15 - HTML Data Tables

Data and Information Engineering

SYS 2202 | Fall 2019

15-webdata.pdf

Contents

Required Packages and Data

```
library(ggrepel)
library(rvest)
library(tidyverse)
# library(stringr) part of tidyverse
```

1 HTML Tables

1.1 HTML Tables

HTML web pages often contain data tables that we want to access. To extract the data, we need to know something about how tables are represented in html format. Here are two descriptions of html tables: Example 1 and Example 2.

We can `View Source` to see the raw html behind the page Bureau of Labor Statistics projection of Statistics jobs.

If you have chrome or firefox browser (and maybe others) you can hover over the top of the desired table and right click and `Inspect` or `Inspect Element`. Find the desired table (which will be highlighted), and find its selector.

The author of `rvest` suggests using selectorgadget with chrome to make this process easier. I think this will only work with Chrome browser.

1.1.1 Javascript Generated Tables

We will show how to get data from usual html tables, but getting javascript generated data takes more effort than we can give. But here and here are links to some details of how to get it with R.

1.2 `rvest` package

The `rvest` package https://rvest.tidyverse.org/ makes it easy to scrape (or harvest) data from html web pages.

But keep in mind that data in html pages (and documents) is formatted for human readability, not computation. So we will often have to do post-processing to get the data into the form we need for calculations or visualization.

2 College Football

2.1 Top 100 Rushers

The webpage http://www.cfbstats.com/2016/leader/national/player/split01/category01/sort01.html has a table of the top 100 rushers in college football in 2016. While it could be possible to copy and paste the data into excel, say, and then load into R, we can also do this directly from R (which will save us much time there are many tables to get).

It will take three steps to get the table.

1. The first step to specify the url and read the webpage into R with `read_html()`.
2. Next, we need to grab a specific node, in this case the *table* items, with `html_node` (or `html_nodes`).
3. Convert it to a data frame with `html_table()`. Here are all three steps together

```
library(rvest)
url = 'http://www.cfbstats.com/2016/leader/national/player/split01/category01/sort01.html'
rushing = read_html(url) %>% html_node("table") %>% html_table()
rushing$Name = parse_character(rushing$Name)    # fix problem encoding
colnames(rushing)[1] = "rank"                   # add the column name
glimpse(rushing)
```

```
#> Observations: 100
#> Variables: 12
#> $ rank      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1...
#> $ Name      <chr> "D'Onta Foreman", "Donnel Pumphrey", "Aaron Jones", ...
#> $ Team      <chr> "Texas", "SDSU", "UTEP", "Stanford", "BYU", "FSU", "...
#> $ Yr        <chr> "JR", "SR", "JR", "JR", "SR", "JR", "JR", "JR", "SR"...
#> $ Pos       <chr> "RB", "RB", "RB", "RB", "RB", "RB", "RB", "RB", "RB"...
#> $ G         <int> 11, 14, 12, 11, 10, 13, 14, 13, 13, 13, 10, 13, 13, ...
#> $ Att       <int> 323, 349, 229, 253, 234, 288, 349, 314, 258, 237, 20...
#> $ Yards     <int> 2028, 2133, 1773, 1603, 1375, 1765, 1860, 1709, 1629...
#> $ Avg.      <dbl> 6.28, 6.11, 7.74, 6.34, 5.88, 6.13, 5.33, 5.44, 6.31...
#> $ TD        <int> 15, 17, 17, 13, 12, 19, 22, 23, 18, 27, 7, 21, 17, 7...
#> $ `Att/G`   <dbl> 29.4, 24.9, 19.1, 23.0, 23.4, 22.1, 24.9, 24.1, 19.9...
#> $ `Yards/G` <dbl> 184, 152, 148, 146, 138, 136, 133, 131, 125, 125, 12...
```

Notice there is a problem with the data frame `rushing`, namely the first column is not named! While the R `data.frame` is OK with this, any dplyr function (e.g., `mutate()`) will not work.
Also, the `parse_character()` function is used to resolve the encoding problem with the apostrophes in the names.

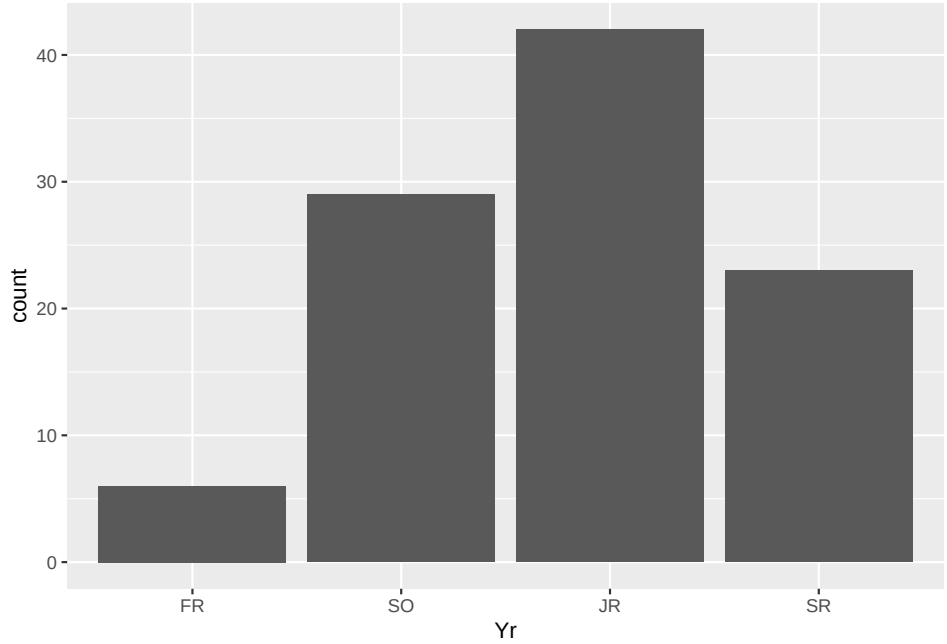
The `read_html()` function produces an XML document, similar to what the browser gives (but some exceptions, like with javascript generated data). The `html_node()` node that is tagged with `"table"`. In this case, there is only one table, but we will have to be more specific when a webpage has more than one table.

Now, for fun, we can check the distribution of class year:

```
#- convert to factor
rushing$Yr = factor(rushing$Yr, levels=c("FR", "SO", "JR", "SR"))

#- counts
count(rushing, Yr)
#> # A tibble: 4 x 2
#>   Yr        n
#>   <fct> <int>
#> 1 FR        6
#> 2 SO       29
#> 3 JR       42
#> 4 SR       23

#- bar plot
ggplot(rushing) + geom_bar(aes(x=Yr))
```



Why are there so many juniors on the top list? Should we expect mostly seniors next year? Note:last year it was sophomores that were on top of the list! You can check yourself here http://www.cfbstats.com/2015/leader/national/player/split01/category01/sort01.html

3 Hockey

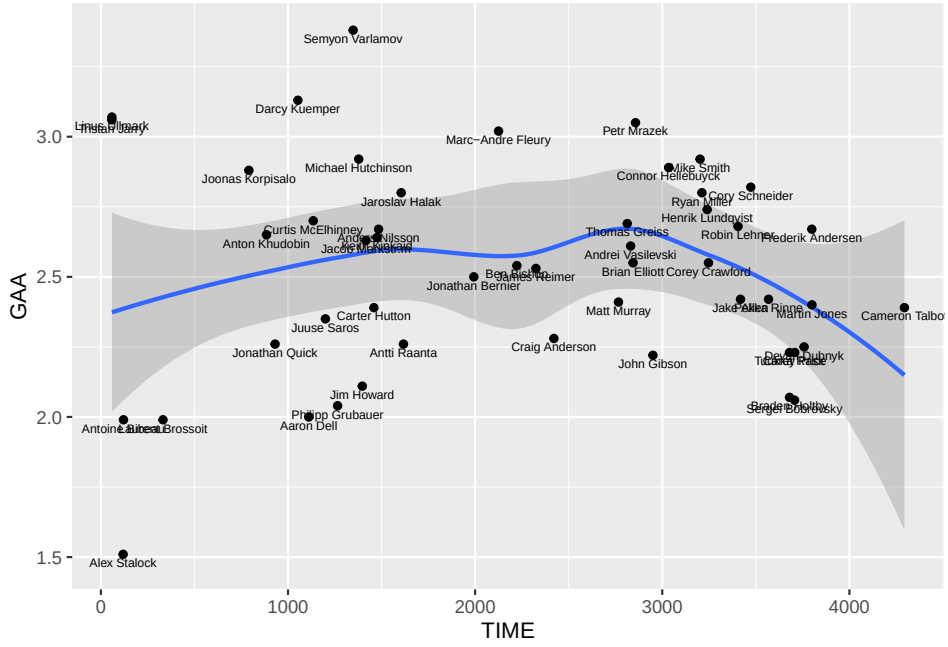
3.1 Goalies

The performance of goalies is found on this page http://www.quanthockey.com/nhl/seasons/2016-17-active-nhl-goalies-stats.html.

```
url = 'http://www.quanthockey.com/nhl/seasons/2016-17-active-nhl-goalies-stats.html'
goalie = read_html(url) %>% html_node("table") %>% html_table()
str(goalie)
#> 'data.frame':    50 obs. of  16 variables:
#>  $ Rk  : int  1 2 3 4 5 6 7 8 9 10 ...
#>  $     : logi  NA NA NA NA NA NA ...
#>  $ Name: chr  "Cameron Talbot" "Frederik Andersen" "Martin Jones" "Devan Dubnyk" ...
#>  $ Team: chr  "EDM" "TOR" "SJS" "MIN" ...
#>  $ Age : int  29 27 26 30 29 28 27 29 34 26 ...
#>  $ GP  : int  73 66 65 65 65 63 63 62 61 61 ...
#>  $ GAA : num  2.39 2.67 2.4 2.25 2.23 2.06 2.07 2.23 2.42 2.42 ...
#>  $ SV% : num  0.919 0.918 0.912 0.924 0.915 0.932 0.925 0.923 0.918 0.915 ...
#>  $ W   : int  42 33 35 40 37 41 42 37 31 33 ...
#>  $ L   : int  22 16 23 19 20 17 13 20 19 20 ...
#>  $ SO  : int  7 4 2 5 8 7 9 3 3 4 ...
#>  $ TIME: int  4294 3799 3800 3758 3680 3707 3680 3708 3568 3418 ...
#>  $ G   : int  0 0 0 0 0 0 0 0 0 0 ...
#>  $ A   : int  0 1 0 0 2 0 0 1 0 1 ...
#>  $ P   : int  0 1 0 0 2 0 0 1 0 1 ...
#>  $ PIM : int  4 16 0 10 0 8 0 4 4 4 ...
```

We can make a scatter plot of the time on ice (`TIME`) against Goals Against Average (`GAA`), adding a smooth fit and labels.

```
ggplot(goalie, aes(x=TIME, y= GAA)) + geom_smooth() +
  geom_point() + geom_text(aes(label=Name), size=2, nudge_y=-.03)
```



4 NFL Football

4.1 Rushing Yards

The regular season rushing yards for NFL players in 2016 is here http://www.nfl.com/stats/categorystats?tabSeq=0&statisticCategory=RUSHING&conference=null&season=2016&seasonType=REG&d-447263-s=RUSHING_YARDS&d-447263-o=2&d-447263-n=1. Notice that this page shows only the 1st page out of 7. First, we will get this first table loaded in, then we can grab the others and combine them.

```
url = 'http://www.nfl.com/stats/categorystats?tabSeq=0&statisticCategory=RUSHING&conference=null&seas

nfl.1 = read_html(url) %>% html_node("table") %>% html_table()
glimpse(nfl.1)
#> Observations: 50
#> Variables: 16
#> $ Rk      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 11, 13, 14, 15, 16,...
#> $ Player  <chr> "Ezekiel Elliott", "Saquon Barkley", "Todd Gurley", "J...
#> $ Team    <chr> "DAL", "NYG", "LA", "CIN", "SEA", "CAR", "TEN", "WAS",...
#> $ Pos     <chr> "RB", "RB", "RB", "RB", "RB", "RB", "RB", "RB", "RB", ...
#> $ Att     <int> 304, 261, 256, 237, 247, 219, 215, 251, 192, 192, 215,...
#> $ `Att/G` <dbl> 20.3, 16.3, 18.3, 16.9, 17.6, 13.7, 13.4, 15.7, 12.8, ...
#> $ Yds     <chr> "1,434", "1,307", "1,251", "1,168", "1,151", "1,098", ...
#> $ Avg     <dbl> 4.7, 5.0, 4.9, 4.9, 4.7, 5.0, 4.9, 4.2, 5.4, 5.2, 4.5,...
#> $ `Yds/G` <dbl> 95.6, 81.7, 89.4, 83.4, 82.2, 68.6, 66.2, 65.1, 69.1, ...
#> $ TD      <int> 6, 11, 17, 8, 9, 7, 12, 7, 9, 8, 12, 5, 7, 9, 6, 9, 10...
#> $ Lng     <chr> "41", "78T", "36", "51", "61", "59", "99T", "90T", "65...
```

```
#> $ `1st`    <int> 73, 50, 70, 60, 61, 53, 51, 47, 49, 47, 56, 36, 51, 53...
#> $ `1st%`   <dbl> 24.0, 19.2, 27.3, 25.3, 24.7, 24.2, 23.7, 18.7, 25.5, ...
#> $ `20+`    <int> 11, 16, 11, 11, 8, 6, 4, 5, 8, 11, 9, 6, 2, 5, 4, 7, 9...
#> $ `40+`    <int> 1, 7, 0, 3, 1, 3, 2, 3, 3, 4, 0, 2, 1, 1, 0, 1, 0, 1, ...
#> $ FUM      <int> 6, 0, 0, 0, 3, 2, 0, 2, 0, 0, 3, 0, 2, 2, 1, 1, 0, 0, ...
```

This looks decent, but the `Yds` are listed as a character vector, not numeric, because of the commas. Also, the `Lng` column has appends a `T` for touchdown. These can be cleaned up:

```
mutate(nfl.1,
       Yds = parse_number(Yds),                    # eliminates the ,
       LngTD = ifelse(str_detect(Lng, 'T'), 1, 0), # add TD column
       Lng = parse_number(Lng)    # remove T's and convert to numeric
       ) %>% glimpse()
```

4.1.1 Combining Tables

Remember, we only got the 1st page but we want all 7. If we check out the links for the other pages, we can see the pattern:

pg1 = 'http://www.nfl.com/stats/categorystats?tabSeq=0&season=2018&seasonType=REG&experience=&Submit=Go&archive=false&conference=null&statisticCategory=RUSHING&d-447263-p=1&qualified=false'

pg2 = 'http://www.nfl.com/stats/categorystats?tabSeq=0&season=2018&seasonType=REG&experience=&Submit=Go&archive=false&conference=null&statisticCategory=RUSHING&d-447263-p=2&qualified=false'

```
compare = str_split_fixed(c(pg1, pg2), '', n=str_length(pg1))
which(compare[1,] != compare[2,])
#> [1] 165
```

So the 165 character indicates the page number. We can use `str_c()` to create the required url for any page

```
url1 = 'http://www.nfl.com/stats/categorystats?tabSeq=0&season=2018&seasonType=REG&experience=&Submit
url2 = '&qualified=false'
page2 = str_c(url1, 2, url2)
identical(pg2, page2)
#> [1] TRUE
```

Now we can read all 7 tables and combine with `bind_rows()`. We can start by combining the first two tables, then we will make a function to do it all.

```
pg1 = str_c(url1, 1, url2)
pg2 = str_c(url1, 2, url2)
t1 = read_html(pg1) %>% html_node("table") %>% html_table()
t2 = read_html(pg2) %>% html_node("table") %>% html_table()
bind_rows(t1, t2)
#> Error: Column `Yds` can't be converted from character to integer
```

We have a problem: looks like `t2` correctly set `Yds` to an integer since these players do not have enough yards to get a comma. So we will have to do the changes for each table.

```
t1 = read_html(pg1) %>% html_node("table") %>% html_table() %>%
      mutate(
       Yds = parse_number(Yds),                    # eliminates the ,
       LngTD = ifelse(str_detect(Lng, 'T'), 1, 0), # add TD column
       Lng = parse_number(Lng)    # remove T's and convert to numeric
       )
t2 = read_html(pg2) %>% html_node("table") %>% html_table() %>%
      mutate(
```

```
       #Yds = parse_number(Yds),                    # eliminates the ,
       LngTD = ifelse(str_detect(Lng, 'T'), 1, 0), # add TD column
       Lng = parse_number(Lng)    # remove T's and convert to numeric
       )
bind_rows(t1, t2) %>% tail()
#>      Rk           Player Team Pos Att Att/G Yds Avg Yds/G TD Lng 1st 1st%
#> 95   95   Aaron Rodgers   GB  QB  43   2.7 269 6.3  16.8  2  23  20 46.5
#> 96   96      Alex Smith  WAS  QB  41   4.1 168 4.1  16.8  1  22  15 36.6
#> 97   97   Jalen Richard  OAK  RB  55   3.4 259 4.7  16.2  1  30  11 20.0
#> 98   98 Damien Williams   KC  RB  50   3.1 256 5.1  16.0  4  25  15 30.0
#> 99   99   Kalen Ballage  MIA  RB  36   3.0 191 5.3  15.9  1  75  10 27.8
#> 100 100  Justin Jackson  LAC  RB  50   3.8 206 4.1  15.8  2  20  14 28.0
#>     20+ 40+ FUM LngTD
#> 95    3   0   2     0
#> 96    1   0   3     0
#> 97    3   0   2     0
#> 98    3   0   1     0
#> 99    1   1   1     1
#> 100   1   0   0     0
```

Now for the function:

```
get_data <- function(pages){
  url1 = 'http://www.nfl.com/stats/categorystats?tabSeq=0&season=2018&seasonType=REG&experience=&Subr
  url2 = '&qualified=false'
  X = tibble()
  for(i in pages){
    url = str_c(url1, i, url2)    # add i for page number
    tb = read_html(url) %>% html_node("table") %>% html_table() %>%
      mutate(
       Yds = as.character(Yds), # convert to character so parse_number() will always work
       Yds = parse_number(Yds),                    # eliminates the ,
       LngTD = ifelse(str_detect(Lng, 'T'), 1, 0), # add TD column
       Lng = as.character(Lng),  # convert to character so parse_number() will always work
       Lng = parse_number(Lng)    # remove T's and convert to numeric
       )
    X = bind_rows(X, tb)
  }
return(X)
}
```

Now to grab all the data:

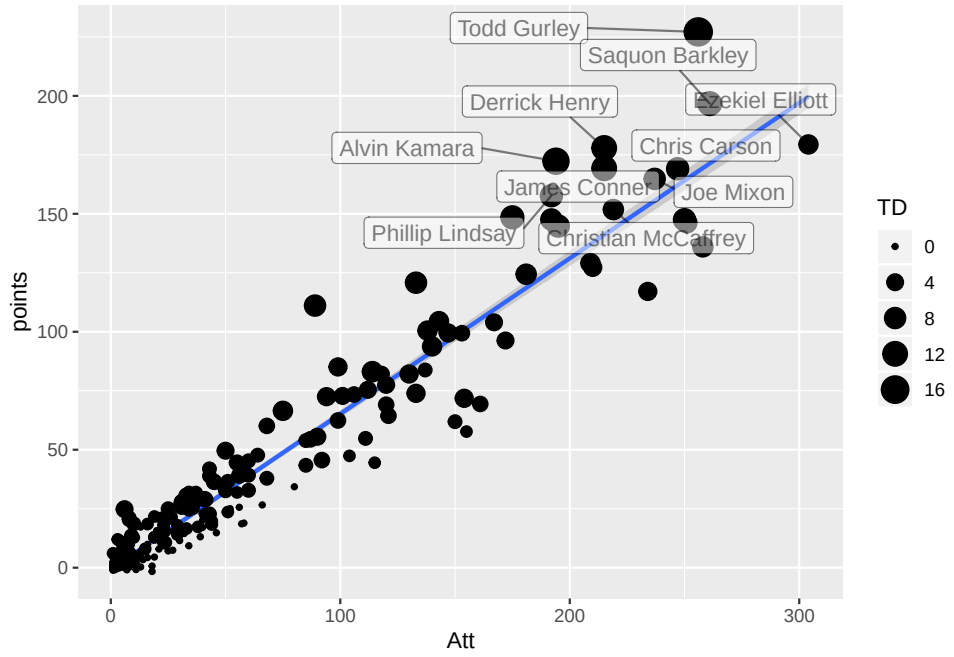
```
NFL = get_data(pages = 1:7)
```

4.1.2 Fantasy Points

In fantasy football, each player gets points for their performance. For example, each touchdown (`TD`) is worth 6 points, every 10 yards (`Yds`) is worth 1 point. We will add the points column and plot points against attempts (`Att`)

```
NFL = mutate(NFL, points = 6*TD + Yds/10)

library(ggrepel)        # for function geom_label_repel()
ggplot(NFL, aes(Att, points)) +
  geom_smooth(method='lm') +
  geom_point(aes(size=TD)) +
  geom_label_repel(data=. %>% filter(min_rank(-points)<=10), aes(label=Player),
                   alpha=.5)
```



5 API

5.1 Introduction to API's

An API or application programming interface, is way to access data from a website. API's can allow a machine to view and edit data, just like a person can by loading pages and submitting forms.

We do not have time to investigate in this course, so I will leave you with two references:

- The `httr` package https://cran.r-project.org/web/packages/httr/vignettes/quickstart.html
- The zapier learn api website: https://zapier.com/learn/apis/chapter-1-introduction-to-apis/
- Many R packages are API's: http://data.library.virginia.edu/using-data-gov-apis-in-r/, gapminder, census data, long list