Scraping financial data from Yahoo! Finance

Hello, in this video, I’m going to show you how to scrape financial data from Yahoo! Finance. I’m going to demonstrate two of my favorite methods for scraping data… using hidden API’s and xtracting embedded json formatted data which can be easily converted to a Python dictionary. Let’s get started.

# Getting Started

For this project, we’re going to use several standard python libraries, but we’ll also use `BeautifulSoup` and `Requests`. So, if you do not have these already, you’ll need to install **BeautifulSoup** with `pip install bs4` and **Requests** with `pip install requests`.

The other libraries that we’ll use include:

* Re
* Json
* Csv
* and from io import StringIO

I’m going to be demonstrating this exercise in a Jupyter Notebook, but feel free to use whatever ID or code editor that you wish.

First, navigate to Yahoo! Finance, and enter the stock ticker of a company you want to scrape.

You’ll notice lots of interesting data, and multiple tabs to choose from. Today, we are going to focus on a few tabs as an example, but you can replicate this process for any of them that you wish.

## URL Template

Next, copy the urls of the tabs that you want to scrape… I’m going to choose:

* Stats
* Profile
* Financials
* and analysis

I’m also going to extract the historical stock data… but I’m going to access that a different way. So we’ll leave that one for now.

You’ll notice that the urls all have a very similar format. All we need to do to make this a template is replace the ticker symbol with a curly brace. This will allow us to insert whatever ticker symbol we want.

Now we can assign the letter ‘F’ to the stock variable, since I’m going to be looking up the data for Ford Motor Company and “F” is their ticker symbol.

The first thing I’d like to do is scrape the financial statements. If you look at the web page, you’ll notice that there are a few statements available… Income, Balance Sheet, and Cashflow. Additionally, you have the option of getting Quarterly or Annual statements. All of the data on the page will be available to use with our get request.

# Requesting the data

Next, let’s use the financials url template and our stock variable to request the data using the requests library. We’ll assign the response to a response variable.

Next, we’ll parse the html data using BeatifulSoup, using the default html parser.

# Reviewing the raw data

To show you what we need to do, let’s take a look at the raw data. You can either save the response text to a text file and review it there, or you can simply right click on the web page and click “view page source”.

There is a lot of messy stuff on this page… and very little actual HTML to scrape. However at some point when you scroll down this mess of code you’ll notice what appears to be json formatted strings. What is happening is that the web page is dynamically loading this data from this java script functions contains with the script tags. Fortunately for use, there is a function that is appropriate named “—data – “. This will contain all of the data that is dynamically loaded onto the page. Now, this isn’t quite a clean as I’d like… often the data will be embedded in a script that has a type of “application/json” or “application/ld+json”. However, this is actually embedded in the function itself. So, what we have to do is get this script tag, extract the contents from it, and then chop off the edges so that what we’re left with is just the json formatted string that we can convert into a Python dictionary. Don’t’ worry, it’s not as hard as it sounds. Back to the code.

# Parsing the json data dictionary

Now, a lot of the scripts are not tagged with any kind of unique identifier such as a class, type, or id. So, what we’ll need to rely on for this project is keyword text that we can find in the tag itself. Fortunately, Beautiful Soup provides a way for us to filter html tags by a text pattern with regular expressions. If you don’t know much about regular expressions… I would definitely recommend you learn more about them.. they are an invaluable tool to have in your web-scraping toolkit. The pattern we are going to use for this project is not complicated at all. So if you’ve never used them before… please don’t worry about it.

I’m going to use the comments “—Data—” that we found in the code to identify the script we want. So, to create a regular expression pattern we use the compile method. This pattern will be a space, followed by two dashes, a space, “data”, and then a space, two dashes, and a space.

Now, we can use this pattern to find a script element that has text that matches this pattern. We’re also going to return the contents of the element. This returns a list. So you’ll need to grab the first and only item in the list.

If you print the script data , you’ll see what we’ve successful identified the correct script. The problem now is that we have this javascript code wrapping the data we actually want. This is easy enough to fix. We are going to slice the data that we want from the scriptdata text. The find the boundaries of the slice, we are going to use the word “context” to mark the front end, and if we look at the back end, we can see that the string starts about 12 characters from the end. So that’s easy enough to do.

The starting position will be 2 characters behind the beginning of the word context, and then we’ll go 12 characters from the end, which is indicated with a negative 12. We can then pass this text into the `json.loads` function and what we should get is a python dictionary.

# Financial Statements

Since this is a dictionary, it will have keys and values. To find out what keys are available, you can use the keys method. You can explore this in more detail if you want, but the data for the financial statements is located in “context-dispatcher-stores-quotesummarystore. And, if you use the keys method, you’ll be able to see what’s available here. The great thing about this data set is that it contains both the annual and quarterly statements. I’ll copy and paste the addresses to them all here. What we’re going to do is explore the income statement as a pattern, and then I’ll show you that you can apply that same pattern to every other financial statement.

The “annual\_is” variable contains a list of financial statements that I got from the json\_data dictionary. To prototype our exercise here, we’re going to grab the first income statement in the list. And then, I’ll print it so you can see what’s inside.

This statement is itself a list of dictionaries, mostly, that contain the account name, and a variety of number formats. Consolidating this data I fairly straight-forward. For example, I I want to grab the account and the raw number I can use a for loop. Some of these accounts do not contain data or do not contain a dictionary. In this case, I’ll need to handle the resulting type error or key error.

And you can see what with a simple loop, I’m able to extract only the data I want into a list of accounts that I can then export to a csv file.

Now, this model can be easily applied to any of the other financial statements by simply changing the variable names and references. I’ll grab the quarterly and annual cash flow statements as an example.

## Profile Data

Alright. Now that you’ve seen the financial statements, let’s look at how to grab the profile data… such as the company officers, their compensation, etc…

Believe it or not, we can simply copy and paste most of what we’ve already done and just change the url to the profile url template.

Similar to the previous example, most of the interesting data is located in the “QuoteSummaryStore”.

Within that, most of the information you’ll want is in the “assetProfile” key. But, the “summaryDetail” and “secFilings” are also worth exploring and I’ll show you briefly what is in there.

The company officers are located in the “companyOfficers” key within assetProfile, and you can see that all the information you would expect is there, including pay, name, title, age, etc...

The the company description is located within the “longBusinessSummary” key

Next, you can also get the links to the SEC filings from edgars. This is with the “QuoteSummaryStore” under “SEC Filings”. You’ll notice there is a url here. This will take you to the edgars website where you can explore this filing further.

There are a lot of summary details that you can get from the “QuoteSummaryStore’ under the key “SummaryDetail”. I’ll print those here, and I’ll let you explore those on your own.

# Statistics

Let’s look at the statistics data. I’m literally going to copy and paste this block that I created for the profile page, and then change the url. That’s all.

Most of the interesting data here Is located in the Quote summary store under the key “defaultKeyStatistics”. I’ll let you explore these on your own, but you can see how easy it is to get to the data.

# Historical Stock Data

Last but not least, let’s extract the historical stock data. If you look at the historical data tab, you’ll see there’s a download button here. Go ahead an right-click on that and then paste it into your code.

Now, this is an api call. You can tell by not so obvious query prefix, the api version, and the parameters used in the query. What this api does is return the requests data in csv format.

So, if I get this stock url, what I’ll get back is csv formatted data, which I can save to a csv file. Simple.

What I want to do though, is make this api call more customizable. So, the first thing I can do is split the query parameters into a python dictionary. Unfortunately, the V7 version of this API doesn’t have a “stock symbol” parameter that I’ve been able to find. The V8 version does, but it returns json data, and honestly, getting this csv file is much easier than parsing json data.

However, we can customize these parameters event further by removing these period parameters.. which are timestamps not recognizable by humans, I think. And then, we can replace them with a range parameters. And the value I’m going to set is 5y, which standard for 5 years. So, what I’m requesting is 5 years of daily stock data.

Then, instead of saving to a file, I’m going to go ahead and use a StringIO object so that I can print this on the screen.

So, how did I know that I could use the range parameters instead of the periods? They key is to take a look at the network activity on the web page and investigate this hidden api. Go back to the website, and click on the summary page. Open up the developer tools, or right click then click "Inspect". At the top you'll see a section call "Network Activity". If there’s any api calls being made, they will show up here... specifically under the XHR filter. According to Google, XHR is a JavaScript object that is used to transfer data between your webrowser, and the webserver. I’m not a web developer so I suppose that you will either accept that explanation or correct me in the comments. If you click on some of these items, you'll notice that a new screen reveals the headers, and also the response. What I typically do is look for a response that appears to be a json formatted string, such as these. Then, I go to the headers to see what calls I need to make. However, since we already know what the api call looks like, we can simply look for it in this list by checking on the request url. You can see that several api calls are being made to the V7 api for different elements on the page. If you look at the parameters, you'll notice that they are slightly different than the one we originally used.

Because this is a hidden API, so they only way you're really going to know how to interact with it is by exploring and experimenting. Now, we know that it accepts a range and interval. So, one clue to finding out what values we can use is to simply look at the filters on the page. Based on what I found, the chart tab seemed to have the best information on what parameters I could use. Then it's a matter of plugging these in and seeing if it works.

Another way of find out how to use this specific api is simply to make a mistake.

Fortunately, this api gives you feedback.

And that my friends, is how you scrape financial data from Yahoo! Finance. If you enjoyed this video, you may enjoy some of the other web scraping videos in this series. Please hit that like button and subscribe for more content like this in the future. See you in the next video.