Webscraping in R - DRAFT

a short primer

Mark Dayton

February 27, 2018

What is it? (webscraping using R)

Webscraping in R is basically....

- using various R package/functions to make retrieval and parsing of "web data" (fairly) easy

What is it? (webscraping using R)

Webscraping in R is basically....

- using various R package/functions to make retrieval and parsing of "web data" (fairly) easy
- caveats: whether the given web page is "nicely formatted" or not...
  as well as the underlying construction of the web page (site)

What is it? (webscraping using R)

Webscraping in R is basically....

- using various R package/functions to make retrieval and parsing of "web data" (fairly) easy
- caveats: whether the given web page is "nicely formatted" or not...
  as well as the underlying construction of the web page (site)
  often times *(but not always)* determines the level of effort required to do the job...

What is it? (webscraping using R)

Webscraping in R is basically....

- using various R package/functions to make retrieval and parsing of "web data" (fairly) easy
- caveats: whether the given web page is "nicely formatted" or not...
  as well as the underlying construction of the web page (site)
  often times *(but not always)* determines the level of effort required to do the job...
- HMTL/CSS complexity? HTML tables rendered via javascript? deciphering XML hierarchy?
  Password protected site?....
  - all can determine which packages/functions to utilize in your effort and the overall approach to take

Ok, so let's see it work...

by way of *illustration*, how about this....

- let's use parsed data from *one website* as a means to gather data from *another* website.....

Ok, so let's see it work...

by way of *illustration*, how about this....

- let's use parsed data from *one website* as a means to gather data from *another* website..... along the way we'll make use of the *rvest* package

Ok, so let's see it work...

by way of *illustration*, how about this....

- let's use parsed data from *one website* as a means to gather data from *another* website..... along the way we'll make use of the *rvest* package
  ...as well as selectorgadget

Ok, so let's see it work...

by way of *illustration*, how about this....

- let's use parsed data from *one website* as a means to gather data from *another* website..... along the way we'll make use of the *rvest* package
  ...as well as selectorgadget
  ...and the *right-click-inspect-element* technique of deciphering the correct css (or xpath)

Ok, so let's see it work...

by way of *illustration*, how about this....

- let's use parsed data from *one website* as a means to gather data from *another* website..... along the way we'll make use of the *rvest* package
  ...as well as selectorgadget
  ...and the *right-click-inspect-element* technique of deciphering the correct css (or xpath)
  - so we can *figure out* the 'correct' options to pass to rvest so it can *do its thing*

Ok, so let's see it work...

by way of *illustration*, how about this....

- let's use parsed data from *one website* as a means to gather data from *another* website..... along the way we'll make use of the *rvest* package
  ...as well as selectorgadget
  ...and the *right-click-inspect-element* technique of deciphering the correct css (or xpath)
  - so we can *figure out* the 'correct' options to pass to rvest so it can *do its thing*
  - after that, we'll go through dealing with XML-based webdata (using the *XML2R* package)

Ok, so let's see it work...

by way of *illustration*, how about this....

- let's use parsed data from *one website* as a means to gather data from *another* website..... along the way we'll make use of the *rvest* package
  ...as well as selectorgadget
  ...and the *right-click-inspect-element* technique of deciphering the correct css (or xpath)
  - so we can *figure out* the 'correct' options to pass to rvest so it can *do its thing*
  - after that, we'll go through dealing with XML-based webdata (using the *XML2R* package)
  - and finish up with using *rdom* or *RSelenium* to parse data from sites that use javascript to dynamically create website content (rvest can't help here...)

But first, some prerequisites for parsing HTML

things to study up on as you learn this topic

- Suggest to get comfortable with 1 of these 2 "utilities" (or rather *techniques*)....

But first, some prerequisites for parsing HTML

things to study up on as you learn this topic

- Suggest to get comfortable with 1 of these 2 "utilities" (or rather *techniques*)....
  - using selectorgadget

But first, some prerequisites for parsing HTML

things to study up on as you learn this topic

- Suggest to get comfortable with 1 of these 2 **"utilities"** (or rather *techniques*)....
  - using selectorgadget
  - using (my preference) **chrome developer tools** (i.e. *right-click-inspect-element*)

But first, some prerequisites for parsing HTML

things to study up on as you learn this topic

- Suggest to get comfortable with 1 of these 2 **"utilities"** (or rather *techniques*)....
  - using selectorgadget
  - using (my preference) **chrome developer tools** (i.e. *right-click-inspect-element*)

- Additionally, here's a link to a *css-for-dummies* interactive game for getting over the hump

But first, some prerequisites for parsing HTML

things to study up on as you learn this topic

- Suggest to get comfortable with 1 of these 2 **"utilities"** (or rather *techniques*)....
  - using selectorgadget
  - using (my preference) **chrome developer tools** (i.e. *right-click-inspect-element*)

- Additionally, here's a link to a *css-for-dummies* interactive game for getting over the hump
  - going though this will greatly ease anxiety when navigating webpages with complex HTML/CSS structures...

But first, some prerequisites for parsing HTML

things to study up on as you learn this topic

- Suggest to get comfortable with 1 of these 2 **"utilities"** (or rather *techniques*)....
  - using selectorgadget
  - using (my preference) **chrome developer tools** (i.e. *right-click-inspect-element*)

- Additionally, here's a link to a *css-for-dummies* interactive game for getting over the hump
  - going though this will greatly ease anxiety when navigating webpages with complex HTML/CSS structures...
  - And, a link to an interactive doc on the selectr::css_to_xpath function...(likely to come in handy at some point)

But first, some prerequisites for parsing HTML

things to study up on as you learn this topic

- Suggest to get comfortable with 1 of these 2 **"utilities"** (or rather *techniques*)....
  - using selectorgadget
  - using (my preference) **chrome developer tools** (i.e. *right-click-inspect-element*)

- Additionally, here's a link to a *css-for-dummies* interactive game for getting over the hump
  - going though this will greatly ease anxiety when navigating webpages with complex HTML/CSS structures...
  - And, a link to an interactive doc on the selectr::css_to_xpath function...(likely to come in handy at some point)

    Just a ***little*** bit of knowledge on deciphering CSS (and later, xpath hierarchies for XML-based data) will go a **looooong** ways!

ok - enough now - let's parse some (html) webdata

Example 1:

- from a webpage listing all R User Groups (*website #1*), grab the groups located in the U.S., and filter on the (fairly large) subset that use meetup.com (*website #2*) as their organizational hub.....so we can gather the following attributes:

ok - enough now - let's parse some (html) webdata

Example 1:

- from a webpage listing all R User Groups (*website #1*), grab the groups located in the U.S., and filter on the (fairly large) subset that use meetup.com (*website #2*) as their organizational hub.....so we can gather the following attributes:
    - location

ok - enough now - let's parse some (html) webdata

Example 1:

- from a webpage listing all R User Groups (*website #1*), grab the groups located in the U.S., and filter on the (fairly large) subset that use meetup.com (*website #2*) as their organizational hub.....so we can gather the following attributes:

    - location
    - inception date

ok - enough now - let's parse some (html) webdata

Example 1:

- from a webpage listing all R User Groups (*website #1*), grab the groups located in the U.S., and filter on the (fairly large) subset that use meetup.com (*website #2*) as their organizational hub.....so we can gather the following attributes:

    - location
    - inception date
    - nbr of members

ok - enough now - let's parse some (html) webdata

Example 1:

- from a webpage listing all R User Groups (*website #1*), grab the groups located in the U.S., and filter on the (fairly large) subset that use meetup.com (*website #2*) as their organizational hub.....so we can gather the following attributes:

    - location
    - inception date
    - nbr of members
    - nbr of meetings since inception

ok - enough now - let's parse some (html) webdata

Example 1:

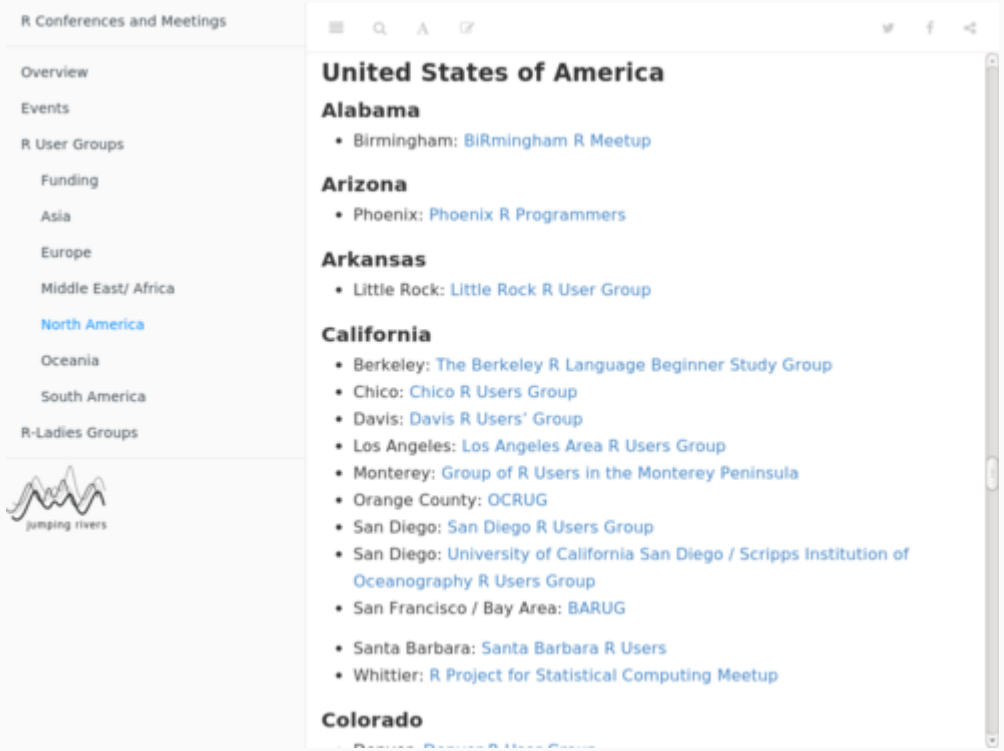
- from a webpage listing all R User Groups (*website #1*), grab the groups located in the U.S., and filter on the (fairly large) subset that use meetup.com (*website #2*) as their organizational hub.....so we can gather the following attributes:

    - location
    - inception date
    - nbr of members
    - nbr of meetings since inception

    NOTE: the 2nd website will be parsed as many times as there are user groups that use meetup.com...not just once!

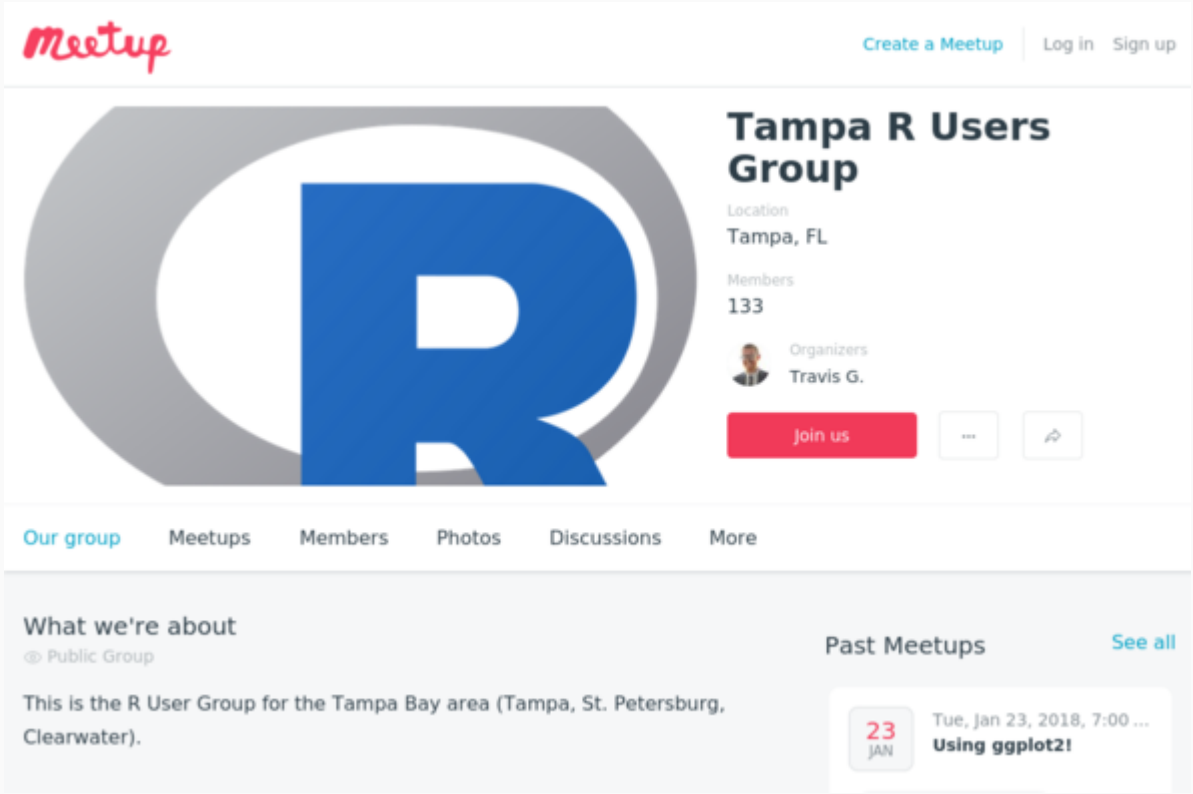
Example 1 - Website 1: R User Groups (in U.S.) parsed one time

https://jumpingrivers.github.io/meetingsR/r-user-groups.html#united-states-of-america



Example 1 - Website 2: meetup.com (parsed multiple times)

https://www.meetup.com/Tampa-R-Users-Group/

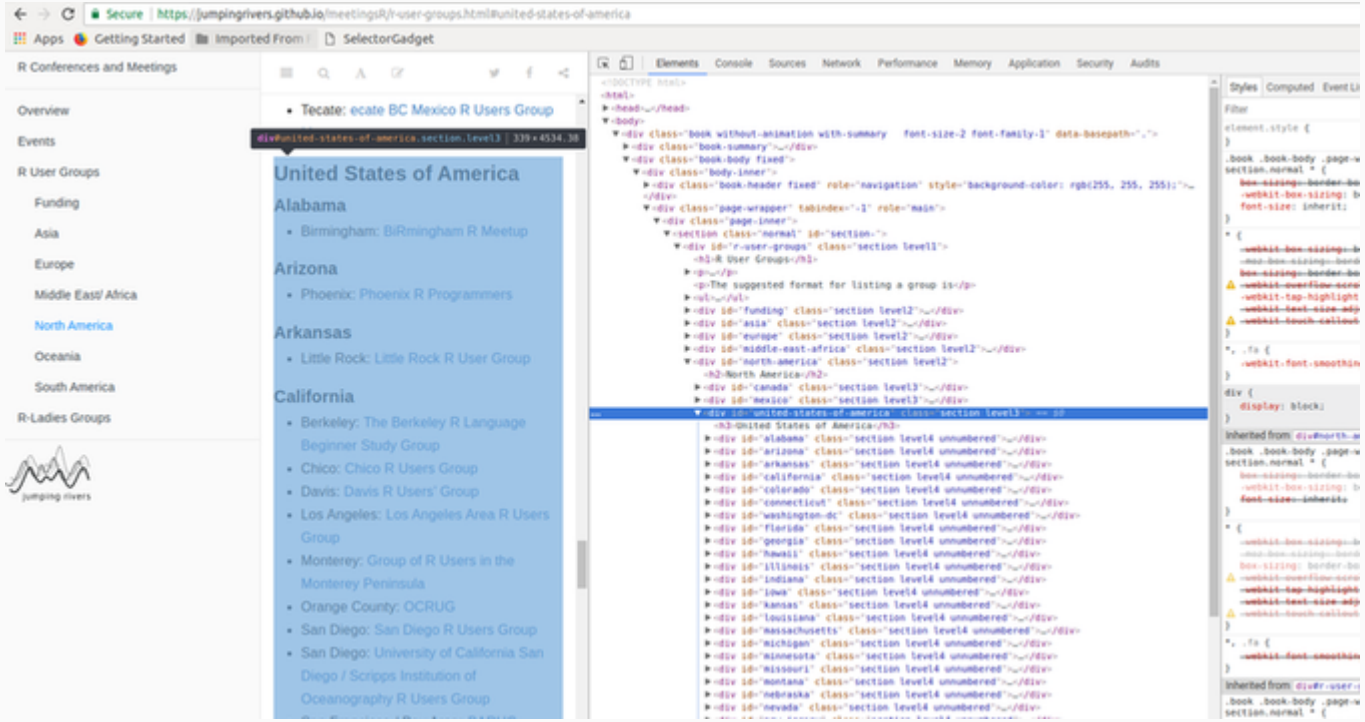


right-click-inspect-element (website #1)

In chrome, *right-click-inspect-element*, and mouse-over the list of R user groups highlighted (in blue!).

The CSS to pass to rvest::html_nodes() is in the **black retangular box**

div#united-states-of-america.section.level3



scrape URLs via css (& xpath)

Now that we have the css, we can use it in the **rvest::html_nodes()** method... as well as pass it to the **selectr::css_to_xpath** method (as a way to show the results will be the same when using **css** or **xpath** parms in the call to **rvest::html_nodes()**

scrape URLs via css (& xpath)

Now that we have the css, we can use it in the **rvest::html_nodes()** method... as well as pass it to the **selectr::css_to_xpath** method (as a way to show the results will be the same when using **css** or **xpath** parms in the call to **rvest::html_nodes()**

Remember, for website #1, we will scrape just URLs within the U.S.

```
library(rvest)
library(selectr)
url1 <- "https://jumpingrivers.github.io/meetingsR/r-user-groups.html#united-states-of-america"

us_rugs <- read_html(url1)                                  # read url1 to create object for US R User Grps

css1a <- "div#united-states-of-america.section.level3"      # from "right-click-inspect"" method

xp1a  <- css_to_xpath(selector=css1a)                       # get xpath equivalent for css (to show all.equal==TRUE)

rugs_urls1a <- html_nodes(us_rugs,css=css1a) %>%
               html_nodes(.,"a") %>%
                html_attr(.,"href")

rugs_urlsxp1a <- html_nodes(us_rugs,xpath=xp1a) %>% # use xpath this time ... "a" -->  denotes nodes w/a URL (with "href" attribute)
                 html_nodes(.,"a") %>%
                 html_attr(.,"href")

all.equal(rugs_urls1a,rugs_urlsxp1a)
```

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

filter & clean URLs before passing to website #2

Not **all** of the 83 URLs use meetup.com (but most do), and we'll clean them up **before the next step**:

```
mu_urls ← rugs_urls1a[grep("meetup.com",rugs_urls1a)] %>%
          gsub("\\d+{4}|events/|members|files/","",.)
length(mu_urls)
```

```
## [1] 64
```

```
mu_urls[1:5]     # 64 out of 83 R User Groups in U.S. use meetup.com
```

```
## [1] "https://www.meetup.com/Bi-R-mingham-R-Meetup/"
## [2] "https://www.meetup.com/Central-Arkansas-R-User-Group/"
## [3] "https://www.meetup.com/r-enthusiasts/"
## [4] "https://www.meetup.com/Chico-R-Users-Group/"
## [5] "https://www.meetup.com/LAarea-R-usergroup/"
```

- OK...for the R-User Groups using meetup.com, *we've got their urls*

filter & clean URLs before passing to website #2

Not **all** of the 83 URLs use meetup.com (but most do), and we'll clean them up **before the next step**:

```
mu_urls ← rugs_urls1a[grep("meetup.com",rugs_urls1a)] %>%
          gsub("\\d+{4}|events/|members|files/","",.)
length(mu_urls)
```

```
## [1] 64
```

```
mu_urls[1:5]     # 64 out of 83 R User Groups in U.S. use meetup.com
```

```
## [1] "https://www.meetup.com/Bi-R-mingham-R-Meetup/"
## [2] "https://www.meetup.com/Central-Arkansas-R-User-Group/"
## [3] "https://www.meetup.com/r-enthusiasts/"
## [4] "https://www.meetup.com/Chico-R-Users-Group/"
## [5] "https://www.meetup.com/LAarea-R-usergroup/"
```

- OK...for the R-User Groups using meetup.com, *we've got their urls*
- and since **meetup.com** is a "well-formatted" website (i.e. consistent in HTML/CSS structure)....

filter & clean URLs before passing to website #2

Not **all** of the 83 URLs use meetup.com (but most do), and we'll clean them up **before the next step**:

```
mu_urls ← rugs_urls1a[grep("meetup.com",rugs_urls1a)] %>%
          gsub("\\d+{4}|events/|members|files/","",.)
length(mu_urls)
```

```
## [1] 64
```

```
mu_urls[1:5]     # 64 out of 83 R User Groups in U.S. use meetup.com
```

```
## [1] "https://www.meetup.com/Bi-R-mingham-R-Meetup/"
## [2] "https://www.meetup.com/Central-Arkansas-R-User-Group/"
## [3] "https://www.meetup.com/r-enthusiasts/"
## [4] "https://www.meetup.com/Chico-R-Users-Group/"
## [5] "https://www.meetup.com/LAarea-R-usergroup/"
```

- OK...for the R-User Groups using meetup.com, *we've got their urls*
- and since **meetup.com** is a "well-formatted" website (i.e. consistent in HTML/CSS structure)....
- it should be quick work to get the CSS (xpath) info we need to parse it

right-click-inspect-element (website #2)

- first, we're going to grab the info we need from the "members" page for each group:

right-click-inspect-element (website #2)

- first, we're going to grab the info we need from the "members" page for each group:

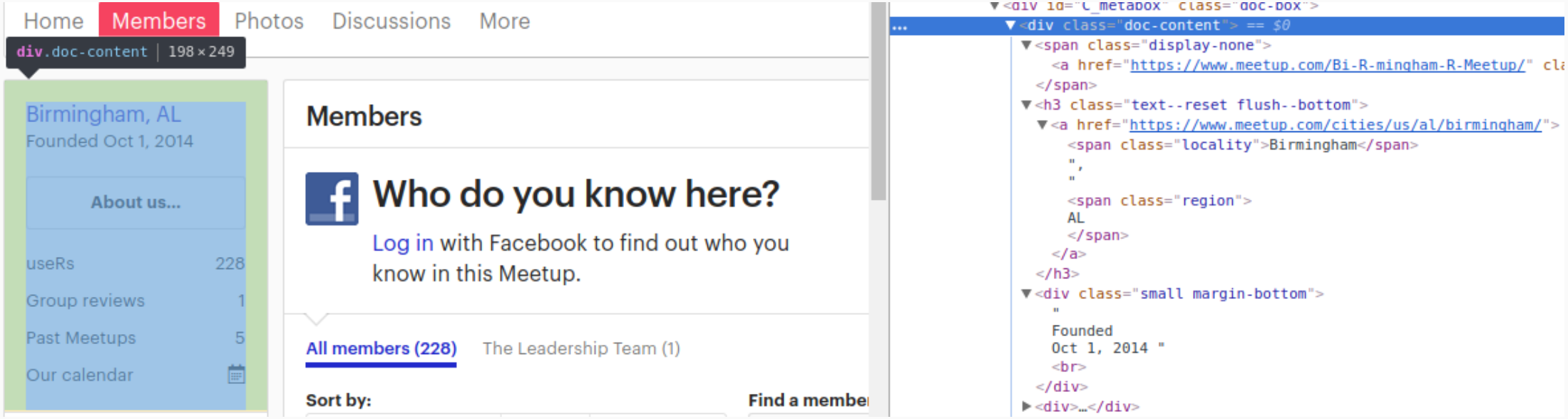
```
# open up one of the R user group URLs from RStudio
browseURL(paste0(mu_urls[1],'/members/'))
```

right-click-inspect-element (website #2)

- first, we're going to grab the info we need from the "members" page for each group:

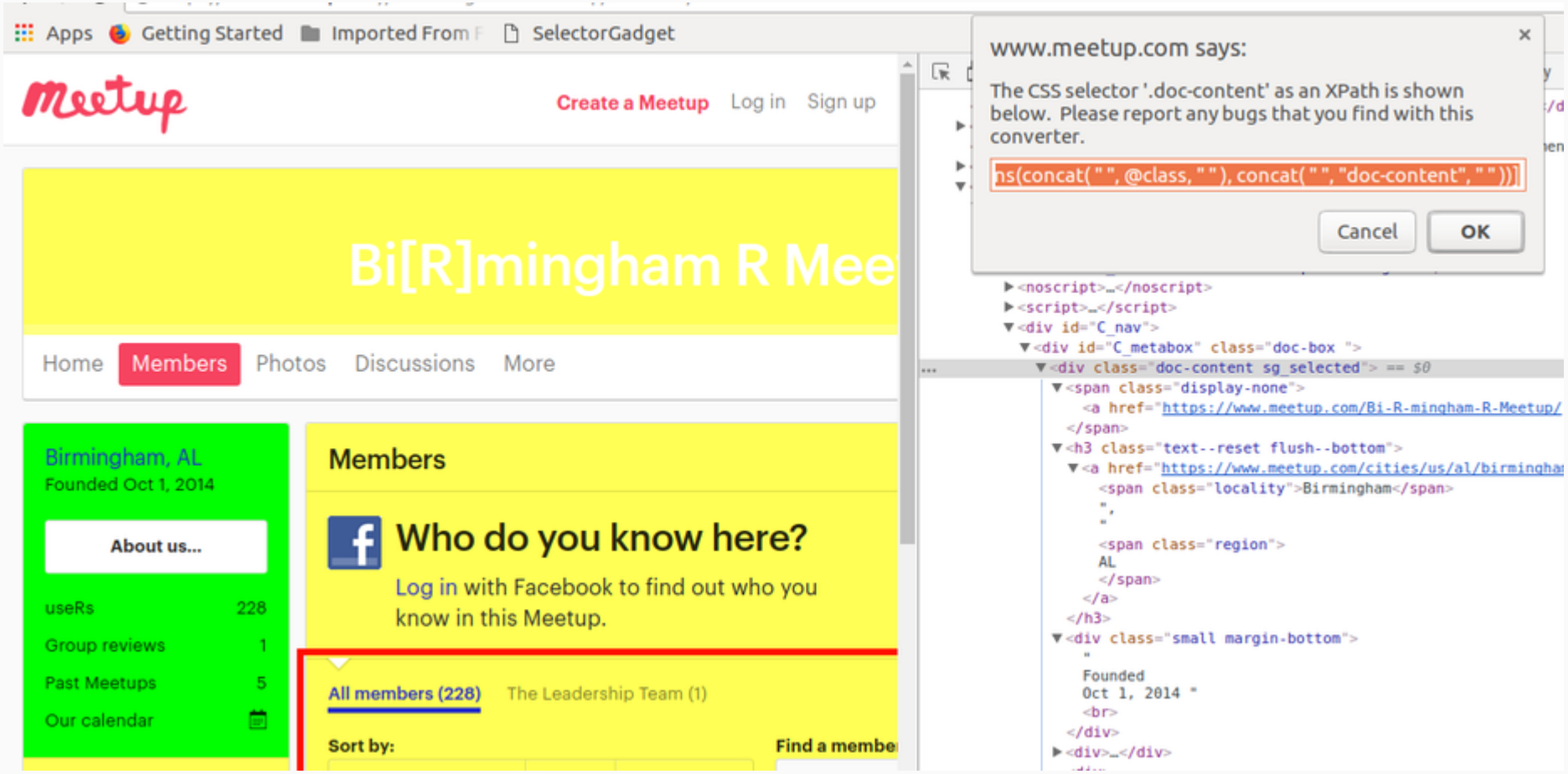
```
# open up one of the R user group URLs from RStudio
browseURL(paste0(mu_urls[1],'/members/'))
```

then, use selectorgadget and/or *right-click-inspect-method* to grab xpath to blue (or green) shaded box



selectorgadget method (website #2)

The gadget info box has the xpath to use (click xpath when green area is correctly set)



scrape meetup.com/meetings for all R User Groups

```
#xpath for to obtain group's attributes
```

```
#from the meetup.com/members page
```

```
mu_htm ← read_html(paste0(mu_urls[1],'/members/'))
```

```
mu_infoxp ← '//*[@id="C_metabox"]/div[1]'
```

```
mu_info ← html_nodes(mu_htm,xpath=mu_infoxp)
```

- We've got the info - now just parse it for....

scrape meetup.com/meetings for all R User Groups

```
#xpath for to obtain group's attributes

#from the meetup.com/members page

mu_htm ← read_html(paste0(mu_urls[1],'/members/'))

mu_infoxp ← '//*[@id="C_metabox"]/div[1]'

mu_info ← html_nodes(mu_htm,xpath=mu_infoxp)
```

- We've got the info - now just parse it for....
  - location (city/state), founded-date, nbr-of-mems, nbr-of-past-mtgs

scrape meetup.com/meetings for all R User Groups

```
#xpath for to obtain group's attributes

#from the meetup.com/members page

mu_htm ← read_html(paste0(mu_urls[1],'/members/'))

mu_infoxp ← '//*[@id="C_metabox"]/div[1]'

mu_info ← html_nodes(mu_htm,xpath=mu_infoxp)
```

- We've got the info - now just parse it for....
  - location (city/state), founded-date, nbr-of-mems, nbr-of-past-mtgs

```
## founded on date has this CSS

dob_css ← 'div.small.margin-bottom'

mu_dob ← gsub("Founded\n","",html_text(html_nodes(mu_info,css=dob_css),trim=T))
```

scrape meetup.com/meetings for all R User Groups

- continuing the parsing setup....

scrape meetup.com/meetings for all R User Groups

- continuing the parsing setup....

```
# CSS for citi and state
```

```
mu_city ← html_text(html_nodes(mu_info,css=".locality"))
```

```
mu_st ← gsub("\\n","",html_text(html_nodes(mu_info,css=".region")))
```

```
# CSS for (text of) "data" elements
```

```
mu_datacss ← 'ul.paddedList.small.margin-bottom'
```

scrape meetup.com/meetings for all R User Groups

putting it together: run code for scraping & parsing the metrics....

```
all_grpinfo ← matrix(nrow=length(mu_urls), ncol=6)
colnames(all_grpinfo) ← c('mu_city', 'mu_st', 'mu_dob', 'mu_nbrmems', 'mu_pstmtgs', 'mu_urls')

mu_infoxp ← '//*[@id="C_metabox"]/div[1]'  # xpath for selecting group attributes from the meetup.com/members page for a given R user group URL
dob_css ← 'div.small.margin-bottom'        # CSS locator for founded date for group
mu_datacss ← 'ul.paddedList.small.margin-bottom' # CSS for (text of) data elements

n ← 5 # scraping 1st 5 urls - see later footnote on wrapping up scrape websites that may limit HTML requests
for (i in 1:n) {
mu_htm ← read_html(paste0(mu_urls[i],'/members/'))
mu_info ← html_nodes(mu_htm,xpath=mu_infoxp)
mu_city ← html_text(html_nodes(mu_info,css=".locality"))
mu_st ← html_text(html_nodes(mu_info,css=".region")) %>%
         gsub("\\n","",.)

mu_dob ← html_text(html_nodes(mu_info,css=dob_css),trim=T) %>%
          gsub("Founded\n","",.)

mu_data ← html_nodes(mu_info,css=mu_datacss) %>%
           html_nodes(.,css="a") %>%
           html_text(.)
mu_nbrmems ← strsplit(mu_data[1],"\n")[[1]][2]
mu_pstmtgs ← strsplit(mu_data[grep("Past Meet", mu_data)],"\n")[[1]][3]

all_grpinfo[i,] ← cbind(mu_city, mu_st, mu_dob, mu_nbrmems, mu_pstmtgs, mu_urls[i])
}
```

sample output

OK - what does it look like?.....

```
options(width = 150)
all_grpinfo[1:n,]
```

```
##      mu_city        mu_st mu_dob         mu_nbrmems mu_pstmtgs mu_urls
## [1,] "Birmingham"   "AL"  "Oct 1, 2014"  "227"      "5"        "https://www.meetup.com/Bi-R-mingham-R-Meetup/"
## [2,] "Little Rock"  "AR"  "Mar 4, 2017"  "79"       "3"        "https://www.meetup.com/Central-Arkansas-R-User-Group/"
## [3,] "Oakland"      "CA"  "Jan 27, 2012" "1,552"    "77"       "https://www.meetup.com/r-enthusiasts/"
## [4,] "Chico"        "CA"  "Dec 7, 2015"  "29"       "25"       "https://www.meetup.com/Chico-R-Users-Group/"
## [5,] "Santa Monica" "CA"  "Mar 21, 2009" "1,694"    "85"       "https://www.meetup.com/LAarea-R-usergroup/"
```

sample output

OK - what does it look like?.....

```
options(width = 150)
all_grpinfo[1:n,]
```

```
##      mu_city        mu_st mu_dob         mu_nbrmems mu_pstmtgs mu_urls
## [1,] "Birmingham"   "AL"  "Oct 1, 2014"  "227"      "5"        "https://www.meetup.com/Bi-R-mingham-R-Meetup/"
## [2,] "Little Rock"  "AR"  "Mar 4, 2017"  "79"       "3"        "https://www.meetup.com/Central-Arkansas-R-User-Group/"
## [3,] "Oakland"      "CA"  "Jan 27, 2012" "1,552"    "77"       "https://www.meetup.com/r-enthusiasts/"
## [4,] "Chico"        "CA"  "Dec 7, 2015"  "29"       "25"       "https://www.meetup.com/Chico-R-Users-Group/"
## [5,] "Santa Monica" "CA"  "Mar 21, 2009" "1,694"    "85"       "https://www.meetup.com/LAarea-R-usergroup/"
```

NOTE: [1]: we won't be scraping all websites here, but see footnote for example of throttling read_html() when websites may limit traffic

sample output

OK - what does it look like?.....

```
options(width = 150)
all_grpinfo[1:n,]
```

```
##      mu_city        mu_st mu_dob         mu_nbrmems mu_pstmtgs mu_urls
## [1,] "Birmingham"   "AL"  "Oct 1, 2014"  "227"      "5"        "https://www.meetup.com/Bi-R-mingham-R-Meetup/"
## [2,] "Little Rock"  "AR"  "Mar 4, 2017"  "79"       "3"        "https://www.meetup.com/Central-Arkansas-R-User-Group/"
## [3,] "Oakland"      "CA"  "Jan 27, 2012" "1,552"    "77"       "https://www.meetup.com/r-enthusiasts/"
## [4,] "Chico"        "CA"  "Dec 7, 2015"  "29"       "25"       "https://www.meetup.com/Chico-R-Users-Group/"
## [5,] "Santa Monica" "CA"  "Mar 21, 2009" "1,694"    "85"       "https://www.meetup.com/LAarea-R-usergroup/"
```

NOTE: [1]: we won't be scraping all websites here, but see footnote for example of throttling read_html() when websites may limit traffic

[1] https://stackoverflow.com/questions/39056103/iterating-rvest-scrape-function-gives-error-in-open-connectionx-rb-time/39057166/

some MOTO conclusions...

- playing the role of *master-of-the-obvious*, what can we conclude?
    - "if" one wanted to, one could create a nice dataset/database of R User Groups *just* by parsing data from meetup.com

some MOTO conclusions...

- playing the role of *master-of-the-obvious*, what can we conclude?
  - "if" one wanted to, one could create a nice dataset/database of R User Groups *just* by parsing data from meetup.com
  - with just a bit more effort, and one could get the 'joined date' for every member (and calculate a *bunch* of stuff)

some MOTO conclusions...

- playing the role of *master-of-the-obvious*, what can we conclude?
    - "if" one wanted to, one could create a nice dataset/database of R User Groups *just* by parsing data from meetup.com
    - with just a bit more effort, and one could get the 'joined date' for every member (and calculate a *bunch* of stuff)
    - one could parse the blurbs in the group organizers and analyze/segment based on textmining (e.g. *which industry is their focus?*)

some MOTO conclusions...

- playing the role of *master-of-the-obvious*, what can we conclude?
    - "if" one wanted to, one could create a nice dataset/database of R User Groups *just* by parsing data from meetup.com
    - with just a bit more effort, and one could get the 'joined date' for every member (and calculate a *bunch* of stuff)
    - one could parse the blurbs in the group organizers and analyze/segment based on textmining (e.g. *which industry is their focus?*)

some MOTO conclusions...

- or, one could parse meetings for the topic/subject matter, and *number of attendees*...etc etc etc

some MOTO conclusions...

- or, one could parse meetings for the topic/subject matter, and *number of attendees*...etc etc etc
  - as a *pedagogical device*, one could create an animated *shiny app* showing growth of R over time and geography (*West Coast to East Coast?*)

some MOTO conclusions...

- or, one could parse meetings for the topic/subject matter, and *number of attendees*...etc etc etc
  - as a *pedagogical device*, one could create an animated *shiny app* showing growth of R over time and geography (*West Coast to East Coast?*)
  - and on and on and on.....(what about expanding to non-U.S. groups?)

Summary take-away....

- learning to scrape data from the web allows one to *create* and *analyze* datasets that simply do not exist anywhere...

Summary take-away....

- learning to scrape data from the web allows one to *create* and *analyze* datasets that simply do not exist anywhere... ....but in the rendering of webpages

Summary take-away....

- learning to scrape data from the web allows one to *create* and *analyze* datasets that simply do not exist anywhere... ....but in the rendering of webpages

And now, let's move on to scraping some XML-formatted data

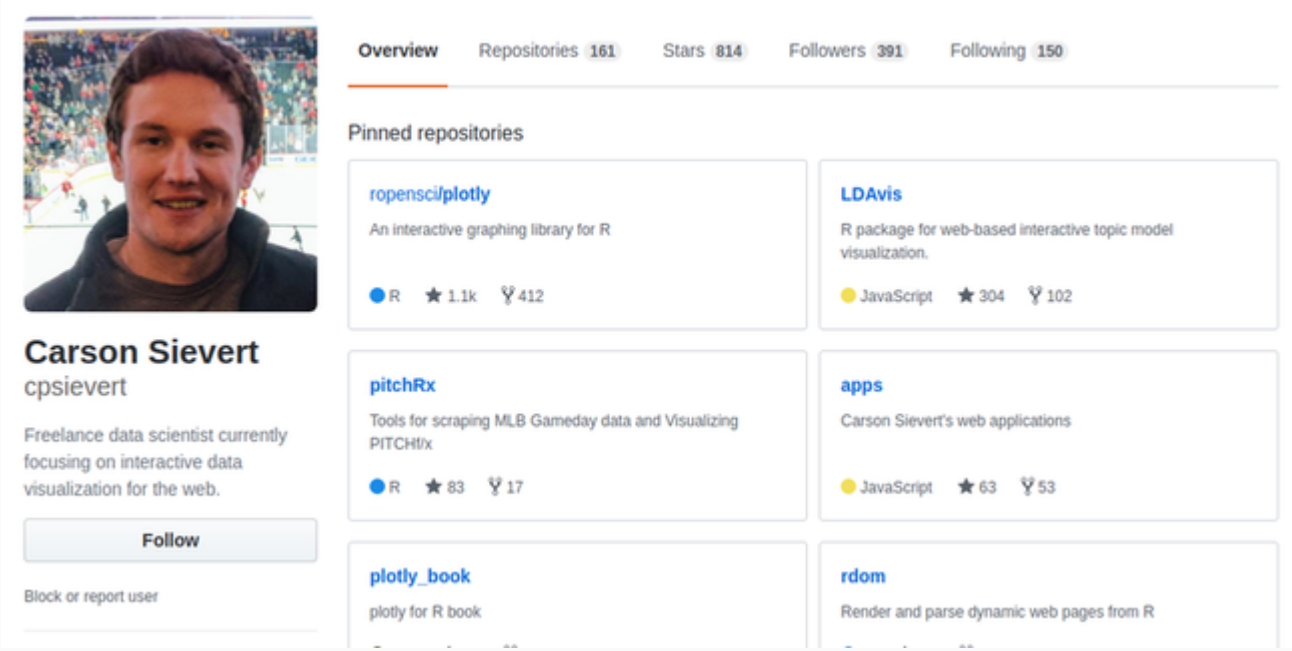
Scraping and Parsing XML

- *First*, let's mention one prolific contributor in the effort to demystify the scraping/manipulation of XML

Scraping and Parsing XML

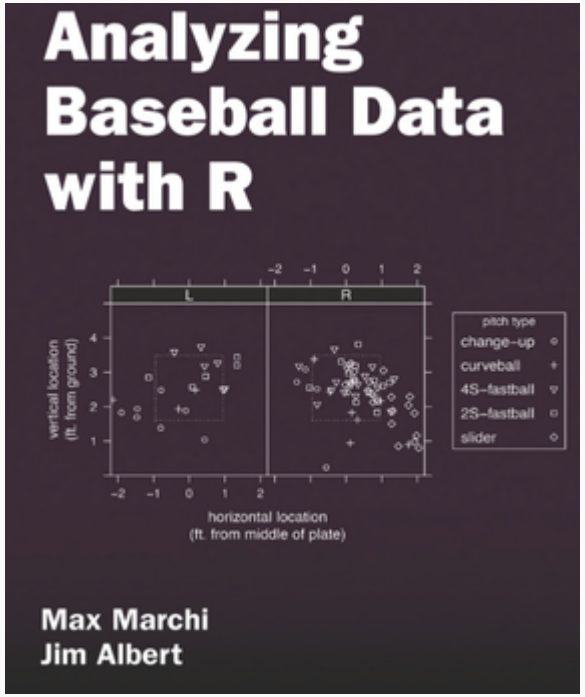
- *First*, let's mention one prolific contributor in the effort to **demystify** the scraping/manipulation of XML

- Carson Seivert wrote the *pitchRx* package (for scraping MLB's "gameday" XML data)



Scraping and Parsing XML

- as referenced in the book *Analyzing Baseball Data in R*



Scraping and Parsing XML

- If there's one way to learn how to deal with XML, it's in putting together your own database of EVERY pitch thrown in EVERY MLB game over the past x years!

Scraping and Parsing XML

- If there's one way to learn how to deal with XML, it's in putting together your own database of EVERY pitch thrown in EVERY MLB game over the past x years!
- Since Carson has provided detailed examples of using XML2R[2] in the context of *baseball*,

Scraping and Parsing XML

- If there's one way to learn how to deal with XML, it's in putting together your own database of EVERY pitch thrown in EVERY MLB game over the past x years!
- Since Carson has provided detailed examples of using XML2R[2] in the context of *baseball*, let's do something related to a *different* sport....

[2] https://xml2r.cpsievert.me/

Scraping and Parsing XML

In the spirit of ***March Madness***, let's grab some NCAA Basketball **"market"** data (i.e., Vegas Oddslines) from *wagertalk.com*

Scraping and Parsing XML

In the spirit of *March Madness*, let's grab some NCAA Basketball "market" data (i.e., Vegas Oddslines) from *wagertalk.com*

College Basketball
Tuesday, February 20th, 2018

Time	Gm#	(-) Team	Score	Opener	Public%	Bookmaker	Pincle	Grande	Bovada	Greek	5Dimes	BOnline
02/20 7:00p	501 502	Boston College NC State		159u-116 8	52% 52%	157 5-105	157 5-106	5-105	157 5	157 5-105	157 5	157½u-115 4½-115
02/20 7:00p	503 504	West Virginia Baylor		143u-115 1	51% 51%	1-115 140½	1-106 140½	1	1-115 141u-115	142 pk	1 140½	1-111 140½u-115

Scraping and Parsing XML

In the spirit of *March Madness*, let's grab some NCAA Basketball "market" data (i.e., Vegas Oddslines) from *wagertalk.com*

College Basketball
Tuesday, February 20th, 2018

Time	Gm#	(-) Team	Score	Opener	Public%	Bookmaker	Pincle	Grande	Bovada	Greek	5Dimes	BOnline
02/20	501	Boston College		159u-116	52%	157	157		157	157	157	157½u-115
7:00p	502	NC State		8	52%	5-105	5-106	5-105	5	5-105	5	4½-115
02/20	503	West Virginia		143u-115	51%	1-115	1-106	1	1-115	142	1	1-111
7:00p	504	Baylor		1	51%	140½	140½		141u-115	pk	140½	140½u-115

The above shows the odds/lines for NCAA BB as of around noon EST on Feb. 20th 2018....

Scraping and Parsing XML

Fortunately, that data is available in XML format here:

http://www.wagertalk.com/spt-opt/schedule.php?host=WAGERTALK&sport=ncaabb&period=0

```
- <ODDS>
    <SCHEDULE value="1519140318"/>
    <TIME value="1519146023" GMT_offset="8"/>
    <STARTED value="Servlet started at (1518975651)"/>
    <LOADED value="Schedule loaded at (1519140317)(1519140318)"/>
  - <LEAGUE number="4" name="College Basketball">
    - <GAME date="20180220" time="1600" seconds="1519113600">
      - <TEAM number="501" name="Boston College">
          <OPENER value="159u-116"/>
          <LINE book="1" value="157" seconds="5798"/>
          <LINE book="3" value="157" seconds="5207"/>
          <LINE book="5" value="157" seconds="5771"/>
          <LINE book="14" value="157" seconds="5742"/>
          <LINE book="16" value="157&frac12;u-115" seconds="5992"/>
          <LINE book="17" value="157" seconds="6027"/>
          <LINE book="22" value="157" seconds="5680"/>
          <LINE book="28" value="52%" seconds="1161"/>
          <LINE book="32" value=""/>
          <LINE book="42" value="157" seconds="5707"/>
          <LINE book="53" value="156&frac12;" seconds="748"/>
          <LINE book="57" value="157" seconds="5512"/>
          <LINE book="71" value="157&frac12;" seconds="5807"/>
          <LINE book="73" value=""/>
          <LINE book="163" value="157&frac12;" seconds="5422"/>
        </TEAM>
```

Scraping and Parsing XML

Fortunately, that data is available in XML format here:

http://www.wagertalk.com/spt-opt/schedule.php?host=WAGERTALK&sport=ncaabb&period=0

```
- <ODDS>
    <SCHEDULE value="1519140318"/>
    <TIME value="1519146023" GMT_offset="8"/>
    <STARTED value="Servlet started at (1518975651)"/>
    <LOADED value="Schedule loaded at (1519140317)(1519140318)"/>
  - <LEAGUE number="4" name="College Basketball">
    - <GAME date="20180220" time="1600" seconds="1519113600">
      - <TEAM number="501" name="Boston College">
          <OPENER value="159u-116"/>
          <LINE book="1" value="157" seconds="5798"/>
          <LINE book="3" value="157" seconds="5207"/>
          <LINE book="5" value="157" seconds="5771"/>
          <LINE book="14" value="157" seconds="5742"/>
          <LINE book="16" value="157&frac12;u-115" seconds="5992"/>
          <LINE book="17" value="157" seconds="6027"/>
          <LINE book="22" value="157" seconds="5680"/>
          <LINE book="28" value="52%" seconds="1161"/>
          <LINE book="32" value=""/>
          <LINE book="42" value="157" seconds="5707"/>
          <LINE book="53" value="156&frac12;" seconds="748"/>
          <LINE book="57" value="157" seconds="5512"/>
          <LINE book="71" value="157&frac12;" seconds="5807"/>
          <LINE book="73" value=""/>
          <LINE book="163" value="157&frac12;" seconds="5422"/>
        </TEAM>
```

.....now left's examine the XML hierarchy....

Scraping and Parsing XML

Fortunately, that data is available in XML format here:

http://www.wagertalk.com/spt-opt/schedule.php?host=WAGERTALK&sport=ncaabb&period=0


```
−<ODDS>
   <SCHEDULE value="1519140318"/>
   <TIME value="1519146023" GMT_offset="8"/>
   <STARTED value="Servlet started at (1518975651)"/>
   <LOADED value="Schedule loaded at (1519140317)(1519140318)"/>
 −<LEAGUE number="4" name="College Basketball">
  −<GAME date="20180220" time="1600" seconds="1519113600">
   −<TEAM number="501" name="Boston College">
      <OPENER value="159u-116"/>
      <LINE book="1" value="157" seconds="5798"/>
      <LINE book="3" value="157" seconds="5207"/>
      <LINE book="5" value="157" seconds="5771"/>
      <LINE book="14" value="157" seconds="5742"/>
      <LINE book="16" value="157&frac12;u-115" seconds="5992"/>
      <LINE book="17" value="157" seconds="6027"/>
      <LINE book="22" value="157" seconds="5680"/>
      <LINE book="28" value="52%" seconds="1161"/>
      <LINE book="32" value=""/>
      <LINE book="42" value="157" seconds="5707"/>
      <LINE book="53" value="156&frac12;" seconds="748"/>
      <LINE book="57" value="157" seconds="5512"/>
      <LINE book="71" value="157&frac12;" seconds="5807"/>
      <LINE book="73" value=""/>
      <LINE book="163" value="157&frac12;" seconds="5422"/>
    </TEAM>
```

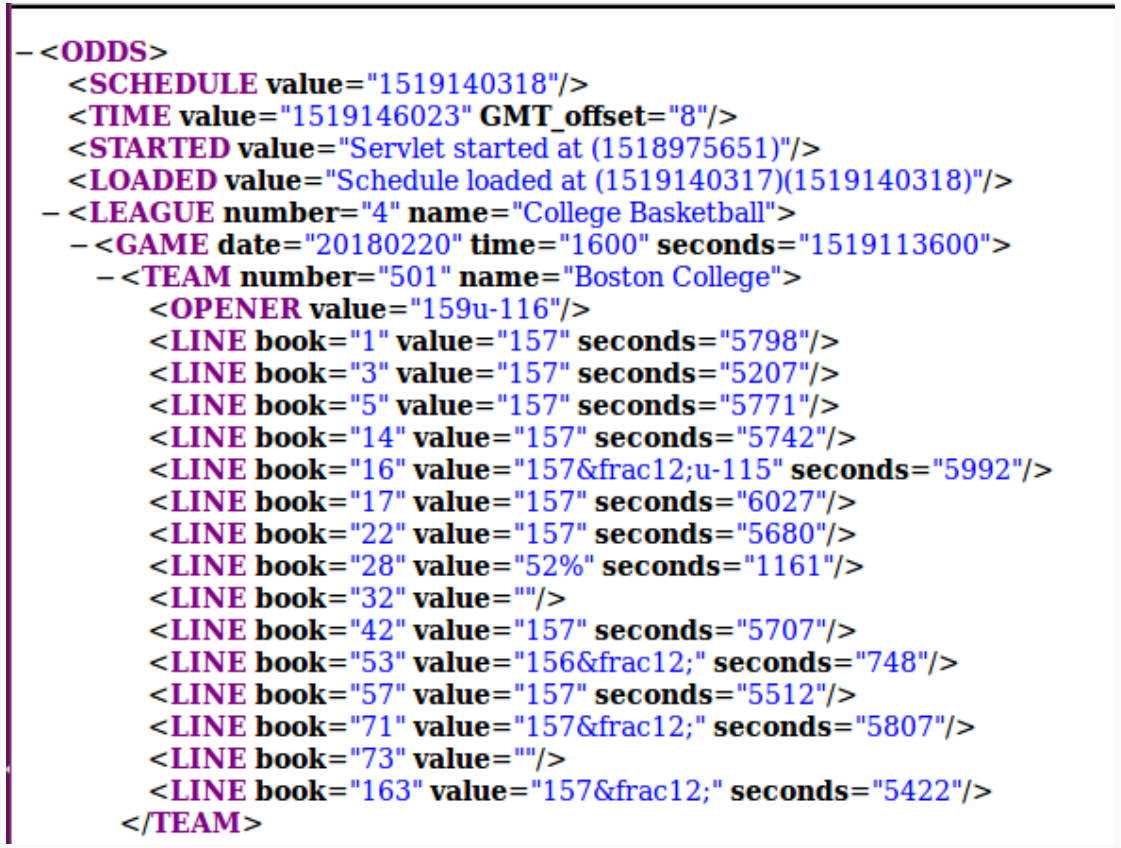

.....now left's examine the XML hierarchy....

.....before running some simple code....

Examining the XML structure

XML Hierarchy

1. open the file and example the structure
   - expand and collapse the XML hierarchy to geta feel for how the parent/child relationships are formatted
2. think of the XML in terms of *facts* and *dimensions*[3], and in terms of 'global' information.
   - in this case, the (my phrase) 'global' tags are SCHEDULE, TIME, STARTED, LOADED, LEAGUE
3. each GAME node has associated date/datetime elements, but also has 2 TEAM child nodes....
4. finally, TEAM node has set of LINE tags
   - LINE tags relate to the line's "provider" (bookmaker)
   - one set of LINEs relate to "point spread", and the other relates to game totals (OVER/UNDER)



[3] https://en.wikipedia.org/wiki/Dimensional_modeling

Why XML2R?

Why XML2R?

XML2R makes scraping/parsing XML 'easy' because it....

Why XML2R?

XML2R makes scraping/parsing XML 'easy' because it....

- reduces the effort required to transform XML into tables

Why XML2R?

XML2R makes scraping/parsing XML 'easy' because it....

- reduces the effort required to transform XML into tables
- and does so while preserving parent to child relationships
  - (which is *why* we examined the XML structure as a first step)

Why XML2R?

XML2R makes scraping/parsing XML 'easy' because it....

- reduces the effort required to transform XML into tables
- and does so while preserving parent to child relationships
  - (which is *why* we examined the XML structure as a first step)
- for this exercise, we'll use just 3 functions:

Why XML2R?

XML2R makes scraping/parsing XML 'easy' because it....

- reduces the effort required to transform XML into tables
- and does so while preserving parent to child relationships
  - (which is *why* we examined the XML structure as a first step)
- for this exercise, we'll use just 3 functions:
  - XML2R::XML2Obs
  - XML2R::add_key
  - XML2R::collapse_obs

Parsing the XML via XML2R functions

let's use XML2R::XML2Obs against the oddsline URL and look at what it returns

Parsing the XML via XML2R functions

let's use XML2R::XML2Obs against the oddsline URL and look at what it returns

```
library(XML2R)
xml1 ← "http://www.wagertalk.com/spt-opt/schedule.php?host=WAGERTALK&sport=ncaabb&period=0"
xmlobs1 ← XML2Obs(xml1, quiet=TRUE)
table(names(xmlobs1))
```

```
##
##                     ODDS//LEAGUE              ODDS//LEAGUE//GAME        ODDS//LEAGUE//GAME//TEAM   ODDS//LEAGUE//GAME//TEAM//LINE
##                                1                              71                             142                              1608
## ODDS//LEAGUE//GAME//TEAM//OPENER                 ODDS//LOADED                 ODDS//SCHEDULE                 ODDS//STARTED
##                              142                               1                               1                               1
##                       ODDS//TIME
##                                1
```

Parsing the XML via XML2R functions

let's use XML2R::XML2Obs against the oddsline URL and look at what it returns

```
library(XML2R)
xml1 ← "http://www.wagertalk.com/spt-opt/schedule.php?host=WAGERTALK&sport=ncaabb&period=0"
xmlobs1 ← XML2Obs(xml1, quiet=TRUE)
table(names(xmlobs1))
```

```
##
##                     ODDS//LEAGUE              ODDS//LEAGUE//GAME        ODDS//LEAGUE//GAME//TEAM   ODDS//LEAGUE//GAME//TEAM//LINE
##                                 1                              71                             142                              1608
## ODDS//LEAGUE//GAME//TEAM//OPENER                ODDS//LOADED                ODDS//SCHEDULE                 ODDS//STARTED
##                               142                               1                               1                                 1
##                       ODDS//TIME
##                                 1
```

After having perused the XML file by collapsing/expanding nodes, the output *makes sense*....

- each GAME has **2** TEAMS

Parsing the XML via XML2R functions

let's use XML2R::XML2Obs against the oddsline URL and look at what it returns

```
library(XML2R)
xml1 <- "http://www.wagertalk.com/spt-opt/schedule.php?host=WAGERTALK&sport=ncaabb&period=0"
xmlobs1 <- XML2Obs(xml1, quiet=TRUE)
table(names(xmlobs1))
```

```
##
##                     ODDS//LEAGUE              ODDS//LEAGUE//GAME        ODDS//LEAGUE//GAME//TEAM   ODDS//LEAGUE//GAME//TEAM//LINE
##                                1                              71                             142                              1608
## ODDS//LEAGUE//GAME//TEAM//OPENER                ODDS//LOADED                ODDS//SCHEDULE                 ODDS//STARTED
##                              142                               1                               1                               1
##                       ODDS//TIME
##                                1
```

After having perused the XML file by collapsing/expanding nodes, the output *makes sense*....

- each GAME has **2** TEAMS
- the 'global' information is associated with a **single** count of a node ('time', 'loaded', etc)

Parsing the XML via XML2R functions

let's use XML2R::XML2Obs against the oddsline URL and look at what it returns

```
library(XML2R)
xml1 <- "http://www.wagertalk.com/spt-opt/schedule.php?host=WAGERTALK&sport=ncaabb&period=0"
xmlobs1 <- XML2Obs(xml1, quiet=TRUE)
table(names(xmlobs1))
```

```
##
##                     ODDS//LEAGUE              ODDS//LEAGUE//GAME        ODDS//LEAGUE//GAME//TEAM   ODDS//LEAGUE//GAME//TEAM//LINE
##                                 1                              71                             142                              1608
## ODDS//LEAGUE//GAME//TEAM//OPENER                ODDS//LOADED                ODDS//SCHEDULE                 ODDS//STARTED
##                               142                               1                               1                                 1
##                       ODDS//TIME
##                                 1
```

After having perused the XML file by collapsing/expanding nodes, the output *makes sense*....

- each GAME has **2** TEAMS
- the 'global' information is associated with a **single** count of a node ('time', 'loaded', etc)
- now, we'll make sure the PARENT node data is passed to the CHILD nodes (before creating data.frame type output)

mapping parent-to-child (via XML2R::add_key)

Remember - the LINE nodes are the "facts", and we need to be able to map those facts to their parent attributes...

mapping parent-to-child (via XML2R::add_key)

Remember - the LINE nodes are the "facts", and we need to be able to map those facts to their parent attributes...

- under GAME, we want the "date" and the "time" values passed down

mapping parent-to-child (via XML2R::add_key)

Remember - the LINE nodes are the "facts", and we need to be able to map those facts to their parent attributes...

- under GAME, we want the "date" and the "time" values passed down

```
# adding key - new col of "gamedate"
```

```
xmlobs1 ← add_key(xmlobs1, parent="ODDS//LEAGUE//GAME", recycle="date", key.name="gamedate")
```

```
# adding key - new col of "gametime"
```

```
xmlobs1 ← add_key(xmlobs1, parent="ODDS//LEAGUE//GAME", recycle="time", key.name="gametime")
```

mapping parent-to-child (via XML2R::add_key)

Remember - the LINE nodes are the "facts", and we need to be able to map those facts to their parent attributes...

- under GAME, we want the "date" and the "time" values passed down

```
# adding key - new col of "gamedate"
xmlobs1 ← add_key(xmlobs1, parent="ODDS//LEAGUE//GAME", recycle="date", key.name="gamedate")
# adding key - new col of "gametime"
xmlobs1 ← add_key(xmlobs1, parent="ODDS//LEAGUE//GAME", recycle="time", key.name="gametime")
```

- under GAME//TEAM, we want the (team) "name" and the (game-wager) "number" passed down

mapping parent-to-child (via XML2R::add_key)

Remember - the LINE nodes are the "facts", and we need to be able to map those facts to their parent attributes...

- under GAME, we want the "date" and the "time" values passed down

```
# adding key - new col of "gamedate"
xmlobs1 ← add_key(xmlobs1, parent="ODDS//LEAGUE//GAME", recycle="date", key.name="gamedate")
# adding key - new col of "gametime"
xmlobs1 ← add_key(xmlobs1, parent="ODDS//LEAGUE//GAME", recycle="time", key.name="gametime")
```

- under GAME//TEAM, we want the (team) "name" and the (game-wager) "number" passed down

```
# new col of "gamewgrnbr"
xmlobs1 ← add_key(xmlobs1, parent="ODDS//LEAGUE//GAME//TEAM", recycle="number", key.name="gamewgrnbr")
# new col of "teamname"
xmlobs1 ← add_key(xmlobs1 , parent="ODDS//LEAGUE//GAME//TEAM", recycle="name", key.name="teamname")
```

collapsing observations (via XML2R::collapse_obs)

Now, we can simply use XML2R::collapse_obs to get a matrix

collapsing observations (via XML2R::collapse_obs)

Now, we can simply use XML2R::collapse_obs to get a matrix

```
#collapse observations from the ....//LINE nodes

#and remove the "url" colname (don't need it)

oddsline ← collapse_obs(xmlobs1[grep("^ODDS//LEAGUE//GAME//TEAM//LINE$",names(xmlobs1))])

oddsline ← oddsline[,-grep("url",colnames(oddsline))]
```

run code and show output

We'll just run the previously shown code and show some output to verify all 3 XML2R functions we've used did their job...

run code and show output

We'll just run the previously shown code and show some output to verify all 3 XML2R functions we've used did their job...

```
xmlobs1 ← add_key(xmlobs1, parent="ODDS//LEAGUE//GAME", recycle="date", key.name="gamedate") # adding key - new col of "gamedate"
xmlobs1 ← add_key(xmlobs1, parent="ODDS//LEAGUE//GAME", recycle="time", key.name="gametime") # adding key - new col of "gametime"
xmlobs1 ← add_key(xmlobs1, parent="ODDS//LEAGUE//GAME//TEAM", recycle="number", key.name="gamewgrnbr") # new col of "gamewgrnbr"
xmlobs1 ← add_key(xmlobs1 , parent="ODDS//LEAGUE//GAME//TEAM", recycle="name", key.name="teamname")
oddsline ← collapse_obs(xmlobs1[grep("^ODDS//LEAGUE//GAME//TEAM//LINE$",names(xmlobs1))])  # this returns a matrix
oddsline ← oddsline[,-grep("url",colnames(oddsline))]            # get rid of the long "url" colname
head(oddsline)
```

```
##      book value         seconds gamedate   gametime gamewgrnbr teamname
## [1,] "1"  "132"         "834"   "20180222" "1600"   "563"      "Connecticut"
## [2,] "3"  "131&frac12;" "13546" "20180222" "1600"   "563"      "Connecticut"
## [3,] "5"  "132"         "767"   "20180222" "1600"   "563"      "Connecticut"
## [4,] "14" "132"         "797"   "20180222" "1600"   "563"      "Connecticut"
## [5,] "16" "131o-115"    "1024"  "20180222" "1600"   "563"      "Connecticut"
## [6,] "17" "132"         "840"   "20180222" "1600"   "563"      "Connecticut"
```

Post-processing: task list

Here's a list of things to do to get the data into a "usable" data frame....

Post-processing: task list

Here's a list of things to do to get the data into a "usable" data frame....

We won't go through them with code, but a screenshot of the *final persistance* to a mysql table will be shown

Post-processing: task list

Here's a list of things to do to get the data into a "usable" data frame....

We won't go through them with code, but a screenshot of the *final persistance* to a mysql table will be shown

- parse the lines for half-points (gsub the "ampersand frac12")
- perform necessary character-to-numeric conversions
- ...and then for dates/datetimes, convert to appropriate POSIX.. classes
- get the "datetime-line-refresh" from the global //TIME attribute and...
- use that with the "seconds" value to calculate the datetime for a given "LINE"
- calculate an "hours-to-gametime" column (optional - but easier to digest!)
- load to a database table (mysql is my preference) for tracking line movements over time

Final resting place (for the parsed & processed XML)

sportlinetype_x	book	gamedttm	game_id	game_name	gamewgri	game	teamname_x	teamname_y	away_	home_	away_ps_	home_ps_	favetes	favet	ps_dttmlinevalue	pshrs2gt
ncaabb_ps	1	2017-11-28 18:30:00	511_512	Baylor_@_Xavier	511	512	Baylor	Xavier	4.5	-4.5	-110	-110	1	0	2017-11-27 17:59:22	24.5
ncaabb_ps	1	2017-11-28 19:00:00	513_514	Appalachian State_@_VA Co...	513	514	Appalachian State	VA Commo...	11	-11	-110	-110	1	0	2017-11-27 17:42:54	25.3
ncaabb_ps	1	2017-11-28 19:00:00	515_516	Brown_@_Rhode Island	515	516	Brown	Rhode Island	20	-20	-110	-110	1	0	2017-11-27 17:43:04	25.3
ncaabb_ps	1	2017-11-28 19:00:00	517_518	Florida State_@_Rutgers	517	518	Florida State	Rutgers	-4.5	4.5	-111	-109	0	1	2017-11-27 17:43:41	25.3

Notice the "pshr2gt" column...(point-spread-hours-to-gametime)

Final resting place (for the parsed & processed XML)

sportlinetype_x	book	gamedttm	game_id	game_name	gamewgri	game	teamname_x	teamname_y	away_	home_	away_ps_	home_ps_	favetes	favet	ps_dttmlinevalue	pshrs2gt
ncaabb_ps	1	2017-11-28 18:30:00	511_512	Baylor_@_Xavier	511	512	Baylor	Xavier	4.5	-4.5	-110	-110	1	0	2017-11-27 17:59:22	24.5
ncaabb_ps	1	2017-11-28 19:00:00	513_514	Appalachian State_@_VA Co...	513	514	Appalachian State	VA Commo...	11	-11	-110	-110	1	0	2017-11-27 17:42:54	25.3
ncaabb_ps	1	2017-11-28 19:00:00	515_516	Brown_@_Rhode Island	515	516	Brown	Rhode Island	20	-20	-110	-110	1	0	2017-11-27 17:43:04	25.3
ncaabb_ps	1	2017-11-28 19:00:00	517_518	Florida State_@_Rutgers	517	518	Florida State	Rutgers	-4.5	4.5	-111	-109	0	1	2017-11-27 17:43:41	25.3

Notice the "pshr2gt" column...(point-spread-hours-to-gametime)

- that's from adding "seconds" to the global "TIME" attribute and subtracting game-datetime

Final resting place (for the parsed & processed XML)

sportlinetype_x	book	gamedttm	game_id	game_name	gamewgri	game	teamname_x	teamname_y	away_	home_	away_ps_	home_ps_	favetes	favet	ps_dttmlinevalue	pshrs2gt
ncaabb_ps	1	2017-11-28 18:30:00	511_512	Baylor_@_Xavier	511	512	Baylor	Xavier	4.5	-4.5	-110	-110	1	0	2017-11-27 17:59:22	24.5
ncaabb_ps	1	2017-11-28 19:00:00	513_514	Appalachian State_@_VA Co...	513	514	Appalachian State	VA Commo...	11	-11	-110	-110	1	0	2017-11-27 17:42:54	25.3
ncaabb_ps	1	2017-11-28 19:00:00	515_516	Brown_@_Rhode Island	515	516	Brown	Rhode Island	20	-20	-110	-110	1	0	2017-11-27 17:43:04	25.3
ncaabb_ps	1	2017-11-28 19:00:00	517_518	Florida State_@_Rutgers	517	518	Florida State	Rutgers	-4.5	4.5	-111	-109	0	1	2017-11-27 17:43:41	25.3

Notice the "pshr2gt" column...(point-spread-hours-to-gametime)

- that's from adding "seconds" to the global "TIME" attribute and subtracting game-datetime

Moving on - knowing when it's time to use RSelenium

Final resting place (for the parsed & processed XML)

sportlinetype_x	book	gamedttm	game_id	game_name	gamewgri	game	teamname_x	teamname_y	away_	home_	away_ps_	home_ps_	favetes	favet	ps_dttmlinevalue	pshrs2gt
ncaabb_ps	1	2017-11-28 18:30:00	511_512	Baylor_@_Xavier	511	512	Baylor	Xavier	4.5	-4.5	-110	-110	1	0	2017-11-27 17:59:22	24.5
ncaabb_ps	1	2017-11-28 19:00:00	513_514	Appalachian State_@_VA Co...	513	514	Appalachian State	VA Commo...	11	-11	-110	-110	1	0	2017-11-27 17:42:54	25.3
ncaabb_ps	1	2017-11-28 19:00:00	515_516	Brown_@_Rhode Island	515	516	Brown	Rhode Island	20	-20	-110	-110	1	0	2017-11-27 17:43:04	25.3
ncaabb_ps	1	2017-11-28 19:00:00	517_518	Florida State_@_Rutgers	517	518	Florida State	Rutgers	-4.5	4.5	-111	-109	0	1	2017-11-27 17:43:41	25.3

Notice the "pshr2gt" column...(point-spread-hours-to-gametime)

- that's from adding "seconds" to the global "TIME" attribute and subtracting game-datetime

  Moving on - knowing when it's time to use RSelenium

- (or its easy-to-use wrapper form via the rdom package)

Intro to RSelenium