L04 – rvest

3. Mai 2021

Contents

Das Paket rvest und ein erstes Beispiel

Mit dem Paket `rvest` lassen sich Datensätze aus Webseiten erstellen, indem ihr HTML-Code eingelesen wird.

HTML besteht aus geschachtelten Tags ggf mit HTML-Attribut `<tag_name attribut_name="attribut_value">Text und Kind-Elemente, dh weitere Tags</tag_name>`, zB wird ein HTML-Dokument mit `der Text wird <b>fett</b>` von einem Browser so angezeigt: der Text wird **fett**. Links stehen im Tag `<a>` mit dem Link-URL im Attribut `href`, zB ergibt `Link zu <a href="https://de.wikipedia.org/">Wikipedia</a>`: Link zu Wikipedia

`rvest`-Funktionen:

- `read_html()` lädt eine HTML-Webseite herunter und gibt sie als R-Objekt zurück.
- `html_nodes()` wählt bestimmte HTML-Tags anhand von CSS-Selektoren aus, siehe https://www.w3schools.com/cssref/css_selectors.asp
- `html_text()` gibt den Text eines HTML-Tags aus.
- `html_attr()` gibt den Wert eines HTML-Attributs eines HTML-Tags aus.
- `html_name()` gibt den Tag-Name aus.

Wir werden die erwähnten CSS-Selektoren zwar benutzen, ein tieferes Verständnis ist für die folgenden Aufgaben jedoch nicht nötig.

In folgendem Beispiel laden wir Daten von *The Lego Movie* von der Seite http://www.imdb.com/title/tt1490017/.

```
library(rvest)
library(tidyverse)
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter()         masks stats::filter()
## x readr::guess_encoding() masks rvest::guess_encoding()
## x dplyr::lag()            masks stats::lag()

# download website
lego_movie <- read_html("http://www.imdb.com/title/tt1490017/")

lego_movie %>%
```

```
  # find HTML-element containing the rating (between 1 and 10) of the movie
  html_nodes("strong span") %>%
  # get the text of that element
  html_text() %>%
  # convert the text to type double
  as.double()
## [1] 7.7

lego_movie %>%
  # for every cast member, there is an image
  html_nodes("#titleCast .primary_photo img") %>%
  # in the image-Tag, the cast member name is in the attribute named alt
  html_attr("alt")
##  [1] "Will Arnett"     "Elizabeth Banks" "Craig Berry"     "Alison Brie"
##  [5] "David Burrows"   "Anthony Daniels" "Charlie Day"     "Amanda Farinos"
##  [9] "Keith Ferguson"  "Will Ferrell"    "Will Forte"      "Dave Franco"
## [13] "Morgan Freeman"  "Todd Hansen"     "Jonah Hill"

lego_movie %>%
  html_nodes(".poster img") %>%
  # get URL to image of poster for movie
  html_attr("src") %>%
  # view image
  magick::image_read()
```

1 Wikipedia-Spiel

Wir starten bei einer beliebigen Wikipedia-Seite. Auf der aktuellen Seite klicken wir auf die erste verlinkte Wikipedia-Seite. Dies wird solange wiederholt, bis wir eine Seite ein zweites Mal aufrufen. Dann ist das Spiel beendet.

Vervollständige den Code unten zum Spielen des Wikipedia-Spiels. Die Ausgabe der Funktion `run_game()` ist ein `character`-Vektor mit den Namen der aufgerufenen Artikel. Artikel, wie in der Ein- und Ausgabe der Funktion `first_linked_article()` sind der hintere Teil der vollständigen URL, zB `"R_(Programmiersprache)"` für die Seite `https://de.wikipedia.org/wiki/R_(Programmiersprache)`.

```
library(rvest)
library(tidyverse)
```

```r
first_linked_article <- function(article) {
  if (is.na(article)) return(NA_character_)
  html <- read_html(str_c("https://de.wikipedia.org/wiki/", article))
  node_a <- html_node(html, '.mw-parser-output > p > a[href^="/wiki/"]')
  if (length(node_a) == 0) {
    node_a <- html_node(html, ".mw-parser-output > ul > li > a")
  }
  node_a
  # TODO: extract last part of link
  # node_a is a html_node of tag <a>. It has an attribute href. Example:
  # > first_linked_article("R_(Programmiersprache)")
  # {html_node}
  # <a href="/wiki/Freie_Software" title="Freie Software">
  # First call html_attr() on node_a to get the link-text (href) of the node.
  # The link-text always starts with "/wiki/...".
  # Remove "/wiki/" from the link-text by calling a stingr function.
  # Return the resulting string, eg "Freie_Software"
}

first_linked_article("R_(Programmiersprache)")
# should result in "Freie_Software"

run_game <- function(start_article = "Spezial:Zuf%C3%A4llige_Seite") {
  # TODO: Call first_linked_article() on start_article.
  # Then call first_linked_article() on the result of the former call.
  # Repeat until the article returned by first_linked_article() has been
  # returned before
}

articles <- run_game()
```

Führe das Spiel 10-mal durch und gib an, auf welche Artikel das Spiel jeweils endet.

2 Mathematiker

Wir öffnen https://de.wikipedia.org/wiki/Liste_bedeutender_Mathematiker#20._Jahrhundert mit Chrome oder Firefox. Wir wollen für jeden der 20 Mathematiker:innen im Abschnitt 20. Jahrhundert (von Hilbert bis Perelman) die zugehörige Wikipedia-Seite öffnen und Informationen extrahieren.

Die Namen der Mathematiker:innen sind Links zu ihren eigenen Wikipedia-Seiten. Um die Links automatisch herauszufiltern, suchen wir ein Muster in den sogenannten **CSS-Selektoren** der zugehörigen `<a>`-HTML-Tags. Dazu klicken wir rechts auf den verlinkten Namen *David Hilbert* und wählen dann "Inspect", "Inspect Element", "Element untersuchen" oder "Untersuchen" aus (je nach Browser und Sprache). Es öffnet sich ein neuer Bereich im Browser der unter anderem `<a href="/wiki/David_Hilbert" title="David Hilbert">David Hilbert</a>` anzeigt. Wir klicken wieder rechts auf diesen Eintrag und wählen "Copy" → "Copy Selector" / "CSS Selector". Damit wurde der CSS-Selektor des HTML-Elements mit dem Link zu Hilbert's Wikipedia in die Zwischenablage kopiert. Diesen fügen wir in einem R-Skript ein und weisen ihn einer String-Variable `sel_hilbert` zu. Einer der beiden folgen den Texte sollte erscheinen.

```
.mw-parser-output > table:nth-child(22) > tbody:nth-child(1) >
  tr:nth-child(2) > td:nth-child(2) > a:nth-child(1)
```

oder:

```
#mw-content-text > div > table:nth-child(22) > tbody >
  tr:nth-child(2) > td:nth-child(2) > a
```

Führen wir den folgenden Code aus, sollte `"/wiki/David_Hilbert"` ausgegeben werden. Der vollständige URL lautet `https://de.wikipedia.org/wiki/David_Hilbert`.

```
library(tidyverse)
library(rvest)

sel_hilbert <- "TODO"
read_html("https://de.wikipedia.org/wiki/Liste_bedeutender_Mathematiker") %>%
  html_node(sel_hilbert) -> # use CSS selector to find specific element
  node_a_hilbert
node_a_hilbert # href is the url we will be sent to by clicking on this element in our browser
node_a_hilbert %>% # extract href-attribute of <a>-Element
  html_attr("href")
```

Eine Referenz zu CSS-Selektoren ist unter https://www.w3schools.com/cssref/css_selectors.asp zu finden. Wir müssen zu diesem Zeitpunkt die CSS-Selektoren nicht wirklich verstehen. Es genügt, wenn wir darin Muster erkennen. Um uns das zu ermöglichen, sehen wir uns nun noch CSS-Selektoren für zB Emmy Noether und Grigori Perelman an.

Es sollte nun möglich sein, die CSS-Selektoren für alle anderen Einträge (in der Tabelle *20. Jahrhundert*) zu erraten.

Wir erzeugen einen `character`-Vektor `selectors` der Länge 20 mit den entsprechenden CSS-Selektoren als Einträge. Dann wenden wir auf jeden Eintrag die Funktion `extract_href()` an, welche den jeweiligen `href`-Eintrag ausgibt. Das Ergebnis speichern wir im `character`-Vektor `link_part2`. Diesen fügen wir zum `character`-Vektor `link` so zusammen, dass dort nun die vollständigen Links stehen.

Hinweis: `sapply(x, f, a)` führt `f(x[i], a)` für `i` aus `1:length(x)` aus. Außerdem sind evtl `str_glue()` und `str_c()` nützlich.

```
library(tidyverse)
library(rvest)

math_list_html <- read_html(
  "https://de.wikipedia.org/wiki/Liste_bedeutender_Mathematiker")

extract_href <- function(sel, html) {
  html %>%
    html_node(sel) %>%
    html_attr("href")
}

selectors <- # TODO
link_part2 <- # TODO
link <- # TODO
```

```
link[1:3]
## [1] "https://de.wikipedia.org/wiki/David_Hilbert"
## [2] "https://de.wikipedia.org/wiki/Hermann_Minkowski"
## [3] "https://de.wikipedia.org/wiki/Felix_Hausdorff"
```

Wir extrahieren nun mit `link` und der Funktion `first_content_p_text()` den ersten Abschnitt der jeweiligen Wikipedia-Seiten in den `character`-Vektor `text`. In der Funktion ist der CSS-Selektor `.mw-parser-output > p` vorgegeben. Diesen hätten wir mit Hilfe von https://www.w3schools.com/cssref/css_selectors.asp und dem Vorgehen zum Finden der CSS-Selektoren für die Links oben auch selbst herausfinden können.

```r
first_content_p_text <- function(url) {
  read_html(url) %>%
    html_node(".mw-parser-output > p")  %>% # first <p>-child of element with class="mw-parser-output"
    html_text() # text content between <p> and </p>
}
text <- # TODO
```

Den Text können wir zB mit `View(text)` in RStudio anzeigen.

Als nächstes extrahieren wir folgende Informationen mittels RegEx aus dem Text und speichern sie in einem Tibble `data`.

- `name`: vollständiger Name, Typ `character`
- `birth_day`, `birth_month`, `birth_year`: Geburtstag, -monat, -jahr, Typ jeweils `integer`
- `death_day`, `death_month`, `death_year`: Todestag, -monat, -jahr falls verstorben, sonst `NA`, Typ jeweils `integer`
- `birth_place`, `death_place`: Geburts-, Todesort (falls verstorben), Typ `character`

Hierzu müssen wir in `text` nachsehen, ob wir Muster erkennen können, wo diese Informationen stehen, zB immer am Anfang des Textes, nach bestimmten Symbolen, usw. Dann extrahieren wir mit regulären Ausdrücken diese Informationen.

Bei den Datumsangaben merkt man schnell, dass es zwar eine Regelmäßigkeit gibt, wo diese Daten wie stehen. Es gibt aber evtl auch Ausnahmen von der Regel. Um mit diesen Ausnahmen umgehen zu können, kann entweder der Reguläre Ausdruck entsprechend vergrößert werden oder ein Vorverarbeitungsschritt, zB mit `str_replace()`, eingefügt werden.

Nützliche Funktionen: `str_extract()`, `str_trim()`, `str_replace()`, `str_match()`, `str_c()`, `as.integer()`.

```r
# name:
text %>%
  # TODO ->
  names

months <- c("Januar", "Februar", "März", "April", "Mai", "Juni", "Juli",
            "August", "September", "Oktober", "November", "Dezember")
months_regex <- # TODO # create a regex to fit any month name from months;

# birth info
text %>%
  # TODO ->
  birth

# death info
text %>%
  # TODO ->
  death

# for converting month name to month number
month_lookup <- 1:12
names(month_lookup) <- months

# data
data <- tibble(
  name = names,
  birth_day = #TODO,
  birth_month = ,
```

```
  birth_year = ,
  birth_place = ,
  death_day = ,
  death_month = ,
  death_year = ,
  death_place =
)
```

```
str_length(names) # for comparison with solution
##  [1] 13 17 15 19 20 19 23 19 13 32 16 20 10 17 20 10 17 22 21 30
head(data, 3)
## # A tibble: 3 x 9
##   name        birth_day birth_month birth_year birth_place death_day death_month
##   <chr>           <int>       <int>      <int> <chr>           <int>       <int>
## 1 David Hilb~        23           1       1862 Königsberg         14           2
## 2 Hermann Mi~        22           6       1864 Aleksotas          12           1
## 3 Felix Haus~         8          11       1868 Breslau            26           1
## # ... with 2 more variables: death_year <int>, death_place <chr>
```

Die fertige Tabelle können wir zB wieder mit `View(data)` betrachten.

Zum Schluss visualisieren wir die Lebenszeiten der Mathematiker:innen mit `ggplot2`.

Wir nutzen `geom_segment()` um eine Linie von `(x,y)` nach `(xend, yend)` zu zeichnen.

```
data %>%
  ggplot(aes(
      x = birth_year,
      xend = replace_na(death_year, 2025),
      y = name,
      yend = name
    )) +
    geom_segment(size=5) + # Linien Breite 5
    labs(x=NULL, y=NULL, color="alive")
```

Um die Anordnung zu ändern, ersetzen wir `name` in den Argumenten `y` und `yend` durch `reorder(name, birth_year)`.

Zuletzt wollen wir die Balken unterschiedlich einfärben (`color`), nämlich in Abhängigkeit davon, ob die Person noch am Leben ist. Nutze `is.na()`.

3 Bonus: IMDb Top 250

Wir laden Daten der IMDb Top 250 Filme von der Website herunter und analysieren sie.

```
library(tidyverse)
library(rvest)
library(lubridate)

top250_html <- read_html("https://www.imdb.com/chart/top/?ref_=nv_mv_250")

extract_href <- function(sel, html) {
  html %>%
    html_node(sel) %>%
    html_attr("href")
}
```

```
selectors <- # TODO: find CSS-selectors to movie pages, like in previous exercise. IMPORTANT: First, st

link_part2 <- # TODO: like in previous exercise
link <- # TODO: like in previous exercise

# CSS-selectors for different pices of information
info_selctors <- c(
  title = ".title_wrapper > h1:nth-child(1)",
  year = "#titleYear > a:nth-child(1)",
  time = ".subtext > time:nth-child(2)",
  genre = ".subtext > a:nth-child(4)",
  release = ".subtext > [title=\"See more release dates\"]",
  rating = ".ratingValue > strong:nth-child(1) > span:nth-child(1)",
  ratings = "span.small",
  director = "div.credit_summary_item:nth-child(2) > a:nth-child(2)")

# extract one info element with selector sel from object html
extract_info <- function(sel, html) {
  html %>%
    html_node(sel) %>%
    html_node(xpath="./text()") %>%
    html_text() %>%
    str_squish()
}

extract_infos <- function(url) {
  html <- read_html(url)
  # TODO use extract_info to get all infos with info_selectors form object html
}

# This may take several minutes for Top250
infos <- # TODO use extract_infos on all elements of link
# infos should be a character matrix

# convert matrix to tibble
data_raw <- as_tibble(t(infos))

# function for extracting duration form string
str2dur <- function(s) {
  s %>%
    str_extract("\\b[0-9]+h") %>%
    str_sub(end=-2) %>%
    as.numeric() ->
    h
  s %>%
    str_extract("\\b[0-9]+min") %>%
    str_sub(end=-4) %>%
    as.numeric() ->
    mi
  is.na(h) <- 0
  is.na(mi) <- 0
  dhours(h) + dminutes(mi)
}
```

```r
# convert from charcter entries to correct type
data_raw %>%
  mutate(year = as.integer(year),
         rating = as.double(rating),
         ratings = as.integer(str_replace_all(ratings, ",", "")),
         release = dmy(str_extract(release, "[^\\(]*")),
         time = str2dur(time),
         minutes = as.numeric(time, "minutes")) ->
  data

# top directors
data %>%
  group_by(director) %>%
  summarise(
    n = n(),
    mean_rating = mean(rating),
    sd_rating = sd(rating),
    mean_dur = mean(minutes, na.rm=T),
    year_from = min(year),
    year_to = max(year)) %>%
  arrange(desc(n)) ->
  top
View(top)

# some potenitally interesting plots
ggplot(data, aes(ratings, rating, color = genre)) + geom_point()
ggplot(data, aes(year, minutes, color = genre)) + geom_point()
ggplot(data, aes(year, rating, color = genre)) + geom_point()
ggplot(data, aes(year, fill = genre)) + geom_histogram(binwidth=10)
```