SOCIAL DATA SCIENCE

Data Gathering

Sebastian Barfort

August 05, 2016

University of Copenhagen
Department of Economics

ETHICS

On the ethics of web scraping and data journalism

If an institution publishes data on its website, this data should automatically be public

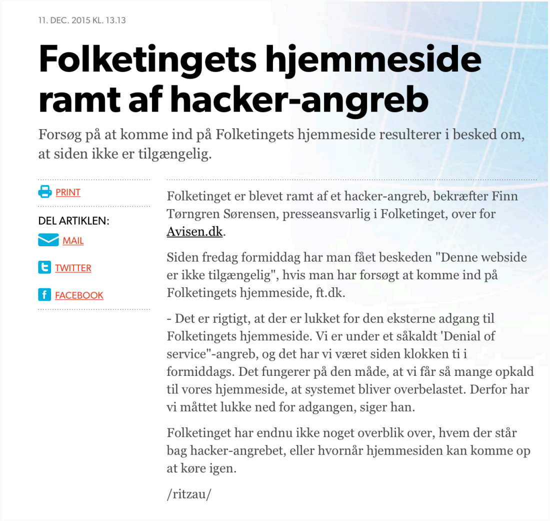
If a regular user can't access the data, we shouldn't try to get it (that would be hacking)

Always read the user terms and conditions

Always check the `robots.txt` *file, which states what is allowed to be scraped*

RULES OF WEB SCRAPING

1. You should check a site's terms and conditions before you scrape them. It's their data and they likely have some rules to govern it.
2. Be nice - A computer will send web requests much quicker than a user can. Make sure you space out your requests a bit so that you don't hammer the site's server.
3. Scrapers break - Sites change their layout all the time. If that happens, be prepared to rewrite your code.
4. Web pages are inconsistent - There's sometimes some manual clean up that has to happen even after you've gotten your data.


11. DEC. 2015 KL. 13.13

Folketingets hjemmeside ramt af hacker-angreb

Forsøg på at komme ind på Folketingets hjemmeside resulterer i besked om, at siden ikke er tilgængelig.

PRINT

DEL ARTIKLEN:

MAIL

TWITTER

FACEBOOK

Folketinget er blevet ramt af et hacker-angreb, bekræfter Finn Tørngren Sørensen, presseansvarlig i Folketinget, over for Avisen.dk.

Siden fredag formiddag har man fået beskeden "Denne webside er ikke tilgængelig", hvis man har forsøgt at komme ind på Folketingets hjemmeside, ft.dk.

- Det er rigtigt, at der er lukket for den eksterne adgang til Folketingets hjemmeside. Vi er under et såkaldt 'Denial of service"-angreb, og det har vi været siden klokken ti i formiddags. Det fungerer på den måde, at vi får så mange opkald til vores hjemmeside, at systemet bliver overbelastet. Derfor har vi måttet lukke ned for adgangen, siger han.

Folketinget har endnu ikke noget overblik over, hvem der står bag hacker-angrebet, eller hvornår hjemmesiden kan komme op at køre igen.

/ritzau/

HOW DOES A WEB PAGE LOOK LIKE?

`https://sebastianbarfort.github.io/`

MOTIVATING EXAMPLE

https://en.wikipedia.org/wiki/Table_%28information%29

EXAMPLE

`rvest` is a nice R package for scraping web pages that don't have an API

To extract something, you start with selectorgadget to figure out which `css` selector matches the data we want

Selectorgadget is a browser extension for quickly extracting desired parts of an HTML page.

With some user feedback, the gadget find out the CSS selector that returns the highlighted page elements.

```r
library("rvest")
link = paste0("http://en.wikipedia.org/",
              "wiki/Table_(information)")
link.data = link %>%
  read_html() %>%
  html_node(".wikitable") %>%
  # extract first node with class wikitable
  html_table()
  # then convert the HTML table into a data frame
```

`html_table` usually only works on ‘nicely’ formatted HTML tables.

First name	Last name	Age
Tinu	Elejogun	14
Blaszczyk	Kostrzewski	25
Lily	McGarrett	16
Olatunkboh	Chijiaku	22
Adrienne	Anthoula	22
Axelia	Athanasios	22
Jon-Kabat	Zinn	22

This is a nice format? Really? Yes, really. It's the format used to render tables on webpages (remember: programming sucks)

```
<table class="wikitable">
  <tr>
    <th>First name</th>
    <th>Last name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Bielat</td>
    <td>Adamczak</td>
    <td>24</td>
  </tr>
   ...
</table>
```

SCRAPING JYLLANDS POSTEN

`http://jyllands-posten.dk/`

SCRAPING JYLLANDS POSTEN IN `rvest`

Assume we want to extract the headlines

- Fire up Selectorgadget
- Find the correct selector
    - `css selector`: `.artTitle a`
    - Want to use xpath? no problem.

SCRAPING HEADLINES

```
css.selector = ".artTitle a"
link = "http://jyllands-posten.dk/"

jp.data = link %>%
  read_html() %>%
  html_nodes(css = css.selector) %>%
  html_text()
```

```
## [1] "\r\n\t\t\tTrump styrtbløder:  »Hillary kan få hi
## [2] "\r\n\t\t\tOmfattende guide til de bedste sportsv
den ene må være verdens bedste "
## [3] "\r\n\t\t\tNovos resultat dykker - men nyt guldhå
## [4] "\r\n\t\t\tSådan bliver du overvåget via din smar
## [5] "\r\n\t\t\tOL-gymnast er træt af kommentarer om »
```

GARBAGE

Notice that there are still some garbage characters in the scraped text

So we need our string processing skills to clean the scraped data

Can be done in many ways

```
library("stringr")
jp.data1 = jp.data %>%
  str_replace_all(pattern = "\\n|\\t|\\r" ,
                  replacement = "")
```

Trump styrtbløder: »Hillary kan få historisk stor sejr«
Omfattende guide til de bedste sportsvogne til prisen: To biler scorer de sja
Novos resultat dykker - men nyt guldhåb går strygende
Sådan bliver du overvåget via din smartphones batteri-tid
OL-gymnast er træt af kommentarer om »en god røv« og »store brystvorter
Nyt studie: Det er langt lettere at blive sygemeldt i Danmark end i Sverige

str_trim

str_trim: Trim whitespace from start and end of string

```r
library("stringr")
jp.data2 = jp.data %>%
  str_trim()
```

Trump styrtbløder: »Hillary kan få historisk stor sejr«
Omfattende guide til de bedste sportsvogne til prisen: To biler scorer de sja
Novos resultat dykker - men nyt guldhåb går strygende
Sådan bliver du overvåget via din smartphones batteri-tid
OL-gymnast er træt af kommentarer om »en god røv« og »store brystvorter
Nyt studie: Det er langt lettere at blive sygemeldt i Danmark end i Sverige

EXTRACTING ATTRIBUTES

What if we also wanted the links embedded in those headlines?

```
jp.links = link %>%
  read_html(encoding = ”UTF-8”) %>%
  html_nodes(css = css.selector) %>%
  html_attr(name = 'href')
```

```
http://jyllands-posten.dk/international/usa/ECE8896067/e
http://www.jyllands-posten.dk/protected/premium/guides/E
http://finans.dk/investor/Regnskaber/ECE8895743/novos-re
http://finans.dk/live/it/ECE8896002/saadan-bliver-du-ove
http://www.jyllands-posten.dk/premium/briefing/ECE889594
http://jyllands-posten.dk/indland/ECE8896168/det-er-lang
```

LOOPING THROUGH COLLECTION OF LINKS

We now have `jp.links`, a vector of all the links to news stories from JP's front page

Let's loop through every link and extract some information.

CLEANING THE VECTOR

```
jp.keep.index = jp.links %>%
  str_detect("http://jyllands-posten.dk/")
jp.remove.index = jp.links %>%
  str_detect("protected|premium")
jp.links.clean = jp.links[jp.keep.index]
jp.links.clean = jp.links.clean[!jp.remove.index]
```

GRAB INFO FROM FIRST LINK

```
first.link = jp.links.clean[1]
first.link.text = first.link %>%
  read_html(encoding = "UTF-8") %>%
  html_nodes("#articleText") %>%
  html_text()
```

```
## [1] ”\r\n\t\tKurven er knækket i meningsmålingerne fo
```

WHILE WE'RE AT IT...

Let's also grab the author of the article

```
first.link %>%
  read_html(encoding = "UTF-8") %>%
  html_nodes(".bylineAuthorName span") %>%
  html_text()
```

```
## [1] "KRISTOFFER ØSTERGAARD KRISTENSEN"
```

TURNING IT INTO A FUNCTION

Function: automate the boring stuff.

Iteration: apply a function to many elements.

Let’s write a function that for each new link will return article text.

SCRAPING FUNCTION

```
scrape_jp = function(link){
  print.link = link %>%
    str_replace(".*[0-9]/", "")
  print(paste0("scraping: ", print.link))
  my.link = link %>%
    read_html(encoding = "UTF-8")
  my.link.text = my.link %>%
    html_nodes("#articleText") %>%
    html_text()
  return(data.frame(
    link = link,
    text = my.link.text ))
}
```

Now we can iterate through all the links and grab the data

```
library("purrr")
jp.article.data = jp.links.clean[1:10] %>%
  map_df(scrape_jp)
```

```
## [1] "scraping: er-hillary-clinton-ved-at-saette-doeds
## [1] "scraping: sportsredaktoer-om-lodtraekning-to-hol
## [1] "scraping: professor-dele-af-hjerneforskningen-sk
## [1] "scraping: roede-partier-ryster-paa-hovedet-ad-fo
## [1] "scraping: burkinibadedag-i-svoemmehallen-vaekker
## [1] "scraping: mads-fenger-faar-disciplinaerstraf-for
## [1] "scraping: wozniacki-skal-baere-fanen-et-hoejdepu
## [1] "scraping: savnet-jaeger-i-groenland-bar-to-doede
## [1] "scraping: liberal-alliance-staar-fast-paa-trusle
## [1] "scraping: se-noegenudstillingen-hvor-bare-bryste
```

Output

```
## Observations: 10
## Variables: 2
## $ link (chr) ”http://jyllands-posten.dk/international
## $ text (chr) ”\r\n\t\tKurven er knækket i meningsmåli
```

EXERCISE

1. Go to `http://www.econ.ku.dk/ansatte/vip/`
2. Create a vector of all links to the researcher's personal home page
3. Go to each researchers page and grab their title
4. Create a data frame of all researchers' names and title

name	title
Kibrom Araya Abay	Postdoc
Steffen Altmann	Lektor
Asger Lau Andersen	Adjunkt
Sebastian Barfort	Post Doc
Jeanet Sinding Bentzen	Adjunkt

Gathering data from APIs

API

API: Application Program Interface

Programmatic instructions for how to interact with a piece of software

Many data sources have API's - largely for talking to other web interfaces

Consists of a set of methods to search, retrieve, or submit data to, a data source

We can write R code to interface with an API (lot's require authentication though)

Many packages already connect to well-known API's

REST v. POST APIs

WTF?

GET: Retrieve whatever is specified by the URL

POST: Create resource at URL with given data

PUT Update resource at URL with given data

Most APIs are REST APIs

Implemented in R in `httr` package.

EXAMPLE: GITHUB ISSUES

`https://developer.github.com/v3/issues/`

```r
library("httr")
url = "https://api.github.com/repos/hadley/dplyr/issues"
get.1 = GET(url, query = list(state = "closed"))
get.2 = GET(url, query = list(state = "closed",
                              labels = "bug"))
```

API OUTPUT

Output from APIs come in one of two formats: `XML` or `JSON`

`JSON`: Javascript Object Notation

- Widely used in web APIs
- Becoming de facto standard for online data format
- Read into R with `jsonlite` package

`XML`: Extensible Markup Language

- Less common today
- Read into R with `xml2` package

JSON

```
{
”Title”: ”Frozen”,
”Year”: ”2013”,
”Rated”: ”PG”,
”Released”: ”27 Nov 2013”,
”Runtime”: ”102 min”,
”Genre”: ”Animation, Adventure, Comedy”,
”Director”: ”Chris Buck, Jennifer Lee”
...
}
```

XML

```
<?xml version="1.0"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applicat
      with XML.</description>
   </book>
```

PARSING

```r
library("jsonlite")
get.1.parsed = content(get.1, as = "text")
get.1.data = fromJSON(get.1.parsed, flatten = TRUE)
get.2.parsed = content(get.2, as = "text")
get.2.data = fromJSON(get.2.parsed, flatten = TRUE)
```

`get.1.data`

number	comments	user.login	closed_at
2050	5	dhagmann	2016-08-02T14:12:36Z
2039	1	Tutuchan	2016-07-27T11:44:36Z
2034	2	joethorley	2016-07-22T19:16:25Z
2032	4	happyshows	2016-07-22T15:18:51Z
2028	2	dxf1david	2016-07-20T18:09:45Z
2027	1	aadler	2016-07-20T16:59:19Z
2026	1	utalo	2016-07-19T15:32:24Z
2025	1	lordyo	2016-07-19T14:22:08Z

get.2.data

number	comments	user.login	closed_at
1870	6	RobertMyles	2016-06-20T13:24:47Z
1831	1	iangow	2016-07-05T12:30:09Z
1803	6	mdsumner	2016-05-26T15:47:38Z
1800	4	jennybc	2016-05-26T13:32:38Z
1789	4	davharris	2016-06-01T19:53:06Z
1779	4	hadley	2016-05-27T20:02:17Z
1751	7	gtumuluri	2016-05-27T19:37:26Z
1750	2	karldw	2016-05-03T09:17:42Z

PACKAGES FOR WORKING WITH APIS

Luckily, you rarely have to access APIs manually

R already has *a lot of* packages for easy access to many APIs

Check some of them out here

twitter

twitteR is an R package which provides access to the Twitter API

Create an app here

```r
library("twitteR")
consumer_key = 'your key'
consumer_secret = 'your secret'
access_token = 'your access token'
access_secret = 'your access secret'

setup_twitter_oauth(consumer_key,
                    consumer_secret,
                    access_token,
                    access_secret)

searchTwitter("#dkpol", n=500)
```