An introduction to web scraping methods

Ken Van Loon
Statistics Belgium

UN GWG on Big Data for Official Statistics
Training workshop on scanner and on-line data
6-7 November 2017
Bogota, Colombia
Background

- Me
  - Statistician?
  - Mostly working on price statistics (consumer price indices/residential property indices)
  - Methodological issues (incl. scanner data and web scraping)

- Web scraping at Statistics Belgium
  - We have around 60 scripts running (some implemented others in test/research phase)
  - Currently we scrape data for the following segments:
    - Clothing
    - Footwear
    - Hotels
    - Airfares
    - Train tickets
    - Second-hand cars
    - Department stores
    - Books
    - DVD & Blu-ray
    - Video games
    - Consumer electronics
    - Student rooms
    - Supermarkets
    - ...
Contents

- What is web scraping?
- HTML – CSS Selectors
- SelectorGadget
- Web scraping in R
  - Rvest
  - Scrape functions
  - RSelenium
- Experimental indices
- Monitoring tools
What is web scraping?

Web scraping focuses on the transformation of unstructured data on the web, typically in HTML format, into structured data that can be stored and analyzed in a central local database or spreadsheet. (wikipedia 😊)

A technique to collect (scrape) data from the web automatically.

Implement web scraping:

- Programming skills
- Data collection
- Data processing
Webpages consist of HTML code/tags:

```html
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>
<a href="http://www.google.com">This is a link</a>

</body>
</html>
```

Knowing HTML is not really necessary for web scraping, but will make life easier!
How to select the specific information on an HTML-page?

- Xpath – XML Path Language (query language for XML)
- **CSS selectors** – used to select elements you want to style
  
  focus here on CSS selectors (personal opinion: more readable then Xpath)

**Cascading Style Sheets (CSS)** are used to style websites:

with CSS

without CSS
How to select the specific information on an HTML-page?

Tag selection in Chrome:

Right click:
Inspect screen
(developer tools)

Right click:
Copy
Copy selector

Also using ctrl + f:
Just search for the tag
How to select the specific information on an HTML-page?

- Using Chrome is quite tedious

- Would be much easier to have a point-and-click interface to select what you want
  - Luckily someone developed this 😊
    (Remember: you can always verify what you select with the CSS selector using Chrome developer tools)

- But before using the point-and-click interface we’ll learn a bit how CSS work... by playing a game
CSS selectors

CSS Diner: [https://flukeout.github.io/](https://flukeout.github.io/)

```html
<div class="table">
  <plate/>
  <plate/>
</div>
```

CSS selector “plate”: selects all plate elements

```html
<div class="table">
  <bento/>
  <plate/>
  <bento/>
</div>
```

CSS selector “bento”: selects all bento tags
CSS selector “#fancy”: selects each tag with id=“fancy”.
“#” indicates the “id=”-tag

CSS selector “.small”: selects each tag with class=“small”.
“.” indicates the “class=“-tag
CSS selector “plate apple”: selects the apple tag within the plate tag

```
<div class="table">
  <bento/>
  <plate>
    <apple/>
  </plate>
</div>
```

CSS selector “#fancy pickle”: selects the tag pickle within the plate tag with id=fancy

```
<div class="table">
  <bento>
    <orange/>
  </bento>
  <bento/>
  <plate id="fancy">
    <pickle/>
  </plate>
  <plate/>
  <plate>
    <pickle/>
  </plate>
</div>
```
CSS selector “orange.small”: selects the orange with class=small
CSS selector “plate, bento”: selects all plate and bento tags
CSS selector “plate *”: selects everything which includes plate (incl. subtags)
CSS selector “plate + apple”: selects all apple tags directly following a plate tag, only the first apple tag after a plate tag is selected

“ + “ selects all tags **directly** (= first tag) following a specified tag

Note: with “ + “ only the first tag is selected
CSS selector “bento ~ pickle”: selects all matching pickle tags after the bento tag

“~” selects all matching tags at the same level after the first specified tag

Note: with “~” all tags at the same level are selected, not only the first one (cfr. +)
CSS selector “plate > apple”: selects all apple tags which follow a plate tag at another level

“ > ” selects all direct ‘children’ of an element
### Overview CSS selectors

<table>
<thead>
<tr>
<th>#</th>
<th>Selector</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>element</code></td>
<td><code>a</code></td>
<td>Selects all “a” tags</td>
</tr>
<tr>
<td>2</td>
<td><code>.class</code></td>
<td><code>.price</code></td>
<td>Selects all elements with the class=&quot;price&quot;</td>
</tr>
<tr>
<td>3</td>
<td><code>#id</code></td>
<td><code>#content</code></td>
<td>Selects all elements, in theory only one, with the id=&quot;content&quot;</td>
</tr>
<tr>
<td>4</td>
<td><code>element element</code></td>
<td><code>div a</code></td>
<td>Selects all “a” tags inside all “div” elements</td>
</tr>
<tr>
<td>5.a</td>
<td><code>.class element</code></td>
<td><code>.price a</code></td>
<td>Selects all “a” tags inside all elements with the class=&quot;price&quot;</td>
</tr>
<tr>
<td>5.b</td>
<td><code>#id element</code></td>
<td><code>#content a</code></td>
<td>Selects all “a” tags inside all elements with the id=&quot;content&quot;</td>
</tr>
<tr>
<td>6</td>
<td><code>element.class</code></td>
<td><code>div.price</code></td>
<td>Selects all “div” tags with the class=&quot;price&quot;</td>
</tr>
<tr>
<td>7</td>
<td><code>element, element</code></td>
<td><code>div, p</code></td>
<td>Selects all “div” tags and all “p” tags</td>
</tr>
<tr>
<td>8</td>
<td><code>element *</code></td>
<td><code>div *</code></td>
<td>Selects all elements within the “div” element</td>
</tr>
<tr>
<td>9</td>
<td><code>element+element</code></td>
<td><code>div+p</code></td>
<td>Selects “p” elements that follow directly after the “div” element (on the same level)</td>
</tr>
<tr>
<td>10</td>
<td><code>element~element</code></td>
<td><code>div~p</code></td>
<td>Selects all “p” elements that follow after the “div” element (on the same level)</td>
</tr>
<tr>
<td>11</td>
<td><code>element&gt;element</code></td>
<td><code>div&gt;p</code></td>
<td>Selects all “p” elements that are direct children of the “div” element</td>
</tr>
<tr>
<td>12</td>
<td><code>[attribute=&quot;value&quot;]</code></td>
<td><code>[size=&quot;small&quot;]</code></td>
<td>Selects all elements with size=&quot;small&quot;</td>
</tr>
</tbody>
</table>
SelectorGadget

Identify CSS selectors with point-and-click interface

- **Chrome extension**
- Click on element you want to select
  - Selected item: marked in green
  - SelectorGadget makes a guess and marks all elements that matches the selector in yellow
- Deselect wrong elements: marked red
- CSS Selector / Tag can be used in programming languages such as R
To select the price on this website: use CSS selector “.sx-price-large”
Select element (price) on website → “.price”
Web scraping with R
Web scraping with R

Use **rvest** package developed by Hadley Wickham (Chief Scientist at RStudio)
install.packages("rvest")

Most important functions:

- `read_html()`: creates an html document from a webpage
  - Without a proxy: e.g. `read_html("https://www.google.com")`
  - With a proxy: e.g. `read_html(httr::GET(url, user_agent(agent), proxy))`

- `html_nodes()`: select tags
  - e.g. `html_nodes(".sx-price-large")`

- `html_node()`: selects exactly one tag
  - e.g. `html_node(".sx-price-large")` will select only the first tag instead of all matching tags
Web scraping with R

Most important functions (continued):

- `html_text()`: extracts text within tags, to be used after `html_node(s)()`
  - e.g. `html_nodes("sx-price-large") %>% html_text()`

- `html_attr()`: extracts the value of the attribute, to be used after `html_node(s)()`
  - e.g. `html_nodes("a") %>% html_attr("href")` will select the url

- `html_table()`: extracts a table, to be used after `html_node(s)()`
  - e.g. `html_node("table css") %>% html_table()`

- All functions can be chained using the `%>%` (a.k.a. pipe) operator
  - e.g. `read_html("url") %>% html_nodes("css") %>% html_text()`
```r
library(rvest)
library(stringr)

#url
start_url <- "https://www.amazon.com/s/ref=sr_nr_p_n_feature_browse-b_1?fst=sAs%20off&rh=n%3A283155%2Cn%3A211000%2Cn%3A4%2Cp_n_feature_five_browse-bin%3A2579000011%2Cp_n_feature_five_browse-bin%3A6118393011%2Cp_n_feature_browse-bin%3A2656020011&bbn=4&ie=UTF8&qid=1507885684&rnid=618072011"

#load html page
main_page <- read_html(start_url)

#scrape price
price <- main_page %>% html_nodes(".sx-price-large") %>% html_text()
price <- str_trim(price)
price <- str_replace_all(price, "\n                            ", ", ", ")

#scrape product name
prod <- main_page %>% html_nodes(".s-access-title") %>% html_text()

#store scraped data in data frame
data <- data.frame(prod=prod, price=price)
```

<table>
<thead>
<tr>
<th>prod</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harry Potter and the Prisoner of Azkaban: The Illust...</td>
<td>$24.17</td>
</tr>
<tr>
<td>Wonder</td>
<td>$10.51</td>
</tr>
<tr>
<td>The Giver (Bk 1)</td>
<td>$9.42</td>
</tr>
<tr>
<td>Harry Potter and the Sorcerer's Stone: The Illustrated...</td>
<td>$27.70</td>
</tr>
<tr>
<td>Harry Potter and the Chamber of Secrets: The Illustrated...</td>
<td>$31.85</td>
</tr>
<tr>
<td>The Parolining of Prince Gleonorgandre</td>
<td>$16.30</td>
</tr>
<tr>
<td>Descendants 2: Mal's Spell Book 2: More Wicked Magic</td>
<td>$7.34</td>
</tr>
<tr>
<td>The Giving Tree</td>
<td>$1.22</td>
</tr>
<tr>
<td>Fantastic Beasts and Where to Find Them: The Illust...</td>
<td>$24.48</td>
</tr>
<tr>
<td>Narnia</td>
<td>$10.40</td>
</tr>
<tr>
<td>The Last Kids on Earth and the Nightmare King</td>
<td>$0.97</td>
</tr>
<tr>
<td>I'm Just No Good at Rhyming. And Other Nonsense for...</td>
<td>$17.99</td>
</tr>
</tbody>
</table>
library(rvest)

#url
start_url <- "https://www.bol.com/nl/l/dvd/-/N/3133+7929/index.html"

#load html page
main_page <- read_html(start_url)

#scrape release date
releasedate <- main_page %>% html_nodes(".product-small-specs li~ li+ li span") %>% html_text()

#scrape product name
prod <- main_page %>% html_nodes(".product-title") %>% html_text()

#check length of scraped data
str(releasedate)
str(prod)

> start_url <- "https://www.bol.com/nl/l/dvd/-/N/3133+7929/index.html"
> main_page <- read_html(httr::GET(start_url, user_agent(agent), proxy))
> releasedate <- main_page %>% html_nodes(".product-small-specs li~ li+ li span") %>% html_text()
> prod <- main_page %>% html_nodes(".product-title") %>% html_text()
> str(releasedate)
chr [1:17] "oktober 2017" "oktober 2017" "september 2017" "maart 2016" ...
> str(prod)
chr [1:24] "Game Of Thrones - Seizoen 6 (Blu-ray)" ...
Web scraping with R

Problem: Missing values

   e.g. number of prices ≠ number of products

Solution: Scrape Functions:

- scrape_css

  ```r
  scrape_css <- function(css, group) {
    txt <- main_page %>>% html_nodes(group) %>% lapply(. %>% html_nodes(css) %>% html_text() %>% ifelse(identical(. character(0)), NA) %>% unlist
    return(txt)
  }
  ```

- scrape_css_attr

  ```r
  scrape_css_attr <- function(css, group, attribute) {
    txt <- main_page %>>% html_nodes(group) %>% lapply(. %>% html_nodes(css) %>% html_attr(attribute) %>% ifelse(identical(. character(0)), NA) %>% unlist
    return(txt)
  }
  ```
Scrape functions

- `scrape_css(css, group)`
- `scrape_css_attr(css, group, attribute)`

  - css: specific element we want to scrape
  - group: refers to CSS selector that captures the whole observation including subcomponents in which we are interested
  - attribute: specific attribute we want to scrape (e.g. url)
Web scraping with R

Scrape functions

- `scrape_css(css, group)`
- `scrape_css_attr(css, group, attribute)`

Statistics Belgium
library(rvest)

#url
start_url <- "https://www.bol.com/nl/l/dvd/-/N/3133+7929/index.html"

#load html page
main_page <- read_html(start_url)

#scrape release date
releasedate <- scrape_css(".product-small-specs li~li+li span",".product-item__info")

#scrape product name
prod <- scrape_css(".product-title",".product-item__info")

#check length of scraped data
str(releasedate)
str(prod)

> start_url <- "https://www.bol.com/nl/l/dvd/-/N/3133+7929/index.html"
> main_page <- read_html(http::GET(start_url,user_agent(agent), proxy))
> releasedate <- scrape_css(".product-small-specs li~li+li span",".product-item__info")
> prod <- scrape_css(".product-title",".product-item__info")
> str(releasedate)
chr [1:24] NA "oktober 2017" "oktober 2017" NA "september 2017" NA "maart 2016" ...
> str(prod)
chr [1:24] "Game of Thrones - Seizoen 6 (Blu-ray)" ...
library(rvest)
library(stringr)

#url
start_url <- "http://www2.hm.com/en_gb/men/shop-by-product/shirts.html"

#load html page
main_page <- read_html(start_url)

#scrape price
price <- main_page %>% html_nodes(".price") %>% html_text()
price <- str_trim(price)

#scrape product name
prod <- main_page %>% html_nodes(".product-item-heading a") %>% html_text()

#store scraped data in data frame
data <- data.frame(prod=prod, price=price)
Simple example

```r
#scrape price
price <- main_page %>% html_nodes(".price") %>% html_text()
```
```r
library(rvest)
library(stringr)

#url
start_url <- "http://www2.hm.com/en_gb/men/shop-by-product/shirts.html"

#load html page
main_page <- read_html(start_url)

#scrape price
price <- main_page %>% html_nodes(".product-item-details div ~ div .price") %>% html_text()
price <- str_trim(price)

#scrape product name
prod <- main_page %>% html_nodes(".product-item-heading a") %>% html_text()

#store scraped data in data frame
data <- data.frame(prod=prod, price=price)

#Alternative scrape functions
price <- scrape_css(".ng-hide .price", ".product-item-details")
prod <- scrape_css(".product-item-heading a", ".product-item-details")
```

<table>
<thead>
<tr>
<th>prod</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy-iron shirt Slim fit</td>
<td>£12.00</td>
</tr>
<tr>
<td>Easy-iron shirt Slim fit</td>
<td>£12.99</td>
</tr>
<tr>
<td>Cotton shirt Regular fit</td>
<td>£17.00</td>
</tr>
<tr>
<td>Cotton shirt Regular fit</td>
<td>£17.99</td>
</tr>
<tr>
<td>Flannel shirt Regular fit</td>
<td>£19.99</td>
</tr>
<tr>
<td>Easy-iron shirt Slim fit</td>
<td>£12.00</td>
</tr>
<tr>
<td>Checked flannel shirt</td>
<td>£19.99</td>
</tr>
<tr>
<td>Poplin shirt Slim fit</td>
<td>£10.00</td>
</tr>
<tr>
<td>Easy-iron shirt Slim fit</td>
<td>£12.00</td>
</tr>
<tr>
<td>Flannel shirt Regular fit</td>
<td>£19.99</td>
</tr>
<tr>
<td>Denim shirt</td>
<td>£24.00</td>
</tr>
<tr>
<td>Cotton shirt Regular fit</td>
<td>£12.99</td>
</tr>
<tr>
<td>Checked flannel shirt</td>
<td>£17.99</td>
</tr>
<tr>
<td>Checked flannel shirt</td>
<td>£17.99</td>
</tr>
<tr>
<td>Flannel shirt Regular fit</td>
<td>£19.99</td>
</tr>
<tr>
<td>Easy-iron shirt Slim fit</td>
<td>£12.00</td>
</tr>
<tr>
<td>Easy-iron shirt Slim fit</td>
<td>£12.00</td>
</tr>
<tr>
<td>Hooded flannel shirt</td>
<td>£24.99</td>
</tr>
<tr>
<td>Oxford shirt Regular fit</td>
<td>£19.00</td>
</tr>
<tr>
<td>Cotton shirt Regular fit</td>
<td>£17.99</td>
</tr>
<tr>
<td>Easy-iron shirt Slim fit</td>
<td>£12.99</td>
</tr>
<tr>
<td>Easy-iron shirt Slim fit</td>
<td>£12.99</td>
</tr>
<tr>
<td>Checked flannel shirt</td>
<td>£17.99</td>
</tr>
<tr>
<td>Cotton shirt Regular fit</td>
<td>£17.00</td>
</tr>
<tr>
<td>Easy-iron shirt Slim fit</td>
<td>£12.00</td>
</tr>
</tbody>
</table>
Scraping strategy

- Select a website
- Read homepage (read_html)
- Scrape all possible URLs (or predefine)
  - Subpages
  - Categories (html_nodes)
- Loop all of the previous URLs
- Scrape information you want
  - Product name
  - Price
  - ...
- Store all data in a data frame
- Export data frame
#url
main_url <- "http://www2.hm.com"
start_url <- "http://www2.hm.com/en_gb/men/shop-by-product/shirts.html"

# load html page
main_page <- read_html(start_url)

# Scrape subcategories
cat <- main_page %>% html_nodes(".section-menu-subcategory a") %>% html_text()
cat <- str_trim(cat)
cat_url <- main_page %>% html_nodes(".section-menu-subcategory a") %>% html_attr("href")

# Loop all categories and scrape price and product name
data <- NULL
for(i in 1:length(cat)) {
  current_page <- as.character(paste0(main_url, cat_url[i]))
  main_page <- read_html(start_url)

  price <- str_trim(scrape_css(".price", ".product-item-details"))
  price <- str_trim(price)

  prod <- scrape_css(".product-item-heading a", ".product-item-details")

  data_cat <- data.frame(prod = prod, price = price, cat = cat[i])
  data <- rbind(data, data_cat)
}
### Looping different categories

Result of the loop for 2 categories:

<table>
<thead>
<tr>
<th>prod</th>
<th>price</th>
<th>cat</th>
<th>prod</th>
<th>price</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>6  3-pack T-shirts Slim fit</td>
<td>£17.00</td>
<td>T-shirts &amp; Vests</td>
<td>25 Merino wool polo shirt</td>
<td>£34.99</td>
<td>T-shirts &amp; Vests</td>
</tr>
<tr>
<td>9  Polo shirt Slim Fit</td>
<td>£8.99</td>
<td>T-shirts &amp; Vests</td>
<td>28 Long T-shirt</td>
<td>£5.99</td>
<td>T-shirts &amp; Vests</td>
</tr>
<tr>
<td>10 Ribbed vest top</td>
<td>£5.99</td>
<td>T-shirts &amp; Vests</td>
<td>29 Jersey top Slim fit</td>
<td>£8.99</td>
<td>T-shirts &amp; Vests</td>
</tr>
<tr>
<td>11 3-pack T-shirts Regular fit</td>
<td>£17.00</td>
<td>T-shirts &amp; Vests</td>
<td>30 Polo shirt</td>
<td>£8.99</td>
<td>T-shirts &amp; Vests</td>
</tr>
<tr>
<td>12 Merino wool polo shirt</td>
<td>£34.99</td>
<td>T-shirts &amp; Vests</td>
<td>31 Checkered flannel shirt</td>
<td>£17.99</td>
<td>Shirts</td>
</tr>
<tr>
<td>14 Long-sleeved jersey top</td>
<td>£12.99</td>
<td>T-shirts &amp; Vests</td>
<td>33 Easy-iron shirt Slim fit</td>
<td>£12.99</td>
<td>Shirts</td>
</tr>
<tr>
<td>15 Wide T-shirt</td>
<td>£12.99</td>
<td>T-shirts &amp; Vests</td>
<td>34 Cotton shirt Regular fit</td>
<td>£12.99</td>
<td>Shirts</td>
</tr>
<tr>
<td>16 T-shirt with a print motif</td>
<td>£12.99</td>
<td>T-shirts &amp; Vests</td>
<td>35 Cotton shirt Regular fit</td>
<td>£12.99</td>
<td>Shirts</td>
</tr>
<tr>
<td>17 3-pack T-shirts Regular fit</td>
<td>£17.99</td>
<td>T-shirts &amp; Vests</td>
<td>36 Cotton shirt Regular fit</td>
<td>£17.99</td>
<td>Shirts</td>
</tr>
<tr>
<td>18 Polo shirt Slim Fit</td>
<td>£8.99</td>
<td>T-shirts &amp; Vests</td>
<td>37 Checkered flannel shirt</td>
<td>£17.99</td>
<td>Shirts</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td>39 Flannel shirt Regular fit</td>
<td>£19.99</td>
<td>Shirts</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td>40 Checkered flannel shirt</td>
<td>£17.99</td>
<td>Shirts</td>
</tr>
</tbody>
</table>
#url
main_url <- "https://www.amazon.com"

#Go to next page
next_page <- main_page %>% html_node("#pagnNextLink") %>% html_attr("href")
current_page< as.character(paste0(main_url, next_page))
main_page <- read_html(start_url)
RSelenium

Rvest downloads the HTML page and using rvest functions information can be selected

→ dynamic interaction is not possible

Dynamic interaction:

- Clicking a button to load more products
- Scrolling down to automatically load more products
- Filling in a form and click search button

Solution: RSelenium → provides R bindings for the Selenium Webdriver
Clicking a button to load more products ("Laad meer" = Load more)

The script on the next slide will open the webpage automatically in Chrome and click on the button until the button is not more available

The whole webpage can then be scraped using `rvest`
library(RSelenium)
library(rvest)

#url
start_url <- "https://be.avanceshoes.com/be/dames/schoenen/pumps.html"

# loading Selenium server and ChromeDriver
remDr <- remoteDriver(browserName = "chrome")
remDr$open()
Sys.sleep(2)

# navigate to the url
main_page <- remDr$navigate(start_url)

# code to find the button via a CSS selector and then clicking the button "laad meer" (CSS selector ".i-next") # until it disappears
webElems <- remDr$findElements(using = 'css selector', ".i-next")
Sys.sleep(4)
while (length(webElems) != 0) {
    webElem <- remDr$findElement(using = 'css selector', ".i-next")
    webElem$clickElement()
    Sys.sleep(4)
    webElems <- remDr$findElements(using = 'css selector', ".i-next")
    Sys.sleep(4)
}

# getting the final page via rvest
main_page <- read_html(unlist(remDr$getPageSource()), encoding="UTF-8")

# closing the Selenium session
remDr$close()
#code to click on button “Accept Cookies”
webElems <- remDr$findElement(using = 'css selector', "body > div.cookie-notification.js-notification.js-cookie-notification > button")
webElems$clickElement()
library(RSelenium)
library(rvest)

#url

#open the webpage
remDr <- remoteDriver(browserName = "chrome")
remDr$open()
main_page <- remDr$navigate(start_url)

#code to scroll, it scrolls 5 times a certain amount of pixels; in this case 10000
for(i in 1:5){
  remDr$executeScript(paste("scroll(0,",i*10000,");"),list(""))
  Sys.sleep(3)
}

#getting the final page via rvest
main_page <- read_html(unlist(remDr$getPageSource()), encoding="UTF-8")

#closing ChromeDriver
remDr$close()
RSelenium – Filling in a form

Book hotel, flight, train tickets,…

Fill in:
- Destination
- Departure date
- Return date
RSelenium – Filling in a form

Predefining:

- List of destinations
- Number of weeks booked in advance

Result: screen with different prices depending on Airline, stops, options,...

→ Rvest to scrape this data
RSelenium – Filling in a form

```r
# Starting Selenium server
server <- startServer()
Sys.sleep(5)

# URL
start_url <- "https://www.expedia.com/"

# Getting the date of today
current_date <- Sys.Date()
current_date_txt <- format(current_date, "%d/ %m/ %Y")

# Departure date 28 days after the current date
dep_date <- current_date + (4*7)
dep_date_txt <- format(dep_date, "%m/%d/%Y")

# Return date, 7 days after the departure date
ret_date <- dep_date + 7
ret_date_txt <- format(ret_date, "%m/%d/%Y")

# Destination Bogota, so a flight from Brussels to Bogota
departure <- "BRU"
destination <- "BOG"

# Navigate to the URL
remDr <- remoteDriver(browserName = "chrome")
remDr$open()
remDr$navigate(start_url)
# remDr$refresh()
```
RSelenium – Filling in a form

```r
# Close pop-ups
webElem <- remDr$findElement(using = 'css selector', 'button#join-rewards-close-btn')
webElem$clickElement()

webElem <- remDr$findElement(using = 'css selector', 'button.btn-close')
webElem$clickElement()

# Click on flights
webElem <- remDr$findElement(using = 'css selector', '#primary-header-flight')
webElem$clickElement()
Sys.sleep(3)

# Finding the CSS selector of the departure airport and filling it in with the airport
input_dep <- remDr$findElement(using = 'css selector', '#flight-origin-flp')
input_dep$sendKeysToElement(list(departure))
blank <- remDr$findElement(using = 'css selector', '.cols-nested+.cols-nested')
blank$clickElement()
Sys.sleep(3)

input_dest <- remDr$findElement(using = 'css selector', '#flight-destination-flp')
input_dest$sendKeysToElement(list(destination))
Sys.sleep(3)

input_date_dep <- remDr$findElement(using = 'css selector', '#flight-departing-flp')
input_date_dep$clearElement()
Sys.sleep(1)
input_date_dep$sendKeysToElement(list(dep_date_txt))
Sys.sleep(3)
```
RSelenium – Filling in a form

```r
input_date_ret <- remDr$findElement(using="css selector","#flight-returning-flp")
input_date_ret$clearElement()
Sys.sleep(1)
input_date_ret$sendKeysToElement(list(ret_date_txt))
Sys.sleep(3)

close_calendar <- remDr$findElement(using="css selector",".datepicker-close-btn")
close_calendar$clickElement()
Sys.sleep(1)

search <- remDr$findElement(using="css","#flight-lap-or-seat-container-flp ~ .cols-nested .gcw-submit")
search$clickElement()

# getting the page with all the prices in rvest
main_page <- read_html(unlist(remDr$getPageSource()), encoding="UTF-8")

# closing ChromeDriver
remDr$close()
server$stop()
```
Implementation at Statistics Belgium
Web scraping – implementation

- Scripts are executed on a Linux server mostly at night
- Pauses are integrated into the script (Sys.sleep() function) to avoid overloading the website (netiquette!)
- Robot identifies itself as “Statistics Belgium”
  - Using proxy server
    - Read_html(httr::GET(start_url, user_agent(agent), proxy))
      - agent: identification to the website (e.g. NSI name)
- Data are saved first in csv files and loaded afterwards in the SAS Data Warehouse of Statistics Belgium
- All products are extracted (bulk scraping)
  - Exceptions: train tickets or airfares: a list of destinations and departure dates are predefined
Monitoring results of scripts

- Check output
  - Number of records
  - Check results
- Change scripts in case of missing records
  - e.g. due to change in website
- Failed scripts (also receive an automatic mail)
  - Server problems
  - Change in website
### Web scraping Dashboard Statistics Belgium – Overview:

<table>
<thead>
<tr>
<th>Date</th>
<th>Month</th>
<th>Site</th>
<th>Duration</th>
<th>Count</th>
<th>Min</th>
<th>Mean</th>
<th>Max</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-07-31</td>
<td>2017-07-01</td>
<td>Esprit-Filles</td>
<td>239.42</td>
<td>615</td>
<td>5.95</td>
<td>24.86</td>
<td>89.99</td>
<td>09:57:42</td>
</tr>
<tr>
<td>2017-07-31</td>
<td>2017-07-01</td>
<td>Mediakracht</td>
<td>468.18</td>
<td>1093</td>
<td>2.99</td>
<td>703.62</td>
<td>19999</td>
<td>09:37:49</td>
</tr>
<tr>
<td>2017-07-31</td>
<td>2017-07-01</td>
<td>Club</td>
<td>37.44</td>
<td>100</td>
<td>5.2</td>
<td>19.18</td>
<td>26.55</td>
<td>09:10:39</td>
</tr>
<tr>
<td>2017-07-31</td>
<td>2017-07-01</td>
<td>Fnac Livres NL</td>
<td>33.09</td>
<td>100</td>
<td>4.75</td>
<td>15.22</td>
<td>37.96</td>
<td>09:01:14</td>
</tr>
<tr>
<td>2017-07-31</td>
<td>2017-07-01</td>
<td>Fnac Livres FR</td>
<td>31.37</td>
<td>77</td>
<td>4.23</td>
<td>7.91</td>
<td>14.73</td>
<td>09:00:41</td>
</tr>
<tr>
<td>2017-07-31</td>
<td>2017-07-01</td>
<td>Connection 16 weeks</td>
<td>3916.84</td>
<td>33</td>
<td>102</td>
<td>231.33</td>
<td>779</td>
<td>09:52:41</td>
</tr>
<tr>
<td>2017-07-31</td>
<td>2017-07-01</td>
<td>Bol boeken</td>
<td>23.84</td>
<td>73</td>
<td>4.99</td>
<td>18.54</td>
<td>55</td>
<td>08:20:25</td>
</tr>
<tr>
<td>2017-07-31</td>
<td>2017-07-01</td>
<td>Amazon</td>
<td>36.89</td>
<td>191</td>
<td>4.99</td>
<td>19.27</td>
<td>84.08</td>
<td>08:02:41</td>
</tr>
</tbody>
</table>
Web scraping Dashboard Statistics Belgium – Global Graphs
# Web scraping Dashboard Statistics Belgium – Specific

## Table

<table>
<thead>
<tr>
<th>date</th>
<th>month</th>
<th>site</th>
<th>desc_cat_1</th>
<th>desc_cat_2</th>
<th>desc_cat_3</th>
<th>count</th>
<th>min</th>
<th>mean</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-07-30</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Zwangerschapsjeans</td>
<td>19</td>
<td>29</td>
<td>34.79</td>
<td>39</td>
</tr>
<tr>
<td>2017-07-30</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Jeans shorts</td>
<td>13</td>
<td>9</td>
<td>20.54</td>
<td>39</td>
</tr>
<tr>
<td>2017-07-30</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Jeggings</td>
<td>16</td>
<td>9</td>
<td>11.69</td>
<td>19</td>
</tr>
<tr>
<td>2017-07-30</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Bootcut &amp; Flare</td>
<td>7</td>
<td>29</td>
<td>36.14</td>
<td>39</td>
</tr>
<tr>
<td>2017-07-30</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Straight</td>
<td>30</td>
<td>19</td>
<td>32.13</td>
<td>39</td>
</tr>
<tr>
<td>2017-07-30</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Slim</td>
<td>14</td>
<td>19</td>
<td>31.57</td>
<td>39</td>
</tr>
<tr>
<td>2017-07-30</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Super skinny</td>
<td>10</td>
<td>19</td>
<td>22</td>
<td>29</td>
</tr>
<tr>
<td>2017-07-30</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Skinny</td>
<td>14</td>
<td>29</td>
<td>30.43</td>
<td>39</td>
</tr>
<tr>
<td>2017-07-30</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Grote maten</td>
<td>24</td>
<td>19</td>
<td>30.91</td>
<td>49</td>
</tr>
<tr>
<td>2017-07-29</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Zwangerschapsjeans</td>
<td>21</td>
<td>29</td>
<td>34.24</td>
<td>39</td>
</tr>
<tr>
<td>2017-07-29</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Jeans shorts</td>
<td>13</td>
<td>9</td>
<td>20.54</td>
<td>39</td>
</tr>
<tr>
<td>2017-07-29</td>
<td>2017-07-01</td>
<td>CenA</td>
<td>Dames</td>
<td>Jeans</td>
<td>Jeggings</td>
<td>16</td>
<td>9</td>
<td>11.69</td>
<td>19</td>
</tr>
</tbody>
</table>
Web scraping Dashboard Statistics Belgium – Specific Graphs
Experimental indices - Fridges

- Daily web scraping
- Bulk scraping
- Low attrition rate
- Hedonic regression: scraping of characteristics
  - Time Dummy with Movement Splice
  - Time Dummy with Window Splice
Experimental indices - Fridges
Experimental indices - Hotels

Manual price collection

- Sample of hotels
- Once a month virtual reservations are made
- 4 weeks before arrival date
  - One price quote for each hotel
- Booking for 2 adults – 2 nights
- Room type is kept stable (if possible)
- ‘Options’ (e.g. free cancelation) are kept stable (if possible)
Experimental indices - Hotels

Web scraping

- Daily web scraping
- 3 Destinations in Belgium: Brussels, Seaside, Ardennes
- 4 – 8 weeks before arrival date
- Arrival on Friday – Departure on Sunday
- Breakfast and free cancelation
- Star rating

Stratification:

1. Destination
2. Area
3. Weeks booked before arrival date
4. Hotel star rating
Experimental indices - Hotels

Sample size – number of prices:

<table>
<thead>
<tr>
<th>Destination</th>
<th>Manual</th>
<th>Web scraping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brussels</td>
<td>17</td>
<td>2,662</td>
</tr>
<tr>
<td>Seaside</td>
<td>25</td>
<td>12,614</td>
</tr>
<tr>
<td>Ardennes</td>
<td>15</td>
<td>23,552</td>
</tr>
</tbody>
</table>
Experimental indices - Hotels

Brussels

Seaside

Ardennes

CPI - Scraped
Experimental indices - Footwear

- Scrapping multiple times a week
- Bulk scraping
- High attrition rate

% of matching items compared to 07-2016
Experimental indices - Footwear

Availability of footwear:
Experimental indices - Footwear

- Non-matched model to avoid downward drift
- Stratification: men - women