# 15 - HTML Data Tables

*ST 597 | Spring 2016*
*University of Alabama*

*15-webdata.pdf*

## Contents



---

**Required Packages and Data**

```
library(tidyverse)
library(stringr)
library(rvest)
```

# 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 http://blog.rstudio.org/2014/11/24/rvest-easy-web-scraping-with-r/ 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
glimpse(rushing)
```

```
#> Observations: 100
#> Variables: 12
#> $         <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1
#> $ Name    <chr> "D'Onta Foreman", "Donnel Pumphrey", "Aaron Jones", "Christian M
#> $ Team    <chr> "Texas", "SDSU", "UTEP", "Stanford", "BYU", "FSU", "Wyoming", "B
#> $ Yr      <chr> "JR", "SR", "JR", "JR", "SR", "JR", "JR", "JR", "SR", "SR", "SO"
#> $ Pos     <chr> "RB", "RB", "RB", "RB", "RB", "RB", "RB", "RB", "RB", "RB", "RB"
#> $ G       <int> 11, 14, 12, 11, 10, 13, 14, 13, 13, 13, 10, 13, 13, 12, 13, 13,
#> $ Att     <int> 323, 349, 229, 253, 234, 288, 349, 314, 258, 237, 209, 260, 232,
#> $ Yards   <int> 2028, 2133, 1773, 1603, 1375, 1765, 1860, 1709, 1629, 1621, 1224
#> $ Avg.    <dbl> 6.28, 6.11, 7.74, 6.34, 5.88, 6.13, 5.33, 5.44, 6.31, 6.84, 5.86
#> $ TD      <int> 15, 17, 17, 13, 12, 19, 22, 23, 18, 27, 7, 21, 17, 7, 18, 15, 15
#> $ Att/G   <dbl> 29.36, 24.93, 19.08, 23.00, 23.40, 22.15, 24.93, 24.15, 19.85, 1
#> $ Yards/G <dbl> 184.4, 152.4, 147.8, 145.7, 137.5, 135.8, 132.9, 131.5, 125.3, 1
```

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.

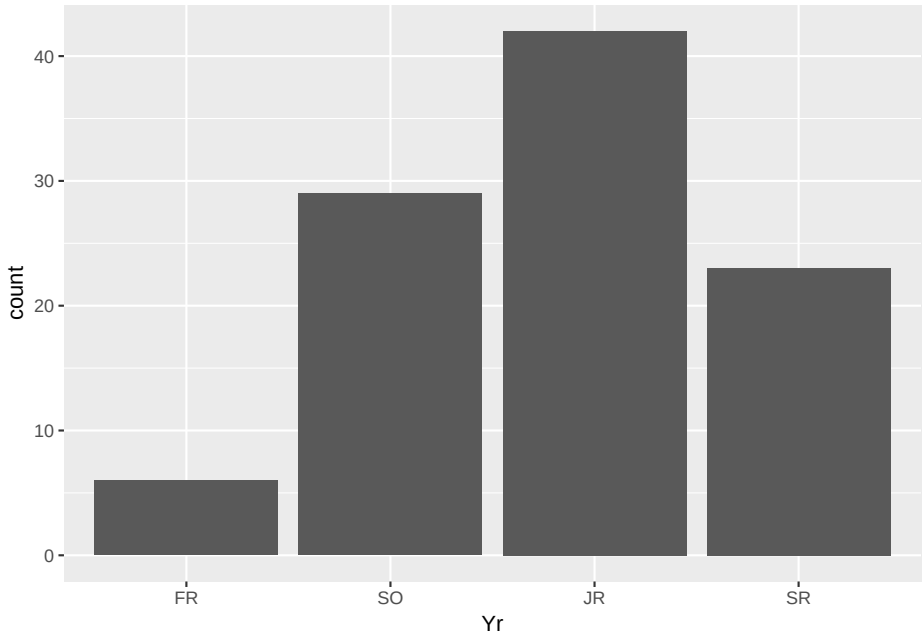
The `read_html()` function produces an XML document, similar to what the browser gives (but some exceptions, like with javascript generated data). The `html_node()` node that is tagged with `"table"`. In this case, there is only one table, but we will have to be more specific when a webpage has more 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 × 2
#>       Yr     n
#>   <fctr> <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

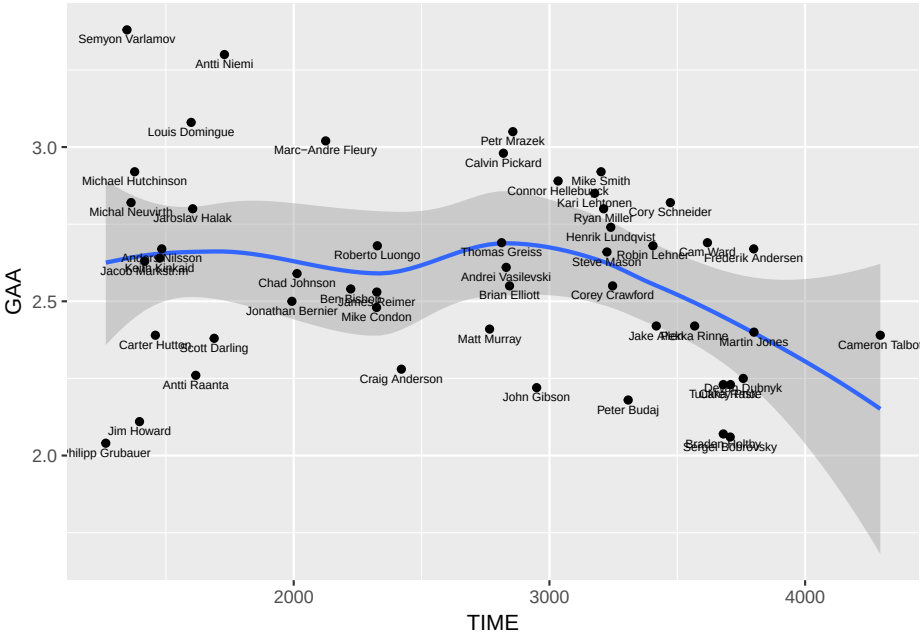
## 3.1 Goalies

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  15 variables:
#>  $ Rk  : int  1 2 3 4 5 6 7 8 9 10 ...
#>  $ Age : int  29 27 26 30 29 28 27 29 32 34 ...
#>  $     : logi  NA NA NA NA NA NA ...
#>  $ Name: chr  "Cameron Talbot" "Frederik Andersen" "Martin Jones" "Devan Dubnyk" ...
#>  $ 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 2.69 2.42 ...
#>  $ SV% : num  0.919 0.918 0.912 0.924 0.915 0.932 0.925 0.923 0.905 0.918 ...
#>  $ W   : int  42 33 35 40 37 41 42 37 26 31 ...
#>  $ L   : int  22 16 23 19 20 17 13 20 22 19 ...
#>  $ SO  : int  7 4 2 5 8 7 9 3 2 3 ...
#>  $ TIME: int  4294 3799 3800 3758 3680 3707 3680 3708 3618 3568 ...
#>  $ G   : int  0 0 0 0 0 0 0 0 0 0 ...
#>  $ A   : int  0 1 0 0 2 0 0 1 1 0 ...
#>  $ P   : int  0 1 0 0 2 0 0 1 1 0 ...
#>  $ PIM : int  4 16 0 10 0 8 0 4 6 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)
#> Warning in grid.Call.graphics(L_text, as.graphicsAnnot(x$label), x$x, x$y, : conversion
#> failure on 'Jacob Markström' in 'mbcsToSbcs': dot substituted for <f6>
```




# 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

nfl.1 = read_html(url) %>% html_node("table") %>% html_table()
str(nfl.1)
#> 'data.frame':    50 obs. of  16 variables:
#>  $ Rk    : int  1 2 3 4 5 6 7 8 9 10 ...
#>  $ Player: chr  "Ezekiel Elliott" "Jordan Howard" "DeMarco Murray" "Jay Ajayi" ...
#>  $ Team  : chr  "DAL" "CHI" "TEN" "MIA" ...
#>  $ Pos   : chr  "RB" "RB" "RB" "RB" ...
#>  $ Att   : int  322 252 293 260 261 234 293 299 227 268 ...
#>  $ Att/G : num  21.5 16.8 18.3 17.3 21.8 15.6 18.3 18.7 14.2 19.1 ...
#>  $ Yds   : chr  "1,631" "1,313" "1,287" "1,272" ...
```

```
#>  $ Avg   : num  5.1 5.2 4.4 4.9 4.9 5.4 4.2 3.9 4.8 4 ...
#>  $ Yds/G : num  108.7 87.5 80.4 84.8 105.7 ...
#>  $ TD    : int  15 6 9 8 7 13 16 18 11 5 ...
#>  $ Lng   : chr  "60T" "69" "75T" "62T" ...
#>  $ 1st   : int  91 70 64 60 69 55 72 67 61 52 ...
#>  $ 1st%  : num  28.3 27.8 21.8 23.1 26.4 23.5 24.6 22.4 26.9 19.4 ...
#>  $ 20+   : int  14 10 4 10 4 11 6 7 7 7 ...
#>  $ 40+   : int  3 2 2 4 1 3 2 3 2 1 ...
#>  $ FUM   : int  5 1 3 4 3 3 5 2 1 2 ...
```

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=2016&seasonType=REG&experience=&Submit=Go&archiv
447263-p=1&qualified=false'

pg2 = 'http://www.nfl.com/stats/categorystats?tabSeq=0&season=2016&seasonType=REG&experience=&Submit=Go&archiv
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=2016&seasonType=REG&experien
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 in eval(expr, envir, enclos): Can not automatically convert from character to inte
```

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% 20+ 40+ FUM
#> 95   95     James Starks   GB  RB  63   7.0 145 2.3  16.1  0  11   9 14.3   0   0   0
#> 96   96 Kerwynn Williams  ARI  RB  18   1.8 157 8.7  15.7  2  49   8 44.4   3   1   0
#> 97   97     Antone Smith   TB  RB  10   3.3  47 4.7  15.7  0   8   1 10.0   0   0   2
#> 98   98      Corey Grant  JAX  RB  32   2.9 164 5.1  14.9  1  57   6 18.8   1   1   0
#> 99   99    Peyton Barber   TB  RB  55   3.7 223 4.1  14.9  1  44  10 18.2   1   1   0
#> 100 100   Kenneth Farrow   SD  RB  60   4.6 192 3.2  14.8  0  11  10 16.7   0   0   1
#>     LngTD
#> 95      0
#> 96      1
#> 97      0
#> 98      1
#> 99      1
#> 100     0
```

Now for the function:

```
get_data <- function(pages){
  url1 = 'http://www.nfl.com/stats/categorystats?tabSeq=0&season=2016&seasonType=REG&experi
  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 = 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
       )
    X = bind_rows(X, tb)
  }
return(X)
}
```

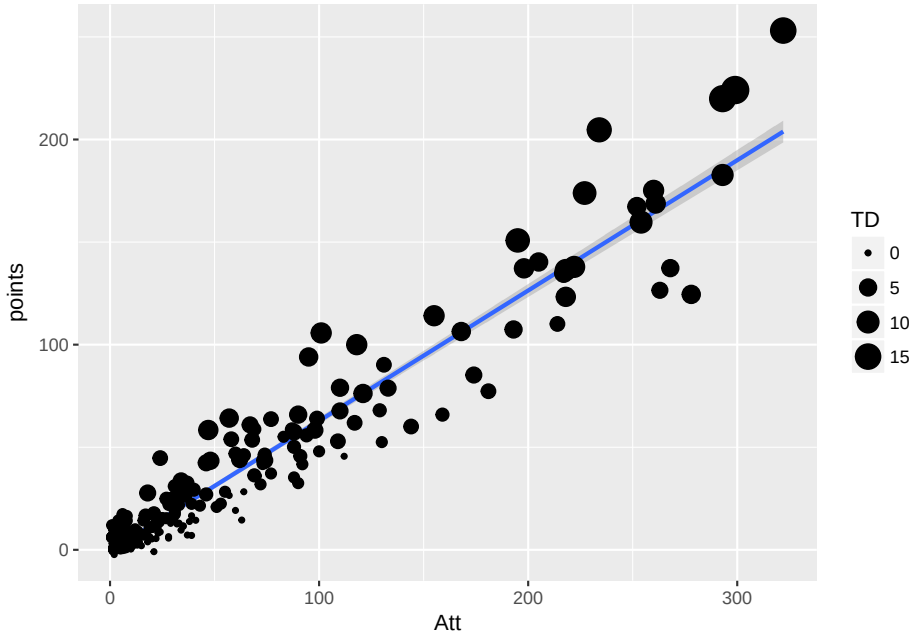
Now to grab all the data:

```
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)
ggplot(NFL, aes(Att, points)) +
  geom_smooth(method='lm') +
  geom_point(aes(size=TD))
```




# 5 Baseball

## 5.1 3000 Hit Club

How many MLB baseball players have at least 3000 hits in their career? When you don't know, check wikipedia http://en.wikipedia.org/wiki/3,000_hit_club. We can try our formula, but it will give us the wrong table and throws a warning:

```
url = 'http://en.wikipedia.org/wiki/3,000_hit_club'
read_html(url) %>% html_node("table") %>% html_table()
```

When there are more than one table, we have to be more specific about which table we want.

There are a two possible tables. The one we want is `table.wikitable.sortable` (notice we can truncate the full class label).

### 5.1.1 Using Chrome or Firefox

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. Here the selector is `table.wikitable.sortable`, which is a table with `class = wikitable.sortable`. Adding this into the `html_node()` function gives us the table we are looking for

```
url = 'http://en.wikipedia.org/wiki/3,000_hit_club'
hits = read_html(url) %>% html_node("table.wikitable.sortable") %>% html_table()
```

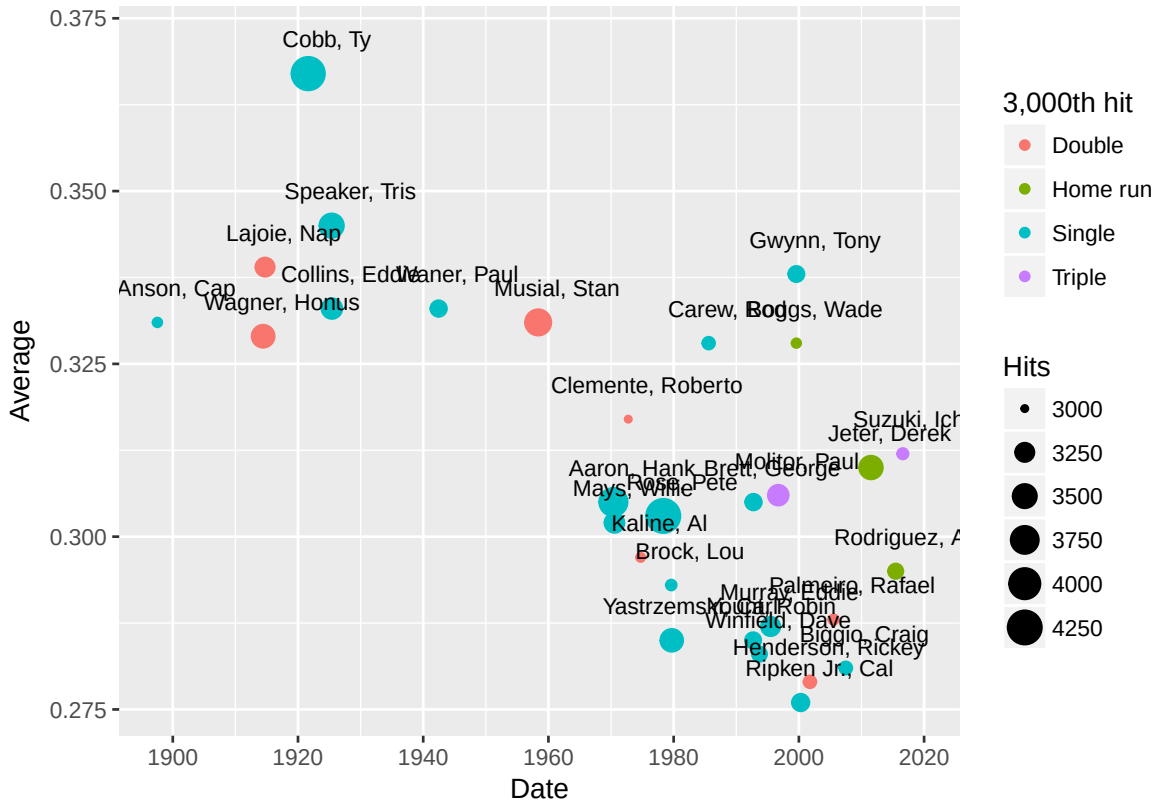
## 5.2 Clean it up

We've got problems with this one. We can use a mix of parsing, substrings, and regular expressions

```
hits.clean =
  hits %>%
  mutate(Hits = parse_number(Hits),            # remove commas and refs
         Date = as.Date(str_sub(Date, 9, 19)), # extract date
         `3,000th hit` = str_extract(`3,000th hit`, "Single|Double|Triple|Home run"),
         Player = str_extract(Player, "[A-Z].+, [A-Z][a-z]+"),
         Seasons = str_replace_all(Seasons, str_sub(hits$Seasons[1],5,7), '-')) %>%
  select(-Ref)
glimpse(hits.clean)
#> Observations: 30
#> Variables: 7
#> $ Player      <chr> "Rose, Pete", "Cobb, Ty", "Aaron, Hank", "Musial, Stan", "Speaker..
#> $ Hits        <dbl> 4256, 4191, 3771, 3630, 3514, 3465, 3430, 3419, 3319, 3314, 3283,..
#> $ Average     <dbl> 0.303, 0.367, 0.305, 0.331, 0.345, 0.310, 0.329, 0.285, 0.306, 0...
#> $ Date        <date> 1978-05-05, 1921-08-19, 1970-05-17, 1958-05-13, 1925-05-17, 2011..
#> $ Team        <chr> "Cincinnati Reds", "Detroit Tigers", "Atlanta Braves", "St. Louis..
#> $ Seasons     <chr> "1963-86", "1905-28", "1954-76", "1941-44, 1946-63", "1907-28", "..
#> $ 3,000th hit <chr> "Single", "Single", "Single", "Double", "Single", "Home run", "Do..
```

## 5.3 Graphics

```
ggplot(hits.clean, aes(x=Date, y=Average)) +
  geom_point(aes(size=Hits, color=`3,000th hit`)) +
  geom_text(aes(label=Player), nudge_x=3*365, nudge_y=.005, size=3)
```




# 6 API

## 6.1 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