Universität Konstanz

Studying News Use with Computational Methods

Data Collection in R, Part II: Collecting News Articles

Julian Unkel
University of Konstanz
2021/05/17

Agenda

Universität Konstanz

The widespread availability of machine-readable news texts is one of the main reasons for the proliferation and advancement of automated content analysis methods in the field of Communication.

Digitally stored news texts (e.g., on online news sites or in large text databases) have made it simpler than ever to acquire large corpora of texts in comparatively little time.

In this session, we will deal with common approaches to collect news articles.

Our agenda today:

- Web scraping
  - Basics
  - Scraping with `rvest`
  - Good practices
- News APIs
  - Basics
  - MediaCloud
- News databases
  - Basics
  - Parsing text files
  - Example: NexisUni with `LexisNexisTools`

Web scraping

HTML

Universität Konstanz

Websites are mainly written in *HTML* (*H*yper*t*ext *M*arkup *L*anguage), marking up plain text into (nested) HTML elements:

```
<html>
  <body>
    <div class="main">
      <h1>A level-1 headline</h1>
      <p>A paragraph</p>
      <p>Another paragraph with <strong>bold</strong> and <a href="link.html">linked</a> text.</p
    </div>
  </body>
</html>
```

HTML *elements* consist of up to three parts:

- a *tag*, defining the element by opening it with `<tagname>` and closing it with `</tagname>`. See here for a comprehensive list of all tags.
- optional *attributes* defined in the opening tag by `key = "value"` pairs
- the plain *text* of the element *Text*

HTML tags

Tag	Description
`<head>`	site head with meta information (title, language, encoding, etc.)
`<body>`	body (actual content of the site)
`<p>`	paragraph
`<a>`	link ("anchor"); link target given by attribute `href`
`<strong>` / `<b>`	bold
`<em>` / `<i>`	emphasis, italics
`<h1>`, `<h2>` etc.	headline of level 1, level 2, etc.
`<table>`	table
`<ol>`, `<ul>`	ordered list, unordered list
`<li>`	list entry
`<div>`	container (formating parts of the website)
`<span>`	inline container (formating single text passages)
`<img>`	image; image file defined by attribute `src`

CSS

Universität Konstanz

Styling of HTML elements is usually handled by one or more stylesheet files in *CSS* (*C*ascading *S*tylesheets). Stylesheets define rules for specific HTML tags, classes (multiple HTML elements on the same page may have the same class) or IDs (unique IDs per element and page):

```
/* A class */
.blueOnRed {
  color: "blue";
  background-color: "red";
}

/* An ID */
#article_hl {
  font-family: sans-serif;
}

/* Tag-specifc styling */
h1 {
  font-size: 2em;
}
```

Classes and IDs can be applied as element attributes in HTML: `<h1 class="blueOnRed" id="article-hl">`.

Web scraping

Universität Konstanz

Web scraping (for our purposes) consists of the following steps:

- Requesting an HTML file from a web server
- Selecting the HTML elements of interest, most commonly by their tag names, classes, IDs, nesting/hierarchical placement, or any combination thereof
- Extracting the relevant information, for example the element's text or specific attributes (e.g., the `href` attribute of <a> link elements)

Two useful helpers for element identification/selection:

- Your browser's "inspect" feature (right-click any part of a website and select "inspect")
- The SelectorGadget bookmarklet

Web scraping with `rvest`

While we could just use `HTTR::GET()` and base text parsing functions for web scraping, the package `rvest` simplifies the whole process:

```
install.packages("rvest")
```

```
library(rvest)
```

The main functions are:

- Request the HTML file with `read_html()`
- Select elements with `html_nodes()`
- Extract relevant information with `html_text()` (plain text), `html_attr()` (element attributes) and/or `html_table()` (convenience function for tables)

Web scraping with `rvest`

Universität Konstanz

Let's scrape some information from Wikipedia's Lake Constance article. First, read the HTML file:

```
wiki_html <- read_html("https://en.wikipedia.org/wiki/Lake_Constance")
```

Extract the level 1 headline:

```
wiki_html %>%
  html_elements("h1") %>%
  html_text()
```

```
## [1] "Lake Constance"
```

Web scraping with rvest

Extract all lower-level headlines:

```
wiki_html %>%
  html_elements(".mw-headline") %>% # All elements of class "mw-headline" (note the . indicating
  html_text()
```

```
##  [1] "Description"
##  [2] "History"
##  [3] "Name"
##  [4] "Key facts"
##  [5] "Historical maps"
##  [6] "Geography"
##  [7] "Divisions"
##  [8] "Emergence and future"
##  [9] "Tributaries"
## [10] "Outflows, evaporation, water extraction"
## [11] "Islands"
## [12] "Peninsulas"
## [13] "Shore"
## [14] "Climate"
## [15] "International borders"
## [16] "Floods"
```

Web scraping with rvest

Universität Konstanz

Extract article text without headlines as single paragraphs:

```
wiki_html %>%
  html_elements("#bodyContent") %>%
  html_elements("p") %>%
  html_text(trim = TRUE) # Trim leading and trailing whitespace
```

```
##   [1] ""
##   [2] "Lake Constance (German: Bodensee) refers to three bodies of water on the Rhine at the northern
##   [3] "The lake is situated where Germany, Switzerland, and Austria meet. Its shorelines lie in the Ge
##   [4] "The most populous towns on the Upper Lake are Constance (German: Konstanz), Friedrichshafen, Br
##   [5] "While in English and the Romance languages, the lake is named after the city of Constance, the
##   [6] "Lake Constance is the third largest freshwater European lake in surface area (and the second la
##   [7] "It is 63 km (39 mi) long, and, nearly 14 km (8.7 mi) at its widest point. I
##   [8] "The lake has two parts: the main east section, called Obersee or \"Upper Lake\", covers about 4
##   [9] "The connection between these two lakes is the Seerhein (lit.: \"Lake Rhine\"). Geographically,
##  [10] "The Lower Lake Constance is loosely divided into three sections around the Island of Reichenau:
##  [11] "The river water of the regulated Alpine Rhine flows into the lake in the southeast near Bregenz
##  [12] "The lake itself is an important drinking water source for southwestern Germany."
##  [13] "The culminating point of the lake's drainage basin is the Swiss peak Piz Russein of the Tödi ma
##  [14] "Car ferries link Romanshorn, Switzerland, to Friedrichshafen, and Konstanz to Meersburg, all ir
##  [15] "Lake Constance was formed by the Rhine Glacier during the ice age and is a zungenbecken lake. A
```

Web scraping with rvest

Extract article link targets in the article text:

```
wiki_html %>%
  html_elements("#bodyContent p > a") %>%
  html_attr("href")
```

```
##   [1] "/wiki/German_language"
##   [2] "/wiki/Body_of_water"
##   [3] "/wiki/Rhine"
##   [4] "/wiki/Alps"
##   [5] "/wiki/Upper_Lake_Constance"
##   [6] "/wiki/Lower_Lake_Constance"
##   [7] "/wiki/Seerhein"
##   [8] "/wiki/Alpine_Foreland"
##   [9] "/wiki/Rhine"
##  [10] "/wiki/Bavaria"
##  [11] "/wiki/Baden-W%C3%BCrttemberg"
##  [12] "/wiki/Canton_of_St._Gallen"
##  [13] "/wiki/Canton_of_Thurgau"
##  [14] "/wiki/Canton_of_Schaffhausen"
##  [15] "/wiki/Vorarlberg"
##  [16] "/wiki/Alpine_Rhine"
```

Web scraping with rvest

Extract "ten largest tributaries" table by Xpath:

```
wiki_html %>%
  html_elements(xpath = '//*[@id="mw-content-text"]/div[1]/table[2]') %>%
  html_table()
```

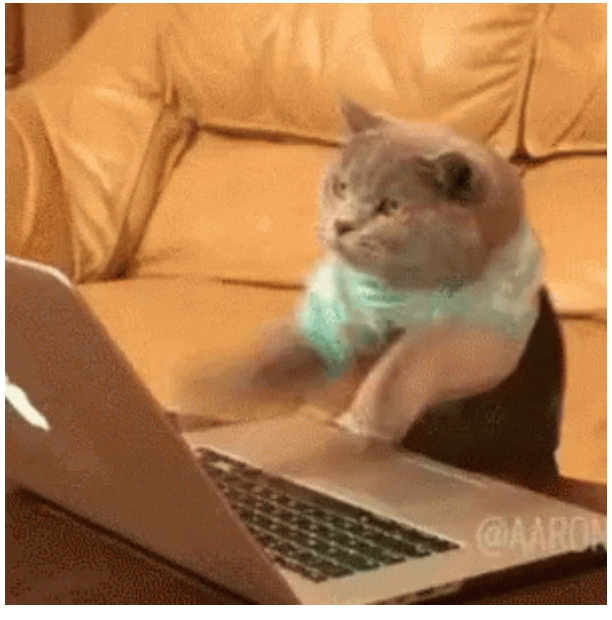
```
## [[1]]
## # A tibble: 12 x 5
##    River     `Average discharg~ `Dischargein %` `Catchment[km2]` `Catchmentin %`
##    <chr>     <chr>              <chr>                          <dbl> <chr>
##  1 Alpine R~ 233                61.1                            6.12 56.1
##  2 Bregenze~ 48                 12,6                          832    7.6
##  3 Argen     19                 5.3                           656    6.0
##  4 Old Rhin~ 12                 3.1                           360    3.3
##  5 Schussen  11                 2.9                           822    7.5
##  6 Dornbirn~ 7.0                1.8                           196    1.8
##  7 Leiblach  3,3                0.9                           105    1.0
##  8 Seefelde~ 3,2                0.8                           280    2,6
##  9 Rotach    2.0                0.5                           130    1.2
## 10 Stockach~ 1.6                0.4                           221    2.0
## 11 Sum of t~ 340                89.6                            9.72 89.2
## 12 Total in~ 381                100.0                          10.9  100.0
```

Web scraping with `rvest`

Universität Konstanz

Exercise 1: Web scraping: Try to obtain the headline, publication date, and article text (excluding lead and lower-level headlines) of the following Spiegel Online article: Failed Football Deal: Investors Wanted to Make €6.1 Billion with Super League

Bonus points: write a function that works with every SpOn article formatted in the same way.



Good practices

Universität Konstanz

As already discussed last time, systematic web scraping can run into some legal grey areas. Common good web scraping practices thus include:

- Respect site owner's terms: professional websites usually define a robots exclusion protocol in a file called `robots.txt` in the root of the web server that defines what may be scraped automatically and what not
- Scrape sparingly: Only extract and store the information you need, do not overload servers with thousands of requests per minute
- Introduce yourself: Define a point of contact in the user-agent string of your bot

The package `polite` simplifies the above practices by automatically reading out the `robots.txt` and adhering to the standards defined within. Main functions:

- `bow()` to a web server, introduce yourself and read out `robots.txt`
- `nod()` to update the current path on the same server (no need to bow multiple times to the same server)
- `scrape()` to actually scrape the current path (and optionally pass parameters to the current path)

Scraping with `polite`

Universität Konstanz

Let's give this a try:

```
library(polite)
wiki_session <- bow("https://en.wikipedia.org/") # Include a custom user agent string with the a
wiki_session
```

```
## <polite session> https://en.wikipedia.org/
##     User-agent: polite R package - https://github.com/dmi3kno/polite
##     robots.txt: 456 rules are defined for 33 bots
##    Crawl delay: 5 sec
##   The path is scrapable for this user-agent
```

Look's like we are allowed to scrape here.

Scraping with `polite`

Update path to specific article and scrape the article:

```
article <- nod(wiki_session, "wiki/Korean_fried_chicken") %>%
  scrape()
```

We can now use `rvest` functions to extract elements:

```
article %>%
  html_elements("h1") %>%
  html_text()
```

```
## [1] "Korean fried chicken"
```

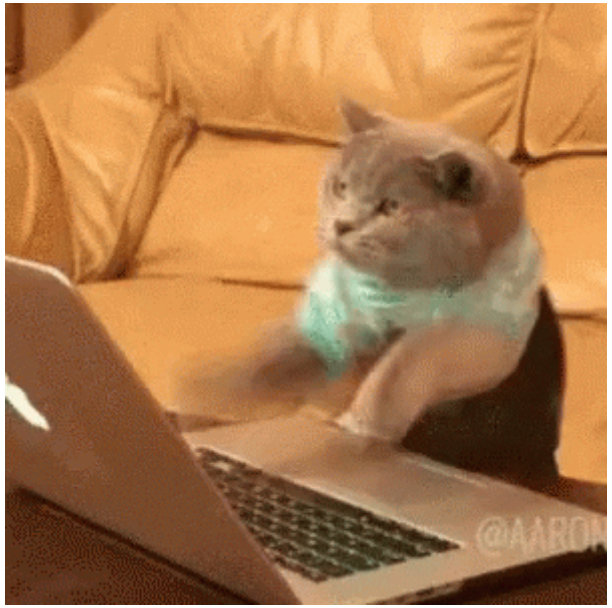
Scraping with `polite` and `rvest`

Universität Konstanz

Exercise 2: Scraping multiple articles

1. Using `polite` functions, create a session for the international portal of Spiegel Online: https://www.spiegel.de/international/
2. Get the links/paths to the three most recent articles
3. Using the `polite` principles, scrape the headline, date and article text of those three articles

Bonus points: update the function from Exercise 1 to follow those principles.



News APIs

News APIs

Universität Konstanz

Several news outlets provide their own content APIs, including:

- The Guardian
- New York Times

There are also overarching APIs dedicated to searching news stories, for example:

- MediaCloud
- News API

The same principles of last week's session apply.

Digression: Storing API keys

When working on your projects, you will probably receive more and more API keys. It is good practice to store those as environment variables in a global- or project-level `.Renviron` file.

Set new variables by `Sys.setenv()`:

```
Sys.setenv(TEST_API_KEY = "abc123456789")
```

Then retrieve the values with `Sys.getenv()`:

```
Sys.getenv("TEST_API_KEY")
```

```
## [1] "abc123456789"
```

This also means you can share code without accidentally exposing your secret API keys to the public.

MediaCloud

Universität Konstanz

MediaCloud is an open-source platform for media analysis, monitoring media sources in 100+ countries, with news stories scraped almost in real time.

The MediaCloud API (https://api.mediacloud.org/) is documented at https://github.com/mediacloud/backend/blob/master/doc/api_2_0_spec/api_2_0_spec.md. Rate limits are "1,000 API calls and 20,000 stories returned in any 7 day period".

The API offers lots of different endpoints. Some important endpoints for collecting news texts are:

- `api/v2/media/list/`: Search for news outlets by name, tag, etc.
- `api/v2/stories_public/list`: Search for news stories
- `api/v2/stories_public/word_matrix`: Retrieve word matrices for news stories

Using the MediaCloud API

Universität Konstanz

Let's try to obtain some Spiegel Online stories. We first need to know SpOn's MediaCloud id, so we may want so search the `api/v2/media/list/` endpoint. For all API calls, we authenticate by passing our API key to the parameter `key`.

```
library(httr)
mc_base_url <- "https://api.mediacloud.org/"
media_endpoint <- "api/v2/media/list"

res <- GET(mc_base_url,
           path = media_endpoint,
           query = list(
             name = "spiegel",
             key = Sys.getenv("MEDIACLOUD_API_KEY")
           ))
```

Using the MediaCloud API

Unpack the response:

```
res %>%
  content() %>%
  purrr::map_dfr(magrittr::extract, c("media_id", "name", "url"))
```

```
## # A tibble: 2 x 3
##   media_id name                url
##      <int> <chr>               <chr>
## 1    19831 Spiegel             http://www.spiegel.de
## 2    14771 Ik ben ulen spiegel http://de-te.livejournal.com
```

Look's like Spiegel's `media_id` is 19831.

Using the MediaCloud API

Universität Konstanz

Now we call the `api/v2/stories_public/list` endpoint for recent stories. Instead of through individual call parameters, media stories can be searched by passing a search string to the `q` parameter. Find more information here: https://mediacloud.org/support/query-guide/

For example, the query string "`media_id:19831+AND+text:medienstaatsvertrag` searches for all Spiegel stories containing the word "Medienstaatsvertrag". However, `httr`'s URL parser reformats certain characters (e.g., `:`, `+`):

```
stories_endpoint <- "api/v2/stories_public/list"
params = list(q = "media_id:19831+AND+text:medienstaatsvertrag",
              rows = 100)

stories_url <- parse_url(mc_base_url)
stories_url$path <- stories_endpoint
stories_url$query <- params
stories_url <- build_url(stories_url)
stories_url
```

```
## [1] "https://api.mediacloud.org/api/v2/stories_public/list?q=media_id%3A19831%2BAND%2Btext%3Amedienstaatsvertrag&rows=100"
```

Using the MediaCloud API

We can just replace those characters again:

```
stories_url <- stringr::str_replace_all(stories_url, c("%3A" = ":", "%2B" = "+"))
stories_url
```

```
## [1] "https://api.mediacloud.org/api/v2/stories_public/list?q=media_id:19831+AND+text:medienstaatsvertr
```

And are ready to call again:

```
stories_url <- paste(stories_url, "&key=", Sys.getenv("MEDIACLOUD_API_KEY"), sep = "")
res <- GET(stories_url)
```

Using the MediaCloud API

And unpack again:

```
res %>%
  content() %>%
  purrr::map_dfr(magrittr::extract, c("stories_id", "publish_date", "title", "url"))
```

```
## # A tibble: 10 x 4
##    stories_id publish_date   title                     url
##         <int> <chr>          <chr>                     <chr>
##  1 1427334496 2019-10-24 02~ Reformpläne: Jetzt kommt~ https://www.spiegel.de/n~
##  2 1620921456 2020-05-30 09~ Soziale Netzwerke: Trump~ https://www.spiegel.de/n~
##  3 1644266463 2020-06-25 02~ Wissenschaftler fordern ~ https://www.spiegel.de/n~
##  4 1776589639 2020-11-22 02~ Rundfunkgebühren: Reiner~ https://www.spiegel.de/p~
##  5 1781748899 2020-11-27 11~ News des Tages: Hotels u~ https://www.spiegel.de/p~
##  6 1783634326 2020-11-30 05~ Sachsen-Anhalt: Reiner H~ https://www.spiegel.de/p~
##  7 1784902678 2020-12-01 12~ Sachsen-Anhalt: CDU will~ https://www.spiegel.de/p~
##  8 1785923586 2020-12-02 13~ Sachsen-Anhalt: Friedric~ https://www.spiegel.de/p~
##  9 1786351446 2020-12-03 01~ Rundfunkbeitrag: Stephan~ https://www.spiegel.de/p~
## 10 1792633545 2020-12-09 06~ SPD: Rolf Mützenich krit~ https://www.spiegel.de/p~
```

Using the MediaCloud API

Universität Konstanz

Finally, let's obtain the word matrices for these stories:

```
wm_endpoint <- "api/v2/stories_public/word_matrix"
params = list(q = "media_id:19831+AND+text:medienstaatsvertrag",
              key = Sys.getenv("MEDIACLOUD_API_KEY"))

wm_url <- parse_url(mc_base_url)
wm_url$path <- wm_endpoint
wm_url$query <- params
wm_url <- build_url(wm_url) %>%
  stringr::str_replace_all(c("%3A" = ":", "%2B" = "+"))
```

Using the MediaCloud API

And call:

```
res <- GET(wm_url)
```

The result contains two lists, `word_list` and `word_matrix`:

```
wm <- content(res)
str(wm, max.level = 1)
```

```
## List of 2
##  $ word_list  :List of 1906
##  $ word_matrix:List of 10
```

From the MediaCloud API documentation:

- The word_matrix is a dictionary with the stories_id as the key and the word count dictionary of as the value. For each word count dictionary, the key is the word index of the word in the word_list and the value is the count of the word in that story.
- The word list is a list of lists. The overall list includes the stems in the order that is referenced by the word index in the word_matrix word count dictionary for each story. Each individual list member includes the stem counted and the most common full word used with that stem in the set.

Using the MediaCloud API

Unpacking depends on how you prefer handling nested lists. To get to a document-feature matrix in Tidytext style, we may first separate both lists:

```
word_list <- wm$word_list
word_matrix <- wm$word_matrix
```

Applying some rectangling functions, we can create a tibble with the word list:

```
word_list <- word_list %>%
    tibble::enframe(name = "word_counts_id", value = "word_forms") %>%
    tidyr::hoist(word_forms, stem = 1, full = 2) %>%
    dplyr::mutate(word_counts_id = word_counts_id - 1) # Because R starts to count at index 1
```

Using the MediaCloud API

Universität Konstanz

For each word contained in our stories, this gives us an ID, the word stem, and the most common full word associated with this stem:

```
word_list
```

```
## # A tibble: 1,906 x 3
##    word_counts_id stem                full
##             <dbl> <chr>               <chr>
##  1              0 inhalteanbiet       inhalteanbieter
##  2              1 ad                  ad
##  3              2 pflichten           pflichten
##  4              3 reform              reform
##  5              4 handvol             handvoll
##  6              5 missbrauch          missbrauch
##  7              6 gemeinsamen         gemeinsamen
##  8              7 nrw                 nrw
##  9              8 öffentlich-rechtlich öffentlich-rechtliche
## 10              9 gegebenheiten       gegebenheiten
## # ... with 1,896 more rows
```

Using the MediaCloud API

We rectangle the word matrix:

```
word_matrix <- word_matrix %>%
    tibble::enframe(name = "stories_id", value = "word_counts") %>%
    tidyr::unnest_longer(word_counts) %>%
    dplyr::mutate(word_counts_id = as.integer(word_counts_id))

word_matrix
```

```
## # A tibble: 2,497 x 3
##    stories_id word_counts word_counts_id
##    <chr>            <int>          <int>
##  1 1427334496           1              0
##  2 1427334496           1              1
##  3 1427334496           1             10
##  4 1427334496           1            100
##  5 1427334496           1            101
##  6 1427334496           1            102
##  7 1427334496           1            103
##  8 1427334496           1            104
##  9 1427334496           1            105
## 10 1427334496           1            106
```

Using the MediaCloud API

And finally join both tibbles:

```
tidy_matrix <- word_matrix %>%
    dplyr::left_join(word_list)
```

```
## Joining, by = "word_counts_id"
```

```
tidy_matrix
```

```
## # A tibble: 2,497 x 5
##    stories_id word_counts word_counts_id stem            full
##    <chr>            <int>          <dbl> <chr>           <chr>
##  1 1427334496           1              0 inhalteanbiet   inhalteanbieter
##  2 1427334496           1              1 ad              ad
##  3 1427334496           1             10 offenlegen      offenlegen
##  4 1427334496           1            100 antworten       antworten
##  5 1427334496           1            101 angeh           angehe
##  6 1427334496           1            102 pflicht         pflicht
##  7 1427334496           1            103 erfolg          erfolg
##  8 1427334496           1            104 spiele-stream   spiele-streamer
##  9 1427334496           1            105 milliardenschwer milliardenschwerer
```

Using the MediaCloud API

Universität Konstanz

Let's compare that to one of the original stories:

```
dplyr::filter(tidy_matrix, stories_id == 1427334496)
```

```
## # A tibble: 312 x 5
##    stories_id word_counts word_counts_id stem             full
##    <chr>            <int>          <dbl> <chr>            <chr>
##  1 1427334496           1              0 inhalteanbiet    inhalteanbieter
##  2 1427334496           1              1 ad               ad
##  3 1427334496           1             10 offenlegen       offenlegen
##  4 1427334496           1            100 antworten        antworten
##  5 1427334496           1            101 angeh            angehe
##  6 1427334496           1            102 pflicht          pflicht
##  7 1427334496           1            103 erfolg           erfolg
##  8 1427334496           1            104 spiele-stream    spiele-streamer
##  9 1427334496           1            105 milliardenschwer milliardenschwerer
## 10 1427334496           1            106 werben           werben
## # ... with 302 more rows
```

Using the MediaCloud API

For your convenience, here's a wrapper package that simplifies the steps of the last few slides:
https://github.com/joon-e/mediacloud

```
#install.packages("remotes")
remotes::install_github("joon-e/mediacloud")
```

This will let you search media, stories, and obtain word matrices:

```
library(mediacloud)
search_stories(title = "dogecoin", media_id = c(19831, 38697), after_date = "2021-05-01")
```

```
## # A tibble: 3 x 9
##   stories_id media_id publish_date        title      url        processed_stori~
##        <int>    <int> <dttm>              <chr>      <chr>                 <dbl>
## 1 1922328226    19831 2021-05-05 13:35:13 Dogecoin:~ https://w~      2328683297
## 2 1925893908    38697 2021-05-09 07:10:42 Dogecoin:~ https://w~      2331981923
## 3 1926994811    19831 2021-05-10 12:44:34 Elon Musk~ https://w~      2333054504
## # ... with 3 more variables: media_name <chr>, collect_date <dttm>,
## #   tags <named list>
```

News databases

News databases

Universität Konstanz

(Commercial) News databases aggregate content from a variety of news sources. The most relevant for news are:

- LexisNexis: You should be able to access *NexisUni* through the university library
- Dow Jones Factiva: Faculty access through Pollux FID

They provide the probably easiest way to obtain full texts from various news sources. However, text output formats are usually unstructured and thus require additional parsing. Furthmore, batch download can be a bit cumbersome (manual selection of texts, limited number of texts per download).

Parsing text in R

Universität Konstanz

Extracting text from a website just like we did before is a special case of parsing any kind of text into a structured format. However, regular text documents often do not provide anchors, tags or other structured elements we can use to extract the text we want and are thus often more complicated to parse. Some helpers:

- The `textreadr` package provides functions to load many different text formats into R, including several proprietary formats (e.g., `.docx`)
- The `stringr` package provides tidyverse-style functions for parsing and manipulating text data, for example pattern detection, matching and extraction.

When parsing text files, at some point you probably need a way to formally express how to search for specific patterns. That's what Regex (*reg*ular *ex*pressions) are for. Some good ressources:

- Learn, build, and test regex on RegExr
- Verbalize Regex with RVerbalExpressions
- Go full nerd with regex crosswords

Example: NexisUni with `LexisNexisTools`

Thankfully, parser packages exist for several databases. Let's import some text from NexisUni with `LexisNexisTools`.

```
install.packages("LexisNexisTools")
```

The easiest way to import multiple texts at once is to use the bulk download as single file (`.docx`) function on NexisUni. We can then import all texts at once using `lnt_read()`

```
library(LexisNexisTools)
texts <- lnt_read("nexis_files.docx")
```

```
## LexisNexisTools Version 0.3.4

## Creating LNToutput from 1 file...

##     ...files loaded [0.087 secs]

##     ...articles split [0.11 secs]

##     ...lengths extracted [0.12 secs]

##     ...headlines extracted [0.12 secs]
```

Example: NexisUni with LexisNexisTools

The result contains three dataframes, 1) meta information:

```
texts@meta
```

```
## # A tibble: 100 x 10
##       ID Source_File Newspaper Date       Length Section Author Edition Headline
##    <int> <chr>       <chr>     <date>     <chr>  <chr>   <chr>  <lgl>   <chr>
##  1     1 temp/nexis~ ZEIT-onl~ 2019-12-05 389 w~ Medien~ Johan~ NA      Ministe~
##  2     2 temp/nexis~ Newstex ~ NA         2233 ~ <NA>    Chris~ NA      Der neu~
##  3     3 temp/nexis~ taz, die~ 2020-10-30 763 w~ MEDIEN~ Peter~ NA      Bitte n~
##  4     4 temp/nexis~ taz, die~ 2006-11-09 386 w~ Nord A~ MARCO~ NA      Nord-Me~
##  5     5 temp/nexis~ Horizont  2019-10-30 513 w~ THEMA ~ Hein,~ NA      Regeln ~
##  6     6 temp/nexis~ dpa-AFX ~ 2020-10-28 235 w~ <NA>    <NA>   NA      Weg für~
##  7     7 temp/nexis~ Internet~ 2020-01-13 813 w~ MEINUN~ <NA>   NA      Kein sm~
##  8     8 temp/nexis~ Computer~ NA         1130 ~ SOCIAL~ [dal]  NA      MEDIENS~
##  9     9 temp/nexis~ dpa-AFX ~ 2020-04-27 542 w~ <NA>    <NA>   NA      ROUNDUP~
## 10    10 temp/nexis~ dpa-AFX ~ 2020-11-06 198 w~ <NA>    <NA>   NA      Regeln ~
## # ... with 90 more rows, and 1 more variable: Graphic <lgl>
```

Example: NexisUni with LexisNexisTools

The result contains three dataframes, 2) all articles:

```
texts@articles
```

```
## # A tibble: 100 x 2
##       ID Article
##    <int> <chr>
##  1     1 " Sean Gallup  BERLIN, GERMANY - MAY 28: In this photo illustration a ~
##  2     2 "Mar 04, 2020( Wilde Beuger Solmecke Lawyers: http://www.wbs-law.de De~
##  3     3 "Von Peter Weissenburger Der Medienstaatsvertrag regelt künftig die Re~
##  4     4 "Grundsätzlich sind alle dafür, doch der Teufel liegt im Detail. Weil ~
##  5     5 "Veranstaltung: Medientage München München Deutschland Wenn alles glat~
##  6     6 "SCHWERIN (dpa-AFX) - Der neue Medienstaatsvertrag in Deutschland mit ~
##  7     7 "Nun ist der Medienstaatsvertrag also beschlossen. Doch erst bei genau~
##  8     8 "Statt zu Hause nach Feierabend einen TV-Sender einzuschalten, schauen~
##  9     9 "(neu: im letzten Absatz - wegen Corona-Krise keine Sitzung, sondern U~
## 10    10 "MAINZ (dpa-AFX) - Der neue Medienstaatsvertrag in Deutschland mit Reg~
## # ... with 90 more rows
```

Example: NexisUni with `LexisNexisTools`

Universität Konstanz

The result contains three dataframes, 3) all paragraphs of the articles separately:

```
texts@paragraphs
```

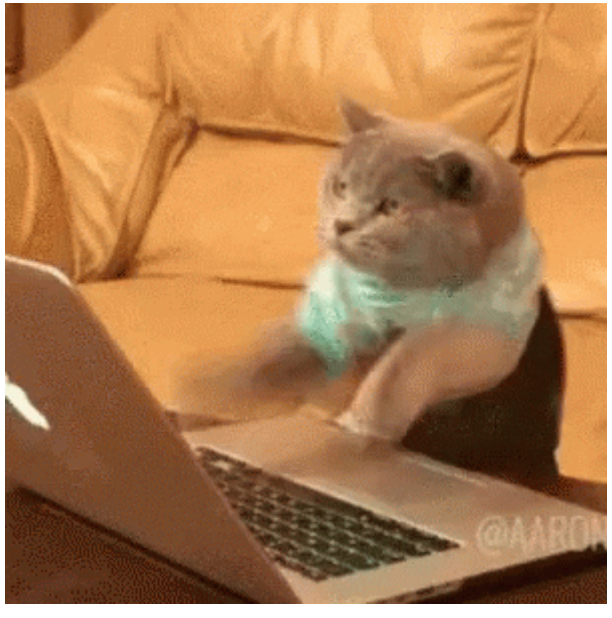
```
## # A tibble: 736 x 3
##    Art_ID Par_ID Paragraph
##     <int>  <int> <chr>
##  1      1      1 " Sean Gallup"
##  2      1      2 " BERLIN, GERMANY - MAY 28: In this photo illustration a young~
##  3      1      3 "Der seit 1991 geltende Rundfunkstaatsvertrag soll durch einen~
##  4      1      4 "Hintergrund des neuen Vertrags ist der digitale Wandel. Der S~
##  5      1      5 "Mit dem Beschluss tritt der Medienstaatsvertrag noch nicht in~
##  6      1      6 "In dem Medienstaatsvertrag geht es nicht um die Höhe des Rund~
##  7      2      7 "Mar 04, 2020( Wilde Beuger Solmecke Lawyers: http://www.wbs-l~
##  8      2      8 "Zustzlich sind auch Internet-Suchmaschinen, Streaming-Anbiete~
##  9      3      9 "Von Peter Weissenburger"
## 10      3     10 "Der Medienstaatsvertrag regelt künftig die Rechte und Pflicht~
## # ... with 726 more rows
```

Example: NexisUni with `LexisNexisTools`

Universität Konstanz

Exercise 3: NexisUni

Download German-language news articles about "Dogecoin" published during the last 7 days.



Exercise solutions

Exercise solutions

Universität Konstanz

Exercise 1:

```
scrape_spon <- function(url) {
  # Read URL
  article <- read_html(url)

  # Extract headline(s)
  hl <- article %>%
    html_elements("h2") %>%
    html_text(trim = TRUE)

  # Extract publication date
  date <- article %>%
    html_elements("time") %>%
    html_text()

  # Extract article body
  body <- article %>%
    html_elements(".word-wrap p") %>%
    html_text()

  return(list(headline = hl, date = date, body = body))
}
```

Exercise solutions

Universität Konstanz

Exercise 1:

Apply our new function:

```
article_url <- "https://www.spiegel.de/international/europe/investors-wanted-to-make-eur6-1-bill
res <- scrape_spon(article_url)
```

Exercise solutions

Universität Konstanz

Exercise 2:

First, let's introduce to the SpOn server:

```
spon_session <- bow("https://www.spiegel.de/international/")
```

Next, get the contents of the international portal homepage:

```
homepage <- spon_session %>%
  scrape()
```

Then, extract the first three links (or all article links and select only the first three):

```
article_links <- homepage %>%
  html_elements("article h2 a") %>%
  html_attr("href")

first_three <- article_links[1:3]
```

Exercise solutions

Let's update the function:

```
scrape_spon_new <- function(path, session) {
  # Update path
  article <- nod(session, path) %>%
    scrape()

  hl <- article %>%
    html_elements("h2") %>%
    html_text(trim = TRUE)

  date <- article %>%
    html_elements("time") %>%
    html_text()

  body <- article %>%
    html_elements(".word-wrap p") %>%
    html_text() %>%
    stringr::str_c(collapse = "\n") # Collapse article content to one string

  return(list(headline = hl, date = date, body = body))
}
```

Exercise solutions

We can now iterate over the article links, for example using `purrr`'s `map_dfr()` function to automatically generate a tibble with all article information:

```
purrr::map_dfr(first_three, scrape_spon_new, spon_session)
```

```
## # A tibble: 3 x 3
##   headline                                date        body
##   <chr>                                   <chr>       <chr>
## 1 "Interview with Afghanistan Presi~ 14.05.2021~ "This interview with Ghani too~
## 2 "Escalation in the Middle East\n\~ 14.05.2021~ "The streets of Lod smell like~
## 3 "Voices from Gaza\n\n\"No Place H~ 14.05.2021~ "On Tuesday evening of this we~
```

Exercise solutions

Universität Konstanz

Exercise 3:

On NexisUni, search for "Dogecoin" in news and set filters to Language = German and Timespan = Last 7 Days. Download in bulk and import with `lnt_read()` - done!

Thanks

Credits:

- Slides created with `xaringan`
- Title image by Digital Buggu / Pexels
- Coding cat gif by Memecandy/Giphy