

# Studying News Use with Computational Methods

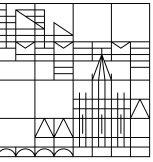
## Data Collection in R, Part I: Collecting Social Media Data

Julian Unkel  
University of Konstanz  
2021/05/10



# Agenda

---



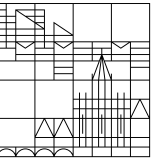
As we have seen in the previous sessions, social media play a key role for users' news consumption.

By their design, they record human behavior (and thus news use behavior) as *digital traces* of users engaging and interacting (clicking on news links, liking news posts, writing comments on said posts) with social media posts.

In this session, we will deal with common approaches to collect digital trace data from social media.

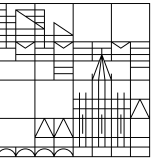
Our agenda today:

- API requests
  - Basics
  - Calling APIs with R
- API wrapper packages
  - Basics
  - Example: Querying the Twitter API with `rtweet`
  - Twitter's Academic API track
- Facepager
- Social monitoring services
  - CrowdTangle
  - Other commercial options
- Grey area tools



# API requests

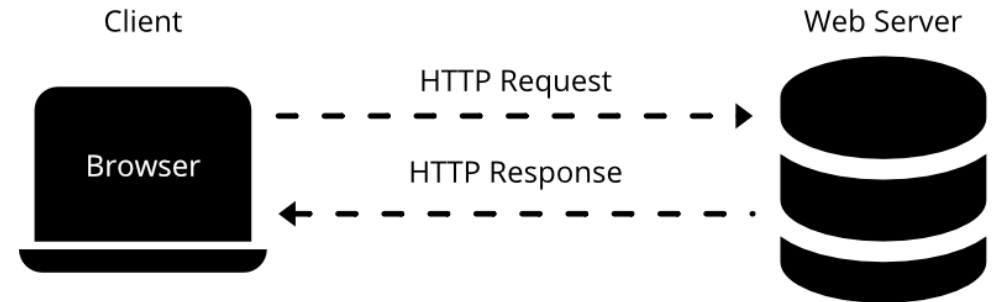
---



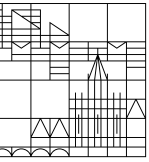
# Basics: HTTP requests

Think of accessing data on web servers (e.g., by opening a web site in a browser) via **HTTP** (Hypertext Transfer Protocol) as ordering a package via mail:

- First, we place an order with our client, for example by typing an URL into a browser (*Request*)
- The server sends our client a package (*Response*), consisting of two parts:
  - *Header*: Sort of like the packing slip; contains lots of meta information, for example whether our package was delivered successfully
  - *Body*: The actual content of the package, for example an HTML file



HTTP requests in the client-server model



# Basics: HTTP methods & status codes

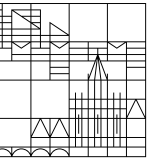
---

There are several different **request methods**, most importantly:

- *GET*: Request data retrieval
- *POST*: Request sending (=posting) data (e.g., web forms)

Response headers contain three-digit **status** codes that tell us if everything went okay or what went wrong. Most importantly:

- *2xx*: Success! We usually want code *200*, telling us that everything is OK
- *4xx*: Oh no, client error! This means: The problem is caused by the client (i.e., us). You have probably already encountered these:
  - *403*: Forbidden - client is not allowed to access the requested resource
  - *404*: Not found - client requested a resource that is not available on the server
- *5xx*: Oh no, server error! For example, *503* (service unavailable) tells us that the server is (currently) too busy to handle our request.



# Basics: Writing HTTP requests in R

We can write our own HTTP requests in R using the `httr` package. Let's install it if we haven't done so already:

```
install.packages("httr")
```

After loading the package, we can use functions named after the request methods to send HTTP requests. Let's request your [SEDS home page](https://www.wiwi.uni-konstanz.de/studium/master-of-science/seds/).

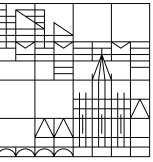
```
library(httr)
seds_resp <- GET("https://www.wiwi.uni-konstanz.de/studium/master-of-science/seds/")
```

The response is a list object containing the 'whole package'. Let's first take a look at the status code:

```
status_code(seds_resp)
```

```
## [1] 200
```

Everything went OK!



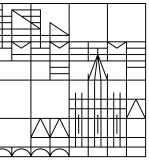
# Basics: Writing HTTP requests in R

We can now investigate the body - the actual content - of our response object:

```
content(seds_resp)

## {html_document}
## <html lang="de">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="faculty">\n\n    <nav class="anchormenu" aria-label="Sprungm ...
## [3] <script src="https://www.wiwi.uni-konstanz.de/typo3temp/assets/compressed ...
## [4] <script src="https://www.wiwi.uni-konstanz.de/typo3temp/assets/compressed ...
## [5] <script type="text/javascript">\n/*<![CDATA[*\n/*TS_inlineFooter*\n\t\t ...
## [6] <script type="text/javascript">\r\n  var _paq = _paq || [];\r\n  /* track ...
## [7] <script type="text/x-mathjax-config">\r\n      MathJax.Hub.Config({\r\n  ...
## [8] <script type="text/javascript" src="/MathJax/MathJax.js?config=TeX-AMS-HT ...
```

The first lines tell us that we have successfully requested an `html_document`. We will deal with working with HTML documents in the next session. But you can already see the first level of contents of the HTML file, namely a `<head>` with meta information, the `<body>` containing all the text of the website (not to be confused with the header and the body of the *response*), and various `<script>`s.



# Basics: What's in a URL?

---

We access resources on the web by providing the corresponding **URL** (Uniform Resource Locator). Let's take a closer look:

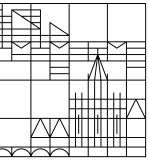
`https://www.google.de/search?q=seds`

- **Scheme**: The scheme specifies the protocol that we are using (HTTPS is a secure version of HTTP)
- **Domain**: The domain name indicates the web server that is being requested
- **Path**: The path points to the specific resource on the web server, just like the folder structure on your computer. It can include the file name (e.g., /path/to/page.html), but on web pages, this is usually handled on the server side.
- **Parameters**: Web servers may accept parameters in a key=value combination to dynamically provide content for a specific resource. They are separated from the path by a single ?. Multiple parameters can be linked by & (e.g., ?key1=value1&key2=value2).

In the above example, we are thus requesting the resource at path /search with the parameter q set to seds of the domain www.google.de via the HTTPS protocol: <https://www.google.de/search?q=seds>

We can add other parameters to change the output: <https://www.google.de/search?q=seds&start=10>



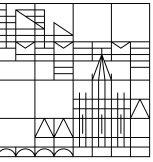


# Basics: JSON

---

Web-APIs usually do not return HTML files, but more structured data, most often in the **JSON** (JavaScript Object Notation, pronounced as in "[Jason and The Argonauts](#)") format. This open, human-readable and flexible text format is able to represent various (also hierarchical and nested) data structures in attribute-value pairs. We will deal with JSON files soon, but the example from [Wikipedia](#) probably already tells you all the basics you need to know:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    }
  ],
}
```



**APIs** (application programming interfaces) are interfaces of software applications to communicate (e.g., share data) with other applications.

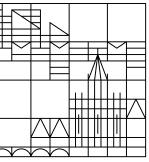
In our context, the term usually references Web APIs, interfaces of web applications/servers using request-response systems.

All Web APIs are different and thus require some engagement with the (hopefully helpful) documentation:

- access requirements
- endpoints and parameters
- response data structures

But all Web APIs are the same:

- we write an HTTP request to the API URL
- the API responds by providing the requested data (usually in JSON, XML, or CSV)



# APIs: Authentication & Rate limits

---

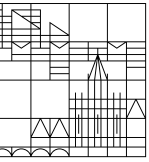
Access to APIs is regulated in many different ways, for example:

- Open (can be called without any authentication)
- Username/password
- API key (often passed as a URL parameter)
- OAuth (a protocol for generating user- or session-specific authentication tokens)

In all but the first case, this requires (often reviewed or even paid) registration.

APIs usually manage access by setting **rate limits**, defining how many calls a user can make within a given time period. Exceeding the rate limit may result in:

- Request errors (e.g., 429 Too Many Requests)
- Request throttling
- Fees (in commercial APIs)



# APIs: Endpoints & Parameters

---

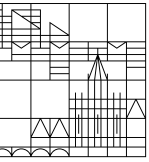
Most APIs offer several **endpoints** for specific actions. Endpoints are thus a combination of an URL path and an HTTP request method.

For example, some endpoints of the Twitter API v2, using the base URL `https://api.twitter.com` are:

- GET `/2/tweets`: Get information about tweets
- GET `/2/users/:id/tweets`: Get tweets of the Twitter user with the id `:id`
- POST `/2/users/:id/likes`: Like a tweet on behalf of the Twitter user with the id `:id`

Calls to endpoints are then usually specified further by providing **parameters**, either as URL parameters or, for example when using the POST method, in the request body.

- For GET `/2/tweets`, we would add a list of tweet IDs to our call by adding the parameter `ids` (e.g, GET `https://api.twitter.com/2/tweets?ids=id1,id2,id3`)
- For POST `/2/users/:id/likes`, we would add the id of the target tweet in the request body in JSON format (e.g., `{"tweet_id": "id1"}`)



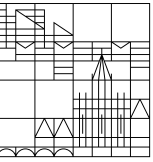
# APIs: Social Media

---

[Programmable Web](#) provides an overview of about 25,000 APIs you may want to use.

Common social media APIs are:

- **Twitter API** (<https://developer.twitter.com/en/docs/twitter-api>)
  - Access to Twitter tweets, timelines, profiles, etc.
  - *Will I get access?* Likely, through the [Academic Research track](#)
- **Facebook Graph API** (<https://developers.facebook.com/docs/graph-api>)
  - Access to Facebook posts, comments, profiles, etc.
  - *Will I get access?* Unlikely (but wait for the rest of the session)
- **Facebook Ad Library API** (<https://www.facebook.com/ads/library/api>)
  - Access to political Facebook ads (content, reach, spendings, etc.)
  - *Will I get access?* Very likely
- **Instagram Graph API** (<https://developers.facebook.com/docs/instagram-api>)
  - Access to Instagram posts, profiles, etc.
  - *Will I get access?* Unlikely (but wait for the rest of the session)
- **Reddit API** (<https://www.reddit.com/dev/api/>)
  - Reddit submissions, comments, etc.
  - *Will I get access?* Actually haven't tried it (because see next slide)



# The Pushshift API

---

**Pushshift** is a privately maintained, open Reddit dataset, ingesting Reddit content in real time. For technical details, see the paper [The Pushshift Reddit Dataset](#).

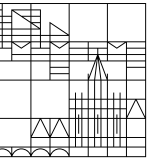
The dataset is accessible, among other pathways, via a public, open API: <https://api.pushshift.io>, documented at <https://github.com/pushshift/api>.

Main advantages over the 'real' Reddit API:

- No authentication required
- Larger response object limits
- Very forgiving rate limits

Drawbacks:

- Unclear state of development, incomplete documentation
- Some issues with deleted posts
- Likely coverage issues



# Calling the Pushshift API with R

Let's write our first API call! The base URL of the Pushshift API is <https://api.pushshift.io>, so we might want to store this for easier retrieval:

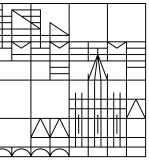
```
ps_base <- "https://api.pushshift.io"
```

As seen in the [documentation](#), the API currently offers two endpoints, both for GET methods:

- `/reddit/search/comment`: Searching individual comments
- `/reddit/search/submission`: Searching submissions

Let's store them as well:

```
ps_comment <- "/reddit/search/comment"  
ps_submission <- "/reddit/search/submission"
```



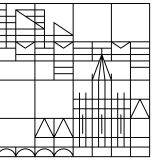
# Calling the Pushshift API with R

The `GET()` function of `httr` offers several arguments to construct a request from the different parts of the call URL. We can use the `url` argument to add the base URL (domain), define the path using the `path` arguments, and add several parameters by passing a named list of key/value pairs to the argument `query`.

In the following, we call the submission endpoint of the API, searching for the latest 100 submissions in the [r/news](#) subreddit that contain the word "biden" in the submission title:

```
ps_resp <- GET(url = ps_base,  
              path = ps_submission,  
              query = list(subreddit = "news",  
                           title = "biden",  
                           size = 100)  
              )
```



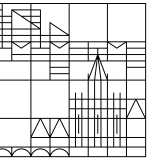


# Calling the Pushshift API with R

Let's take a look:

```
ps_resp
```

```
## Response [https://api.pushshift.io/reddit/search/submission?subreddit=news&title=biden&size=100]
##   Date: 2021-04-29 15:27
##   Status: 200
##   Content-Type: application/json; charset=UTF-8
##   Size: 478 kB
## {
##   "data": [
##     {
##       "all_awardings": [],
##       "allow_live_comments": false,
##       "author": "paulfromatlanta",
##       "author_flair_css_class": null,
##       "author_flair_richtext": [],
##       "author_flair_text": null,
##       "author_flair_type": "text",
##     }
##   ]
## }
```



# Calling the Pushshift API with R

We can 'unpack' the response body by using the `content()` function:

```
ps_content <- content(ps_resp, type = "application/json")
str(ps_content, max.level = 1)
```

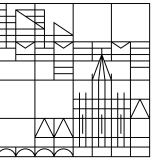
```
## List of 1
## $ data:List of 100
```

Further moving through the list levels, we can access information about the individual entries:

```
ps_data <- ps_content$data
ps_data[[1]]$title
```

```
## [1] "Biden administration bans menthol cigarettes"
```

(Your results may vary as I'm using a cached response in this presentation.)



# Calling the Pushshift API with R

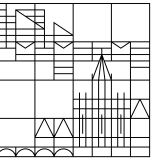
Using some [Tidyverse](#) functions - specifically, from the `purrr` package for functional programming - we can quickly transform the response to a rectangular dataframe:

```
library(tidyverse)
fields <- c("id", "title", "created_utc", "permalink", "url")
ps_data %>%
  map_dfr(magrittr::extract, fields)
```

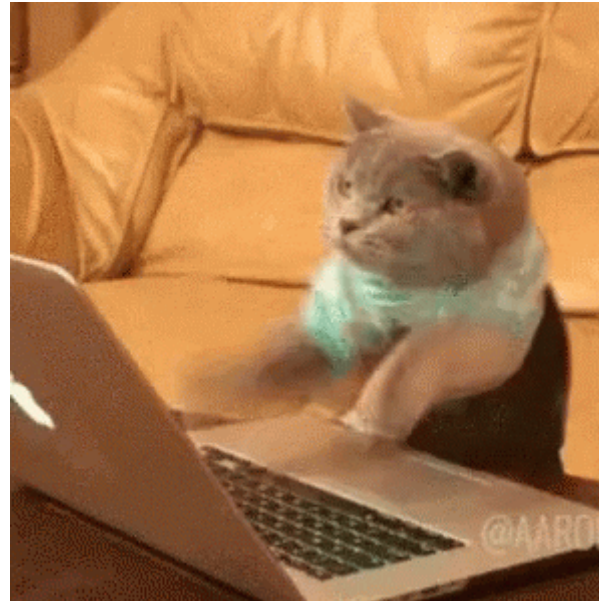
```
## # A tibble: 100 x 5
##   id      title      created_utc permalink      url
##   <chr> <chr>          <int> <chr>          <chr>
## 1 n16pfm Biden administrat~ 1619709687 /r/news/comments/n16~ https://www.cbsn~
## 2 n15u35 Biden Tax Plan Le~ 1619707290 /r/news/comments/n15~ https://www.wsj.~
## 3 n14vxn Biden Seeks Shift~ 1619704570 /r/news/comments/n14~ https://www.nyti~
## 4 n14tp9 President Biden p~ 1619704385 /r/news/comments/n14~ https://abcnews.~
## 5 n148h3 Indian-Americans ~ 1619702541 /r/news/comments/n14~ http://news.meim~
## 6 n13nyb Women Make Histor~ 1619700751 /r/news/comments/n13~ https://www.scr~
## 7 n12jtg &#x27;White S~ 1619696752 /r/news/comments/n12~ https://www.news~
## 8 n11j6p At 100 Days, Bide~ 1619692414 /r/news/comments/n11~ https://newsnati~
## 9 n112x0 Ex-Trump aide Ste~ 1619690395 /r/news/comments/n11~ https://apple.ne~
## 10 n10wv1 Joe Biden Unveils~ 1619689554 /r/news/comments/n10~ https://www.rayz~
## # ... with 90 more rows
```

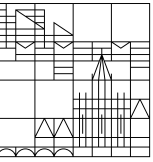
# Calling the Pushshift API with R

---



**Exercise 1: Write your own call:** Try to obtain the *first* 50 posts that were posted in German-language subreddit [r/de](https://www.reddit.com/r/de/). Consult the documentation for help on the necessary parameters: <https://github.com/pushshift/api>





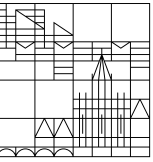
# APIs: Iteration & Pagination

---

If an API call matches more results than can be returned with a single response, we need an iteration mechanism to retrieve all results. For example, if the call matches 500 results and the response object limit is 100, we need to make (at least) 5 calls to retrieve all results. Keep rate limits in mind when iterating over results!

Most APIs provide one or more of the following forms of pagination:

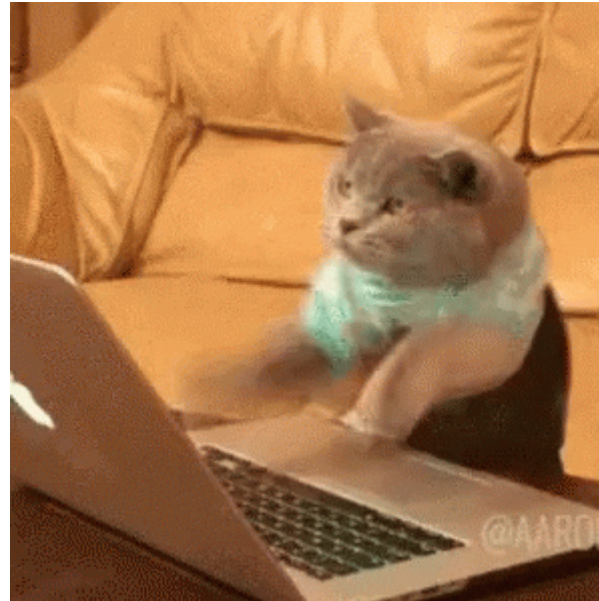
- **Pages:** Results are spread over pages (e.g., results 1 to 100 on page 1, 101 to 200 on page 2). We can then iterate over results by simply adding 1 to the page number (e.g., by adding the query parameter `page=page_num`) in each successive call.
- **Keys:** Results are ordered by ascending/descending keys (e.g., Tweet IDs). We can then iterate over results by retrieving the minimum/maximum key of each call and requesting results below/above said key in the next call.
- **Timestamps:** Results are ordered by UNIX timestamps or [DIN ISO 8601](#) date formats. We can then iterate over results by retrieving the minimum/maximum timestamp of each call and requesting results before/after said timestamp in the next call (but beware that multiple results can have the same timestamp).
- **Cursors:** Results are spread over pages, but single pages are identified by an opaque cursor (i.e., usually a seemingly random sequence of characters) instead of integer numbers. We can then iterate over results by retrieving the cursor for the next/previous page which should be provided in the response.

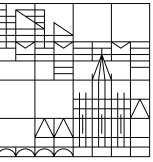


# Calling the Pushshift API with R

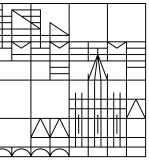
---

**Exercise 2: Pagination:** Try to obtain the *latest* 200 comments posted in the [r/politics](https://www.reddit.com/r/politics) subreddit that contain the phrase "lol". Consult the documentation for help on the necessary parameters: <https://github.com/pushshift/api>





# API wrapper packages



# API wrapper packages

---

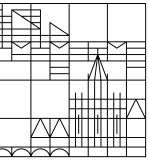
**API wrappers** are language-specific packages that simplify calling specific APIs. In addition to providing convenience functions for the actual calls, they sometimes also include pagination and rate limit handling.

You will probably find R wrapper packages for most common APIs. If in doubt, just google "r + API name".

If there is none, why not do some good and create your own wrapper package? Some resources:

- [CRAN: Best practices for API packages](#)
- [Colin Fay: How to build an API wrapper package in 10 minutes](#)





# Example: rtweet

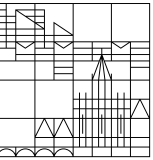
---

`rtweet` is probably the most common Twitter API wrapper package for R (and also somewhat *official*, as it is co-developed by the RStudio team).

Results follow tidy data conventions and are thus easily processed further; furthermore, the package can be used without access to Twitter's developer API (but you will still need a Twitter account, and a developer account is highly encouraged for large-scale data collection).

Currently, the package is not (yet) optimized for Twitter's API v2 (and thus the academic research track).

```
install.packages("rtweet")
```

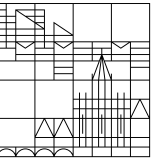


# Example: rtweet

Let's download the latest 1000 tweets containing #impfung.

```
library(rtweet)
vac_tweets <- search_tweets("#impfung", n = 1000, include_rts = FALSE)
vac_tweets
```

```
## # A tibble: 1,000 x 90
##   user_id    status_id  created_at          screen_name  text          source
##   <chr>      <chr>      <dtm>           <chr>       <chr>        <chr>
## 1 13518414~ 1390228032~ 2021-05-06 08:52:46 Rudi_4711    "@SZ #Impfung~ Twitte~
## 2 73765817 1390227820~ 2021-05-06 08:51:56 KarlheinzIl~ "Wollte mich ~ Twitte~
## 3 12362187~ 1390227682~ 2021-05-06 08:51:23 domiwi194    "Super Immuns~ Twitte~
## 4 16859954~ 1390227377~ 2021-05-06 08:50:10 black_purpl~ "Als generati~ Twitte~
## 5 16859954~ 1390020034~ 2021-05-05 19:06:16 black_purpl~ "\"Allerdings~ Twitte~
## 6 13570926~ 1390227085~ 2021-05-06 08:49:00 lujustsays   "Und im übrig~ Twitte~
## 7 13570926~ 1389951298~ 2021-05-05 14:33:08 lujustsays   "Vielleicht h~ Twitte~
## 8 19710089 1390226501~ 2021-05-06 08:46:41 mattimerker  "\U0001f44d<U+2935><U+FE0F> ~ Twitte~
## 9 19710089 1389998448~ 2021-05-05 17:40:29 mattimerker  "Das Impftemp~ Twitte~
## 10 19710089 1390019166~ 2021-05-05 19:02:49 mattimerker  "Impfquote (2~ Twitte~
## # ... with 990 more rows, and 84 more variables: display_text_width <dbl>,
## #   reply_to_status_id <chr>, reply_to_user_id <chr>,
## #   reply_to_screen_name <chr>, is_quote <lgl>, is_retweet <lgl>,
```

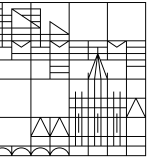


# Example: rtweet

---

**Exercise 3: rtweet:** Try to obtain both the latest 500 tweets posted by Annalena Baerbock, Armin Laschet & Olaf Scholz, and the 500 latest tweets favorited by them. Consult the documentation for help on the necessary functions: <https://github.com/ropensci/rtweet>





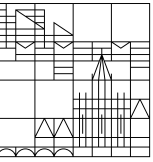
# Wrappers for Twitter's Academic API

---

Twitter's new API v2 offers an [Academic Research track](#) free for non-commercial academic research, including master's students. It includes access to the full Twitter archive ('historic data') and offers high rate and tweet limits (up to 10,000,000 tweets per month).

As the academic track is still new, there is not one definitive wrapper package (and most are still in active development). Choose your fighter:

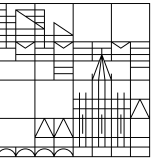
- `academictwitterR`
- `RTwitterV2`
- `twitterAcademic`
- `twitteRacademic` (note the different capital letter)



# Facepager

---

# Facepager



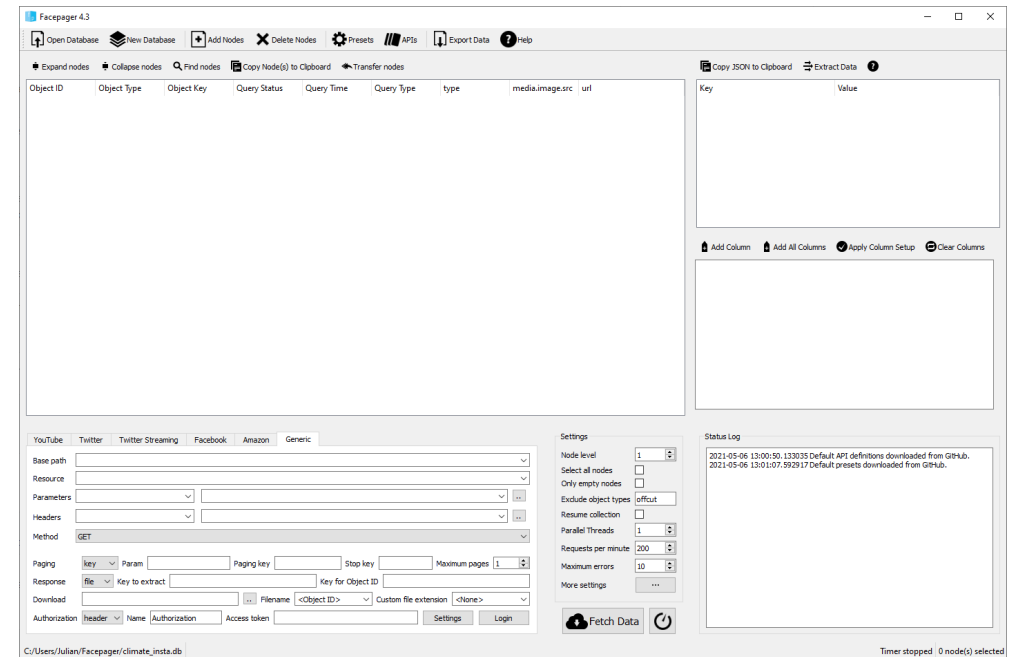
Facepager is a tool for automated data collection (APIs, webscraping) of publicly available data.

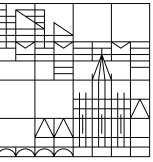
Main advantages:

- Free and open source
- Easy to use
- Good [documentation](#) and [tutorial videos](#)
- Several presets for common use cases
- App-level access to Facebook Graph API

Drawbacks:

- some outdated information on the Wiki
- Potential bottleneck of app-level rate limits





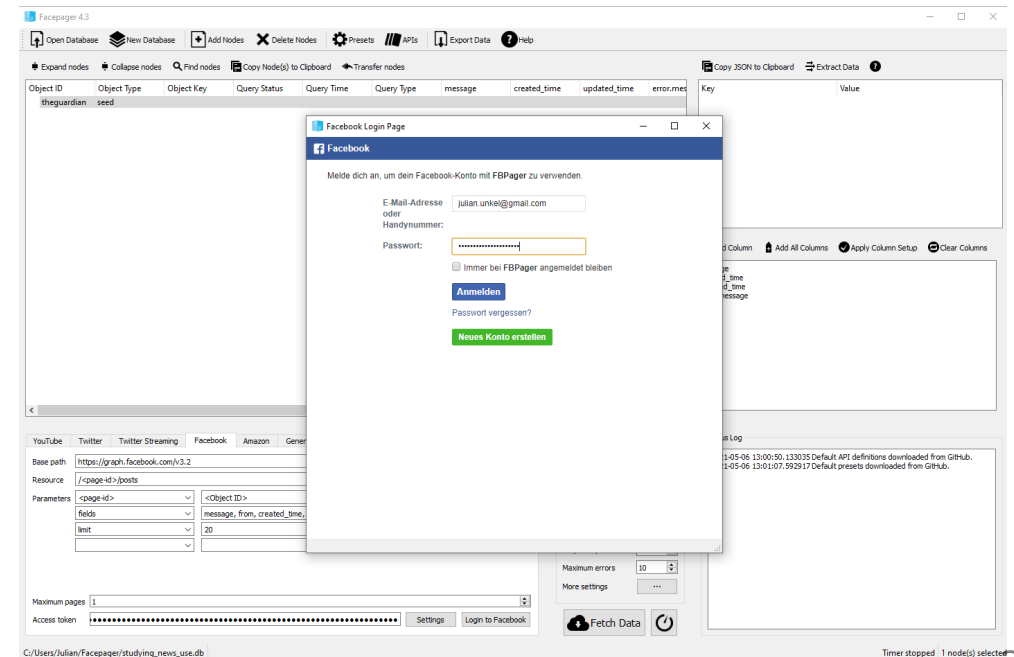
# Fetching Facebook data with Facepager

Apart from providing a point-and-click solution to API calls, the main advantage is the app-level access to Facebook's Graph API. Thus, it is possible to obtain data from public Facebook pages (including comments!) without an own Developer API authorization (however, you still need a Facebook account).

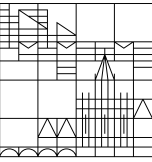
There are also presets for the most common tasks (fetching page data, fetching posts from pages, fetching comments from posts).

Getting started - fetching posts and comments:

1. [Download and install Facepager](#)
2. Create new local database
3. Login to Facebook via Facepager
4. Add Facebook pages as nodes (id or name)
5. Fetch posts for these pages using preset "2 Get Facebook posts"
6. Switch node level to 2 and fetch comments using preset "3 Get comments".
7. Data can be exported as a CSV file for further analysis.



# Fetching Facebook data with Facepager



The screenshot displays the Facepager 4.3 application interface. The main window shows a table of fetched data with columns for Object ID, Object Type, Object Key, Query Status, Query Time, Query Type, message, created\_time, like\_count, and comment\_count. A specific row is highlighted, showing a post with the message "Brexit. The gift that keeps on giving." and 20 likes.

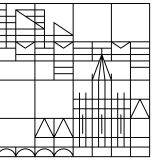
Object ID	Object Type	Object Key	Query Status	Query Time	Query Type	message	created_time	like_count	comment_count
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Brexit. The gift t...	2021-05-06T10:...	20	5
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Doris Joh'Nelso...	2021-05-06T10:...	1	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	WW3	2021-05-06T10:...	0	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Good movie	2021-05-06T10:...	0	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Hope you are n...	2021-05-06T10:...	0	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	We had a refere...	2021-05-06T10:...	14	9
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Cod Wars II... N...	2021-05-06T10:...	5	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Jennifer Juniper...	2021-05-06T10:...	4	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Tories to fight l...	2021-05-06T10:...	5	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Bobby Murphy ...	2021-05-06T10:...	1	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	On election day...	2021-05-06T10:...	4	1
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Jersey crisis esc...	2021-05-06T10:...	2	1
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Perfidious Albio...	2021-05-06T10:...	2	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Humans are kill...	2021-05-06T10:...	2	2
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Jennifer Juniper...	2021-05-06T10:...	8	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Fire!	2021-05-06T10:...	0	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Of course we v...	2021-05-06T10:...	2	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Brexit is going s...	2021-05-06T10:...	5	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Oswald A L Alm...	2021-05-06T10:...	0	0
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Do you remem...	2021-05-06T10:...	0	1
1016...	data	data.*	fetches (200)	2021-05-06 13:1...	Facebook/<po...	Chris Barrett LO...	2021-05-06T10:...	0	0

The interface also includes a settings panel at the bottom with the following configuration:

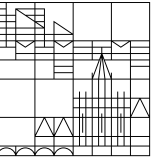
- Base path: `https://graph.facebook.com/v3.2`
- Resource: `/<post-id>/comments`
- Parameters: `<post-id>`
- Fields: `message, created_time, parent, comment_count, like_count`
- Node level: 3
- Exclude object types: `offcut`

A status log at the bottom right shows the execution progress of the data fetching process.





# Social monitoring services



# Social monitoring services

---

Social monitoring services are (commercial) services for, ahem, monitoring social media, for example:

- [Crowdtangle](#)
- [Synthesio](#)
- [BuzzSumo](#)

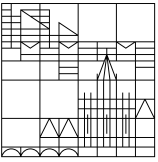
Advantages:

- Easy to use, dashboards
- App-level API access
- Monitor multiple social media at once

Drawbacks:

- Commercial and often costly
- Not primarily made for research
- Intransparent (coverage?)

# CrowdTangle



Crowdtangle is a social monitoring service owned by Facebook.

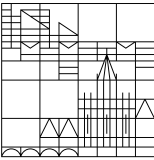
- Access to both public Facebook & Instagram data (but no comments)
- Free **academic track** (but currently only PhD students+ and subject to application)
- **Own API** for programmatic access
- Time-series data on posts
- Some preprocessing included (e.g., image text recognition for Instagram posts)
- Lisa or Julian can provide data ;)

The screenshot displays the CrowdTangle interface for a specific link. At the top, a search bar shows the URL: <https://www.nytimes.com/2021/05/05/us/politics/biden-carters-photo.html>. Below this, there are three main sections: 'LINK PREVIEW', 'PUBLIC REFERRALS WE'VE SEEN', and 'FACEBOOK ACTIVITY'. The 'LINK PREVIEW' shows a photo of Joe Biden and Jill Biden with their grandchildren, titled 'Wide-Angle Oddity: Giant Bidens Meet Tiny Carters' from NYTIMES.COM, dated MAY 5, 2021. The 'PUBLIC REFERRALS WE'VE SEEN' section shows a bar chart for 'Total Interactions' with 4,543 total interactions, broken down by platform: Facebook (4.5K), Instagram (0), and Reddit (9). The 'FACEBOOK ACTIVITY' section shows 'Facebook Interactions' with a total of 6,053, broken down by type: 4,325 likes, 847 comments, and 881 shares.

Below these sections, there is a tabbed interface for social media platforms: Facebook (selected), Instagram, Reddit, and Twitter. The Facebook tab is active, showing a table of 'WHO SHARED THIS LINK?'. The table is sorted by 'Total Interactions' and has the following data:

WHO SHARED THIS LINK?	MESSAGE	DATE	INTERACTIONS
<b>The New York Times</b> 17,675,594 Page Likes	How did this photograph seemingly turn the Bidens into giants and shrink the Carters? We explain.	MAY 5, 2021	3,579
<b>Mocha Party</b> 2,920 Members	A picture of two presidents and first ladies led to many Hobbit jokes online. We explain what happened.	MAY 6, 2021	4
<b>WheresSpot</b> 10,109 Members	cc photographers and DPs	MAY 6, 2021	3
<b>Olympia Camera Club Facebook G...</b> 765 Members	It isn't a good idea to use a wide-angle lens to photograph people.	MAY 6, 2021	3
<b>Azi Paybarah</b> 882 Followers	Good explainer on why this photo looks like this, via Heather Tal Murphy	MAY 5, 2021	1
<b>America News</b> 842 Page Likes	Wide-Angle Oddity: Giant Bidens Meet Tiny Carters	MAY 6, 2021	0

# CrowdTangle



Example Facebook dashboard:

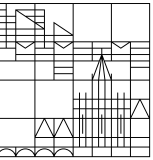
The screenshot shows the CrowdTangle interface for a Facebook dashboard. The top navigation bar includes 'DASHBOARDS', 'LIVE DISPLAYS', 'INTELLIGENCE', and 'SEARCH'. The left sidebar contains 'Notifications', 'Explore', 'Lists', '+ Create List', 'MY FAVORITES', 'PAGES', 'Saved Searches', 'Saved Posts', and 'Weights'. The main content area is titled 'All Page Lists' and shows a search bar. Below the search bar, there are tabs for 'Posts', 'Leaderboard', and 'Manage'. The 'Posts' tab is active, showing a list of posts. The first post is from 'ZEIT ONLINE' with the headline 'Bundestag beschließt Lockerungen für Geimpfte'. The second post is from 'WELT Nachrichtensender' with the headline 'Corona: Bundestag beschließt Lockerungen für Geimpfte und Genesene'.

Example Instagram dashboard:

The screenshot shows the CrowdTangle interface for an Instagram dashboard. The top navigation bar is the same as the Facebook dashboard. The left sidebar is the same as the Facebook dashboard. The main content area is titled 'Politiker' and shows a search bar. Below the search bar, there are tabs for 'Posts', 'Leaderboard', 'Notifications', and 'Manage'. The 'Posts' tab is active, showing a list of posts. The first post is from 'markus.soeder' with the headline 'Dr. Markus Söder - 1 hour ago'. The post features a photo of Markus Söder wearing a face mask and holding a document. Below the photo, there is a caption in German: 'Der @bayerischer.landtag hat heute die Bayerische Verfassungsmedaille in Gold verliehen als Anerkennung für Verdienste um die Verfassung des Freistaats Bayern. Es war immer eine große Ehre, an Demokratie, Gerechtigkeit und Freiheit in Bayern mitzuwirken. Das bestärkt im Einsatz für unser schönes Land und seine Menschen.'

# CrowdTangle

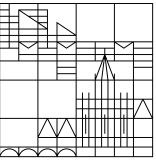
---



Sample data:

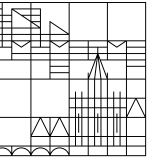
```
## # A tibble: 271 x 40
##   `Page Name`     `User Name`     `Facebook Id` `Page Category` `Page Admin Top ~
##   <chr>           <chr>           <dbl> <chr>           <chr>
## 1 ZEIT ONLINE     zeitonline      37816894428 NEWS_SITE        DE
## 2 WELT Nachric~  weltnachrich~  95242447553 BROADCASTING_MED~ DE
## 3 ZEIT ONLINE     zeitonline      37816894428 NEWS_SITE        DE
## 4 t-online        tonline         24897707939 MEDIA_NEWS_COMPA~ DE
## 5 ZEIT ONLINE     zeitonline      37816894428 NEWS_SITE        DE
## 6 WELT            welt            97515118114 NEWS_SITE        DE
## 7 ntv Nachricht~ ntvNachricht~  126049165307 TV_CHANNEL       DE
## 8 DER SPIEGEL     derspiegel      38246844868 NEWS_SITE        DE
## 9 FOCUS Online    focus.de        37124189409 NEWS_SITE        DE
## 10 Bild           bild            25604775729 NEWS_SITE        DE
## # ... with 261 more rows, and 35 more variables: Page Description <chr>,
## #   Page Created <chr>, Likes at Posting <dbl>, Followers at Posting <dbl>,
## #   Post Created <chr>, Post Created Date <date>, Post Created Time <time>,
## #   Type <chr>, Total Interactions <dbl>, Likes <dbl>, Comments <dbl>,
## #   Shares <dbl>, Love <dbl>, Wow <dbl>, Haha <dbl>, Sad <dbl>, Angry <dbl>,
## #   Care <dbl>, Video Share Status <chr>, Is Video Owner? <chr>,
## #   Post Views <dbl>, Total Views <dbl>, Total Views For All Crossposts <dbl>,
```

# CrowdTangle



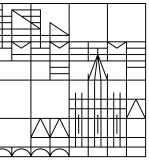
Sample post time-series data:

```
## # A tibble: 36 x 35
##       ID `Score Date (GMT)` Timestep Likes `Average Likes` Comments
##       <dbl> <dtm>           <dbl> <dbl>         <dbl>      <dbl>
## 1 1.02e16 2021-05-05 14:58:33     1     20           7        115
## 2 1.02e16 2021-05-05 15:15:54     2     36           7        200
## 3 1.02e16 2021-05-05 15:32:51     3     48           9        274
## 4 1.02e16 2021-05-05 15:50:52     4     59           9        346
## 5 1.02e16 2021-05-05 16:08:39     5     67           9        405
## 6 1.02e16 2021-05-05 16:25:32     6     79          11        486
## 7 1.02e16 2021-05-05 17:02:41     7     93          13        611
## 8 1.02e16 2021-05-05 17:40:12     8    110          14        726
## 9 1.02e16 2021-05-05 17:57:41     9    117          14        758
## 10 1.02e16 2021-05-05 18:34:05    10    131          15        839
## # ... with 26 more rows, and 29 more variables: Average Comments <dbl>,
## #   Shares <dbl>, Avg Shares <dbl>, Loves <dbl>, Avg Loves <dbl>, Wows <dbl>,
## #   Avg Wows <dbl>, Hahas <dbl>, Avg Hahas <dbl>, Sads <dbl>, Avg Sads <dbl>,
## #   Angrys <dbl>, Avg Angrys <dbl>, Cares <dbl>, Avg Cares <dbl>,
## #   Reactions <dbl>, Avg Reactions <dbl>, Post Views <dbl>,
## #   Avg Post Views <dbl>, Total Views <dbl>, Avg Total Views <dbl>,
## #   Total Views for all Crossposts <dbl>,
```



# **Grey area tools**

---



# Grey area tools

---

As Bruns (2019) noted, "break[ing] the rules" (p. 16) is one way to deal with an increasingly restrictive API landscape. For most social media platforms, there are several 'unofficial' tools like [TikTok-API](#) or [Instaloader](#) to access (public) data.

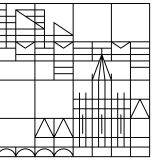
These tools often make use of:

- Browser emulation
- Web scraping
- Private APIs

They are often the only viable way for automated data fetching from these platforms. They are also likely violating the platforms' ToS (but [German law may be on your side](#)) and are subject to cease working at a moment's notice.

Use them if you want and need to, but always have a backup plan available.



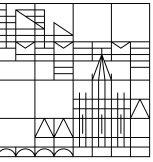


# Exercise solutions

---

# Exercise solutions

---

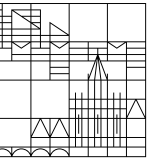


## Exercise 1:

```
ex1_resp <- GET(url = ps_base,  
               path = ps_submission,  
               query = list(subreddit = "de",  
                            sort = "asc",  
                            size = 50)  
               )
```

# Exercise solutions

---



## Exercise 2:

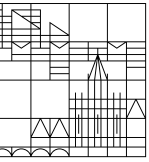
```
# Get first 100 comments
ex2_resp_1 <- GET(url = ps_base,
                 path = ps_comment,
                 query = list(q = "lol",
                             subreddit = "politics",
                             size = 100))

ex2_data_1 <- content(ex2_resp_1)$data

# Extract timestamp of last result
last_comment_timestamp <- tail(ex2_data_1, 1)[[1]]$created_utc

ex2_resp_2 <- GET(url = ps_base,
                 path = ps_comment,
                 query = list(q = "lol",
                             subreddit = "politics",
                             size = 100,
                             before = last_comment_timestamp))
```

(Note that to make sure we do not miss any comments posted at the same time, we could add +1 to the `last_comment_timestamp` and then filter out eventual duplicates.)



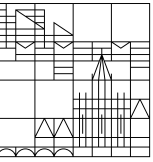
# Exercise solutions

---

## Exercise 3:

```
candidates <- c("ABaerbock", "ArminLaschet", "OlafScholz")
timelines <- get_timelines(candidates, n = 500)
favs <- get_favorites(candidates, n = 500)
```

(Note that the favs tibble contains one additional variable, indicating `favorited_by`)



# Thanks

---

## Credits:

- Slides created with [xaringan](#)
- Title image by [Tracy Le Blank / Pexels](#)
- Icons by [Bootstrap](#)
- Coding cat gif by [Memecandy/Giphy](#)