WZB
Wissenschaftszentrum Berlin
für Sozialforschung

R Tutorial at the WZB

11 – Collecting data from the web

Markus Konrad

January 24, 2019

Today's schedule

1. Review of last week's tasks
2. Tapping APIs
3. Use case: Twitter API
4. Use case: Geocoding with Google Maps API
5. Web scraping

Review of last week's tasks

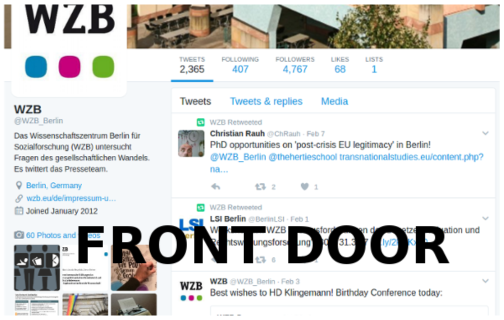
Solution for tasks #10

now online on https://wzbsocialsciencecenter.github.io/wzb_r_tutorial/

Tapping APIs

What is an API?

- stands for Application Programming Interface
- defined interface for communication between software components
- Web API: provides an interface to structured data from a web service

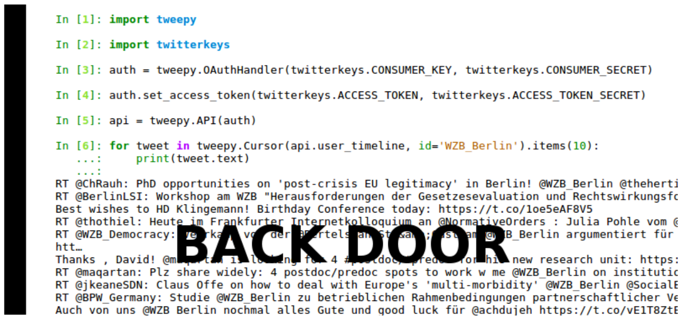


WZB
@WZB_Berlin
Das Wissenschaftszentrum Berlin für Sozialforschung (WZB) untersucht Fragen des gesellschaftlichen Wandels. Es twittert das Presseteam.
Berlin, Germany
wzb.eu/de/impressum-u…
Joined January 2012
60 Photos and videos

WZB Retweeted
Christian Rauh @ChRauh · Feb 7
PhD opportunities on 'post-crisis EU legitimacy' in Berlin! @WZB_Berlin @thehertieschool transnationalstudies.eu/content.php?na...
2 1

WZB Retweeted
LSI Berlin @BerlinLSI · Feb 1
W[illegible] WZB [illegible] ation und Re[illegible]ntsw[illegible]ungsfors[illegible]ung [illegible]
6

WZB @WZB_Berlin · Feb 3
Best wishes to HD Klingemann! Birthday Conference today:



```
In [1]: import tweepy

In [2]: import twitterkeys

In [3]: auth = tweepy.OAuthHandler(twitterkeys.CONSUMER_KEY, twitterkeys.CONSUMER_SECRET)

In [4]: auth.set_access_token(twitterkeys.ACCESS_TOKEN, twitterkeys.ACCESS_TOKEN_SECRET)

In [5]: api = tweepy.API(auth)

In [6]: for tweet in tweepy.Cursor(api.user_timeline, id='WZB_Berlin').items(10):
   ...:     print(tweet.text)
   ...:
RT @ChRauh: PhD opportunities on 'post-crisis EU legitimacy' in Berlin! @WZB_Berlin @theherti
RT @BerlinLSI: Workshop am WZB "Herausforderungen der Gesetzesevaluation und Rechtswirkungsfc
Best wishes to HD Klingemann! Birthday Conference today: https://t.co/1oe5eAF8V5
RT @thothiel: Heute im Frankfurter Internetkolloquium an @NormativeOrders : Julia Pohle vom @
RT @WZB_Democracy: [illegible] @WZB_Berlin argumentiert für
htt…
Thanks , David! @m[illegible] new research unit: https:
RT @maqartan: Plz share widely: 4 postdoc/predoc spots to work w me @WZB_Berlin on institutic
RT @jkeaneSDN: Claus Offe on how to deal with Europe's 'multi-morbidity' @WZB_Berlin @SocialE
RT @BPW_Germany: Studie @WZB_Berlin zu betrieblichen Rahmenbedingungen partnerschaftlicher Ve
Auch von uns @WZB Berlin nochmal alles Gute und good luck für @achdujeh https://t.co/vE1T8ZtE
```

When should I use an API?

Whenever you need to **collect mass data** in the web in an automated manner.

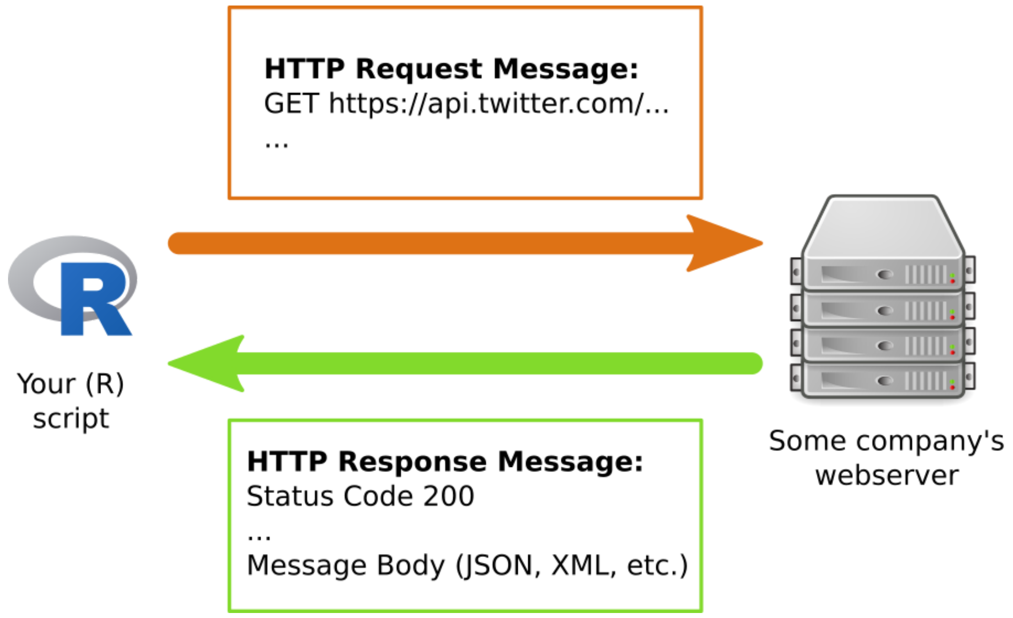
Whenever you need to **enrich or transform your existing data** with the help of a web service (automatted translation, geocoding, …)

Whenever you want to **run (semi-)automated experiments** in the web (MTurk, Twitter bots, eBay, etc.).

It should definitely be preferred over web scraping. (We'll later see why.)

How does a web API work?

Web APIs usually employ a **client-server model**. The client – that is you. The server provides the API endpoints as URLs.


HTTP Request Message:
GET https://api.twitter.com/...
...
Your (R) script
Some company's webserver
HTTP Response Message:
Status Code 200
...
Message Body (JSON, XML, etc.)

WZB Wissenschaftszentrum Berlin für Sozialforschung

How does a web API work?

Communication is done with request and response messages over Hypertext transfer protocol (HTTP).

Each HTTP message contains a header (message meta data) and a message body (the actual content). The three-digit HTTP status code plays an important role:

- 2xx: Success
- 4xx: Client error (incl. the popular 404: Not found or 403: Forbidden)
- 5xx: Server error

The message body contains the requested data in a specific format, often JSON or XML.

WZB Wissenschaftszentrum Berlin für Sozialforschung

Let's query an API

We can directly query the Twitter search API endpoint.

In your browser: https://api.twitter.com/1.1/search/tweets.json?q=hello

In R:

```
library(RCurl)

# add argument verbose = TRUE for full details
getURL('https://api.twitter.com/1.1/search/tweets.json?q=hello')
```

```
## [1] "{\"errors\":[{\"code\":215,\"message\":\"Bad Authentication data.\"}
```

→ we get an error because we're not authenticated (we'll do that later).

WZB Wissenschaftszentrum Berlin für Sozialforschung

10/59

Reading JSON

APIs often return data in JSON format, which is a nested data format that allows to store key-value pairs and ordered lists of values:

```
{
  "profiles": [
    {
      "name": "Alice",
      "age": 52
    },
    {
      "name": "Bob",
      "age": 35
    }
  ]
}
```

- example from abgeordnetenwatch API: https://www.abgeordnetenwatch.de/api/parliament/bundestag/profile/a merkel/profile.json
- to process with R: jsonlite package
- can be viewed in the browser, but requires a browser extension like JSON Formatter to make it actually readable

WZB Wissenschaftszentrum Berlin für Sozialforschung

Examples of popular APIs

Social media:

- Twitter
- Facebook Graph API (restricted to own account and public pages)
- YouTube (Google)
- LinkedIn

Google (see API explorer):

- Cloud (Translation / NLP / Speech / Vision / …)
- Maps (now also part of cloud platform) (geocoding, directions, places, etc.)
- Civic Information (political representation, voting locations, election results, …)
- Books

Other:

Microsoft Face API, Amazon Mechanical Turk API, Wikipedia, etc.

For more, see programmableweb.com.

WZB Wissenschaftszentrum Berlin für Sozialforschung

What's an "API wrapper package"?

Working with a web API involves:

- constructing request messages
- parsing result messages
- handling errors

→ much implementation effort necessary.

For popular web services there are already "API wrapper packages" on CRAN:

- implement communication with the server
- provide direct access to the data via R functions
- examples: rtweet, ggmap (geocoding via Google Maps), wikipediR, genderizeR

Access to web APIs

Most web services require you to set up a user account on their platform.

Many web services provide a free subscription to their services with **limited access** (number of requests and/or results is limitted) and a paid subscription as "premium access" or as usage-dependent payment model. Some services (like Google Cloud Platform) require you to register with credit card information and grant a monthly free credit (e.g. $300 for Translation API at the moment).

In both cases you're required to authenticate with the service when you use it (→ **API key** or **authorization token**).

A few warning signs

Always be aware that you're using a web service, i.e. **you're sending (parts of) your data to some company's server.**

Using a web API is a complex and often long running task. Be aware that many things can go wrong, e.g.:

- the server delivers garbage
- the server crashes
- your internet connection is lost
- your computer crashes
- your script produces an endless loop

→ **never** blindly trust what you get
→ **always** do validity checks on the results (check NAs, ranges, duplicates, etc.)
→ use defensive programming (e.g. save intermediate results to disk; implement wait & retry mechanisms on failures; etc.)

Use case: Twitter API

Which APIs does Twitter provide?

Twitter provides several APIs. They are documented at https://developer.twitter.com/en/docs

The most important APIs for us are the **"Search API" (aka REST API)** and **"Realtime API" (aka Streaming API)**.

WZB Wissenschaftszentrum Berlin für Sozialforschung

Free vs. paid

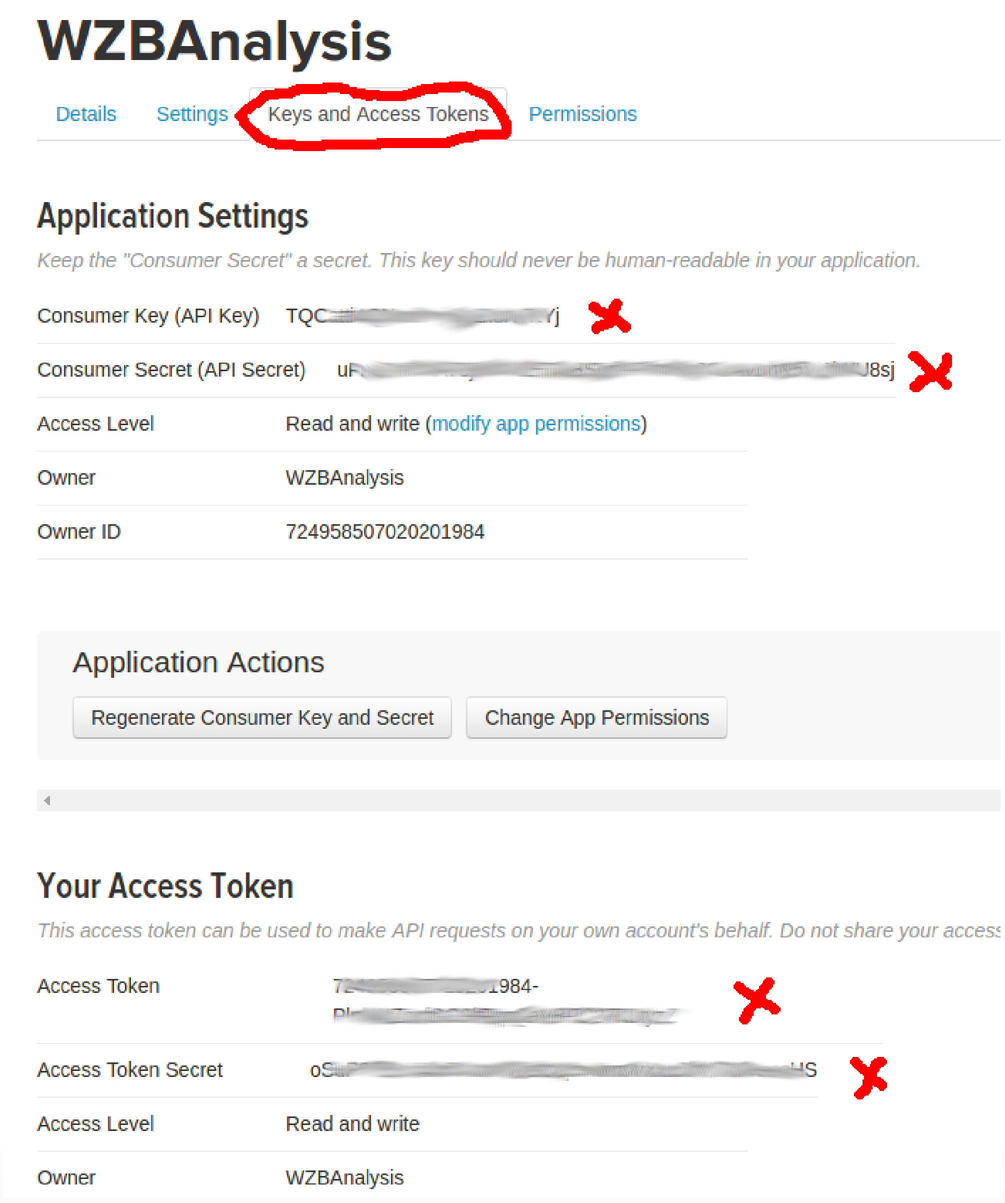
Twitter provides three subscription levels:

- Standard (free)
  - search historical data for up to 7 days
  - get sampled live tweets
- Premium ($150 to $2500 / month)
  - search historical data for up to 30 days
  - get full historical data per user
- Enterprise (special contract with Twitter)
  - full live tweets

The rate limiting also differs per subscription level (number of requests per month).

What else do I need?

1. A Twitter account
2. Authentication data for a "Twitter app"
   - create your Twitter app on https://apps.twitter.com/
   - retrieve four authentication keys:



WZBAnalysis

Details Settings Keys and Access Tokens Permissions

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	TQC[redacted]Yj
Consumer Secret (API Secret)	uF[redacted]J8sj
Access Level	Read and write (modify app permissions)
Owner	WZBAnalysis
Owner ID	724958507020201984

Application Actions

Regenerate Consumer Key and Secret | Change App Permissions

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access

Access Token	72[redacted]1984- Pl[redacted]
Access Token Secret	oS[redacted]HS
Access Level	Read and write
Owner	WZBAnalysis

WZB Wissenschaftszentrum Berlin für Sozialforschung

What else do I need?

Keep your authentication keys safe!

Do not publish them anywhere!

Wissenschaftszentrum Berlin für Sozialforschung

R Packages for the Twitter API

Several "API wrapper" packages for Twitter exist on CRAN:

- twitteR: Search API only
- streamR: Streaming API only
- rtweet: both APIs

I'll use rtweet on the following slides.

Creating an authentication token

You need to construct an authentication token and provide the keys from the "Twitter Apps" page:

```
library(rtweet)

token <- create_token(
    app = "WZBAnalysis",
    consumer_key = "...",
    consumer_secret = "...",
    access_token = "...",
    access_secret = "...")
```

Searching tweets

Sample of 10 tweets (excluding retweets) from the last 7 days containing "#wzb":

```
tw_search_wzb <- search_tweets('#wzb', n = 10, include_rts = FALSE)
# display only 3 out of 88 variables:
tw_search_wzb[c('screen_name', 'created_at', 'text')]
```

```
## # A tibble: 6 x 3
##   screen_name   created_at          text
##   <chr>         <dttm>              <chr>
## 1 Kevin_Bernal… 2019-01-16 00:46:28 Without even knowing, Wonder is teach
## 2 WZB_Berlin    2019-01-15 14:43:44 #Brexit als Symptom und die Wut gegen
## 3 WZB_Berlin    2019-01-14 15:06:54 #Digitalisierung spaltet: Geringquali
## 4 WZB_Berlin    2019-01-14 08:59:08 „Unser Bild von der #Universität ist
## 5 Daver_eiss    2019-01-15 09:28:18 Und dass sie stattfindet, beweist so
## 6 WZB_GlobCon   2019-01-13 14:00:00 "Ende des letzten Jahres sind die neu
```

WZB Wissenschaftszentrum Berlin für Sozialforschung

Retrieve timelines for selected users

Retrieve 10 latest tweets from timelines of selected users:

```
tw_timelines <- get_timelines(c("WZB_Berlin", "JWI_Berlin", "DIW_Berlin"), n

tw_timelines %>%   # "favorite_count" is number of likes:
  select(screen_name, favorite_count, retweet_count, text) %>%
  group_by(screen_name) %>%
  arrange(screen_name, desc(favorite_count)) %>%
  top_n(3)
```

```
## # A tibble: 9 x 4
## # Groups:   screen_name [3]
##   screen_name favorite_count retweet_count text
##   <chr>                <int>         <int> <chr>
## 1 DIW_Berlin               6             4 „Die #Geschlechterquote für Au
## 2 DIW_Berlin               3             2 "„Österreichs Weg zu einer kli
## 3 DIW_Berlin               2             2 “Von den 200 umsatzstärksten U
## 4 JWI_Berlin              13             4 "Weizenbaum Insights: Berlin i
## 5 JWI_Berlin              10             3 We're excited that so many chi
## 6 JWI_Berlin               5             5 We're happy to invite you and
## 7 WZB_Berlin               5             4 "Und zur Vorbereitung auf die
## 8 WZB_Berlin               3             3 Mehr Beschäftigte, weniger Aus
## 9 WZB_Berlin               1             0 Und auf Deutsch hier in den WZ
```

WZB Wissenschaftszentrum Berlin für Sozialforschung

Sending a tweet

Posting a tweet to the timeline of your "app" account:

```
rand_nums <- round(runif(2, 0, 100))
# sprintf creates a character string by filling in placeholder
new_tweet <- sprintf('Hello world, it is %s and %d + %d is %d.
                     Sys.time(), rand_nums[1], rand_nums[2],
                     sum(rand_nums))
post_tweet(new_tweet)

## your tweet has been posted!
```

→ will be posted on twitter.com/WZBAnalysis

WZB Wissenschaftszentrum Berlin für Sozialforschung

Live streaming

Live streaming of tweets is especially practical when run during events of interest (elections, demonstrations, etc.). This is because Twitter only allows limited download of historical data (see "Free vs. paid" slide before). So always try to collect the data during an event!

Realtime retrieval of tweets from **sampled** live stream. By default, this will collect tweets for 30 seconds according to optional search criteria:

```
stream_ht2019 <- stream_tweets('#2019')

# Streaming tweets for 30 seconds...
# Finished streaming tweets!
```

→ results in data frame with 88 variables as with previous functions.

Live streaming

A practical way to collect tweets during events is to specify the recording length and let the tweets be written to a file:

```
stream_tweets(
  "oscars,academy,awards",
  timeout = 60 * 60 * 24 * 7,    # record tweets for 7 days (specified in se
  file_name = "awards.json",
  parse = FALSE
)
```

Make sure you have enough disk space and that the internet connection is stable!

After recording, load the data file as data frame:

```
awards <- parse_stream("awards.json")
```

For more functions, see the introductionary vignette to rtweet.

Use case: Geocoding with Google Maps API

What is geocoding?

Geocoding is the process of finding the geographic coordinates (longitude, latitude) for a specific query term (a full address, a postal code, a city name etc.).

Reverse geocoding tries to map a set of geographic coordinates to a place name / address.

Getting access

As of June 2018, the Maps API is part of Google's "Could Platform". This requires you to have:

1. A Google account (surprise!).
2. Start a Google Cloud Platform (GCP) free trial (valid for one year).
3. Register for billing (they want your credit card). They promise not to charge it after the end of the free trial…

Inside GCP, you can go to APIs & Services > Credentials to get your API key.

Using ggmap for geocoding

You need to install the package ggmap, at least of version 2.7.

```
library(ggmap)

# provide the Google Cloud API key here:
register_google(key = google_cloud_api_key)

places <- c('Berlin', 'Leipzig', '10317, Deutschland',
            'Reichpietschufer 50, 10785 Berlin')
place_coords <- geocode(places) %>% mutate(place = places)
place_coords
```

```
##        lon      lat                             place
## 1 13.40495 52.52001                            Berlin
## 2 12.37307 51.33970                           Leipzig
## 3 13.48475 52.49854                10317, Deutschland
## 4 13.36509 52.50640 Reichpietschufer 50, 10785 Berlin
```

WZB Wissenschaftszentrum Berlin für Sozialforschung

Reverse geocoding

Take the WZB's geo-coordinates and see if we can find the address to it:

```
# first longitude, then latitude
revgeocode(c(13.36509, 52.50640))
```

```
## [1] "Reichpietschufer 50, 10785 Berlin, Germany"
```

Tweets also sometimes come with geo-coordinates. With reverse geocoding it is possible to find out from which city a tweet was sent.

WZB Wissenschaftszentrum Berlin für Sozialforschung

Web scraping

What is web scraping?

Web scraping is the process of **extracting data from websites** for later retrieval or analysis.

→ usually done as automated process by a web crawler, web spider or bot:

1. Fetch website (i.e. download its content)
2. Extract the parts in which you're interested and store them
3. Optionally follow links to other website → start again with 1.

Google and other search engines do it all the time in a big scale – Google: "Our index is well over 100,000,000 gigabytes"

WZB Wissenschaftszentrum Berlin für Sozialforschung

What is web scraping?

The problem: this huge amount of data is **largely unstructured**.

Web scraping or web mining tries to **extract structured information** from this mess.

When should I use web scraping?

Web scraping should be your **last resort** if you can't get the data otherwise (we'll see why).

1. Check for access to structured data (APIs, databases, RSS feeds) or already implemented scrapers (packages on CRAN…)
2. Ask the website owner
3. Consider the implementation work when scraping multiple or very complex websites
   - check aggregator websites
   - consider manual data collection and/or sampling

WZB Wissenschaftszentrum Berlin für Sozialforschung

Legal issues

Web scraping might lead to (among others):

- Copyright infringement
- Violation of the Computer Fraud and Abuse Act (US)

Depends mainly on:

- what the website's **Terms and Conditions** say
- how scraped data is used
- how scraping is done

Some general rules

- check the Terms and Conditions and the robots.txt file
- do not publish the scraped data unless you have got the permission
- do not crush the website's servers with requests
- when unsure: ask the website owner and/or a lawyer

robots.txt

- specifies how a web crawler (e.g. Google "web spider") should behave when visiting the website
- respect it!
- located at domain.com**/robots.txt**
- excerpt from https://wzb.eu/robots.txt:

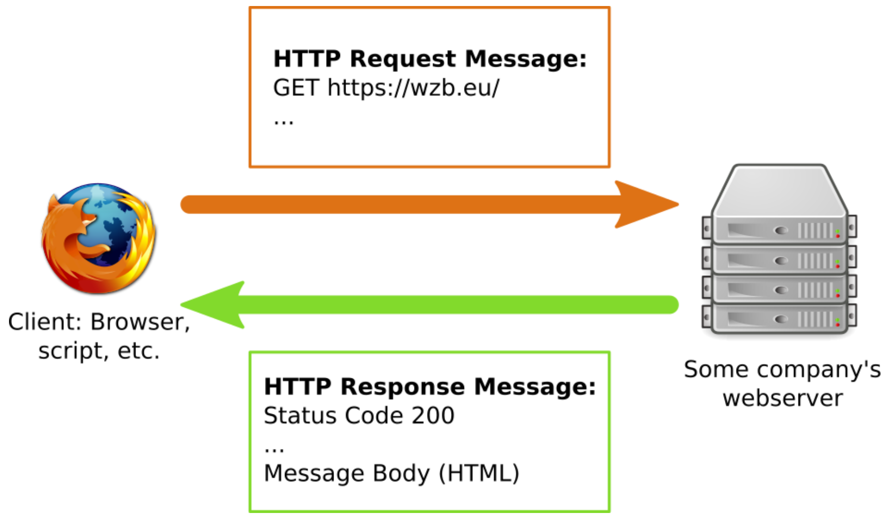
```
User-agent: *
Crawl-delay: 10
# Directories
Disallow: /includes/
...
```

- do not crawl anything under https://wzb.eu/includes/
- use a delay of 10 seconds between subsequent requests

WZB Wissenschaftszentrum Berlin für Sozialforschung

HTTP again

Again, we have a **client-server model**:



This is very similar to the Web API concept (we use the same HTTP protocol), only that the server delivers HTML content this time.

HTTP again

In R:

```
library(httr)
response <- GET('https://wzb.eu')
response
```

```
## Response [https://wzb.eu/de]
##   Date: 2019-01-23 11:53
##   Status: 200
##   Content-Type: text/html; charset=UTF-8
##   Size: 348 kB
## <!DOCTYPE html>
## <html  lang="de" dir="ltr" prefix="content: http://purl.org/rss/1.0/modu.
##   <head>
##     <meta charset="utf-8" />
## <meta name="title" content="WZB-Startseite | WZB" />
## <link rel="shortlink" href="https://wzb.eu/de" />
## <link rel="canonical" href="https://wzb.eu/de" />
## <link rel="shortcut icon" href="/favicon.ico" />
## <link rel="mask-icon" href="/safari-pinned-tab.svg" color="#000000" />
## <link rel="icon" sizes="16x16" href="/favicon-16x16.png" />
## ...
```

WZB Wissenschaftszentrum Berlin für Sozialforschung

41/59

Technical problem #1

Too many HTTP requests.

The webserver may notice when you send too many requests in a small amount of time.

It might be considered as an attack (DoS – Denial of Service attack) and your IP gets blocked for some time.

Solution: Use delays during requests (for example with `Sys.sleep()`)

→ You will need patience when you crawl many web pages!

A look at the HTML code

HTML (Hypertext Markup Language) describes the structure of a website, e.g.:

- Which part of the website represents the main content?
- What is the navigation menu?
- What is the headline of an article inside the main content area?

Represented as nested tags with attributes:

```
<body>
  <nav width="100%"> ... </nav>
  <article>
    <h1>Some headline</h1>
    <img src="some_image.png" alt="My image" />
  </article>
</body>
```

- tags are written as `<tag> ... </tag>`
- attributes of tags are written as `<tag attrib="value"> ... </tag>`

Technical problem #2

Two websites rarely have the same HTML structure.

Examples:

- https://www.diw.de/de
- https://www.leibniz-gemeinschaft.de/start/

Both websites have news but the HTML structure is completely different → specific data extraction instructions for both websites necessary

WZB
Wissenschaftszentrum Berlin
für Sozialforschung

Technical problem #2

Pages on a single website with the same "page type" usually share the same structure, e.g.:

- all Wikipedia articles have similar HTML structure
- all WZB news articles have similar HTML structure

It's very hard to do web scraping on a big range of different websites.

→ before you start a project assess the HTML code of the websites → try to find similarities

→ try to find aggregator websites, public databases or similar platforms that gather information from different websites

Technical problem #3

Websites can get very complex

- more and more "interactive" / "dynamic" content on websites featuring JavaScript
- data is loaded dynamically/asynchronously into websites

Example: Pages that load more articles when you scroll down (Facebook!)

→ what you see in your browser might not be what get when crawling the website!

Solutions: Automated web browsers, e.g. via Selenium → quite complex to implement

Technical problem #4

Websites change – They do relaunches or disappear.

- no guarantee that the website you scrape today will have the same HTML structure tomorrow
- problem when monitoring websites for a long time (i.e. longitudinal research)

Sometimes it is possible to recover websites from the Internet Archive.

Scraping abgeordnetenwatch.de

> On abgeordnetenwatch.de (which translates as “member of parliament watch”) users find a blog, petitions and short profiles of their representatives in the federal parliament as well as on the state and EU level.
>
> – source

Example: Research on Twitter networks among MPs → find Twitter name for each MP.

abgeordnetenwatch.de links to Twitter account on MP's profiles, see for example:

https://www.abgeordnetenwatch.de/profile/christian-lindner *

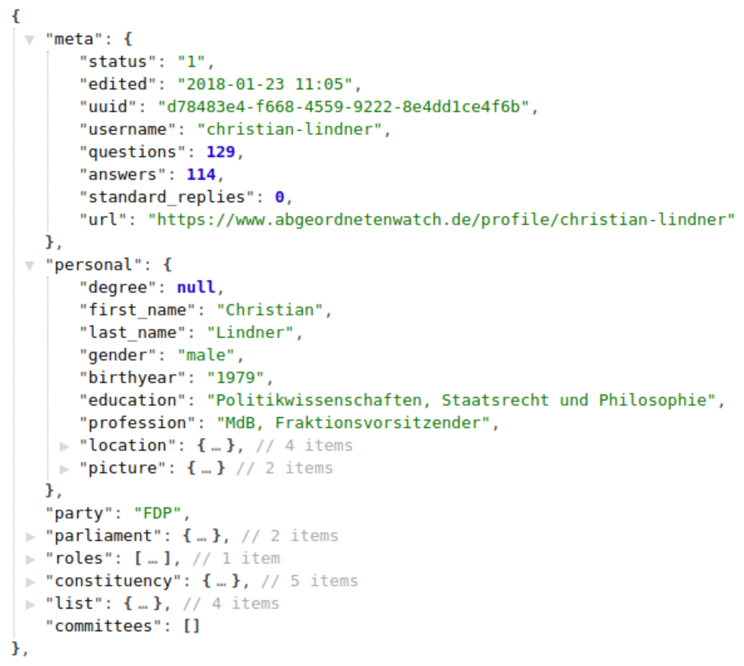
* Personal remark: I'm no CL fanboy, I was just sure that he has a Twitter account...

WZB Wissenschaftszentrum Berlin für Sozialforschung

Scraping abgeordnetenwatch.de

First: Check if we can avoid scraping!

→ they provide an API: https://www.abgeordnetenwatch.de/api; all profiles of MPs are at: https://www.abgeordnetenwatch.de/api/parliament/bundestag/deputies.j



```
{
▼ "meta": {
      "status": "1",
      "edited": "2018-01-23 11:05",
      "uuid": "d78483e4-f668-4559-9222-8e4dd1ce4f6b",
      "username": "christian-lindner",
      "questions": 129,
      "answers": 114,
      "standard_replies": 0,
      "url": "https://www.abgeordnetenwatch.de/profile/christian-lindner"
  },
▼ "personal": {
      "degree": null,
      "first_name": "Christian",
      "last_name": "Lindner",
      "gender": "male",
      "birthyear": "1979",
      "education": "Politikwissenschaften, Staatsrecht und Philosophie",
      "profession": "MdB, Fraktionsvorsitzender",
    ▶ "location": {…}, // 4 items
    ▶ "picture": {…} // 2 items
  },
  "party": "FDP",
▶ "parliament": {…}, // 2 items
▶ "roles": […], // 1 item
▶ "constituency": {…}, // 5 items
▶ "list": {…}, // 4 items
  "committees": []
},
```

Unfortunately, no link to Twitter in the data from the API!

We could ask the owners of the website if they want to provide the data, but let's use this website as illustrative example for web scraping.

Extracting specific data from HTML

For web scraping, we need to:

1. identify the elements of a website which contain our information of interest;
2. extract the information from these elements;

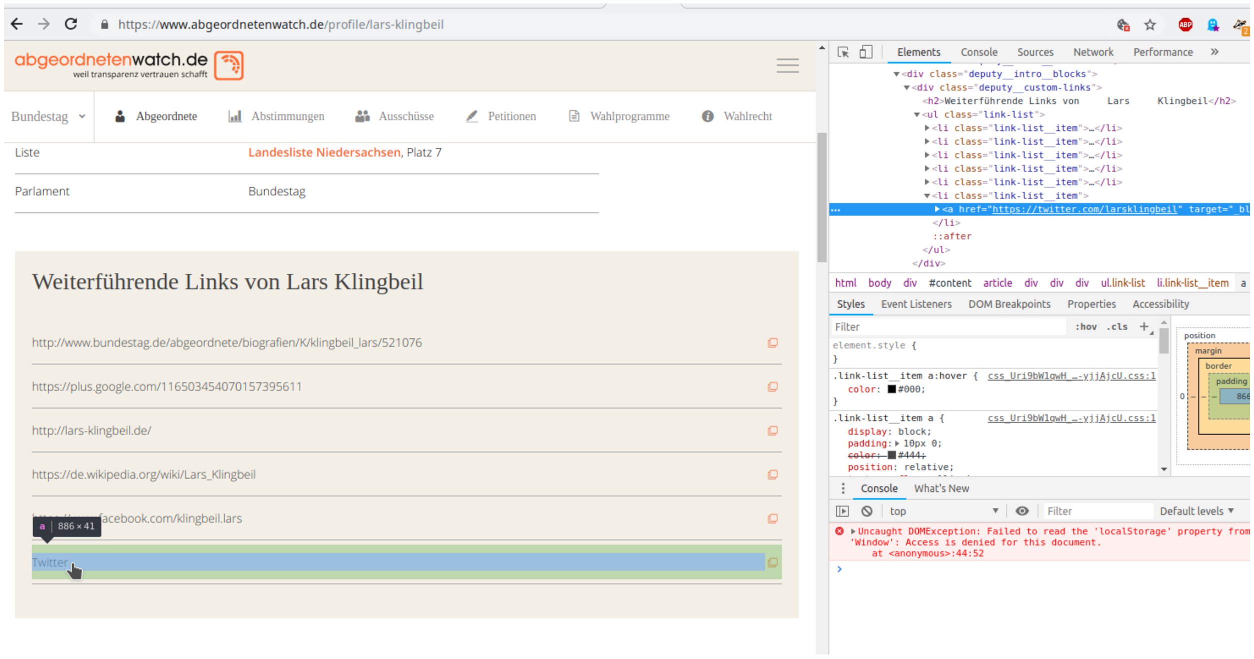
Both steps require some basic understanding of HTML and CSS. More advanced scraping techniques require an understanding of XPath and regular expressions.

We won't cover any of these here, but I will give you a short example trying to show the basics.

Inspecting the HTML source

A different profile, this time with many links (Facebook, Wikipedia, Twitter, etc): https://www.abgeordnetenwatch.de/profile/lars-klingbeil

In a browser, right click on the element of interest and select "Inspect". This opens a new pane on the right side which helps to navigate through the HTML tags and find a CSS selector for that element. This gives us a "path" to that element.



Liste	Landesliste Niedersachsen, Platz 7
Parlament	Bundestag

Weiterführende Links von Lars Klingbeil

http://www.bundestag.de/abgeordnete/biografien/K/klingbeil_lars/521076
https://plus.google.com/116503454070157395611
http://lars-klingbeil.de/
https://de.wikipedia.org/wiki/Lars_Klingbeil
facebook.com/klingbeil.lars
Twitter

```
▼<div class="deputy__intro__blocks">
  ▼<div class="deputy__custom-links">
     <h2>Weiterführende Links von      Lars      Klingbeil</h2>
   ▼<ul class="link-list">
     ▶<li class="link-list__item">…</li>
     ▶<li class="link-list__item">…</li>
     ▶<li class="link-list__item">…</li>
     ▶<li class="link-list__item">…</li>
     ▶<li class="link-list__item">…</li>
     ▼<li class="link-list__item">
...     ▶<a href="https://twitter.com/larsklingbeil" target="_bl
       </li>
       ::after
     </ul>
   </div>
```

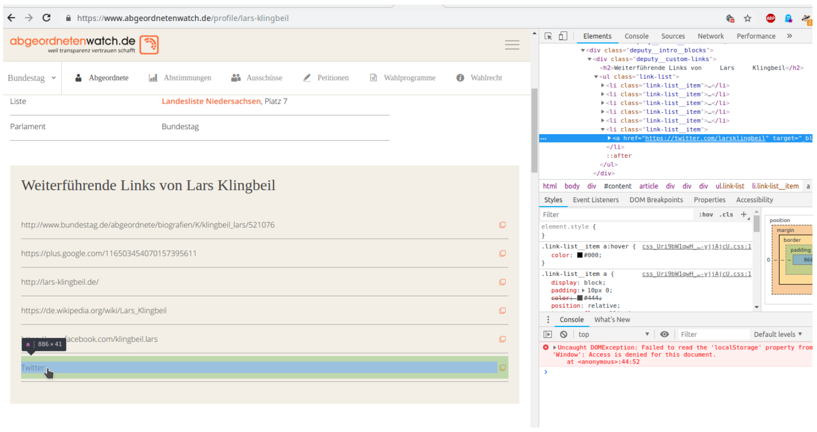
html body div #content article div div div ul.link-list li.link-list__item a

```
element.style {
}
.link-list__item a:hover {  css_Uri9bWlqwH_…-yjjAjcU.css:1
   color: ■#000;
}
.link-list__item a {        css_Uri9bWlqwH_…-yjjAjcU.css:1
   display: block;
   padding: ▶ 10px 0;
   ~~color: ■#444;~~
   position: relative;
```

```
▶Uncaught DOMException: Failed to read the 'localStorage' property from
 'Window': Access is denied for this document.
     at <anonymous>:44:52
```

WZB Wissenschaftszentrum Berlin für Sozialforschung

Extracting specific data from HTML



Liste	Landesliste Niedersachsen, Platz 7
Parlament	Bundestag

Weiterführende Links von Lars Klingbeil

http://www.bundestag.de/abgeordnete/biografien/K/klingbeil_lars/521076
https://plus.google.com/116503454070157395611
http://lars-klingbeil.de/
https://de.wikipedia.org/wiki/Lars_Klingbeil
facebook.com/klingbeil.lars
Twitter

```
▼<div class="deputy__intro__blocks">
  ▼<div class="deputy__custom-links">
     <h2>Weiterführende Links von     Lars     Klingbeil</h2>
    ▼<ul class="link-list">
      ▶<li class="link-list__item">…</li>
      ▶<li class="link-list__item">…</li>
      ▶<li class="link-list__item">…</li>
      ▶<li class="link-list__item">…</li>
      ▶<li class="link-list__item">…</li>
      ▼<li class="link-list__item">
...     ▶<a href="https://twitter.com/larsklingbeil" target="_bl
        </li>
        ::after
     </ul>
   </div>
```

```
html  body  div  #content  article  div  div  div  ul.link-list  li.link-list__item  a
```

```
element.style {
}
.link-list__item a:hover {  css_Uri9bW1qwH_…-yjjAjcU.css:1
  color: ■#000;
}
.link-list__item a {        css_Uri9bW1qwH_…-yjjAjcU.css:1
  display: block;
  padding: ▶ 10px 0;
  ~~color: ■#444;~~
  position: relative;
```

```
▶Uncaught DOMException: Failed to read the 'localStorage' property from
'Window': Access is denied for this document.
    at <anonymous>:44:52
```

The crucial information for the "path" to the elements of interest, which is the links specified by an `<a>...</a>` tag, is:

- `<div>` container with class attribute `"deputy__custom-links"`
- inside this, a list `<ul>` with class attribute `"link-list"`
- inside this, list elements `<li>`
- inside this, the links `<a>` that we want

WZB Wissenschaftszentrum Berlin für Sozialforschung

52/59

Extracting specific data from HTML

We can now use this information in R. The package `rvest` is made for parsing HTML and extracting content from specific elements. First, we download the HTML source and parse it via `read_html`:

```
library(rvest)

html <- read_html('https://www.abgeordnetenwatch.de/profile/lars-klingbeil')
html
```

```
## {xml_document}
## <html lang="de" dir="ltr">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=
## [2] <body class="html not-front not-logged-in no-sidebars page-user page
```

Extracting specific data from HTML

We apply the CSS selector (the "path" to the links) in order to extract only the specific link elements of the website:

```
links <- html_nodes(html, 'div.deputy__custom-links ul.link-list li a')
links
```

```
## {xml_nodeset (6)}
## [1] <a href="http://www.bundestag.de/abgeordnete/biografien/K/klingbeil_
## [2] <a href="https://plus.google.com/116503454070157395611" target="_bla
## [3] <a href="http://lars-klingbeil.de/" target="_blank">http://lars-klin
## [4] <a href="https://de.wikipedia.org/wiki/Lars_Klingbeil" target="_blan
## [5] <a href="https://www.facebook.com/klingbeil.lars" target="_blank">ht
## [6] <a href="https://twitter.com/larsklingbeil" target="_blank">Twitter</
```

WZB
Wissenschaftszentrum Berlin
für Sozialforschung

Extracting specific data from HTML

And finally, we extract only the value of each link's `href` attribute in order to get the actual URLs:

```
urls <- html_attr(links, 'href')
urls
```

```
## [1] "http://www.bundestag.de/abgeordnete/biografien/K/klingbeil_lars/5210
## [2] "https://plus.google.com/116503454070157395611"
## [3] "http://lars-klingbeil.de/"
## [4] "https://de.wikipedia.org/wiki/Lars_Klingbeil"
## [5] "https://www.facebook.com/klingbeil.lars"
## [6] "https://twitter.com/larsklingbeil"
```

Extracting specific data from HTML

In order to select only the link to Twitter and extract the Twitter name from there, we can apply a regular expression. Note that this is a quiet advanced topic. The gist is that you can create character string patterns and extract specified key information if this pattern matches:

```
# a pattern that matches:
#   http://twitter.com/user
#   https://twitter.com/user
#   http://www.twitter.com/user
#   https://www.twitter.com/user
# and extracts the "user" part
matches <- regexec('^https?://(www\\.)?twitter\\.com/([A-Za-z0-9_-]+)/?', ur
twitter_name <- sapply(regmatches(urls, matches),  # the "user" part is numb
                       function(s) { if (length(s) == 3) s[3] else NA })
twitter_name[!is.na(twitter_name)]
```

```
## [1] "larsklingbeil"
```

Final words on web scraping

This whole process can be applied to all MPs (whose profile URLs we can get from the abgeordnetenwatch.de API). If we obey to the crawl limit of 1 request per 10 seconds as specified in their robots.txt file, it would take about 2h to fetch the profile pages of all 708 MPs and extract the Twitter name from it.

You can see that web scraping is really a powerful tool for automated data extraction from websites, but also that it involves much programming effort and many things can go wrong (see legal and technical issues slides before).

WZB Wissenschaftszentrum Berlin für Sozialforschung

Literature

- Munzert et al. 2015: Automated Data Collection with R
- Matthew A. Russel 2014: Mining the Social Web (uses Python)
- H. Wickham: Scraping with rvest and "SelectorGadget"

Tasks

See dedicated tasks sheet on the tutorial website.