Workshop: Data Wrangling of Web Data in R

Simon Ress | Ruhr-Universität Bochum

October 18, 2021

Content

1. Setup
2. Functions
3. Iteration
    - Loops
    - Apply-Family
    - Purr-package
4. Dplyr - Gramma of Data Manipulation
5. Helpful Sources

Setup

Target

Meta information

- Finanzausschuss
- Ausschüsse der 19. Wahlperiode (2017-2021)
- Öffentliche Anhörungen

URL: https://www.bundestag.de/webarchiv/Ausschuesse/ausschuesse19/a07/Anhoerungen

Unit information

- Committees

URL: Needs to be scraped from main page

Configurate & Start Selenium/Browser

```
library(RSelenium)
library(rvest) #for read_html(), html_elements()...
#Free all ports
  system("taskkill /im java.exe /f", intern=FALSE,
   ↪  ignore.stdout=FALSE)
#Start a selenium & Assign client to an R-object
  rD <- rsDriver(port = 4561L, browser = "firefox")
  remDr <- rD[["client"]]
```

Functions

Overview

- Functions are **blocks of codes** which can be executed repeatedly by calling them
- **Parameters** (data) can be passed into them, which are used by the code inside
- **Data can be returned** from a function

Syntax:

```
function_name <- function(arg_1, arg_2, ...) {
     Function body
}
```

Function Components

The four parts of a function are:

- **Function Name:** This is the actual name of the function. It is stored in R environment as an object with this name.
- **Arguments (*optional*):** An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments *can* have default values.
- **Function Body:** The function body contains a collection of statements that defines what the function does.
- **Return Value:** The return value of a function is the last expression in the function body to be evaluated.

Examplary Function

```
square <- function(value = 1, factor = 1) {
     return(value^factor)
}
square() #use defaut args
```

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

```
square(2,3) #use args by position
```

```
## [1] 8
```

```
square(factor=2, value=5) #use args by name
```

```
## [1] 25
```

Use Case: define savepage()

```
#Load url & return content as r-object
  savepage <- function(url){
    #Navigate to starting page
      remDr$navigate(url)
    #Wait until page is loaded
      Sys.sleep(abs(rnorm(1, 2, 1)))
    #Save content to an R-object
      remDr$getPageSource(header = TRUE)[[1]] %>%
        read_html() %>%
        return()
  }
```

Note: [[1]] behinde getPageSource() unlist the output -> makes it searchable

Usage of savepage()

```
#navigate to url & save content as r-object
page <- savepage("https://www.bundestag.de/
 ↪  webarchiv/Ausschuesse/ausschuesse19/a07/
 ↪  Anhoerungen")
page
```

```
## {html_document}
## <html xml:lang="de" dir="ltr" class="detection-firefox"
## [1] <head>\n<meta http-equiv="Content-Type" content="te
## [2] <body class="bt-archived-page">\n  <div class="bt-a
```

Iteration

Overview

- Often specific tasts needs to be executed multiple times
- Iteration can be performed using loops or apply-functions

Example:

```
#Extract urls of all indiv. meetings
      urls <- html_elements(page, xpath =
 ↪  "/html//ul/li/a") %>%
        html_attr("href") %>%
        paste0("https://www.bundestag.de",.)
#Save content of pages
  meeting1 <- savepage(urls[1])
  meeting2 <- savepage(urls[2])
  meeting3 <- savepage(urls[3])
  ...
```

Loops

...

For-Loop

- A for loop is used for iterating over a sequence:
- With the break statement, we can stop the loop before it has looped through all the items:
- With the next statement, we can skip an iteration without terminating the loop:

```
for (x in names(iris[1:4])) {
  print(mean(iris[,x]))
}
```

```
## [1] 5.843333
## [1] 3.057333
## [1] 3.758
## [1] 1.199333
```

for-loop: break

Breaking the loop at certain conditions

```
for (x in cars$dist) {
  if (x > 20)  break
  print(x)
}
```

```
## [1] 2
## [1] 10
## [1] 4
```

for-loop: next

Skip the code below and start over at certain conditions

```
fruits <- list("apple", "banana", "cherry")

for (x in fruits) {
  if (x == "banana") next
  print(x)
}
```

```
## [1] "apple"
## [1] "cherry"
```

Use Case: loop over urls

- Create an empty list
- save content of page pages as list within first list

```
meetings <- list()
for (i in 1:length(urls)) {
meetings[i] <- savepage(urls[i]) %>% list()
}
```

while-loop

- Execute a set of statements as long as a condition is TRUE
- *break* statement stops the loop (even if while-condition=TRUE):
- *next* statement skips an iteration without terminating the loop:

```
i <- 0
while (i < 3) {
  i <- i + 1
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

while-loop: break

Breaking the loop at certain conditions

```
i <- 0
while (i < 20) {
  i <- i + 1
  if (i == 5) break
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

while-loop: next

Skip the code below and start over at certain conditions

```
i <- 0
while (i < 10) {
  i <- i + 1
  if (i %% 2) next
  print(i)
}
```

```
## [1] 2
## [1] 4
## [1] 6
## [1] 8
## [1] 10
```

Use Case: collect urls of all meetings

```
page <-
 ↪  savepage("https://www.bundestag.de/webarchiv/Ausschuesse/ausschuess
button <- 1
while (length(button)>0) {
  page <- remDr$getPageSource(header = TRUE)[[1]] %>%
          read_html()
  urls <- html_elements(page, xpath =
 ↪  "/html/body/main/section/div[2]/div/div/div/div/div/ul/li/a")
 ↪  %>%
    html_attr("href") %>%
    paste0("https://www.bundestag.de",.)

  webElem <- remDr$findElement("css", "body")
  webElem$sendKeysToElement(list(key = "end"))

  # find the element
  button <-
 ↪  remDr$findElements("/html/body/main/section/div[2]/div/div/div/div/
 ↪  = "xpath")
  if(length(button)>0) {
    button[[1]]$clickElement()
  }

}
```

Apply-Family

- The apply in R function can be feed with many functions to perform redundant application on a collection of object (data frame, list, vector, etc.).
- The purpose of apply() is primarily to avoid explicit uses of loop constructs.
- Any function can be passed into

Main apply functions

Function	Arguments	Objective	Input	Output
apply	apply(x, MARGIN, FUN)	Apply a function to the rows or columns or both	Data frame or matrix	vector, list, array
lapply (list)	lapply(X, FUN)	Apply a function to all the elements of the input	List, vector or data frame	list
sapply (simple)	sapply(X, FUN)	Apply a function to all the elements of the input	List, vector or data frame	vector or matrix
tapply (tagged)	tapply(X, grouping, FUN)	Apply a function for each factor variable in an vector	Vector	matrix or array

apply()-usage

apply(x, MARGIN, FUN)

```
m1 <- matrix(C<-(1:10),nrow=4, ncol=4)
apply(m1, 1, sum) #1=by row
```

```
## [1] 18 22 16 20
```

```
apply(m1, 2, sum) #2=by column
```

```
## [1] 10 26 22 18
```

lapply()-usage

lapply(X, FUN)

```
lapply(cars, mean)
```

```
## $speed
## [1] 15.4
##
## $dist
## [1] 42.98
```

sapply()-usage

sapply(X, FUN)

```
sapply(1:4, print)
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

```
## [1] 1 2 3 4
```

tapply()-usage

tapply(X, grouping, FUN)

```
tapply(iris$Sepal.Length , list(iris$Species), mean)
```

```
##     setosa versicolor  virginica
##      5.006      5.936      6.588
```

Similar to tapply() is aggregate() which returns a data frame

```
aggregate(iris$Sepal.Length , list(iris$Species), mean)
```

```
##      Group.1     x
## 1     setosa 5.006
## 2 versicolor 5.936
## 3  virginica 6.588
```

Overview

- "purrr enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors."
- We focus on the map()-functions:
  - transform input by applying a function to each element of a list or atomic vector
  - returning an object of the same length as the input

map()-usage

Syntax: *map(.x, .f, ...)*

```
if(!require("purrr")) install.packages("purrr")
  library(purrr) # for fill()
mtcars %>%
  split(.$cyl) %>% # from base R
  map(~ lm(mpg ~ wt, data = .)) %>%
  map(summary) %>%
  map_dbl("r.squared")
```

```
##         4         6         8
## 0.5086326 0.4645102 0.4229655
```

Versions of map()-function

- map_**dbl**(.x, .f, ...)
  - Return a double vector.
  - map_dbl(x, mean)
- map_**int**(.x, .f, ...)
  - Return an integer vector.
  - map_int(x, length)
- map_**chr**(.x, .f, ...)
  - Return a character vector.
  - map_chr(l1, paste, collapse = "")
- map_**lgl**(.x, .f, ...)
  - Return a logical vector.
  - map_lgl(x, is.integer)

Use Cases: extract dates of meetings

```
page <-
 ↪  savepage("https://www.bundestag.de/webarchiv/Ausschuess
date <- html_elements(page,xpath =
 ↪  "/html/body/main/section/div[2]/div/div/div/div/div[@c
 ↪  'bt-listenteaser']") %>%
  map(function(x)
    rep(html_element(x,xpath = "./h4") %>%
          html_text(),
        length(html_elements(x,xpath = ".//a"))))
date[3]
```

```
## [[1]]
## [1] "17. Mai 2021" "17. Mai 2021"
```

Dplyr - Gramma of Data Manipulation

Overview

Pipe-Operator

filter()

summarise()

across

group_by()

ungroup()

rowwise()

mutate()

distinct

slice

select

Helpful Sources

Helpful Sources

- purr: Overview
- purr: References
- purr: Cheat Sheet
- dplyr: Cheat Sheet