Data Management With R: Web scraping

Matthias Haber

06 November 2017

Prerequisites

Packages

```
library(tidyverse)
library(stringr)
library(rvest) #install.packages("rvest")
library(twitteR) #install.packages("twitteR")
library(streamR) #install.packages("streamR")
library(RCurl) #install.packages("RCurl")
library(ROAuth) #install.packages("ROAuth")
library(httr) #install.packages("httr")
```

Last week's homework

Email addresses

```
email <- c("tom@hogwarts.com",
    "tom.riddle@hogwarts.com",
    "tom@hogwarts.eu.com",
    "potter@hogwarts.com",
    "harry@hogwarts.com",
    "hermione+witch@hogwarts.com")
str_extract(email, "^[\\w\\.]*")
```

```
## [1] "tom"         "tom.riddle" "tom"         "potter"
## [6] "hermione"
```

HTML codes

```
htmlcode <- c("<a>This is a link</a>",
    "<a href='https://github.com'>Link</a>",
    "<div class='test_style'>Test</div>",
    "<div>Hello <span>world</span></div>")
```

HTML codes

```
str_extract(htmlcode, "<\\w+")
```

```
## [1] "<a"   "<a"   "<div" "<div"
```

```
str_extract(htmlcode, ">([\\w\\s]*)<")
```

```
## [1] ">This is a link<" ">Link<"           ">Test<"
## [4] ">Hello <"
```

```
str_extract(htmlcode, "='([\\w://.]*)'")
```

```
## [1] NA                       "='https://github.com'" "='test_style'"
## [4] NA
```

file names

```
file <- c(".bash_profile",
    "workspace.doc",
    "img0912.jpg",
    "updated_img0912.png",
    "documentation.html",
    "favicon.gif",
    "img0912.jpg.tmp",
    "access.lock")
str_extract(file, "(\\w+)\\.(jpg|png|gif)$")
```

```
## [1] NA                     NA                     "img0912.jpg"
## [4] "updated_img0912.png" NA                     "favicon.gif"
## [7] NA                     NA
```

log file

```
log <- c("W/dalvikvm( 1553): threadid=1: uncaught exception",
    "E/( 1553): FATAL EXCEPTION: main",
    "E/( 1553): java.lang.StringIndexOutOfBoundsException",
    "E/( 1553):   at widget.List.makeView(ListView.java:1727)",
    "E/( 1553):   at widget.List.fillDown(ListView.java:652)",
    "E/( 1553):   at widget.List.fillFrom(ListView.java:709)")
str_extract(log, "\\.(\\w+)\\(")
```

```
## [1] NA           NA           NA           ".makeView(" ".fillDown("
## [6] ".fillFrom("
```

```
str_extract(log, ":(\\d+)")
```

```
## [1] NA      NA      NA      ":1727" ":652"  ":709"
```

```
str_extract(log, ":(\\d+)")
```

```
## [1] NA      NA      NA      ":1727" ":652"  ":709"
```

URLs

```
url <- c("ftp://file_server.com:21/top_secret/life_changing_plans.pdf",
         "https://github.com/mhaber/slides#section",
         "file://localhost:4040/zip_file",
         "https://s3cur3-server.com:9999/",
         "market://search/angry%20birds")
str_extract(url, "(\\w+)://")
```

```
## [1] "ftp://"    "https://"  "file://"   "https://"  "market://"
```

```
str_extract(url, "://([\\w-\\.]+)")
```

```
## [1] "://file_server.com"   "://github.com"        "://localhost"
## [4] "://s3cur3-server.com" "://search"
```

```
str_extract(url, "(:(\\d+))")
```

```
## [1] ":21"   NA      ":4040" ":9999" NA
```

Regex help

RStudio string cheatsheet

Webscraping intro

Example from my own work

I was asked to scrape all job ads listed on bund.de

Browsing vs. scraping

Browsing

- you click on something
- browser sends request to server that hosts website
- server returns resource (often an HTML document)
- browser interprets HTML and renders it in a nice fashion

Browsing vs. scraping

Scraping with R

- you manually specify a resource
- R sends request to server that hosts website
- server returns resource
- R parses HTML (i.e., interprets the structure), but does not render it in a nice fashion
- it's up to you to tell R which parts of the structure to focus on and what content to extract

Online text data sources

- **web pages** (e.g. `http://example.com`)
- **web formats** (XML, HTML, JSON, . . .)
- **web frameworks** (HTTP, URL, APIs, . . .)
- **social media** (Twitter, Facebook, LinkedIn, Snapchat, Tumbler, . . .)
- **data in the web** (speeches, laws, policy reports, news, . . .)
- **web data** (page views, page ranks, IP-addresses, . . .)

Before scraping, do some googling!

- If the resource is well-known, someone else has probably built a tool which solves the problem for you.
- ropensci has a ton of R packages providing easy-to-use interfaces to open data.
- The Web Technologies and Services CRAN Task View is a great overview of various tools for working with data that lives on the web in R.

What's HTML?

HyperText Markup Language

- markup language = plain text + markups
- standard for the construction of websites
- relevance for web scraping: web architecture is important because it determines where and how information is stored

Inspect the source code in your browser

Firefox

1. right click on page
2. select “View Page Source”

Chrome

1. right click on page
2. select “View page source”

Safari

1. click on “Safari”
2. select “Preferences”
3. go to “Advanced”
4. check “Show Develop menu in menu bar”
5. click on “Develop”
6. select “Show Page Source.”

CSS?

Cascading Style Sheets

- style sheet language to give browsers information of how to render HTML documents
- CSS code can be stored within an HTML document or in an external CSS file
- selectors, i.e. patterns used to specify which elements to format in a certain way, can be used to address the elements we want to extract information from
- works via tag name (e.g.,`<h2>`,`<p>`) or element attributes `id` and `class`

CSS?

How does it work?

CSS Diner

XPath

- XPath is a query language for selecting nodes from an XML-style document (including HTML)
- provides just another way of extracting data from static webpages
- you can also use XPath with R, it can be more powerful than CSS selectors

Example

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

Table (information)

From Wikipedia, the free encyclopedia

"Tabular" redirects here. For the typewriter key, see tab key.
For sortable tables in Wikipedia, see Help:Sorting

This article may **require cleanup** to meet Wikipedia's quality standards. No cleanup reason has been specified. Please help improve this article if you can. *(October 2009)* *(Learn how and when to remove this template message)*

A **table** is an arrangement of data in rows and columns, or possibly in a more complex structure. Tables are widely used in communication, research, and data analysis. Tables appear in print media, handwritten notes, computer software, architectural ornamentation, traffic signs, and many other places. The precise conventions and terminology for describing tables vary depending on the context. Further, tables differ significantly in variety, structure, flexibility, notation, representation and use.[1][2][3][4][5] In books and technical articles, tables are typically presented apart from the main text in numbered and captioned floating blocks.

Purchased Equipments (June, 2006)			
Item Num#	Item Picture	Item Description	Price
		Shipping Handling, Installation, etc	Expense
1.		IBM Clone Computer.	$ 400.00
		Shipping Handling, Installation, etc	$ 20.00
2.		1GB RAM Module for Computer.	$ 50.00
		Shipping Handling, Installation, etc	$ 14.00
Purchased Equipments (June, 2006)			

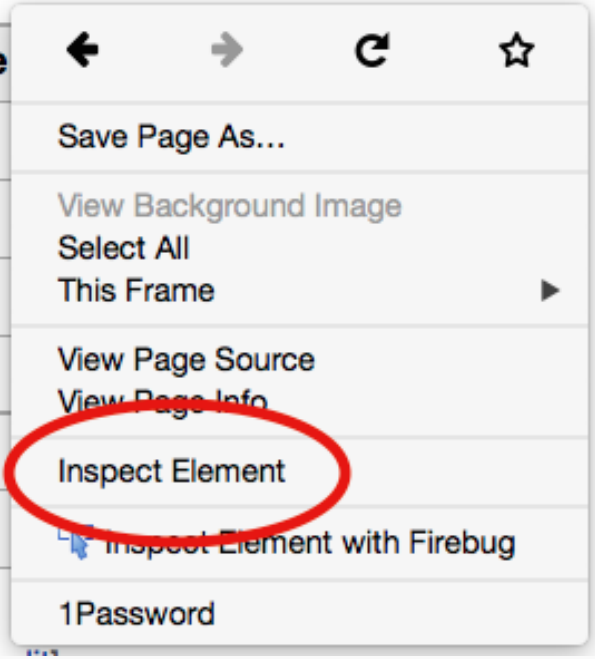
Inspecting elements

Simple table [edit]

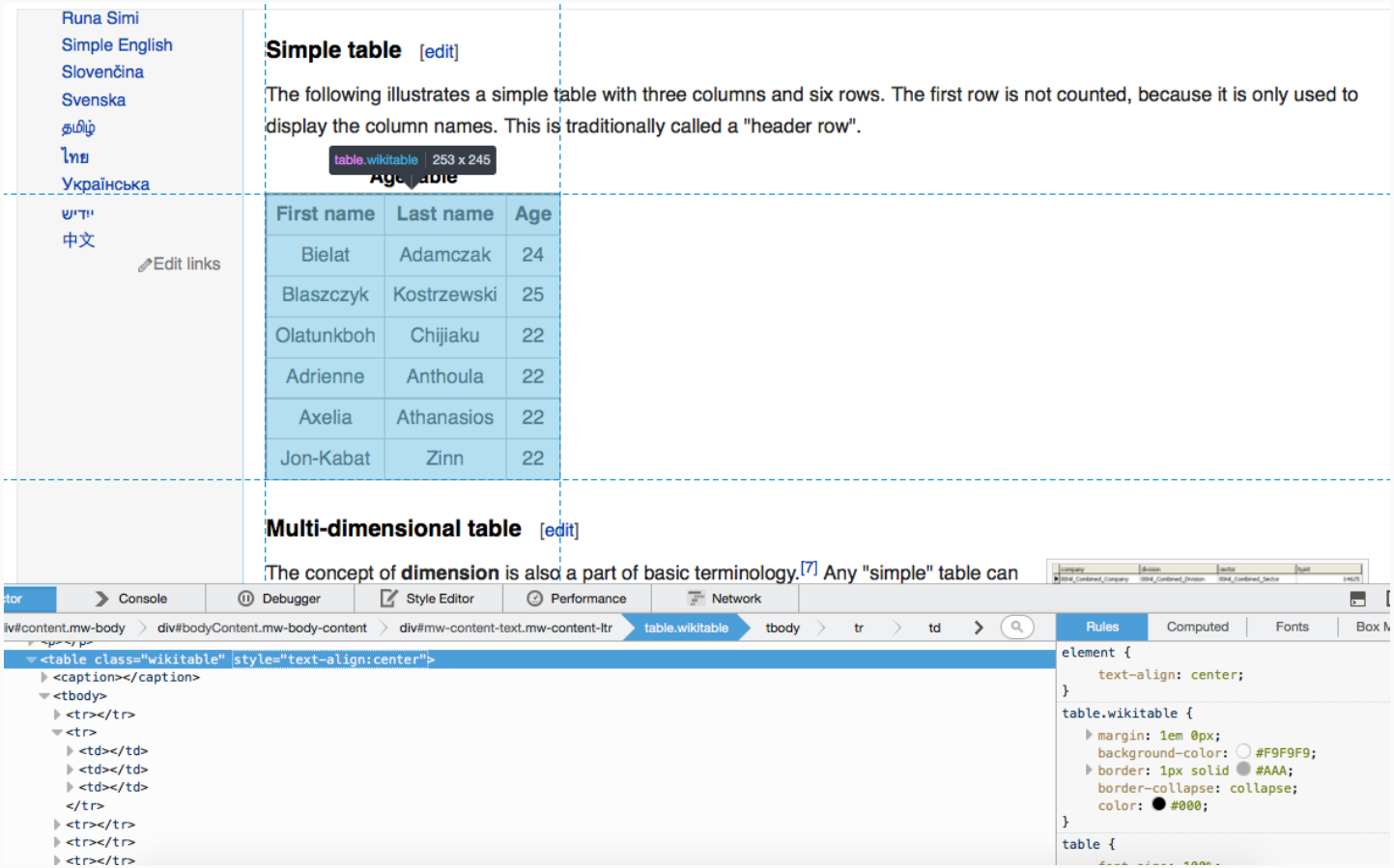
The following illustrates a simple table with three columns and six rows. display the column names. This is traditionally called a "header row".

Age table

First name	Last name	Age
Bielat	Adamczak	24
Blaszczyk	Kostrzewski	25
Olatunkboh	Chijiaku	22
Adrienne	Anthoula	22
Axelia	Athanasios	22
Jon-Kabat	Zinn	22



Hover to find desired elements



Rvest

rvest is a nice R package for web-scraping by (you guessed it) Hadley Wickham.

- see also: `https://github.com/hadley/rvest`
- convenient package to scrape information from web pages
- builds on other packages, such as xml2 and httr
- provides very intuitive functions to import and process webpages

Basic workflow of scraping with rvest

```
# 1. specify URL
"http://en.wikipedia.org/wiki/Table_(information)" %>%

# 2. download static HTML behind the URL and parse it into an XML file
read_html() %>%

# 3. extract specific nodes with CSS (or XPath)
html_node(".wikitable") %>%

# 4. extract content from nodes
html_table(fill = TRUE)
```

```
##   First name   Last name Age
## 1       Tinu    Elejogun  14
## 2  Blaszczyk Kostrzewski  25
## 3       Lily   McGarrett  16
## 4  Olatunkbo    Chijiaku  22
## 5   Adrienne    Anthoula  22
```

Selectorgadget

- Selectorgadget is a Chrome browser extension for quickly extracting desired parts of an HTML page.
- to learn about it, use vignette("selectorgadget")
- to install it, visit `http://selectorgadget.com/`

Selectorgadget

```
url <- "http://spiegel.de/schlagzeilen"
css <- ".schlagzeilen-headline"
url_parsed <- read_html(url)
html_nodes(url_parsed, css = css) %>% html_text
```

Technologies and Packages

- Regular Expressions / String Handling
  - **stringr**, stringi
- HTML / XML / XPAth / CSS Selectors
  - **rvest**, xml2, XML
- JSON
  - **jsonlite**, RJSONIO, rjson
- HTTP / HTTPS
  - **httr**, curl, Rcurl
- Javascript / Browser Automation
  - **RSelenium**
- URL
  - **urltools**

APIs

Working with APIs

- Application programming interfaces (APIs) allow a direct connection to data hosted on a website
- Most modern APIs use HTTP (HyperText Transfer Protocol) for communication and data transfer between server and client
- R package httr as a good-to-use HTTP client interface
- Most web data APIs return data in JSON or XML format
- R packages jsonlite and xml2 good to process JSON or XML-style data

Working with APIs

- If you want to tap an existing API, you have to
  - figure out how it works (what requests/actions are possible, what endpoints exist)
  - (register to use the API)
  - formulate queries to the API from within R
  - process the incoming data

Twitter

Twitter has two types of APIs

- REST APIs –> reading/writing/following/etc.
- Streaming APIs –> low latency access to 1% of global stream - public, user and site streams
- authentication via OAuth
- documentation at `https://dev.twitter.com/overview/documentation`

Accessing the twitter APIs

To access the REST and streaming APIs, you will need to create a twitter application, and generate authentication credentials associated with this application. To do this you will first need to have a twitter account. You will also need to install at least the following R packages: `twitteR`.

Create a twitter application

To register a twitter application and get your consumer keys:

1. Go to `https://apps.twitter.com` in a web browser.
2. Click on 'create new app'.
3. Give your app a unique name, a description, any relevant web address, and agree to the terms and conditions. Set the callback URL to `http://127.0.0.1:1410`.
4. Go to the keys and access section of the app page, and copy your consumer key and consumer secret to the code below.
5. (optional): For actions requiring write permissions, generate an access token and access secret.

Use twitter in R

```r
library(twitteR)
library(streamR)
library(ROAuth)

consumerKey <- 'your key here'
consumerSecret <- 'your secret here'

# Try this first, to use twitteR
setup_twitter_oauth(consumerKey, consumerSecret)
results <- searchTwitter('#Trump')
df <- as.data.frame(t(sapply(results, as.data.frame)))
```

Then try these instructions, to use streamR:
https://github.com/pablobarbera/streamR#installation-and-authentication

Scraping dynamic pages

RSelenium

To scrape dynamic web pages (e.g. with Java running in the background) use `RSelenium`

Group excercises

Breweries in Germany

Fetch a list of cities with breweries in Germany:

`http://www.biermap24.de/brauereiliste.php`

CSS Diner

Repeat playing CSS diner and complete all levels!

Group excercises solution

Breweries in Germany

```
url <- "http://www.biermap24.de/brauereiliste.php"
content <- read_html(url)
anchors <- html_nodes(content, ".bgcolor7 > table td+ td a"
cities <- html_text(anchors)
cities <- str_trim(cities)
cities <- cities[!str_detect(cities, "\\(\\d+\\)")]
length(unique(cities))
```

```
## [1] 806
```

```
cities <- data.frame(sort(table(cities)))
```

Homework Exercises

Homework Exercises

For this week's homework exersises go to Moodle and complete the assignment posted in the Web Scraping section.

Deadline: Sunday, November 12 before midnight.

That's it for today. Questions?