Data Journalism Guidelines

Bruin Sports Analytics

Ethan Allavarpu Kathir Ilango

Contents

1 Why Data Journalism?

At Bruin Sports Analytics, especially with the Data Journalism team, we highlight the importance of data journalism, not just journalism. What, exactly, is data journalism? How is it different from an article in the sports section of the Daily Bruin?

The Daily Bruin has its own articles about sports, but if we closely compare their articles to ours on the website, their articles seem to be more descriptive, using much more expressive language as opposed to hard facts. With Bruin Sports Analytics, though, we emphasize using evidence, facts, data and statistics, to support our arguments as they are linked to the world of sports.

For data journalism, we want to present evidence to interweave within our articles. We want the focus to be on statistics, our data, with the journalism and writing simply connecting the dots and helping guide the reader to our central argument. We want to take the data, modify it, and present it visually and numerically to substantiate our claims. Ultimately it's data journalism, not journalism with data: the data comes first, then the journalism follows.

2 Collecting and Importing Data

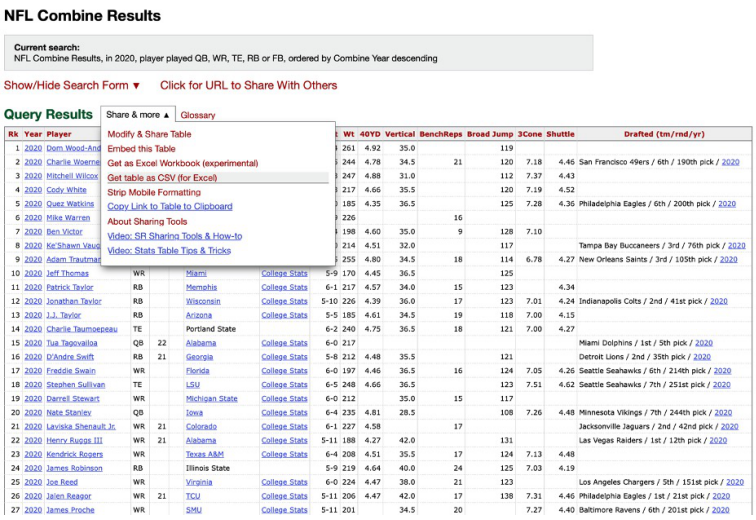
When we collect data, we need to ensure they are useful as well as reliable. With respect to utility, we should search for specific data relevant to our topic: for example, "nfl combine statistics for qbs" will more likely yield the proper data than "combine stats" if the sole focus is quarterbacks. While this appears obvious, it also requires us to define, with clarity, the question of our article: what exactly do you want to answer with your article? Once you can clarify that question, finding the data becomes easier.

Reliability is of equal importance for data. When gathering data, we want to ensure that the data itself is correct. Therefore, though seemingly obvious, try to stick to well-known data sources, including—but not limited to—ESPN, Pro Football Reference, nba.com, nfl.com, etc.

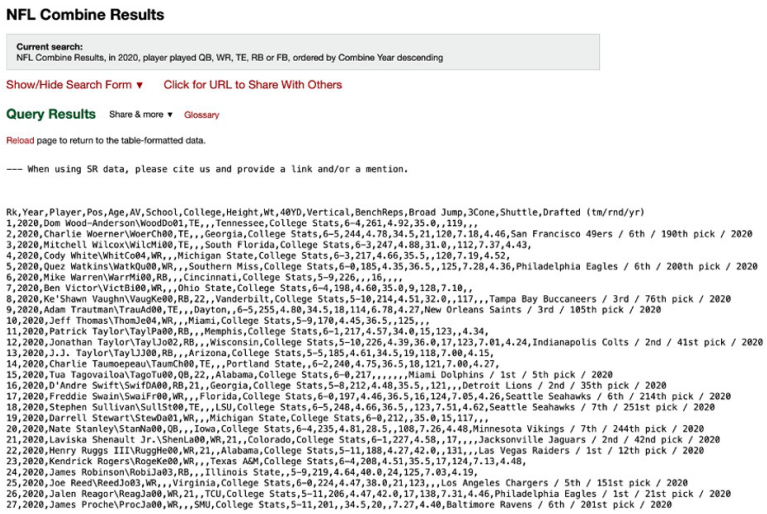
Now that we have found our data, the first step is gathering it for us to use and analyze. There are multiple ways to do this, including a method known as web scraping (described in this PDF below). Here, I will describe how to get .csv files for your computer, using my previous article (Fantasy Football and the NFL Combine) as an example.

Some websites (in my case, Pro Football Reference) have easy-to-access downloadable .csv files. Once I have found the specific data table, there is an easy link to downloading the .csv file.

Now, for my computer, I have two options: I can either choose (1) "Get table as CSV (for Excel)" or (2) "Get as Excel Workbook (experimental)." If we choose the second option, we will need to open the Excel file and then convert it to a CSV file.



If you choose "Get table as CSV (for Excel)", it will appear as a giant table of numbers and commas.



NFL Combine Results

Current search:
NFL Combine Results, in 2020, player played QB, WR, TE, RB or FB, ordered by Combine Year descending

Show/Hide Search Form ▾ Click for URL to Share With Others

Query Results Share & more ▾ Glossary

Reload page to return to the table-formatted data.

--- When using SR data, please cite us and provide a link and/or a mention.

```
Rk,Year,Player,Pos,Age,AV,School,College,Height,Wt,40YD,Vertical,BenchReps,Broad Jump,3Cone,Shuttle,Drafted (tm/rnd/yr)
1,2020,Dom Wood-Anderson\WoodDo01,TE,,,Tennessee,College Stats,6-4,261,4.92,35.0,,119,,,
2,2020,Charlie Woerner\WoerCh00,TE,,,Georgia,College Stats,6-5,244,4.78,34.5,21,120,7.18,4.46,San Francisco 49ers / 6th / 190th pick / 2020
3,2020,Mitchell Wilcox\WilcMi00,TE,,,South Florida,College Stats,6-3,247,4.88,31.0,,112,7.37,4.43,
4,2020,Cody White\WhitCo04,WR,,,Michigan State,College Stats,6-3,217,4.66,35.5,,120,7.19,4.52,
5,2020,Quez Watkins\WatkQu00,WR,,,Southern Miss,College Stats,6-0,185,4.35,36.5,,125,7.28,4.36,Philadelphia Eagles / 6th / 200th pick / 2020
6,2020,Mike Warren\WarrMi00,RB,,,Cincinnati,College Stats,5-9,226,,,16,,,,
7,2020,Ben Victor\VictBi00,WR,,,Ohio State,College Stats,6-4,198,4.60,35.0,9,128,7.10,,
8,2020,Ke'Shawn Vaughn\VaugKe00,RB,22,,Vanderbilt,College Stats,5-10,214,4.51,32.0,,117,,,Tampa Bay Buccaneers / 3rd / 76th pick / 2020
9,2020,Adam Trautman\TrauAd00,TE,,,Dayton,,6-5,255,4.80,34.5,18,114,6.78,4.27,New Orleans Saints / 3rd / 105th pick / 2020
10,2020,Jeff Thomas\ThomJe04,WR,,,Miami,College Stats,5-9,170,4.45,36.5,,125,,,
11,2020,Patrick Taylor\TaylPa00,RB,,,Memphis,College Stats,6-1,217,4.57,34.0,15,123,,4.34,
12,2020,Jonathan Taylor\TaylJo02,RB,,,Wisconsin,College Stats,5-10,226,4.39,36.0,17,123,7.01,4.24,Indianapolis Colts / 2nd / 41st pick / 2020
13,2020,J.J. Taylor\TaylJJ00,RB,,,Arizona,College Stats,5-5,185,4.61,34.5,19,118,7.00,4.15,
14,2020,Charlie Taumoepeau\TaumCh00,TE,,,Portland State,,6-2,240,4.75,36.5,18,121,7.00,4.27,
15,2020,Tua Tagovailoa\TagoTu00,QB,22,,Alabama,College Stats,6-0,217,,,,,,Miami Dolphins / 1st / 5th pick / 2020
16,2020,D'Andre Swift\SwifDA00,RB,21,,Georgia,College Stats,5-8,212,4.48,35.5,,121,,,Detroit Lions / 2nd / 35th pick / 2020
17,2020,Freddie Swain\SwaiFr00,WR,,,Florida,College Stats,6-0,197,4.46,36.5,16,124,7.05,4.26,Seattle Seahawks / 6th / 214th pick / 2020
18,2020,Stephen Sullivan\SullSt00,TE,,,LSU,College Stats,6-5,248,4.66,36.5,,123,7.51,4.62,Seattle Seahawks / 7th / 251st pick / 2020
19,2020,Darrell Stewart\StewDa01,WR,,,Michigan State,College Stats,6-0,212,,35.0,15,117,,,
20,2020,Nate Stanley\StanNa00,QB,,,Iowa,College Stats,6-4,235,4.81,28.5,,108,7.26,4.48,Minnesota Vikings / 7th / 244th pick / 2020
21,2020,Laviska Shenault Jr.\ShenLa00,WR,21,,Colorado,College Stats,6-1,227,4.58,,17,,,,Jacksonville Jaguars / 2nd / 42nd pick / 2020
22,2020,Henry Ruggs III\RuggHe00,WR,21,,Alabama,College Stats,5-11,188,4.27,42.0,,131,,,Las Vegas Raiders / 1st / 12th pick / 2020
23,2020,Kendrick Rogers\RogeKe00,WR,,,Texas A&M,College Stats,6-4,208,4.51,35.5,17,124,7.13,4.48,
24,2020,James Robinson\RobiJa03,RB,,,Illinois State,,5-9,219,4.64,40.0,24,125,7.03,4.19,
25,2020,Joe Reed\ReedJo03,WR,,,Virginia,College Stats,6-0,224,4.47,38.0,21,123,,,Los Angeles Chargers / 5th / 151st pick / 2020
26,2020,Jalen Reagor\ReagJa00,WR,21,,TCU,College Stats,5-11,206,4.47,42.0,17,138,7.31,4.46,Philadelphia Eagles / 1st / 21st pick / 2020
27,2020,James Proche\ProcJa00,WR,,,SMU,College Stats,5-11,201,,34.5,20,,7.27,4.40,Baltimore Ravens / 6th / 201st pick / 2020
```

Once in this format, you can copy and paste this data into a .txt file, then save that file as a .csv file for us to use. Then, once we save the file, we can import the data into R using the `read.csv()` function, as shown below:

```
qb_combine_1 <- read.csv("2000QBcombine.csv", stringsAsFactors = FALSE)
```

The `stringsAsFactors = FALSE` argument serves the purpose of leaving characters, such as names, as character vectors instead of factors; making them factors does not make sense for player names. If we wish to switch to factors later in the process, we can do so manually.

3 Cleaning and Parsing Data

Once we have imported and coalesced the data, then we have to decide which variables and which observations to keep and which to filter out of the final data sets to use for analysis.

3.1 %>% (The pipe operator)

This operator helps make code easier to follow and read, especially for the less code-inclined. In addition, it provides the ability to shorten your code. Essentially, the pipe takes the output of the left side of the operator and uses that as the input for the right side.

3.2 summarize() [or summarise()]

The `summarize()` function helps add columns to your data frame with important values. The typical syntax is:

```
summarize(data.frame, newvar = calculation)
```

where you input your data frame in for `data.frame` and state the variable you wish to create (say most bench reps) in for `newvar` (e.g. `max_bench`). Then on the other side (where it says calculation), you add how you can get the values for each observation in the data frame (e.g. `max(bench)`). Therefore, when completed, the syntax would look something like this:

```
summarize(qb_combine_1, max_bench = max(bench))
```

Note: To save the additional variable to the initial data frame, make sure to use the assignment operator as well

```
qb_combine_1 <- summarize(qb_combine_1, max_bench = max(bench))
```

3.3 select()

The `select()` function helps trim the data frame by selecting specific columns (variables) of the data frame to keep. The typical syntax is:

```
select(data.frame, var_a, var_b, ...)
```

where you input your data frame for `data.frame` and then list the variables (column names), separated by commas, which you wish to keep.

Note: If you want to save multiple columns that appear sequentially in a data frame, you can optionally use the colon (:) to save space. State the first variable of the sequence, then the colon, then the last variable of that consecutive sequence to get all the variables in between the two variables, ends included.

3.4 filter()

The `filter()` function helps trim the data frame by filtering specific observations (rows) of the data frame to keep based on the criteria you mention. The typical syntax is:

```
filter(data.frame, condition(s))
```

where you input the data into `data.frame` and state the conditions by which you want to filter in place of `condition(s)`. For example, if I wish to filter the QB combine participants into only those who had at least 10 bench press reps and a 40-yard dash time of 5.5 seconds or faster, the code would look like the following:

```
filter(qb_combine_1, Bench >= 10, Forty_Yard <= 5.5)
```

3.5 dplyr Cheat Sheet

For a detailed dplyr cheat sheet, go to https://rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

4 Presenting Data: Visuals

With regards to visuals, there are two main methods in R: base R and ggplot. In this document, I will focus on base R graphics, with a link to a ggplot cheat sheet at the end.

4.1 plot()

This function is perhaps the most versatile in base R graphics. Generally speaking, we will use `plot()` to plot two numeric vectors against one another. There are two main ways to do this: (1) `plot(x, y)` or (2) `plot(y ~ x)`. Either method will plot the vector x on the x-axis and the y vector on the y-axis. There are also some (optional) arguments hidden within the `plot()` function for beautification.

- `xlab`: Change the label for the x-axis
- `ylab`: Change the label for the y-axis
- `main`: Change (or add) the title
- `xlim`: Change the range for the x-axis variable
- `ylim`: Change the range for the y-axis variable
- `pch`: Change the shape of the plotted points
- `col`: Change the color of the plotted points

4.2 hist()

The `hist()` function can be used to create a histogram for a single numerical (quantitative) variable to visualize its distribution. Typically, we will do this with the code `hist(x)` where x represents the numerical vector/data. The optional arguments of `hist()` are similar to those of `plot()`, with the exception of `pch`. There are two additional arguments which you should know when using `hist()`.

- `breaks`: Change the cutoffs for the bins of the histogram
  - *Note: The input should either be the number of bins you want or a vector of values at which to have the breaks in the bins*
- `freq`: Decide whether to plot frequencies/number of occurrences (`freq = TRUE`) or to plot densities (`freq = FALSE`)
  - *Note: If plotting overlapping histograms of two groups with different population sizes, `freq = FALSE` is highly recommended to provide an accurate comparison of the groups. To plot overlapping histograms of data sets, use the `hist()` function twice, then include the argument `add = TRUE` to add this second histogram to the plot of the first histogram.*

4.3 boxplot()

The `boxplot()` function is used to create boxplots for the distribution of a single numerical (quantitative) variable, offering an alternative method to the histogram for displaying the data. The `boxplot()` function may also be used to compare the distribution of data based on a certain characteristic (e.g. SAT scores by state). To plot one quantitative variable without splitting into groups, the syntax would be a simple `boxplot(sat)`. To plot a distribution based on a characteristic, the syntax is similar to the function-style syntax for `plot()` (the second method): `boxplot(sat ~ state)`. With respect to optional arguments for plot beautification, many of the arguments are the same as for the `plot()` function.

4.4 barplot()

The `barplot()` function helps compare the distribution for a categorical (qualitative) variable, such as favorite NFL team. The main method of using this function is `barplot(team)`, which will display a bar plot with all of the eligible categories for the variable along the x-axis. The optional arguments parallel those found for the `plot()` function. Now, the `barplot()` function can also be used for two categorical variables; if you input a matrix, you get a segmented bar plot, with the separate columns representing the columns of the matrix and the segments representing the rows.

Note: If you wish to have side-by-side bar plots instead of segmented bar plots, include the optional argument `beside = TRUE` *into your code.*

4.5 Creating JPEG Images

In order to ensure high-quality plots in your article–rather than blurry screenshots–we recommend including the R plots as JPEG files. To do this, there are two methods:

1. `jpeg()` a. This function creates a JPEG file directly from RStudio. The syntax for this method is `jpeg(filename, width, height)`, which will create a JPEG file in your local directory with the name `filename` and of a set width and height (these arguments are in pixels, not inches). After running this command in the console, run the code you used to create the plot, and the plot image will be saved into the aforementioned file

```
jpeg("fantasyfootball.jpeg", width = 960, height = 720)
plot(scores ~ year, data = fantasyfootball, ...)
```

2. `pdf()` to `jpeg()` a. The `pdf()` function creates a PDF file in a manner similar to the `jpeg()` function. The syntax is `pdf(filename, width, height)` and the process for creating the PDF file is identical to the method described above. After creating your PDF, open that PDF and choose ***File***, then ***Export***, taking care to change the format from PDF to JPEG. *Note: the* `pdf()` *function records height and width in inches, unlike the* `jpeg()` *function (which records in pixels).*

```
pdf("fantasyfootball.jpeg", width = 7, height = 5)
plot(scores ~ year, data = fantasyfootball, ...)
```

Personally, I prefer the second option because the quality of the image is better when initially storing as a PDF before conversion, but you are welcome to choose either option for your own plot images.

4.6 ggplot Cheat Sheet

If you prefer ggplot for visuals, a cheat sheet can be found here: https://rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

5 Writing the Article

When writing an article, there is an important order of operations. In data journalism, you're not setting out to prove a point and then finding data to back it up the way you may try to do in a persuasive essay. Instead, you are asking a fundamental question, finding data and creating visuals to help answer this question, and then analyzing your results. **The data comes first.** Collect the data, create the necessary graphs and visuals, and then write the article around these visuals in order to produce the most objective and nicely flowing piece.

But it is also important to note that data journalism articles are not formal reports; they generally have a more casual feel to them while still maintaining focus on the topic. This when you need to factor your audience into the equation. These articles are not read by professors but rather general fans of sports (often people who do not have a background in statistics), so be sure to constantly ask yourself what kind of writing such an audience would find interesting. Keep their attention span in mind and don't hesitate to make the article have a more conversational tone than a typical paper. On the flip side, don't take this mentality too far by getting too opinionated or attempting to force humor when it is unnecessary.

Finally, providing ample context may be the single most overlooked aspect of writing a data journalism article. While the statistics and discussion of the data are extremely important, this is not a statistics club; it's a sports analytics club. Typically in an article, we're not breaking down the stock market for financial analysts; we're breaking down a game for a fan of it. So ultimately, once you finish presenting any data or visual, it's important to circle back to the sport itself. What do your findings mean in terms of the story of the game? What do your findings prove about a certain athlete? On top of being a data journalist, consider yourself a storyteller and **always** bring your article home by providing clear context and helping the reader understand what they should actually be taking away from all of your numbers and visuals. In most cases, you're likely writing about a certain sport because you are very familiar with it; the best way to put that domain knowledge on display is to tell a complete story rather than throwing visuals and analysis at your reader and leaving it at that.

6 Article Formatting Checklist

Follow these guidelines and include these items before submitting any article for review:

- **Title**
- **Cover Photo**: Include an image that does not display any data
- **High-quality photos**: Use high-quality images. Blurry images take away from the professionalism of your work
- **Cite each image** taken from the internet directly underneath the image
- **Cite all data sources** at the end of the article (even if you did it in-text)
- **Visuals need to be clear on their own**: Include all necessary titles and labels
- **Visuals should maintain aesthetic consistency** across article: Keep the general appearance of your graphs and tables somewhat uniform
- **Don't overuse hyperlinks**: Only add them when appropriate
- **Link code repository** at the end of the article if your code is online (e.g. Github)

7 Web Scraping with BeautifulSoup

Web scraping data might sound intimidating, but it is actually rather simple and can save a great amount of time with regards to data collection for a sports analytics project. With a basic understanding of HTML and BeautifulSoup, you will be able to scrape many sports data websites with just a few lines of code.

7.1 Basic HTML Review

Before jumping into using BeautifulSoup to parse a website's HTML content and collect the specific data we want, it is important to first have a basic understanding of how HTML works. In a nutshell, HTML is a markup language that tells your browser how to format a certain website's content. The elements that make up HTML are called tags; content goes inside of these tags, and these tags can be nested within one another or stacked on top of one another depending on how the creator wants the website to be organized. Here is an example of the most basic HTML imaginable:

```
<html>
</html>
```

This `<html>` tag simply means that whatever goes inside of it is HTML. Continuing on, HTML generally contains `<head>` and `<body>` tags. The body is the more important tag here, as the meaningful content of the website will go in there:

```
<html>
    <head>

    </head>
    <body>

    </body>
</html>
```

As you can see, these tags are nested within the `<html>` tag because they are HTML, but they are stacked on top of each other because they are separate sections of the website. This is the pattern HTML generally works in: tags within tags that contain certain types of content. Here are example of some basic common HTML tags:

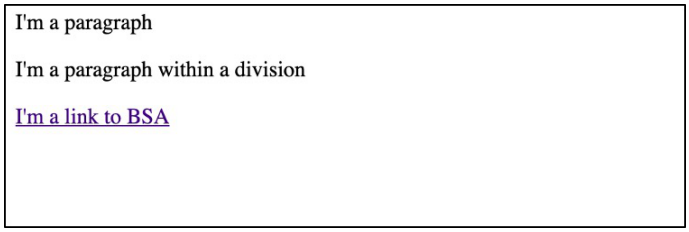
- `<div>` This tag simply indicates a division of the page
- `<table>` Predictably, this tag contains a table
- `<p>` This tag contains a "paragraph", which is just some amount of plain text
- `<a>` This tag contains a link to either another part of the site or a different site entirely.

Here is the HTML for a very simple website that displays some of these concepts:

```
<html>
    <head>
    </head>
    <body>
        <p>I'm a paragraph</p>
        <div>
            <p>I'm a paragraph within a division</p>
            <a href= "https://bruinsportsanalytics.com">I'm a link to BSA</a>
        </div>
```

```
    </body>
</html>
```

And this is what the website looks like for the code above:



7.2 BeautifulSoup Basic Tutorial

BeautifulSoup essentially allows us to go through a website's HTML and collect whatever we want from any of its tags. What we need to know now is which tags of a website contain the information we want.

For the purpose of this tutorial, we will scrape the "Team Per Game Stats" table for the 2019-20 NBA season. This table is located at: <www.basketball-reference.com/leagues/NBA_2020.html>.

We will scrape every NBA team's name and 3 point percentage during the 2019-20 regular season and store it in a .csv file. We'll slowly build the python script that we will use for this.

7.2.1 Installations

First and foremost, we'll start with the installations we will need: BeautifulSoup4, pandas, and chromedriver (we will use Google Chrome).

1. Chromedriver can be downloaded here.
2. BeautifulSoup4 and pandas can be installed through the command line with the python package manager `pip`:

```
$ pip install beautifulsoup4
$ pip install pandas
```

7.2.2 Python Script

Now that we have what we need, we can start scraping our data. Firstly, we'll have to set up the driver and collect the entire HTML content of the website:

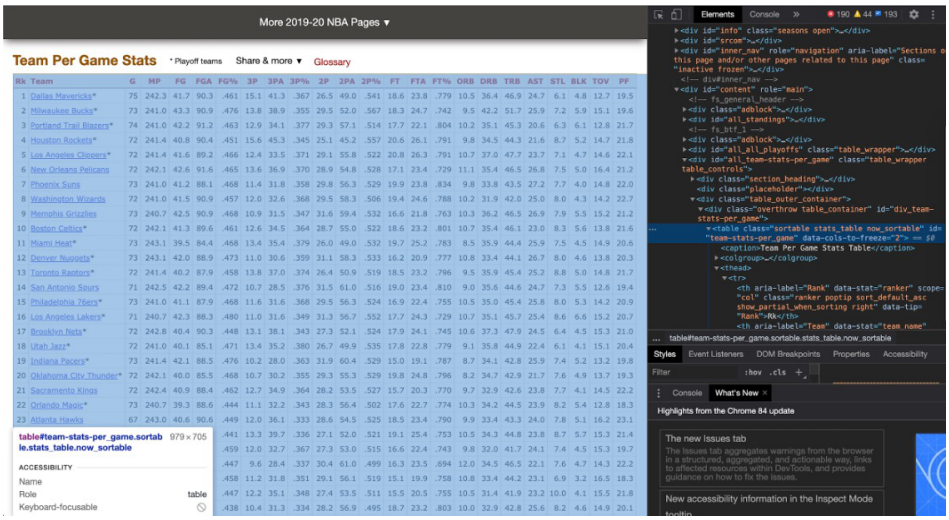
```
from selenium import webdriver
from bs4 import BeautifulSoup
import pandas as pd

# Set up driver
driver = webdriver.Chrome("/Users/joebruin/Documents/chromedriver")

# Collect website html
driver.get("https://www.basketball-reference.com/leagues/NBA_2020.html")
content = driver.page_source
soup = BeautifulSoup(content)
```

Now, the variable `soup` contains the entire HTML of the site. You can verify this by printing this variable to the console if you want to make sure. Next, we will have to go to the website and find out which tags contain the information we want: Team name and team 3 point percentage. This is what the table looks like on the site:

Team Per Game Stats * Playoff teams Share & more ▾ Glossary

Rk	Team	G	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
1	Dallas Mavericks*	75	242.3	41.7	90.3	.461	15.1	41.3	.367	26.5	49.0	.541	18.6	23.8	.779	10.5	36.4	46.9	24.7	6.1	4.8	12.7	19.5	117.0
2	Milwaukee Bucks*	73	241.0	43.3	90.9	.476	13.8	38.9	.355	29.5	52.0	.567	18.3	24.7	.742	9.5	42.2	51.7	25.9	7.2	5.9	15.1	19.6	118.7
3	Portland Trail Blazers*	74	241.0	42.2	91.2	.463	12.9	34.1	.377	29.3	57.1	.514	17.7	22.1	.804	10.2	35.1	45.3	20.6	6.3	6.1	12.8	21.7	115.0
4	Houston Rockets*	72	241.4	40.8	90.4	.451	15.6	45.3	.345	25.1	45.2	.557	20.6	26.1	.791	9.8	34.5	44.3	21.6	8.7	5.2	14.7	21.8	117.8
5	Los Angeles Clippers*	72	241.4	41.6	89.2	.466	12.4	33.5	.371	29.1	55.8	.522	20.8	26.3	.791	10.7	37.0	47.7	23.7	7.1	4.7	14.6	22.1	116.3
6	New Orleans Pelicans	72	242.1	42.6	91.6	.465	13.6	36.9	.370	28.9	54.8	.528	17.1	23.4	.729	11.1	35.4	46.5	26.8	7.5	5.0	16.4	21.2	115.8
7	Phoenix Suns	73	241.0	41.2	88.1	.468	11.4	31.8	.358	29.8	56.3	.529	19.9	23.8	.834	9.8	33.8	43.5	27.2	7.7	4.0	14.8	22.0	113.6
8	Washington Wizards	72	241.0	41.5	90.9	.457	12.0	32.6	.368	29.5	58.3	.506	19.4	24.6	.788	10.2	31.9	42.0	25.0	8.0	4.3	14.2	22.7	114.4
9	Memphis Grizzlies	73	240.7	42.5	90.9	.468	10.9	31.5	.347	31.6	59.4	.532	16.6	21.8	.763	10.3	36.2	46.5	26.9	7.9	5.5	15.2	21.2	112.6
10	Boston Celtics*	72	242.1	41.3	89.6	.461	12.6	34.5	.364	28.7	55.0	.522	18.6	23.2	.801	10.7	35.4	46.1	23.0	8.3	5.6	13.8	21.6	113.7
11	Miami Heat*	73	243.1	39.5	84.4	.468	13.4	35.4	.379	26.0	49.0	.532	19.7	25.2	.783	8.5	35.9	44.4	25.9	7.5	4.5	14.9	20.6	112.0
12	Denver Nuggets*	73	243.1	42.0	88.9	.473	11.0	30.6	.359	31.1	58.3	.533	16.2	20.9	.777	10.8	33.4	44.1	26.7	8.0	4.6	13.8	20.3	113.3
13	Toronto Raptors*	72	241.4	40.2	87.9	.458	13.8	37.0	.374	26.4	50.9	.519	18.5	23.2	.796	9.5	35.9	45.4	25.2	8.8	5.0	14.8	21.7	112.8
14	San Antonio Spurs	71	242.5	42.2	89.4	.472	10.7	28.5	.376	31.5	61.0	.516	19.0	23.4	.810	9.0	35.6	44.6	24.7	7.3	5.5	12.6	19.4	114.1
15	Philadelphia 76ers*	73	241.0	41.1	87.9	.468	11.6	31.6	.368	29.5	56.3	.524	16.9	22.4	.755	10.5	35.0	45.4	25.8	8.0	5.3	14.2	20.9	110.7
16	Los Angeles Lakers*	71	240.7	42.3	88.3	.480	11.0	31.6	.349	31.3	56.7	.552	17.7	24.3	.729	10.7	35.1	45.7	25.4	8.6	6.6	15.2	20.7	113.4
17	Brooklyn Nets*	72	242.8	40.4	90.3	.448	13.1	38.1	.343	27.3	52.1	.524	17.9	24.1	.745	10.6	37.3	47.9	24.5	6.4	4.5	15.3	21.0	111.8
18	Utah Jazz*	72	241.0	40.1	85.1	.471	13.4	35.2	.380	26.7	49.9	.535	17.8	22.8	.779	9.1	35.8	44.9	22.4	6.1	4.1	15.1	20.4	111.3
19	Indiana Pacers*	73	241.4	42.1	88.5	.476	10.2	28.0	.363	31.9	60.4	.529	15.0	19.1	.787	8.7	34.1	42.8	25.9	7.4	5.2	13.2	19.8	109.4
20	Oklahoma City Thunder*	72	242.1	40.0	85.5	.468	10.7	30.2	.355	29.3	55.3	.529	19.8	24.8	.796	8.2	34.7	42.9	21.7	7.6	4.9	13.7	19.3	110.4
21	Sacramento Kings	72	242.4	40.9	88.4	.462	12.7	34.9	.364	28.2	53.5	.527	15.7	20.3	.770	9.7	32.9	42.6	23.8	7.7	4.1	14.5	22.2	110.1
22	Orlando Magic*	73	240.7	39.3	88.6	.444	11.1	32.2	.343	28.3	56.4	.502	17.6	22.7	.774	10.3	34.2	44.5	23.9	8.2	5.4	12.8	18.3	107.3
23	Atlanta Hawks	67	243.0	40.6	90.6	.449	12.0	36.1	.333	28.6	54.5	.525	18.5	23.4	.790	9.9	33.4	43.3	24.0	7.8	5.1	16.2	23.1	111.8
24	Minnesota Timberwolves	64	243.1	40.4	91.6	.441	13.3	39.7	.336	27.1	52.0	.521	19.1	25.4	.753	10.5	34.3	44.8	23.8	8.7	5.7	15.3	21.4	113.3
25	Detroit Pistons	66	241.9	39.3	85.7	.459	12.0	32.7	.367	27.3	53.0	.515	16.6	22.4	.743	9.8	32.0	41.7	24.1	7.4	4.5	15.3	19.7	107.2
26	New York Knicks	66	241.9	40.0	89.3	.447	9.6	28.4	.337	30.4	61.0	.499	16.3	23.5	.694	12.0	34.5	46.5	22.1	7.6	4.7	14.3	22.2	105.8
27	Cleveland Cavaliers	65	241.9	40.3	87.9	.458	11.2	31.8	.351	29.1	56.1	.519	15.1	19.9	.758	10.8	33.4	44.2	23.1	6.9	3.2	16.5	18.3	106.9
28	Chicago Bulls	65	241.2	39.6	88.6	.447	12.2	35.1	.348	27.4	53.5	.511	15.5	20.5	.755	10.5	31.4	41.9	23.2	10.0	4.1	15.5	21.8	106.8
29	Golden State Warriors	65	241.9	38.6	88.2	.438	10.4	31.3	.334	28.2	56.9	.495	18.7	23.2	.803	10.0	32.9	42.8	25.6	8.2	4.6	14.9	20.1	106.3
30	Charlotte Hornets	65	242.3	37.3	85.9	.434	12.1	34.3	.352	25.2	51.6	.489	16.2	21.6	.748	11.0	31.8	42.8	23.8	6.6	4.1	14.6	18.8	102.9

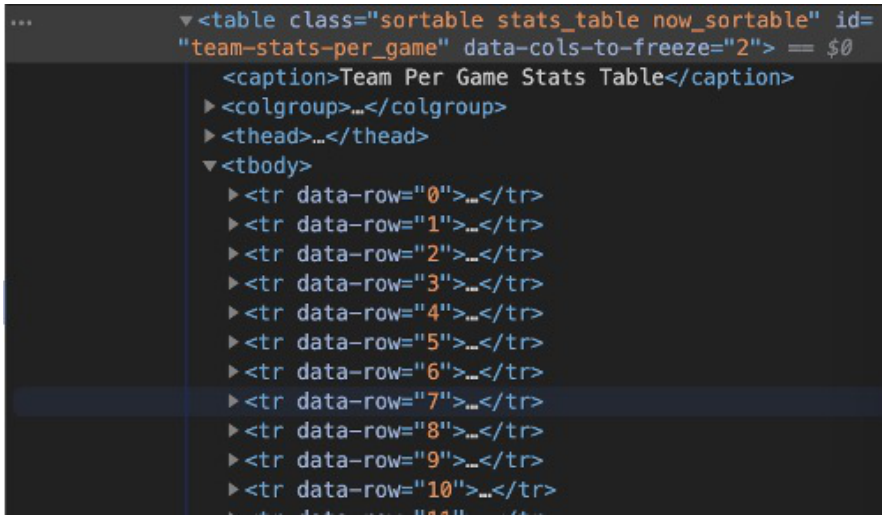
To find out what the HTML for this table looks like, we can use the "inspect element" feature in our browser (simply right-click the table and select "inspect"):



As can be seen, the table is stored in the highlighted `<table>` tag shown in the HTML on the right. Notice how the `<table>` tag contains a few additional attributes within it: *class*, *id*, and *data-cols-to-freeze*. We can use the *id* attribute to specify the exact table we want because that attribute appears to contain a unique title for this table: "team-stats-per_game". With just one more line of code, we can now continue and collect the table we want:

```
# Find and store desired table
table = soup.find('table', attrs = {'id': 'team-stats-per_game'})
```

The first argument of the `find()` function is the type of tag we are looking for. The second is a set of attributes to narrow the search; we simply use the *id* that we saw above and we are good to go. Now that we have the table, we can go back to the website and see what the inside of this table looks like in HTML:



The table contains a few tags within it, but `<tbody>` is the one we want. Within `<tbody>` we can see that there are a bunch of `<tr>` tags; each `<tr>` tag represents one row of the table. We can further expand one of these `<tr>` tags to see what is inside each row:

Each `<tr>` tag contains a series of `<td>` tags; a `<td>` tag defines a data cell in HTML, so each of these tags contains a piece of data (as you can already see with the white values). Each `<td>` tag contains a *data-stat* attribute that specifies what data it is holding; we want the "team_name" data and the "fg3_pct" data. Going back to our main goal, we can use the following logic to get what we need from this table:

For each `<tr>` tag in the table, grab the team name and the team 3 point percentage by searching for the `<td>` tags with "team_name" and "fg3_pct" in their *data-stat* fields.

This is what this looks like in our code:

```
# Find and store desired table
table = soup.find('table', attrs = {'id': 'team-stats-per_game'})

# Create lists to store our data
team_names = []
team_3pt_pcts = []
```

```
body = table.find('tbody')
all_rows = body.findAll('tr')
for row in all_rows:
    # Collect team name
    team_name_cell = row.find('td', attrs = {'data-stat': 'team_name'})
    team_name = team_name_cell.text
    team_names.append(team_name)

    # Collect 3 pt pct
    three_pt_cell = row.find('td', attrs = {'data-stat': 'fg3_pct'})
    three_pt_pct = float(three_pt_cell.text)
    team_3pt_pcts.append(three_pt_pct)

# Print our results
print(team_names)
print(team_3pt_pcts)
```

Let's go through the code above. First, we create two empty lists to store our team names and our three point percentages as we scrape the table. Then, we get the table body (the `<tbody>` tag we discussed above) from our table object. Then, we get a list of all of the rows in the table by using the `findAll()` function to collect every `<tr>` tag within our table body. Once we have this list, we loop through each `<tr>` tag (remember each `<tr>` represents a row) and collect our data. To get the specific cells that we want, we use the *data-stat* attribute as we discussed earlier to look for 'team_name' and 'fg3_pct'. But the cell is not the actual data. We have to use the `.text` attribute of the cell to get the actual string of data we want (we convert the string to a decimal number for the 3 point percentages as you can see using the `float()` function). Once we get the values we want, we append them to our final lists. After the loop, we can print the lists to verify our results.

All that's left now is to store these results in a .csv, which is incredibly simple:

```
data_frame = pd.DataFrame({'Team Name':team_names, '3 Point %':team_3pt_pcts})
data_frame.to_csv('data.csv', index=False, encoding='utf-8')
```

We should now see a .csv file pop up in the same folder as our script that contains the data we scraped. This is just a simple example of how to webscrape, but it should put you in a position where you can experiment more with Python and BeautifulSoup to scrape far more data from far more websites.