Web scraping and Covid-19: Example 1

Welcome back. This is the third lesson in our web scraping to build social research data training series. In the first two lessons, we looked at the fundamentals of web scraping. So we had a look at what it is how it works, and reasons why you should or possibly shouldn't engage in this research method. We also looked at a particular technical area of knowledge relating to how web pages are structured, and written using a language called HTML. I was trying to bring those two pieces of knowledge together and actually implement a web scraping script.

So today, in this lesson, we're going to have a look at a very simple website, one has been designed for testing your web scraping skills, so we have permission to do it, we don't need to worry about upsetting anyone or breaking anything. It's a free play. I've posted a separate video in the supplementary materials section, showing you how you can follow along with me on how you can execute the code yourself.

So in this video, I won't focus on that, I'll actually just really get on with demonstrating how we actually write Python code to scrape data from webpages. And this lesson, as you can see, this, this notebook contains a lot more for you to read in in your own time. So I'll just highlight the kind of key sections we need to work through just now. The first is i'll show you a little taster of what you can do with Python. This is a very simple piece of code just here. And what I'm doing is asking, you know, Python to print a message to the screen, then I'm asking it to take input from the user. And then based on that input, print another message to the screen. So it's asked me for my name. So I'll, I'll be truthful, and I'll enter it, then it sends me a nice little anodyne message about enjoying Python.

There's lots more for you to practice and to play around with here, you can alter the code yourself and make changes and not worry about breaking anything. But today, let's get straight stuck in to scraping data from our very first web page. So let's just very quickly remind ourselves of the six stages in the web scraping process. So we need two key pieces of information. That's the web address or the URL of the link, or the link of the web page, we're interested in scraping. And then on that web page, we're interested in the location of the information we need to scrape. So that's a visual process, as we saw in lesson two, how you work through a webpage, and scrolling, looking for the paragraph or the table or the list or the image that you need. And then, you know, looking underneath the hood at the actual HTML code, and the identifiers of that information. And we'll see how that works in practice, and very shortly.

So once we have those two key pieces of information, we need to request the web page using Python, I'd say it very similar to typing a link into your browser, you know, and pressing Enter. And you can do that through programming code, which is very exciting. And that's the first step, we need to parse the structure of the web page. So basically, we need to tell Python that it's working with a web page. Otherwise, your programming language will just see a blob of text, which you know, may be readable

and searchable, but it makes it much, much more difficult to extract the information you need. So we tell Python, this is a web page. And as a result, there's all kinds of tools and functions we can use to extract the information we need. that gets onto the the core of what we're doing, which is actually, you know, extracting the information we need in a couple of lines of code. And then finally, we obviously want to save that information, because we're no we're social scientists and we want to use it for research purposes, and at a later date.

So we'll cover two examples. Today, the first one will go through that kind of full six step process, as we extract the text from a web page. And our second simple example will follow the process as well. But this time for downloading a file directly from a webpage. So let's start with the text example. So it's a very basic, simple kind of playground website for testing your web scraping skills. Here's the link here, it's http bin.org. And then this particular web page called helpfully enough HTML, that we can click up, click on that link, you know, and it will request the web page through your browser. And we can see it contains an extract from the classic Herman Melville novel, Moby Dick. Usefully, we can also do this within the Jupyter Notebook using Python itself. So this piece of code here, as you can see, requests the web page and displays it here, in a little cell in the notebook. We can do it both ways. You know, we can do the manual inspection through a browser, which I do encourage you to do, but it also can be useful through Python. Just to take a quick look at the web page you're interested in scraping. Great, So we can see that the webpage exists, and it has some text we're interested in scraping.

So the next thing we need to know is the location of the information we're interested in. So for this web scraping exercise, I'm not interested in in the heading, I'm interested in this paragraph here. So this is what I want to scrape and then what I want to save to a file. So yes, so I've done that visual step. But of course, it's not as simple as saying, I want that paragraph in HTML. So the language that web pages are written in, there'll be an identifier of that paragraph. So I've put the extract of a web page here. So this is the HTML underpinning this web page, and just here, so I've just copy and pasted it into the Jupyter Notebook. So you can see again, that, yeah, it's an HTML page as identified by the top tag. But as we said, In lesson two, it's everything in the body tags, the content of the web page, and what we're potentially interested in scraping. And then this example, I can see that there's one p tag, p representing paragraph and you can see there's lots of texts within those p tags. So this is what I want to scrape. And I'll show you how to do that very shortly.

So the first thing to do is we want to request the web page. So similar to this process here, where I go into my browser, enter the link and press return. They say want to mimic that in Python. So the first thing to do is we kind of do some preliminaries in Python. And there's lots written about this in in this notebook. So I'll leave you to read about the kind of technicalities of what's going on. But basically, you're just loading in the functions, you need to perform the web scrape. And as you can see, it's only three lines of code and then a little message to say that I successfully imported the modules that I need. Great. So I've imported the modules and ready to get stuck into the requests. So in order to request the web page, we obviously need the first key piece of information, which is the web address of the web page, or the URL or the link or however you want to call it. Then using the requests module, which we've just imported. Previously, the requests module has helpfully a get function. So you want to get a web page and the web page I'm getting is stored in this variable called URL. So a different way of doing it is I could just, you know, type out the web page, that would work, that would work as well. And

proof that it works is I get a status code of 200. Again, there's a description here and the link of what those different status codes mean. We're probably just from using a browser you've seen before, or four page or a 400 page, or 501 page. And they all have different meanings. Basically 200 means you've successfully requested the web page, anything in the four hundreds means you've made a bad request. So you've possibly misspelt or mis typed the web address, or maybe the web address no longer exists. And something in the 500 is a problem with the server that stores the web page. So possibly it's done for maintenance, the website no longer exists, or there's some issue on the servers. And so I encourage you to read up a little bit more about that.

But just for now, we'll focus more on the process what we're doing. So that's one way of doing it. But you'll find and you're probably used to it if you're a quantitative researcher, and or even a qualitative, you know, you use codes to represent longer bits of text or you know, longer kind of results or statistics or figures. And so basically, we have a URL variable, and that stores the URL that we're interested in. Let's just rerun that to show that it works both ways. So excellent, very simple. We've made a successful request to the web page, it's a lot less exciting than using your browser because, well, it doesn't actually present the contents of the web page. However, we can access them shortly.

So we can also look at the metadata associated with our request. Now often, this isn't of particular interest to us as researchers because we're interested in the content of the webpage. But it's just worth knowing that there's some metadata behind the request as well. So we can see that, you know, this is the time and the date that I made the request for the webpage that might be useful for you. If you've got a much larger data collection exercise and you want to, you know, keep track of data that's updated on a daily basis, for example. So it can be useful to know when your web scraping scripts actually run. But besides that this kind of metadata is more interesting to web web developers. As I said, if we're interested in that date, we can pull it out from the metadata. So now we have a variable called date. Which has the value up here. But we're more interested in the content and the content of the webpage. As we can see from requesting it using a browser is a heading and a paragraph, we can view this using the text attribute of the response variable. So here's what I mean by why it's so important to tell your programming language that it's working with a webpage.

So what we've done so far is we've requested the web page and now we're we've had a look at what's been returned by the web server. So the web servers sent back the web page to us. But because we're not using a browser, basically, all we see is this kind of blob of text. And it's not just the content. So it does have the content that's rendered by our browser. But what it has is the entirety of the HTML page that has all the HTML code plus plus the content, as you can see here with the HTML tags, and the the opening doctype tags, telling you that this is a web page file.

So that brings us to our second step, which is telling Python or whatever programming language, you're dealing with that, hey, this is a web page, you need to understand it in this way. And then that allows me to pick out the bits of it that I want to scrape. So this brings us to the second module we imported into Python, which is called Beautiful Soup, no idea why it's called that, possibly a little exercise for yourself. Beautiful Soup as a way of basically telling Python, we're working with a web page. And because of that, here's a series of tools and functions that make it easier to pick out and extract the information we need. So all I'm doing with this stage here is I'm creating a new variable called soup

response. You can call it what you want, I'm tending to follow kind of standards. And if you look at the help forums for, for Python, in particular, when people post solutions, you know, they'll name variables the same way all the time, you know, just swap out its standard. So all I'm doing is I'm taking all of the text stored in the response variable. So all of this here. And I'm saying, hey, this is HTML. So I'm using the soup module to say, hey, this is the HTML page and store it in this variable here.

So what I do it now we see the nice structure, we know it comes with HTML. So HTML, again, is a hierarchical structure. And now by using the Beautiful Soup module, and the soup function, specifically, no, Python understands that we're working with a HTML file. So now we can see the nice hierarchical structure, we can see the clear separation of different elements of the web page. And those elements are represented by different tags. So that's excellent. That's the second step completed, very simple. And it's essentially only one line of code. Here on the second line just tells, you know, Python, a print the contents of the soup, underscore response variable, so excellent. Python knows what we're dealing with.

Now, let's get down to the actual extraction of the information we need. So again, we're going to use the Beautiful Soup module, this time, we're going to use the find function. So we take the soup response variable learner, so that contains the HTML page, and all I'm asking you to do is find the p tag. So P represents paragraph, I'm basically just telling Python go find a p tag. So one p tag in in that webpage, we know from inspecting the web page that there is only one p tag. When there's more than one, which we'll see in an example later. In a different lesson, you know, we have to do a bit more digging. But this is a very simple example. There's only one p tag. Let's find it. And let's return its contents. Excellent. So now I have a new variable called paragraph. And that contains the contents of the p tag. Actually, at the moment, it contains the P tags themselves, as well as the content contained within them.

So that brings us to our final step really, of the extraction, which is obviously we want to save this as research data, we don't want the tags in it, because our futures stage, if we're using in vivo, if we're using states or whatever our whatever package you're you're using, you're going to have to remove those p tags, because they're not germane to your analysis. They're not the content of what you're interested in. So all I want to do is take my new paragraph variable, which contains the P tags in it, and I want to say Just give me the text of those p tags. And by doing that, you can see now I've extracted the text within the P tags, which is excellent. So now we get to our final step. So we've requested the web page. We've told Python, we're dealing with a web page and because of that, we've been able to use some tools to pick out the particular paragraph and the text within that paragraph that we need. That information is still stored in a variable called data. That's really no good to us as researchers, of course, we want to actually, you know, export that to a file, and then re import it into something like invivo, or stata or SPSS at a later date. So all I'm going to do here is I'm going to create a new file, I'm going to call it Moby Dick scraped data, it's going to be a plain text file. So just a txt file, I'm not saving it as a CSV or an Excel file or anything more specific than that, just now, I open the file in write mode. And I'm just using a quick shorthand to refer to it here. I open the file, I write to that file, what's contained in the data variable.

That's a lot going on. So that's a bit abstract. So let's actually look at the result of that step. So what I'm going to do is I'm going to check if that file was created, I can do that manually, you know, I can go, I can actually go to the folder itself on my machine, which I'm doing on a separate screen, in case you think. If you think I'm doing nothing, bring that down to just now as you can see. Yep, so here we can see Moby Dick scraped data 24th of the 8th, which is exactly right. Now, as you can see that it's a new file, open it. And as you can see, here's all the text from the webpage. So I've extracted and successfully saved the text.

It is possible to do that through Python as well. So I can confirm that it actually worked using Python. So if I use the list directory function, which is part of the OS module, basically just lists the contents of the current kind of working directory. So I'm currently in the code folder, which contains all the different Jupyter notebooks. And you can see that Yep, there is a file called Moby Dick scraped data dot txt. Is there anything in it? Well, as I just demonstrated manually, Yes, there is. But I can use Python again to actually confirm, and there is information in it. So again, I'm opening the file that I've created, this time in read mode. So I'm trying to read in the contents of the file. Again, the F is just saying, just refer to all of this here as a shorthand. So just like write F, instead of writing this again, create a new variable called new data, and that is, contains the information in the file. So reading the contents of the file, store it in a variable called new data and print the contents of the new data variable. And voila, that is our very first successful web scrape.