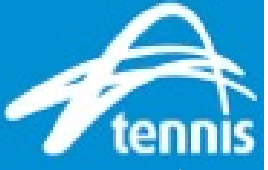
Getting Sports Data

Inspecting and Scraping Data from the Web



What Next?

If you've found the data you want on the Web, what comes next?

What Next?

If you've found the data you want on the Web, what comes next?

If you want to capture these or similar data repeatedly, you will want to retrieve it in a reproducible way.

Ways of Getting Sports Data

- Not always fun but a necessary part of sports analysis
- There are two major ways to get data from Web:

    1. Import a file directly

    2. Extract from HTML

Example: Import Data File

- Files that can be read with `read.table` or related functions can be directly imported from a URL.
- Here we extract the most recent Australian Open match results and betting odds using `read.csv`.

```
url <- "http://www.tennis-data.co.uk/2017/ausopen.csv"

read.csv(url)
```

Practice: URL Patterns & Importing

Consider the previous example. If we wanted the same data for the 2016 US Open, how do you think we could do that?

Practice: URL Patterns & Importing

Consider the previous example. If we wanted the same data for the 2016 US Open, how do you think we could do that?

1. Test a possible URL for the 2016 US Open
2. Import the file
3. Run a `str` on the dataset to determine what info it contains

Solution: URL Patterns & Importing

Changing the year and tournament names in the URL are enough to get the correct file.

```
url <- "http://www.tennis-data.co.uk/2016/usopen.csv"

usopen <- read.csv(url)
```

When we look at the variables, we find it contains the scores and betting odds for all 127 main draw matches for the US Open.

```
str(usopen) # Data contents
```

Scraping from a Website

If you can't directly import data from the Web--which is usually the case--you can still capture the data but you need to know whether it is *static* or *dynamic* data.

Scraping from a Website

If you can't directly import data from the Web--which is usually the case--you can still capture the data but you need to know whether it is *static* or *dynamic* data.

What is *static* data?

What is *dyanmic* data?

And how do you determine which type of data you have?

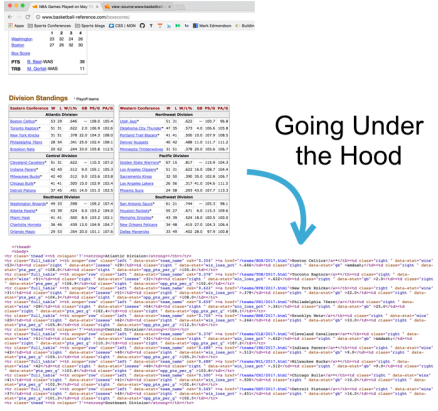
Static vs. Dynamic Data

We use different methods to get Web data depending on which type it is, so it is very important to be able to identify each type.

- *Static* web data is data you can see in the source code.
- If you can't see the data, the data is *dynamic*.

How Do I Know if Data is Static or Dynamic?

- You need to be able to inspect HTML and CSS
- This means being able to "View Source"
- Being able to identify CSS elements in source.



Web Developer Tools

Every modern browser has a suite of "developer tools". These include useful functions for Web scrapers, including:

1. Viewing the source code
2. Inspecting elements

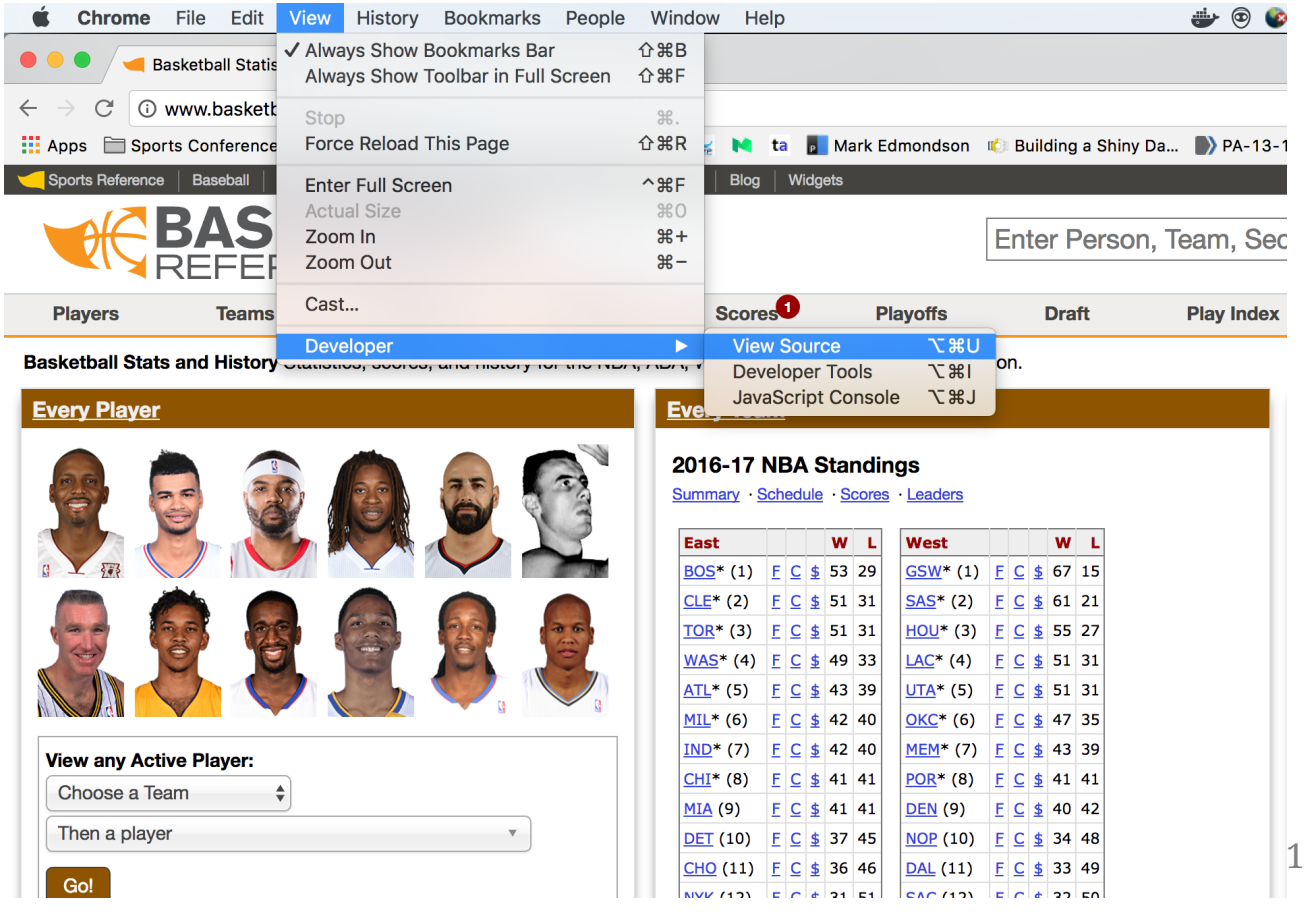
Web Developer Tools

Every modern browser has a suite of "developer tools". These include useful functions for Web scrapers, including:

1. Viewing the source code
2. Inspecting elements

I personally like Chrome's suite of Developer Tools.

Viewing Source in Chrome



Practice: Static or Dynamic?

Look at the source code for each of the following sites and determine whether they are examples of *static* or *dynamic* data.

Case 1. http://tennisabstract.com/reports/atp_elo_ratings.html

Case 2. http://www.espncricinfo.com/ci/content/stats/

Solution: Static or Dynamic?

Case 1 is static data.

Case 2 is dynamic data.

Finding Elements

- Whether you are working with static of dynamic data you need to be able to locate the elements that contain your data
- It is the information about this element which you will need to automate data capture
- CSS `class` and `id` fields are the most common ways to uniquely identify the element containing your data

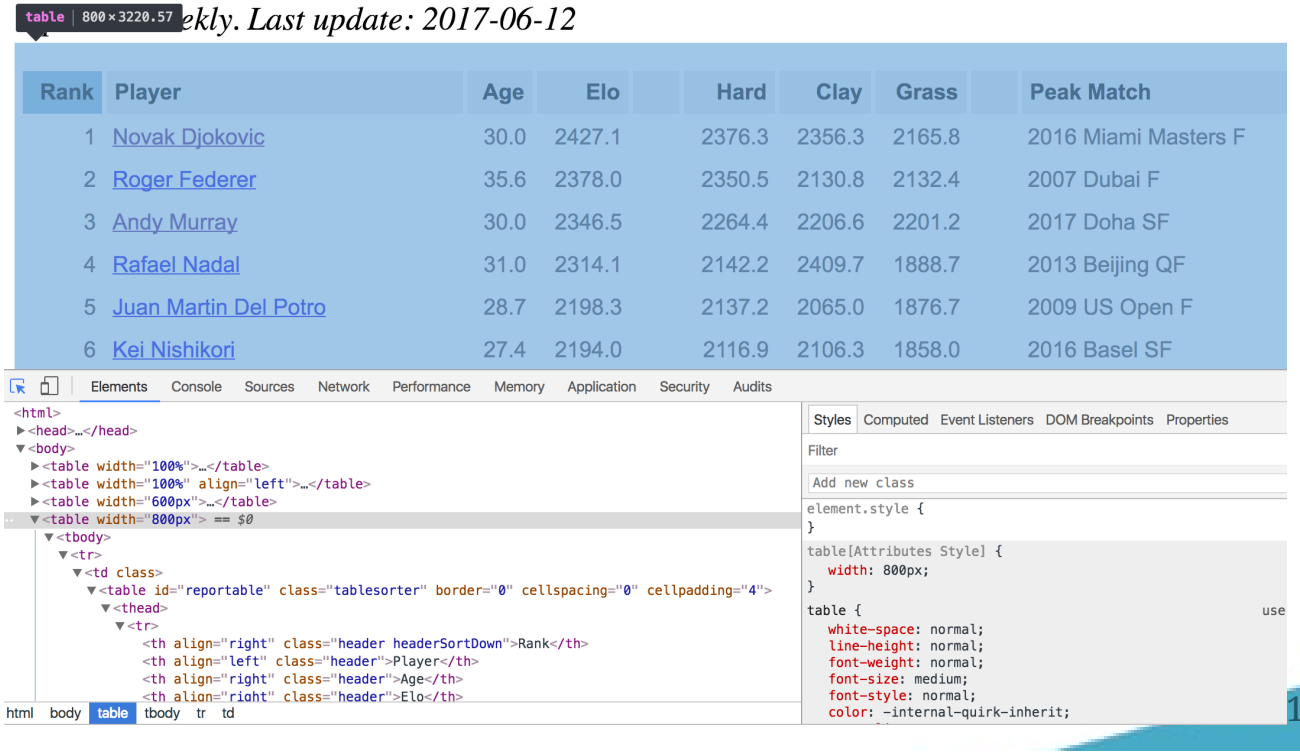
Practice: Inspect Element

Use the following static data example:
http://tennisabstract.com/reports/atp_elo_ratings.html

1. Find the CSS class or id that contains the Elo ratings data

Solution: Inspect Element

Here, I use the "inspect element" settings from Chrome's developer tools to learn about the CSS of the table containing the Elo ratings.



Solution: Inspect Element

- The CSS **class** for the table is "tablesorter"
- The CSS **id** is "reportable"

Scraping Static Data

There are a few options for extracting static HTML data.

1. `readLines` is an option if the data is *not* nicely formatted, in other words, when there is a lack of structure
2. More typically, the data is *nice* (e.g. if it is contained in a HTML table or other predictable tag) and we can use scraping packages like `rvest` or `RCurl` to get the data in a format we can work with.

Using `rvest`

Using `rvest`

- Suite of tools for scraping static Web data and putting them in easy-to-use objects (like data.frames)

Using `rvest`

- Suite of tools for scraping static Web data and putting them in easy-to-use objects (like data.frames)
- Works with `magrittr` and allows piping commands with `%>%` operator

Using `rvest`

- Suite of tools for scraping static Web data and putting them in easy-to-use objects (like data.frames)
- Works with `magrittr` and allows piping commands with `%>%` operator
- Allows some browsing functionality

Using `rvest`

- Suite of tools for scraping static Web data and putting them in easy-to-use objects (like data.frames)
- Works with `magrittr` and allows piping commands with `%>%` operator
- Allows some browsing functionality
- Authored by Hadley Wickham

Example: Scraping Box Scores

In this example, we will use `rvest` to extract the Eastern Division Standings.

First, we import the page content.

```
library('rvest')

# Creating object with the address
url <- 'http://www.basketball-reference.com/boxscores/'

#Reading the code from the site
webpage <- read_html(url)
```

Example: Scraping Box Scores

The `html_nodes` function is the work horse function for extracting specific elements of a site. We can specify the element we want using its CSS tag or using an XPATH selector.

```
# Using the CSS table tag to get all tables
data <- webpage %>%
   html_nodes(css = 'table') %>%
   html_table()

length(data) # List of multiple tables
```

```
## [1] 7
```

Example: Scraping Box Scores

Using an XPATH (XML Path Language) can help to make our extraction more specific, though the syntax is more opaque.

```
# Using an XPATH selector to get the specific table of interest
data <- webpage %>%
   html_nodes(xpath = '//*[@id="divs_standings_E"]') %>%
   html_table(header = T)

head(data[[1]])
```

```
##   Eastern Conference                 W                 L               W/L%
## 1  Atlantic Division Atlantic Division Atlantic Division Atlantic Divisior
## 2    Boston Celtics*                53                29               .646
## 3   Toronto Raptors*                51                31               .622
## 4    New York Knicks                31                51               .378
## 5 Philadelphia 76ers                28                54               .341
## 6      Brooklyn Nets                20                62               .244
##                   GB              PS/G              PA/G
## 1 Atlantic Division Atlantic Division Atlantic Division
## 2                  —             108.0             105.4
## 3                2.0             106.9             102.6
## 4               22.0             104.3             108.0
```

Practice: Static Data Extraction

The following site lists the Elo ratings of professional male tennis players:
Tennis Abstract Elo

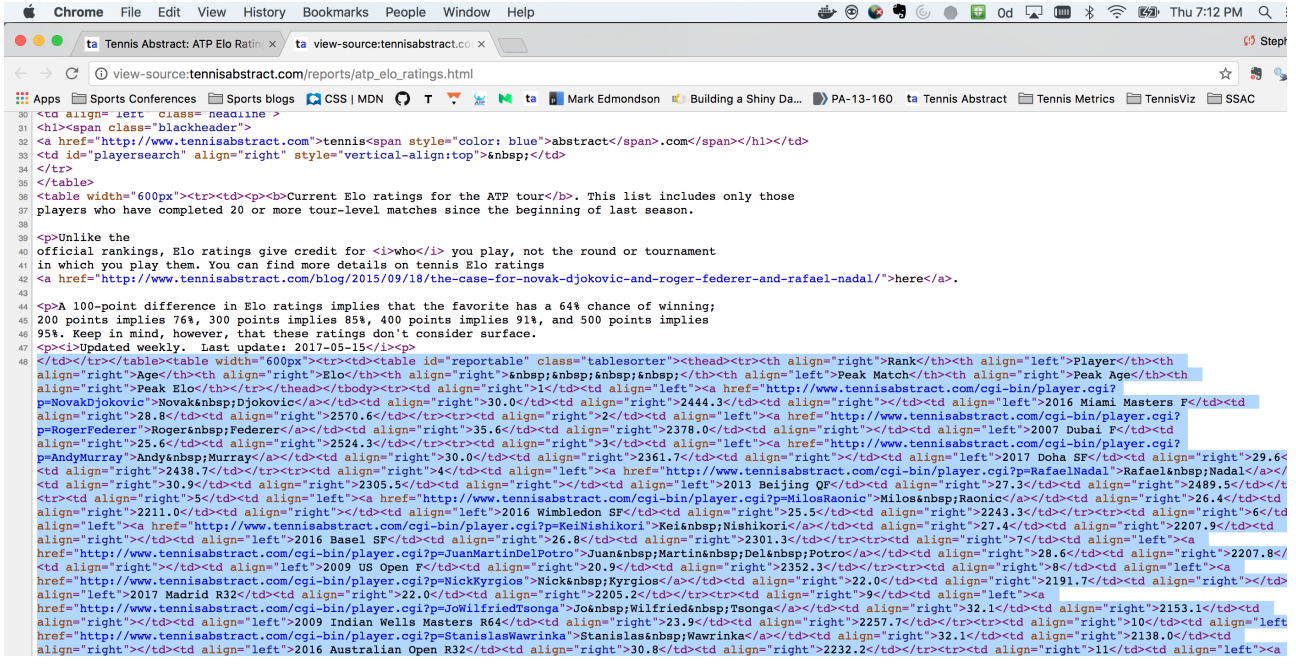
Practice: Static Data Extraction

The following site lists the Elo ratings of professional male tennis players:
Tennis Abstract Elo

1. Use your Web inspection tools to determine if the ratings are static data
2. Use `rvest` to scrape the data as efficiently as you can

[1] For a 'table' with class 'x' you can use 'table.x' as a shortcut

Solution: Elo Rating Extraction



Solution: Elo Rating Extraction

```
url <- "http://tennisabstract.com/reports/atp_elo_ratings.html"

page <- read_html(url)

# Use table class to extract Elo table
elo <- page %>%
    html_nodes("table.tablesorter") %>%
    html_table()

head(elo)
```

```
## [[1]]
##     Rank                     Player  Age    Elo     
## 1      1             Novak Djokovic 30.0 2427.1   NA 2376.3 2356.3 2
## 2      2              Roger Federer 35.6 2378.0   NA 2350.5 2130.8 2
## 3      3                Andy Murray 30.0 2346.5   NA 2264.4 2206.6 2
## 4      4               Rafael Nadal 31.0 2314.1   NA 2142.2 2409.7 1
## 5      5      Juan Martin Del Potro 28.7 2198.3   NA 2137.
## 6      6              Kei Nishikori 27.4 2194.0   NA 2116.9 2106.3 1
## 7      7               Milos Raonic 26.4 2185.0   NA 2125.1 2004.5 1
## 8      8              Dominic Thiem 23.7 2178.0   NA 1892.1 2217.6 1
## 9      9         Stanislas Wawrinka 32.2 2171.5   NA 2078.3 2138.2 1
## 10    10           Alexander Zverev 20.1 2156.2   NA 1959.9 2066.9 1
```

Dynamic Data & Automated Browsing

Dynamic Data & Automated Browsing

- Because dynamic data is created on-the-fly (in response to user interactions) we have to browse to get access to it

Dynamic Data & Automated Browsing

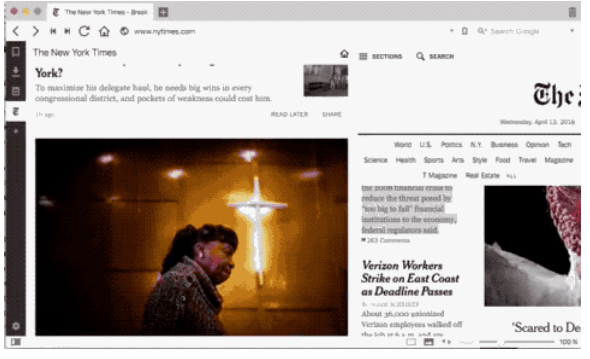
- Because dynamic data is created on-the-fly (in response to user interactions) we have to browse to get access to it
- Fortunately, we can automate browsing

Dynamic Data & Automated Browsing

- Because dynamic data is created on-the-fly (in response to user interactions) we have to browse to get access to it
- Fortunately, we can automate browsing
- We just need to find what instructions to give to mimic the browsing that generates the data and get familiar with tools that can implement these instructions

Scraping Dynamic Data with `RSelenium`

- We have to automate Web browsing to get dynamic data
- *Selenium* is software that allows automated Web browsing
- RSelenium is a package that provides Selenium functionality in R



RSelenium: Basic Steps

1. Set the Web driver (select browser and port)
2. Find the elements with the data
3. Extract the content
4. Parse the contents

Installing RSelenium

There are a few steps you need to get started with RSelenium.

1. Install a Selenium server which is a standalone java program and can be downloaded here:selenium-release.storage.googleapis.com
2. Run the Selenium server with the command: `java -jar selenium-server-standalone-x.xx.x.jar` where the `x.xx.x` will be the specific version (Default port is 4444)
3. Install RSelenium from CRAN

Example: Tennis Match Statistics

Consider the following match summary: 2017 Australian Open Final

Example: Tennis Match Statistics

If we inspect the page, we find that these stats are dynamic data. We also find that the main table of content has the id `detail`.

```
    <body id="top" class="tennis detailbody">

        <div id="detail" class="sport-tennis"><div id="detcon">
         <table class="detail">
             <thead>
             <tr>
                 <th class="header">
                     <div class="fleft">
                         <span class="flag fl_3473162"></span>ATP - SINGLES: <a href="#" onclick="window.open('/tennis/atp-singles/australian-open/');
return false;">Australian Open (Australia), hard - Final</a>

                     </div>

                 </th>
             </tr>
             </thead>
             <tbody>
             <tr>
                 <td class="hclean"></td>
             </tr>
             </tbody>
```

Using RSelenium

Below we activate the driver using a port that is not in use.

Note: You may need to activate javascript in the background for this driver to work.

```
# Running java -Dwebdriver.chrome.driver="chromedriver" -jar seleniu

library(RSelenium) # Load the package

# Match statistics URL
url <- "http://www.flashscore.com/match/Cj6I5iL9/#match-statistics;0"

# Establish remote driver using Chrome
remDr <-remoteDriver(port = 5556, browser = "chrome")
remDr$open(silent = TRUE)
remDr$navigate(url) # Navigate page
```

Using RSelenium

Next we extract the table of stats using the CSS `id` node.

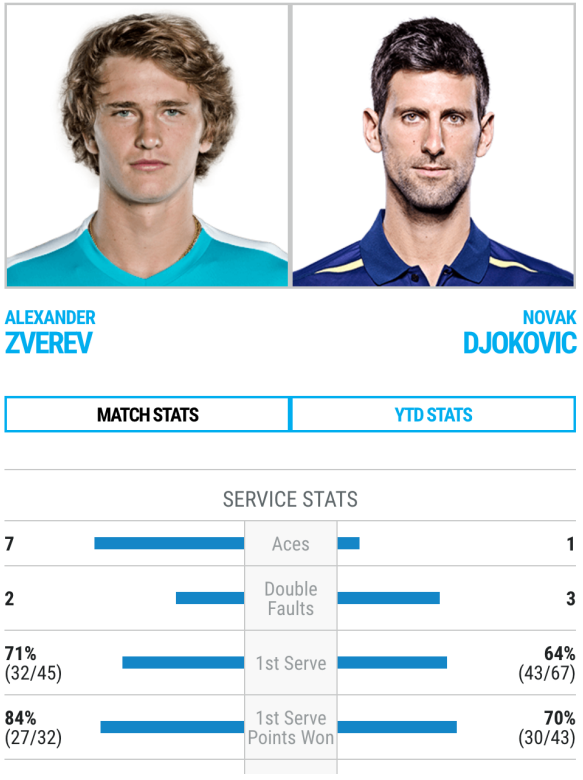
```
# Get id element
webElem <- remDr$findElements(using = 'id', "detail")

#  Use getElementText to extract the text from this element
unlist(lapply(webElem, function(x){x$getElementText()}))[[1]]

remDr$close() # Close driver when finished
```

Practice: RSelenium

Take a look at the following match summary that you can find here.



Practice: RSelenium

Use what we've covered about `RSelenium` to extract the statistics for this match.

1. Start by inspecting the Web site
2. Determine which CSS element is most likely to contain the stats
3. Create a remote driver, navigate to that element, and check if the text for the match statistics are contained in the element

Solution: RSelenium

Inspection of the source code suggests that the Element with id *modalScoresMatchStatsTable* is likely to contain the statistics.

```
<div id="modalScoresContentContainer" class="modal-scores-tab-container">
    <div id="modalScoresMatchStats" class="modal-scores-match-stats">

        <div id="modalScoresMatchStatsTable" class="modal-scores-match-stats-table">
            <div class="modal-scores-match-stats-players">
                    <div class="match-stats-player-left">
        <div class="player-left-image">
            <a href="/en/players/alexander-zverev/z355/overview">
                <img src="/-/media/tennis/players/head-shot/2017/02/12/01/20/zverev_a-headshot-ao17.png" />
            </a>
        </div>
        <div class="player-left-name">
            <a href="/en/players/alexander-zverev/z355/overview">
                <span class="first-name">
                    Alexander
                </span>
                <span class="last-name">
                    Zverev
                </span>
            </a>
        </div>
    </div>
```

Solution: RSelenium

Now we navigate to the site.

```
# Match statistics URL
url <- "http://www.atpworldtour.com/en/players/novak-djokovic/D643/ov

# Establish remote driver using Chrome
remDr <- remoteDriver(port = 5556, browser = "chrome")
remDr$open(silent = TRUE)
remDr$navigate(url) # Navigate page
```

Solution: RSelenium

Then we find the `id` element of interest and extract the text it contains.

```
# Get id element
webElem <- remDr$findElements(using = "id",
    "modalScoresMatchStatsTable")

# Use getElementText to extract
# the text from this element
unlist(lapply(webElem, function(x) {
    x$getElementText()
}))[[1]]
```

```
## [1] ""
```

Summary

- Web data can be classed into three main categories: directly importable, static, or dynamic
- We can use source inspection and CSS selector tools to determine which data type we are working with and the site elements that contain the data
- We have seen how we use tools like `rvest` to capture static Web data
- For dynamic data, we can use automated browsing with `RSelenium`

Resources

- CSS and HTML crash course
- XPATH
- rvest
- RSelenium