Webscraping with R

We will cover two key packages for doing this: rvest (a Hadley Wickham number) and RSelenium, a very useful but also kind of finicky package to deal with complex sites.

rvest

```
#install.packages("rvest")
library(rvest)
library(reshape2)
```

rvest reads in and parses html objects - extremely handy for extracting data from reasonably well-organized websites. There are 3 easy steps:

1. Read in html code from url
2. Identify the element(s) of interest using css selector or other
3. Extract the relevant information

Suppose we care about the box score of some random baseball game:

```
# define url
startpage<- "http://www.espn.com/mlb/boxscore?gameId=370413106"

# extract page info
twins <- read_html(startpage) %>%
  # id selector of interest, from inspect feature
  html_nodes(css = "#matchup-mlb-370413106-awayScore") %>%
  # extract text
  html_text()

tigers <- read_html(startpage) %>%
  html_nodes(css = "#matchup-mlb-370413106-homeScore") %>%
  html_text() %>%
  as.numeric()
```

But if we want to get both scores at once, we need to use a little less specificity with the selector:

```
both <- read_html(startpage) %>%
  html_nodes(css = "#matchup-mlb-370413106 > div > div > h3 > span") %>%
  html_text()

# can also extract tables quite easily
innings<- read_html(startpage) %>%
  html_nodes(css = "#gamepackageTop > div.line-score.clear > div.line-score-container > table") %>%
  html_table(header = T)

# can use xpath instead of css
fullbox<- read_html(startpage) %>%
  html_nodes(xpath = '//*[@id="my-players-table"]/div[2]/div[1]/div/table') %>%
  html_table(fill = T, header = F) %>%
  melt()
# ugly, but we can work with it
```

If you needed to, say, scrape multiple pages with known/formulaic urls, you can see how you'd easily construct a function that takes as an argument all or part of the url and returns the data of interest. You can also identify links on a page and follow them to the next page.

RSelenium

This is a more versatile but also (much) more finicky package that lets you run a browser from R. I'm only going to cover installation and setup on a Mac, because I have never done it on Windows/Linux.

Here is a reference guide for getting Selenium set up on a Mac with Firefox. That being said, these instructions did not work for me recently!

RSelenium with Chrome

1. Download selenium server
2. Download chrome driver
3. Update Java
4. Install RSelenium package

When you want to use selenium to run a browser, click-open the standalone server. Then call the library in R and open your browser.

```
#install.packages("RSelenium")
library(RSelenium)
remDr <- remoteDriver(browser = "chrome")

remDr$open()

startpage<- "https://www.google.com"

# navigate to a url
remDr$navigate(startpage)
remDr$refresh()

# name elements you need to interact with (click, enter text, etc.)
searchbox<- remDr$findElement(using = "css", "#lst-ib")

# interact with elements
searchbox$clickElement()
searchbox$sendKeysToElement(list("hello world", key = "enter"))

newlink<- remDr$findElement(using = "css", "#rso > div:nth-child(1) > div > div:nth-child(3) > div > h3
newlink$highlightElement(wait = 1)
newlink$getElementText()
newlink$clickElement()

# where are we now?
remDr$getCurrentUrl()

remDr$goBack()
remDr$getCurrentUrl()
```

```
# other fun things
remDr$screenshot(display = TRUE)

# close browser (or manually quit)
remDr$close()
```

It's worth noting that while getting selenium up and running is the most complicated part, it can still be a bit delicate - although largely this is the natural result of operating a browser and interacting with websites. If you tell it to look for an element and it can't find it (say, because the page is still loading), the script will break off. So you may want to incorporate calls like Sys.sleep() or try() to build a more robust function.