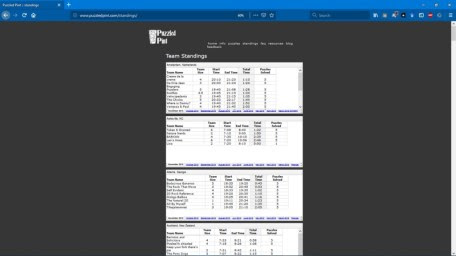
Photo by [freestocks.org](https://unsplash.com/photos/ZArDeAtxj0Q?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) on [Unsplash](https://unsplash.com/search/photos/web-scraping?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText" \t "_blank)

I love to learn new things and one of ways I learn best is by doing. Also it’s been said that [you never fully understand a topic until you are able to explain it](https://kottke.org/17/06/if-you-cant-explain-something-in-simple-terms-you-dont-understand-it), I think blogging is a low barrier to explaining things.

Someone I met at a local data science meetup in Montréal wanted help web scraping to get team standings from the [PuzzledPint](http://www.puzzledpint.com/standings/). I jumped at the opportunity because I knew this would be my opportunity to finally learn [**RSelenium!**](https://github.com/ropensci/RSelenium)



**Static Scraping vs. Dynamic Scraping**

Static scraping ignores JavaScript. It fetches web pages from the server without the help of a browser. You get exactly what you see in “view page source”, and then you slice and dice it. If the content you’re looking for is available, you need to go no further. However, if the content is something like an `iframe`, you need dynamic scraping.

Dynamic scraping uses an actual browser (or a headless browser) and lets JavaScript do its thing. Then, it queries the DOM to extract the content it’s looking for. Sometimes you need to automate the browser by simulating a user to get the content you need. In order for me to get the same details for the remaining posts, i would need to first navigate to the next page, which involves clicking the Next button at the bottom of the search results page.

**Setting up RSelenium with Docker**

RSelenium provides **R** bindings for the Selenium Webdriver API. [Selenium](http://docs.seleniumhq.org/) is a project focused on automating web browsers.

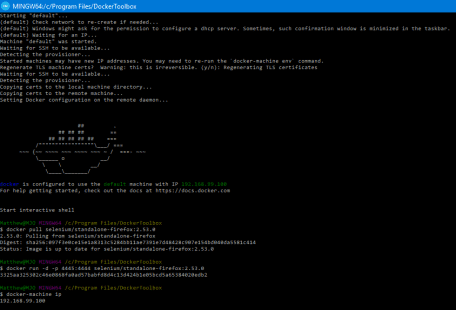
You need to follow the [installation instructions for Docker Toolbox on Windows or Ubuntu](https://docs.docker.com/toolbox/toolbox_install_windows/).



Docker is used to run applications by using containers. Containers are simply a bundle of libraries and other dependencies in one package. You can think of it like a virtual machine, but rather than creating a whole OS it allows applications to use the same Linux kernel with only the *things* not already running on the host computer. Basically, it gives a significant performance boost and reduces the size of the application. Moreover, you can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

You’ll also need to [install TightVNC](http://www.tightvnc.com/download.php) which will allow you to see how you’re manipulating the web page in real-time with RSelenium.

Next follow the [instructions to create a Docker container running a selenium server and its own firefox](http://ropensci.github.io/RSelenium/articles/docker.html).



*Note: Once you’ve set up the docker container (and everytime you restart your computer or start-up fresh again) open the Docker Quickstart Terminal and run the following command.*

docker run -d -p 4445:4444 selenium/standalone-firefox:2.53.0

Now that you’ve booted your **Docker Quickstart Terminal** go into **R** and connect to a running server.

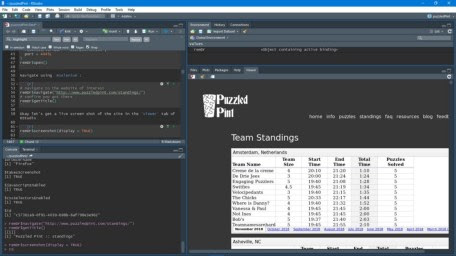
library(RSelenium)  
remDr <- remoteDriver(  
 remoteServerAddr = "192.168.99.100",  
 port = 4445L  
)  
remDr$open()

Navigate to the page using Rselenium.

# navigate to the website of interest  
remDr$navigate("<http://www.puzzledpint.com/standings/>")  
# confirm you got there  
remDr$getTitle()0

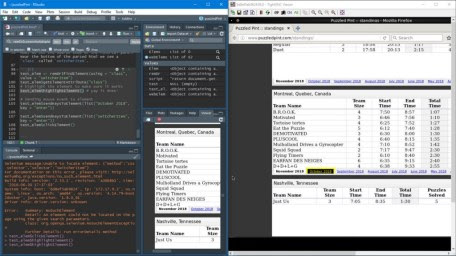
Okay let’s get a live screen shot of the site in the **Viewer** tab of RStudio.

remDr$screenshot(display = TRUE)



Keep in mind that this is just a static screen shot. Your going to want to use **TightVNC** to get a live view of your interactions while your developing your pipeline so you can see how your interacting with the website.

It’s important to be watching TightVNC as you use the …$highlightElement() in-between your …$findElement() and …$switchToFrame()/…$clickElement() commands so that you actually know your selecting the appropriate things!



Open the **TighVNC Viewer** and enter the port number; in this case 192.168.99.100 and enter that in the **Remote Host:** field. Click **Connect** and for the password enter the word: secret.

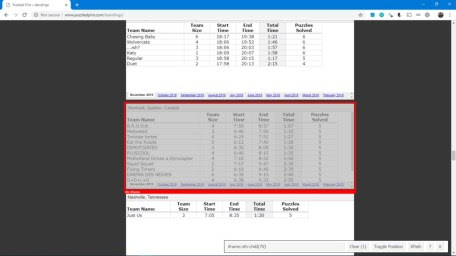
Note: If TightVNC ever stops working (on Windows 10 it did often) and gives you the Error: “NO CONNECTION COULD BE MADE BECAUSE THE TARGET MACHINE ACTIVELY REFUSED IT” then follow the steps for [“Debugging Using VNC” here](http://ropensci.github.io/RSelenium/articles/docker.html).

**Accessing Elements in the DOM**

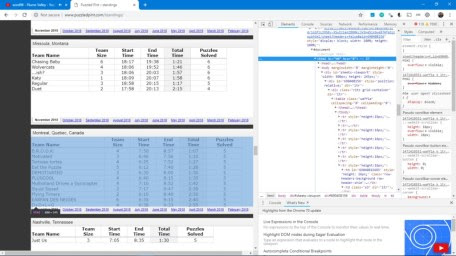
Web pages are a set of nest objects (together, they are known as the **Document Object Model** or **DOM** for short). It’s a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Interacting with the DOM will be very important for us with RSelenium.

Hadley Wickham recommends using [Selectorgadget](http://selectorgadget.com/), a Chrome extension, to help identify the web page elements you need. And he recommends [this page](http://flukeout.github.io/) for learning more about selectors.

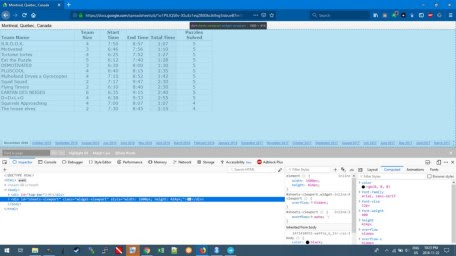
For example by using SelectorGadget you can the table you’re interested in. In this case it says it’s an **iframe**. To isolate just the Montréal standings we’ll click another box to only select the one of interest: iframe:nth-child(68).



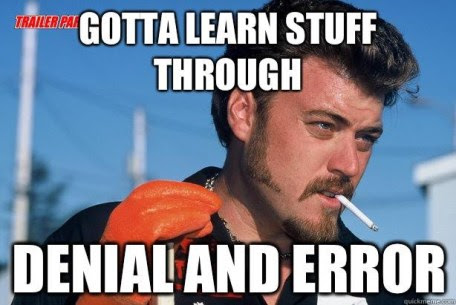
In the context of a web browser, a frame is a part of a web page or browser window which displays content independent of its container, with the ability to load content independently. In this case the website is pulling in data from another source to display these tables interactively apart from the main standings page. Luckily for me they are all from Google Sheets so this will make it *much* easier for me. Unfortunately, you can’t find the links for these sheets with `selectorgadget`. You will need to take a closer look at the source code using the Developer’s Tool called `Inspector` in either Chrome of Firefox. If you have Windows and Firefox you would click **Open Menu** then Web Developer > Inspector or just Ctrl+Shift+c. Then I used the search box to look for the link (src=) for Montréal.



For me it was a big pain to actually find what I was looking for as sometimes the highlight looks like what you want but it’s not. For example:



In the end I guess to figure out HTML it involves gradual “**Denial and Error**” attempts.



**Webscraping Google Sheets with RSelenium**

**Legal Disclaimer**

It’s worth mentioning that administrators may want to protect certain parts of their website for a [number of reasons](http://www.robotstxt.org/norobots-rfc.txt), such as “*indexing of an unannounced site, traversal of parts of the site which require vast resources of the server, recursive traversal of an infinite URL space, etc.*”

Therefore, one should always check if they have permission. One way to do this, is to use the robotstxt package to check if your web-bot has permission to access certain parts of a web-page.

# check permissions  
library(robotstxt)  
paths\_allowed("<https://docs.google.com/spreadsheets/d/1o1PlLIQS8v-XSuEz1eqZB80kcJk9xg5lsbueB7mTg1U/pub?output=html&widget=true#gid=690408156>")

If it says **TRUE** on the specific page you have permission. Alternatively, just go to the robots.txt file on the url of the main page to get a broader sense of what is (and isn’t) allowed.



**Method # 1**

Sometimes websites can be composed using frames. These are in effect seperate webpages which are brought together in a frameset. We will need to jump back-and-forth between these frames.

library(RSelenium)  
library(XML)  
library(janitor)  
library(lubridate)  
library(magrittr)  
library(dplyr)

remDr <- remoteDriver(  
 remoteServerAddr = "192.168.99.100",  
 port = 4445L  
)  
remDr$open()

# Now open TightVNC to follow along with Selenium driving the browser. Set 192.168.99.100:5901 and password: secret

# navigate to the main page  
remDr$navigate("<https://docs.google.com/spreadsheets/d/1o1PlLIQS8v-XSuEz1eqZB80kcJk9xg5lsbueB7mTg1U/pub?output=html&widget=true#gid=690408156>")

# <https://docs.google.com/spreadsheets/d/1o1PlLIQS8v-XSuEz1eqZB80kcJk9xg5lsbueB7mTg1U/pub?output=html&widget=true#gid=552987877>

# look for table element  
tableElem <- remDr$findElement(using = "id", "pageswitcher-content")

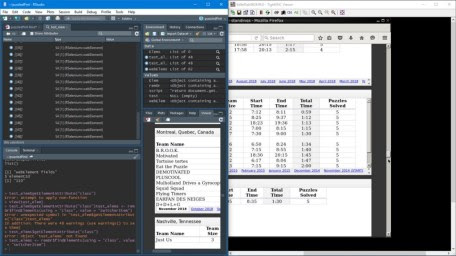
# switch to table  
remDr$switchToFrame(tableElem)  
# parse html  
doc <- htmlParse(remDr$getPageSource()[[1]])  
table\_tmp <- readHTMLTable(doc)  
table\_tmp <- table\_tmp[[1]][-2,-1]  
table\_tmp <- table\_tmp[-1,]  
colnames(table\_tmp) <- c("team\_name", "team\_size", "start\_time", "end\_time", "total\_time", "puzzels\_solved")  
table\_tmp$city <- rep("montreal", nrow(table\_tmp))  
table\_tmp$date <- rep(Sys.Date()-5, nrow(table\_tmp))

Now that we have the first month we can create a for loop for the rest of the dates. First let’s switch back the outer frame and select the elements we will be manipulating.

# switch back to the main/outter frame  
remDr$switchToFrame(NULL)

# find the elements you'll manipulate with Inspector mode in a browser  
webElems <- remDr$findElements(using = "css", ".switcherItem") # Month/Year tabs at the bottom  
arrowElems <- remDr$findElements(using = "css", ".switcherArrows") # Arrows to scroll left and right at the bottom  
tableElem <- remDr$findElement(using = "id", "pageswitcher-content") # The inner table frame

So I know there’s going to be many tables, but just how many? We can check this use this via length(webElems).



There is actually 49 tables in total but since we started on the first one above there is only 48 links. Rather than hard-coding 1:48 it’s better to do it via code as there will be more tables added in the future.

# Create NULL object to be used in forloop  
big\_df <- NULL  
for (i in seq(length(webElems))){ # for every   
check <- try(expression, silent = TRUE) # or suppressMessages(try(expression, silent = TRUE))  
if (any(class(check) == "try-error")) {  
 # choose the i'th Month/Year tab   
 webElem <- webElems[[i]]  
 webElem$clickElement()

# Find the tableElem again other wise you get a StaleElementReference   
## TO DO: look into WebDriverWait: <https://stackoverflow.com/questions/5709204/random-element-is-no-longer-attached-to-the-dom-staleelementreferenceexception>   
tableElem <- remDr$findElement(using = "id", "pageswitcher-content") # The inner table frame

# switch to table frame  
remDr$switchToFrame(tableElem)  
Sys.sleep(3)  
# parse html with XML package  
doc <- htmlParse(remDr$getPageSource()[[1]])  
Sys.sleep(3)  
# Extract data from HTML table in HTML doucment  
table\_tmp <- readHTMLTable(doc)  
Sys.sleep(3)  
# put this into a format you can use  
table <- table\_tmp[[1]][-2,-1]  
table <- table[-1,]  
# rename the columns  
colnames(table) <- c("team\_name", "team\_size", "start\_time", "end\_time", "total\_time", "puzzels\_solved")  
# add city name to a column  
table$city <- rep("Montreal", nrow(table))

# add the Month/Year this table was extracted from  
today <- Sys.Date() %m-% months(i + 1)  
table$date <- today

# concatenate each table together  
big\_df <- dplyr::bind\_rows(big\_df, table)

# Switch back to the main frame  
remDr$switchToFrame(NULL)

arrowElem <- arrowElems[[1]]  
# once you "click"" the element it is "held down" as far as I know there is no way to " unclick"  
# to prevent it from scrolling too far I make sure not to take too long by setting the sleep short  
arrowElem$clickElement()  
# give it "just enough time" to scroll right  
Sys.sleep(0.3)  
# switch back to outer frame to re-start the loop  
remDr$switchToFrame(NULL)  
 }  
}

temp1 <- dplyr::bind\_rows(table\_tmp, big\_df)

The problem here is that the for loop eventually fails at the end when it tries to click the right arrow but it’s as far to the right as it can go — therefore it won’t download the last few tables (~5). Typically one would handle such conditions with something like:

check <- try(expression, silent = TRUE) # or suppressMessages(try(expression, silent = TRUE))  
if (any(class(check) == "try-error")) {  
 # do stuff  
}

and it usually works fine, including when using selenium. The issue encountered here however is clicking on the arrow once would always bring me to the *last* visible sheets — skipping everything in middle. Therefore my work around to get the remaining sheets was this:

# ctrl+x from the google sheet then use the read.delim() to assign it to an object  
march <- read.delim("clipboard")  
february <- read.delim("clipboard")  
january <- read.delim("clipboard")  
december <- read.delim("clipboard")  
november <- read.delim("clipboard")

# add the city and date  
january$city <- rep("montreal", nrow(january))  
january$date <- rep("2015-01-30", nrow(january))

february$city <- rep("montreal", nrow(february))  
february$date <- rep("2015-02-15", nrow(february))

march$city <- rep("montreal", nrow(march))  
march$date <- rep("2015-03-15", nrow(march))

december$city <- rep("montreal", nrow(december))  
december$date <- rep("2014-12-15", nrow(december))

november$city <- rep("montreal", nrow(november))  
november$date <- rep("2014-11-15", nrow(november))

# clean up the column names  
january %<>% janitor::clean\_names()  
february %<>% janitor::clean\_names()  
march %<>% janitor::clean\_names()  
december %<>% janitor::clean\_names()  
november %<>% janitor::clean\_names()

# reorder the columns  
xyz %<>% dplyr::select(team\_name, team\_size, start\_time, end\_time, total\_time, puzzles, city, date)  
# rename this column to match temp1  
xyz <- rename(xyz, puzzels\_solved = puzzles)  
# change to a character  
xyz$puzzels\_solved <- as.character(xyz$puzzels\_solved)  
# add NA for team size  
xyz$team\_size <- rep(NA, nrow(xyz))  
# concatenate these five dataframes together  
xyz <- bind\_rows(march, february, january, december, november)  
# convert characters into actual dates  
xyz$date <-as.Date(xyz$date)

# concatenate this onto the larger dataframe  
temp2 <- bind\_rows(temp1, xyz)

# save the object  
write\_csv(temp2, "puzzeld\_pint\_raw.csv")

**Method # 2**

Big thanks to [Nate on SO](https://stackoverflow.com/questions/54084659/exception-handling-rselenium-switchtoframe-error-elementnotvisible/54276021#54276021) for pointing out an alternative solution that solves the task of *scrapping the tables* but **not** the task of exception handling in the above sense.

# table yielding function  
# just for readability in the loop  
create\_table <- function (remDr) {  
 # parse html with XML package  
 doc <- XML::htmlParse(remDr$getPageSource()[[1]])  
 Sys.sleep(3)  
 # Extract data from HTML table in HTML document  
 table\_tmp <- XML::readHTMLTable(doc)  
 Sys.sleep(3)  
 # put this into a format you can use  
 table <- table\_tmp[[1]][-2, -1]  
 # add a check-up for size mismatch  
 table\_fields <- as.character(t(table[1,]))  
 if (! any(grepl("size", tolower(table\_fields)))) {  
 table <- table[-1, ]  
 # rename the columns  
 colnames(table) <- c("team\_name", "start\_time", "end\_time", "total\_time", "puzzels\_solved")  
 table$team\_size <- NA\_integer\_  
 table <- table[,c("team\_name", "team\_size", "start\_time", "end\_time", "total\_time", "puzzels\_solved")]  
 } else {  
 table <- table[-1, ]  
 # rename the columns  
 colnames(table) <- c("team\_name", "team\_size", "start\_time", "end\_time", "total\_time", "puzzels\_solved")  
 }  
 # add city name to a column  
 table$city <- rep("Montreal", nrow(table))  
  
 # add the Month/Year this table was extracted from  
 today <- Sys.Date()  
 lubridate::month(today) <- lubridate::month(today)+1  
 table$date <- today  
  
 # returns the table  
 table  
}  
  
# II. Scrapping the content  
#   
# 1. selenium to generate the pages  
# 2. use create\_table to extract the table  
#   
big\_df <- NULL  
for (k in seq\_along(sheet\_url)) {  
 # 1. navigate to the page  
 remDr$navigate(sheet\_url[k])  
 # remDr$screenshot(display = TRUE) maybe one wants to see progress  
 table <- create\_table(remDr)  
  
 # 2. concatenate each table together  
 big\_df <- dplyr::bind\_rows(big\_df, table)  
  
 # inform progress   
 cat(paste0('\nGathered table for: \t', sheet\_months[k]))  
}  
  
# close session  
remDr$close()

To perform the task, what was done was firstly generating the links to all spreadsheets in the document. To do this:

* Navigate once to the document
* Extract source code
* Extract the sheet months and URLs (via gid digit) using regex
* Once this is done, loop through the Urls, gather and bind the tables

There is a small function called create\_table which returns the final table in the proper format with a safety measure for the number of columns (some of the spreadsheets do not have the team\_size field – in those cases I set it to NA\_integer).