Web scraping and Covid-19: Example 2

Welcome to the second part of the third lesson in our web scraping to build social research data training series. In the first part, we looked at how we could use Python to extract text from a web page. In this quick second part, we're going to look at how we can repeat that process. But this time, instead of downloading a file from a webpage, we're gonna use a quick example from my area of research, which is charitable organisations.

So to research these organisations, I tend to use administrative data, that data is provided by a regulator, and it's updated daily. So it's a real time continuously updated data set. So I'm sure you can see the advantages of having an automated scheduled means of collecting that data and from my research. So what I'm interested in is downloading the register of charities. So this is basically a census of charitable organisations that operate in England and Wales. So we'll follow the same six step process, basically, for web scraping any web page, and but this time, the slight differences, instead of, you know, extracting text or tables or lists, we're going to identify a file on a web page, request that and then download it and save it somewhere on our machine locally.

So here's the webpage that I'm interested in. So as I said, there's a charity regulator in England and Wales. And it provides Yeah, daily extracts of multiple different datasets, got datasets relating to financial histories, trustees, governing documents, particular events that happen charities, etc. We're just simply interested in this example, in the kind of complete list of registered charities in England and Wales. So this is the link to the web page, which then itself contains links to the files I need. So in essence, what I want is basically these links, here, you can see as I hover, and down in the bottom corner, you can see a quick preview of what the link is. So this is then what I'm requesting. So instead of requesting this web page appear by manually inspecting the web page, but I actually want to use Python to request or these links here are one of them in this example.

So again, you can see that the manual process of inspecting web pages, it's I think, Brooker called it the detective work of web scraping. And it is almost the most critical part of the process. In fact, the writing of coders, hopefully, you can see, it's actually relatively straightforward. It's, you know, well documented, there's plenty of examples. And here's one, you know, for you to follow yourself. The real work is digging into the structure of the web page, you know, visually and figuring out where the text of the file is, and how that can be identified using HTML. So this is a very similar process to the first part of the lesson. So I won't, I won't spend too much time. So there's the preliminary preliminary step of importing modules that we need. And this is very simple, we don't need Beautiful Soup this time, actually, we just need the requests module. So that's worked. So the URL that I define is, we can find it here. And it will be this one here. Yeah. You know, so if I right click, and I say copy link address, and I put that into my browser, you can see this is the link here to the data file.

So again, I'm using a browser, I can manually just, you know, press enter, and then that'll start downloading the zip file to my machine. But again, I want to use Python to do that. If this was just a

once every three or four months, then I probably wouldn't write a programming script. But actually, this is something I do at least on a monthly basis. And not just for this file, but for all of these files. And that's also part of a wider project where for other countries, I'm also scraping, you know, charity data. So there's a good case to be made, and for using web scraping as a data collection method. So let's just do essentially what I've done manually but using Python. So I'm going to request the web address of that file. It's a big enough file. So we'll take a few seconds longer than requesting, you know, a typical web page and still shouldn't take more than, you know, 10 to 15 seconds, hopefully. Again, yep, you can see when there's an asterix that means the code is running. And now that it's completed, there's a number you remember from part one, but a status response code of 200 is a successful request of the web address. So Excellent. So that's worked. Again, I request nicer data to do with the request. So again, you can see some kind of date information. When was the file last modified. Again, it's a daily update and it was modified two o'clock this morning and GMT time and here's the date of when I've actually requested the data, you can see it's 10 to 11. In my time, the server's an hour behind. So that's the discrepancy there. That is effectively it.

So downloading files is a lot easier, because you don't then have to tell Python, we're working with a web page, you're not working with a web page, you're working with a link or a URL to a file, you request that link. And now essentially, you save what you've requested, which is, in my case, now, the zip file. So again, all I want to do here is basically create a file. So the original file has this kind of long, kind of name here. But really, what we're doing is downloading this file here, which is called public extract dot charity dot zip. And I just want to call it something else, you know. And that's the good thing about Python, you know, I've requested the file, and now I can save it under whatever name I want. So I'm going to give it a more meaningful name. So the Charity Commission for England and Wales is the register of charities, and I know what is it. So I specify a file that I want to create, I open this file here, and I open it in right mode. So it's slightly different than the right mode before it's called the right binary mode. And you don't need to worry about that. But if you're working with files, you want to use the wb option. Until that file, I want to write the content of what I requested, which is the file that's a lot of of abstract talking.

So let's actually look at the results itself. And previously, you know, we can, we can open up the folder, and we can look around and I'll do that again to demonstrate. But we can use Python to check if it worked successfully. So again, I can list all of the files and folders in my current working directory. You can see from the part one of lesson three, we've got the Moby Dick data that we scraped, that's the layer. And now you can see the CCEW registered charities file that I just created up here, it now exists, there's a bit more work now to unzip it, because it's a it's a zip file, so I can't just import it directly into Python. Thankfully, Python provides a way of dealing with zip files. So all I need now is the zip file module. And from that there's a zip file function helpfully enough. So basically, take the file I've downloaded, open it and read mode. So I want to read in its contents. And then I want to extract everything in that set, again, may take a moment or two just because it's a reasonably large file.

And now for this for the directory again, you can see that here's the zip file I downloaded. And here's the actual extracted file that is contained in the zip file. And it's called public extract dot charity dot txt. And again, just to actually prove that this exists, and it's real, and I'm not just, you know, doing weird tricks for Python, I'm going to open that file again, I'm going to use a slightly different Python module to

do that. And it's called the pandas module, really good for data management and kind of data formatting and marshalling. It's something we'll use a bit more in the next lesson, so don't worry about it. Now, I basically want to open up that file that I've extracted, and randomly sample five observations or rows from the file, there's a couple of warnings because it's a txt file. So it's, Python has a bit of trouble importing the data in. That's nothing to do with Python, what I do in the stata, I get the same kind of warning message. But you can see that the code does execute, it does open the file, and it does give me a random sample of five observations. And we're information about, you know, registered charities in England and Wales. So here we can see, you know, charity numbers, the name of the charity, whether it's, you know, a company or a trust or a foundation, lots of other information that I'm interested in for my research, and that you may be as well also, just for general purposes. Excellent.

So that's two good examples of how we can use web scraping. So we can extract extract information in the form of text or tables or lists from a web page itself. And we can directly download files that are stored on web pages. Also, in the next lesson, we're going to move on to a bigger, more real example, which involves a bit more of intermediate or a bit more complicated python programming. But again, we'll follow the exact same principles and the exact same process that we've implemented. So looking forward to it