SOCIAL DATA SCIENCE

Data Gathering

Sebastian Barfort

August 03, 2016

University of Copenhagen
Department of Economics

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.

HOW DOES A WEB PAGE LOOK LIKE?

MOTIVATING EXAMPLE

rvest EXAMPLE

`rvest` is a nice R package for web-scraping

```
library("rvest")
read_html("http://en.wikipedia.org/wiki/Table_(informati
  html_node(".wikitable") %>% # extract first node with
  html_table() # then convert the HTML table into a data
```

```
##    First name   Last name Age
## 1        Tinu    Elejogun  14
## 2   Blaszczyk Kostrzewski  25
## 3        Lily   McGarrett  16
## 4 Olatunkboh    Chijiaku  22
## 5    Adrienne    Anthoula  22
## 6      Axelia  Athanasios  22
## 7   Jon-Kabat        Zinn  22
```

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

This is a nice format? Really? Yes, really. It’s the format used to render tables on webpages.

```
<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>
  <tr>
    <td>Blaszczyk</td>
    <td>Kostrzewski</td>
    <td>25</td>
  </tr>
  <tr>
```

We’re rarely that lucky - the data we want is not often in a `<table>` format

Luckily, selectorgadget can help

Selectorgadget is a Chrome 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.

Let’s give it a shot on Jyllands Posten’s web page

SCRAPING EXAMPLE

SCRAPING JYLLANDS POSTEN IN `rvest`

Assume we want to extract the headlines

- fire up Selectorgadget
- find the correct selector
    - css selector: `.header`
    - want to use xpath? look at the XPATH tool: Chrome extension "Xpath Helper" by Adam Sadovsky

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

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

```
##  [1] "\r\n\t\t\tFransk imam til JP: Kirketerrorist er
##  [2] "\r\n\t\t\tNu starter kampen: Disse dyr og plant
##  [3] "\r\n\t\t\tKendt sportskæde opgiver butikkerne i
##  [4] "\r\n\t\t\tStorbritannien lader op til \"super-t
##  [5] "\r\n\t\t\tNetop meldt ud: Den store Windows 10-
##  [6] "\r\n\t\t\tEfter fire måneder i et ødelagt telt:
##  [7] "\r\n\t\t\tDansk designer i spidsen for genopstå
##  [8] "\r\n\t\t\tNy Mercedes E-klasse: Selv med den bi
##  [9] "\r\n\t\t\tInternationalt sportsmagasin:  Så man
## [10] "\r\n\t\t\t16-årig naturist: »Jeg har hele tiden
## [11] "\r\n\t\t\tDemokratisk oase kan sande til: Mysti
```

GARBAGE

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

```
head(jp.data, 5)
```

```
## [1] "\r\n\t\t\tFransk imam til JP: Kirketerrorist er
## [2] "\r\n\t\t\tNu starter kampen: Disse dyr og plante
## [3] "\r\n\t\t\tKendt sportskæde opgiver butikkerne i
## [4] "\r\n\t\t\tStorbritannien lader op til \"super-to
## [5] "\r\n\t\t\tNetop meldt ud: Den store Windows 10-o
```

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

```
library("stringr")
jp.data.clean = jp.data %>%
  str_replace_all(pattern = "\\n" , replacement = " ") %
  str_trim()
```

```r
head(jp.data.clean, 15)
```

```
##  [1] "Fransk imam til JP: Kirketerrorist er »afskyeli
##  [2] "Nu starter kampen: Disse dyr og planter skal be
##  [3] "Kendt sportskæde opgiver butikkerne i Danmark"
##  [4] "Storbritannien lader op til \"super-torsdag\""
##  [5] "Netop meldt ud: Den store Windows 10-opdatering
##  [6] "Efter fire måneder i et ødelagt telt:  Nogle ga
##  [7] "Dansk designer i spidsen for genopstået bilmærk
##  [8] "Ny Mercedes E-klasse: Selv med den billigste mo
##  [9] "Internationalt sportsmagasin:  Så mange OL-meda
## [10] "16-årig naturist: »Jeg har hele tiden kunnet fø
## [11] "Demokratisk oase kan sande til: Mystiske dobbel
## [12] "Sådan bruger du din afdragsfrihed bedst"
## [13] "Dansk målmand ankommer til OL uden et minut som
## [14] "Fem sendt på hospitalet efter masseslagsmål på
## [15] "Debat: Hvad er der sket med Post Danmark?"
```

LINKS

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

```
jp.links = read_html(link, encoding = "UTF-8") %>%
  html_nodes(css = css.selector) %>%
  html_attr(name = 'href')
head(jp.links, 5)
```

```
## [1] "http://jyllands-posten.dk/international/europa/E
## [2] "http://jyllands-posten.dk/nyviden/ECE8890751/nu-
## [3] "http://finans.dk/live/erhverv/ECE8892016/stadium
## [4] "http://finans.dk/investor/ECE8891852/storbritann
## [5] "http://finans.dk/live/it/ECE8891988/netop-meldt-
```

LOOPING THROUGH COLLECTION OF LINKS

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

Let's loop through them and grab all the text

Looping in R is pretty easy (although often inefficient)

LOOPS IN R

```
for(i in 1:5){
  print(paste("I'm now at number", i, sep = " "))
}
```

```
## [1] "I'm now at number 1"
## [1] "I'm now at number 2"
## [1] "I'm now at number 3"
## [1] "I'm now at number 4"
## [1] "I'm now at number 5"
```

WE NEED A FUNCTION TO GRAB ALL THE TEXT AT JP

Functions are easy to write (but be careful)

```
my.first.function = function(number){
  return(number + 5)
}
my.first.function(10)
```

```
## [1] 15
```

```
my.first.function("hello")
```

```
## Error in number + 5: non-numeric argument to binary o
```

```
my.first.function = function(number){
  if(!is.numeric(number)){
      stop("your 'number' is not numeric")
  }
  else{
    return(number + 5)
  }
}
my.first.function(10)
```

```
## [1] 15
```

```
my.first.function("hello")
```

```
## Error in my.first.function("hello"): your 'number' is
```

```r
my.first.function = function(number){
  if(!is.numeric(number)){
      number = ”you did not provide a number”
      return(print(number))
  }
  else{
    return(number + 5)
  }
}
my.first.function(10)
```

```
## [1] 15
```

```r
my.first.function(”hello”)
```

```
## [1] ”you did not provide a number”
```

RETURNING TO OUR EXAMPLE

Let’s look at the first link

```
jp.links[1]
```

```
## [1] ”http://jyllands-posten.dk/international/europa/E
```

There might be an encoding error, see actual R output

Let’s go to that page

GRAB INFO FROM FIRST LINK

```r
first.link = jp.links[1]
first.link.text = read_html(first.link, encoding = "UTF-
  html_nodes("#articleText p") %>%
  html_text()
head(first.link.text, 3)
```

```
## [1] "»En person, som er afskyelig og kvalmende.«"
## [2] "Sådan siger imamen Mohammed Karabila til Jylland
## [3] "Efter hændelsen nægter det lokale muslimske samf
```

Very close...

```
first.link.text.collapsed = paste(first.link.text, colla
head(first.link.text.collapsed, 3)
```

```
## [1] ”»En person, som er afskyelig og kvalmende.«Sådan
```

WHILE WE'RE AT IT...

Let's also grab the author of the article

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

```
## [1] "FREDERIK HJORTH GERNIGON"
```

TURNING IT INTO A FUNCTION

We need a function that for each new link will return the text we're interested in

```
scrape_jp = function(link){
  my.link = read_html(link, encoding = "UTF-8")
  my.link.text = my.link %>%
    html_nodes("#articleText p") %>% html_text() %>%
    paste(collapse = "")
  my.link.author = my.link %>%
    html_nodes(".bylineAuthorName span") %>% html_text()
  return(cbind( my.link.author, link, my.link.text ))
}
```

```r
scrape_jp(first.link)
```

```
##      my.link.author
## [1,] "FREDERIK HJORTH GERNIGON"
##      link
## [1,] "http://jyllands-posten.dk/international/europa/
##      my.link.text
## [1,] "»En person, som er afskyelig og kvalmende.«Såda
```

Now we can loop through the links and grab the data

We store the data in a list that we later turn into a data frame

```r
my.jp.data = list() # initialize empty list
for (i in jp.links[1:5]){
  print(paste("processing", i, sep = " "))
  my.jp.data[[i]] = scrape_jp(i)
  # waiting one second between hits
  Sys.sleep(1)
  cat(" done!\n")
}
```

```
## [1] "processing http://jyllands-posten.dk/internation
##  done!
## [1] "processing http://jyllands-posten.dk/nyviden/ECE
##  done!
## [1] "processing http://finans.dk/live/erhverv/ECE8892
##  done!
## [1] "processing http://finans.dk/investor/ECE8891852/
##  done!
```

transforming it into a data.frame

```
library("plyr")
df = ldply(my.jp.data)
df$article = jp.data.clean[1:5]
head(df, 2)
```

```
##
## 1   http://jyllands-posten.dk/international/europa/EC
## 2 http://jyllands-posten.dk/nyviden/ECE8890751/nu-sta
##                          my.link.author
## 1          FREDERIK HJORTH GERNIGON
## 2 KRISTOFFER ØSTERGAARD KRISTENSEN
##
## 1   http://jyllands-posten.dk/international/europa/EC
## 2 http://jyllands-posten.dk/nyviden/ECE8890751/nu-sta
##
## 1 »En person, som er afskyelig og kvalmende.«Sådan si
## 2
##
```

YOUR TURN

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 r.interest
## 1           Kibrom Araya Abay    Postdoc
## 2             Steffen Altmann     Lektor
## 3          Asger Lau Andersen    Adjunkt
## 4           Sebastian Barfort   Post Doc
## 5      Jeanet Sinding Bentzen    Adjunkt
## 6         Ulf Michael Bergman     Lektor
## 7       Simon Halphen Boserup    Postdoc
## 8    Carl-Johan Lars Dalgaard  Professor
## 9              Jeppe Druedahl   Post Doc
## 10               Mette Ejrnæs  Professor
## 11        Ferran Elias Moreno    Adjunkt
```

GATHERING DATA FROM APIS

API

API = **A**pplication **P**rogram **I**nterface

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 (we'll look at a couple today)

twitter

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

streamR provides access to Twitters streaming 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)
```

NYT API

rtimes is a collection of functions to search and acquire data from various New York Times APIs.

Register for your own API keys here

```
library("rtimes")
out = as_search(q = "bailout",
                begin_date = "20081001",
                end_date = '20081201',
                 key = "XXX")
out$data[1:2]
```

STATISTICS DENMARK API

This R package connects to the StatBank API from Statistics Denmark.

```
library(”devtools”)
install_github(”rOpenGov/dkstat”)
```

Let’s you programatically work with Statistics Denmark data

```r
library("dkstat")
dst_search(string = "bnp", field = "text")
```

```
aulaar = dst_get_data(table = "AULAAR", KØN = "Total",
                      PERPCT = "Per cent of the labour
                      Tid = 2013,
                      lang = "en")
aulaar
```