

Automated Web Scraping with R

Resul Umit

May 2021

[Skip intro — To the contents slide.](#)

[I can teach this workshop at your institution — Email me.](#)

Who am I?

Resul Umit

- post-doctoral researcher in political science at the University of Oslo
- teaching and studying representation, elections, and parliaments
 - **a recent publication**: Parliamentary communication allowances do not increase electoral turnout or incumbents' vote share
- teaching workshops, also on
 - **writing reproducible research papers**
 - **version control and collaboration**
 - **working with Twitter data**
 - **creating academic websites**
- more information available at **resulumit.com**

The Workshop — Overview

- One day, on how to automate the process of extracting data from websites
 - 200+ slides, 75+ exercises
 - a **demonstration website** for practice
- Designed for researchers with basic knowledge of R programming language
 - does not cover programming with R
 - e.g., we will use existing functions and packages
 - ability to work with R will be very helpful
 - but not absolutely necessary — this ability can be developed during and after the workshop as well

The Workshop — Motivation

- Data available on websites provide attractive opportunities for academic research
 - e.g., parliamentary websites were the main source of data for my PhD
- Acquiring such data requires
 - either a lot of resources, such as time
 - or a set of skills, such as automated web scraping
- Typically, such skills are not part of academic training
 - for my PhD, I hand-visited close to 3000 webpages to collect data manually
 - on members of ten parliaments
 - multiple times, to update the dataset as needed

The Workshop — Motivation — Aims

- To provide you with an understanding of what is ethically possible
 - we will cover a large breath of issues, not all of it is for long-term memory
 - hence the slides are designed for self study as well
 - awareness of what is ethical and possible, Google, and perseverance are all you need
- To start you with acquiring and practicing the skills needed
 - practice with the demonstration website
 - plenty of data, stable structure, and an ethical playground
 - start working on a real project

The Workshop — Contents

Part 1. Getting the Tools Ready

- e.g., installing packages

Part 2. Preliminary Considerations

- e.g., ethics of web scraping

Part 3. Webpage Source Code

- e.g., elements, tags, and classes

Part 4. Scraping Static Pages

- e.g., getting text from an element

Part 5. Scraping Dynamic Pages

- e.g., clicking before scraping

To the list of references.

The Workshop — Organisation

- ~~Sit in groups of two~~ Breakout in groups of two for exercises
 - participants learn as much from their partner as from instructors
 - switch partners after every other part
 - leave your breakout room manually, when everyone in the group is ready
- Type, rather than copy and paste, the code that you will find on these slides
 - typing is a part of the learning process
 - slides are, and will remain, available at resulimit.com/teaching/scrp_workshop.html
- When you have a question
 - ask your partner
 - google together
 - ask me

The Workshop — Organisation — Slides

03 : 00

Slides with this background colour indicate that your action is required, for

- setting the workshop up
 - e.g., installing R
- completing the exercises
 - e.g., downloading tweets
 - there are 75+ exercises
 - these slides have countdown timers
 - as a guide, not to be followed strictly

The Workshop — Organisation — Slides

- Codes and texts that go in R console or scripts appear as such – in a different font, on gray background
 - long codes and texts will have their own line(s)

```
# read in the tweets dataset  
df <- read_rds("tweets.rds") %>%  
  
# split the variable text, create a new variable called da_tweets  
unnest_tokens(output = da_tweets, input = text, token = "tweets") %>%  
  
# remove rows that match any of the stop words as stored in the stop_words dataset  
anti_join(stop_words, by = c("da_tweets" = "word"))
```

The Workshop — Organisation — Slides

- Codes and texts that go in R console or scripts appear as such – in a different font, on gray background
 - long codes and texts will have their own line(s)
- Results that come out as output appear as such — in the same font, on green background
 - except very obvious results, such as figures and tables
- Specific sections are highlighted yellow as such for emphasis
 - these could be for anything — codes and texts in input, results in output, and/or texts on slides
- The slides are designed for self-study as much as for the workshop
 - *accessible*, in substance and form, to go through on your own

Part 1. Getting the Tools Ready

[Back to the contents slide.](#)

Workshop Slides — Access on Your Browser

- Having the workshop slides* on your own machine might be helpful
 - flexibility to go back and forward on your own
 - especially while in a breakout room
 - ability to scroll across long codes on some slides
- Access at https://resulunit.com/teaching/scrp_workshop.html
 - will remain accessible after the workshop
 - might crash for some Safari users
 - if using a different browser application is not an option, view the [PDF version of the slides](#) on GitHub

* These slides are produced in R, with the `xaringan` package ([Xie, 2020](#)).

Demonstration Website — Explore on Your Browser

05 : 00

- There is a demonstration website for this workshop
 - available at <https://parliament-luzland.netlify.app/>
 - includes fabricated data on the imaginary Parliament of Luzland
 - provides us with plenty of data, stable structure, and an ethical playground
- Using this demonstration website for practice is recommended
 - tailored to exercises, no ethical concern
 - but not compulsory — use a different one if you prefer so
- Explore the website now
 - see the four sections
 - click on the links to see an individual page for
 - states, constituencies, members, and documents

R — Download from the Internet and Install

- Programming language of this workshop
 - created for data analysis, extending for other purposes
 - e.g., accessing websites
 - allows for all three steps in one environment
 - accessing websites; scraping and processing data
 - an alternative: `python`
- Download R from <https://cloud.r-project.org>
 - optional, if you have it already installed — but then consider updating*
 - the `R.version.string` command checks the version of your copy
 - compare with the latest official release at <https://cran.r-project.org/sources.html>

* The same applies to all software that follows — consider updating if you have them already installed. This ensures everyone works with the latest, exactly the same, tools.

RStudio — Download from the Internet and Install

- Optional, but highly recommended
 - facilitates working with R
- A popular integrated development environment (IDE) for R
 - an alternative: **GNU Emacs**
- Download RStudio from <https://rstudio.com/products/rstudio/download>
 - choose the free version
 - to check for any updates, follow from the RStudio menu:

Help -> Check for Updates

RStudio Project — Create from within RStudio

- RStudio allows for dividing your work with R into separate projects
 - each project gets dedicated workspace, history, and source documents
 - [this page](#) has more information on why projects are recommended
- Create a new RStudio project for for this workshop, following from the RStudio menu:

```
File -> New Project -> New Directory -> New Project
```
- Choose a location for the project with Browse . . .
 - avoid choosing a synced location, e.g., Dropbox
 - likely to cause warning and/or error messages
 - if you must, pause syncing, or add an sync exclusion

R Packages — Install from within RStudio*

02:00

Install the packages that we need

```
install.packages(c("rvest", "RSelenium", "robotstxt", "polite",  
                  "tidyverse", "tidytext"))
```

* You may already have a copy of one or more of these packages. In that case, I recommend updating by re-installing them now.

R Packages — Install from within RStudio

Install the packages that we need

```
install.packages(c("rvest", "RSelenium", "robotstxt", "polite",  
                  "tidyverse", "tidytext"))
```

We will use

- `rvest` ([Wickham, 2021](#)), for scraping websites
- `RSelenium` ([Harrison, 2020](#)), for browsing the web programmatically
- `robotstxt` ([Meissner & Ren, 2020](#)), for checking permissions to scrape websites
- `polite` ([Perepolkin, 2019](#)), for compliance with permissions to scrape websites

R Packages — Install from within RStudio

```
install.packages(c("rvest", "RSelenium", "robotstxt", "polite",  
                  "tidyverse", "tidytext"))
```

- tidyverse (Wickham & RStudio 2019), for various tasks
 - including data manipulation, visualisation
 - alternative: e.g., base R
- tidytext (Robinson & Silge, 2021), for working with text as data

Java — Download from the Internet and Install

- A language and software that RSelenium needs
 - for automation scripts
- Download Java from <https://www.java.com/en/download/>
 - requires restarting any browser that you might have open

Chrome — Download from the Internet and Install

- A browser that facilitates web scraping
 - favoured by R Selenium and most programmers
- Download Chrome from <https://www.google.com/chrome/>

ScrapeMate — Add Extension to Browser

- An **open source software** extension to Chrome, Firefox
 - facilitates selecting what to scrape from a webpage
 - optional, but highly recommended
- Add the extension to your preferred browser
 - for Chrome, search at <https://chrome.google.com/webstore/category/extensions>
 - for Firefox, search at <https://addons.mozilla.org/>
- If you cannot use Chrome or Firefox
 - drag and drop the following link to your bookmarks bar: **SelectorGadget**
 - another — similar but older — **open source software** with the same functionality

Other Resources*

- R Selenium vignettes
 - available at <https://cran.r-project.org/web/packages/R Selenium/vignettes/basics.html>
- R for Data Science (Wickham & Grolemund, 2019)
 - open access at <https://r4ds.had.co.nz>
- Text Mining with R: A Tidy Approach (Silge & Robinson, 2017)
 - open access at tidytextmining.com
 - comes with a [course website](#) where you can practice

* I recommend these to be consulted not during but after the workshop.

Part 2. Preliminary Considerations

[Back to the contents slide.](#)

Considerations — the Law

- Web scraping might be illegal
 - depending on who is scraping what, why, how — and under which jurisdiction
 - reflect, and check, before you scrape
- Web scraping might be more likely to be illegal if, for example,
 - it is harmful to the source
 - commercially
 - e.g., scraping a commercial website to create a rival website
 - physically
 - e.g., scraping a website so hard and fast that it collapses
 - it gathers data that is
 - under copyright
 - not meant for the public to see
 - then used for financial gain

Considerations — the Ethics

- Web scraping might be unethical
 - even when it is legal
 - depending on who is scraping what, why, and how
 - reflect before you scrape
- Web scraping might be more likely to be unethical if, for example,
 - it is — edging towards — illegal
 - it does not respect the restrictions
 - as defined in `robots.txt` files
 - it harvests data
 - that is otherwise available to download, e.g., through APIs
 - without purpose, at dangerous speed, repeatedly

Considerations — the Ethics — robots.txt

- Most websites declare a robots exclusion protocol
 - making their rules known with respect to programmatic access
 - who is (not) allowed to scrape what, and sometimes, at what speed
 - within robots.txt files
 - available at, e.g., www.websiteurl.com/robots.txt
- The rules in robots.txt cannot not enforced
 - but should be respected for ethical reasons
- The language in robots.txt files is specific but intuitive
 - easy to read and understand
 - the robotstxt package makes it even easier

Considerations — the Ethics — robots.txt — Syntax

- It has pre-defined keys, most importantly
 - User-agent indicates who the protocol is for
 - Allow indicates which part(s) of the website can be scraped
 - Disallow indicates which part(s) must not be scraped
 - Crawl-delay indicates how fast the website could be scraped
- In case you write your own protocol one day, note that
 - the keys start with capital letters
 - they are followed by a colon :

```
User-agent:  
Allow:  
Disallow:  
Crawl-delay:
```

Considerations — the Ethics — robots.txt — Syntax

- Websites define their own values
 - after the colon and a white space
- Note that
 - * indicates the protocol is for everyone
 - / indicates all sections and pages
 - /about/ indicates a specific path
 - values for Crawl-delay indicate seconds
 - this website allows anyone to scrape, provided that
 - /about/ is left out, and
 - the website is accessed at 5-seconds intervals

```
User-agent: *  
Allow: /  
Disallow: /about/  
Crawl-delay: 5
```

Considerations — the Ethics — robots.txt — Syntax

Files might include optional comments, written after the number sign #

```
# thank you for respecting our protocol
```

```
User-agent: *
```

```
Allow: /
```

```
Disallow: /about/
```

```
Crawl-delay: 5 # five second delay, to ensure our servers are not overloaded
```

Considerations — the Ethics — robots.txt — Syntax

The protocol of this website only applies to Google

- Google is allowed to scrape everything
- there is no defined rule for anyone else

```
User-agent: googlebot  
Allow: /
```

Considerations — the Ethics — robots.txt — Syntax

The protocol of this website only applies to Google

- Google is **disallowed** to scrape **two** specific paths
 - with no limit on speed
- there is no defined rule for anyone else

```
User-agent: googlebot  
Disallow: /about/  
Disallow: /history/
```


Considerations — the Ethics — robots.txt — Syntax

This website has different protocols for different agents

- Google is allowed to scrape everything, with a 5-second delay
- Bing is not allowed to scrape anything
- everyone else can scrape the section or page located at www.websiteurl/about/

```
User-agent: googlebot
```

```
Allow: /
```

```
Crawl-delay: 5
```

```
User-agent: bing
```

```
Disallow: /
```

```
User-agent: *
```

```
Allow: /about/
```

Considerations — the Ethics — robots.txt

- The robots.txt packages facilitates checking website protocols
 - from within R — no need to visit websites via browser
 - provides functions to check, among others, the rules for specific paths and/or agents
- There are two main functions
 - robots.txt, which gets complete protocols
 - paths_allowed, which checks protocols for one or more specific paths

Considerations — the Ethics — robots.txt

Use the `robots.txt` function to get a protocol

- supply a base url with the `domain` argument
 - as a string
 - probably the only argument that you will need

```
robots.txt(  
    domain = NULL,  
    ...  
)
```

Considerations — the Ethics — robots.txt

```
library(robots.txt)
robots.txt(domain = "https://parliament-luzland.netlify.app")
```

...

User-agent: googlebot
Disallow: /states/

User-agent: *
Allow: /

\$robexclobj

<Robots Exclusion Protocol Object>

\$bots [1] "googlebot" "*"

...

Considerations — the Ethics — robots.txt

Use the `paths_allowed` function to check protocols for one or more specific paths

- supply a base url with the `domain` argument
- `path` and `bot` are the other important arguments
 - notice the default values
- leads to either `TRUE` (allowed to scrape) or `FALSE` (not allowed)

```
paths_allowed(  
  domain = "auto",  
  paths = "/",  
  bot = "*",  
  ...  
)
```

Considerations — the Ethics — robots.txt

```
paths_allowed(domain = "https://parliament-luzland.netlify.app")
```

```
[1] TRUE
```

```
paths_allowed(domain = "https://parliament-luzland.netlify.app",  
              paths = c("/states/", "/constituencies/"))
```

```
[1] TRUE TRUE
```

```
paths_allowed(domain = "https://parliament-luzland.netlify.app",  
              paths = c("/states/", "/constituencies/"), bot = "googlebot")
```

```
[1] FALSE TRUE
```

Exercises

07:30

1) Check the protocols for <https://www.theguardian.com>

- via a browser and with the `robotstxt` function
- compare what you see

2) Check a path with the `paths_allowed` function

- such that it will return `FALSE`

3) Check the protocols for any website that you might wish to scrape

- with the `robotstxt` function

Considerations — the Ethics — Speed

- Websites are designed for visitors with human-speed in mind
 - computer-speed visits can overload servers, depending on their bandwidth
 - popular websites might have more bandwidth
 - but, they might attract multiple scrapers at the same time
- Waiting a little between two visits makes scraping more ethical
 - waiting time may or may not be defined in the protocol
 - lookout for, and respect, the `Crawl-delay` key in `robots.txt`
 - **Part 4** covers how to wait
- Not waiting enough might lead to a ban
 - by site owners, administrators
 - for IP addresses with undesirably high number of visits in a short period of time

Considerations — the Ethics — Purpose

Ideally, we scrape for a purpose

- e.g., for academics, to answer one or more research questions, test hypotheses
 - developed prior to data collection, analysis
 - based on, e.g., theory, claims, observations
 - perhaps, even pre-registered
 - e.g., at [OSF Registries](#)

Considerations — Data Storage

Scraped data frequently requires

- large amounts of digital storage space
 - internet data is typically big data
- private, safe storage spaces
 - due to local rules, institutional requirements

Part 3. Webpage Source Code

[Back to the contents slide.](#)

Webpage Source Code — Overview

- Webpages include more than what is immediately visible to visitors
 - not only text, images, links
 - but also code for structure, style, and functionality — interpreted by browsers first
 - HTML provides the structure
 - CSS provides the style
 - JavaScript provides functionality, if any
- Web scraping requires working with the source code
 - even when harvesting the visible only
 - but source code is rarely a nuisance
 - allows choosing one or more desired parts of the visible
 - e.g., text in table and/or bold only
 - offers more, invisible, data
 - e.g., URLs hidden under text

Webpage Source Code — View in Browser

The `Ctrl + U` shortcut is to display source code — alternatively, right click and `View Page Source`

Webpage Source Code — View in Browser — DOM

Browsers also offer putting source codes in a structure

- known as DOM (document object model), initiated by the F12 key on Chrome

Exercises

05:00

4) View the source code of a page

- as plain code and as in DOM
- compare the look of the two

5) Search for a word or a phrase in source code

- copy from the front-end page
- search in plain text code or in DOM
 - the `Ctrl + F`
- compare the look of the front- and back-end

Webpage Source Code — HTML — Overview

- HTML stands for Hypertext Markup Language
 - it gives the structure to what is visible to visitors
 - text, images, links
 - would a piece of text appear in a paragraph or a list?
 - depends on the HTML code around that text

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      h1 {color: blue;}
    </style>
  </head>
  <body>
    <h1>A header</h1>
    <p>This is a paragraph.</p>
    <ul>
      <li>This</li>
      <li>is a</li>
      <li>list</li>
    </ul>
  </body>
</html>
```


Webpage Source Code — HTML — Syntax

HTML consists of **elements**

```
<p>This is a one sentence paragraph.</p>
```

This is a one sentence paragraph.

Note that

- there is only one element on this page
 - a paragraph

Webpage Source Code — HTML — Syntax

Most elements have opening and closing tags

```
<p>This is a one sentence paragraph.</p>
```

This is a one sentence paragraph.

Note that

- tag name, in this case **p**, defines the structure of the element
- the closing tag has a forward slash **/** before the element name

Webpage Source Code — HTML — Syntax

Most elements have some content

```
<p>This is a one sentence paragraph.</p>
```

This is a one sentence paragraph.

Webpage Source Code — HTML — Syntax

Elements can be nested

```
<p>This is a <strong>one</strong> sentence paragraph.</p>
```

This is a **one** sentence paragraph.

Note that

- there are two elements above, a paragraph and a strong emphasis
- strong is said to be the child of the paragraph element
 - there could be more than one child
 - in that case, children are numbered from the left
- paragraph is said to be the parent of the strong element

Webpage Source Code — HTML — Syntax

Elements can have **attributes**

```
<p>This is a <strong id="sentence-count">one</strong> sentence paragraph.</p>
```

This is a **one** sentence paragraph.

Note that

- the id attribute is not visible to the visitors
- attribute string **sentence-count** could have been anything I could come up with
 - unlike the tag and attribute names — e.g., strong, id as they are pre-defined
- there are some other attributes that are visible

Webpage Source Code — HTML — Syntax

There could be more than one attribute in an element

```
<p>This is a <strong class="count" id="sentence-count">one</strong> sentence paragraph.</p>  
<p>There are now <strong class="count" id="paragraph-count">two</strong> paragraphs.</p>
```

This is a **one** sentence paragraph.

There are now **two** paragraphs.

Note that

- there could be more than one attribute in an element
 - with a white space in between them
- the `class` attribute can apply to multiple elements
 - while the `id` attribute must be unique on a page

Webpage Source Code — HTML — Important Elements & Attributes

Links

```
<p>Click <a href="https://www.google.com/">here</a> to google things.</p>
```

Click [here](https://www.google.com/) to google things.

Note that

- href (hypertext reference) is a required attribute for the the a (anchor) tag
- most attributes are optional, some are required

Webpage Source Code — HTML — Links

Links

```
<p>Click <a title="This text appears when visitors hover over the link"  
      href="https://www.google.com/">here</a> to google things.</p>
```

Click [here](https://www.google.com/) to google things.

Note that

- the a (anchor) tag is used with href (hypertext reference)

Webpage Source Code — HTML — Lists

The `` tag introduces unordered lists, while the `` tag defines lists items

```
<ul>
  <li>books</li>
  <li>journal articles</li>
  <li>reports</li>
</ul>
```

- books
- journal articles
- reports

Note that

- Ordered lists are introduced with the `` tag instead

Webpage Source Code — HTML — Notes

By default, multiple spaces and/or lines breaks are not meaningful

```
<ul><li>books</li><li>journal          articles</li><li>reports</li></ul>
```

- books
- journal articles
- reports

Note that

- plain source code may or may not be written in a readable manner
- this is one reason why DOM is helpful

Webpage Source Code — CSS — Overview

- CSS stands for Cascading Stylesheets
 - it gives the style to what is visible to visitors
 - text, images, links
 - would a piece of text appear in black or blue?
 - depends on the CSS for that text
- CSS can be defined
 - inline, as an attribute of an element
 - internally, as a child element of the head element
 - externally, but then linked in the head element

Webpage Source Code — CSS — Syntax

CSS is written in rules

```
p {font-size:12px;}
```

```
.count {background-color:yellow;}
```

```
#sentence-count {color:red;}
```

Webpage Source Code — CSS — Syntax

CSS is written in rules, with a syntax consisting of

- one or more **selectors**

```
p {font-size:14px;}  
h1 h2 {color:blue;}  
.count {background-color:yellow;}  
#sentence-count {color:red; font-size:14px;}
```

Note that

- selector type defines the syntax
 - elements are plain
 - e.g., p, h1, h2
 - classes start with a full stop
 - ids start with a number sign

Webpage Source Code — CSS — Syntax

CSS is written in rules, with a syntax consisting of

- one or more selectors
- a **declaration**

```
p {font-size:14px;}  
h1 h2 {color:blue;}  
.count {background-color:yellow;}  
#sentence-count {color:red; font-size:14px;}
```

Note that

- declarations are written in between two curly brackets

Webpage Source Code — CSS — Syntax

CSS is written in rules, with a syntax consisting of

- one or more selectors
- a declaration, with one or more **properties**

```
p {font-size:14px;}  
h1 h2 {color:blue;}  
.count {background-color:yellow;}  
#sentence-count {color:red; font-size:14px;}
```

Note that

- properties are followed by a colon

Webpage Source Code — CSS — Syntax

CSS is written in rules, with a syntax consisting of

- one or more selectors
- a declaration, with one or more properties and values

```
p {font-size: 14px;}  
h1 h2 {color: blue;}  
.count {background-color: yellow;}  
#sentence-count {color: red; font-size: 14px;}
```

Note that

- values are followed by a semicolon
- property:value; pairs are separated by a white space

Webpage Source Code — CSS — Internal

- CSS rules can be defined internally
 - within the `style` element
 - as a child of the `head` element
- Internally defined rules apply to all matching selectors
 - on the same page

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      h1 {color: blue;}
    </style>
  </head>
  <body>
    <h1>A header</h1>
    <p>This is a paragraph.</p>
    <ul>
      <li>This</li>
      <li>is a</li>
      <li>list</li>
    </ul>
  </body>
</html>
```

Webpage Source Code — CSS — External

- CSS rules can be defined externally
 - saved somewhere linkable
 - defined with the the linked element
 - as a child of the head element
- Externally defined rules
 - are saved in a file with .css extension
 - apply to all matching selectors
 - on any page linked

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="styles" href="simple.css">
  </head>
  <body>
    <h1>A header</h1>
    <p>This is a paragraph.</p>
    <ul>
      <li>This</li>
      <li>is a</li>
      <li>list</li>
    </ul>
  </body>
</html>
```

Webpage Source Code — CSS — Inline

CSS rules can also be defined inline

- with the `style` attribute
- does not require selector
- applies only to that element

```
<p>This is a <strong style="color:blue;">one</strong> sentence paragraph.</p>
```

This is a **one** sentence paragraph.

Part 3. Data Collection

[Back to the contents slide.](#)

Part 4. Data Preperation

[Back to the contents slide.](#)

Data Preperation — Overview

- The `rtweet` package does a very good job with data preperation to start with
 - returns data frames, with mostly tidy data
 - although Twitter APIs return nested lists
 - some variables are still lists
 - e.g., hastags
- Further data preparation depends on your research project
 - most importantly, on whether you will work with texts or not
 - we will cover some common preparation steps

Data Preperation — Overview — Strings

- Most researchers would be interested in textual Twitter data
 - tweets as a whole, but also specifically hashtags *etc.*
- There are many components of tweets as texts
 - e.g., mentions, hashtags, emojis, links *etc.*
 - but also punctuation, white spaces, upper case letters *etc.*
 - some of these may need to be taken out before analysis
- I use the `stringr` package (Wickham, 2019) for string operations
 - part of the `tidyverse` family
 - you might have another favourite already
 - no need to change as long as it does the job

Data Preperation — Overview — Numbers

- There is more to Twitter data than just tweets
 - e.g., the number of followers, likes *etc.*
 - see Silva and Proksch (2020) for a great example
- I use the `dp`lyr package (Wickham et al, 2020) for most data operations
 - part of the tidyverse family
 - you might have another favourite already
 - no need to change as long as it does the job

Data Preparation — Remove Mentions

```
tweet <- "These from @handle1 are #socoool. 🙌 A #mustsee, @handle2!  
👉 t.co/aq7MJJ1  
👉 https://t.co/aq7MJJ2"
```

```
str_remove_all(string = tweet, pattern = "[@][\\w_-]+")
```

```
[1] "This from are #socoool. 🙌 A #mustsee, ! 👉 t.co/aq7MJJ1 👉 https://t.co/aq7MJJ2"
```

Data Preperation — Remove Hashtags

```
tweet <- "These from @handle1 are #socoool. 🙌 A #mustsee, @handle2!  
👉 t.co/aq7MJJ1  
👉 https://t.co/aq7MJJ2"
```

```
str_remove_all(string = tweet, pattern = "[#][\\w_-]+")
```

```
[1] "These from @handle1 are . 🙌 A , @handle2! 👉 t.co/aq7MJJ1 👉 https://t.co/aq7MJJ2"
```

Data Preperation — Remove Links

```
tweet <- "These from @handle1 are #socoool. 🙌 A #mustsee, @handle2!  
👉 t.co/aq7MJJ1  
👉 https://t.co/aq7MJJ2"
```

```
str_remove_all(string = tweet, pattern = "http\\S+\\s*")
```

```
[1] "These from @handle1 are. 🙌 A, @handle2! 👉 t.co/aq7MJJ1"
```

- Notice that
 - links come in various formats
 - you may need multiple or complicated regular expression patterns

Data Preperation — Remove Links — Alternative

08:00

Use the `urls_t.co` variable to remove all links

- if there are more than one link in a tweet, they are stored as a list in this variable

```
# start with your existing dataset of tweets
df_tweets <- df_tweets %>%

# limit the operation to within individual tweets
  group_by(status_id) %>%

# create a new variable of tweets without links
  mutate(tidy_text =

# by removing them from the existing variable text
    str_remove_all(string = text,

# that matches the urls_t.co variable, after being collapsed into a string
    pattern = str_c(unlist(urls_t.co), collapse = "|"))
```

Data Preperation — Remove Emojis

```
tweet <- "These from @handle1 are #socoool. 🙌 A #mustsee, @handle2!  
👉 t.co/aq7MJJ1  
👉 https://t.co/aq7MJJ2"
```

```
iconv(x = tweet, from = "latin1", to = "ASCII", sub = "")
```

```
[1] "These from @handle1 are #socoool. A #mustsee, @handle2! t.co/aq7MJJ1 https://t.co/aq7MJJ2"
```

Data Preparation — Exercises — Notes

- The exercises in this part are best followed by
 - using `tweets.rds` or similar dataset
 - saving a new variable at every step of preparation
 - observing the newly created variables
 - to confirm whether the code works as intended
- The `mutate` function, from the `dplyr` package, can be helpful, as follows
 - recall that `text` is the variable for tweets

```
tweets <- read_rds("data/tweets.rds")  
  
clean_tweets <- tweets %>%  
  mutate(no_mentions = str_remove_all(string = text, pattern = "[@][\\w_-]+"))
```

Exercises

10:00

41) Remove mentions

- hint: the pattern is "`[@][\\w_]+`"

42) Remove hastags

- hint: the pattern is "`[#][\\w_]+`"

43) Remove links

- by using the links from the `urls_t.co` variable

44) Remove emojis

- pull the help file for the `iconv` function first

Data Preperation — Remove Punctuations

```
tweet <- "These from @handle1 are #socool. 🙌 A #mustsee, @handle2!  
👉 t.co/aq7MJJ1  
👉 https://t.co/aq7MJJ2"
```

```
str_remove_all(string = tweet, pattern = "[[:punct:]]")
```

```
[1] "This from are socool 🙌 A mustsee handle2 👉 tcoa7MJJ1 👉 httpst.coaq7MJJ2"
```

Notice that

- this removed all punctuation, including those in mentions, hashtags, and links
- if tweets are typed with no spaces after punctuation, this might lead to merged pieces of text
 - alternatively, try the `str_replace` function to replace punctuation with space

Data Preperation — Remove Punctuations — Alternative

```
tweet <- "This is a sentence.There is no space before this sentence."
```

```
str_remove_all(string = tweet, pattern = "[[:punct:]]")
```

```
[1] "This is a sentenceThere is no space before this sentence"
```

```
str_replace_all(string = tweet, pattern = "[[:punct:]]", replacement = " ")
```

```
[1] "This is a sentence There is no space before this sentence "
```

Data Preperation — Remove Punctuations — Alternative

```
tweet <- "This is a sentence.There is no space before this sentence."
```

```
str_replace_all(string = tweet, pattern = "[[:punct:]]", replacement = " ")
```

```
[1] "This is a sentence There is no space before this sentence "
```

Data Preperation — Remove Repeated Whitespace

```
tweet <- "There are too many spaces after this sentence.   This is a new sentence."
```

```
str_squish(string = tweet)
```

```
[1] "There are too many spaces after this sentence. This is a new sentence."
```

Note that

- white spaces can be introduced not only by users on Twitter, but also by us, while cleaning the data
 - e.g., removing and/or replacing operations above
 - hence, this function might be useful after other operations

Data Preperation — Change Case

```
tweet <- "lower case. Sentence case. Title Case. UPPER CASE."
```

```
str_to_lower(string = tweet)
```

```
[1] "lower case. sentence case. title case. upper case."
```

Note that

- there are other functions in this family, including
 - `str_to_sentence`, `str_to_title`, `str_to_upper`

Exercises

10:00

45) Remove punctuations

- by using the `str_replace_all` function
- hint: the pattern is `[[:punct:]]`

46) Remove whitespace

- hint: the function is called `str_squish`

47) Change case to lower case

- hint: the function is called `str_to_lower`

Data Preperation — Change Unit of Observation

Research designs might require changing the unit of observation

- aggregation
 - e.g., at the level of users, locations, hashtags etc.
 - summarise with `dplyr`
- dis-aggregation
 - e.g., to the level of words
 - tokenise with `tidytext`

Data Preperation — Change Unit of Observation — Aggregation

Aggregate at the level of users

- the number of tweets per user

```
# load the tweets dataset  
df <- read_rds("tweets.rds") %>%  
  
# group by users for aggregation  
group_by(user_id) %>%  
  
# create summary statistics for variables of interest  
summarise(sum_tweets = n())
```

Data Preperation — Change Unit of Observation — Aggregation

What is aggregated at which level depends on your research design, such as

- aggregate the tweets into a single text
- at the level of users by source

```
# load the tweets dataset
df <- read_rds("tweets.rds") %>%

# group by users for aggregation
group_by(user_id, source) %>%

# create summary statistics for variables of interest
summarise(merged_tweets = paste0(text, collapse = ". "))
```


Data Preperation — Change Unit of Observation — Dis-aggregation

Disaggregate the tweets, by splitting them into smaller units

- also called **tokenisation**

Note that

- by default `sep = "[^[:alnum:]]+"`, which works well with separating tweets into words
 - change this argument with a regular expression of your choice
- this creates a tidy dataset, where each observation is a word
 - all other tweet-level variables are repeated for each observation

```
# load the tweets dataset
df <- read_rds("tweets.rds") %>%

# split the variable text
separate_rows(text)
```

Data Preperation — Change Unit of Observation — Dis-aggregation

The tidytext has a function that works better with tokenising tweets

- with token = "tweets", it dis-aggregates text into words
 - except that it respects usernames, hashtags, and URLs

```
# load the tweets dataset  
df <- read_rds("tweets.rds") %>%  
  
# split the variable text, create a new variable called da_tweets  
unnest_tokens(output = da_tweets, input = text, token = "tweets")
```

Data Preparation — Change Unit of Observation — Dis-aggregation

Tokenise variables to levels other than words

- e.g., characters, words (the default), sentences, lines

```
# load the tweets dataset
df <- read_rds("tweets.rds") %>%

# split the variable text into sentences, create a new variable called da_tweets
unnest_tokens(output = da_tweets, input = text, token = "sentences")
```

Data Preperation — Change Unit of Observation — Dis-aggregation

Tokenise variables other than tweets

- recall that `rtweet` stores multiple hastags, mentions *etc.* as lists

```
# load the tweets dataset
df <- read_rds("tweets.rds") %>%

# unlist the lists of hashtags to create strings
  group_by(status_id) %>%
  mutate(tidy_hashtags = str_c(unlist(hashtags), collapse = " ")) %>%

# split the string, create a new variable called da_tweets
  unnest_tokens(output = da_hashtags, input = tidy_hashtags, token = "words")
```

Data Preperation — Remove Stop Words

Remove the common, uninformative words

- e.g., the, a, i

Note that

- this operation requires a tokenised-to-word variable
- stop words for English are stored in the `stop_words` dataset in the `tidytext` variable
- list of words for other languages are available elsewhere, including
 - the `stopwordslangs` function from the `rtweet` package
 - the `stopwords` function from the `tm` package
 - e.g., use `tm::stopwords("german")` for German

```
# load the tweets dataset
df <- read_rds("tweets.rds") %>%

# split the variable text, create a new variable called da_tweets
unnest_tokens(output = da_tweets, input = text, token = "tweets") %>%

# remove rows that match any of the stop words as stored in the stop_words dataset
anti_join(stop_words, by = c("da_tweets" = "word"))
```

Exercises

10:00

48) Aggregate text to a higher level

- e.g., if you are not using `tweets.rds`, to MP level
 - if not, perhaps to source level

49) Dis-aggregate text to a lower level

- e.g., to words

50) Dis-aggregate hashtags

- i.e., make sure each row has at most one hashtag

51) Remove stop words

References

[Back to the contents slide.](#)

References

- Harrison, J. (2020). **RSelenium: R Bindings for 'Selenium WebDriver'**. R package, version 1.7.7.
- Meissner, P., & Ren, K. (2020). **robotstxt: A 'robots.txt' Parser and 'Webbot'/'Spider'/'Crawler' Permissions Checker**. R package, version 0.7.13.
- Perepolkin, D. (2019). **polite: Be Nice on the Web**. R package, version 0.1.1.
- Robinson, D., & Silge, J. (2021). **tidytext: Text mining using 'dplyr', 'ggplot2', and other tidy tools**. R package, version 0.3.0.
- Silge, J., & Robinson, D. (2017). **Text mining with R: A tidy approach**. O'Reilly. Open access at
- Wickham, H. (2019). **stringr: Simple, Consistent Wrappers for Common String Operations**. R package, version 1.4.0.
- Wickham, H. (2021). **rvest: Easily Harvest (Scrape) Web Pages**. R package, version 1.0.0.
- Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H. and Dunnington, D. (2020). **dplyr: A grammar of data manipulation**. R package, version 0.8.5.
- Wickham, H. and Grolemund, G. (2019). **R for data science**. O'Reilly. Open access at <https://r4ds.had.co.nz>.
- Wickham, H., RStudio (2019). <https://cran.r-project.org/web/packages/tidyverse/index.html>. R package, version 3.3.3.

The workshop ends here.

Congradulations for making it this far, and
thank you for joining me!

[Back to the contents slide.](#)