A Primer to Web Scraping with R

Simon Munzert

Department of Social Sciences

Humboldt University of Berlin

simonmunzert.github.io

github.com/simonmunzert

@simonsaysnothin

Slides and Materials: https://git.io/vyAde

Overview

1. Introductory example
2. Why web scraping?
3. Web scraping with R
4. Tapping APIs with R
5. Outlook and helpful resources

Introductory example

Data on the web

Data on the web

borough's mayors make up the council of mayors (*Rat der Bürgermeister*), which is led by the city's Governing Mayor and advises the Senate. The neighborhoods have no local government bodies.

Twin towns – sister cities [edit]

See also: List of twin towns and sister cities in Germany

Berlin maintains official partnerships with 17 cities.[100] Town twinning between Berlin and other cities began with its sister city Los Angeles in 1967. East Berlin's partnerships were canceled at the time of German reunification but later partially reestablished. West Berlin's partnerships had previously been restricted to the borough level. During the Cold War era, the partnerships had reflected the different power blocs, with West Berlin partnering with capitals in the Western World, and East Berlin mostly partnering with cities from the Warsaw Pact and its allies.

There are several joint projects with many other cities, such as Beirut, Belgrade, São Paulo, Copenhagen, Helsinki, Johannesburg, Mumbai, Oslo, Shanghai, Seoul, Sofia, Sydney, New York City and Vienna. Berlin participates in international city associations such as the Union of the Capitals of the European Union, Eurocities, Network of European Cities of Culture, Metropolis, Summit Conference of the World's Major Cities, and Conference of the World's Capital Cities. Berlin's official sister cities are:[100]

- 1967 Los Angeles, United States
- 1987 Paris, France
- 1988 Madrid, Spain
- 1989 Istanbul, Turkey
- 1991 Warsaw, Poland[101]
- 1991 Moscow, Russia
- 1992 Brussels, Belgium
- 1992 Budapest, Hungary[102]
- 1993 Tashkent, Uzbekistan
- 1993 Mexico City, Mexico
- 1993 Jakarta, Indonesia
- 1994 Beijing, China
- 1994 Tokyo, Japan
- 1994 Buenos Aires, Argentina
- 1995 Prague, Czech Republic[103]
- 2000 Windhoek, Namibia
- 2000 London, United Kingdom

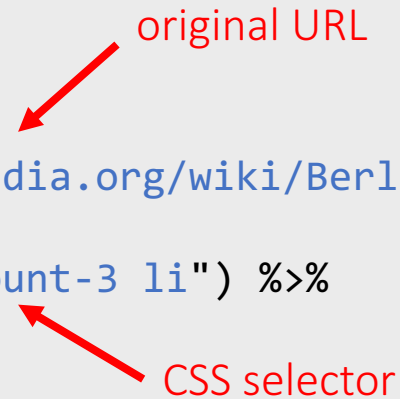
Capital city [edit]

Berlin is the capital of the Federal Republic of Germany. The President of Germany, whose functions are mainly ceremonial under the German constitution, has his official residence in Schloss Bellevue.[104] Berlin is the seat of the German executive, housed in the Chancellery, the *Bundeskanzleramt*. Facing the Chancellery is the Bundestag, the German Parliament, housed in the renovated Reichstag building since the government relocated to Berlin in 1998. The Bundesrat ("federal council", performing the function of an upper house) is the representation of the Federal States (*Bundesländer*) of Germany and has its

Let‘s grab these data!

```
# load packages
library(rvest)
library(stringr)

# scrape page source
cities_string <- read_html("https://en.wikipedia.org/wiki/Berlin") %>%
# extract list items
                 html_nodes(css = ".column-count-3 li") %>%
                 html_text()
cities_string
```

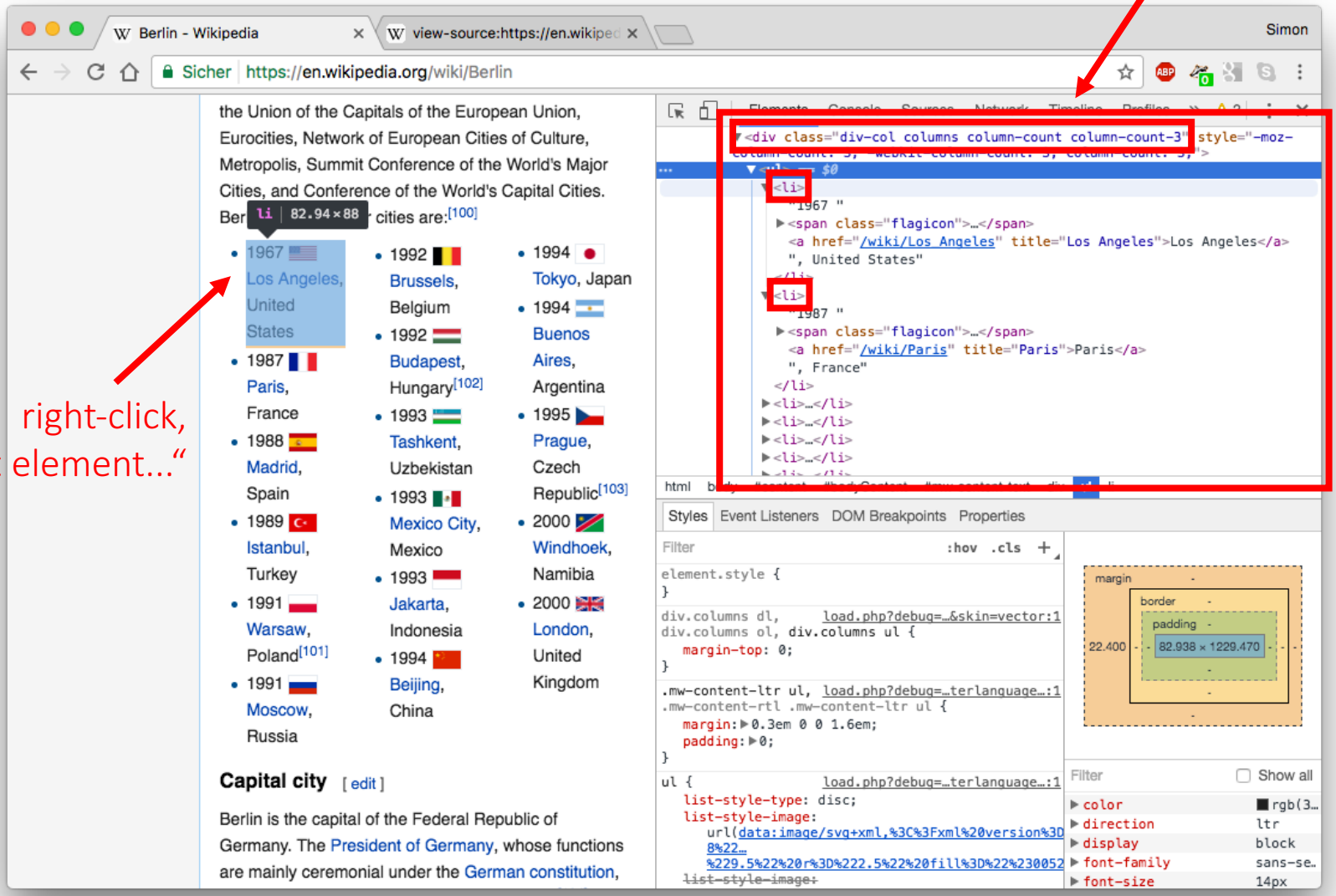



```
 [1] "1967  Los Angeles, United States"  "1987  Paris, France"
 [3] "1988  Madrid, Spain"               "1989  Istanbul, Turkey"
 [5] "1991  Warsaw, Poland[101]"         "1991  Moscow, Russia"
 [7] "1992  Brussels, Belgium"           "1992  Budapest, Hungary[102]"
 [9] "1993  Tashkent, Uzbekistan"        "1993  Mexico City, Mexico"
[11] "1993  Jakarta, Indonesia"          "1994  Beijing, China"
[13] "1994  Tokyo, Japan"                "1994  Buenos Aires, Argentina"
[15] "1995  Prague, Czech Republic[103]" "2000  Windhoek, Namibia"
[17] "2000  London, United Kingdom"
```

Why was this so easy?

view HTML source code

right-click,
„inspect element...“



Let‘s clean up these data!

```
# remove footnotes ([101], [102], ...)
cities_string <- str_replace(cities_string, "\\[\\d+\\]", "")

# extract data
year <- str_extract(cities_string, "\\d{4}")
city <- str_extract(cities_string, "[[:alpha:] ]+") %>% str_trim
country <- str_extract(cities_string, "[[:alpha:] ]+$") %>% str_trim
```


```
 [1] "1967  Los Angeles, United States"  "1987  Paris, France"
 [3] "1988  Madrid, Spain"               "1989  Istanbul, Turkey"
 [5] "1991  Warsaw, Poland"              "1991  Moscow, Russia"
 [7] "1992  Brussels, Belgium"           "1992  Budapest, Hungary"
 [9] "1993  Tashkent, Uzbekistan"        "1993  Mexico City, Mexico"
[11] "1993  Jakarta, Indonesia"         "1994  Beijing, China"
[13] "1994  Tokyo, Japan"               "1994  Buenos Aires, Argentina"
[15] "1995  Prague, Czech Republic"     "2000  Windhoek, Namibia"
[17] "2000  London, United Kingdom"
```

Let‘s clean up these data!

```
# remove footnotes ([101], [102], ...)
cities_string <- str_replace(cities_string, "\\[\\d+\\]", "")

# extract data
year <- str_extract(cities_string, "\\d{4}")
city <- str_extract(cities_string, "[[:alpha:] ]+") %>% str_trim
country <- str_extract(cities_string, "[[:alpha:] ]+$") %>% str_trim
```


```
year[1:5]
[1] "1967" "1987" "1988" "1989" "1991"

city[1:5]
[1] "Los Angeles"  "Paris"        "Madrid"       "Istanbul"     "Warsaw"

country[1:5]
[1] "United States" "France"        "Spain"         "Turkey"        "Poland"
```

Let‘s clean up these data!

```
# remove footnotes ([101], [102], ...)
cities_string <- str_replace(cities_string, "\\[\\d+\\]", "")

# extract data
year <- str_extract(cities_string, "\\d{4}")
city <- str_extract(cities_string, "[[:alpha:] ]+") %>% str_trim
country <- str_extract(cities_string, "[[:alpha:] ]+$") %>% str_trim

# put everything into data.frame
cities_df <- data.frame(year, city, country)
head(cities_df)
```

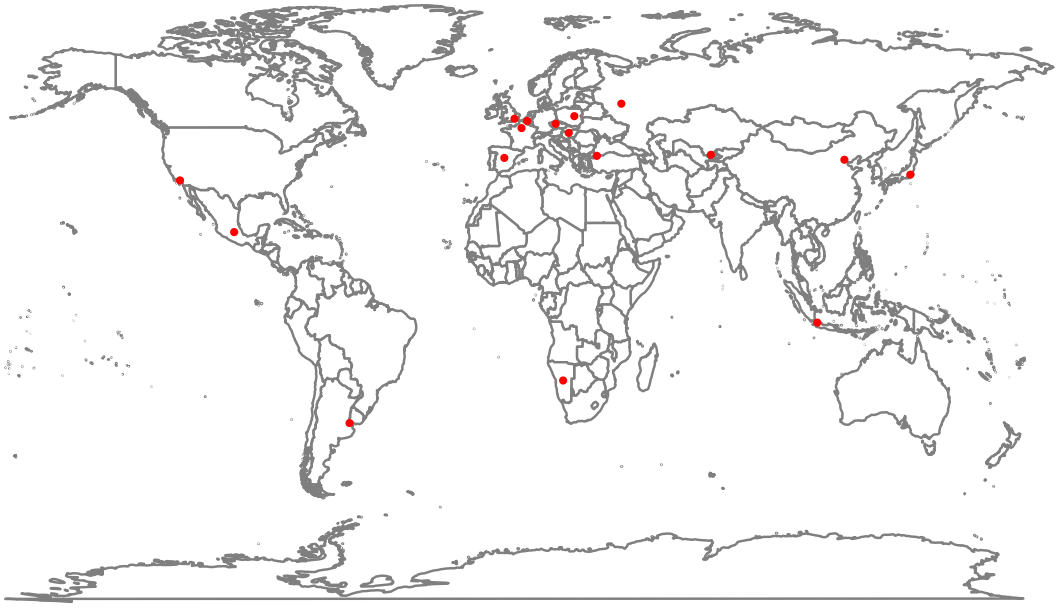

```
  year         city       country
1 1967 Los Angeles United States
2 1987        Paris        France
3 1988       Madrid         Spain
4 1989     Istanbul        Turkey
5 1991       Warsaw        Poland
6 1991       Moscow        Russia
```

Let‘s map these data!

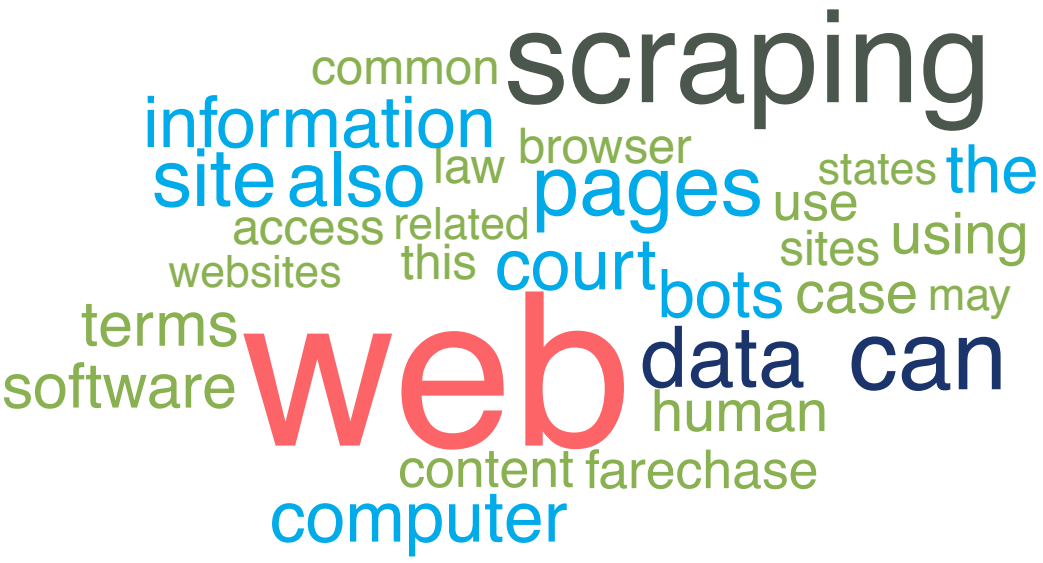
```
# geocode cities
library(ggmap)
cities_coords <- geocode(paste0(cities_df$city, ", ", cities_df$country))
cities_df$lon <- cities_coords$lon
cities_df$lat <- cities_coords$lat

# plot world map, add coordinates
map_world <- borders("world", colour = "gray50", fill = "white")
ggplot() + map_world + geom_point(aes(x = cities_df$lon, y = cities_df$lat),
color = "red", size = 1) + theme_void()
```

query Google Maps API



Why web scraping with R?



Why web scraping?

A data analyst‘s view

- data abundance online
- social interaction online
- services track social behavior
- online data meant for display, not download

A pragmatists‘s view

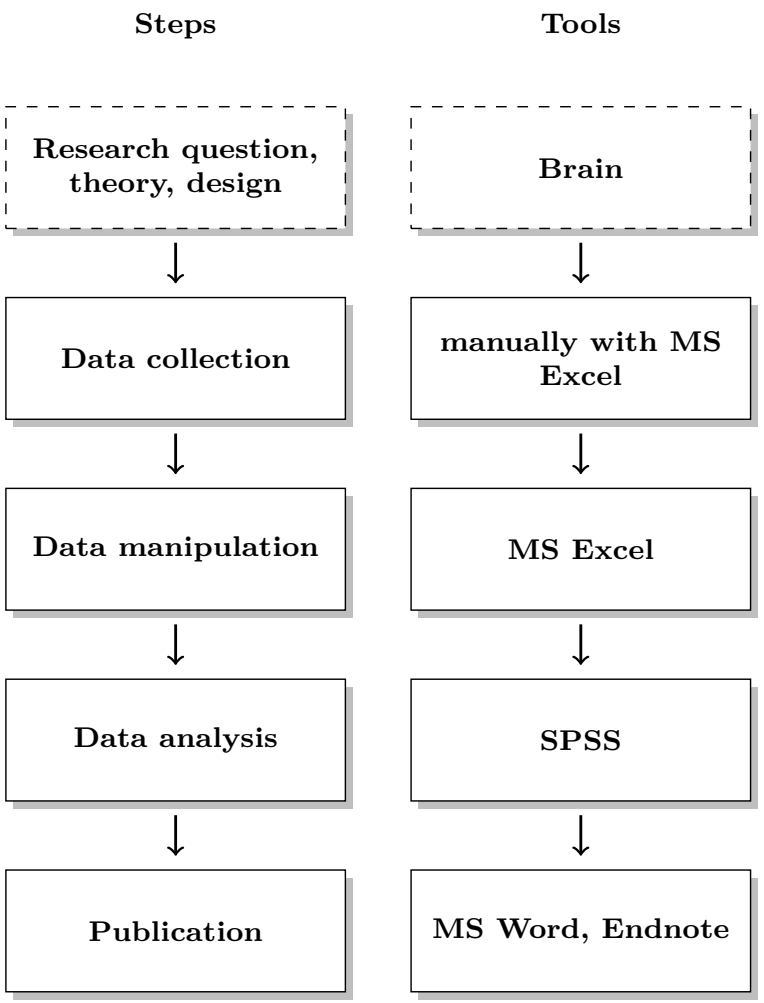
- financial resources
- time resources
- reproducibility
- updateability

Web scraping is the business of

- getting (unstructured) data from the web and
- bringing it into shape (e.g., clean, make tabular format)

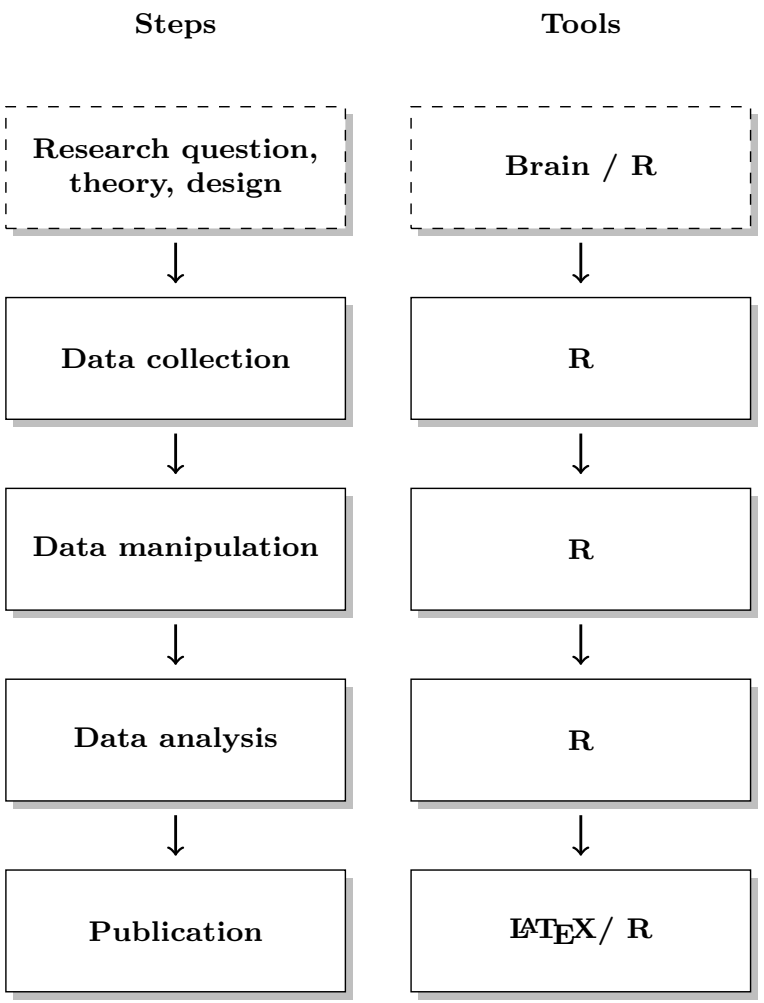
Why R?

- free
- open source
- large community
- powerful for statistical analysis
- powerful for visualization
- flexible in processing all kinds of data types
- useful in every step of the workflow



Why R?

- free
- open source
- large community
- powerful for statistical analysis
- powerful for visualization
- flexible in processing all kinds of data types
- useful in every step of the workflow

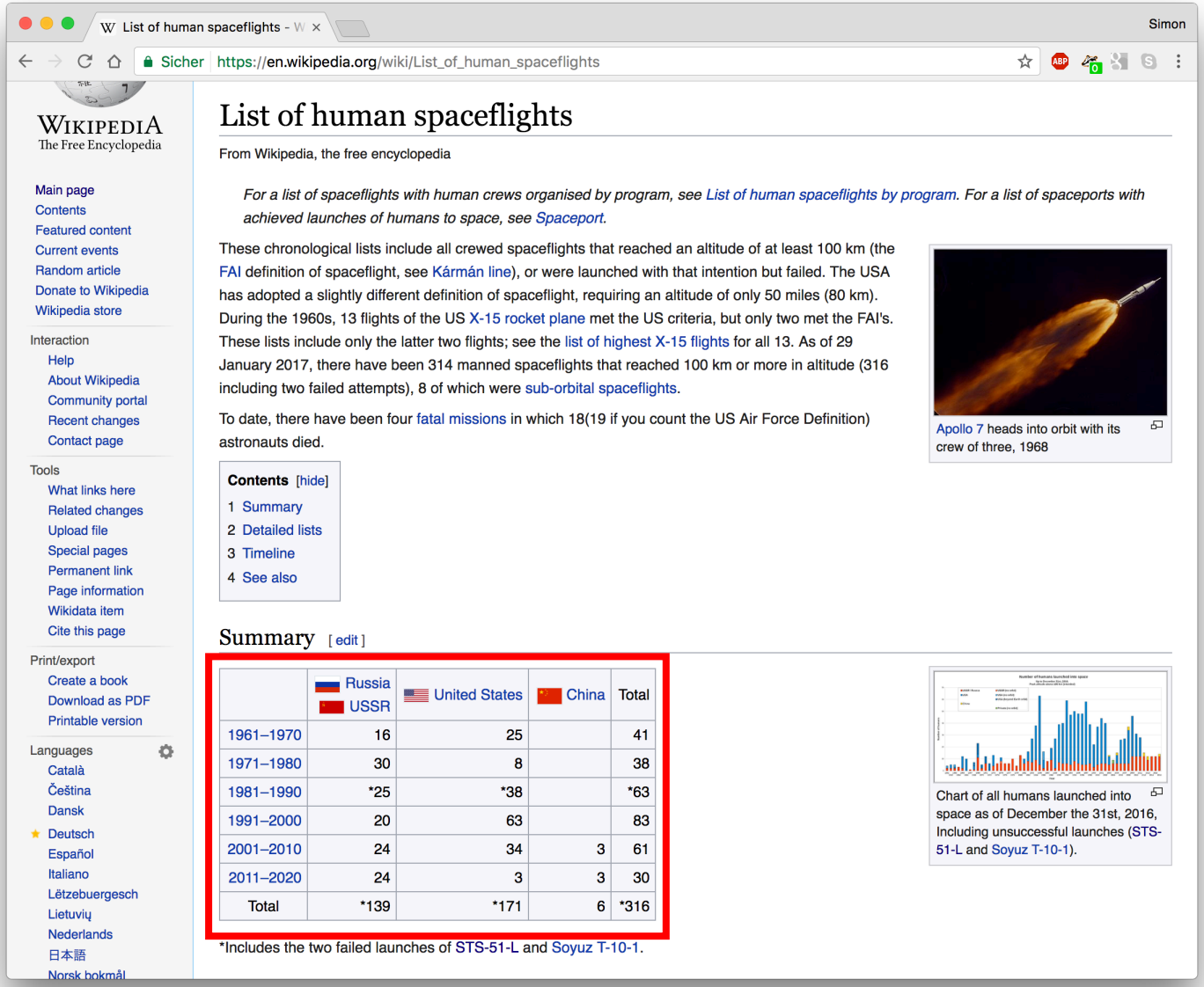


Web scraping with R at a glance

A beginner‘s toolbox (also helpful to experts)

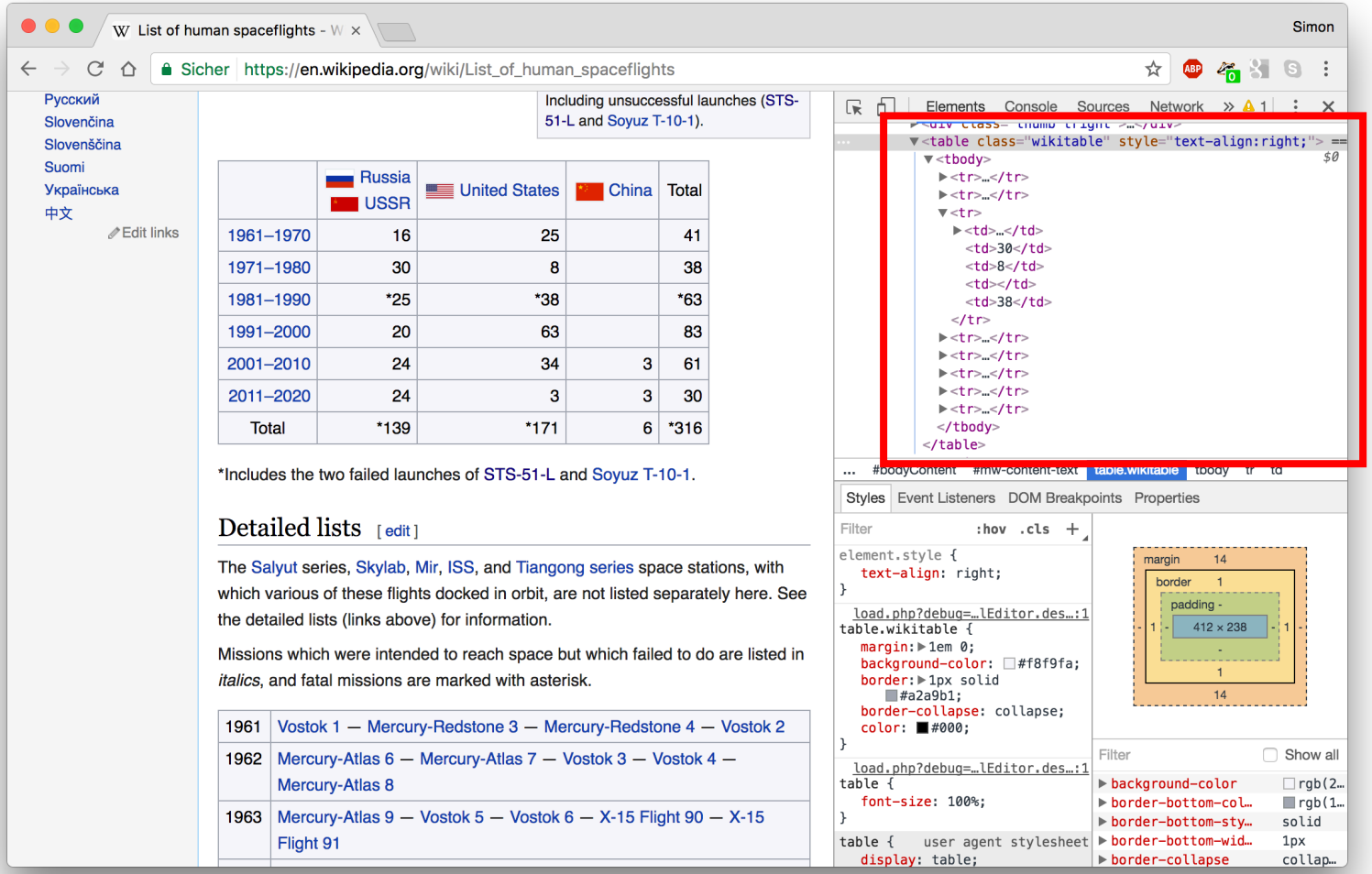
- `rvest`: easy web scraping (“harvesting”) suite (by Hadley Wickham)
- `xml2`: engine to parse XML-style files (by Hadley Wickham and James Hester)
- `stringr`: tools for string processing (by Hadley Wickham)

Scraping HTML tables



Scraping HTML tables

- the web developer tools and source code inspector are your friends (remember: right-click, “inspect element...”)



Scraping HTML tables

- the web developer tools and source code inspector are your friends (remember: right-click, “inspect element...”)

- HTML tables use to share a common format

```
<table class="wikitable">
    <tr>
        <td></td>
        <td>Russia USSR</td>
        <td>United States</td>
        <td>China</td>
        <td>Total</td>
    </tr>
    <tr>
        <td>1961–1970</td>
        <td>16</td>
        <td>25</td>
        <td></td>
        <td>41</td>
    </tr>
    ...
</table>
```

Let‘s grab these data!

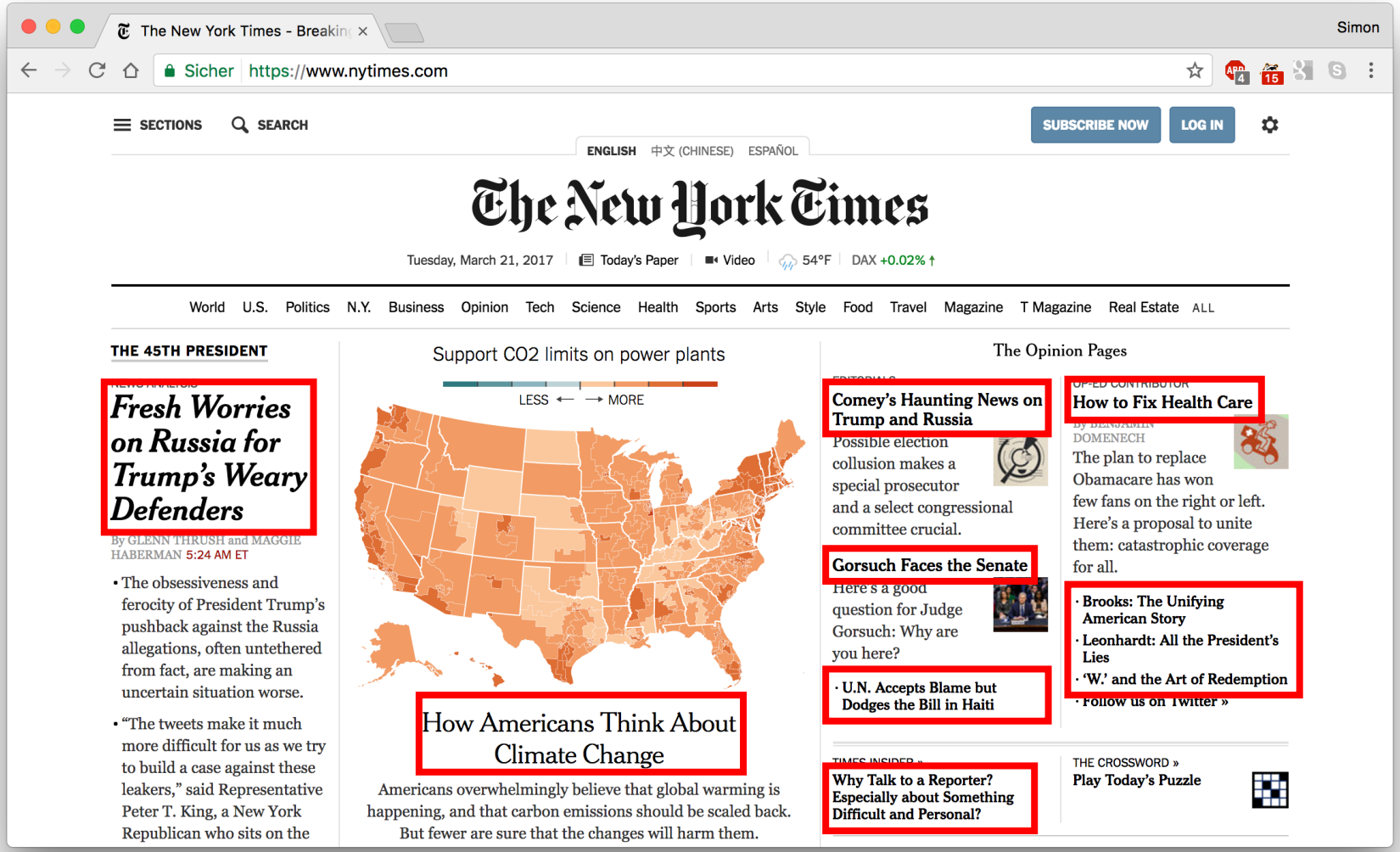
```
# load package
library(rvest)

# import webpage
url_p <- read_html("https://en.wikipedia.org/wiki/List_of_human_spaceflights")

# extract HTML tables
tables <- html_table(url_p, header = TRUE, fill = TRUE)
spaceflights <- tables[[1]]
spaceflights
```

```
            Russia USSR United States China Total
1 1961–1970          16             25    NA    41
2 1971–1980          30              8    NA    38
3 1981–1990         *25            *38    NA   *63
4 1991–2000          20             63    NA    83
5 2001–2010          24             34     3    61
6 2011–2020          24              3     3    30
7     Total        *139           *171     6  *316
```

What if data do not come in nice HTML tables?

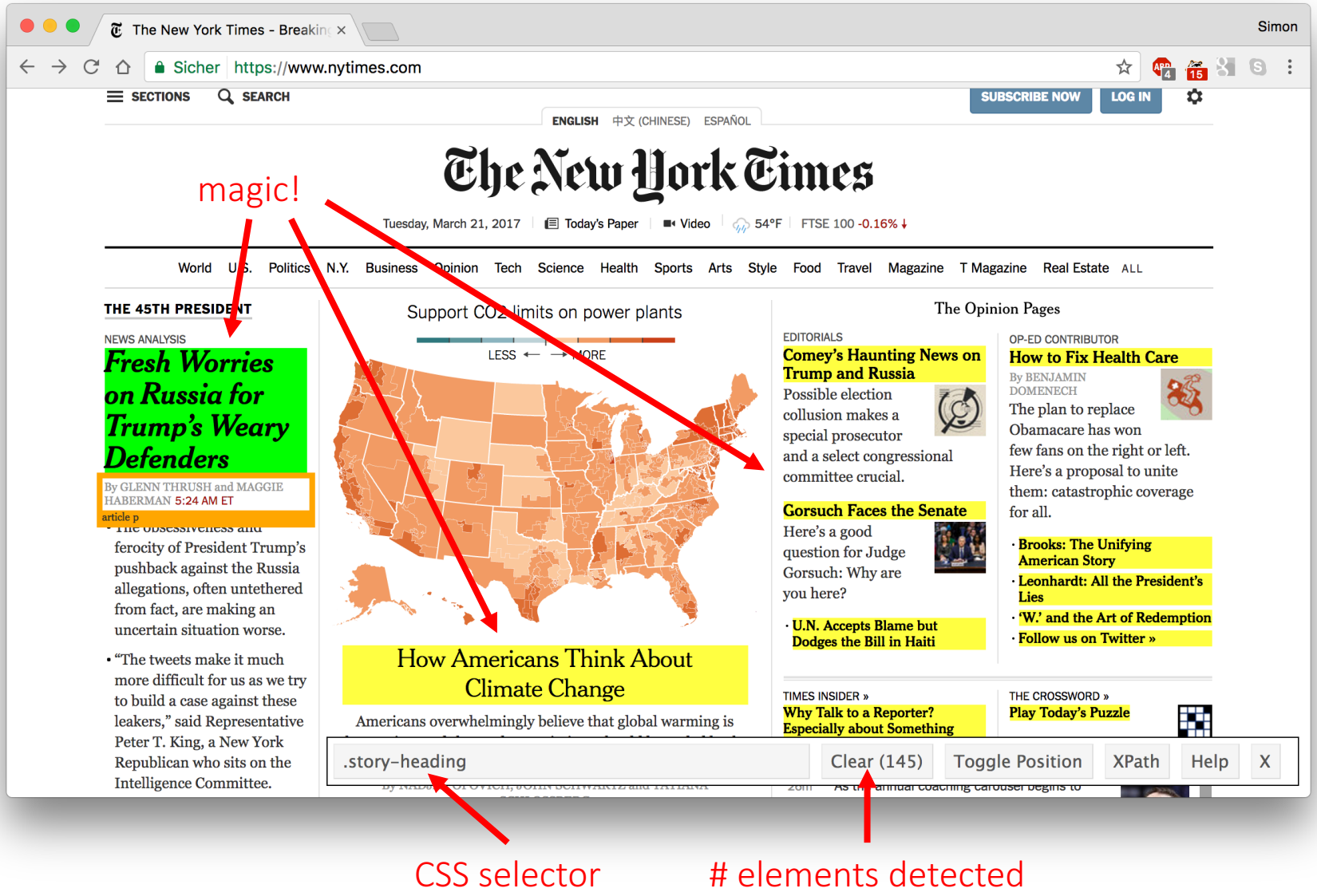


Easy solution: SelectorGadget

- available at http://selectorgadget.com/, works as browser plugin
- helps you construct CSS selectors (or XPath expressions)
- with SelectorGadget, you can
  - click on elements you want to scrape
  - un-click elements you do not want to scrape
  - “point-and-click” the information you are interested in
- caveat: solves 80% of your standard scraping tasks – for the rest, learn how to build CSS selectors / XPath expressions

SelectorGadget

SelectorGadget



Let‘s grab these data!

```
# load package
library(rvest)

# import webpage
url_p <- read_html("https://www.nytimes.com")

# look for nodes (parts of the HTML document that we want to extract)
headlines <- html_nodes(url_p, css = ".story-heading")

# extract the headline text
headlines_raw <- html_text(headlines)
head(headlines_raw)
```

```
[1] "Fresh Worries on Russia for Trump’s Weary Defenders"
[2] "F.B.I. Confirms Inquiry on Trump Team’s Russia Ties"
[3] "Takeaways From the Hearing "
[4] "\n                                Trump and the Russians: Links? No Links?"
[5] "G.O.P. Responds by Changing Subject"
[6] "U.S. Limits Devices on Foreign Airlines From 8 Countries"
```

Let‘s clean up these data!

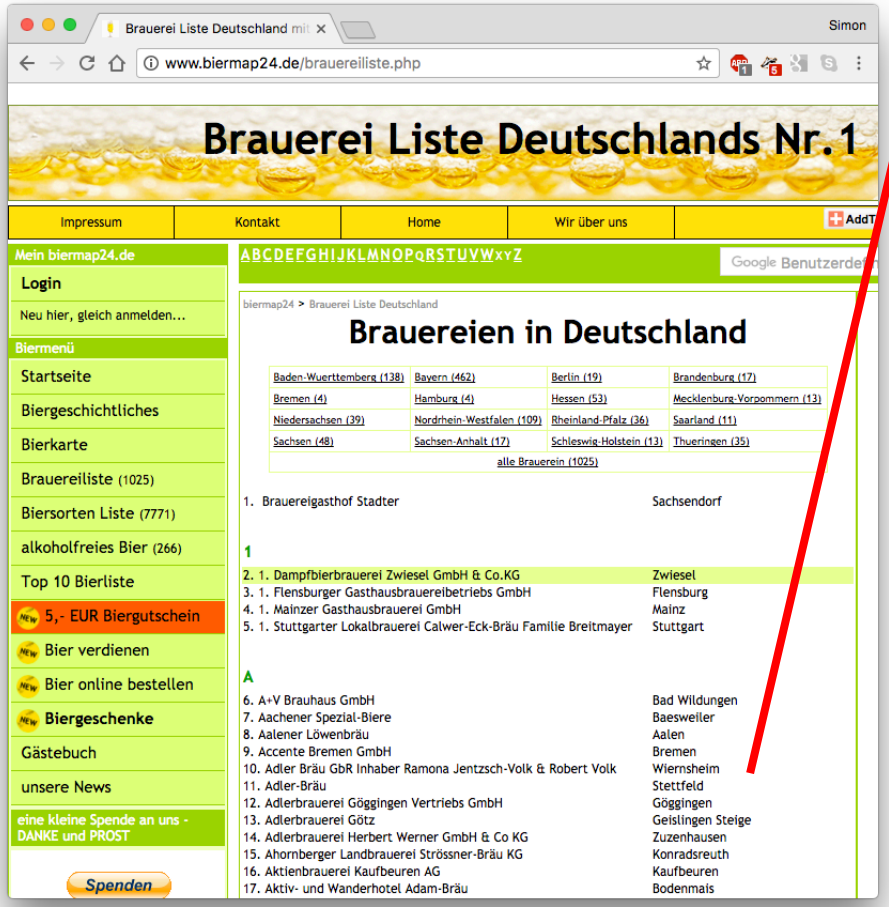
```
# remove line breaks and empty spaces
headlines_clean <- headlines_raw %>% str_replace_all("\\n", "") %>% str_trim()

# how many headlines contain the word "Trump"?
str_detect(headlines_clean, "Trump") %>% table()
```

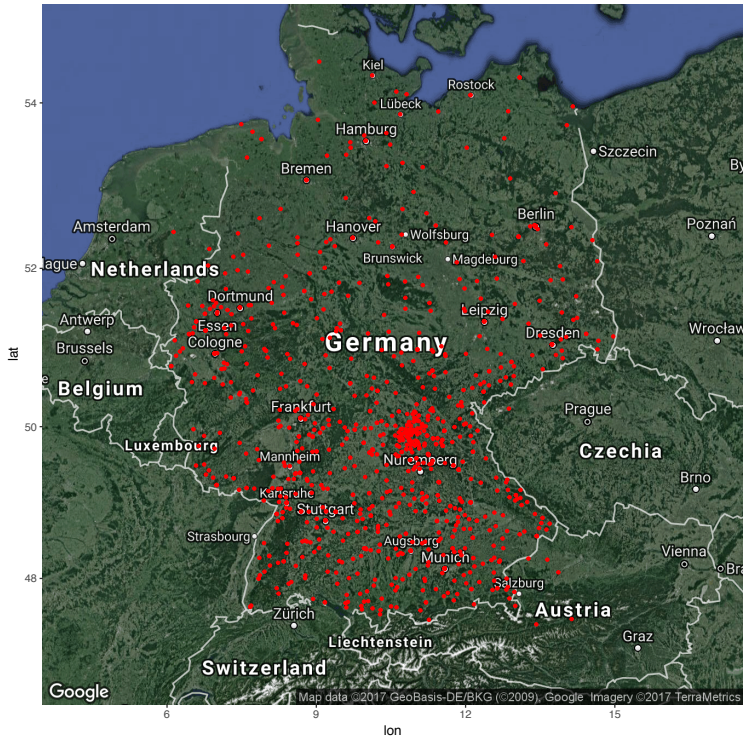
```
FALSE  TRUE 
  133    12 
```

Other scraping scenarios

- pull information out of lists
- geocode cities

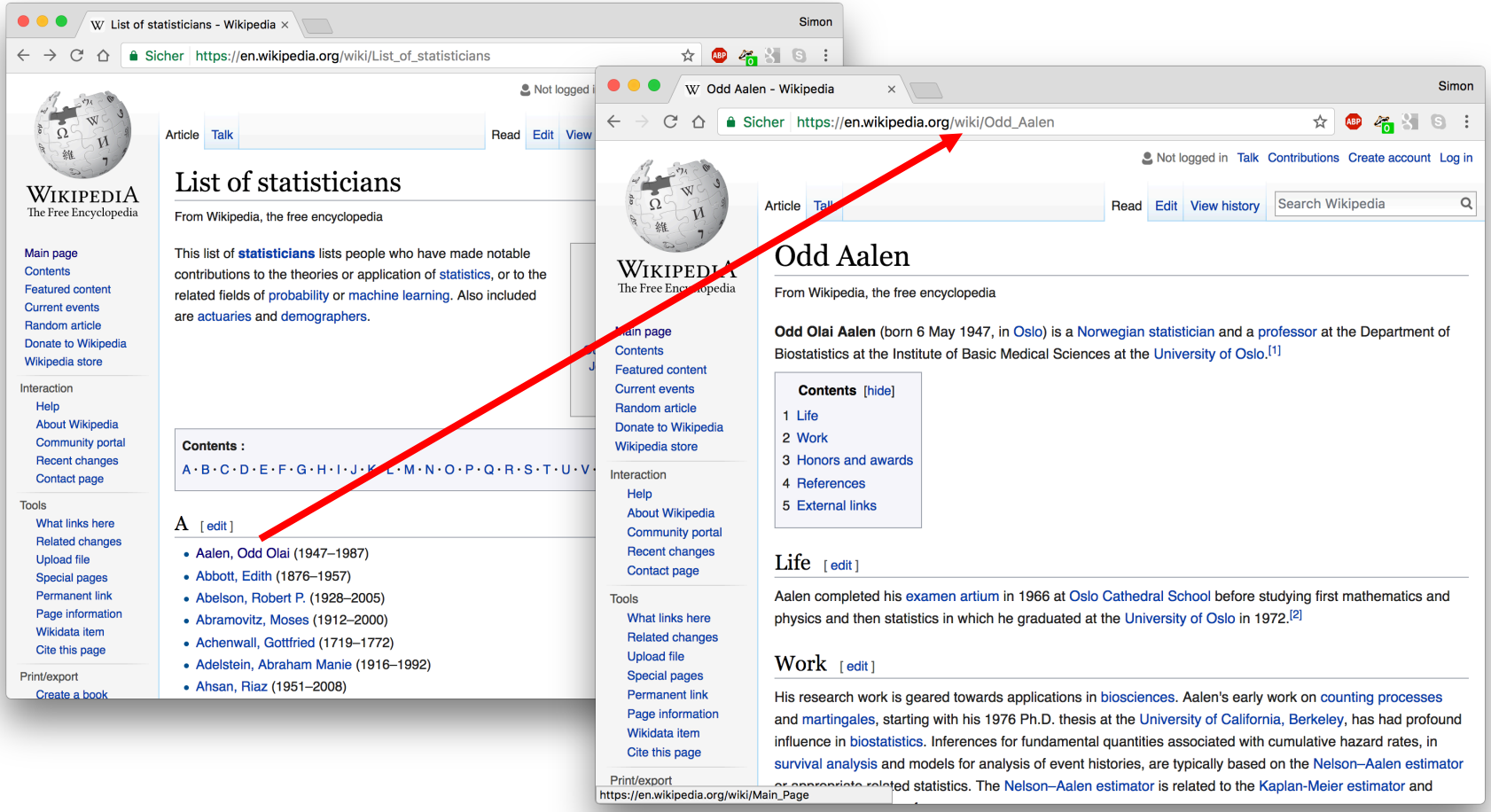



```
cities[1:15]
[1] "Flensburg"          "Mainz"
[3] "Stuttgart"          "Bad Wildungen"
[5] "Baesweiler"         "Aalen"
[7] "Bremen"             "Wiernsheim"
[9] "Stettfeld"          "Göggingen"
[11] "Geislingen Steige" "Zuzenhausen"
[13] "Konradsreuth"       "Kaufbeuren"
[15] "Bodenmais"
```



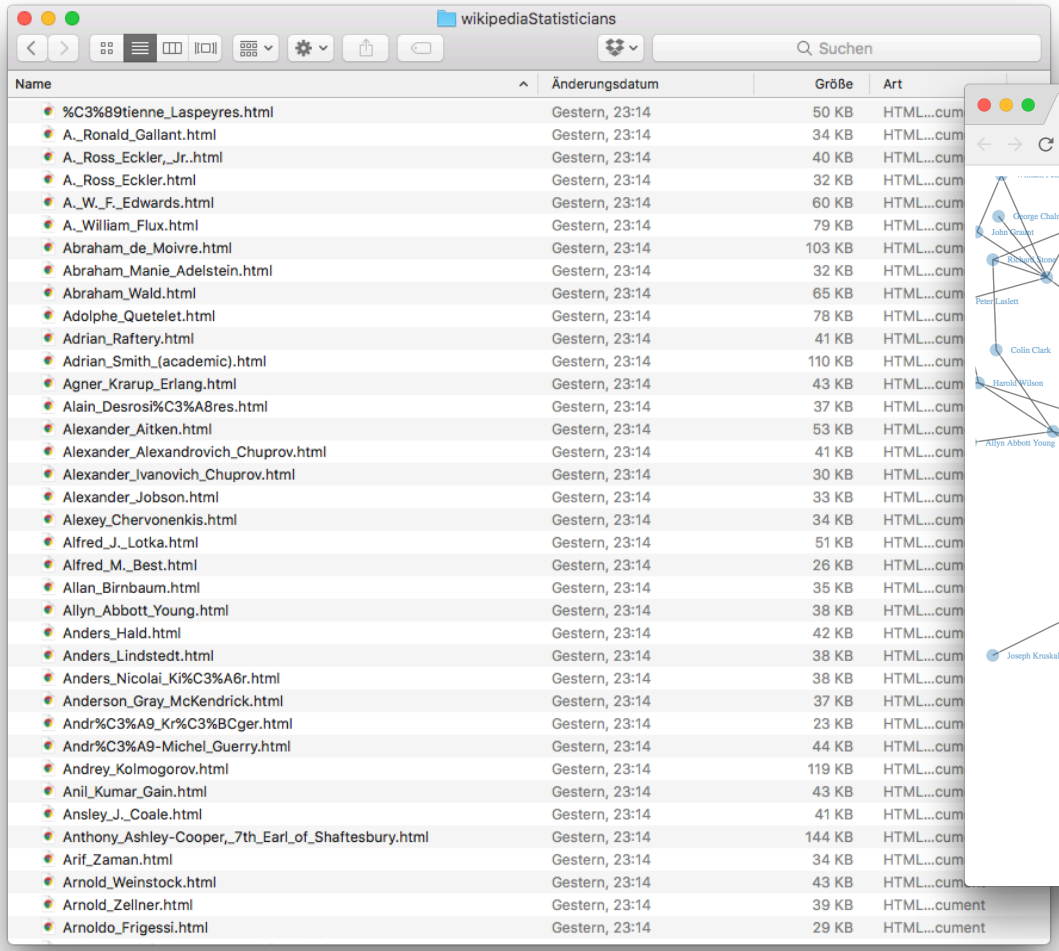
Other scraping scenarios

- download hundreds of (thousands of) HTML pages
- pull information from multiple websites
- identify links between HTML documents



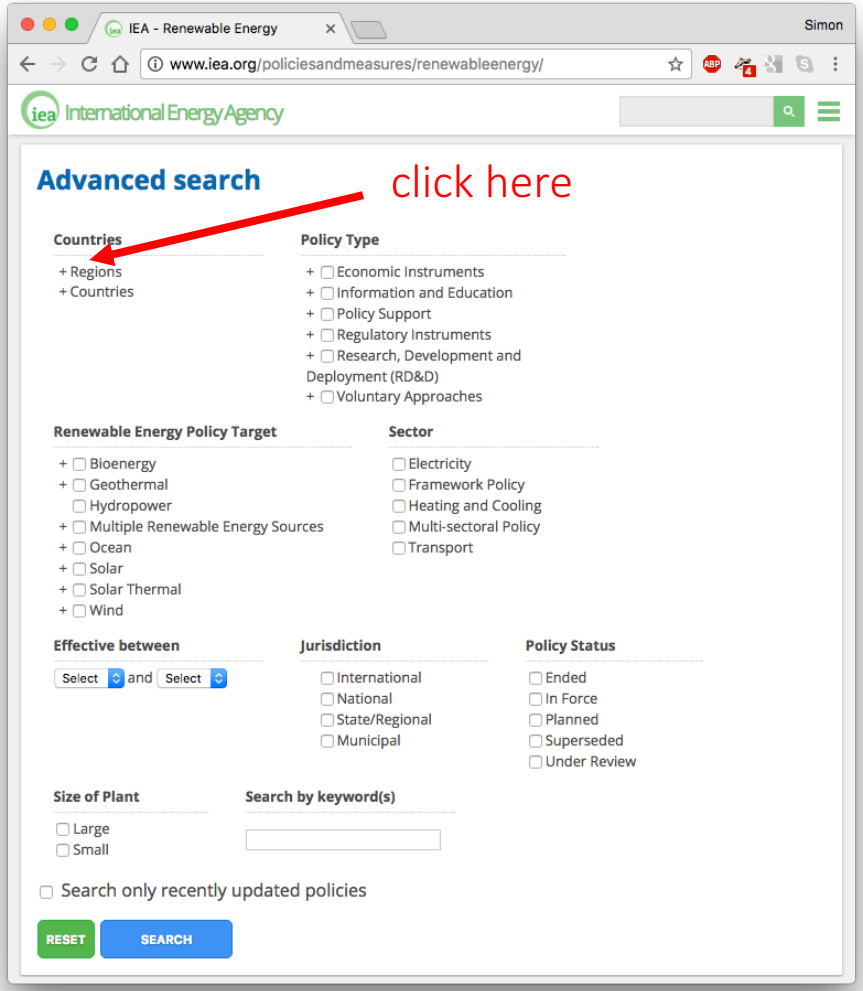
Other scraping scenarios

- download hundreds of (thousands of) HTML pages
- pull information from multiple websites
- identify links between HTML documents



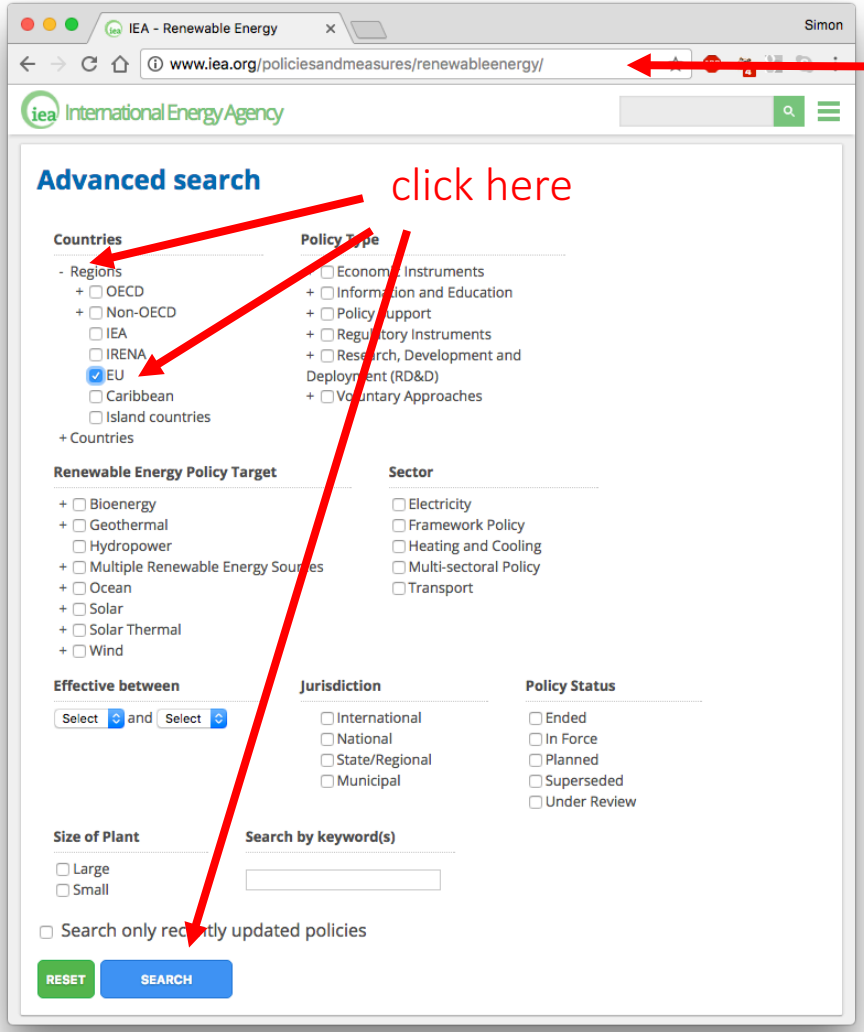
Other scraping scenarios

- gather data from dynamically rendered websites
- ...one of the more complex scraping scenarios



Other scraping scenarios

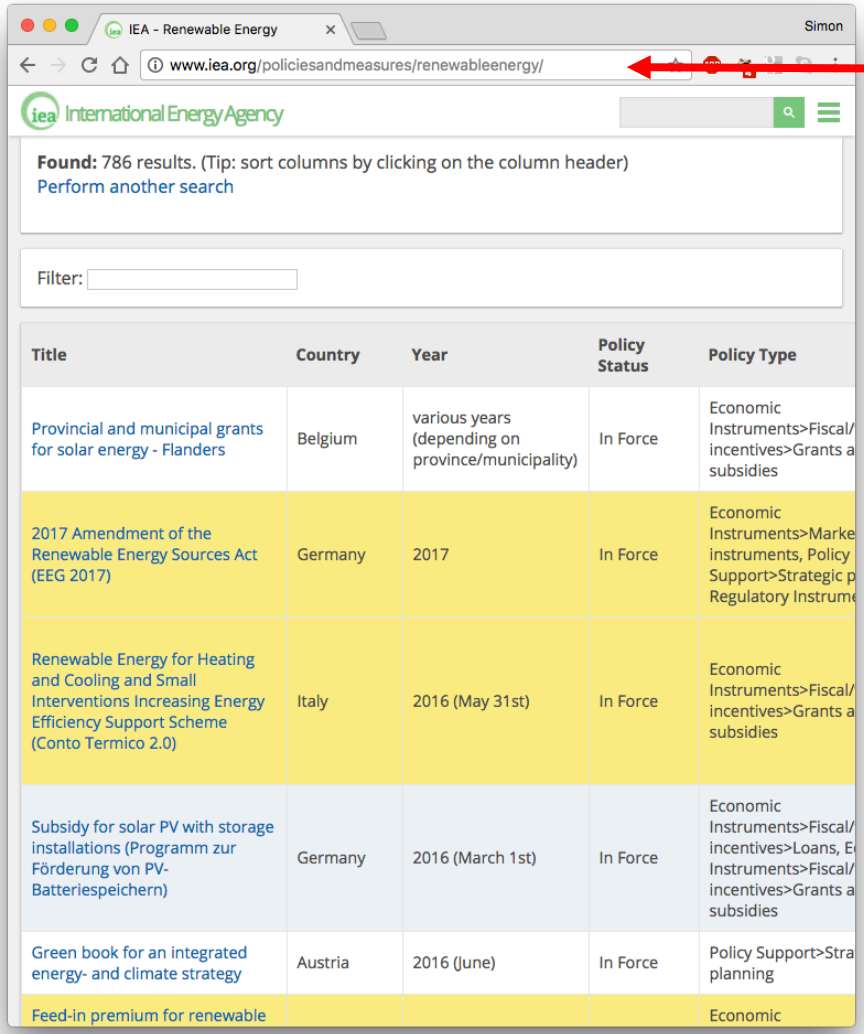
- gather data from dynamically rendered websites
- ...one of the more complex scraping scenarios



URL does not change

Other scraping scenarios

- gather data from dynamically rendered websites
- ...one of the more complex scraping scenarios



URL does not change

- make use of `RSelenium` together with Selenium Webdriver
- the basic idea: simulate activities in the browser with R, e.g. “click here“, “type something there“, and log the results using external software

Wrap-up

- the `rvest` package is your friend (plus SelectorGadget plus regular expressions)
- you don‘t have to become an HTML expert, but looking into a website‘s source code helps
- scraping data from static HTML tables can be as easy as this:

```
# load package
library(rvest)

# parse html
url_parsed <- read_html("https://www.example.com")

# extract table(s)
tables <- html_table(url_parsed, fill = TRUE)
your_table <- tables[[1]]

# tidy data
your_clean_table <- magic_data_tidier(your_table)
```

Wrap-up

- the `rvest` package is your friend (plus SelectorGadget plus regular expressions)
- you don‘t have to become an HTML expert, but looking into a website‘s source code helps
- scraping data from static HTML pages can be as easy as this:

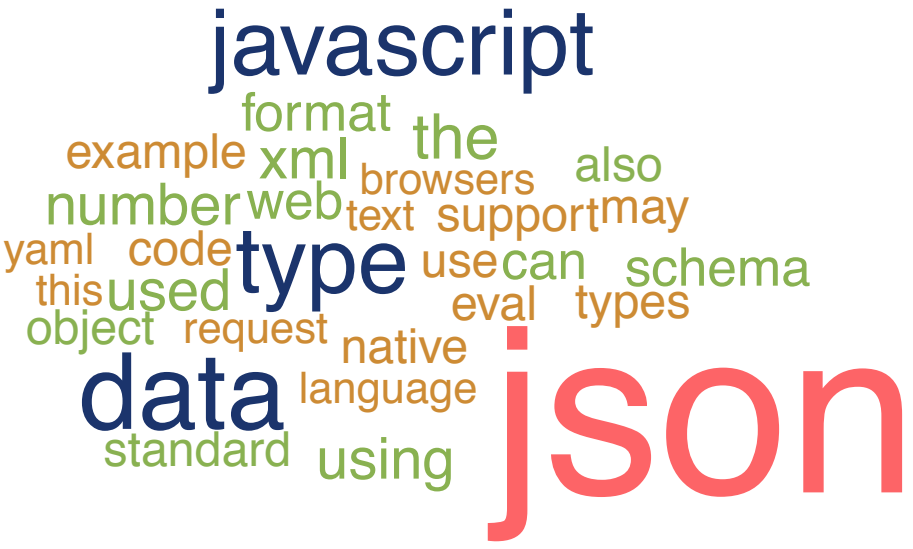
```
# load package
library(rvest)

# parse html
url_parsed <- read_html("https://www.example.com")

# extract content you like with CSS selector
nodes <- html_nodes(url_parsed, css = “CSS selector”)
your_data <- html_text(nodes)

# tidy data
your_clean_data <- magic_data_tidier(your_data)
```

Tapping APIs with R at a glance



What are APIs?

- Application Programming Interface
- many web services provide APIs to access their data and services (Google, Twitter, Facebook, Wikipedia, ...)
- often works like a “data search engine”: you pose a request, the API answers with a bulk of data
- common data return formats: JSON, XML

APIs in the context of web scraping

- instant access to clean data
- frees you from building manual scraping programs

➢ collecting data from the web using APIs provided by the data owner represents the gold standard of web-based data retrieval

Example: Google Maps API

- Google provides access to powerful location services
- free service (at least when used modestly)
- input/output: places, names, coordinates, meta information, maps, ...
- see also: https://developers.google.com/maps/documentation/

Potential use cases

- geocode observations based on address, city, post code, ...
- calculate distances between observations
- map observations

Example: Google Maps API

```
library(ggmap)
geocode("Humboldt University, Berlin")
```

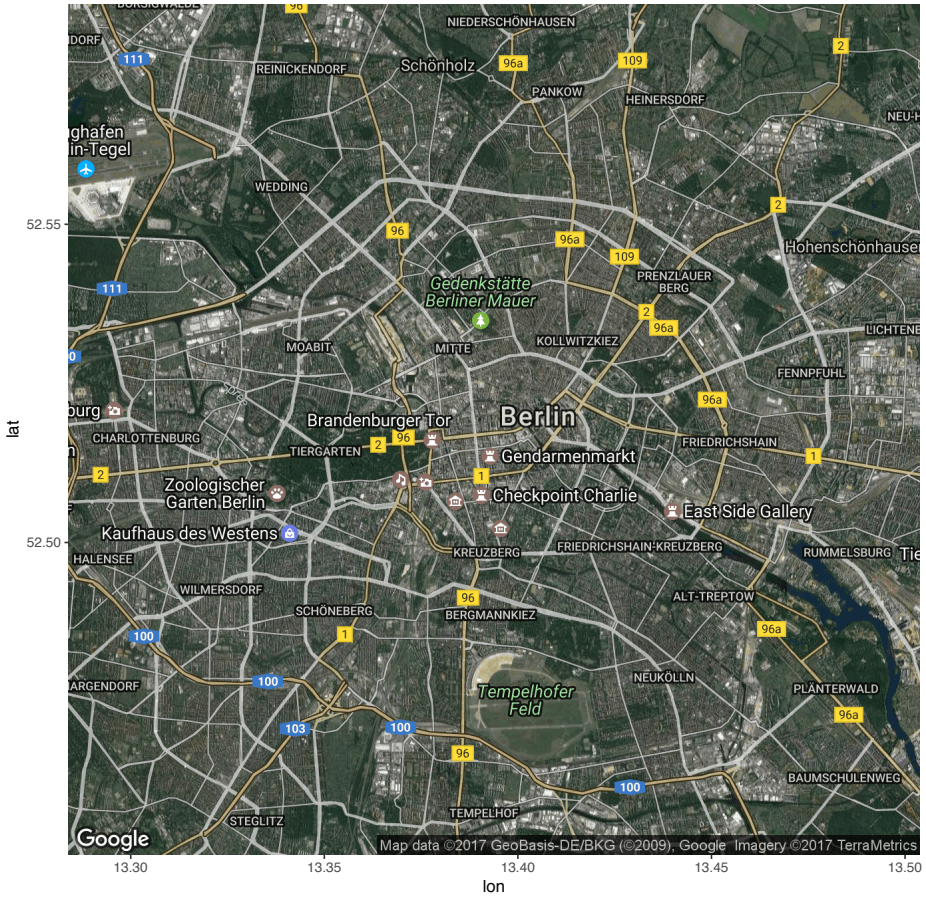
```
Source : https://maps.googleapis.com/maps/api/geocode/json?address=
Humboldt%20University%2C%20Berlin
       lon      lat
1 13.39366 52.51788
```

Excerpt from raw JSON


```
{
"address_components" : [
         { "long_name" : "Germany",
           "short_name" : "DE",
           "types" : [ "country", "political" ]},
         { "long_name" : "10099",
           "short_name" : "10099",
           "types" : [ "postal_code"] }
],
"formatted_address" : "Unter den Linden 6, 10099 Berlin, Germany",
"location" : { "lat" : 52.517883, "lng" : 13.3936551 },
"place_id" : "ChIJI0K8KtZNqEcRxSbyV3Fx5Kk",
"types" : [ "establishment", "point_of_interest", "university" ]
}
```

Example: Google Maps API

```
library(ggmap)
geocode("Humboldt University, Berlin")
get_googlemap("Humboldt University, Berlin", zoom = 12, maptype = "hybrid") %>%
ggmap()
```



Example: Google Maps API

```
library(ggmap)
geocode(“Humboldt University, Berlin”)
get_googlemap(“Humboldt University, Berlin”, zoom = 12, maptype = “hybrid”) %>%
ggmap()
ggmap_credentials()
```

```
Google -
key :
account_type : standard
day_limit : 2500
second_limit : 50
client :
signature :
```

Example: Twitter APIs

- Twitter provides access to much of their data for free
- output: user profile, follower data, tweet collections (keyword-based), ...
- registration mandatory (see, e.g., http://bit.ly/2mM4WwV)
- see also: https://dev.twitter.com/rest/public

Potential use cases

- analyze users‘ output (tweets, retweets)
- analyze follower networks
- real-time tracking of activity on the platform

Example: Twitter APIs

```
library(rtweet)

## search for tweets
tweets_aapor <- search_tweets("#AAPOR", n = 100)
names(tweets_aapor)
tweets_aapor$text[1:3]
```

```
Searching for tweets...
Finished collecting tweets!

 [1] "screen_name"                 "user_id"                     "created_at"
 [4] "status_id"                   "text"                        "retweet_count"
 [7] "favorite_count"              "is_quote_status"             "quote_status_id"
[10] "is_retweet"                  "retweet_status_id"           "in_reply_to_status_status_id“
[13] "in_reply_to_status_user_id"  "in_reply_to_status_screen_name" "lang“
[16] "source"                      "media_id"                    "media_url"
[19] "media_url_expanded"          "urls"                        "urls_display"
[22] "urls_expanded"               "mentions_screen_name"        "mentions_user_id"
[25] "symbols"                     "hashtags"                    "coordinates"
[28] "place_id"                    "place_type"                  "place_name"
[31] "place_full_name"             "country_code"                "country"
[34] "bounding_box_coordinates"    "bounding_box_type"

 [1] "Come on, #AAPOR research nerds. I know you want to register for this. https://t.co/GpS7vGX6pb"
 [2] " @kyleymcg: Have you registered for the next #AAPOR Webinar on 4/12? A Primer to Web Scraping w/ R
by @simonsaysnothin https://t.co/exFDD… "
 [3] " @AAPOR: 2017 #AAPOR Award Winners Announced and will be honored during the Annual Conference in New
Orleans. https://t.co/eLfP0F5iNR "
```

Wrap-up

Advantages of data gathering with APIs

- pure data collection without “layout waste“
- standardized data access, robustness of calls
- de facto automatic agreement of data owner

Potential pitfalls

- requires you to understand the basics of the API architecture (or to the piece of software that links to it, e.g., an R package)
- dependent upon API suppliers; sometimes requires registration
- not always free

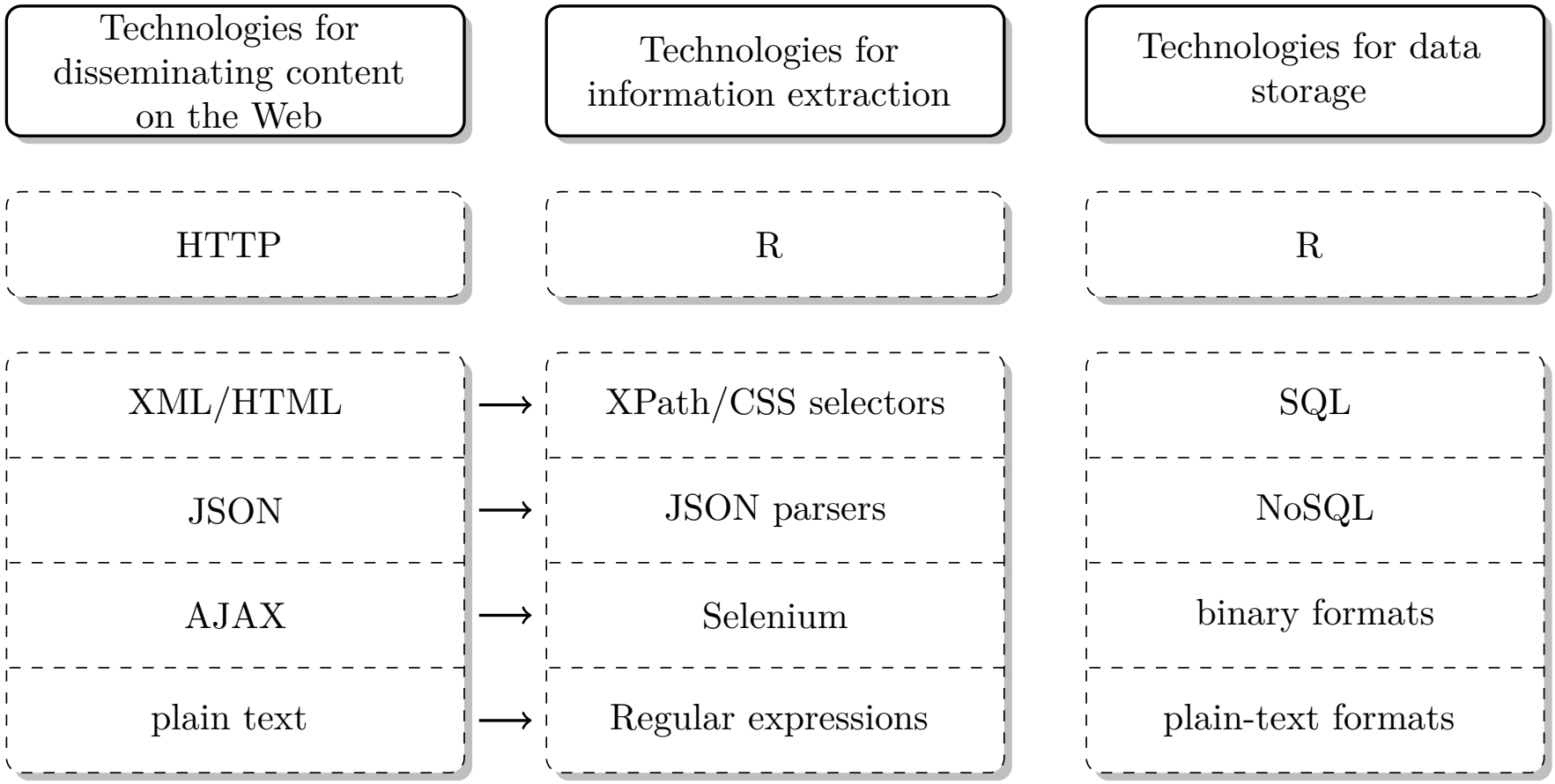
Wrap-up

Resources

- Collection of APIs on the web: http://www.programmableweb.com/apis
- rOpenSci: Collection of R-API interfaces: http://ropensci.org/packages/
- Twitter mining with R: https://mkearney.github.io/rtweet/index.html

Outlook and helpful resources

Web technologies



Three next steps for you (takes less than 3 hours)

1. Go to the following website and 

the repository to your desktop:

   https://git.io/vyAde

2. Start with opening `01-intro.r` and work through the commented R code. Then do the same with `02-scraping-with-rvest.r`

3. By modifying existing examples in the code, get better at understanding the individual steps in the process and ultimately start your own scraping projects

Midterm goals to become a seasoned web scraper (takes 3-5 projects)

1. Learn to understand and construct regular expressions

   http://stat545.com/block022_regular-expression.html

2. Learn to build CSS selectors (or XPath expressions) by hand

   http://flukeout.github.io/

3. Integrate more packages in your scraping toolbox

   https://cran.r-project.org/ web/views/WebTechnologies.html

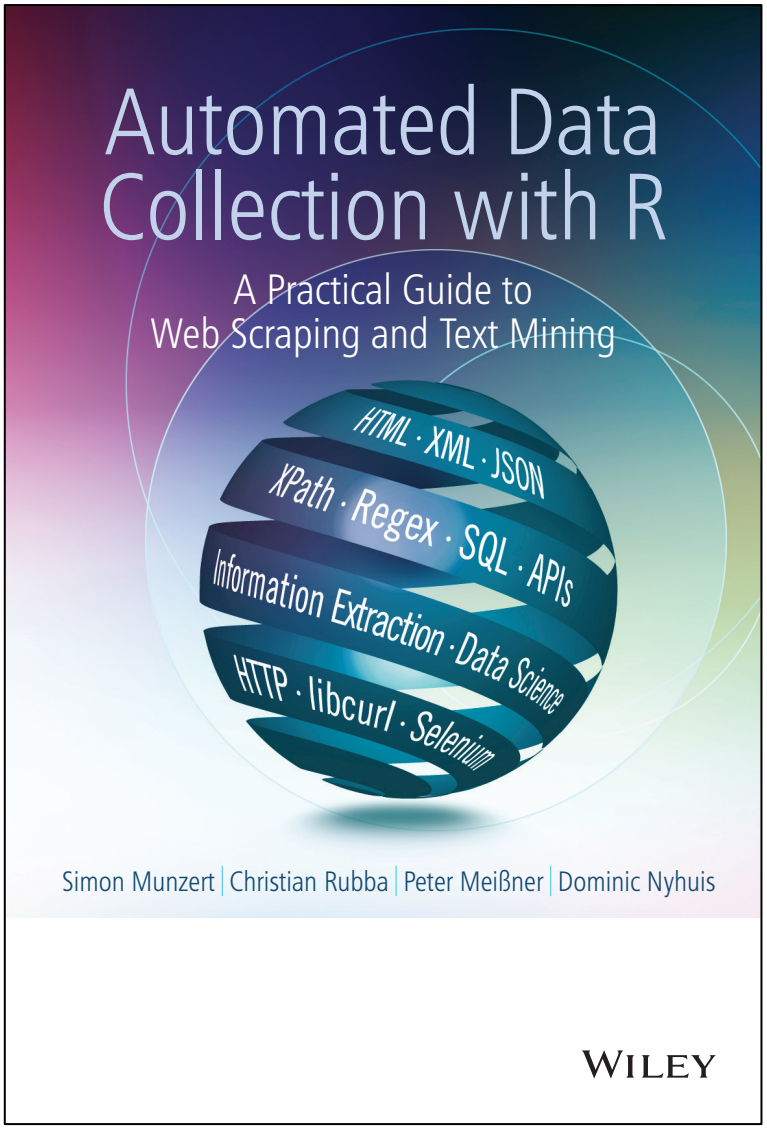
An accompanying book

- covers the entire scraping workflow in R and:
  - fundamentals of web technologies (HTTP, HTML, XML, JSON, XPath)
  - regular expressions
  - text mining
  - much more

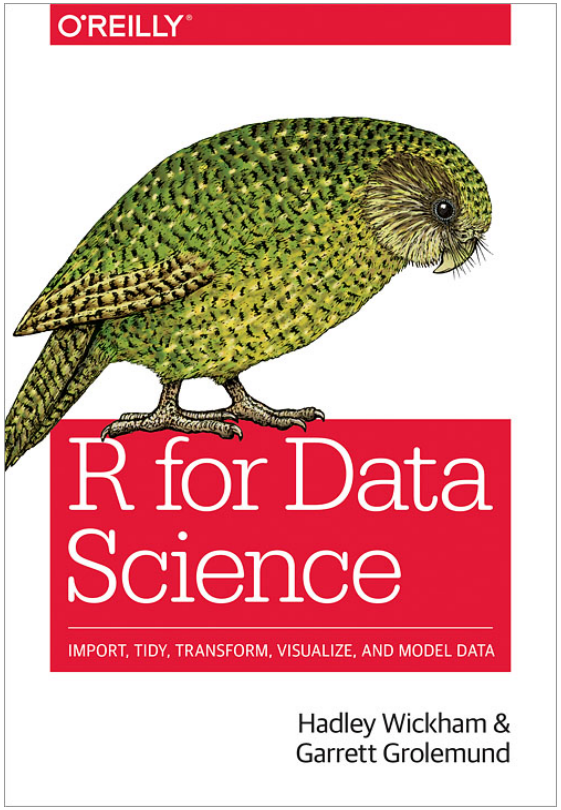
- published in late 2014 → not entirely up-to-date anymore (we‘re working on 2nd ed.)

- homepage with materials:

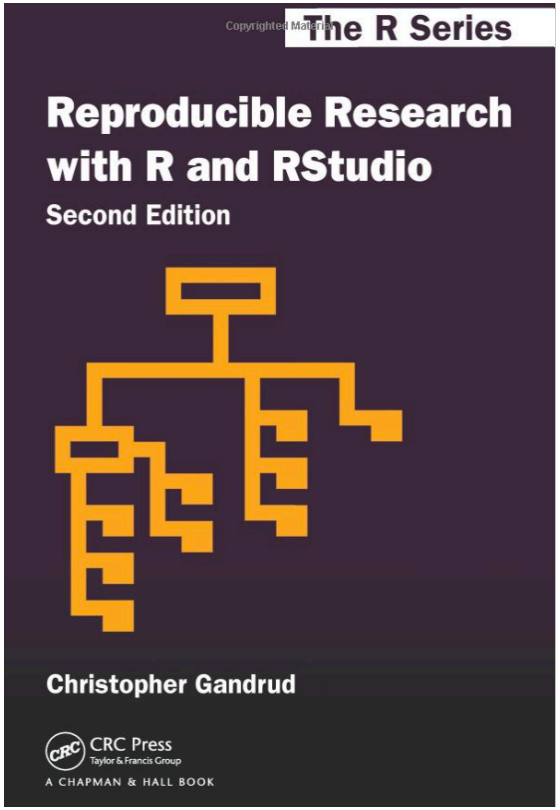
  r-datacollection.com



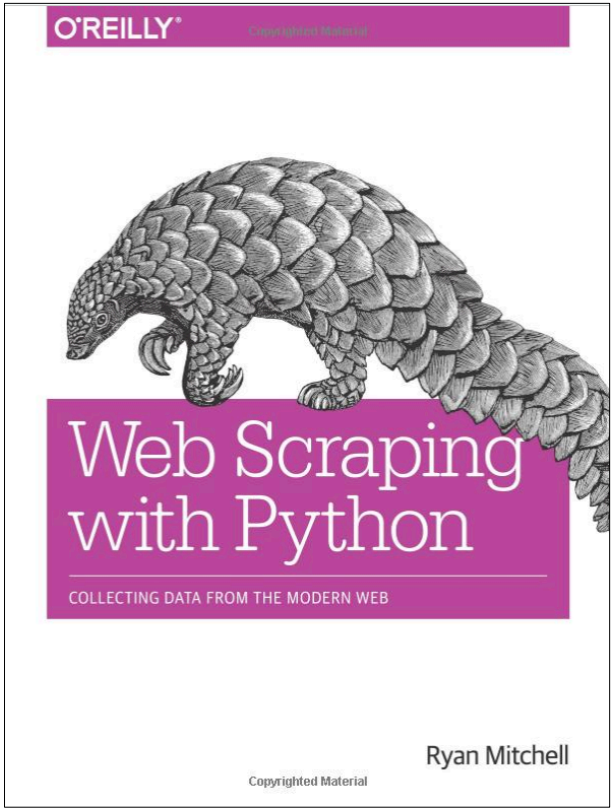
Other useful books on the market



- modern intro to R
- available for free at: http://r4ds.had.co.nz/



- helpful at establishing robust (scraping) workflow



- web scraping works with other programming languages, too!

Slides and Materials:

https://git.io/vyAde

Book:

r-datacollection.com
@RDataCollection

Me:

simonmunzert.github.io
github.com/simonmunzert
@simonsaysnothin