NCRM text data workshop: Part 3

Hello, and welcome to the last session of this sort of three-part workshop for the NCRM on text data analysis.

So, in the first video, we covered the basics of Python. In the second video, we took what we learned in the first one and built our first sort of very basic web scraper in order to extract text data or user profiles or one of the University of Exeter start pages. In this third part, when we're now going to start doing some very basic text analysis, okay, we don't have time here to get into some of the more advanced techniques such as supervised learning methods and so on, but we are going to start with some basic stuffs upon which you can build as you go along.

Okay, so this whole notion of text data analysis is linked to something known as natural language processing or NLP, which is essentially a subfield which combines various aspects of artificial intelligence and linguistics fields such as that, okay. But in a practical sense, it's the whole idea of using technology, computers and algorithms and so on in order to understand natural language i.e., human produce language, gave us the idea of being able to decipher understand it, analyse it, and so on. And we see many applications of NLP in our day to day lives, such as personal voice systems, such as our Alexa's, Google devices, and so on. Google Translate, it's, that's just a form advanced form of NLP, Microsoft, the spell checker, my grammar checker, Microsoft Office, and so on and so forth.

There are however, some difficulties with NLP, okay, firstly, it's not simple, okay. It's a very, very natural language processing, i.e getting computers that deal exclusively in zeros and ones understand human language is intrinsically a very, very hard problem. Okay. But despite that, in recent decades, a large amount of progress has been made particularly in the last 10 years or so. Some of these problems related to how it relates to the nature of linguistic rules for passing and understanding language, okay, a large amount of our understanding of language use comes from context, okay?

So, for example, sarcasm involves high levels of abstraction that computers don't understand. The elements such as that make NLP very, very difficult. And as such, a comprehensive understanding of human language requires both words and concepts in order to make sense of what is being said. Obviously, computers can't have access to the sort of mental constructs and concepts. Okay, and as such, we often make mistakes. An example here, an example is this, okay? So we've got an English phrase here, this is the spirit is willing, but the flesh is weak. You try that in somewhere like Google Translate, and it gets translated into Russian, you'll get the vodka is good, but the meat is rotten, okay, because each of these have very specific concepts associated with them, that that and that the NLP algorithms in Google Translate haven't learned or haven't got access to so translate it as best as it can. Okay?

And these kinds of issues are due to how NLP works, okay? Because essentially involves the application of machine learning algorithms in order to identify and extract language rules, okay? And therefore, converting it into a form that computers can understand. Okay, this means that these algorithms are only as good as the sample set that's been trained upon, okay. So for example, you might be able to train sort of NLP algorithm on, i don't know, on a textbook for computer science for example, It'll learn it. But then the language it learns, because it's only sort of learned the patterns in that textbook, it might not be a, it probably won't be able to understand language use in other in other forms of printed text, such as magazines and gossip columns, for example. So there will always be mistakes, okay, and it's worth bearing that in mind as we go along, no natural language processing is perfect.

However, as we're now living in an age of, quote unquote big data, which is the buzzword everyone likes to throw around, we use NLP more and more. And the reason for this is because when people say they're working with a big data set, most of the time, they mean they actually refer to a text data set, you know, so data has been extracted from Twitter, for example, YouTube comments, something like that. Okay, so, this has really driven some of the advancements in NLP in recent years. And like I said, there are now some really complex ways in order to visualise text data, such as words avec algorithms, dialect models, and so on, which can be quite complicated and draw upon many concepts we see in machine learning. We don't have time to get into those today, because this is meant to be a very basic introduction workshop, where I'm going to introduce you to some of the basic concepts that you still need to know before you can move on to that kind of stuff. Okay, so we're going to cover things like tokenization bigrams, engrams, stuff like that. And then I'll show you something fairly trivial, such as how to make a word cloud, which people like to do and so on. And then hopefully, then this last part will then set you up to be able to learn some more advanced methods as you go along.

So let's begin by importing the packages we're going to use. We won't use all of these packages, but we will lose most of them. And I'll explain what each of the packages are as we go through instead of just listening them all off now. And the time being, we're just going to run them and load them all in. We can see my little circle up here is dark grey slash black, depending on the contrast of your screen. That means it's thinking, and it's gone hollow again. So it's finished thinking now, i.e it's finished loading in all the packages.

Okay, first of all we're going to want to do is loading the data file that we created last time. So, if you remember in video two, we, the last thing we did was save our profiles as a text file in adjacent style format, where each profile is saved as dictionary type object within a wider list. Okay. We're going to want to load that in. And we do that using pandas. Okay, so we've already imported pandas as PD here. So we say df equals PD dot v. Json. Okay. Now, df is just computer scientist’s shorthand for data frame. That's all that means. Okay. And we're just gonna call it data frame here, because it makes sense. You can call it whatever you want. You could call it profile data, if you want, whatever, I'm just calling the data frames. We then say from the pandas package, we want to use the read underscore Json function, which we're going to feed and we're going to feed one argument into this, which is a string, which contains a location, the file location of my file, along with the file name, and its type, which is dot json. Okay, we've got profiles dot Json to loading in here. We run that line of code. And we can see it very quickly loaded the data in.

Okay, I'm now going to print the data to the screen. As you can see that our json file which is in this format here has been loaded in and what you can see, if I click this. So, we're wanting to do, if I put this over here a little bit, you can see that it's taken each of the name keys in each of the dictionaries, and it's put them all into one column. And if you scroll down here, you can see it's taking each of the page content keys and put that in a second column. And finally taking each of the page URL keys and put them into a column there. And then same with position ok so we've got four columns, okay, it's, in other words, this v Json. function is taken written in JSON format, and that JSON file create a data frame, whereby each of the four columns, the four columns are really individual's name, the contents of their page, the page URL, and their job title, we call position, okay, in each of rolles a different individual profile from the university staff page, okay.

This command here print list df, just tells Python to print the column names of the data frame and if we scroll down, that's what this is. Here. You can see we've got four variables to play with. Okay. In this session today, we're just going to play around with the position title this is a nice simple variable, which gives us some stuff to work with. And it will serve as a nice introduction to text analysis.

Okay, so for those of you who are familiar with standard statistical analysis, you will know that before you go on and do your advanced statistical tests, your regressions and so on, you'll quite often do some descriptive statistics, okay, so calculate some measures of central tendency, print some graphs of distributions, and so on. And you do these to get a feel for the data. So, you can see if there's anything or any outliers, or just the general layout of the data, you do a similar thing with text data, okay? Only we call it basic feature extraction. And this includes things such as, you know, calculating word count, average word length, and so on. And just like descriptive, statistical analysis and standard statistical analysis, the point of this is to just to give you a feel for the data and what you're working with.

So, first we want to do is look at what the word counts. Okay? So, we're going to look at position field, and for each row, we're going to count the number of words contained within it. Okay? Now, it's important to note that each of the cell, each of the cells in this variable, okay, let's say that one, all the content is saved as one specific string, okay, so this is the first column and that's saved as sort of the first row not saved as one string. This will be saved as this as an also saved as one string, and so on and so forth. So in order to count the words here, we're going to use two of pythons inbuilt functions in tandem. We're going to use split and the length function. Okay? Now, if we were to apply this to one string, we would write this bit of code, that you can see right here, okay, where x would be the string we're feeding in, okay, so think of x as a string of feeding. Okay, so for first time, we're feeding in, we're feeding in this text here, professor of anthropology, sociology, and so on. Okay. And what this command says here is take x, split it wherever there's a whitespace. Okay, so wherever there's a, there's got one string, now we can include each of the words and the whitespace. This command says, split this string into separate strings, whether there's a whitespace. So now Professor becomes its own string, anthropology becomes its own string, whatever the other words are, all each become their own string. Okay, we then feed that into this length command, which then gives us the number of individual strings contained in this new element we've created, which is in the form of a list. Okay, that's all that does. However, because we are working with a data frame, we obviously don't want to have to type this bit code to do this to each of the individual rows in our data frame, because then we'd have to do 118 times and that requires too much effort.

So, what we do we use the lambda function. And the lambda function works in a similar way to what functions do that we looked at in the first video, only, it's a way of easily applying, similar to be functions in the sense that they're a way of getting a piece of code to run on each of individual elements in a sequence without having to do it manually. Okay. So that's what lambda x does here, it says take that variable and do this for each of the variables in whatever sequence we feed in. Okay, so that's all that it says there.

What this bit says is that we're going to take our data frame, and we're going to create a new column. And this column is going to be called word counts, and it's going to be equal to the position column. Okay, but for each row in the position column, we're going to apply this function, which is going to be the one that splits it splits the string and counts it. Okay. That's all that says, and if we run that, you will see. And, yeah, we then assigned out to this new variable name, and we then asked Python to print out this specific variable and we see here we've got we've got our new word count variable, okay? Likewise, if I ask it to print out the whole data frame, df, okay, you can see here that we've gone from four columns to five columns, because we've got this extra column here. And you can see here we've got the position, which is out, which was the last column in our original data frame. We've got this nice new column added on to the end, which counts the words. And we can see here that perhaps not a surprise might be in the head of department has the most to talk about in his job titles. He's got 39 words, and then I've got a lowly four, because I am not that important.

Okay, so then we want to do sort of an average word length. Okay. So you might see, you know, looking at some of these words, you'll see things like professor and anthropology are quite long, things like data, and of and are quite short. Although, in this trivial example, you perhaps won't need average which word length when you start to analyse text data, so things such as Twitter posts, or blog posts, for example, average word, word length can actually tell you a lot about how perhaps how research the post it.

So, for example, blog posts that have a large average word length, tend to indicate that these posts are using a large amount of scientific terms. And so that's what gives you an initial insight into the nature of the text. And, for example, you can kind of see what I mean when I say these kinds of steps are like the descriptive stats steps we see in normal traditional statistics.

Now, in order to calculate average word length, we're going to build a little function. And hopefully you remember these functions from the first video. If you remember from that, we build their function by typing def. And we're not going to always function out which word and save it, I'm going to feed in mind, it's going to be a sentence. Okay? I've been saying my words are the sentence, and I call the split function. This is the same split function we called a moment ago. So that takes a string and divides a string into individual words for me. And if there are more than zero words in the list, because some people have nothing in a position in the job description name, if it's more than zero, then give me the sum of the length of each word in this list divided by the total number of words. Okay? So again, just calculating the average of it. And the way we call this and apply this to of each of the rows in our data frame, is exactly the same. So, we did a moment ago, we just specify a new column name called AV word. And we again, we say we're going to apply this new function, average words on to the position variable. If you run that, just like we did before, you can see we've got our new average words. So, we can see that Mike, who we know is the first one there, has an average word length of 7.7. in his job title, I have an average of 5.5, perhaps in which perhaps shows off how lazy I am at writing.

Okay, now, the next thing we want to do is remove stop words. Okay? When you come to do text analysis on a large text corpus, you're going to want to remove stop words. And the reason is twofold. One, sometimes they're not analytically interesting. Okay, so, okay, so you know, you're interested in us of, and and, and so on. Secondly, when you come to supervise models, i.e., training algorithms in order to calculate those aspects and highlight certain features of text data, including words, including the stop words, words, such as and the and so on, can bias the algorithm and cause the algorithm to focus on the stop words, instead of more analytically interesting words. Okay. So, we just want to remove them. Okay. And there's loads of ways in which you can do that.

What we're going to use here is NLTK stop word function. So, if you look over the top here, one of the things we imported was from the NLTK package, which is the natural language toolkit package in Python, we imported the stop words aspect of it. Okay, now, the NLTK package is a huge package, so you can't import the whole thing in one go. And even if you did, it would slow puckered up quite a bit. So, we import the specific functions we want. So, we've imported the stop words function here. So, we just type stop words and it calls this package from the NLTK packet. So, if we scroll down here, and so like, again, you've ever called stop and stop is all the stuff all the English stop words from this NLTK stop word dot words function. Okay? This basic says go to the stopwords package, go to the words lis and finally the English words, okay. And this produces some, a whole list of stop words that we've seen in English language so, and the of things like that.

I then we then do again, this lambda function exactly like we've done twice before. And this time we say for each of the words, in our words list, if remove the x, i.e. remove the word if the word appears in this list, as all this says here, okay? Here, x refers to each of the specific words in a string, okay, so we split our string. And for each of the words, in the string, it appears in the stop list, okay, count it, okay, count it for us. Okay. And you can see here, again, Mike has like 10, stop words, that's 10, and the and so on, I have one, that's because my job title is, lecturer in data analysis is a stop word. So that's the one stop word with can't get there. Okay. And obviously, you can also do things such as remove these from the text if you still wanted to, but for the time being, I'm just showing you how to access them.

On special character count. So this isn't so much of an issue here, because obviously, job titles don't have many special characters in. If you analyse it, internet data, such as Twitter data, or so on moment to count the number of special characters, okay, and you can do that really easily with Python. Okay, you start off by creating a dictionary. Okay? Hopefully by now you've got comfortable to seeing dictionaries. I'm going to create a dictionary of just two key word pairs here are key value pairs here just as an example. So, we have our first key, which is hashtags, and it's a hashtag. And our second key was Twitter name with a value of the at symbol. And the reason I put this here is because if you remember, if you know Twitter, talking about username, start off with an at symbol. Okay. I then say, we will cycle through each of the key value pairs in this dictionary, as this line says here, so for each entry in the dictionary. And for each entry, create a new data frame with the name of the key, okay, and apply to the position column. This function and this function again, says, you know, split the position column into individual. So, each word is an individual list, just like we've done before. But if each of the each of the words in that list, can't it if it starts with one of these two, okay, so now let you know whether or not you're counting whether or not a word is a hashtag, or username, for example. Okay, if you ask it to print, you see, we've got that there. Okay. And again, if I print the data frame, okay, you can see scroll down here, you can see our new columns have been added on and we've got one for each of the special characters. And like I said, it's not a surprise that we don't have any of those in here.

Okay, so uppercase count. Sometimes, it can be useful to count the number of words that feature in a specific case, okay, so if you think about people talking on the internet, okay, YouTube comments are so on. Often people type in all uppercase in order to emphasise anger or something like that. So again, it can be it's a very simple scripted count, but can actually be quite useful. And fortunately, the NLTK package, which we've employed at its core function called is up, which basically just counts if a word is in uppercase, all in uppercase. And we apply this to each of individual words, just like we've done in all the other functions so far in this session and create a new variable for it and run that. You can see here Mike has no upper-case characters. My colleague Alexei has one, Verb is here is none, I have none. Okay. Now at this point, it's worth remembering that we've created a lot of columns into our data frame. So we might just want to just run some counts on the data frame to make sure we know what we're working with.

The dot shape function tells us the shape of the data frame. And you can see here we still have 118 leads Those 118 profiles that we were doing mean basic analysis we've done above, we've gone from four columns, 1to ten columns. And we might go, Okay, what are these new columns are creating again, because you know, as computer scientists are very short memory. So, we print off this list df again, and you can see the names of all our column names. Okay, so these are the four that we generated from scraping the data in the in video two, and then these other ones we've created in this video here, which just add descriptive stats, okay. And if we just wanted to see what this looks like in the data, we can print off the DF dot heads. And that just gives us the first five rows of data in our data set. Okay. So at this point, we go, okay, yep, we skipped steps, but good, okay, they look fine. And now we feel with that start to feel comfortable now that we've got a sort of feel for the data, we know what kind of data we're working with, we know how many average words we see the kind of stop words with special characters, and we've kind of got a feel for it.

Okay. So now we need to enter this sort of what we call data preprocessing. And again, to draw an analogy with traditional statistical analysis, this is very similar to cleaning your data, okay. And those data analysts among you will know that cleaning your data is one of the most important steps in any data analysis project. And is exactly the same here. If you don't get this step, right, you can completely mess up your analysis. Okay. Fortunately, there's some, there's a lot of good guides out there, and I'll link to some of these in the materials in the session. But if you follow the steps, we're going to follow through now, nine times out of 10, you'll be fine, with your data.

So the first thing we want to do is convert all of our text to lowercase. Okay, and this is because that, if you remember way back in the first video from today, I said that when quoting compares strings, it does so by looking for an exact match. Okay. And so, if one version of a text in capitals and the other one is a lowercase python won't text match because technically they are different strings, okay? So, in order to avoid that, and to avoid jeopardising our future analysis, we want to break this down into, we want to convert each of these words in the position, into a lowercase, okay. And to do that, we just use the dot lower command, okay. And you'll see this command looks exactly the same as, as the commands and sorry as the commands above it. Okay, and that all we're doing here, but we've kind of done two things. First of all, actually, this, a bit more, this is in two steps. So, this bit here is very similar to the commands above. Okay, now, we use the split command to break our sentence string down into individual strings where each string is a word, we then use the dot lower command to make sure each of those words are converted into lowercase. We then feed all of that into this join command and say separate each of these words by space in order to essentially reconstruct the sentence back to how it was, but with all lowercase characters, okay. And then the lambda, lambda works exactly the same ways as before, and that that just enables us to apply to every row in the data frame. And if we run that, and I'll just see the first five records, because you can see we've covered everything now to lowercase. So even the professor's the seniors, the lectures, all these letters first, as would often be capitals, as well, they've all been converted now to lowercase. See that there?

Okay, the next thing you want to do is remove those stop words we spoke about before, okay. And this is for exactly the same reason I mentioned a moment ago and that they can really sort of bias and skew your analysis. And you're going to want to remove punctuation as well, because that can also cause havoc, so it's best just remove it. Now fortunately, NLTK package makes it incredibly easy for us to do. So. However, I want to make sure that my stop words fully comprehensive. Okay, so what I do here, I create a new variable NLTK list and that is the stop words list from the NLTK package being issued stop words list that we saw a moment ago.

Okay, I also there's another category we put in earlier on, word cloud. And that also has an inbuilt stop word list, which will also go into you. And I, which I'm also going to call upon which is this here by saying instead of storing it in JSON format, which is stored in stored in a list format for me, the same as this one, I then say, punctuation list, which is piping when a pipe is inbuilt punctuation lists are convert that into a list as well. Okay? I then convert, and then combine these three lists into one big list. However, because I'm analysing this data, I suddenly become aware that there are some words that could appear my analysis that I don't, that I want to move, but are unlikely to appear in standard stock word lists. So, things such as HTTP, HTTPS for the URLs, those are unlikely to be in the stop word list, but are likely to be in my data file. And I want to remove those. Same with the word website. And just some examples of punctuation here. So, I create a fourth list, which is my own custom list. And I also add that to this list, and this then combines these four lists together to give me one big list of stop words I want to remove.

Okay. Now that we've got our, then once we've got our stop word list, we can then do a similar function that we did before. Okay, so what does this do? So firstly, we call our split command, which does the same thing again, or taking a sentence string, and breaking down into individual strings with each strings and individual words, we then say that we keep the word, if that word is not in the stop word list.

We then feed that all into this joint command, which then combines each of individual strings back together, but with participates between them. Okay. And again, the lambda works exactly the same way as before. And if we print that off, you can see now that the stop words have been removed. So in my job title, for example, I'm lecturer in data analysis in has been removed. My cutting is Professor of Anthropology and so on, of has been removed. Okay to remove the stock lists the stop words. Very nice and very simple.

Now, that leads us to tokenization. You'll hear a lot about tokenization as you start to read more and more about textual analysis, that is essentially the process whereby we convert text data into a sequence of words or sentences. Okay. Now, for this example, we're going to again, call upon the NLTK package, which is you, if you've not guessed by now is a package, you're going to use a lot when it comes to text, textual data analysis. Okay. Now, the NLTK package has a function called work, word underscore tokenize. Okay. And we can use this function in order to convert strings into tokens, ie individual words, okay. Now, in order to make this work, we'll have to loop through each of the rows in our data frame. And we'll then save the output of each row to a list.

Okay. So, if we run this, okay, you'll see all this is done here is essentially broken down each of the words in our column into separate individual token, words, or tokens we can work with, okay? Now, in a basic level, this might look a lot like the split command we used a moment ago. But it's a little bit different in the fact that here, it's not treating each one as an individual string. It's separate them as tokens, which is important for when you can't play around with some of the other NLTK functions. And you'll see why now when we start to look at this process of lemmatization. Okay, I can't pronounce it and be different people pronounce his word differently anyway. But essentially, it's essentially a way of stripping these various topics from words in order to break them down to that core word. And this is important to do. So say for example, you have, Yeah, so things like going to be broken down to go and so on. Okay, which again is important for analysis because although they're different words, they have the same meaning. And for analysis, we may need to know what this meaning it is okay. In, as I say here on the slide, right? In more fancy terms, it makes sense to make use of vocab. It does sort of morphing morphological analysis to obtain the root words, okay. And because of this lemmatization is more commonly used, that it's sort of brother/sister method, which is known as stemming, okay, stemming interest trumps the end off, which is fine until you encounter words such as caring, where we just chop the suffix off of that you get car, which is a very different, which has a very different meaning.

Okay. lemmatization, however, uses the morphological meaning of the word to try and remove the suffix while maintaining the meaning of the word. Okay. Now, we're going to run this operation on both the tokens column and our position column. And in both cases, we're going to overwrite the original column. Okay, you don't have to if you wanted to create a new column called, you know, tokenize words, or, sorry, lemmatize, token words, whatever. But although we have two slightly different function for the two, because of how they're laid out, remember, once token format ones in normal string format, the process of calling dot level ties is exactly the same. Okay? So here we call the NLTK function word with a capital W, parentheses, and we feed in each of the individual words in turn, as created by the lambda function for x, and then called dot lemmatize on it. And if we were to print off, and it's going to take a second to run and reprint the off here, do you see any examples here? We don't see many examples immediately because of the nature of the data set. But if there were limits, if there were words, such as you know, teaching, for example, in any of these job titles, they'll just now be teach. Okay, So that brings us to the end of text preprocessing, there are some other methods you might have to use as you use more complicated and advanced data sets. But that's the sort of basic fundamentals of Sphinx fundamentals or preprocessing aspects that you will have to do.

Now, that brings us to text processing, it will be actually starts here, we're actually going to start to do some actual textual analysis. And the first analysis method that's quite useful or what we call bigrams or engrams. Okay, so these are really useful way to evaluate which, if which words occur together, so if you using bigrams, you're looking at which two words appear a lot. So, in our data set, we you know, you might expect to see senior lecturer occur a lot, maybe quite common bigram. trigrams combination of three words, engrams, you know, if you look into combinations of 456, whatever words, okay. And there's a basic principle here that you can sort of see on the slides and that principle is that engrams capture language structure. Okay, i.e., what letter or word is most likely to follow? No one into consideration. Okay, it's worth noting that the higher the N in engrams, so quite often the lessons have what tends to be equally sometimes bigrams aren't that useful, it's all about finding the sort of sweet n. And that takes, there's no scientific way to work out what that is, I'm afraid it's just sort of involves a little bit of play just to see what works. Fortunately, thanks to NLTK, calculating engrams and text data is incredibly easy. So, this code looks like a lot here, but we're doing a lot with this code. Okay.

So, first things first here, we're just this BGS thing here is how we calculate the bigrams. Okay. So, we feed in our to, what we're saying is that for each row in the data frame, is for each row i.e. each profile in the data frame. Okay. We're going to go to the token’s column, and all the tokens in that cell we're going to work out. We're going to calculate each combination of two words. Okay, we're then going to calculate the frequency at which each of these two words occur. And then we're going to create a new one, which will be saved as dictionary, we're going to create a new, a new data frame for each individual profile, which holds the engrams for each one. Okay?

Actually, just as a teaching example, we because if I run this, now it's going to run on all 118 records. And it's going to take, it's going to take a while to run. So, this code here, as it stands, is to do it, it'll run these operations at every single row. In the data frame. I'm just going to make sure they work operates on the first row, it just, Mike's account, Mike's profile, just so you know, we were just looking at one example what we learned, but bear in mind if you left this code as it was everyone and each of the individual profiles, okay, that's kind of been that. Okay, and so this okay. So, it this calculates the end. This calculates the engram the bigrams, because we're doing two this slide here, called the pandas function, data frame from dic up, which is just a pandas function for taking the dictionary and converts it, convert it into a data frame. Okay.

This line here, okay, organises our data frame by the highest value, okay, organise it from the highest value to the lowest value. And when I say value, here, I'm talking about the value in the key value pairs, okay? It just happens in this case, the value has happened to be the frequency of which they occur. So this creates a list whereby the most frequently occur bigrams at the top going down to these frequently occurring, okay? This just name this, this here gets the name of the profiles, in this case, this will be Mike's name. And we just use that name to to create a string that we can name this, the new data file, we're going to click for this specific account, we then export this as a CSV file, just like we did before with the json folder at the end of video two. And once all that's done, then we go to our folder, you'll see we have a new file here. You can see this file name has. It's called engrams underscore, frick underscore, short for frequency. And its a plus profile name because it's taken Michaels name, and then dot CSV above to add on at the end. If we open that, you can see it saved bigrams break down on a file for us. And like I said, if you left this as it was, it would run through all of this for every single profile. So, you would have 118 of these files, one for each individual profile. Okay.

Here I just asked it to give me the 20 most frequently occurring bigrams. Okay, we're using very small data sets. So, it's not an issue. But if you're dealing with datasets that have you know, like a blog, if you scraped a blog website, for example, that has 300,000 posts, you might only want the top X number bigrams, just to sort of whittle it down a little bit so it makes it easier to interpret. Okay. All this here then, is just me visualising the network, okay? You quite often don't need to do this, but I just thought I'd do this to shoot sort of as a teaching example. This goes pretty self-explanatory, but it's just using if you google python, pythons network x package, you'll see this code here is the basic default code you see on the website to activate and all this does is take each of the bigrams and work it work out which word is a bigram of which and includes a network of it, and then create a network of it and then plot that network as an image to create this. Okay, so all that's done is it's worked out right. That the words biomedical, biotechnology, biomedical work biotechnological, are a bigram, it's also really the other biotechnological and technoscience are also bigram so it's combined. It's creating this network by combining the two biotechniques biotechnological nodes together and connect it to biomedical and techno science. Equally, it's worked out the biomedical is a battleground with innovation, and that engage us public and innovation also bigram connected on innovation again, and it's not hard for each of the bigrams to create this network of cocoa in words, okay. And like I said, this, this here, if you google python network x, this is a default code to create a network image. It's nothing special, I literally copied and pasted. And this headline here saved the network.

So, if you look at, go back to your directory, work, you'll see what that network image has been saved for you. Okay. And again, like I said, this is being run on. If you didn't alter this code, this would be run every single profile, so you'd have 118 of these images. Okay, hope you see now why I only work on the first one for teaching purposes.

Speaking of which, I'm going to do the same for this example otherwise we're going to be all day while this run. Now, what I'm going to do here is just create a heat map of terms. Okay, so this is again, just taken out bigrams exactly the same as before taking the most frequent ones. And all it's doing is creating a co-occurrence matrix. So, for those of you who do statistical analysis, and they've done a correlation test before, you know, you play heat table correlations, this is exactly the same process as that. And enables you to see which words occur together. Some people for heat maps, I prefer the sort of network breakdown, I find that a little bit more useful, because it sort of shows which words co-occur, which, but obviously, this shows the frequency. So, we see here that technology, more frequency occurs of sites and so on. Okay. It's not often you create heat maps; I'm just going to breeze pass that a little bit.

Term frequency is a very useful piece of analysis. Okay. So, term, term frequency, as I said, it's merely the ratio that a specific word occurs as a function of the length of the sentence, i.e., how many times does that word appear in the document as a function of the overall number of words in the documents, okay. And so the code for calculating it is really easy. So, we clean first, the for loop. And this again, just goes to our entire each goes to each row in our data frame. Okay, I'm only going to do it on the first room, save us some processing time, while we're teaching says for each row in our data frame, create a dictionary called term frequency dictionary, which is empty, it then goes to our tokens list, and it gets a list set of it. Okay.

In Python function sets will create, take a sequence, and it will create a new sequence which only, which only contains one occurrence of each element in that sequence. So, let's say you have a sentence that says, you know, the cat sat on the cat. If you figure out the set argument, it will produce a list that says the cat sat on the because cat already in there, and that only contains one it one, the codes of each element, and a piece to add to the list. Okay. We then say for each token, in this term list, count the number of times out token appears in token cell for this row, okay? And then divide that count and then divide that count the amount of times it occurs by the number of words in that cell. Okay. Here, I've asked it to count it, as it causes the number of unique terms. So here, it's going to count the frequency of it at an old sentence. So, the cat sat on the cat. Equally here, I've asked it to count I wasn't coming up with time has been as a unique number times a person's unique element, which as I say out loud, can't remember exactly why there's not really that useful, this is the best one to go for. Anyway, even sees these as a list and feeds into it sees into a dictionary. Okay, I then create a new data frame of this, again, sort the values. So here, it just creates a data frame that then takes the values and just reorders the data frame so that those words it most frequently occurs at the top of the data frame. And there's at least frequently at the bottom, I do exactly the same thing we did before in order to get the name of the profile to feed into the file name, and then export it as a CSV file.

Now, if I run that, and then again, go to folder, you'll see that we've got term frequency here. And you can see, it calculates the ratios of the amount of times those words appear, because like I said, again, if you didn't change a split here, you'd have 118 of those files, one for each profile. I'm just gonna show. Just before we get on to that concluding section, I'm just going to show you how to create a word cloud. Now, it's worth noting that if you do actual textual analysis for research people, so one, don't use a word cloud, because they're literally not that interesting. However, they are quite popular in certain circles or for like, PR advertising of your research papers, and so on, because they're, like catchy. So, it's worth knowing how to create one, and especially since helps, it's incredibly simple. Through that, we just use the word cloud package, which we imported right at the very start. And we call the word cloud function. And if you google the word cloud Python package and the word cloud function, you'll see that you've got all these different variables here. And these just adjust the image, okay, there's how big you want the maximum words up font size to be. So obviously, different words are different sizes and word clouds dependent how frequently they occur. So, this just says how big you want the maximum one to be. And I'll just set it to 50, how wide you want the image to be, how tall do you want it to be? What background colour, you want, stuff like that. And then you just picked up generate, and do that from this row, from this cell here, this column, this row to this specific cell here. And again, all these is just to adjust the finger size, length, colour. And to save the file on again, just only going to run this on the first profile, we've done that. And you see it produces our word cloud for Mike's profile. Okay? Likewise, again, if you left this here, you get 118 of these one for each of the different profiles. So, for example, if I was to put let's do my supply put for you see that I get a very small word cloud because my job title, I've only got four words in it, one of which is in which has been removed. Okay. In short, Mike has a much interest, much more interesting job title than I have.

Okay. So that brings us to add final piece for today, where we're going to look at some sentiment analysis. Now, it's worth noting that sentiment analysis shows two here, it's a very, very, very basic form of sentiment analysis. And it's the kind of sentiment analysis should do just to start off with just to get a feel for what you're looking at. Okay, so we're still very much in that descriptive realm of this. And the reason for that is because there's a whole host of different sentiment analysis that that you can use using either unsupervised or semi supervised approaches. And we'll need a lot more than the three hours we've been allocated today in order to go through that. So, I'm just going to show you how to do a very basic analysis using text blob package in Python.

Okay. So, what does this do? Okay, well, here, we're going to be using the textbook packages, do a quick piece of analysis, okay. And this code here, basically returns a tuple for each of our rows, okay. And these two, and this tuple contains two numbers. The first number represents the polarity, and the second number represents the subjectivity of the whatever document you're feeding into it. Okay. Now, for the time being, at this point in our analysis, the only number we're actually interested in is the polarity. And this is what indicates his sentiments, okay, the closer these polarity numbers of minus one, the more negative the document is, likewise, the closer it is to one, the more positive it is, and obviously, close it. And obviously, if it's zero, then it's sort of neutral. Okay. And I'm only showing you this now, because this sort of forms the basis of some of the more complicated machine learning based sentiment analysis that come later. So, but even that turned off and you see our polarity here. And it's perhaps no surprise that most of these numbers are very close to zero. Because you know, these are job titles. So, it's hard to have emotion in a job title. And here, we're just going to add this to our data frame. And, yeah, we just create a new data frame of it, which makes me pretty, the reason I've showed you how to do this is because this is how you perhaps want to represent this in a paper, if you're going to do sentiment analysis in there. And you can see your text there and sentiment there. Okay.

Okay, so that brings us to the end of this session, I apologise It is very, very rushed. But um, obviously, we've had to cover a lot in the three hours we've had. Yeah, but the whole purpose of this session was to provide you someone who's never done any text analysis before, with a tool to start doing some very, very basic stuff. So, we've covered in that brief introduction to Python, how to build a very basic web scraper and how to do some very basic descriptive level text analysis is a lot to build on from here I'd recommend, like I said before, in the last video, I'd recommend looking at the links I've provided in the first video that take you to that happening here. They'll take you to the content. If you look at the first video and the first Jupyter Notebook that contains links, they'll take you to a page that has a lot more. No more tutorials in. I will open that up here. If you go to this address here, you'll see that there's other introductions to python that I've recorded, which are much more in depth than the one you got in the first session. And there's also some text analysis stuff here, which is more advanced than what we've had chance to cover here. Yeah, this is this is been intended to serve as a very brief introduction, and hopefully you found it useful. If you have any questions, please feel free to contact me via email. I'm always happy to help as and when I can. And yeah, hopefully you found it useful. Thanks a lot. and have a good day.