Working with Twitter Data in R

Resul Umit

February 2021

Skip intro — To the contents slide.

Who am I?

Resul Umit

- post-doctoral researcher at the University of Oslo
- interested in representation, elections, and parliaments
  - a recent publication: Parliamentary communication allowances do not increase electoral turnout or incumbents’ vote share
- working with Twitter data
  - a publication based on Twitter data: The voices of Eurosceptic members of parliament (MPs) echo disproportionately louder on Twitter
    - 400,000+ tweets from 1,000+ MPs
    - automated classification, using a bag-of-words approach
    - manual validation
  - an app based on Twitter data: LikeWise — a *Shiny* app that facilitates searching the tweets a user liked
- more information available at resulumit.com

The Workshop — Overview

- Two days, on how to collect, process, and analyse data from Twitter
    - 150+ slides, 75+ exercises
    - support for applications to Twitter, developing real projects

- Designed for researchers with basic knowledge of R programming language
    - does not cover programming with R
        - e.g., writing functions
    - existing ability to work with data in R will be very helpful
        - but not absolutely necessary — this ability can be developed during and after the workshop as well

The Workshop — Motivation

- Twitter provides attractive opportunities for academic research
    - a simple search for "twitter data" returns 55,000+ results on Google Scholar
        - by the end of 2020

- Research based on Twitter data requires a set of skills
    - typically, these skills are not part of academic training

The Workshop — Motivation — Opportunities

- Popularity of the network
    - about 350 million users, more than half are daily active
    - your subjects are likely Twitter users
        - e.g., for me, members of parliament

The Workshop — Motivation — Opportunities

- Popularity of the network
  - about 350 million users, more than half are daily active*
  - your subjects are likely Twitter users
    - e.g., for me, members of parliament

- Richness of the data
  - about 500 million tweets per day*
  - with up to 90 variables for each tweet

* These statistics are compiled at the end of 2020, by BusinessOfApps.

The Workshop — Motivation — Opportunities

- Popularity of the network
    - about 350 million users, more than half are daily active
    - your subjects are likely Twitter users
        - e.g., for me, members of parliament

- Richness of the data
    - about 500 million tweets per day
    - with up to 90 variables for each tweet

- Accessibility of the data
    - most Twitter data are public
        - i.e., relatively few private profiles
    - APIs (application programming interfaces) enable programmatic access to Twitter
        - e.g., downloading tweets with R, as opposed to viewing tweets as visualised via browsers

The Workshop — Motivation — Challenges

- Research based on Twitter data requires certain skills
    - e.g., availability of API is meaningless for researchers who cannot code yet
    - processing and analysing data are no less challenging without programming skills

- The required skills are often uncovered in the academic training of social scientists
    - e.g., methodology courses
    - but help is available elsewhere, including
        - collaboration with researchers with the skills
            - at the cost of sacrificing some control over your research and research agenda
        - acquiring the skills through individual effort
            - e.g., workshops such as this one

The Workshop — Motivation — Aims

- To provide you with an understanding of what is 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 possible, `Google`, and perseverance are all you need
- To start you with acquiring and practicing the skills needed
    - practice with ready-written code
    - start working on a real project

The Workshop — Contents

Part 1. Preliminary Considerations

- e.g., considering Twitter for research

Part 2. Getting the Tools Ready

- e.g., downloading course material

Part 3. Data Collection

- e.g., acquiring a user's tweets

Part 4. Data Preperation

- e.g., creating a tidy dataset of tweets

Part 5. Data Anaysis: User-Based

- e.g., conducting network analysis

Part 6. Data Anaysis: Tweet-Based

- e.g., conducting sentiment analysis

Part 7. Data Anaysis: Machiene Learning

- *To be added*

Part 8. Next Steps

- e.g., applying for a developer account

To the list of references.

The Workshop — Organisation

- Sit in groups of two
    - participants learn as much from their partner as from instructors
    - switch partners after every part

- Type, rather than copy and paste, the code that you will find on these slides
    - typing is a part of the learning process

- 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. Preliminary Considerations

Back to the contents slide.

Considerations — Research Questions & Hypotheses

- Ideally, we have one or more research questions, hypotheses
  - developed prior to data collection, analysis
    - based on, e.g., theory, claims, observations
  - perhaps, even pre-registered
    - e.g., at OSF Registries
- Not all questions can be answered with Twitter data
  - see relevant literature for what works, what does not
    - e.g., for political science, the review by Jungherr (2016)
    - for public health, the review by Sinnenberg et al. (2017)

Considerations — Potential Biases

There are at least two potential sources of bias in Twitter data

- sampling
    - Twitter users are not representative of the people out there
        - see, for example, Mellon and Prosser (2017)
    - Tweeting behaviour has a strategic component
        - see, for example, Umit (2017)
- mediation
    - the behaviour on Twitter is mediated through written and unwritten rules
        - e.g., there is a button to like, but no dislike
            - might systematically bias the replies towards negative
        - e.g., the common use of the like function as a bookmark
            - what would a study of Twitter likes be measuring?

Considerations — Constraints over Data Access

- Twitter has restrictions on data access
    - how much data is available to download
    - how quickly, how frequently, how far dating back *etc*.

- These restrictions vary across API types
    - e.g., Standard v1.1 is the most restrictive APIs
        - others first generation APIs are the Premium v1.1 and Enterprise: Gnip 2.0 APIs — both with paid subscriptions
        - there are also the second generation APIs, including the newly announced Academic Research product track

- These restrictions also vary within APIs types, across different operations
    - e.g., collecting tweets in real time *vs*. collecting historical tweets
        - but also, collecting historical tweets from a specific user *vs*. tweets from any user

Considerations — Constraints over Data Redistribution

- Twitter restricts content redistribution
    - e.g., only the tweet and/or user IDs can be made publicly available in datasets over 50,000 observations
        - e.g., not the tweets themselves
        - and no more than 1.5M IDs
            - with some exceptions for academic research
    - see Twitter Developer terms for further details

- Reproducibility of research based on Twitter data is limited in practice
    - i.e., reproducibility after publication, by others
    - technically, they can retrieve the same tweets with IDs
        - very demanding on reproducers
        - may even be impossible
            - e.g., some tweets, or whole accounts, might be deleted in time

Considerations — Changes in the Twitter APIs

- Twitter is currently switching to a new generation of APIs
    - replacing APIs v1 with v2
        - each with various types of APIs
    - the switch is not complete, outcome is not clear
        - see the early access options

- Twitter might change the rules of the APIs game at any anytime, again
    - making the existing restrictions more or less strict
        - e.g., while you are in the middle of data collection
    - breaking your plans, code

Considerations — Changes in the Twitter APIs — Notes

- Existing codes to collect tweets may or may not be affected, depending on
  - how the APIs v2 will look in the end
    - it is still a work in progress
  - whether and how the `rtweet` package* will adopt
    - personal communication: it will support v2

* This is the R package that we will use to collect tweets. More details are in Part 2.

Considerations — Changes in the Twitter APIs — Notes

- Existing codes to collect tweets may or may not be affected, depending on
    - how the APIs V2 will look in the end
        - it is still a work in progress
    - whether and how the `rtweet` package will adopt
        - personal communication: it will support V2
- Not all changes are bad
    - among others, APIs v2 introduces the Academic Research product track
        - 'to serve the unique needs and challenges of academic researchers'
            - ranging from master's students to professors
        - access to all public tweets
            - by up to 1.5M a month at a time

Considerations — Law and Ethics

- It is often impossible to get users' consent
    - i.e., for collecting and analysing their data on Twitter
    - Twitter itself has no problem with it, but others might disagree
        - e.g., your law makers, (funding and/or research) institution, subjects, conscience
- Check the rules that apply to your case
    - rules and regulations in your country, at your institution
- Reflect on whether using Twitter data for research is ethical
    - even where it is legal and allowed, it may not be moral

Considerations — Data Storage

Twitter data frequently requires

- large amounts of digital storage space
    - Twitter data is typically big data
        - many tweets, up to 90 variables
    - e.g., a dataset of 1M tweets requires about 300MB
        - when stored in R data formats
- private, safe storage spaces
    - due to Twitter Developer terms
    - but also local rules, institutional requirements

Considerations — Language and Context

- Some tools of text analysis are developed for a specific language and/or context
    - e.g., dictionaries for sentiment analysis
        - might be in English, political texts, only
    - these may not be useful, valid for different languages, and contexts

- Some tools of text analysis are developed for general use
    - e.g., a dictionary for sentiments in everyday language
    - these may not be useful, valid for a specific context
        - e.g., political texts

Part 2. 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
  - ability to scroll across long codes on some slides

- Access at https://resulumit.com/teaching/twtr_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).

R — Download from the Internet and Install

- Programming language of this workshop
    - created for data analysis, extending for other purposes
        - e.g., accessing APIs
    - allows for all three steps in one environment
        - collecting, processing, and analysing Twitter data
    - an alternative: python

- Optional, if you have R already installed
    - consider updating your copy, if it is not up to date
        - type the `R.version.string` command in R to check the version of your copy
        - compare with the latest official release at https://cran.r-project.org/sources.html

- Download R from https://cloud.r-project.org
    - choose the version for your operating system

RStudio — Download from the Internet and Install

- Optional, but highly recommended
  - facilitates working with Twitter data in 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
  - consider updating your copy, if it is not up to date, following 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, following from the RStudio menu

> `File -> New Project -> New Directory -> New Project`

- A non-synced directory is recommended
    - e.g., perhaps Documents, but not Dropbox

R Packages — Install from within RStudio[*]

```r
install.packages(c("rtweet", "httpuv", "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.packages(c("rtweet", "httpuv", "tidyverse", "tidytext"))
```

- `rtweet` (Kearney, 2020), for collecting tweets
  - alternatives: used to be `twitteR` (Gentry, 2015), but not anymore; running Python code in R

- `httpuv` (Cheng & Chang, 2020), for API authorization
  - alternative: using your own access tokens
    - necessitates making an application through a developer
    - has advantages that we will discuss later on

R Packages — Install from within RStudio

```
install.packages(c("rtweet", "httpuv", "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
    - alternative: e.g., `quanteda` (Benoit et al., 2020)

Course Materials — Download from the Internet

- Download the materials from https://www.dropbox.com/s/fej5husakvnf8tz/data.zip?dl=0
    - to a sub-directory of your RStudio project
    - in the code that follows, my subdirectory will be called `data`
- `data/mps.csv`
    - a dataset on the members of parliament (MPs) in the British House of Commons, as of January 2021
    - it includes variables on electoral results as well as Twitter usernames
- `data/status_ids.rds`
    - a dataset with a single variable: `status_id`
    - lists the status IDs of all tweets posted by the MPs listed in `mps.csv`, during January 2021
- `data/tweets.rds`
    - similar to `data/status_ids`, except that
        - the time period is now limited to 15 to 31 January, reducing the number of observations below 50,000, allowing for all variables to be posted online

Twitter — Authorisation

- Authorization to use Twitter APIs requires at least three steps
    - open a user account on Twitter
        - a personal or an institutional (perhaps, for a research project) one
    - with that user account, apply for a developer account
        - so that you are recognised as a developer, have access to the developer portal
        - done once per account
    - with that developer account, register a Twitter app
        - so that you have the keys and tokens for authorisation
        - repeated for every project
- There may be additional steps
    - e.g., registering for the Academic Research product track
    - per project, depending on meeting certain criteria

Twitter — Authorisation — Notes

- It is possible to interact with Twitter APIs without your own Twitter app
  - `rtweet` has a its own Twitter app — `rstats2twitter` — that anyone can use
    - anyone with a Twitter account, who authorises `rstats2twitter` via a pop-up browser

- I suggest that
  - we follow only step one (open an account) now, which
    - you might already have done
    - is otherwise automatic
    - allows us to use `rstats2twitter` and follow the workshop
  - we leave the remaining steps until after the first day
    - to allow you to think and write your applications carefully
    - to get my feedback if you prefer so

Twitter — Open an Account

Sign up for Twitter at https://twitter.com/

- a pre-condition for interacting with Twitter APIs
    - e.g., you must be authorized to collect tweets with R
    - and you must have an account to apply for authorisation in the first place

- helpful for getting to know what you study
    - e.g., the written and unwritten rules that mediate the behaviour on Twitter
        - as discussed in Part 1

- with a strategic username
    - usernames are changeable, but nevertheless public
        - either choose an anonymous username (e.g., `asdf029348`)
        - or choose one carefully — they become a part of users' online presence

Other Resources*

- 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

- quanteda tutorials
  - 'a step-by-step introduction to quantitative text analysis'
  - with the `quanteda` package

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

Part 3. Data Collection

Back to the contents slide.

Data Collection — Overview — APIs

- We will collect data through APIs
    - i.e., Twitter's Standard v1.1 APIs
    - provides more variables than available through browsers
    - comes with rules and restrictions
        - enforced through authentication

- Collecting data through web scraping is also possible
    - e.g., with `GetOldTweets3` — a python library
        - scrapes, scrolls down, and scrapes again to collect all matching data
    - does not require, is not limited by, Twitter APIs
    - limited with what is available on browsers
    - may or may not be ethical and/or legal
        - while there are APIs for this task

Data Collection — Overview — APIs — Types

- In general, there are two main types of APIs
    - REST and Streaming
    - applies to APIS elsewhere, not just at Twitter
    - functions, arguments, behaviour differ slightly
- REST APIs are for single, one-off requests
    - e.g., search for tweets posted in the last 6 to 9 days
    - but also, post or delete tweets
- Streaming APIs are for continuous requests
    - e.g., collect tweets as they are being posted

Data Collection — Overview — APIs — Types

- At Twitter, there is a further differentiation among the APIs
    - e.i., API v2, Enterprise: Gnip 2.0, Premium v1.1, Standard v1.1
    - with each, you can make single or continuous requests
- Rules and restrictions differ from one type to another
    - as does the cost
    - some remove the restrictions on how much data we can access
    - restrictions on how quickly we can access data exist in all types
        - these restrictions are called rate limits
- Rules and restrictions can also differ within one type
    - for different operations
        - e.g., for collecting historical *vs*. live data

Data Collection — Overview — Standard v1.1 API

- We will collect data through Twitter's Standard v1.1 API
    - free of charge
    - thanks to `rweet`'s `rstats2twitter` app, can be used immediately
    - comes with the strictest of restrictions
        - e.g., searches tweets posted in the last 6 to 9 days

- You can surpass these restrictions later on
    - academic researchers can apply for the newly announced the Academic Research product track
    - others can purchase an alternative
    - the principles of data collection are likely to remain the same
        - `rweet` has the `search_30day` and `search_fullarchive` functions for the Premium V1.1 APIs
        - the package may require an update, in line with the Academic Research product track

Data Collection — Overview — APIs — Limitations

Our attempts to collect data will be limited for various reasons, including

- the intended restrictions by Twitter
    - by the limitations of Standard v1.1
    - e.g., by rate limits
        - maximum number of requests
        - per app, type of request, time interval
- any lack of tweets or users matching our search criteria
    - stricter the criteria, more likely to occur
- connections timing out
    - depends on type of requests, and your internet connection
    - more likely for continuous searches
- reasons unknown to humankind
    - sometimes things just happen

Data Collection — `rtweet` — Overview

- A powerful R package for collecting Twitter data

  - created by Michael W. Kearney (University of Missouri)
  - used widely, replacing previous packages for this task
    - e.g., `twitteR`
  - last updated a year ago
    - personal communication: the package will continue to be maintained in the future

- A lot has already been written on this package. See, for example,

  - the package repository on GitHub
  - the package documentation
  - this journal article by its creator
  - this book by Bob Rudis — a user of the package
  - numerous tutorials, such as this, this, and this

- Comes with its own app, `rstats2twitter`
  - allows for collecting tweets without a developer account
  - offers the option of using your own keys and tokens, if/once you have them

Data Collection — `rtweet` — Basics

There are four main groups of functions to collect historical data, starting with

- **`search_`**
  - such as `search_tweets` or `search_users`
  - 4 functions, for general use
    - 2 for standard APIs, 2 for premium APIs

- **`lookup_`**
  - such as `lookup_tweets` or `lookup_users`
  - 5 functions, for expanding an existing variable into a dataset
    - e.g., a list of IDs for tweets or users

- **`get_`**
  - such as `get_followers` or `get_friends`
  - 11 functions, for specific tasks

- **`lists_`**
  - such as `lists_members` or `lists_statuses`
  - 6 functions, for tasks related to Twitter lists specifically

Data Collection — `rtweet` — Basics

There is also one function to collect tweets in real time

- **`stream_tweets`**
  - queries the streaming API
  - returns a small random sample of all tweets as they are posted
  - can be filtered by keywords, users, and/or locations
- For other functions, see the package documentation, including
  - those starting with `post_`, to use your account from within R
    - e.g., posting tweets, direct messages

Data Collection — `rtweet` — Authentication

- There are two different methods of authentication

  - through `rtweet`'s `rstats2twitter` app

    - the app makes requests on your behalf
    - you simply approve this, via a browser that pops up
    - nice and easy, but comes with some limitations

  - through your own app

    - you make requests on your behalf
    - through your own app that you register as a developer
    - takes a little effort, but comes with additional stability and functionality
      - removing the dependency on `rstats2twitter`, over which you have no control
      - getting rid of the pop up windows
      - not only for collecting tweets, but also posting your own tweets
        - and, reading and writing your own direct messages

Data Collection — Start Your Script

- Check that you are in the right project
  - created in Part 2
  - indicated at the upper-right corner of RStudio window
- Create a new R Script, following from the RStudio menu

  > `File -> New File -> R Script`

- Name and save your file
  - to avoid the `Untitled123` problem
  - e.g., `data_collection.R`
- Load the `rtweet` and other packages
  - no need to load the `httpuv` package, enough if installed

```r
library(rtweet)
library(tidyverse)
library(tidytext)
```

Data Collection — Start Your Script

If you are using your own app to authenticate, create a token

- using the `create_token` function
- the `app` argument requires for the name of your own app, as registered on developer.twitter.com
- the other arguments to be filled with the information from the `Keys and tokens` tab on the same website

```
tw_token <- create_token(
    app = "",
    consumer_key = "",
    consumer_secret = "",
    access_token = "",
    access_secret = ""
  )
```

Data Collection — Start Your Script — Alternatives

- You may wish to put your keys and tokens elsewhere
    - they are personal, just like a password
- There are at least two alternatives
    - create a separate script, which you can then `source` at the top of your main script

`keys_tokens.R`

```
tw_token <- create_token(
   app = "",
   consumer_key = "",
   consumer_secret = "",
   access_token = "",
   access_secret = ""
  )
```

`data_collection.R`

```
library(rtweet)
source("keys_tokens.R")
```

Data Collection — Start Your Script — Alternatives

- You may wish to put your keys and tokens elsewhere
  - they are personal, just like a password
- There are at least two alternatives
  - create a separate script, which you can then `source` at the top of your main script
  - store your keys and tokens in your `.Renviron` file, which can be created at the project level as well

`.Renviron`

```
TWITTER_APP=name_of_my_app
TWITTER_CONSUMER_KEY=akN...
TWITTER_CONSUMER_SECRET=HJK...
TWITTER_ACCESS_TOKEN=345...
TWITTER_ACCESS_SECRET=SDF...
```

`data_collection.R`

```
 library(rtweet)
tw_token <- create_token(
    app = Sys.getenv("TWITTER_APP"),
    consumer_key = Sys.getenv("TWITTER_CONSUM
    consumer_secret = Sys.getenv("TWITTER_CON
    access_token = Sys.getenv("TWITTER_ACCESS
    access_secret = Sys.getenv("TWITTER_ACCES
 )
```

search_

Data Collection — `search_tweets`

Collect tweets posted in the last 6 to 9 days

- filter by search query, with the `q` argument
- limited to 18,000 tweets, per 15 minutes, per token[*]
    - set the limit, higher or lower, with the `n` argument[**]
        - works best the multiples of 100
    - if set higher, wait automatically by setting the `retryonratelimit` argument to `TRUE`

```
search_tweets(q,
              n = 100,
              type = "recent",
              include_rts = TRUE,
              geocode = NULL,
              max_id = NULL,
              parse = TRUE,
              token = NULL,
              retryonratelimit = FALSE,
              verbose = TRUE,
              ...
)
```

[*] All limits are for the standard v1.1 APIs.

[**] This argument is common to many functions in the package. I recommend setting it to a small number, such as 200, for the exercises in this workshop. This will save computation time and avoid running into rate limits.

Data Collection — `search_tweets`

- Collect the latest 100 tweets that
  - include the hashtag "publish"

```
search_tweets(q = "#publish")
```

- Note that
  - by default, `type = "recent"`, returning the latest tweets
    - other options are "popular" and "mixed"
  - by default, `n = 100`, returning 100 tweets
  - here we are relying on `rtweet`'s `rstats2twitter` app, which is the default
    - as `token = NULL`

Exercises

10:00

1) Collect the top 300 tweets that

- include the hashtag "AcademicTwitter"
- and assign the resulting data frame to `df_tweets`
  - so that you can observe the results better
  - hence, I recommend doing the same for all APIs searches that follow

2) Observe how the `rstats2twitter` app works

- when you call the function, pay attention to what happens on your R console and on default browser
  - this will happen only once per R session

3) Take some time to explore the data frame

- see which variables are in there, and how they are called
- think about how you could use these variables for research
- hint: use functions like `View`, `str`, `names`, `tibble::glimpse`

4) Conduct the same search on a browser

- using the advanced search form
- compare and contrast the API and browser-based searches

Data Collection — Notes

- Twitter usernames, or handles, are stored under variable `screen_name`
  - can be misleading, as users also have display names
- Twitter allows user to change their usernames as well as display names
  - user IDs, however, do not change
    - `user_id` is a better variable for reproducible research
- The date and time data are matched to Greenwhich Mean Time
  - stored under the variable `created_at`
  - no matter where users actually are at the time of posting
- You may wish to exclude retweets
  - depending on the research question and design
  - by setting `include_rts = FALSE`

Data Collection — `search_tweets`

- Collect the latest 100 tweets that
    - include the word "publish"
    - using your own keys and tokens
- Note that
    - this assumes you have pre-defined `tw_token`
        - with the `create_token` function
        - using the keys and token that you own

```
search_tweets(q = "publish",
              token = tw_token)
```

Data Collection — search_tweets

Collect the top 200 tweets that

- include the word "publish"

```
search_tweets(q = "publish",
                 n = 200,
                 type = "popular")
```

Data Collection — search_tweets

- Collect the top 200 tweets that
  - include the word "publish" and "perish", not necessarily in that order

```
search_tweets(q = "publish perish",
              n = 200,
              type = "popular")
```

- Note that
  - space is treated as the boolean AND operator

Data Collection — `search_tweets`

- Collect the top 200 tweets that
  - include the word "publish" or "perish"

```
search_tweets(q = "publish OR perish",
              n = 200,
              type = "popular")
```

- Note that
  - the boolean OR operator must be specified, in capital letters

Data Collection — search_tweets

- Collect the top 200 tweets that
  - include the exact phrase "publish perish"

```
search_tweets(q = "\"publish perish\"",
              n = 200,
              type = "popular")
```

- Note that
  - double quotation marks " need to be escaped with a backslash \

Data Collection — `search_tweets`

- Collect the top 200 tweets that
  - include "publish" but not "perish"

```
search_tweets(q = "publish -perish",
              n = 200,
              type = "popular")
```

- Note that
  - words can be negated with a hyphen -

Data Collection — `search_tweets`

- Collect the top 200 tweets that
  - include "publish", and
  - are otherwise written in German

```
search_tweets(q = "publish lang:de",
              n = 200,
              type = "popular")
```

- Note that
  - query parameters, such as `lang`, are followed by a colon `:`
    - other parameters include `filter`, `from`, `to`, `since`, `until`, `min_retweets` *etc*.

  - there is an official guide for a comprehensive but not complete list of search operators
    - alternatively, fill in this advanced search form, and observe the resulting syntax

Data Collection — `search_tweets`

Collect the top 200 tweets that

- include "publish", and
- are not in German

Note that

- operators can be negated with a hyphen as well

```
search_tweets(q = "publish -lang:de",
              n = 200,
              type = "popular")
```

Data Collection — `search_tweets` — Notes

- Some query parameters can be passed into the function as arguments as well
  - e.g., `lang`, `filter`

```
search_tweets(q = "publish lang:en filter:rep
              n = 200,
              type = "mixed")
```

- Note that
  - these functions on the right return the same observations
  - there are slight differences in syntax, such as
  - `lang:en` as a parameter
  - `lang = "en"` as an argument

```
search_tweets(q = "publish",
              n = 200,
              type = "mixed",
              lang = "en",
              filter = "replies")
```

Data Collection — `search_tweets` — Notes

- This function returns a data frame
    - as do many other functions in the package
    - because, by default, `parse = TRUE`

- Under the hood, Twitter APIs return nested lists
    - can be tidied into rectangular shape, but in itself not tidy data

Exercises

20:00

5) Collect the latest 200 tweets that include

- the phrase "publish or perish"
- and the word "academia" but not the word "phd"

6) Collect the most recent 200 tweets that

- include the word "Luzern"
- sent within 25 mile radius around Lucerne city centre
- excluding retweets
- hint: pull the help file for the function to see which arguments you can use
    - get help from Google about the coordinates

7) Collect the most recent 35,000 tweets that

- include the word "phd"
- note that this is over the limit of 18,000
- hint: pull the help file for the function to see which argument you must use to surpass the limit

Data Collection — `search_users`

- Collect information on users
  - filter usernames, names, bios
  - by search query with the `q` argument
  - returns information from recently active users
  - limited to 1,000 users

```
search_users(q,
             n = 100,
             parse = TRUE,
             token = NULL,
             verbose = TRUE)
```

- Note that
  - there is no `retryonratelimit` argument
  - you can still use the complete list of search operators and this advanced search form for help with queries

Exercises

20:00

8) Collect information on 300 users that

- are associated with the word "PhD"
- but not with the word "rstats"

9) Collect the latest 300 tweets that

- include the word "PhD"
- but not the word "rstats"

10) Take some time to explore the resulting data frames

- how do they compare to each other?

11) Conduct one or more searches that interest you

- for tweets and/or users

Data Collection — Notes — `rate_limit`

- Check rate limits at any time
  - for all operations
  - for a specific operation
    - e.g., searching tweets with the `search_tweets` function

```
rate_limit(token = NULL,
           query = NULL,
           parse = TRUE)
```

- Note that
  - if no token is specified, the function uses the `rstats2twitter` app
  - rate limits decrease as you use them, increase again after a time threshold
  - Twitter Developer terms do not allow for multiple tokens to be used for the same project

Data Collection — Notes — `rate_limit`

Check you remaining rate limits, for all operations

```
all_limits_now <- rate_limit()
```

Check your remaining rate limits for the `search_tweets` function

```
all_limits_now <- rate_limit(query = "search/tweets")
```

Exercises

10:00

12) Check all your remaining rate limits

- hint: assign the call `rate_limit` to a name
  - so that you can observe the resulting data frame better

13) Check your remaining limits for the `search_tweets` function

- assign it to a name so that you can remember

14) Collect the top 5000 tweets that

- include the word "PhD"
- and posted by a verified account

15) Check your remaining limits for the `search_tweets` function again

- how do they compare to the results from exercise 13?

Data Collection — Notes — `retryonratelimit`

- `retryonratelimit` is not available for all functions in the package
  - e.g., `search_users`
  - does not mean you will not run into limits
- You can create your own safety net
  - e.g., with loops, where the system sleeps between iterations
    - until a specific rate limit increases

Data Collection — Notes — retryonratelimit — Iteration

```
datalist <- list() # create an empty list, to be filled later
for(i in 1:length(df_users$screen_name)) { # for one user, in the data frame df_users, at a time
  if (rate_limit(query = "application/rate_limit_status", token = tw_token)$remaining > 2 &
      rate_limit(query = "get_timeline", token = tw_token)$remaining > 20) { # if your are still

  dat <- get_timeline(df$screen_name[i], n = 3200, # collect the tweets
                      token = tw_token)

  datalist[[i]] <- dat # fill the list with data, for one user at a time
  }else{ # if there is no limit, wait a little
  wait <- rate_limit(query = "get_timeline")$reset + 0.1
  Sys.sleep(wait * 60)

  }
}
df_tweets <- as.data.frame(do.call(rbind, datalist)) # put all data in one data frame
```

```
lookup_
```

Data Collection — `lookup_tweets`

- Collect data on one or more tweets
  - whose status ID you already know
  - limited with 90,000 posts per 15 minutes
    - there is no `retryonratelimit` argument

```
lookup_tweets(statuses,
              parse = TRUE,
              token = NULL)
```

- Note that
  - this function would be useful for replicating studies
  - status IDs are visible on browsers
    - if you click on a specific tweet

Data Collection — `lookup_tweets`

Collect data on one or more status IDs

```
lookup_tweets(statuses = c("567053242429734913", "266031293945503744", "440322224407314432"))
```

Collect data on status IDs in a data frame

```
lookup_tweets(statuses = df_ids$status_id)
```

Data Collection — `lookup_users`

- Collect data on one or more users
  - whose user ID or username you already know
  - limited with 90,000 users per 15 minutes
    - there is no `retryonratelimit` argument

```
lookup_users(users,
             parse = TRUE,
             token = NULL)
```

- Note that
  - usernames can change
  - rely on user IDs where possible

Data Collection — `lookup_users`

Collect data on one or more status IDs

```r
lookup_users(users = c( "drob", "hadleywickham", "JennyBryan"))
```

Collect data on status IDs in a data frame

```r
lookup_users(users  = df_usernames$screen_name)
```

Data Collection — `lookup_friendships`

- Collect data on friendship status of two users
  - e.g., whether they follow each other
  - whose user ID or username you already know
  - limited with 90,000 users per 15 minutes
    - there is no `retryonratelimit` argument

```
lookup_friendships(source,
                   target,
                   parse = TRUE,
                   token = NULL)
```

- Note that
  - usernames can change
  - rely on user IDs where possible

Exercises

15:00

16) Find a status ID through your browser and look it up in R

17) Find a user ID through your browser and look them up in R

18) Look up a subset of tweets whose ids stored in `status_ids.rds`

19) Look up a subset of users whose usernames stored in `mps.csv`

20) Check the friendship status of two MPs in the dataset

- hint: not all MPs are on Twitter, but most are
  - e.g., check if the 10th and 20th observations follow each other

`get_`

Data Collection — `get_timeline`

- Collect the latest posts from one or more users
    - specified by username or user IDs, with the `user` argument
    - limited to 3,200 tweets per user-timeline
        - there is no `retryonratelimit` argument
    - returns the most recent only, if there is more

```
get_timeline(user,
             n = 100,
             max_id = NULL,
             home = FALSE,
             parse = TRUE,
             check = TRUE,
             token = NULL,
             ...
)
```

Data Collection — `get_timeline`

Collect the most recent 200 tweets by David Robinson

- e.i., tweets from the user-timeline of one user

```
search_tweets(user = "drob",
              n = 200)
```

Data Collection — `get_timeline`

- Collect the most recent posts by David Robinson and Hadley Wickham
    - tweets from the user-timeline of multiple users

```
get_timeline(user = c("drob",
                      "hadleywickham"),
             n = 200)
```

- Note that
    - this results in a dataframe of 400 observations
    - 200 from each specified user
    - with increasing number of users, you are likely to run out of rate limit

Data Collection — `get_timeline` — Home Timeline

- The package documentation suggests that `get_timeline` can also retrieve home-timelines
    - i.e., the tweets that appear on a given user's home, as posted by accounts followed by that user
        - if the `home` argument is set to `TRUE`
- This does not seem to be true
    - this code returns my home-timeline, not Wickham's
        - from the last 6 to 9 days
    - the `user` argument is ignored when `home = TRUE`
    - but the `user` argument cannot be missing

```r
get_timeline(user = "hadleywickham",
             n = 200,
             home = TRUE)
```

Exercises

10:00

21) Collect the most recent tweets posted by three users

- 100 from the first user, 200 from the second, and 300 from the third
- hint: see the function documentation on how to vary the `n` argument by user

22) Collect as many tweets as possible from your own home-timeline

23) Collect data from timelines of the first five MPs in `mps.csv`

- collect as many tweets as possible

Data Collection — `get_followers`

- Get a list of followers, following one user
    - returns a single column of user IDs, not usernames
    - limited with 75,000 followers per 15 minutes
        - use `retryonratelimit = TRUE` to surpass the limit

```
get_followers(user,
              n = 5000,
              page = "-1",
              retryonratelimit = FALSE,
              parse = TRUE,
              verbose = TRUE,
              token = NULL
)
```

- Note that
    - this function does not accept multiple users
    - it can be combined with `lookup_users` if usernames are needed

Data Collection — get_followers

Get a list of Hadley Wickham's followers on Twitter

```
get_followers(user = "hadleywickham",
              n = 10000,
              retryonratelimit = TRUE)
```

Data Collection — `get_friends`

Get a list of users, followed byone or more users

- this returns a single column of user IDs, not usernames
- limited to 15 users
    - use `retryonratelimit = TRUE` to surpass the limit

- limited also to 5,000 followers per user
    - here `retryonratelimit = TRUE` does not help

- use the `page` argument instead to surpass the limit
    - learn the correct value with the `next_cursor` argument

```
get_friends(users,
            n = 5000,
            retryonratelimit = FALSE,
            page = "-1",
            parse = TRUE,
            verbose = TRUE,
            token = NULL
)
```

Data Collection — `get_friends`

Collect a list of users followed by Jenny Bryan and Hadley Wickham on Twitter

- with 20 friends from each

```
get_friends(users = c("hadleywickham", "JennyBryan"),
            n = 20)
```

Exercises

07:30

24) Collect a list of accounts following *Universität Luzern*

- search for the Unversity's username with `search_users`
- search for the followers `get_followers`

25) Collect a list of accounts that *Universität Luzern* follows

- find out more about these accounts with `lookup_users`
    - hint: use `df_friends$user_id` as the `users` argument

26) Check your rate limits

Data Collection — `get_favorites`

Collect tweets liked by one or more users

- by username or user IDs, with the `user` argument
- limited to 3,000 likes
  - there is no `retryonratelimit` argument
- returns the most recent only, if there is more

```
get_favorites(user,
              n = 200,
              since_id = NULL,
              max_id = NULL,
              parse = TRUE,
              token = NULL
)
```

Data Collection — `get_favorites`

Collect a list of tweets liked by Jenny Bryan

```
get_favorites(user = "JennyBryan")
```

Data Collection — `get_retweets`

- Collect information on the retweets of one tweet
  - using the `status_id` argument
  - available in the data frames returned by many functions in the package
    - e.g., `get_timeline`
    - also available on browsers
  - limited to 100 retweets
    - even if there might be more

```
get_retweets(status_id,
             n = 100,
             parse = TRUE,
             token = NULL,
             ...
)
```

Data Collection — `get_retweets`

- Collect the most recent 50 retweets
  - of the post announcing the Academic Research product track

```
get_retweets(status_id = "1354143047324299264")
```

Exercises

07:30

27) Collect a list of favorites by three users

- compare and contrast the resulting data with the same information available on browser

28) Collect a list of accounts retweeting a recent tweet of yours

- compare and contrast the resulting data with what you can see on a browser

Data Collection — `get_trends`

- Collect information on twitter trends
  - by town or country, specified with
    - the `woeid` argument,[*] or
    - the `lat` and `long` arguments

- Note that
  - not all locations have trending data
  - use the `trends_available` function to check availability
    - with no argument

```
get_trends(woeid = 1,
           lat = NULL,
           lng = NULL,
           exclude_hashtags = FALSE,
           token = NULL,
           parse = TRUE
)
```

[*] It stands for "where on earth identifier", which is 44418 for London. Google for more!

Data Collection — `get_trends`

Collect the trends data for London

- using the `woeid` argument

```
get_trends(woeid = 44418)
```

Collect the same trends data for London

- using the `lat` and `long` arguments instead

```
get_trends(lat = "51.50", lng = "0.12")
```

Exercises

07:30

29) Collect a list of places where the trends data is available

- hint: use the `trends_available` function

30) Collect the lists of trends for two locations

- compare and contrast the resulting data for two locations

31) Collect the list of trends for your location

- compare and contrast the resulting data with what you see on your browser

`lists_`

Data Collection — `lists_memberships`

- Collect data on lists, where one or more users are listed
  - i.e., the lists where a user appears
  - limited to 200 lists

```
lists_memberships(user = NULL,
                  n = 200,
                  cursor = "-1",
                  filter_to_owned_lists = FAL
                  token = NULL,
                  parse = TRUE,
                  previous_cursor = NULL
)
```

Data Collection — `lists_memberships`

Collect data on lists where Jenny Bryan is listed

```
lists_memberships(user = "JennyBryan")
```

Collect data on lists where Jenny Bryan and/or Hadley Wickham is listed

```
lists_memberships(user = c("JennyBryan", "hadleywickham"))
```

Data Collection — `lists_members`

- Collect data on users listed in one list
  - specify the list with the `list_id` argument
    - e.g., with data from `lists_memberships`
  - or the `owner_user` and `slug` arguments together
  - limited to 5,000 members

```
lists_members(list_id = NULL,
              slug = NULL,
              owner_user = NULL,
              n = 5000,
              cursor = "-1",
              token = NULL,
              parse = TRUE,
              ...
)
```

- Note that
  - lists also appear at twitter.com/USERNAME/lists
  - see, for example, twitter.com/tweetminster/lists

Data Collection — `lists_members`

Collect data on the list of MPs in the House of Commons

- using the `list_id` argument

```
lists_members(list_id = "1405362")
```

Collect the same data, with different arguments

- using the `owner_user` and `slug` arguments

```
lists_members(owner_user = "tweetminster", slug = "UKMPs")
```

Data Collection — `lists_statuses`

Collect tweets from the timeline of a list

- i.e., tweets posted by those listed on a given list
- specify the list with the `list_id` argument
- or the `owner_user` and `slug` arguments together

```
lists_statuses(list_id = NULL,
               slug = NULL,
               owner_user = NULL,
               since_id = NULL,
               max_id = NULL,
               n = 200,
               include_rts = TRUE,
               parse = TRUE,
               token = NULL
)
```

Data Collection — `lists_statuses`

Collect tweets posted by the members of the UKMPs list

- using the `list_id` argument

```
lists_statuses(list_id = "1405362")
```

Collect the same data, with different arguments

- using the `owner_user` and `slug` arguments

```
lists_statuses(owner_user = "tweetminster", slug = "UKMPs")
```

Data Collection — lists_subscribers

Collect data on users subscribed to a given list

- i.e., users who are following a list
- specify the list with the list_id argument
- or with the owner_user and slug arguments
- limited to 5,000 users

```
lists_subscribers(list_id = NULL,
                  slug = NULL,
                  owner_user = NULL,
                  n = 20,
                  cursor = "-1",
                  parse = TRUE,
                  token = NULL
)
```

Data Collection — `lists_subscribers`

Collect data on users subscribed to the UKMPs list

- using the `list_id` argument

```
lists_subscribers(list_id = "1405362")
```

Collect the same data, with different arguments

- using the `owner_user` and `slug` arguments

```
lists_subscribers(owner_user = "tweetminster", slug = "UKMPs")
```

Data Collection — `lists_subscriptions`

Collect data on the lists a user is subscribed to

- specify the user with `user` argument
    - takes user ID or username
- limited to 1,000 subscriptions

```r
lists_subscriptions(user,
                    n = 20,
                    cursor = "-1",
                    parse = TRUE,
                    token = NULL)
```

Data Collection — `lists_subscriptions`

Collect data on the lists that Aimee Paige is subscribed to

```
lists_subscriptions(user = "AimeePaige")
```

Exercises

10:00

32) Collect data on lists where Hadley Wickham is listed

- hint: his username is "hadleywickham"

33) For one of these lists, see who else is listed with Hadley Wickham

- compare and contrast this data with what you can see on a browser

34) Collect the latest posts from that list

35) Collect data on users subscribed to that list

36) For one of these users, see if they are subscribed to any other lists

`stream_tweets`

Data Collection — `stream_tweets`

- Collect tweets as they are posted real time
    - about 1% of all new public Tweets, randomly chosen
    - set the length of search with the `timeout` argument
- The search can be limited with the `q` argument
    - up to 400 keywords
    - up to 5,000 user IDs or usernames
    - location coordinates of geographical boxes
        - not two, but four coordinates
- Note that
    - this function uses the stream APIs
    - unlike any other function covered so far

```r
stream_tweets(q = "",
              timeout = 30,
              parse = TRUE,
              token = NULL,
              file_name = NULL,
              verbose = TRUE,
              ...
)
```

Data Collection — `stream_tweets`

- Collect a random sample of tweets being sent
  - continuously

```
stream_tweets(q = "",
              timeout = Inf)
```

- Note that
  - the `timeout` function can be set to infinity
  - you are likely to run into connection problems at some point

Data Collection — `stream_tweets`

- Collect a random sample of tweets being sent
    - for 30 seconds

```r
stream_tweets(q = "",
              timeout = 30)
```

- Note that
    - `timeout` values are otherwise in seconds

Data Collection — stream_tweets

- Collect a random sample of tweets being sent
    - for 30 seconds
    - filtered by a search query
- Note that
    - q accepts a comma separated character string

```
stream_tweets(q = "switzerland, schweiz,
                   suisse, svizzera",
              timeout = 30)
```

Data Collection — stream_tweets

- Collect a random sample of tweets being sent
  - for 30 seconds
  - filtered by usernames

```
stream_tweets(q = c("UniLuzern", "hslu",
                   "phluzern"),
              timeout = 30)
```

- Note that
  - q accepts a comma separated list

Data Collection — stream_tweets

- Collect a random sample of tweets being sent
  - for 30 seconds
  - filtered by coordinates

```
stream_tweets(q = c(6.02, 45.77,
                    10.44, 47.83),
              timeout = 30)
```

Exercises

10:00

37) Stream for all tweets, for 30 seconds

- observer the outcome
    - observe the outcome of each step below as well
- hint: assign each stream to a different name, so that you can compare

38) Limit your stream to Switzerland

- and stream for 60 seconds

39) Further limit your stream by a popular keyword

- e.g., "und"
- stream for 60 more seconds

40) Further limit your stream to a not so popular word

- e.g., "Luzern"
- stream for 60 more seconds

Part 4. Data Preperation

Back to the contents slide.

Data Preperation — Overview

- the `rtweet` package does a very good job 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
    - first for strings
    - than the rest

- In this part, please follow along the examples with actual data
  - you can use `tweets.rds` or your own dataset

Data Preperation — Overview — Strings

- Most researchers would be interested in textual data on Twitter
    - 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 a 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 `dplyr` package (Wickham et al, 2020 for most other data operations
  - part of the `tidyverse` family
  - you might have a favourite already
    - no need to change as long as it does the job

Data Preperation — Remove Mentions

04:00

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

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

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

Data Preperation — Remove Hashtags

04:00

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

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

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

Data Preperation — Remove Links

04:00

```
tweet <- "These from @handle1 are #socool. 👏 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 expressions

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

04:00

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

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

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

Data Preperation — Remove Punctuations

04:00

```
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 👉 tcoaq7MJJ1 👉 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

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

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

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

04:00

Data Preperation — Remove Repeated Whitespace

04:00

```
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

04:00

```
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`

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

04:00

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

04:00

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

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

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

04:00

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")
```

04:00

Data Preperation — 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")
```

04:00

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")
```

04:00

Data Preperation — Remove Stop Words

04:00

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

```r
# 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"))
```

Part 5. Data Anaysis: User-Based

Back to the contents slide.

Part 6. Data Anaysis: Tweet-Based

Back to the contents slide.

Part 7. Data Anaysis: Machiene Learning

Back to the contents slide.

Part 8. Next Steps

Back to the contents slide.

Twitter — Apply for a Developer Account

- On developer.twitter.com, click `Apply`
  - at the top-right corner of the page*
  - follow the instructions on consecutive pages

*It takes a few days for Twitter to review and hopefully approve your request to have an account. You might have created an account before. In that case, you will see `Developer Portal` instead of `Apply`.

Twitter — Apply for a Developer Account

- On developer.twitter.com, click `Apply`
  - at the top-right corner of the page*
  - follow the instructions on consecutive pages

- Write a detailed case for your intended use of Twitter APIs
  - not just that you are attending this workshop
  - a quick application is likely to slow the review process
    - as Twitter often asks for more details via email
  - rejection is also a possibility

- Carefully review the Twitter's Developer Agreement and Policy
  - especially important if you will develop an app that
    - others will use
    - will write as well as read Twitter data

Twitter — Register an App

- On developer.twitter.com/en/portal/projects-and-apps, click `+ Create App`
    - follow the instructions on consecutive pages
        - you will need a unique name for your app
    - note that, once the app is registered, you are provided with keys and tokens
        - you will use these for authentication
            - if you choose not to authenticate through `rtweet`'s own app, called `rstats2twitter`
            - more on this in Part 3
- does not mean you have to create an actual app
    - e.g., an app for smart phones

Twitter — Keys and Tokens — Notes

- Keys and tokens are personal
  - should not be shared with others
    - e.g., through replication files, when asking a question on Stack Overflow
- Keys and tokens can be re-generated anytime
  - on your applications page, under the `Keys and tokens` tab
  - this requires updating your R script with the new tokens as well
- Twitter allows for further, optional settings involving keys and tokens
  - relevant, mostly, for apps to do more than just collecting tweets
  - e.g., settings for 3-legged OAuth
    - like `rstats2twitter`, to allow for other users to authenticate through a browser pop up
  - not covered in this workshop

Twitter — Keys and Tokens — Notes — Definitions

- Consumer key and Consumer secret
    - identifiers for an application
    - provide project authorization
        - identify the application
        - check whether the application has been granted access
    - like your application's username and password
- Access token and Access token secret
    - identifier for users of an application
        - this may or may not include anyone other than the owner
        - e.g., for `rstats2twitter`, there are many users
    - provide user authentication
        - identify the user
        - check whether the user should have access to a request

References

Back to the contents slide.

References

Benoit, K., Watanabe, K., Wang, H., Nulty, P., Obeng, A., Müller, S., Matsuo, A., Lua, J. W., Kuha, J., & Lowe, W. (2020). quanteda: Quantitative Analysis of Textual Data. R package, version 2.1.2.

Cheng, J., & Chang, W. (2020). httpuv: HTTP and WebSocket Server Library. R package, version 1.5.4.

Gentry, J. (2015). twitteR: R Based Twitter Client. R package, version 1.1.9.

Jungherr, A. (2016). Twitter use in election campaigns: A systematic literature review. Journal of Information Technology & Politics, 13(1), 72–91.

Mellon, J., & Prosser, C. (2017). Twitter and Facebook are not representative of the general population: Political attitudes and demographics of British social media users. Research & Politics, July-September 2017, 1–9.

Robinson, D., & Silge, J. (2021). tidytext: Text mining using 'dplyr', 'ggplot2', and other tidy tools. R package, version 0.3.0.

Sievert, C., Parmer, C., Hocking, T., Chamberlain, S., Ram, K., Corvellec, M., & Despouy, P. (2020). plotly: Create Interactive Web Graphics via 'plotly.js'. R package, version 4.9.3.

References

Silge, J., & Robinson, D. (2017). Text mining with R: A tidy approach. O'Reilly. Open access at

Silva, B. C., & Proksch, S. O. (2020). Fake It 'Til You Make It: A Natural Experiment to Identify European Politicians' Benefit from Twitter Bots. American Political Science Review, 1-7.

Sinnenberg, L., Buttenheim, A. M., Padrez, K., Mancheno, C., Ungar, L., & Merchant, R. M. (2017). Twitter as a tool for health research: A systematic review. American Journal of Public Health, 107(1), 1–8.

Umit, R. (2017). Strategic communication of EU affairs: An analysis of legislative behaviour on Twitter. Journal of Legislative Studies, 23(1), 93–124.

Wickham, H. (2019). stringr: Simple, Consistent Wrappers for Common String Operations. R package, version 1.4.0.

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.

Xie, Y. (2020). xaringan: Presentation Ninja. R package, version 0.19.

The workshop ends here.

Congradulations for making it this far, and

thank you for joining me!

Back to the contents slide.