NCRM text data workshop: Part 2

Hello, everyone. Welcome back to this NCRM workshop on text data analysis. This is part two of the sort of three-part workshop we're doing today. In the second part, we're going to build upon what we looked at in the first part a moment ago, where we were looking at sort of a basic introduction to Python. And we're going to take what we learned there, and we're going to use it to build a web scraper in order to scrape some very simple text data from a from a relatively simple website. Okay. And then in the first part of today, we'll go on to do some very basic analysis about text data.

Okay, so collecting textual data, is perhaps one of the most difficult parts of any research project that uses textual data analysis. And this is despite the fact that there are numerous ways in which to gain access to such text data. So, for example, you can find if there is service already being created and analysed by someone. So, for example, in my personal research, I make what data make all my data freely accessible via my GitHub page. So, if you go on there, you'll find text data files, you can download news, you can bind, you can use an API to access and text data. One example of this, for example, is the Python package tweepi, which is which enables you to easily build scrapers to scrape data from Twitter via the Twitter API. You can if you're part of a higher education institution, quite often your university you have subscriptions to databases that contain though is text data sets that you can download. Or Finally, you can scrape your own data.

For the vast majority of your research projects pretty much what you're going to want to do right, because you your research project probably has a very specific focus and the chances of the data that already existed that the field your needs. So quite slim. So that's what we're going to focus on here today is scraping our data. In doing so we're going to use two main modules. Hopefully you remember what modules are from part one, we're going to use the request module. And we're going to use the HTML aspect of the lxml package. Sorry, that's a bit of a mouthful.

Okay, so what are these packages? requests is essentially a package that enables the user to communicate with a website. Okay, it enables you to build your Python code. And using the request package, you can then get your Python code to simulate a person using a web browser to visit a website. That's essentially the request packages, i.e., HTML aspect of the xml. Sorry, the lxml package, then is just a way to take what the computer gets from the web page and convert it into a text that we can actually work with as humans and data analysts. Okay. And you'll see what means a little bit more as we sort of go through this second session.

Okay, so, first of all we want to do is get a website. So, for this exercise, we are going to scrape the University of Exeter's sociology staff list page, okay. And there's absolutely no specific reason as to why I chose this particular page at all, okay. And we're going to sort of get this information, which is all freely available on the internet. Anyway, we're going to do some text, some simple text data analysis on the job descriptions that people have for the various roles. So, we start off by getting a URL, which we've done here, we just take this, and we create a new variable called URL. And this variable is a string variable. And it's just making a new variable called page. Page involves calling the requests module that we've already imported.

And then form request we use with dot get function, and then feed in our URL. S,o this just says right python. I want you to use the request package and form that I want you to to get function. And I'll basically want you to go and get the data. The HTML code is contained at this URL with a map. Okay. And we can test whether or not this works by typing the word by using the page content function. Note here that the HTML for an entire web page is incredibly long. So, all I'm saying is I'm just using an index here just to say, you know, just give me the first few 100 characters, just so we can see that it's actually grabbed something. And you can see here it has, okay, equally, if you were to remove that, you get the entirety of the HTML page, but I be really long on the screen. So, we're not going to bother looking at the whole thing. Right now. we just want to; we just want to make sure it's grabbed something.

Okay, so we now have this raw HTML code, okay, which we've collected from the URL using the request package. Okay. We now need to sort of be able to handle this code in a way that enables us to a make sense of it and be extract the data that we want. And to do that, we're going to use this lxml package in order to in order to sorry, apologise. I do apologise I said the word package and set off my Alexa, where was I. Yes. So yeah, the lxml package, then enables us to get this HTML code and sort of build what we call a tree. Okay. And what this essentially does, is it simulates. It simulates a user going to a web page. And clicking on a link like this is gay. So, it simulates that behaviour. Okay. I apologise my stock photo image. Yeah, so simulats our behaviour, okay. And so to actually do this, we create a new local tree. And we call our HTML, the package when poured, and we say, from string, page content, page, your content. So what this does here takes us page dot content, which, if you remember, generates all the HTML form this page. And that's saved as one single string. Okay, so just as this is a string here, all of our HTML code is saved as a single string just like that. This command then takes such a string, and it converts it into this sort of hierarchical sort of tree structure. And if we were to print that out, you see this print element HTML at, blah, blah, blah.

Okay. So what it's done, though, is that it hasn't print out HTML or a sort of representation of this tree. Instead, it's created an element that internally it knows it's HTML coded a tree. So, it's made the different colour, we don't want to see that. And it just tells us that and said, this is a HTML elements, which is useful. That's all we need to know as users from time being. Now, there are a bunch of different ways to access different elements of the string. And it's also important to note that this tree contains the entirety of the layout of the page, right. So, all of this stuff there, all these little pop ups and everything, all of this here, this here contains all of that. But we obviously don't want all of that, okay, we only want to access specific parts. Specifically, here, we're only interested in the user profile was going down here. Okay, so we need to find a way to isolate these particular tree branches from the overall tree. And the way we're going to do that, in today's session is using these things called xpaths.

Now what are xpaths, it's worth we're going to spend some time here look at xparts because they're a little bit tricky, but they're worth understanding. Because once you understand how they work, it makes extracting data on websites incredibly easy. Okay, so essentially, yeah, xpaths way of navigating this tree structure that we've built. And you can think of it as each sort of subbranch on the tree having its own specific address, and that xpaths are a way to essentially an address to get to that part of the tree. Okay, so he metaphors are not my forte.

Okay. So as an example, let's say that on our web page, Hey, let's say we wanted to look at the research section. And so let's see, yeah, we want to look at, we want to get the research section of this. Okay? We can get the xpath. Okay? So, if you look at this line here, okay, we were to run that. Okay, we see that has grabbed our research section here. Okay. And this is the xpaths to get to there. Okay. Now at this point, you're perhaps asking yourself, well, that's fairly well and good. But how do we get this rather complicated looking piece of code, if you will? Well, with most web browsers, I'm using Chrome here, Firefox has a similar feature, it's actually really easy. So, you right click on your web page, go down to inspect. And it pulls up this, which is all the HTML for this page. You then go to the element you're interested in, click right click, click inspect. And it takes you to that corresponding part which will be highlighted as it is here, right? You can right click on that and say, copy and then x path that will then give you an xpath, which you can feed into your string variable, just like I've done here and it will give you that, okay.

Now, you might notice a little bit difference between what I've just typed in now I typed in here. That's because I just added a forward slash text parenthesis onto the end. Okay. And that was just me saying, I want to get the text from this section. Okay. So instead of just giving me the HTML element, which just being in a normal path will, by putting forward slash, text parentheses, I get the actual text set, which is our research. Okay. And for all the xpath, you see, as we go down, that's all we're going to do. That's exactly how you get the xpath.

Okay. So, let's say we wanted to get the xpath for specific staff members profile. Okay, go down to here. Again, one, click inspect placement at that location there. Now, right click on it again, on the highlight that again, and click copy, x. And that gives me the x bar here, right. And if I print that out, I get told that HTML elements, and again, like with forward slash text, speech marks, I get the text, which is my name, because it's my profile, and I have that much of an ego. I'm running this on my own name. Okay. So what if we then wanted to get the URL of my staff page? Okay, so some of you might have noticed that when you click on a user profile, it goes to a separate page, which has a slightly different URL, right? As staff for the landing pages sociology forward slash star forward slash. And then the website is made up that uses our surnames and to take us to the page that contains our specific profile. Okay. This is relatively simple here. But some websites have more complicated URL structure, we wanted to get the URL of that, well, just as we type here, just as we put, after the xpath, we put text parentheses in order to get the text said, if we put the outside href, which stands for hyperlink reference, we get the reference for it. Okay. We can then combine this with our URL. they're all from up here. Right, we can then just say, ask Python to print out our URL variable plus this command, here I am, and because you can add strings together, the output is our full URL. Because Jupyter Notebooks which we're working on here, are interactive. That means I can actually just click on that it takes me straight to the page.

So, there are some other xpath expressions which are quite useful. Okay, so we've seen text and href. There are some others that are quite useful. So the contains, the contains function enables you to get the xpaths have various elements that contain specific texts. So let's say for example, I wanted to look up, find any texts that contain the name, Chris. Okay. So I create a new variable called Chris. And I'll say equals tree, xpath and then contains Chris. And this looks for any a tag that contains the word Chris, what do you mean by a tags? Well, like I said, if you look at a specific profile, you'll see that the individual profiles now you'll see the href, which we call a moment ago, the text, which is the name if he was all included within a oversight of it. Okay. And that's just because you can think of it out of its simplest being an A elements. Okay? So all this code here saying, right is that these two forward slashes, say, you know, skip all of this stuff above here, okay, it simulates new typing all that out for you. And it says, Gget me any of these a elements, a elements that contain the word, Chris. And it does that. And when I then ask it to print this out for me, and it tells me that there are four a tags of Chris. Okay. If I also ask it to print out, Chris, you'll see we get four set foot elements. Okay. And if I did the the technical stop text, is it. Its text contains content, and I've got that link down there anyway. So yeah, if we just print out the chris, you see, we've got the four separate elements. Okay.

And if we can't be done on our list, and if let's say we wanted to look at the actual name of the first person on our list, we type this command, which says, you know, Chris zero. So as we saw, if we did this, you see, we get a Python list, where each element in the list is a specific HTML element. Okay. And this is the four, it tells us there's four elements in this list, which we can see that down here, I say, get the first element and get the text on it, we can see that the first element is for Christine. Remember that department? second element is Chris, my colleague, Chris Playford, and so on and so forth. Okay, and what a Christopher there. So you can see, it hasn't just found a match specific match to just Chris. But any word that starts with a capital a name that begins with a capital C, and then contains h, r,i, s.

Okay. There are some powerful operators. And these work exactly the same way as the arithmetic operators do in Python that we looked at during the first session. They enable us to do certain things. So for example, if we said we got fed up of me looking at my own profile, and we said, we wanted to set all the links that do not have my name, Lewys, in the text, we can do something similar to this. Okay, so let's run this, see what we get. So, I've asked it to give me all of these a tags, okay. Remember, each of these a tags corresponds to a specific user profile. Okay. So basically by saying how many a, give me the length of the list of a tags here, I'm essentially asking how many profiles are on the page. And I print out tells me there's 167.

Okay, I then say that I want to get all of the tags again, but ones that do not contain my name. Okay. And I say that as a variable, no, Lewys. And if I asked you to print me know, Lewys, you can see the output is 166 profiles, because it's removed one profile, specifically mine, because it contains the word Lewys. And because no one else spells Lewys in this weird way, is obviously just my profile has been removed. You know, if I wanted to be more specific, I could put a space and we still get to say we still get the same result. They'll just be more specific to me. Okay. Likewise, let's say we wanted to get all of the Chris's If you remember, we have four hits, the names that contained the letters C, capital C, h, r is, let's say we wanted to get all of those, but we wanted to remove my colleague, Chris Playford, we can use the sink minus, we use the bug to get over Chris's name be this bit here. Okay, and not contains Playford. All we're saying here is, do we match exactly like it did before, exclude any of the profiles that also contain the string playford. And if we run that, you can see we get three, which makes sense to have four hits move on, we have three. And there are some other LXML methods, we can use, in addition to x pop stuff in order to access data from the HTML method. Okay, and HTML objects. We've already seen the text content method, that's what we used up there.

One package, one element, one aspect of the HTML package we can use in all is a different way to select objects. And we can do that by selecting what we call the cascading style sheet names. Okay. And that's basically what was having a bit more of a knowledge of how HTML works. And so, if you look down here, we'll find in the main, a div class section called main main menu. If you scroll down through the collection, here, eventually you'll find all the core div object, if you look at does exactly the same as the sort of inferiors exactly the same as the elements we're looking at a moment ago, which is a different type of elements. And we'll find one called Main Menu, okay. And then we use the HTML method dot get tech content to get all the content from the main menu. And you can see what it's done is get all of the text from this specific object here. And again, I find that by using the right click and inspect, open that up, you'll find all the details, all the information you need down here.

Okay. There's also a defined class function, which basically enables me to, to select the profile lists a different way again. So, again, you as you scroll down here, you'll find elements who profile lists. And, yeah, and where is this section appears titled main menu in the CSS selector? For one, down the profile name to say this profile lists, okay, and we can print out the P list there, we see the HTML elements. And then I use the just use the dot content, text content function again, in order to get the text for the first entry, the first element in this list, which is the content for a head of departments, Mike Michaels, who is obviously at the top of the list, because he's the head of department. He also get elements by ID, which is, again, another way of selecting of using a different part of the tag to select the part of the HTML code your one, okay. And the main reason for going through these different bits here is just to get you to realise that each tag has different ways of accessing it.

Okay, you can access it by the file path, by the tag name, by the tag tight, and so on. There's different ways to access the same, the exact same part of the HTML tree we've created. And I suggest, that's all we're going to cover in terms of x paths, and CSS is CSS today. If you start to do a lot of web scraping, or you start to build other scrapers, I suggest you spend some time learning about XPath and CSS. There's a lot of useful resources online. If you just type in x path into Google or CSS into Google, and you'll find tutorials that will help you understand that I'm looking at all wheels and also having a web page open and looking at examples of HTML code like this means perhaps a bit little bit and you'll pick it up in no time. Okay, it looks really complicated at first, but the more you practice, the easier it becomes much like learning a natural human language.

Okay with that covered. Let's now take a look at building our first website scraper. Okay. As I said before, we're going to scrape the staff profiles of the University of Exeter's spa page, their sociology, philosophy and anthropology page. Okay. And we're going to do this in two steps. Okay, first, we're going to get the links to each of the staff members profiles, and then we're going to extract the profile content. Okay? So, if you were a user, this is going to be equivalent to you going to the page, clicking on the profile, getting the data, going to the next profile, getting the data, and so on and so forth. For all of the 167 profiles, we have one here, okay. Obviously, as a human being, you don't want to do that it takes too much effort. So, we're gonna go to web scraper to do it for us.

Okay. Now, we've already created a tree, which we did up here, we started looking at exports. And we can still access that. So, all we do here is say, is, again, use some CSS, right? We're saying, look, this is a cross what we've got here, big cobis with x pass and sort of CSS inside. So we're saying, we're going to create a new variable called table. And that table is going to be equal to getting the XPath. For our tree element, where all the x paths, we're going to get the x path for each of the class for each of the tags. That as a class object, that equals profile list. Okay? So that involves going to the page here. It's navigate down, you see, we've got app tags here, right? Because several academics out there, and you can see here that we have classes, and they're called profile lists. And if we click on that, contained within here, so so what we have each of our individual profiles, and you can see the corresponding profile by being highlighted on the left-hand side as I scroll down, okay, so these are all, each of these eight tags that correspond to output files are saved under this class object called lists, profile lists, okay. And like I said, before, these two forward slashes in the a star just says, simulate all of this stuff above that, okay, so it's saying get each of these for us. Okay. That's all that line does. If we run that, we then ask it tell us the length of the table. This is 127. And if we have to print the first element of that, it gives us that tells us this HTML element. Okay. Now, if you look closely at these, right, you will see that each of the elements in our profile list contains this annoying header, right? academic staff. Okay. We want to get rid of that. Okay, we don't want that in our analysis at all. Okay.

And these will be sprinkled throughout. Yeah, so these headers are sprinkled throughout our content for out page. You see the head department here? Yep,the academic staff, one here, and then the others all at the same. So we're gonna need to deal with this issue. Okay. And also, you'll notice that the information we want is actually contained within that tag, and that the header rows don't have these tags. Okay. So that's all shall we deal with that is by using this, we say, in table we find all that begin to TD, okay. And it will print this out. And the way we get rid of these two, if we select our table, we set the first element here, we print this out, we see that our header contains none of these TD tags, okay? So, if you look at these, these don't contain any TDs. Okay, but if you scroll down to here, in tr, you'll see that the attributes and protocols do have the TD tags. Again, these are just like the A element tags but just a different type of them. You don't need to worry about the difference between at this moment in time, okay, they think the takeaway here is that each of the actual bits of individual one has a TD tag, these annoying headers that we don't want, don't have a TD tag, they have a th tag instead. So, we run this, you know, find all. The Find all element is just a function in the request package, it says, find all of the HTML elements that fulfil this criteria. It acts much like a conditional that we saw in part one, right. And we say find all of the tags that contain this TD element. And if I print out, you simply print a list here that has no objects in it, because there are no TD elements in these annoying header bits, okay. But if we say look at the non-headers, okay, and find the TD, you'll find it will print stuff out.

Okay, that's this print statement here, which gives us free HTML objects. That's because in the non header section, like I said, here that we do have the TD objects, and you see we have three of them. Okay, the first one corresponds to the profile name. The second one is that sort of detail is the profile user details. And the third one is our building name on campus. And so in these print statements here, then I just say, for my non header variable, give me the text content of the first and the second bit, and you can see, we get the corresponding, we get the name and the bio, if you will have individual concerned.

Now, this above code, the purpose of this was to do two things, one to show you so that you have to be aware of how some websites are structured to avoid you could end up including bits of text into your data set you don't want but was also to showcase this useful find all method, which is basically a way of finding the next path within another x path. Now, now, we need a close to sort of actually build or scrape to extract our content, the last thing we need to do is get the profile link for each of the individual profiles. And again, if you were to look each of the staff pages, you'll see that as you scroll down, if you look for the TD element that contains the view, the profiles name, and if you were to expand that you'll find the page with i.e the url for that specific protocols contained within this username function. Okay, so this profile name function, okay, which means so we can access that, right? We just, we just call the specific HTML object, we want to say find the a tag, the a element, yeah. Yeah, V tag loops there. And from that extract, the H ref attribute, so our code goes to the a ref. So it goes sorry, it goes the a tag, finds the H ref attribute and gives it back to us. Okay. And if we were to print that out, you see gives us the Michael, which is the surname, which is the you the specific URL for that part of the profile. Okay.

Now, we pretty much have everything we need. The only thing we need to do now is build a for loop, like the kind we saw in part one, in order to loop through each of the profiles on this page and collect each in collect the data that we want. So that's what this bit of code here does. Okay. So first, we create a table object, which is a collection, if you remember collecting the HTML using the request package to go to a URL and take all the HTML data and then convert it into this tree. And we say we get the main content as pretty much everything on this page. We then create a blank list member list from the first one from the first video, right, that can store all types of content, we create a blacklist. And if you remember this app, like data containers, so we created this blank place with the intention that we're going to put our user profiles into this container to store them there.

Okay. We know that each of the profiles is saved as a specific row in our table object. So, we say for row in table are you for each profile on the page do a thing okay? this for loop works exactly the same way as our species list for loop. In the first video works, okay, that's just iterating through each element in the sequence. But instead of just printing out that element to the screen, like we did in the first video, we're going to be a little bit more complicated. We're saying, from each of these elements in this list, clean object called cell, wherever cells contain all these TD objects, ie, remember, ignore the headers, which don't contain TD. But yet the parts of the staff profile but do contain TD, ie, the profile name, the bio and the building location. Yeah. And if it doesn't equal zero, this is a condition we're about to put it because some staff members don't have any information on their profile pages, which could cause an error when scraping, so I just say, you know, if they don't have any content, ignore them. If they do, add them to the list, again, we don't want a scraper won't work. If we don't put this conditional in B, it's not really useful to analysis big have any data anyway.

Okay. We then say, Okay, this next bit of code man says our page URL is going to be the current cell is going to be the first element of our cells list that has an a tag, okay. So, the cells, if you remember, contain three HTML elements, which correspond to these 1233 parts of the page, we're saying, we're going to get through to the first element, which is the name, which if you remember, contains the href object, which is the URL part profile, and we're going to extract that, okay. And again, some profiles don't have a, well, one of the profiles on this page doesn't actually have a hyperlink to a user profile for some reason I'm unaware of. So I just say, you know, if, if this object actually exists, do this. If not ignore that. Okay, that's all this says, Okay.

In Python, and requests package terminology. If we ask to get an element like this, that doesn't exist, then it's saved as what we call a non-type object. So, for that one specific profile, it doesn't obviously have an a tag because it doesn't have a proper name associated with it an H ref object associated with it, that will be saved as a nun type object, okay. So, I just said, you know, if this object is numb, ignore it. Okay? If however, it does contain a type object, which the vast majority of the protocols do, so we can see here, find the hatred left object from that a tag, i.e., yeah, go to the first TD, go to the a tag and grab that hatred. Okay? object. And then save the content for each of these three sections. To a dictionary, okay, so we save name. And remember dictionaries from the first video, okay, where we have key value pairs, key things you want to look up. The values are the definitions if you're thinking of the analogy to the natural language dictionary, so we say first key is name. That's going to be the name of the profile. Okay, so the value is going to be the literal the text. So, in this case, Professor Mike Michaels, Dr. Alexei, so on. The second element in the dictionary is going to be the Pedro L. So, it's going to be you know, this bit here, Michael. The third one's going to be position. Okay. And so that corresponds to the job title here. Okay. We're basically going to say this all as the position, okay. That's just what we're going to call it now. And that's what this bit of code here extracts. Okay, move on that. And then we just picked out this line here, which just tells us that this operation worked successfully. And it successfully extracted information for 108 team members of staff. And then we can just ask to explore one of them at random.

So if we asked to look at phone number 12 in the list, we see we get all the details for Professor john lucre What am I 012 so if we be too small Look at mine. Just cuz I'm not sure how much my colleagues will appreciate us talking about them in depth and also because I can take it out. Because I'm lazy, I've got a lot less information on my page and others don't tell HR about me. So ,you can see here we've got three, we've got a dictionary that contains three key value pairs. Okay, locks have got name, which contains the text of a name, which corresponds to that on the website, Dr. Lewys Brace. Page url, if we were to minimise this a bit, and we go down to the profile list here, and we'll find the profile link to me, here we go. One, and then we go into the Object here and the href here we've seen our URL, and that matches to that. And then position is that current data analysis, which it says click import from there.

So, for our second step, okay, we now need to go to, we now want to go to each of these profiles, okay, and actually extract the content from them. Okay. And we're essentially going to do this, like I said a moment ago, by going to the page and looping through each of the profiles in turn. So, we start off by going to the page in usual way. And you can see here, I'll print out my usual thing to tell me that it successfully extracted all the information form this specific page, and here of our stick, you know, just as a test just to teach you guys, I've asked it to extract all the content from Mike's page here.

Now, after looking at the page, if we were to go and look at the HTML for this page, right, we're finding that all the text and stuff we want is contained within these p tags, right? It also has little p tags. Okay. And so we can actually use these to extract the content that we want. And so, for example, if we wanted to get all of the overview text, I all of the text contained in here, events in multiple paragraphs, we can just ask it to go to the point, we can use the CSS selector to find the part that has the ID equals overview and select all of the P tags from that. And if we were to run that, and everything out, you see it's extracted the text from there, okay. And we can use this to extract pull out aspects of Mike's page. Okay, so instead of looking at the overview, we wanted to look at the research section, okay, you'll see this corresponds to the research. So Oh, we can grab everything at once. So here, I've just gone, you know, this code essentially says, go to this page here, and just extract all of the text because all of the text we know is saved in these little p tags. So just extracted all you can do that and print out, you see, we get everything we need.

Okay, so we have this, you know, telephone number office name, you know, so on, even right down to the copyright text that we see at the bottom of the university page. Okay. So ,we say build now this first scrape of ours, we're going to keep things simple, we're going to use all of the available page content moving forward. Now, now we go, we start to enter one of the last stages of what we're going to do today, okay. And that is, we're going to do all the above, but for every single profile on the page, okay. And we just do this in a very simple way by putting into a for loop. Okay, you can see all this is what we've done above that, because it's in a four loop we're going to do for each of the staff profiles in our staff profiles. list, which if you remember is the list we created above where we saved all of these staff profile information. If we run that. In Jupyter Notebooks when the circle here is fully black, or grey, depending on the contrast of your screen. It means pythons currently thinking economy operating. Okay, so we can see it's currently going through a collective all this profile information. So, we just have to wait for that to them for a second. So, I'm just gonna pause the video while that's running. It's going to take a couple of minutes. You won't notice the difference cuz I'm just going to pause it and pause the recording again. Okay, so that code ran and he collected all the details. Okay. Just as just an example, we select a random profile type to look at profile number 11 on the page, and just grab all the content for that. You can see we have all the information there. I was number three so if i run that all my details there, including my interest what i teach, and so on.

Okay. Now let's say as an example, we just wanted to look up the number of staff on the spar page that say that they work with quantitative analysis or statistical analysis, for example, we can build a very, very basic search. And that's what we do here. So, I create a new list called quantitative, which is empty, just like our staff profile, this was at the start. Okay, I then say for each of the rows in our staff profiles, which we've gone through a moment ago, if the page calm, okay, if the page content in the vo contains over the word quant or statistic, add that to this list. Okay. So, it goes through this page content for each member of staff on their staff page. And if it contains the word once, or statistic, statistics adds it to this list here. Okay, adds it to this list. I then just asked Python to print out that list. And it tells me it's found eight members of staff that contained that have the word quantum or statistic on their profile page. You'll notice mine hasn't come up. And that's because if you didn't, if you look at my page content, I have quantitative, not quite. And that's to do what I mentioned in the first video, where in this specific case, pythons looked for an exact match. It says look for quants, not quantitative, okay. But it's not eight members of staff, they can take the Have either of these two words in their name.

Okay. So that's just a quick way that you can sort of serve data if you wanted to explore it. But now that we've got our data, okay, we've got all our staff profiles, we now want to save this text data to a file, okay, and then that gives us some text data we can then do some analysis with. To save it to a file, we import this package called JSON, we're gonna save as a JSON file. And a JSON file is just a way of saving hierarchical data. Okay, so all this is going to do is save each of the dictionaries, if you remember, each profile on this page has been saved as its own independent dictionary, it's gonna take each of those dictionaries and put them into a list and save that as a text file. Okay, as I import the JSON package, and then I just use the with open, which is a way of saving a file path, save as j file. Okay, all this, this open function takes two arguments. The first argument is the file location and the file name. Okay, so say, I wanted to save my file, I'm about to create this location, I want to save it as profiles dot JSON, and I'm opening it with W. Okay, so I'm going to write to the file, okay. And I'm going to save it as a j file, short JSON file. I'll leave a link in the literature. So you can look up the various differences for these. But basically say here, I want you to see my staff files information, as a j file, we have a point with a four point index, and it just, it's four whitespace characters in just so I can view it easily and separate things, we have a comma. Okay. And then this line here just says I want you to separate each of the elements. So, each of the dictionaries in my list, I want to be separated with a blank line, I save them to a new line. And when you run that line, you'll get a file that looks something like this, you'll see my profiles, this has been generated where I wanted it to. And if I were to open that up, you'll see that each of my dictionaries that correspond to a user profile has been saved, separate by comma and a space as I asked them to.

Okay, so that concludes the second part of this workshop. We've got one more part to go and in that next part, we are actually going to start doing some very, very basic text analysis, just so you can get a grip to some of the basics of it. Okay, I shall see you in the next video.