Learning How to Webscrape

Abrielle Agron

Today, we will be looking at how we can get data from websites that don't allow us to simply download a .csv file. We will be using a technique called webscraping. Webscraping is the process of automatically extracting data from a webpage. Oftentimes, data isn't simply available for download as a .csv file, and may be located within graphics, on different pages, or in other inconvenient ways. Rather than manually copy and paste large amounts of data manually, it can be easier and more efficient to let a webscraping tool do the work for you.

Example

For this example, I will be looking at team statistics for NCAA Division 1 Women's Soccer teams in the 2018-2019 season found here. Before we get started, I highly recommend using Google Chrome and installing the extension linked below. You don't need to, but it will make your life a lot easier.

Get Selector Gadget

To get started, we will install the package rvest, which is the most common web scraping package for R.

```
#install.packages('rvest')
library('rvest')
```

The first step when webscraping is to access all of the HTML code of the site.

```
site <-
  "https://stats.ncaa.org/rankings/institution_trends?academic_year=2019.0&division=1.0&sport_code=WSO"
site_info <- read_html(site)
site_info
```

```
## {html_document}
## <html>
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <script>\n\n  $.ajaxSetup ({\n    // Disable caching of AJAX responses\n  ...
## [3] <body id="body" onload="if (top != self) { top.location=self.location; }" ...
## [4] <script>\n      //ui-datepicker = calendar month prev/next buttons, \n    ...
```

After we have this information, we will then use the tools built in rvest to parse it. Let's say that the first data item we want is simply the names of the teams. We use the selector gadget to find the CSS tags for only the team names, and then pass that in as an argument to the html_nodes() function. When using the selector gadget, first click on the data you want. If anything that you don't want is highlighted, click on it again to deselect it. You can click any data that isn't highlighted to add it to your selection. Once this process is finished, copy and paste the tag given by the gadget. In this case, we get '.text td:nth-child(1)'. Don't worry if this doesn't mean anything to you, that's why we use the tools that we do.

NOTE: In this case, make sure that you aren't sorting by the column that you're trying to get data from. If the CSS tag reads '.sorting_1' you've made this mistake. Sort the table by a different column and then try again.

```
#Retrieve all HTML node objects that match the given CSS descriptor
team_html <- html_nodes(site_info, '.text td:nth-child(1)')
```

```
#Convert HTML node objects to text
team <- html_text(team_html)
#Look at the first few elements to make sure that they're formatted correctly
head(team)
```

```
## [1] "A&M-Corpus Christi" "Abilene Christian"  "Air Force"
## [4] "Akron"              "Alabama"            "Alabama A&M"
```

We repeat this exact process for each column of data that we wish to use. For this example, I create a dataframe using only a few columns of the data.

```
conf_html <- html_nodes(site_info, '.text td:nth-child(2)')
conf_text <- html_text(conf_html)
#Because conference is a categorical variable, we convert it to a factor
conf <- as.factor(conf_text)
head(conf)
```

```
## [1] Southland     Southland     Mountain West MAC           SEC
## [6] SWAC
## 32 Levels: AAC ACC America East ASUN Atlantic 10 Big 12 Big East ... WCC
```

```
goals_html <- html_nodes(site_info, '.text td:nth-child(3)')
goals <- html_text(goals_html)
#This is a quantitative variable, so we trim any excess whitespace and convert it to a numeric
goals <- as.numeric(trimws(goals, 'both'))
head(goals)
```

```
## [1] 14 32  9 27 37 21
```

```
goals_allowed_html <- html_nodes(site_info, '.text td:nth-child(13)')
goals_allowed <- html_text(goals_allowed_html)
goals_allowed <- as.numeric(trimws(goals_allowed, 'both'))
head(goals_allowed)
```

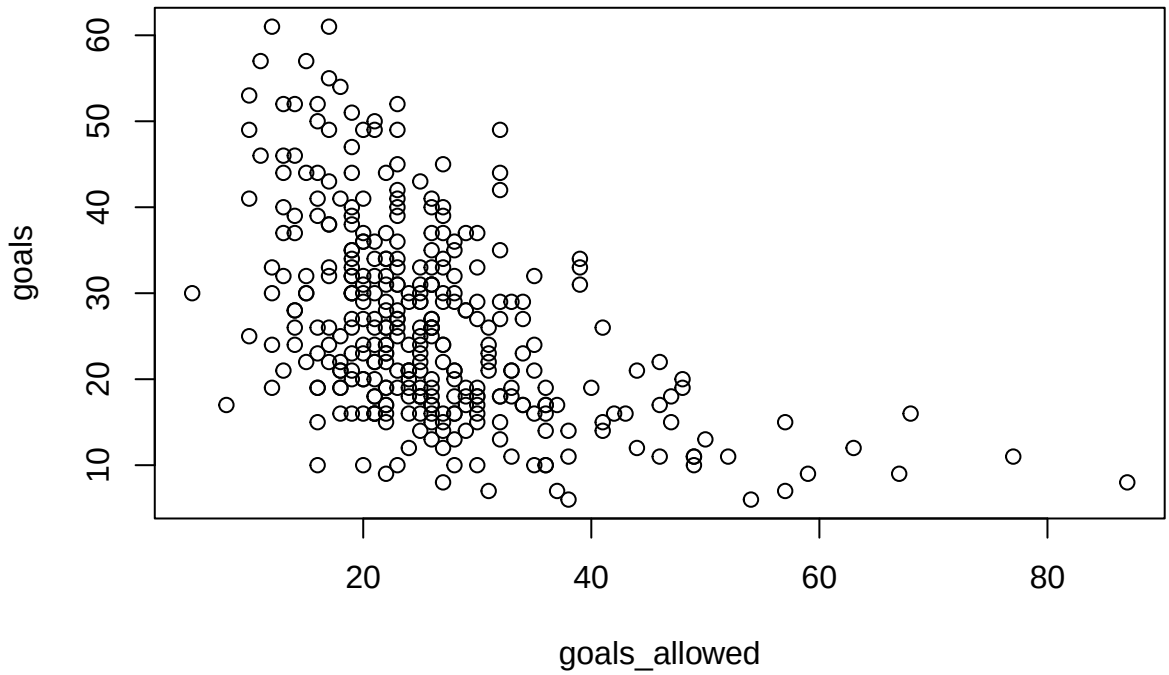
```
## [1] 25 13 22 23 26 33
```

```
#Collect all data into a single dataframe
soccer.df <- data.frame(team, conf, goals, goals_allowed)
head(soccer.df)
```

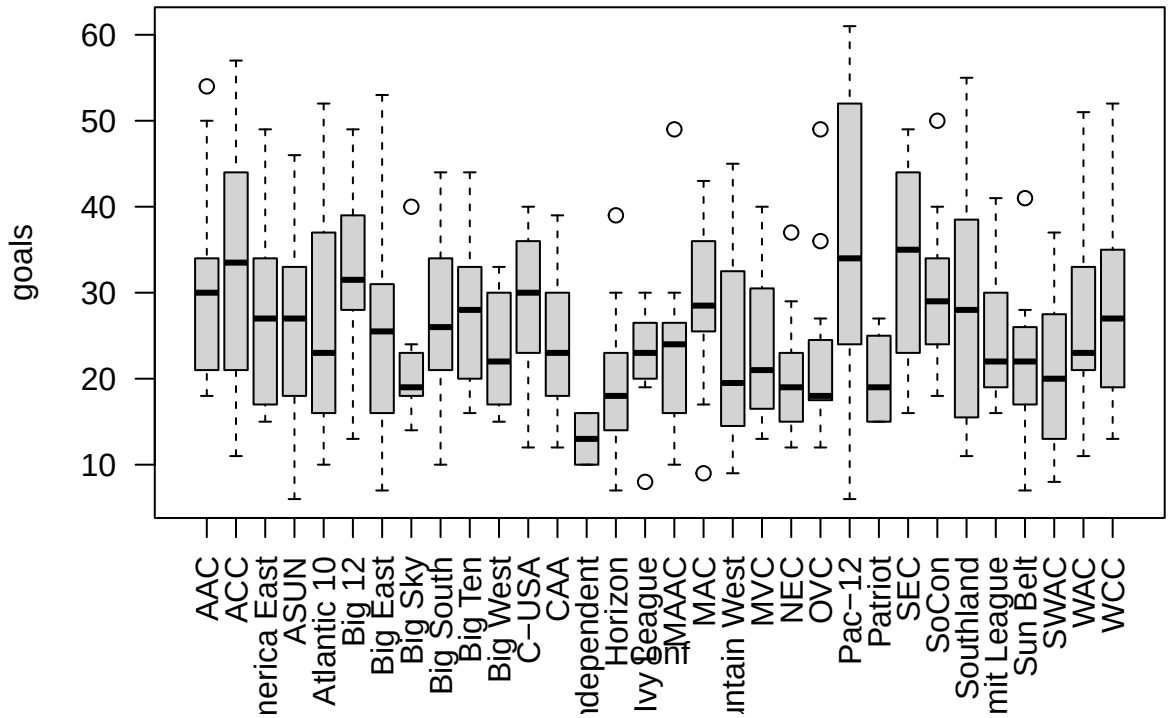
```
##                 team          conf goals goals_allowed
## 1 A&M-Corpus Christi     Southland    14            25
## 2  Abilene Christian     Southland    32            13
## 3          Air Force Mountain West     9            22
## 4              Akron           MAC    27            23
## 5            Alabama           SEC    37            26
## 6        Alabama A&M          SWAC    21            33
```

With this limited dataset, let's look at a few plots that compare the data

```
plot(goals ~ goals_allowed, data = soccer.df)
```




```
plot(goals~conf, data = soccer.df, las = 2)
```



Now you know the basics! It's time to do some scraping of your own.

Exercise

Option 1:

Choose a sport from the list below and scrape that website. Create a dataframe with as many columns as you want and create some kind of metric to predict points, goals, wins, or some other statistic. Make sure that you do some research and figure out what each of the statistics mean!

Links Men's Baseball
Men's Basketball
Men's Ice Hockey
Men's Lacrosse
Men's Soccer
Men's Volleyball
Women's Field Hockey
Women's Ice Hockey
Women's Lacrosse
Women's Softball
Women's Soccer
Women's Volleyball

Option 2:

Find a website of your own to scrape! Comfortable programming? Try to write a script to get data on all UFC athletes from each of their pages at https://www.ufc.com/athletes.
Interested in the history of track and field? Check out some interesting datasets about everything from sub 4-hour marathons to the best non-qualifying results ever at https://trackandfieldnews.com/stats-and-more/statistics/.
These are just a couple interesting sites that I found with the potential to be scraped. Feel free to find data on the sport of your choosing, and do some analysis on what you collect.