@class='division-two']"`
By id	`"exemplary-id"`	`"//*[@id='exemplary-id']"`
By tag with class (or id)	`"div.division-two"`	`"//div[@class='division-two']`
Tag structure (p as a child of div)	`"div > p"`	`"//div/p"`
Tag strucure (p which is a second child of the div node with class name division-two)	`"div.division-two>p:nth-of-type(2)"`	`"//div[@class='division-two']/p[2]"`

See this guide and this converter app for XPath and CSS

Using rvest

Recall our simple HTML with some CSS:

Heading of the first division

A first paragraph.

A second paragraph with some **formatted** text.

A third paragraph with a hyperlink.

Heading of the second division

Webscraping is a tricky thing to do...

Let's first make R recognize the HTML code

```
page <- read_html('<!DOCTYPE html>
<html>
  <head>
    <style>
    .text-about-web-scraping{color:blue;
    .division-two h3{color: green;}
    </style>
  </head>
  <body>
    <div>
      <h3>Heading of the first division<
      <p>A first paragraph.</p>
      <p>A second paragraph with some <b
      <p>A third paragraph with a <a hre
    </div>
    <div class="division-two">
      <h3>Heading of the second division
      <p class="text-about-web-scraping'
    </div>
  </body>
```

Parsing HTML with rvest

```
page                                  # let's look at the parsed HTML file
```

```
## {html_document}
## <html>
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">\n<style
## [2] <body>\n    <div>\n      <h3>Heading of the first division</h3>\n      <p>A first p
```

Using CSS selectors

```
page %>% html_nodes(css= 'h3') %>%
  html_text()
```

```
## Heading of the first division
## Heading of the second division
```

```
page %>% html_node(css= 'a') %>%
  html_attr('href')
```

```
## http://www.zu.de
```

Using XPath

```
page %>% html_nodes(xpath= '//h3') %>%
  html_text()
```

```
## Heading of the first division
## Heading of the second division
```

```
page %>% html_nodes(xpath= '//a') %>%
  html_attr('href')
```

```
## http://www.zu.de
```

Scraping with RSelenium

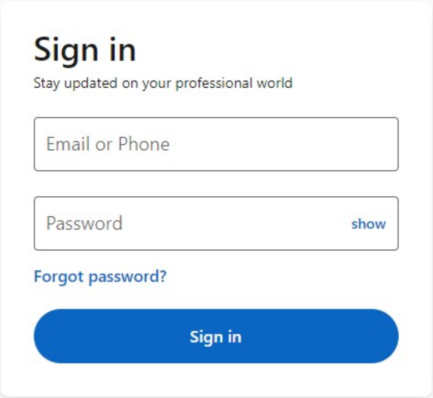
Why RSelenium?

Many websites cannot be scraped as easily as a simple HTML

Authentication

Sometimes you have to be logged in to access the content of a website

Thus, we need a way to fill the authentication form!

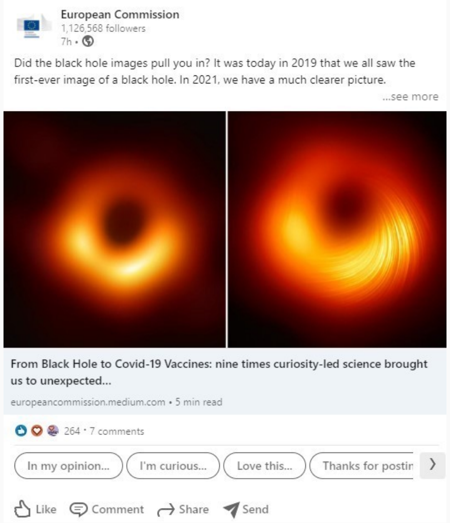


Sign in

Stay updated on your professional world

Dynamic contents

Some websites do not load all content at once but only if you scroll down, e.g. on social media



European Commission

1,126,568 followers

7h •

Did the black hole images pull you in? It was today in 2019 that we all saw the first-ever image of a black hole. In 2021, we have a much clearer picture.

...see more

From Black Hole to Covid-19 Vaccines: nine times curiosity-led science brought us to unexpected...

europeancommission.medium.com • 5 min read

264 · 7 comments

Selenium

Selenium is a technology for browser automation. RSelenium is a R binding for Selenium

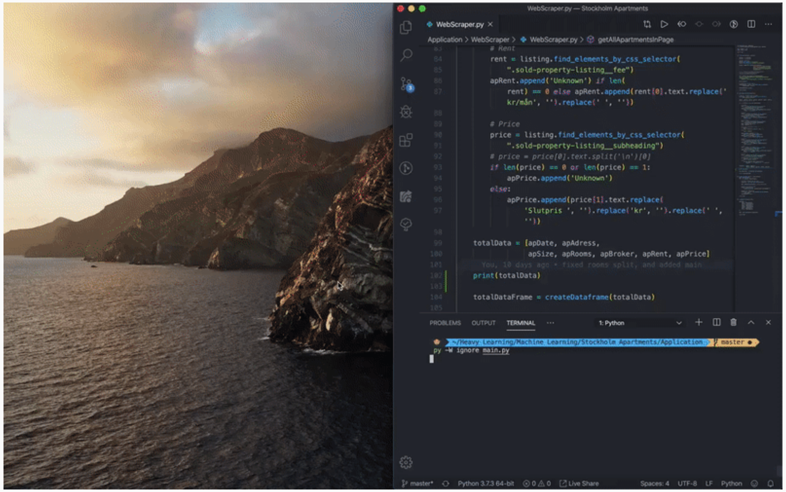
Idea

Launch a browser session and all communication will be routed through that browser session

We scrape websites loaded in that browser session

How it works

1. The webscraper opens a browser window,
2. navigates to apartments listings
3. downloads all apartment-listings on that page
4. and navigates to the next page to repeat the process



WebScraper.py — Stockholm Apartments
WebScraper.py
Application > WebScraper > WebScraper.py > getAllApartmentsInPage
You, 10 days ago • fixed rooms split, and added main
PROBLEMS
OUTPUT
TERMINAL
1: Python
master*
Python 3.7.3 64-bit
0 ⚠ 0
Live Share
Spaces: 4
UTF-8
LF
Python

```python
            # Rent
            rent = listing.find_elements_by_css_selector(
                ".sold-property-listing__fee")
            apRent.append('Unknown') if len(
                rent) == 0 else apRent.append(rent[0].text.replace('
                kr/mån', '').replace(' ', ''))

            # Price
            price = listing.find_elements_by_css_selector(
                ".sold-property-listing__subheading")
            # price = price[0].text.split('\n')[0]
            if len(price) == 0 or len(price) == 1:
                apPrice.append('Unknown')
            else:
                apPrice.append(price[1].text.replace(
                    'Slutpris ', '').replace('kr', '').replace(' ',
                    ''))

        totalData = [apDate, apAdress,
                     apSize, apRooms, apBroker, apRent, apPrice]

        print(totalData)

        totalDataFrame = createDataframe(totalData)
```

```
~/Heavy Learning/Machine Learning/Stockholm Apartments/Application  master ●
py -W ignore main.py
```

Selenium drivers

There are two general strategies to run Selenium drivers

1. Normal browsers

   - Chrome
   - Firefox
   - etc.

2. Headless browser (will not display website)

   - Allows to set up the browser in a situation where you do not have a visual device (i.e. Crawler on the cloud) or do not need an open browser window
   - Common headless browser: phantomJS
   - Selenium in Python allows to also run Chrome or Firefox in headless mode

Key functions of RSelenium I

Load RSelenium package

```
library("RSelenium")
```

Create browser instance

```
rD <- rsDriver(browser=c("firefox"))
driver <- rD$client
```

Navigate to website

```
driver$navigate("https://www.zu.de")
```

Find element on website

```
some_element <- driver$findElement(using = "xpath", value = "...")
```

Key functions of RSelenium II

Click on element

```
some_element$clickElement()
```

Type text into box/element

```
search_box <- driver$findElement(using = "xpath", value = "...")
search_box$sendKeysToElement(list("some text"))
```

Press enter key

```
search_box$sendKeysToElement(list(key = "enter"))
```

APIs

APIs

- API: Application Programming Interface
- Provides access to data!
- In web APIs, a set of structured HTTP requests can return data in a lightweight format e.g. JSON or XML
- The API user sends a request to the API (e.g. with a software such as R) and the API returns data from the API provider's database
- We will use the `rtweet` package to to access the Twitter API from R

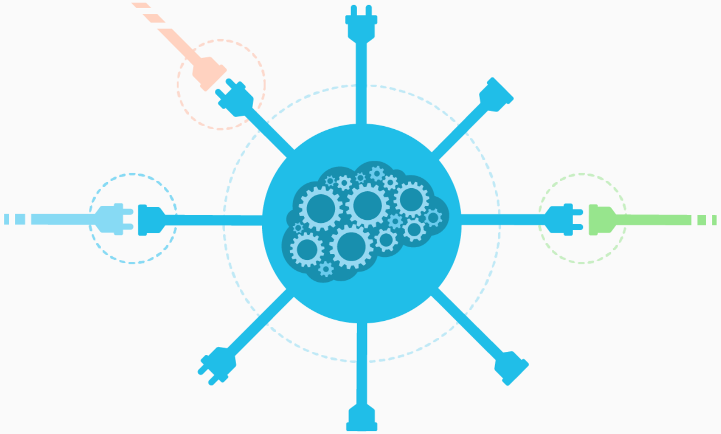


Figure from seamgen.com

Why APIs?

Advantages

- Cleaner data collection: Avoid malformed HTML, no legal issues, clear data structures, more trust in data collection...
- Standardized data access procedures: Transparency, replicability
- Robustness: Benefits from "wisdom of the crowds"

Disadvantages

- Not always available
- Dependency on API providers
- Rate limits

Packages that wrap existing APIs

rtweet

- Excellent, well-maintained R package that wraps the Twitter API
- Twitter search API is limited to recent tweets, so you cannot go back more than a few days
- rtweet provides functions to retrieve tweets as well as user profiles and social graphs (friends, followers)

tubeR

- Medium quality R package that wraps Googles Youtube API
- It allows you to fetch video statistics, comments, and statistics on comments

Other packages

- `wbstats` for easy access to World Bank data
- `WikipediR`, `wikipediatrend`, or `WikidataR` for data on Wikipedia
- A more extensive list is in this CRAN view

Twitter APIs

Two different methods to collect Twitter data

1. REST API
   - Queries for specific information about users and tweets
   - Search recent tweets
   - Examples: User profile, list of followers and friends, tweets generated by a given user ("timeline"), users lists, etc.
2. Streaming API
   - Connect to the "stream" of tweets as they are being published
   - Three streaming APIs:
     - Sample stream: 1% random sample of tweets
     - Filter stream: tweets filtered by keywords (when volume reaches 1% of all tweets, it will also return a random sample)
     - Geo stream: tweets filtered by location

Twitter APIs

- Tweets can only be downloaded in real time, historical data is generally much harder to obtain (exceptions: last seven days or user timelines, where ~ 3,200 most recent tweets are available)
- Very recent special access for researchers allows to obtain more historical data

Bias in social media data

Biases in sampling

Morstatter et al (2013) "Is the Sample Good Enough? Comparing Data from Twitter's Streaming API with Twitter's Firehose"

- 1% random sample from Streaming API is not truly random
- Less popular hashtags, users, topics... less likely to be sampled
- But for keyword-based samples, bias is not as important

González-Bailón et al (2014) "Assessing the bias in samples of large online networks"

- Small samples collected by filtering with a subset of relevant hashtags can be biased
- Central, most active users are more likely to be sampled
- Data collected via search (REST) API more biased than those collected with Streaming API

Biases in social media data

Population bias

Sociodemographic characteristics are correlated with presence on social media

Self-selection within samples

Partisans are more likely to post about politics (Barberá & Rivero 2014)

Proprietary algorithms for public data

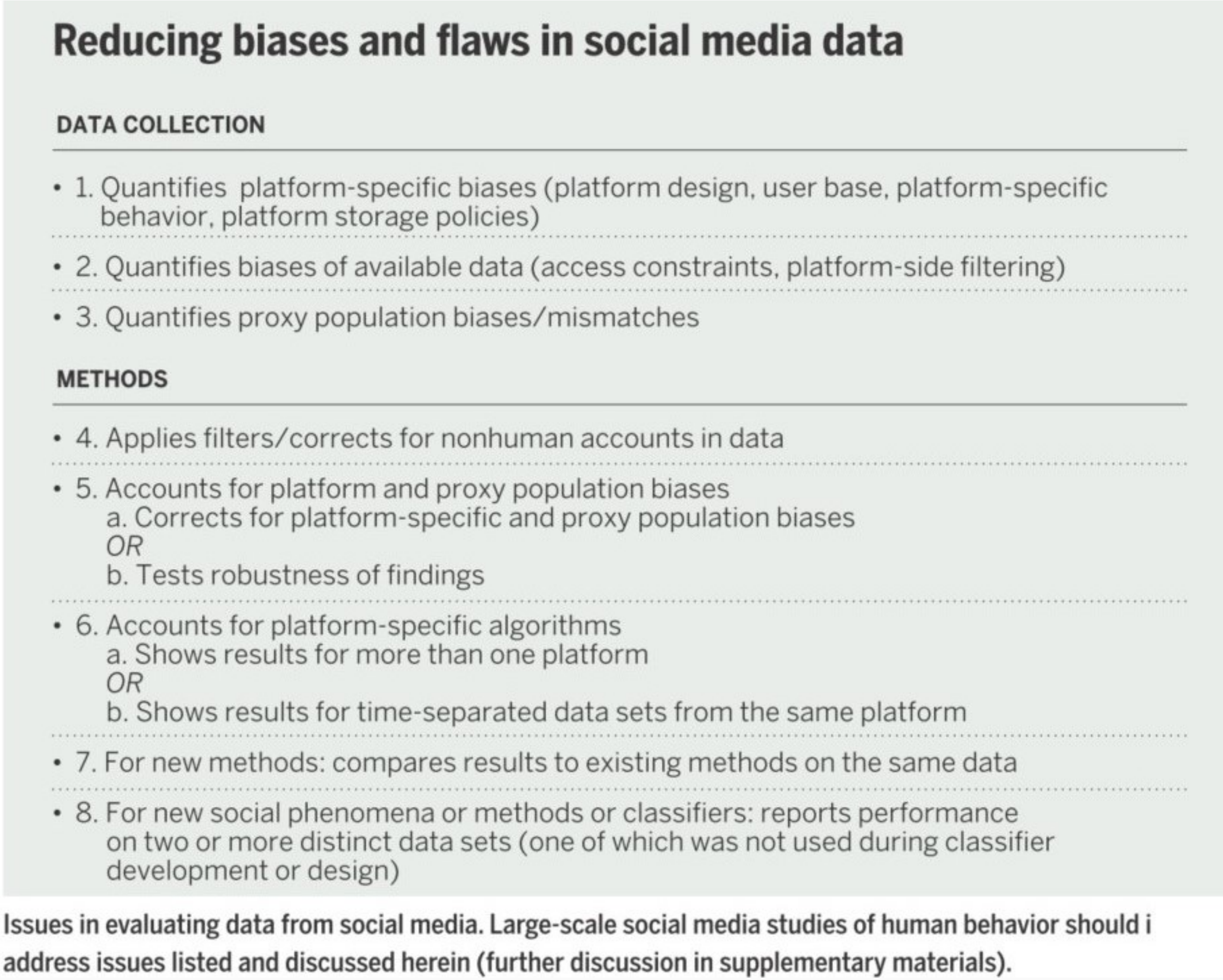
Twitter API does not always return 100% of publicly available tweets (Morstatter et al 2014)

Human behavior and online platform design

e.g. Google Flu (Lazer et al 2014)

⇒ For an overview of sources of bias see Ruths and Pfeffer (2015) and Lazer et al (2017)

Biases in social media data



Reducing biases and flaws in social media data

DATA COLLECTION

- 1. Quantifies platform-specific biases (platform design, user base, platform-specific behavior, platform storage policies)
- 2. Quantifies biases of available data (access constraints, platform-side filtering)
- 3. Quantifies proxy population biases/mismatches

METHODS

- 4. Applies filters/corrects for nonhuman accounts in data
- 5. Accounts for platform and proxy population biases
  a. Corrects for platform-specific and proxy population biases
  OR
  b. Tests robustness of findings
- 6. Accounts for platform-specific algorithms
  a. Shows results for more than one platform
  OR
  b. Shows results for time-separated data sets from the same platform
- 7. For new methods: compares results to existing methods on the same data
- 8. For new social phenomena or methods or classifiers: reports performance on two or more distinct data sets (one of which was not used during classifier development or design)

Issues in evaluating data from social media. Large-scale social media studies of human behavior should i address issues listed and discussed herein (further discussion in supplementary materials).

Ruths and Pfeffer (2015) "Social media for large studies of behavior"

Computer exercises