

[Sign in](#)[Get started](#)[Follow](#)

542K Followers

·

[Editors' Picks](#)[Features](#)[Explore](#)[Contribute](#)[About](#)

You have **1** free member-only story left this month. [Sign up for Medium and get an extra one](#)

Web Scraping Yahoo Finance

Pull financial statements and stock data from any publicly traded company



Randy Macaraeg Feb 1, 2020 · 5 min read ★





Photo by [Markus Spiske](#) on [Unsplash](#)

The code to this blog can be found on my [GitHub](#).

In the business world, it's important to know the financial health of a company. Looking at the financial statements is a great way to get some insight into how well a company is doing.

In this blog, I'll go over pulling the financial statements from Yahoo Finance for any company in their database in Python. Because Yahoo Finance uses JavaScript, we utilize a combination of BeautifulSoup and Selenium

Import the Libraries

Let's start with the necessary libraries:

```
import pandas as pd
from bs4 import BeautifulSoup
import re
from selenium import webdriver
import chromedriver_binary
import string

pd.options.display.float_format = '{:.0f}'.format
```

Set up and run the driver

```
is_link = 'https://finance.yahoo.com/quote/AAPL/financials?p=AAPL''

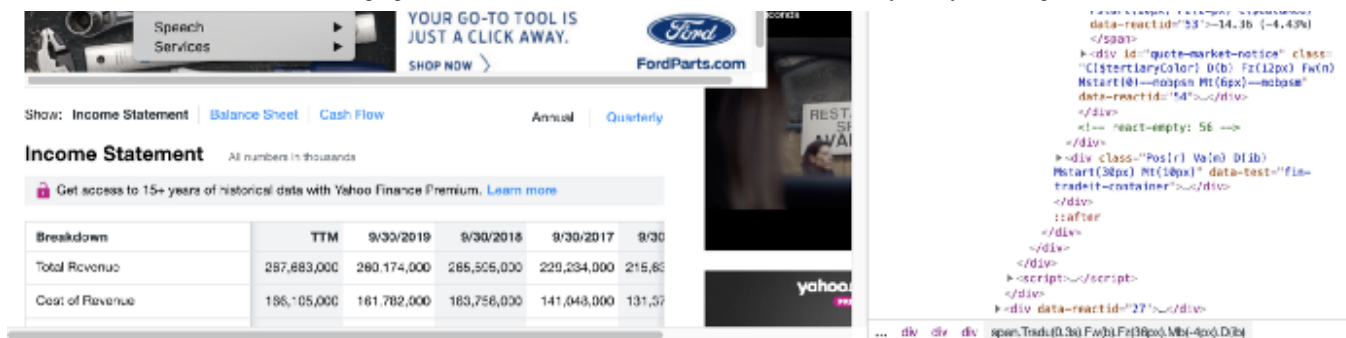
driver = webdriver.Chrome()
driver.get(is_link)
html = driver.execute_script('return document.body.innerHTML;')
soup = BeautifulSoup(html, 'lxml')
```

I prefer using Chrome as my web browser, but feel free to use whatever you're most comfortable with (Firefox, Safari, etc.). I'm also using Apple as my example company, but you can change the AAPL ticker in the link to another company's stock ticker to change the data.

The above code will open the page in a dummy browser and pull all of the data within the body of the website. Since Yahoo Finance operates on JavaScript, running the code through this method pulls all of the data and saves it as if it were a static website. This is important for pulling the stock price, as those are dynamic items on the webpage and can refresh/update at regular intervals.

To pull the stock data, we start by getting the location of the stock prices. By hovering over the stock price we can use the Inspect tool to find the exact syntax for the stock price.

The screenshot shows the Yahoo! Finance website with the Apple Inc. (AAPL) stock page. The page displays the current stock price of \$309.51, a 1-day price change of +1.26%, and a 52-week high of \$329.34. The page includes a search bar, navigation tabs for Finance Home, Watchlists, My Portfolio, Screeners, Premium, Markets, and Industries. The main content area shows the Apple Inc. (AAPL) stock page with a price of \$309.51, a 1-day price change of +1.26%, and a 52-week high of \$329.34. The page also features a 'Copy' button, a 'Search Google for "309.51"' button, and a 'Print...' button. The right sidebar shows 'Conversations', 'Statistics', 'Historical Data', 'Profile', 'Financials', and 'Analysis'.



The screenshot shows the Yahoo Finance website for Ford. The 'Income Statement' tab is selected, displaying a table with financial data for various periods. To the right, a snippet of BeautifulSoup HTML code is shown, demonstrating how to parse the page's DOM to find specific elements like the stock price.

Income Statement All numbers in thousands

Breakdown	TTM	9/30/2019	9/30/2018	9/30/2017	9/30/2016
Total Revenue	257,883,000	280,174,000	285,305,000	229,234,000	215,650,000
Cost of Revenue	195,105,000	161,782,000	163,756,000	141,043,000	131,370,000

```

<div data-reactid="33">14.36 (-4.43%)
</div>
<div id="quote-market-notice" class=
"CltertiaryColor" D(b) Fz(12px) Fw(n)
Mstart(0)---span Mt(6px)---mbase"
data-reactid="54">
</div>
<!-- react-empty: 56 -->
</div>
<div class="Pos(r) Val(n) D(ib)
Mstart(36px) Mt(16px)" data-test="fin=
tradeit-container">
::after
</div>
</div>
</div>
<script>
</div>
</div>
<div data-reactid="27">
... div div div span.Tradu(0.3s) Fw(b) Fz(16px) Mb(-4px) D(ib)

```

Transferring this information into Python, the code would look like this:

```
close_price = [entry.text for entry in soup.find_all('span',
{'class': 'Trsdu(0.3s) Fw(b) Fz(36px) Mb(-4px) D(ib)'})]
```

This code searches for the 'span' tag within all of the HTML code and looks for the class attribute that matches the one entered. Luckily this pulls only one number, which is the stock price at the close.

Now that we have this we can move on to the financial statements.

Pulling financial statement data

Continuing on with the scraping, we search the page to find all of the div containers, and dive in a bit further to find the features we want to work with. I found that each row of the financial data is stored within a div container with a common class attribute of 'D(tbr)'. In the example below there are additional pieces of data in the class attribute, but as long as the first portion matches what we're searching it will pull that data.

The screenshot displays the Yahoo Finance Income Statement for Alphabet Inc. (GOOGL). The table shows financial data from 2019 to 2021. The HTML source code snippet on the right highlights the 'D(tbr)' class attribute used for each row of the financial data table.

	9/30/2019	9/30/2018	9/30/2017	9/30/2016
Total Revenue	267,683,000	260,174,000	265,535,000	229,234,000
Cost of Revenue	168,105,000	161,782,000	163,756,000	141,048,000
Gross Profit	101,578,000	98,392,000	101,839,000	88,186,000
Operating Expenses				
Research Development	16,756,000	16,217,000	14,236,000	11,581,000
Selling General and Administr...	18,659,000	18,245,000	16,735,000	15,261,000
Total Operating Expenses	35,425,000	34,462,000	30,941,000	26,842,000
Operating Income or Loss	66,153,000	63,930,000	70,898,000	61,344,000
Interest Expense	-	3,576,000	3,240,000	2,323,000
Total Other Income/Expenses Net	628,000	1,807,000	-441,000	-133,000
Income Before Tax	67,749,000	68,737,000	72,933,000	64,089,000
Income Tax Expense	10,222,000	10,461,000	13,372,000	15,736,000

Each line on the financial statements is stored in a div

```
features = soup.find_all('div', class_='D(tbr)')
```

This will pull a lot of messy looking data which confused me for a while. After digging into the data pulled, I used a find function to see where each line of data was that I wanted. After a lot of trial and error, I was able to produce the code to pull only the financial data:

```
headers = []
temp_list = []
label_list = []
final = []
index = 0

#create headers
for item in features[0].find_all('div', class_='D(ib)'):
    headers.append(item.text)

#statement contents
while index <= len(features)-1:
    #filter for each line of the statement
    temp = features[index].find_all('div', class_='D(tbc)')
    for line in temp:
        #each item adding to a temporary list
        temp_list.append(line.text)
    #temp_list added to final list
    final.append(temp_list)
    #clear temp_list
    temp_list = []
    index+=1

df = pd.DataFrame(final[1:])
df.columns = headers
```

The headers were separated from the rest of the data since it was causing some issues when putting all of the data together. After creating a list of the headers and multiple lists for the data within the financial statements, it combines everything together to produce a copy!

	Breakdown	ttm	9/30/2019	9/30/2018	9/30/2017	9/30/2016
0	Total Revenue	267,683,000	260,174,000	265,595,000	229,234,000	215,639,000
1	Cost of Revenue	166,105,000	161,782,000	163,756,000	141,048,000	131,376,000
2	Gross Profit	101,578,000	98,392,000	101,839,000	88,186,000	84,263,000
3	Operating Expenses					
4	Research Development	16,766,000	16,217,000	14,236,000	11,581,000	10,045,000
5	Selling General and Administrative	18,659,000	18,245,000	16,705,000	15,261,000	14,194,000
6	Total Operating Expenses	35,425,000	34,462,000	30,941,000	26,842,000	24,239,000
7	Operating Income or Loss	66,153,000	63,930,000	70,898,000	61,344,000	60,024,000
8	Interest Expense	-	3,576,000	3,240,000	2,323,000	1,456,000
9	Total Other Income/Expenses Net	628,000	1,807,000	-441,000	-133,000	-1,195,000
10	Income Before Tax	67,749,000	65,737,000	72,903,000	64,089,000	61,372,000
11	Income Tax Expense	10,222,000	10,481,000	13,372,000	15,738,000	15,685,000
12	Income from Continuing Operations	57,527,000	55,256,000	59,531,000	48,351,000	45,687,000
13	Net Income	57,527,000	55,256,000	59,531,000	48,351,000	45,687,000
14	Net Income available to common shareholders	57,527,000	55,256,000	59,531,000	48,351,000	45,687,000
15	Reported EPS					
16	Basic	-	11.97	12.01	9.27	8.35
17	Diluted	-	11.89	11.91	9.21	8.31
18	Weighted average shares outstanding					
19	Basic	-	4,617,834	4,955,377	5,217,242	5,470,820
20	Diluted	-	4,648,913	5,000,109	5,251,692	5,500,281

21	EBITDA	-	76,477,000	87,046,000	76,569,000	73,333,000
----	--------	---	------------	------------	------------	------------

It looks just like Apple's income statement (minus formatting), but while it looks complete, the next issue is that all of the numbers on the page are saved as strings. If we want to do any future calculations on it, we'd have to change them all to integers:

```
#function to make all values numerical
def convert_to_numeric(column):

    first_col = [i.replace(',', '') for i in column]
    second_col = [i.replace('-', '') for i in first_col]
    final_col = pd.to_numeric(second_col)

    return final_col
```

With a quick for loop, we can turn all of the number strings into integers:

```
for column in headers[1:]:

    df[column] = convert_to_numeric(df[column])

final_df = df.fillna('-')
```

This converts all numbered strings into actual numbers, and for all NaNs it is replaced by a dash.

The final product looks pretty good and can now be used for calculations!

	Breakdown	ttm	9/30/2019	9/30/2018	9/30/2017	9/30/2016
0	Total Revenue	267683000	260174000	265595000	229234000	215639000
1	Cost of Revenue	166105000	161782000	163756000	141048000	131376000
2	Gross Profit	101578000	98392000	101839000	88186000	84263000
3	Operating Expenses	-	-	-	-	-
4	Research Development	16766000	16217000	14236000	11581000	10045000
5	Selling General and Administrative	18659000	18245000	16705000	15261000	14194000
6	Total Operating Expenses	35425000	34462000	30941000	26842000	24239000
7	Operating Income or Loss	66153000	63930000	70898000	61344000	60024000
8	Interest Expense	-	3576000	3240000	2323000	1456000
9	Total Other Income/Expenses Net	628000	1807000	441000	133000	1195000
10	Income Before Tax	67749000	65737000	72903000	64089000	61372000
11	Income Tax Expense	10222000	10481000	13372000	15738000	15685000
12	Income from Continuing Operations	57527000	55256000	59531000	48351000	45687000
13	Net Income	57527000	55256000	59531000	48351000	45687000
14	Net Income available to common shareholders	57527000	55256000	59531000	48351000	45687000
15	Reported EPS	-	-	-	-	-
16	Basic	-	12	12	9	8
17	Diluted	-	12	12	9	8
18	Weighted average shares outstanding	-	-	-	-	-

19	BASIC	-	4617834	4955377	5217242	5470820
20	Diluted	-	4648913	5000109	5251692	5500281
21	EBITDA	-	76477000	87046000	76569000	73333000

Finalized Income Statement

While this pulls only the income statement from Apple, if you use Yahoo Finance's link on the balance sheet or statement of cash flows, the code should pull everything fine and put the statement together just like the above.

I'm sure there's a lot of additional work that can be done to clean this up and I'd love to make it look more like an official statement in the near future. While continuing to work on this, I'd like to go over some deep dives into the data to find basic financial ratios and go over things like company valuation, investment analysis, and other trends/findings I can dig out from this data. Stay tuned!

I hope this code helps, and if you have any feedback or suggestions I'd love to hear it!

Sign up for The Daily Pick

By Towards Data Science

Hands-on real-world examples, research, tutorials, and cutting-edge techniques delivered Monday to Thursday. Make learning your daily ritual. [Take a look](#)

Your email

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

Finance

Python

Web Scraping

Accounting

Stock Market

About

Help

Legal