COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Week 10
Video Transcripts

Video 1 (5:34): Text Mining Setup

We're going to work with the natural language toolkit, which is the basic library for text mining on Python. Text mining involves looking at pieces of text and trying to make sense out of them, and we see what kind of sense we can make out of text—pieces of text. Keeping in mind that text mining is not perfect, so the idea is, you try to extract whatever information you can from various documents, and maybe decide which particular documents you want a human to read or examine later, if you want to do that or if you don't, then that's fine too because you can get credible information out of a text thing itself. But, before we start, we need to set up our environment because there's some basic setup stuff to do here. So, the first thing we're going to do is to—let me restart the kernel here, is to download the—so, the 'nltk' library comes with a large number of preinstalled data sets, so we're going to use one of those data sets, so we need to install that stuff here.
So, to do that we run this 'nltk' download. And, when you run it, it's actually going to run behind your screen. So, as you see here at the back of the screen, you're going to get this window at the back here, which is this one here, and I make this bigger here. And, it has a whole bunch of things there, and all you want to do is to click the download button on this. Okay, this will take a little while to download, and so you probably want to pause the video here. And, once it's downloaded, I already have it downloaded, once it' downloaded it's going to say, "Finish downloading collection all", okay! So, that means that it's completed. And, what you can do then is you can go to file here and click exit, and then go back to your thing here, and you should be all ready. It should say true over here, and it should be all ready to move forward. So, once you have that downloaded, you might need to do one other thing, so let's do that while we're at it.
So, you should have downloaded the files that are required for this class—today's session, and from those files you want to essentially take a bunch of data, if you go into this here, if you go into this folder called data. So, the files that you downloaded would be data, text mining, and yelp, well you won't have those, data and text mining. And, for the...for the data, the data part that you downloaded, there are...there are two files over there called 2013-Obama.txt and 2017-Trump.txt. So, these two files you need to move from this location into the location where we downloaded all the 'nltk' download stuff. And that, depending on your machine, could be in different locations but most likely it's going to be in your hard drive. So, if I look at my thing over here, for example, I can see that—I don't have that here. You know, on a Mac you can find your hard drive by going to finder, preferences, and then clicking—going to advanced I guess, no.
Sidebar, sorry, go to sidebar, and click hard drive—hard discs over here, and that will produce the hard discs on the side. So, as you can see, it says Macintosh hard disc, and go to users, go to whatever the user is, and your user name whatever that is. And, if you look over there, there should be now a directory called 'nltk_data'. On Windows, most likely it is in your home directory if you look for that, or

in your Windows explorer, you can search for 'nltk_data' and look for it accordingly. Once you go to that nltk_data, you want to go down here and go to 'copora'. 'Corpora' are the texts that are required by the—provided by 'nltk' for analysis, and we are going to be going into one 'corpora' here, which is called 'inaugural'. The 'inaugural' corpora contains all the speeches by US presidents when they were inaugurated in—for the inauguration ceremonies.
Unfortunately, it stops with 2009, Obama.txt, so I added to this. I downloaded 2013-Obama.txt and 2017-Trump.txt, just for completeness, you know, we want to be using data that's recent and copy those and paste them into inaugural. Oops! That's it, okay! So, with this, we should be ready to go. And, you should keep this data directory in the same location as wherever your notebook is because we're going to refer to these other files, amigos, community, and all that kind of stuff, when we are working on our data. So, that's it, if you've done all that we should be ready to go and ready to move on to our thing here.

Video 2 (7:24): Sentiment Analysis Part 1

Now, that you are ready with your data properly set up, we can start working with text itself. So, let's see what we can do with it. The first thing, and probably the easiest thing to do with text, is to figure out the sentiment of piece of text. Sentiment means is it saying something positive. Is it saying something negative? Is it…it going to inspire? Is it surprise or fear or something like that? And, that's the idea in sentiment analysis and, you know, it's, you know, it's not a simple subject because if you look at our examples over here that I have, you can say something like I had an 'excellent soufflé' at the restaurant 'Cavity Maker', right! So, well, this is the thing, the word 'excellent soufflé' is telling us that we-that I really liked this. It's a very positive statement about the restaurant because 'excellent soufflé' at that place. 'Excellent' is a positive word, and it...it implies that we like the 'soufflé', and we like the restaurant.
It's positive about both. But then, there are certain case, and English language being a funny language, there are certain cases where, things in fact all languages, I guess, but it's not always so clear. For example, if I had a statement that said, 'The on the Train is an excellent book for a stuck at home snow day.' Now that's a positive word, right, but it says, 'The Girl on the Train, an excellent book for a stuck at home snow day.' You're stuck at home. You can't go anywhere. It's like piles of snow outside. So, what do you do? You curl up with the girl on the train and, you know, that sounds like a good way to spend your time. So, that's a positive statement. However, if I say, 'The Girl on the trail—Train' is an excellent book for using as a liner for your cat's litter box.' But clearly, if this comes up in a review, then that—it would indicate that the reviewed...reviewer did not like the girl on the train, right, because they want to use it for something where a book is not intended to be used for. Just to be clear, I actually enjoyed the book a lot.
So, it's not an...it's not something that I would say the book is not nice; it's actually a great book, but this is just an example, right! So, this is in view, you know anyone reading this would understand that the reviewer did not like the book, but for a computer to figure this out is going to be really hard, right! because how does it know that 'liner for your litter, cat's litter box' is not the normal use for a book? It's

not, you know, it's a negative connotation for us, but for a computer program, it's really hard. Then, there is a third case here, 'The Girl on the Train is better than Gone Girl.' Now, these are two books, and we are saying this book is better than that. But what you could have disliked gone—the reviewer, for example, because disliked 'Gone Girl' a lot and decided he didn't really like the book at all. You know, I hate 'Girl on the Train', but it actually, but it's better than 'Gone Girl.' It's not as bad as that, which is not necessarily a positive statement.

All we can say is that this is in the eyes of the viewer or the document is better than this, but we can't say whether it's a good or bad or assign a more general sentiment to that. So, when you're working with language…you should bear these things with various things in mind that in general the English language is not always clear as to whether something is being pos...is being stated positively or negatively. And sometimes, if you have a comparator, you should just use it only for comparison. So if—we can say 'The Girl on the Train is better than Gone Girl', but we can't really say anything independently about it. That said, that's a caveat we should keep in mind. But that said, when you, a lot of text mining is done on technical official documents, which are not going to be using, you know, sarcasm in this form, right! So, they're not going to be saying stuff like this. So in—if...if, for example, you're reading analyst reports for stocks, then unlikely to be saying things in this format. They're more likely to be very clearly saying that there's positive growth, or growth is on target, you know, things like that.

So, it's...it's with official documents or technical documents or documents that are—have content and are not fiction and are not reviews or things like that. The odds are, not necessarily, of course, and this can be as ironic or sarcastic as anybody else, the odds are that your data is going to look pretty good. But the bottom line on this is that ‘sentiment analysis’ is a good starting point, but you know bear in mind that it's not always going to work very well. 'Sentiment analysis is usually done using a corpus of positive and negative words.' So, this is another tricky aspect of 'sentiment analysis.' So, out there on the Internet, you're going to find lots of collections of words that are positive and negative, and you...you can use those, but of course if you have a very well-defined domain that you are doing 'sentiment analysis' in, you might be better off in constructing your own positive and negative words because in your domain, it may be a different set of words that are applicable.

So, that's the idea there, and there are several different sets of analysis, and I've listed a few of them here. There are many, many more. There are, these are just sets of positive and negative words. So, they're 'Hu and Liu sentiment analysis lexicon'. There are 'NRC Emotion Lexicon' which I have included actually in my data set that I'm giving you guys. But the lexicon itself is no longer available without permission from on the Internet. So, bear in mind that this, the use of this lexicon is solely for this class. If you want to use it for anything else, you should probably get permission for it. There is a 'SentiWord' list which is 'list of words' 'weighted by positive and negative sentiment', and we're not going to use this one, but it says like, you know, for example, it says excellent is, you know, highly positive; so, it has like a zero point seven reading, whereas good is maybe zero point five or zero point four, that kind of thing. So, positive, negative words are just words. So, there's no polarity attached to them. These ones have some polarity, and then there's 'Vadar sentiment tool', which has '7,800 words with positive or negative polarity'.

This is becomes a part of Python nltk that you have to download something for it, we'll do that download...along the way. And the thing about the other sentiment, which we will use here, and again,

not perhaps in the context it's meant to be used, but it's really being constructed using email and chats and, you know, informal text. So, it contains things on emoticons, emojis, and those kinds of things are also included in it. So, it's really good if you're working with Twitter feeds or Facebook posts or things like that, and you want to do some sentiment analysis on those, then 'Vadar Sentiment' actually works very well or should work very well; nothing is certain.
For the most of our class, we're going to use these two examples. I've 'compiled a set of '15' reviews of four neighborhood restaurants,' and neighborhood being the Columbia University neighborhood and these are just taken off from yelp, so they're just like general reviews. I'm also going to use the yelp API, so as a sort of side benefit, we're going to see how to use the yelp API, which is kind of nice. And then, I have the 'Presidential Inaugural Addresses' which are included in the Corpus, and I've added, like we just did in the first video, added Obama and Trump, Obama 2013 and Trump 2017, to that corpus. And that's about it really. So, these three examples we're going to use.

Video 3 (11:43): Sentiment Analysis Part 2

Let's start by computing the proportion of positive and negative words in a piece of text. This should be fairly simple kind of 'sentiment analysis'. We look at the text and we add up—we count the number of positive words, we count the number of negative words. And we decide whether something is positive—the text itself, the document itself, is positive of negative, depending upon whether they're more positive or more negative words. Very naive but, you know, it's a simple way to do it. So, we can write a function here called, 'get_words', which is actually going to get the words from a URL, right! So, the two URL's that we're looking at are the positive words dot txt here, this one, and the negative words dot txt, and we're going to get these words here. So, this is a very straightforward function, all it does is it takes a request library, which we've used before, and get user request dot get. And then decodes it from Latin one text to get the actual list of words. And, the list...the...the URL contains the words listed one after the other you can take a look at it here if you like. So, this is it. It's got some junk on top, which is all—it's start with semicolons. And then, it has—this is the positive word. So, it contains all the words it thinks are positive. Okay! Lots of words, there are...there are here but they are without polarities. And similarly, for the negative words.
So, all this function does is, it takes—goes—gets the data, removes any line that starts with a semicolon, which is a comment line and then, it takes every other line which contains one word and adds it to this list of words, and returns a list of words. So, we get a list of positive words and a list of negative words. That's what we get here. So, we run this and we can do length, positive words, '2006' positive words, and length. Oops! There are '4783' negative...negative words, pretty much over weighted...over weighted on negative words. So now, all we need to do is to read out texts and count the number of words. So, I have sample text here. This is not the entire data, but it just contains a couple of reviews of community, which is a restaurant in Manhattan, in—near Colombia University called, community food and juice, and Le Monde, which is another restaurant. They are actually right next to each other on Broadway. And, I'm going to read that data.

So, all I need to do is open the file and then, read it. Okay! So that's it. So, I read these two and I get community and Le Monde. And then, I want to compute the sentiment! So, the sentiment is, again, straightforward. In this case, I just want to count the number of words. So, I go through text—a piece of text here, community, for example, and if—take a word, if it is in positive words, I increment the counter by one. So, 'cpos', I started at zero and increment the counter by one. It is a negative word, I increment that counter by one. This way I take care of the possibility that a word is both positive as well as negative. I don't really know, I haven't looked at it in detail, but it's possible that there's a word that fits in both columns. So…okay! In that case, I take negative and add one. And, the same thing I do for this thing here. The only thing I'm doing here adding to this is from the 'nltk' library, I am— package on python, I am importing the function 'word_tokenize'. What 'word_tokenize' does is, it takes a word and makes it into a single token.
So, it will remove any punctuation, like for example, if you have a word that, you know, a sentence that ends with, let's say, it was—This was a delightful experience, period. So, experience period, if you're looking separating words by just spaces, experience period will become a word. But, it's not really a word. So, word tokenize will take care of the period for us. So, it will hopefully anyway. So, you do that. And then, once I've got that then, I can compute the proportion of positive words to the total number of words in the thing. This is the length of the number of words in the community reviews, community full industry reviews, and I compute the proportion of that and the proportion of negative words, and we end up with this thing over that says '5.09 percent' of the words used in community were positive and '5.33 percent' were positive in Le Monde. So, community in a sense has slightly lower positive words proportion to the total text. Because, that's the way you want to look at it. Or, the text has smaller, larger, is you know to compare them when we compare apples and oranges.
So, people are more positive and more negative about Le Monde in their reviews. So, it inspires slightly higher feelings, I guess, and less positive and less negative about community. But, on the average, if we like, take the difference between positive and negative words, community comes out a little bit ahead of Le Monde on this very naive analysis. So, that's the first thing we can do. The second thing we're going to do is use the 'NRC data' to look at equally 'simple sentiment analysis'. 'NRC data' is little bit interesting because what it does is it—the words are coded into two sentiments, positive and negative, and eight emotions. So, every word can be either positive or negative, and it can also have associated with it the emotions of 'anger', 'anticipation', 'disgust', 'fear', 'joy', 'sadness', 'surprise', and 'trust'. Okay! So, this is what the it contains. So, it's kind of a nice thing to do. And again, you know, it's probably better at reading—using this for like, Facebook posts, where people are actually commenting with more emotion than for restaurant reviews. But, you know, let's take a look at it, we might get something interesting out of it. So, the first thing we're going to do is read the 'NRC sentiment data'. And like I said, this is not available online without permission anymore.
So please, it's part of the data set, I've given you guys, so please, use it only for this class. And, all you're going to do here is we read the sentiment data and we're going to construct a dictionary out of it. So, let me just run this, and insert a cell above. And, we construct this emotion dictionary...dictionary out of it that looks like this. For every word, we attach a list of emotions and sentiments to it, right! So, this is 'abandonment' for example, as the 'anger', 'fear', 'negative', 'sadness', and 'surprise' associated with it. Abhorrent has 'anger', 'disgust', 'fear', and 'negative'. What a positive word, let's see if we get something

positive here. Here, 'absolution' has 'joy', 'positive', and 'trust'. So, these are the things we do and all we're really doing is we are going through this stuff over here, ignoring the first 46 lines, which is I guess just junk in the text and then, taking the...the text itself. Let me see, if I can take a look at it. Here actually, there is no point. So—and then, you can take a look at the text and see how it's structured and extracting the emotions, and then appending them to this—if the dictionary already has a key called 'abrupt', or a key called 'abscess', so the first time it goes it creates this key and attaches the word 'negative' to it as a value. And, the second time on it just appends or, you know, adds that to the list of values that are the value for the key abscess.
All right! That's what it does. So, that's all it's really doing here. So, it's really straight forward and we can make a function out of this if you like. So, let's do that. You probably don't need it for this, but if you want to use it again then this is going to take care of that for you. Right! You have this function forever. So–and, we can check it out. We look at abandonment—'abandoned', and we see that it has 'anger', 'fear', 'negative' and 'sadness' associated with it. Now, that we've downloaded the NRC data, let's see if you can attach emotions to various restaurant reviews. This is not going to work very well because you're going to use the yelp API. And the yelp API, doesn't give us reviews. All it does is it gives us a review snippet. So, yelp, I guess, protects their reviews and they don't want people downloading them because that's where they're intellectual property really is, right! So, you get one review snippet, so what we're going to end up doing is we're going to analyze these reviews snippets for emotion. So, hopefully that will give us some information. And, more importantly, it'll give us some experience both using NRC data as well as using the yelp API. Before, you can use the yelp API, you need to actually get API keys.
So, you need to go to this website and get the API keys, and where it asks you for a host just put anything in, just put, you know, whatever, Google dot com. Just make sure that the HTTP:// in front of it, and then you'll be all set. And then, you copy the various keys into the variables below. So, what do you get? Four different keys, 'CONSUMER_KEY', 'CONSUMER_SECRET', 'TOKEN' and 'TOKEN_SECRET.' And, you want to put those keys in over here. Okay! So, each key value you want to put in over here. So, once you've got those, you can get moving. I am actually—have saved mine to a file. I don't want like millions of people using those keys and get me into trouble with yelp. So, I've thrown mine into a file and I'll just load those in. So, those are loaded in now, right! And, once I've got those, we can start doing a few things with that. So—Yeah, first thing we're going to do is the yelp API, the first thing it requires is it's requires you to give it the location that you are looking for. So, what we're going to do is we're going to say, go to this location and find all the restaurants within a certain radius of that location, All right! We'll see all that in a second. So, the first thing you want to do is get the latitude and longitude of Colombia University. So, that's pretty easy. We wrote this function way back in week three, yeah, week three; where we got latitude and longitude, we're going to use the same function again. And, this is what you do when you're working with bytes of data. You write function, the reason you write functions is that you want to be able to reuse them later for stuff, right!
So, we wrote this function so we'll reuse it. So, generally you want to keep it in a nice, safe place, where you can access it, so—that we can get the lat/long for Colombia University and then, we can setup the search parameters for yelp. So, the search parameters for yelp are—we have four different things you want to give it. We want to give it the things we're looking for. We're going to look for 'restaurants' in

this neighborhood. We want to give it the latitude and longitude, and that's the format that we want to use inside curly braces. We give it a string that contains the latitude. And inside another set of curly braces here, we give it a string that contains the longitude, then we give it a radius filter and the radius filter is the, you know, the distance that you want to look around, maybe like in meters or maybe 500 meters or something like that, half a kilometer, and then, you want a limit, which tells you how many results you want.
Okay! So, you know don't have to have a limit, but it's a good idea to limit your results when you're testing stuff out because you don't want to get blocked by anybody. So, always when you're testing stuff on the internet, you want to make sure that you don't exceed stuff. So, let's say, we set our search parameters here with this latitude, this longitude, and 200-meter radius. Okay! So, we do this, oops, I did not run that. And, we get this thing, okay! So now, we've got that stuff there.

Video 4 (6:06): Sentiment Analysis Part 3

Once we have the parameter set up, let's get our results. So, the first thing we want to do is we want to import this library. 'rauth' is a nice library that deals with authentication, and it's often used for API's and it works well in Python. So, let's write that down, let's import that in. So, install that first. So, let's go up here and do, 'pip install rauth', and hopefully that will work fine. Yeah, so it successfully built that we have 'rauth' done. So once, we have that done we can get our results, so we to—we 'import rauth', we get the 'consumer_key', secret, blah, blah, we don't need to actually enter all this stuff in. We can use these directly, we don't need separate variables, but it's there...it's there. And then, we set up a rauth session. So, what the rauth session does is it passes these keys and, sort of, logs us into it. You can think of it as being logged into the yelp website, so that you can get data. And, I should point out, if you want to—before we were here when we were trying to get the yelp keys, in this, you know, way back here where it said we want to get the yelp keys from this site, you need a yelp account to get the keys.
So make sure you make an account. It doesn't really matter, if you make an account and get it, but you need an account for that. So, go here. So, we set that up, so we set a session and we pass these keys to it. And, once you set up a session, we can then, much like we did when we were logging into request dot get with sessions, we can use that session as our method for getting data from there. So each time, we send a request it's going to use the same session. So, it will keep us essentially logged in. And, we give it our parameters. The parameters we give it are the ones that we constructed over here, 'set_search_parameters', we got that, right! And then, we do 'request' dot 'JSON', get all this stuff and we get our results. So, we got this, right! So now, here we call it. We call 'get_results'. To get results, we want to pass to it the parameters that we're going to give it. And, to get the parameters, we're going to call 'set_search_parameters', give it the name here. We're actually getting the latitude longitude of 'community food and juice'.
I don't need to use Columbia for this, and get 200 meters around that, okay! The reason is that, well, this is the same 'get_lat_long' function and returns a couple of different things. We want to get the latitude, which is the zeroth element and longitude, which is the first element and that gives us latitude and longitude. So, we got that and we get the results. And now, once we get the results, we can extract

snippets. So, let's take a look at this for a second. Insert cell below. All the–take a look at what we get back. And then, we look at response and this is what we get back, right! We know—we have already done 'JSON', so we know what 'JSON' looks like. And in this case, we're getting a dictionary. The dictionary is a bunch of keys and the keys are 'businesses', which contains a list of businesses, I guess. So, like id, Shaking Crab New York, which is a restaurant...new restaurant here, actually, and 'Community Food and Juice New York', etc., etc., right!
So, it's got a whole bunch of stuff. So, we can go through this thing and pick out what we want from this, so we can get snippets, and it turns out that we want to get the businesses. So, if I look at this in step-by-step–if I do 'response' 'businesses', it will give me the value of that key, and that's all the stuff here. And, if I do 'response' 'businesses' 'name', oops, oh, for each business, so it's a list, right! So, I want to look at 'element' '0' and get the name 'Shaking Crab', and then I can get the 'snippet_text', which is what I'm interested in, 'Easily the best and freshest Cajun-style crawdads ever.' And then, I can get the 'id'. The 'id' is, kind of, probably not necessary for what we are doing here, but if you are using yelp, you know, for more than just what we are doing here, the 'id' is very useful to get more data from the—from yelp itself, okay! So, it's a—you can use that 'id' to send the 'id', and you get more business data, like open hours, and things like that, which is kind of useful. But, we don't really want all that right now, so we'll just—so I'll just give an example here.
So, what I do then is, I get these three values, the id for the restaurant, and I probably should use id over here. Let me call it id, n-o. The reason is id is also, a Python reserve word, right! It tells you it's a function, so it tells you the id, the location and memory of something. We don't mess that up. So, what I do is I take the–get the 'business', the 'name', the id number and the 'snippet', these three things, and I put them inside a list and I append inside a tuple, and I append these tuples to a list that contains all snippets, and I get this nice little thing that says 'Shaking Crab', that's the id number here, that's the name, and that's the review. For all these restaurants over here, right! So, that's what we got there.
And, we can functionalize this. Always good to functionalize stuff, so I functionalize this, and make this—just take this stuff and make it into a function.

Video 5 (7:07): Sentiment Analysis Part 4

And now, we want to analyze this to the emotions analysis. So, we can set up a function for that as well. So, what we do is we take a piece of 'text', which will be a snippet in our case. We take our emotion dictionary, the one that we computed earlier. And then, for each—if we take—for every...every 'y' in emotional dict or values we get a emotions, right! So, what the emotions are going to do is—let me print this out. It's going to tell us what the emotions are, 'print(emotions)'. What you should see is? See, there is a list of emotions, 'restaurant', 'fear', 'trust', 'negative', 'positive', 'joy', blah, blah that kind of stuff, right! That's the—what we are seeing—this is what I just printed, and, not that it's very useful, but what we are doing is we're getting the list of emotions and this actually helps a little bit because if you create your own emotion dictionary then, you're not stuck with just the emotions that are there in NRC.
So, let's say you're doing your own text and you want to, it doesn't have to be emotions it could be any kind of labels. So, you have a piece of text and like maybe analysis reports, a bunch of analysis reports

and you want to label the reports with various characteristics, like for example, is it positive or negative, is it indicating that it will achieve its earnings potential by a little bit or a lot or you know that kind of stuff right. So, you have multiple different categories that you want to associate with that report and you don't have to call them emotions they could be anything right. We call it emotion there but it could be anything at all. So–and then so, what we will do first is analyzer will extract those the list of emotions from the thing here.
So, this extracts it by creating a set of all emotions. So, every it goes through the entire dictionary and every label it has it adds it to the set. And remember, sets have unique values, so we end up with only a unique set of emotions there. Then, we create a count for—a dictionary for the emotion count, and we initialize that to zero. So, if you're giving a text and you want to find in that text we want to find how many times does it have words that are 'anger', words that are fear, words that are 'surprise', words that are 'joy', you know that kind of stuff. So, we're going to count that, for each emotion we're going to count that. And then, once we do that we basically just add them up and we go through it and keep adding the number of, incrementing the count depending on whether a word has that emotion associated with it or not. And then, we divide that by the total number of words in the text, that's what this is here okay! So that will be our emotion count.
So once—that function does that and now you can take this thing and do a little bit of work with this. So what we'll do is, oh let me get rid of that again, clean it up a little bit, yeah. So what we'll do is we'll run this and we're going to print essentially the percent of, the proportion rather not percent, proportion of words that express these emotions. Obviously, the proportion of these will be low because many of the words are like the and is ideally, we should always just take them out, you know, because we don't really care about those words so that might be a little bit better. But for starters this is a good one. So, when we look at this we look at our various emotions and we find here we actually have oddly enough community has inspires 'fear' in '0.04' proportion of the cases. It's 'negative' in '0.04', maybe the same '0.04'. And has 'sadness' in '0.04'. There's actually something here that's quite high. Yeah, Tom's restaurant has a '0.24' positive, so that's a very high positive rating, 'joy' of '0.14' 'trust' of '0.14'.
Tom's restaurant in case you don't know is featured in the TV series Seinfeld, so it's the restaurant that is where, you know, the cast, the characters of Seinfeld meet very often and you see the exterior of the restaurant, you don't see the name, but the exterior is featured in that restaurant, not the interior. So, it's a popular spot, people come here as tourists this—to see that restaurant and perhaps that's why they like it so much, they trust it and feel positive about it, they enjoyed seeing it or whatever. So, this is what we get here! So, it's kind of interesting when you look at it that there is, obviously you're not going to have a high proportion of these words like I said because they're these things. So, but to some extent and these are just snippets so I wouldn't take it, you know, wouldn't consider them very judgmental on the restaurants. And we'll see later what we can do with a lot more reviews, but this is sort of what we want to do with this stuff here. And we can 'functionalize this', it's okay to 'functionalize this' because we're going to use it again later anyway.
So, this is going to produce the same result here. So, I just took the stuff that we had and we wrote a big function out of it. And finally, we'll functionalize the yelp stuff as well, so we can now create or analyze nearby restaurants thing which is going to be really nice to use. With a—give it an address and give it a radius that we want to see and then we use our 'get_lat_lng' function here to get the latitude and

longitude for that address. We use the get—'set_parameters' to set the parameters, 'set_search' parameters. We use our 'get_results' function that we wrote earlier to get a response. We wrote 'get_snippets' function to get the snippets. And we use our 'emotion analyzer' to analyze them. And now we can run this and it tells us, it gives us the same result again.
But the nice thing now is that we can take any location anywhere in the world and look for restaurants within certain distance of it. So for example, we're going—visiting London and we want to go visit the Sherlock Holmes Museum at '221 Baker Street', there is no '221B'. So, we can just enter 221B...221 Baker Street and give '200' or you can give the full address if you like but it'll figure this one out correctly. And it tells us here that there are these restaurants nearby and we've got you know emotions attached to them, like 'anticipation' of '0.25' for 'Beast and Barrel', 'Beast' plus 'Barrel' that's interesting. Positive for 'Beast' plus 'Barrel'. The 'Beast' plus 'Barrel' seems to be the place to go to when you are visiting the Sherlock Holmes Museum, it's got high 'trust'—you have some 'fear' as well interestingly enough. 'Positive', 'joy', we can see the snippet and figure it out.
Remember this is just one sentence, so you know I wouldn't take this as, let's say it's just fun. So you could take your...your home address, put that in over here, and you know if it's not a well-known place like '221 Baker Street' or Columbia University you might want to put in something like you know the full, entire address. And then give a distance around it, again if you're in a suburban area where you need to drive to everything, then you probably want to give a longer distance there. If you're in an urban setting, then a shorter distance. And see what it says, what restaurants are near you.

Video 6 (9:56): Word Cloud

Another thing we can do with text documents is to look at 'Word Clouds'. 'Word Clouds' are not really rocket science. All you do is you look at the text, you look at the words in the text, you count their frequency, and then you throw them out into the cloud where each word is sized based relative to the frequency—its frequency in the text. So, they're not really rocket science but they're really very useful because you can very quickly see what the text is saying, in...in—and you get the top few words and it tells you, oh yeah that's what—that's what this thing is all about. So, let's take a look at that, and to do that we also need to install 'Word Cloud', we should probably do. Insert cell above, and install 'Word Cloud'; 'pip install wordcloud'. Okay, so once we've got 'Word Cloud' installed we— what we'll do is we will take all our snippets and combine them into one string, and then generate a 'Word Cloud' using the words in the string just to see what do they say about restaurants in the neighborhood that we have around here. And, so, let's do that. So, all snippets are these ones. So, what you want to do, essentially, is run through this here and extract the item for each tuple in the third location, location two, right!
So, we want to run through this list of snippets and extract whatever is in location two of each tuple and then concatenate them all into one giant string; so that should be easy, so we'll do that here. We do this and we get a nice long string that does all that stuff here. And now you want to set up a 'Word Cloud' for this. A word is pretty easy to set up. All we do is we import from 'Word Cloud' we import 'Word Cloud' and 'STOPWORDS'. 'STOPWORDS' is a set of words that, and this we'll use these stop words in other places too, which are very commonly used, like often, the and at and in and all that kind of stuff; so, we

eliminate all those words. And then we're going to display it, so we want to import 'matplotlib' as well; it uses that. And make sure it's in line, again, we want to display it in our thing here. And then we create the 'Word Cloud', we say 'WordCloud' equals 'Word Cloud', which is the object we've imported over here, give it the 'stopwords'; so, you can give any list, any set for the 'stopwords' over here, you can create your own stop words if you like. Create a 'background_color', 'white', and that's right there, a 'width' and a 'height', and so to make it bigger or smaller, and then you generate it, okay. And you generate it using—you're generating the word cloud here using the text that we created over here; that's the variable we're passing to it. And then we use the—the show, matplotlib show—image show button; we're creating an image so we want an image, 'imshow' is image show and we say show this 'wordcloud', right, that's this one. And then axis is off, doesn't matter, we don't need axes for these; we don't need x and y. And then show it, right! So, this is what it does here.
So, it says 'pretty', 'French', 'food', 'though', 'place', 'restaurant', blah blah, right! But this is all the restaurants. What we would like to do, of course, is compare the restaurants and see how they work together. So, in our data set that I...that you should have downloaded, there are four sets of reviews; a review for community, and that is in a location 'data' slash 'community', and there are, I think, 15 files per review, so 15 reviews. And then, le monde is in 'data' slash 'le monde', and heights is in 'data' slash 'heights', and is in 'data 'slash 'amigos'. These are four different restaurants in this neighbourhood, 'community' is a, sort of, local American food restaurant, which focus on local foods. 'Le Monde' is like a sort of 'Frenchish' kind of restaurant. 'heights' is Tex-Mex kind of restaurant, and 'Amigo' is also, sort of, tex, sort of, mixed Mexican restaurant. None of them are pure Mexican but, you know, they're—but they are kind of Mexican. So—and so, what we set up the thing here, we from—we're going to use 'nltk' for this and from 'nltk' dot 'corpus' the library we import this 'Plaintextcorpusreader'. Plain text corpus reader is a way of reading data into as a corpus into your—into your application here.
So, what it does is it essentially takes a file. If I look at our set of files here, I can go here, 'data', if you go to 'data' 'community' it has all these files here, right! So, it takes these files and it can load them into as separate files and each—each of them has separate documents into your application and that set of documents is what's called a corpus in this—in this...in the lexicon that we use here. And the nice thing is then that you can actually sort them. So here, we have community 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; you can sort them by—by the names of these things, and we'll use that for... for this it doesn't matter but for the inaugural text we'll use that. If you—actually they're right here. You can see the inaugural texts are numbered by year so we can always assume—we know that '1941' comes before '1953', right! '1953' comes before '1993', so these become sorted and we can use that sort—that fact that they are in a certain sequence when we're doing analysis. Not so much for community and all that, we don't really care, plus it's not going to work very well because they're not really, you know, sortable. The texts of one zero are going to come before two, for example, which is not what we want. Oh sorry, that is what we want. That is what we want, right!
One, one zero, one one, yeah, that actually what we want, so it'll actually work. All right, so what we do is we set up the root directory where we are going to get the files from. So, we have a root for community and we set up the—the format of the files, like your directory might contain many other files. The files we're looking for are 'community' dot 'star' for community. We could put all of them in one location and then we have community files, 'community' dot 'star', le monde files in 'le_monde' dot

'star', etc. etc. But we can work with that. And then, we set up our corpuses and our application by creating this plain text corpus reader, which has the root, it needs to know the root where the files are located and how to identify the files in that folder, right! So, this could be anywhere, it doesn't have to be in data, community, it could be anywhere you want to give the complete path to this. In our case we don't have the complete path but we are giving a relative path because data– the directory data is in the same location as where our current text mining dot ipymb file is. So, we do all that stuff and we can look at the file IDs of these things and we see that they correspond to the file IDs there. So, what the plain text corpus reader is doing is it's storing these things with their file IDs and...and—in a...in...in a sort of internal directory structure, so to speak, and you can access each one separately, if you want to. Or, you can access all the data. So, you can say 'amigos_data', which is our variable that we put in this one here, right! This is 'amigos_data' where we can use the raw function to extract all the stuff in this here.
So, this is the entire text of all the reviews, and these reviews were actually—this cut and paste. I just cut and pasted them from yelp, so that's all they are there. So now we—I am going to just modify 'Comparative Analyzers' so that it tells us where the restaurant name is; it's a very simple one, you can take a look at it, and then we run it and we get this over here now. This is our all snippets, so okay. I run it with 'restaurant_data'. So, 'restaurant_data', I'm going to give it the tuples, I'm going to give our—the name of the restaurant. So, essentially, I've taken out the ID field from there; that's what I've done, right! So, we have 'community' and 'community data raw', 'le monde', 'le monde data raw', 'heights', 'heights data raw', and 'amigos', 'amigo data raw'. So, I'm taking all the text of those 15 reviews and we're going to look at the—compare the emotions on those things there. So, for these four we get this over here. Looks very different now. The point zero four for 'community' 'fear' has vanished. The reason being that the snippet is not included in the reviews; the snippet is what yelp is providing me right now; the reviews are what I had downloaded earlier. So that snippet is not there and so we're not seeing any 'fear' in community right now. We are seeing a bit of 'fear' in 'amigos'. So again, keep—bear in mind that these numbers though they look small are actually quite meaningful because there's a lot of other stuff that doesn't have emotion attached to it in the reviews. So, what this tells us is that 'community' has the highest 'positive' rating, '0.05'.
It also—they all are equally 'negative', and equal 'trust' inspiration, but for some reason 'amigos' inspires some 'fear' and both 'community' and 'amigos' have at least something that is disgusting about it, about them. 'heights' has the highest 'anticipation' and this actually makes sense to me because it's a very nice bar and people like to go there and drink. There's some 'sadness', I don't know where that comes from, associated with 'community', 'heights', and 'amigos', and there's also 'surprise' in all of them. So that's kind of nice, right! So, we can do this kind of stuff here. So that's the—this thing. And again, just bear...bear in mind that you can—you don't have to use 'fear', 'trust', 'negative', 'positive', 'joy. If you're working in a domain you can actually attach your own words and use those instead of using these, and then use the same procedure to, you know, figure out what the documents are trying to tell you.

Video 7 (9:48): Text Analysis - Complexity Analysis Part 1

Another thing we can do very–for very–at a very simple level is to look at how complex a particular document is. What's complex? 'Complexity' is the 'average word length' of a document. In general, if you have a technical document, it might have longer words. The average sentence length of more complex—You know people are expressing more complex ideas then, they would probably use longer sentences, rather than shorter ones, and you would also like to see the vocabulary of the sentence. So, for this, it's a little bit iffy, but if you want to see the ratio of unique words used to the total number of words in the document, and this is a little bit iffy, because, obviously, as a document gets longer, the variety in the words is going to be lower in proportion. I mean, there are only so many words in English language, after all.
So, if you have a book that is 500 pages long, and you have another book that is five pages long or a short story of five pages long, the proportion of different words in the five pages long text is going to be higher than the proportion of different words in the 500-page text. I mean, it makes sense. So, keeping that in mind, so, when you do the vocabulary proportion, you probably want to be also comparing documents that are similar in size. Just keep that in mind. So, in general, when you're dealing with text mining, you start with the idea of a token. The 'token' is 'a sequence or a group of characters of interest.' In—for example, if you've been using a 'word' as a group of interest. So, the word is our token, but it could be a sentence. So, it could be a document. It depends on how you—on what level you want to do the analysis, but generally, a word is not a bad way to look at it. So...so, the first step is always to convert stuff into tokens, and an 'nltk' provides two functions that are useful for that. There's 'sentence_tokenize' and 'word_tokenize', and each of them are there to create words and sentences out of the, you know, basically get rid of stuff that is not—something that's a word or a sentence, right, like, for example, punctuation or emojis or, you know, stuff like that. So, if you take Le Monde data and we word tokenize and sentence tokenize it, we get '188' sentences and '2595 words in our compendium of reviews of that thing, and we can now look at the sort of complexity of it.
So, what we can do is we could say, "Hey, how many characters are there?" Well, we have a piece of text. We can count the number of characters. So, that's the '12,332' characters. How many words are there? If—we can count that, and then we can divide the number of words by—we can divide the total number of characters by the total number of words to get the average length of a word. So, in the Le Monde reviews, the average length of a word is four, which is pretty much par for the course. You would expect four or five to be in most documents to be the average length, because there are many words like to, in, and, a that are short words, and they get used very, very often. Then you can take a number of words in the text and divide it by the total number of sentences, right! We know how to calculate those, too, and you get a length of sentences feature here, which is '13', which is kind of on the low side, I think, but you now, it's okay. And then, you take the vocabulary that is the number of unique words, and the number of unique words is easy to do. What you do is each word, as you get it, you add it to a set of words.
So, sets are kind of useful for this, because sets, if you add a word twice, it's just going to ignore the second one. So, you get the unique words. That's our vocabulary, and we can take the length of the vocabulary and divide by the number of words to get a proportion of unique words. So, this tells us they

are '29 percent' of Le Monde's text is words that are used—are the unique words in that text. And, we can take this stuff and 'functionalize this'. We're going to use us a little bit. So, it's a good idea to 'functionalize this'. So, we try to get complexity function, and we can do that for that. So, what it does is it returns us a touple that contains the length of the unique words, 756 unique words, four words per...four characters per word, 13 words per sentence, and '0.29' is the proportion of unique words to total number of words. And then, we can run this for all of them, and we can compare across the texts. So, we see here that, in general, 'Le Monde' actually has the lowest sentence complexity...complexity. It's '13', and whereas 'community' and 'heights' have '16'. So, people who write reviews, very naive. Remember, we are looking at 15 reviews out of possibly thousands, right!
So, it's fairly naive this thing, but what we are saying is that people—roughly, what we would say that is that the people who write reviews for 'community' and 'heights' tend to write more complex sentences than people who write for 'Le Monde'. And, you know, 'amigos' is in the middle, but they all use the same number of characters per word, and they have, pretty much, the same vocabulary percent here. Though 'amigos is really low, even though, if you look at the—here. community has a very high number of words, but if you look at these three, '756', '720', and '792', they're fairly close to each other, and comparatively fewer words are used, fewer unique words are used in 'amigos'. So, they tend to repeat the same words more often, for whatever that means. Remember, this is not really a very scientific piece of analysis here, but that's the idea. We could do a 'word cloud' comparison'. So, what we could do here is—and let me just run this and show you what we're going to get out of it. What we want to do is we want to see are there differences in the way these restaurants are viewed by the reviewers? If you had a thousand reviews, this would actually be very, very useful. So, what we can do is we create 'Word Clouds', and what we're going to do here is to a couple of things.
One is we're doing 'Word Clouds'. We saw how to do that before, and the second is, we're going to this sub-plotting like we did with our bureaus and agencies at all that stuff in the data visualization, right! So, this is a very similar to that thing here, and in this, we have a bunch of stop words. So—oh goodness, this thing is not moving very well, is it! Here...Here, so... So, here, what we do is we construct our 'word clouds' but we remove unwanted words and we can add any words as we like to this. So, what I'm doing is I'm just saying if we have a string of delete words, then we can just delete them, but I don't have a string of delete words, do I? I have an empty string. I can delete them if I don't like them, right! And then, I, of course, have the stop words from here, but I can delete words that are not gemane, or don't make sense, in my domain, right! So, that's an additional thing. I'm just adding that to it. I can remove short words. Short words are words that are shorter than a certain length, minimum length, and here I just said 0 initially, but we can do that stuff here. And now, we have four restaurants.
So, I have a two-by-two array, or two-by-two subplot structure, and then, we use the same thing that we did before and show our thing here, and this is a little bit on the big side here. Let me see if it fits. Yeah, kind of, let me see if I can... And I make the 'height' 900. So, that's smaller. That's a little better. So, what this tell us is that, if you look at this, 'community' is good for 'brunch'. 'Le Monde' also has some 'brunch', but you don't see 'brunch' in 'heights' and 'amigos'. So clearly, people don't really use heights and amigos—or don't talk about 'brunch' anyway when they talk about 'heights' and 'amigos'. 'heights' has a 'bar', and like I said before, it has a good bar. People like it because of the bar, and we see 'bar' shows up as a huge word over here. This has 'pancake'. So, I guess 'pancakes' are good. 'Coffee'. This

probably has 'burritos', 'quesadillas', and 'margaritas', and also, it's more of a Mexican kind of place, right! So, 'quesadillas', 'margaritas', and those kind of things, and so, what we look at these...these things, you know, 'night', 'food', 'bar', we can clearly almost sort of visualize that this place is good for 'brunch'. It has 'pancakes', you know, 'brunchfood', eggs, that kind of thing. This place is also—has 'French food'.
It says French. It's got—people talk about service. We don't know whether they say good service or bad service, but it also has dinner. So, it's probably a 'dinnerish' place, and 'drinks' are good. 'Coffee' is good over here, or at least they talk about 'coffee'. We don't know if it's good or bad, and these ones are more Mexican-oriented here with 'margaritas' and 'quesadillas' and those sort of things and 'heights' also has 'tacos', Mexican food, and the bar is really one reason why people like it so much. So, comparing restaurant reviews is not going to get anything useful. So, what we want to do is we want to look at the—we'll look at the 'nltk's' inaugural text. So, 'nltk contains this huge corpora of pre-tokenized text', which we downloaded upfront before we started this thing here, what you want to do is you want to import that corpora. So, let's just do that.

Video 8 (8:07): Text Analysis - Complexity Analysis Part 2

So, now that you've downloaded all the stuff and uploaded it onto your—imported it into the application that we have here, we're going to focus on the inaugural address corpus. It's worth looking at everything else, as an example, too. For example, 'Sense and Sensibility' by 'Jane Austen' is really nice. You can look at—take a look at the—easily identify the main characters and even with a little bit of effort, figure out which character is, you know, 'Sense and Sensibility' is about love interests and stuff, but which characters have a love interest with which other character, that kind of thing. So it's out of—just by looking at the pure text locations of items in the text. So, but we want to look at the inaugural address corpus, mostly over here. So, let's—what we're going to start by is to look at the presidential inaugural speeches and see what we can say about them in terms of complexity, emotions, you know, whatever else you want to do. So, the importing that we did earlier over here has imported all that stuff into our application, and we can now look at them.
So, if we look at 'inaugural' dot 'fileids', we get to see all the IDs for the files for each president. So, we have everything from starting with 'Washington' in 19—'1789' all the way to 'Trump' in '2017'. If you don't have '2013' 'Obama' and '2017' 'Trump', if you couldn't copy it for some reason, that's fine. You don't have to use that. You can just use somebody else, right! It is just a example. So, we can look the raw— we can use 'inaugural' dot 'raw' and earlier we saw inaugural, if you give corpus and say dot 'raw', we get all the text in the corpus. But, you can select which particular file you want to look at, which file ID you want to look at. So, we'll take, for example, Lincoln's '1861' speech and that becomes his speech over here. So, that's what we want to do here. So, what we want to do now is we'll look at four presidents, and I'm just going to take these four over here and please, up front, this is not a political statement. In fact, we find that there's nothing much we can say about any president anyway, mostly. So, here we got four presidents, 'Trump', 'Obama', 'Jackson', and 'Washington'. And 'Jackson', 'Washington', and 'Obama' all have had two terms.

So, we have two different inaugural addresses for them. So, I'm concatenating their raw speeches for 'Obama' '2009', 'Obama' '2013' into one piece of text, Jackson's '1829' and Jackson's '1833' into one piece of text, and Washington's '1789' and Washington's '1793' speech into one piece of text. So, these become our raw text. And, we can see each one of them is a separate text here. We've got 'Trump' and 'inaugural' dot 'raw', 'Obama', and then this concatenated stuff over here, like this. So that's one piece of text and, similarly, this is one piece of text, right! This is together. So, we have four pieces of text total, 'Trump', 'Obama', 'Jackson', and 'Washington', and we're going to look at them to see what kind of complexity we can say about them. So, if we run our complexity function that we wrote earlier on this, we get this over here. We find that the vocabulary used by Trump is 540, which is fine, because you think about it, he's the only person with just one speech.
They all have fairly, you know, five words per five characters per word, which is, as I said, is on the higher side. Four to five is usually what you would expect. The number of words per sentence is '18' for 'Trump', '25' for 'Obama', '45' for 'Jackson', and '62' for 'Washington'. So, we can see here that in general 'Trump' speaks in shorter sentences than 'Obama' does, and both 'Trump' and 'Obama' speak shorter sentences, than 'Jackson' did, and all three of them speak in far shorter sentences than 'Washington' did. And then, we can look at the vocabulary-to-text ratios and we find that it's '0.32' for 'Trump', '0.27' for 'Obama', '0.33' for 'Jackson', '0.38' for 'Washington'. And, this is partly because 'Obama' also has probably the longest speeches, so he has the lowest vocabulary-to-length ratio, like I said, that's not a good indicator at all. In fact, if you look at this here, 'Washington' actually has really small speeches, you know, than we see as earlier on, presidents didn't speak for as long as they do now, that's for sure. So—and also 'Trump' has a very short, only one speech compared to other—the other three.
So, that's our basic analysis. Now what we can—Looking at this, we see that 'Washington' had a sentence to, word-to-sentence ratio of '62', right! So, he had '62' words per sentence, that's long. I mean, can you imagine listening to someone speak in '62' words per sentence? And 'Jackson', '45'. 'Obama' '25'. 'Trump' '18'. So, we know that 'Washington' was the president before 'Jackson', 'Jackson' before 'Obama', 'Obama' before 'Trump'. So, perhaps this is not something that is peculiar to these presidents but maybe there's a secular trend, where presidents in general and probably presidents in specific and probably people in general speak in shorter sentences. So, if that's the case, we should try to figure that out. So, we know that all our text are ordered by presidents. So, we can see that if we look at our corpus over here, it's '1789' 'Washington', '1793' 'Washington', '1797' 'Adams', so they're ordered by president.
So, what we can do then is we could try to see what is the word-to-sentence proportion that we have, a number of words divided by the number of sentences for each of these things over time. If we can graph that then, that'll tell us whether there is a secular trend in shorter sentences, you know, over time, over here. So, let's see if we can do that. So, what we can do is we take our corpus and plot our sentence lengths. So here, what we do is we get the complexity, we call off 'get_complexity' function here, All right! For each inaugural file address, so take all the inaugural addresses, I think there are '58' of them, take all the inaugural addresses and for each address, we compute the complexity, extract just the sentence, this number from that, that's what we do here with the '2', right! Zero is '636', one is five, and two is '62'. So that's what we extract over here, All right! That's what we're extracting here from

'get_complexity', and appending that, to a list of sentence lengths, and then we can just plot it. So, it's pretty straightforward, right! And looking at this, we can say, "Hey, yes, definitely".
The—Washington in his first speech was long sentences, but they plunged a little bit but over time there's a secular trend in declining sentence lengths and it sort of stabilized somewhere over here, right! So that's Trump, that's down there, that's Obama, Obama one, Obama two. Obama two was a little bit longer sentences than Obama one. Bush two was longer sentences than Bush one. So, you know, it's a...it's a...it's a secular trend, and if we're looking at this, we might think, "Hey, Trump is, you know, extraordinarily low but, you know, in reality, we have, you know, Bushs and Clintons who have been speaking at pretty much the same length and even someone down here, you've to figure that out, right!" Somewhere, in some in—I would say about 42 or 43 in the presidential inaugural address list. So, it's not as far out of the ordinary as you would think, okay!

Video 9 (5:31): Text Analysis-Dispersion Plots

So, that's the first thing. So, we already know this, now we can say, "Hey, yes, presidential addresses, we can see this was confidence, that presidential addresses have decreased in complexity from the first president to today, right!" So, we can say that with confidence, because the graph clearly shows that there's a trend of that sort. The next thing we can do is we could look at dispersion plots. Dispersion plots are really useful, because what they do is they can show us, given certain words, they can show how the relative frequency of those words change over time. So, for example, if you look at our presidential addresses, and we are interested in seeing roughly what sort of things presidents consider important, presumably, if they consider it important, they will talk about it in their inaugural speeches. So, I just took a bunch of words, and this is not, again, scientific, 'government', 'citizen', 'freedom', 'duties', 'America', 'independence', 'God', and 'patriotism', and we want to see how the frequency of these words, or how the usage of this words—these words have changed over time, right!
So, there's a nice function called dispersion plot that applies to 'nltk' texts, and text 4 is the entire inaugural address corpora. So, in that corpora, we're going to call dispersion plots. And, we get a dispersion plot are all these things, and we run this and we see that this is what it looks like. So, it tells us that 'government' is fairly, consistently used across the board. 'citizen', there was—there's some gaps in the middle, gets used a lot in the middle somewhere here. 'Freedom' is a word that wasn't used very often early on, but off late it's been used a lot. 'duties' was used a lot early on, and this could be 'duties' of the president, it could be 'duties' of the people or the citizens, we don't know or the government, we don't know what 'duties' they're talking about. But 'duties' was used a lot in the early days, and then no one talks about 'duties' anymore, pretty much it's gone. 'America'–and this is an odd kind of thing. 'America' is barely used in early years, but in this century, it's a... it's sorry in the 20th century onwards or probably mid 20th century or early 20th century onwards, it's been used a lot. And, it rarely, an address goes down without America in it. And my guess is, if you look at the 'God' part of it as well, I don't know if I can highlight this, no I can't. Okay!
So, if you look at 'God', if you look at 'God'...'God' was never used in the early days, or very rarely used. But, now 'God' is always there in a presidential address. So, most likely this comes with the "God bless

America", which is a feature of 20th century America. It's not—wasn't really used that much before that. and 'patriotism' has on the decline apparently. And independence, is also sort of there, here. But, you know, if you look at these words, we might want to, sort of, stem them a little bit because after all, if you have free–'freedom', duty–'duties'. We want these words to be used interchangeably, right! We don't want duty or duties to be used as separate words, right, so, patriot, 'patriotism', these kinds of things.
So, 'nltk' has a nice way of stemming these words, so that you get a stem for, like for 'freedom', you'll get 'free', will be the stem for 'freedom', and you get sort of a subset that tells you what the root word is, or the stem of the word is. And, it uses an algorithm called 'Porter Stemming Algorithm' and that's explained in this website over here, which you can take a look at. And, what we'll do is, we'll take all the words, make sure they're lower case, and stem them, and then run our dispersion analysis on that. So, let's do that and see what we get. So, now we're going to get something that looks a little bit different, right! So, here we've got, some things are pretty much similar, 'god' and 'America' whether we stem them or not are pretty much used in the same proportion, right!
They're recent additions to our presidential addresses. 'patriot' is now—becomes, over here not, it's pretty much the same. 'patriotism' has not changed that much, Independence not changed that much. But 'govern' and 'citizen'– and I guess I got rid of duties for some reason, I guess I—we need to know the stem of duties, but anyway, 'citizen' now is, let's see how this compares with before, 'citizen' which was, you know, if you noticed was very, sort of patchy– there were patches where it was used, we find it's used a lot more, most likely because citizenship or citizens is now getting included in this. So, you could say, "fellow citizens" or you could say, "a citizen of America", or, maybe the context is different and it's worth thinking about that too. But the word citizen now, we find is actually used a lot more frequently. Though it was used a lot more frequently before, but it's still used a lot. So, this gives us a better picture of what's going on over here. And—so, this gives us some idea of what changes we can see over time on our presidential inaugural addresses, and clearly, 'God', 'America' are big. 'patriotism' is out. And we saw earlier, and I think the reason is I couldn't figure out what the stem of 'duties' is, that 'duties' was something that was there earlier, but not anymore. And that's pretty much it.

Video 10 (5:38): Text Analysis-Sentiment Analysis Part 5

Now, let's look at word analysis when we have polarities attached to words, so something could be more positive or less positive and we want to know that. We don't want to just say it's a positive or a negative, like good, best, greatest, awesome. You know, when someone says, this is awesome, and this is—it's okay, it's—okay is positive-ish, and awesome is like awesome, hopefully. So, we're going to use another technique, the Vader Sentiment Analysis. Vader contains the list of 7,500 words that are weighted, that have polarities attached to tell how positive or negative they are. And it uses these—so it also does—so it gives you a positive—for any passage, any sentence, it will tell you how positive the sentence is, how negative it is. So, the same sentence can have a positive and a negative polarity. And then it also computes a compound sentiment, which is really based on more than just the, you know, positive, negative difference. It's some heuristics that it uses to give a rating between negative '1' and

positive '1' on the sentence, so you get like a compound rating on the sentence, itself. And it's...it's contained on Twitter data, it's considered to be very good for informal communication, and it's—trained by using 10 humans who rated every tweet in context from negative '4' to positive '4', each feature in a tweet from positive '4' negative '4'. And then, these were used to compute overall score on the—on this thing here. and it also contains emoticons, punctuation, and other things that are used in social media, so it's really good for social media. Unfortunately, we won't be using it for that. And before you use it, you need to install it, so let's install it. And we got that. And now what we want to do is we want to import the sentiment intensity analyser from Vader Sentiment, and we can start doing the analysis. The analysis is going to be pretty much similar to what we've done so far. So, if you take all our restaurant data, right, and look at the positive, negative, neutral, and compound rating for each of them. So, what we're going to do is we take each text in turn, so that would be 'community', 'amigos', 'heights', and 'Le Monde', one—each one of them. And get the name, that's the first element, right? So, if you look at our texts here, for example, just to recap a little bit. So, texts—oh, sorry, that's the wrong one. Restaurant data. You have 'community' and then community text, right! And so, similarly, that's item one. And then below this is going to be restaurant data one, for example. Le monde and Le Mondes, the text, so that's what we get here.
We take each for 'i in range', we get—we get the name of the text, so that would be community or whatever. We get the sentences and we use a sentence to tokenize analytical feature to organize it. And then for each sentence, we get the polarity scores for that sentence using the Vader Sentiment Analyzer. And then—which is here, we create a sentiment intensity analyzer here, so—and that's this, right! And we get the polarity score, and then we take the polarity score and divide it by the length of the sentences for the positive, the neutral, the negative, and the compound, okay! So, that...so that we get some sense for how many, you know—based on we're getting the score for all of them and then we are adding them, cumulating them, for every sentence we have this thing, we divide by the total number of sentence and cumulate it in to pos, compound, neutral, or negative, right! So, if the first sentence has let's say point two and the number says 10, we're going to add a point zero two to pos, that's the idea here. So, we run this, and we find we get this stuff here, right! And we can functionalize this. Let me functionalize this first so that we actually have a better looking result and we get this over here. So, this tells us that in our world over here, community has a '0.20' positive and a '0.03' negative. It actually has the highest positive and the lowest negative rating of all of them. And neutral, we can ignore, and we get a high compound. So, you can see that this is not scaled, so it's like '0.20', '0.03', '0.19', '0.04', but it's '0.34', '0.28' because compound is using more than just the positive and negative, it's using some other heuristics as well. So, looking at this stuff, we find that 'community' is the most positive restaurant and, you know, that's kind of borne out with the fact that, to be honest, if you live in this neighborhood, you'll see that to get a table in 'community' at night is really, really hard. It's—you got to wait for a while and it's—they got...they got lots of tables, it's a big restaurant with outdoor seating as well. 'heights', 'amigos', and 'Le Monde'—you can always get a little table at 'Le Monde'. I like 'Le Monde' actually, but it's never difficulty getting a table there, so it has the lowest compound score. So, this actually reflects reality, to some extent, right! It's kind of nice.

Video 11 (10:42): Named Entity Analysis

So, that's what we do with this. We can do the same thing, if we like, with the presidential addresses, but we won't do that right now, and I'll leave that to you guys to do. The next thing we can do is we can look at 'Named Entities' inside a text. So, 'Named Entities' are very useful when you're analyzing documents because 'Named Entities', what they do is they are useful for identifying people, places, organizations, and those kinds of things. And—because, often what you want to do is you want to see a piece of text and say, is it saying something positive about so and so, or something negative about so and so. You're reading People Magazine, you know, is it saying something positive about Kim Kardashian or negative about Scarlett Johansson, you know that kind of stuff, right! So, you want to be able to identify those entities and work with them. And similarly, if you're reading a document that's, you know, a stock document, an analyst document and it's talking about different products and you want to grab those products and see, is the document saying positive things or negative things about those products? That's where, identifying the 'Named Entities' is a very useful thing. The NLTK provides a nice framework for doing that. So, what it really does is it uses—starts with something part of speech tagging. So, it looks at sentences and tries to tag them based upon what part of speech they represent, Are they nouns, verbs, adverbs, and those kind of things? And then, it takes the sentences and then chunks them together so that you get different parts of speech. So, you have noun clauses or noun phrases, verb phrases, and so it takes the sentence and breaks it up into its components and then tries to figure out based on these components, it tries to figure out what things represent people, places, and organizations. And, this could be more than one word, and they may even be separated by something else, right! But it'll try to figure that out. So, let's just run this and take a look at it. So, in our document here, which is 'community' data for the 'community' food and juice restaurant, it pulls out these things as 'Named Entities'. So typically, when you use 'Named Entities', what you want to do is you want to run through a large number of documents. So, let's say you're looking at analyst reports and you have a set of 1,000 reports. You want to take a few hundred of them and run through those, pull out all the 'Named Entities' from there and then eyeball them and see, do they make sense? And then, remove things that are not 'Named Entities'. So, for example, my guess is this is not actually a 'Named Entity'. The reason it got identified as a name entity is because it's a four-word uppercase—it's a four letter uppercase—four uppercase letters in a single word, so it thinks it's an abbreviation, might be an entity, I don't know. But, it probably is not. Bill Boston Bottomless. Maybe, Bottomless Margaritas, right! So, that's really where it came from. I'm not sure that this is a 'Named Entity' at all. It's probably just a mistake somewhere in that. And—but generally, it's pretty good, right! It's good 'community' food. It's figured out that's an entity. We can see that it's bringing out Brooklyn, which is an entity. Figuring out few— Fire Island Beer, which is three different words, but it's figured out that it's probably some kind of entity that—'Named Entity' of this kind of thing. Greendale community college, three different words. Troy Barnes. You know, so these are probably—it's done a reasonably good job of that. Now, the whole goal here of course is that once you have the 'Named Entities' you can use these 'Named Entities' to compute sentiments on them. So, let's say we want to—we know that service is a 'Named Entity', and you want to find out whether the restaurant has— is positive on service or negative on service. So, what we can do then is we could say, hey, we got our sentences—so we look at the sentiment on the— using a vader comparison to figure out whether the sentences are positive or negative, right! So, here and then maybe computer composite score on it. So here, for example, we've got the word service in 'community', and we can see that, in general the compound is negative '0.17' over here, but '0.84', '0.49', '0.49', '0.74' right! So, it's reasonably positive on service for this one here. So, what we've done here, of course is we've taken the sentences here that are—have the word service in them, any sentence that has the word service in it, and then figured out the—appended that to a list of meaningful sentences and then for those meaningful sentences, we've done a comparison and we look at those six sentences of these scores. So, you can do an affect calculator for all of these kind of things. So, let's say we've got—for example, a food item. So, we run this thing here, and what it does it'll—for service, it'll tell us the composite score is '0.432'. So, all we're really doing is we're using the same stuff that we did over here, computing a polarity score for each sentence, and then totaling up the polarity score. So, that's negative '0.17' plus '0.84' plus '0.49', blah, blah. And then, dividing it by the total number of sentences, which is six, and getting the average of that, which is an average compound score, really. And, that tells us that we have an affect on community data raw that says it's '0.43'. And, we can try this on 'Le Monde' data, and that's '0.22'. So, 'Le Monde' has poorer service than 'community'. We can try it on 'heights', that's '0.75'. So, 'heights' is very high on service rating. And, we can try on 'amigos', and that's '0.02'. So, if you're looking for service, you don't want to go to 'amigos', that's the bottom line, right! That's got a really low rating, at least for these 15. And, I should please point out that these are the 15 reviews of a subset of a large number of reviews, and when we include all of them, the whole thing could just change. But, you know, we get the idea here. So, that's the way we can use the compound score along with 'Named Entities' to try to figure out how it rates on various features. So, for example, if I wanted to look at brunch, right, we know that—well, we only see—actually, it's not in—is that a 'Named Entity'? Let's take another 'Named Entity'. And— yup, brunch is a 'Named Entity'. So, let's take brunch and see if you can work with that. So, if I do brunch here and look at community, I get a '0.43'. And then, if I look at 'Le Monde'—it's '0.25'. If I get—look at— 'amigos', '0.5', 'amigos' is good for brunch, who would have thought that, right! And then heights, it's '0.42', all right! So, this is—for each one of the 'Named Entities', we can do this analysis. And then compute, you know, based on how we consider—what we consider important 'Named Entities' for our documents, we can decide what the document is trying to say about those entities itself. The only other thing I should point out here actually before we go forward is that when you're doing the part of speech tagging, you want to use this thing called English pickle, which is really the grammar that is required for this. And 'nltk' has various such pickles for, I think, Spanish and a couple of others, I forget which ones, but you can look them up in your data, 'nltk' underscore data they sit around over there. And then the last thing we can do is we can say, all right, we know that we have service so we can—we've looked at it, we've found that service and community is good, but we want to see what they really do say about it. So, there's a nice function called concordance in 'nltk' that shows you the words around the number of characters. So, they're saying 100 characters around the word service in our document. So, this says the food, service and ambiance is exactly what we were looking for. Due right brunch hunger and slow service, that's the negative service that we found. Items on the menu, service is great, service, nice company. And, we can do this for each one of them, if we want to. And, that's just a way of checking to see whether these words make sense or not. Let's do 'Le Monde', and it says, we've never had a bad experience here.

Horrible service. Service was poor. Mediocre food, slow service. Oops. Awful service. I figured service would be quick. Better service.
Go a block down for better—that's not really a good place for service. Good service, yes, they were attentive. Quite good food and good service, make a returning customer. So yeah, it's mixed. OK. So that's the deal here. And notice, that I think—I suspect that our...our analyser didn't really find this very well. So if you look at—let me go back here. And if you looked at this here, and if I replace—oh God, this thing doesn't move. If I change this to 'Le Monde' data, and then run this here, which is our service thing here. So—well, there are quite a few negative ones, right! So these are our awful service ones. But I suspect some of these which look positive are probably more like our excellent for cat litter kind of lining the cat litter box kind of examples. Because the service looked, little better than this. So that's where we want to go. So again, just to recap, what you want to do is, we want to take out a document, look at the 'Named Entities', and then for the 'Named Entities' try to figure out whether the document is talking positively or negatively by the 'Named Entities'. And then if we know our domain and we know which entities you're looking for, we can actually see a lot about our document by looking at whether it says something positive or negative about things that we are interested in, in that particular document.

Video 12 (9:36): Text Summarization Part 1

The next thing we want to focus on is text summarization. Text summarization is a very useful thing, you can generate a short summary which of a large piece of text, automatically, kind of useful, if you have read lots of papers for a...for a...class or lots of textbooks or whatever. You can just summarize them and hope for the best. So, text summarization, really, most of text summarization works on something called extraction-based summarization. And, the idea there is to take the text and look for what sentences you consider important and then, report those sentences in your summary. It's not perfect, but that tends to—especially in structured documents, tends to work reasonably well. There are also certain advanced techniques called abstraction based, which try to abstract knowledge from the document itself, but those are more complicated. We want to focus only on extraction-based summarization over here. A very naïve sort of way, what you do is you identify the most frequent words in a piece of text and use the occurrences of those words in a sentence to decide whether the sentence is important or not. So, if the word, let's say, America is used a lot inside a text, then what you could do is you could say look at the sentences that use—say America, and that—those sentences become important. And, if another second most important is God, then you look for God. And then, the sentences have both America and God in them become the more important ones and they come on, and so on and so forth, right! So, God bless America would be, you know, in that kind of stuff there. So, we've got this, so what you want to do of course is import your stuff first, so we want to import word tokenize, sentence tokenize. We're going to import the Freq Distribution thing which is the frequency distribution of words, so that's kind of useful. We're going to import stopwords, which is words that are unimportant, like a, and, the, and all that stuff. And we're going to import ordered dictionary, which is a special kind of dictionary.

Remember dictionaries in Python are not ordered. So, they are not ordered at all. But, you can force them to be ordered by importing ordered dictionary. We don't really need that, actually, but it's there right now. Then, we want to prep the text, so let's say the 'community' data raw, I'm just going to move all the extra end of line characters from there. So, we get a bunch of things here. Striptext is really—we don't need the other stuff there. I don't know why it's there. Maybe I'll use it later I do use it later. Okay! So, I'm sending candidates sentences to an empty set and summary sentences to the—an empty list. So, the summary will contain the summary, candidates sentences are— candidate sentences and the counts are the counts of the—each sentence in the thing. So, we get essentially, like, set—candidate sentence 'x' has, you know, 20 times or 20 occurrences of useful words. So, the sentences that have the maximum number of useful words are the ones that will get included in our final analysis here. So, what we do is we take striptext, which is a text without our backslash ends in them and remove any stop words from it and any words that are not words, you know, that contain numbers and all that kind of stuff, we get rid of them. And, we lowercase it all so that we don't have a case thing inside it. And, I need to run this first. I haven't done this. Ok. So, that does our words generation for us. And, once we've done that then we can construct our word frequencies, and choose the most common end of those words here. So, we use the frequency distribution to construct the word frequencies and then, choose the most common 20 of them, or whatever you want to choose, and we can print them. So, let's take a look at that. So, it tells us that I is 73, the 46, food 20. So, our stopwords doesn't work very well, that's for sure. Brunch 20, good 19, please 15. These are the most common words in the sentence. So interesting.
Okay! So, we—we've lowercased them, we've done that. We can tokenize it and get bunch of candidate sentences. So, we take our striptext and we've sentence tokenize it. So, it figures out based on the punctuation where sentences start and end, and takes each sentence in that and makes a list order of it. And then, for each candidate sentence, we create a dictionary here. Let's say candidate sentence, this is a candidate sentence, and that's a lowercase version of it, right! So, we've got it here. It contains the sentence itself and then contains the lowercase version of it. The reason for this is very straightforward, when we report our summary when we compute our summary, we want to be using only lowercase letters. But, when we report a summary, we want to report it with the original cases. Remember, we're extracting actual sentences from there. So, we don't want, like, Columbia here to be lowercase and that, or America. We want it to make, you know, meaningful sense to the reader with the proper punctuation in it, So, this becomes our candidate sentences. So, these are all the sentences that we have. Next thing we do is we take the—candidate sentences. We have the long and short, which is really our, you know, uppercase, lowercase really. And then, for—in most frequent words, so most frequent words, let's recall what that looked like. insert set above.
Most frequent words, if you recall, is each word with the frequency of the words. So, that the list of tuples here. And, what you want to do is you want to extract from this the word and the frequency score of all the words. And, if the—a word is in short—Okay, so remember we have two loops here. The first loop is extracting this here. So, it takes a sentence number one in long and short. We don't care about long for a second, but let's say the short—the lowercase form of the sentence is here. And then, we take each word and its frequency. And, if the—each word—take a word like say take, coffee, for example. If the word coffee is in this lowercase sentence, then we add its frequency score, which is 12, to count, Ok! And then, we take that count and set it equal to the candidate sentences count start long

so that we now have—oh, this should actually be here, right! It's in the wrong place, this should be here. Yeah. So, we get the total count of the frequency score there and we get our candidate sentence count for long, which is — you might want to correct that in your thing here. For some reason, this got messed up. But, what you want to do is you want to compute this count, that is, a total frequency score for every word that is in the sentence. And then, throw that into the—as our total count for that long thing. So, at the end of this, we're going to get—let's take a look at it. You know what, let me do this here. We can take a look at it, candidate, sentence counts, we can print that, so we get here, I have a degree from Columbia and now I get one from America is total of 92. Jeff Winger community, 83. And we can see, you know, 15 minutes wait for the wait staff, 45 minutes before we got the wrong dish, 129. So, we have the sentences the uppercase version of the sentences with the counts of them. So, all we need to do now actually is sort this by the count. And, if we sort it by the count, then we can take the first end sentences and that becomes our naïve summary of this thing here. So, we can do the 'OrderedDict' for this. So, 'OrderedDict' is going to sort our candidate sentence counts dictionary and is going to sort it based on 'x''1', which is the second element, the number here, that's 'x''1'. Right, 'x''0' is that and 'x''1' is that, and make sure that it's ordered in descending order. Reverse equals to make sure it's ordered in descending order. And we get the first four sentences. That's what this is. It's going to put out the first four sentences and that's our summary, really, right! So, we get this here. I've come here several times, blah, blah, and that becomes our—these four sentences—and it's really up to us how many sentences we want. And, I guess the number of sentences depends upon the entire total proportion of sentences in the text or how many sentences there are in the text. So, this is coming off of brunch. There are nice places in the city, where you can get a very good breakfast of one-third to half less than 'community'. Brunch with my wife after she found a good review and it did not disappoint. The beans were seasoned perfectly. Tortilla was fresh and soft, blah, blah. So that's our summary, really, right!

Video 13 (7:13): Text Summarization Part 2

We can package all this into a function so that we can do this for all kinds of stuff. And, I'm just going do that here, and let me see if I get that in the wrong place there. Ya. That should be there, and right on that again. And then, I can—I'm going to use a joint function, what the joint function does is it takes a list...it takes a list and elements and joins them with whatever separator we give here, right! So, our list is build naïve summary community data dot raw, and that produces something. And, we join the elements of this list by using a backslash 'n' over the—this thing there. And, we print this summary and we get a summary which says—essentially what we saw before. And, we can see the Le Monde summary. This is a little bit slow because as you go through every different thing, it's— we haven't been using—we are not using a really efficient algorithm, but that's probably okay. So, I recommend for brunch or any other meal. Everything ordered for brunch has been quite good. Shakshuka is great, love to try them out for dinner. We checked regularly. Give them a better review because the food was good, but the waiting really was long. The food was really good, and they're known for their brunch menu. So, I mean, it's actually a pretty good summary, when you think about it.

They're known for—brunch is important, so is...is service. Okay, that's the deal there. And, we can now use this to summarize the speeches of our presidents. So, we know that to get George Washington's first inaugural speech, we just have to take this and we can build a nice summary of this. And, this will show you why George Washington has a large complexity. Because each sentence of Washington's speech is this...this is the length. So, this is pretty much his entire speech is there in the summary itself. Like, four sentences, and I'm pretty sure that covers almost everything in this thing there. And, we can try some other president. So, if you want to try, let's say, 2013 Obama dot text. We the people declare today most evident truths, blah, blah. just as it guided our forebears [inaudible]. And times change, so must we. We must harness new ideas in technology to remake our government, revamp our tax code. I mean this is pretty good, when you think about it. I mean, this is his speech, right! He's trying to say these are—Obama's, you know, points there on this one and this one. So, we've got a pretty good set from that summary, if you don't want to read his entire speech, which was quite long. And we can look at Trump as well. And, I haven't looked at this, so we shall see what it is. 2013 is not correct; 2017. Trump dot text. Now, joined a great national effort to rebuild our country and restore its promise for all our people. People of welfare back to work, American hands and American labor. It's a good summary, too. Factories shuddered left our shores, thought about millions. We've enjoyed all the glorious freedoms, we—I mean, this summarizes pretty much Obama's—Trump's mandate anyway. So, we can see that summaries, even our naïve summary worked pretty well with this. And, it worked pretty well because the documents are—you know, inaugural speeches tend to be very, very focused. So, if the documents are focused, you're going to get a good summary. There's another summary—summarizer which is probably a little bit better is the gensim summarizer, and what it does is it uses nodes and lexical similarity as weights on the arc. So, it's sort of looking at each sentence as a node and then, it looks at how each sentence is related to other sentences and then, the nodes that have high relations with other sentences are more likely to be in the—you know, reflective or the document contains, so they show up in the summary. So, it's worth looking at this gensim to take a quick look and see how it defers from our naïve summarizer. So, we'll just do all our imports. Oh, I shouldn't have done that. Oh, maybe I did it. Doesn't matter. Import star. And then, I run all my stuff again. And, I just want to make sure that community data is there. It's plain text corpus reader. Ok, just to make sure we have all this stuff there, Okay! And now, what I want to do is import gensim summarization. And, I need to import install gensim. Ok, so let's do that. I'm going to go up here. And, so it's set above. So, let's make sure you import gensim—install gensim as well. So, once you've installed that then, we can import gensim summarization and then, summarize our community text, print summary. Oh, this is actually 'Le Monde' text, right! That's what we did here. So now, this tells us here and—so I should compare it with this. So, that's our naïve summary versus this one. You'll notice one thing that gensim was much faster, much faster than our naïve summary. because our naïve summary, is your report algorithm. So, here we have the slight differences in sentences. So, it says I've come here several times with friends for brunch, and—Actually, wait a sec, that is community. I'm sorry. I should be using community there. Yeah, striptext was community, right, not 'Le Monde', so you do that. So, here we see the first sentence is pretty much the same for both of them. And then we don't have this country breakfast and eggs were delicious, fluffy, over there, but we have—we don't have the pancakes thing. And, really good for breakfast, great pancakes, sausages, eggs. Actually, in a way—I guess—it's hard to say, but I would, you know, put this as a slightly better summary than this one. So, gensim is doing a little bit better job with this here. But these documents are small. So, it's well worth using that down the road, if you ever want to do it, you can use genism rather than community. You can try the same thing for 'Le Monde' or for gensim rather than the naïve one. Try the same thing for 'Le Monde' or for 'Amigo's' or for 'heights'.

Video 14 (10:57): Topic Modeling Part 1

Now, let's look at topic modeling. Topic modeling is a little bit more complicated than anything we've looked at, because it requires a lot of input from the reader. What...what Python can do is it can give you a sense for what the main topics of a document are. But, you have to make sense of those topics. It'll give you a bunch of words and say these are the words that are likely to be going together as a topic. And then, you have to say does it make sense to me or not. So, we're going to use topic modeling over here, that's a simple example. What topic modeling does is... topic modeling does is it uses unsupervised learning. You don't need to know anything else before that, and no apriori knowledge is necessary, so to speak. And, the algorithm used is called a Latent Dirichlet Allocation Model which is–it...it–it's sort of...it's there's a lot of mathematics behind it. But, the idea is it computes conditional probabilities for topic word sets and then, identifies the most likely topics as a combination of words. So, it's taking words, looking at the occurrences of these words and then, seeing how closely they go together and then, using the words that go together, for computing probabilities that they are likely to be topics and then, using that as a, the probabilities as a...as a way of figuring out which sets of words are more important, than which other sets of words and then, those become topics. And this, it does it over multiple passes. So, initially just randomly picks stuff and, sort of, guesses at it and then keeps pruning and refining until it gets it. So, there's a lot of information about it you can look up on this website over here. And—but, if you do more than the micro masters, then you probably would learn more of these things on the way anyway. But, for now, we'll just assume that it does something really nice for us, all right! So, what do we need to do? We need to import again—it's gensim includes the LDA model. And, we're going to import gensim we want to import Corpora. We need an 'LDA model'. We need to import 'stopwords'. And, we will import preprint of course for preprinting stuff. So, the first thing we do is prepare the text. So, one of the texts that I have in our data set here is a review of the Nikon Coolpix 4300 camera. And, we're just going to use that as an initial topic. Well, that's probably the easiest to deal with. And, we again clean it up a little bit.
We get a script text that is cleaned up on this thing. And then we take our words, remove anything that's in stop words here, remove anything that's a number by doing this, and make sure everything is in lowercase, and we get a text that is a bunch of words. So, let's take a look at this. It does it at 360 words in texts. So, we can take a look at texts, as well. So that's what it is, right! So, it's telling us, we have here, oops. The sentences that we have are sentence one, contains 'annotated' 'minqing' 'hu' 'bing', which is the author of the review by the way. So, we want to ignore all this 'computer' 'science' 'university'. So, this is the best four NP compact digital available camera perfect enthusiastic amateur photograph, right! The—after thinking of the stop words, we get a sentence here. So, our text is actually a list of sentence words. Each–a list of lists and each sublist is the words in a sentence. We get this. Then, we create a

dictionary and a corpus. So, let's take a look at what these things are here, let me run this again, let's see, the dictionary, the dictionary here is a, a dictionary dot—actually I have all that there right! Why am I doing this? So, the dictionary is the—every word in the text is given a number. So, annotated is given the number 0, 'minqing' the number 1, number 2, number 3, etc. So, the goal here roughly is to construct sentences that are these numbers instead of the actual words itself, because like I said earlier, what LD is doing it's creating a network of these sentences, and we'll see how closely related they are. So, if it says, sentence one is word 7, 14, 23, 24, 25 and sentence two is 7, 14, 6, 9, 25 then, it knows that there's a linkage of three of the words are connected in those two sentences. So, rather than looking at words, it looks at numbers, it's much faster, right! So then, you—text is much harder to handle. So, what dictionary, the dictionary that they're computing here using Corpora dictionary text is all it's doing is it's replacing the word by a number, and this is keeping track of what the linkage between the word and the number is. The next thing we have is so if you look at the keys, the keys are essentially guess. Okay, wait I need to, comment that out because you're going to see the other thing.
The keys are just 0, 1, 2, 3, 4, right! That's the idea here, and then if we look at the corpus, the corpus is now the dictionary, the...the numbers of the words, so it's like text except that it's replacing the words by numbers. So, it's telling us that sentence number 9 has occurrence of 17, words number 17 occurs once, word number 66 occurs twice, word number 73 occurs once, word number 72 occurs once etc, okay. So, if we look at word number 63, or if you look at the text lines just to see what the words are in that, text 9, these are the words and in text number nine, okay! and then if you look at word say 73, that's included. Actually, wait a sec, I've got the wrong one here. Let me do corpus 9. So, we want to look at 66, and see what the word for 66 is. And that's rechargeable, and we can see that the reason that's coming twice in this is we are rechargeable here and rechargeable here, so this sentence is rechargable twice. So, that's why we got 66 comma two in our corpus. And, that's the...the whole linkage here, right! So that...that makes sense, at all So, that's the goal with all this, is to replace our sentences by–in a–the words in a sentence by numbers and then, create a corpus that has tuples, where you have the number of the word in a sentence and the frequency of its occurrence in that sentence, right! So, rechargable occurs twice in this, so that's an important word in that sentence. That's the rough, vaguely rough idea. So, that's it.
Once you've got all this stuff together, we actually do the LDA itself. And to the LDA, we want to tell it how many topics we want and how many passes we want to do. The number of passes is kind of important because it's...it starts with a very rough estimate. So, the more passes you do the better you're going to get. But, of course, there's a certain period after which you're not going, it's not going to help you at all. So, you really want to be able to run this a number of times and figure it out. So, let's see what we get here. So, we said 5 and 10. I'm going to run this and that. it'll take a couple of minutes, maybe less than a couple of minutes, it should be pretty fast. And, once we get that we will print the results. So, we get these results here. So, we are telling it over here that we have five topics, so there's 0, 1, 2, 3, 4, —0, 1, 2, 3, 4, that's the five topics. We're telling it over here that we want to print only three words per topic. We can ask for more if we like. So, I could do five words per topic, and then I would get five words per topic and this—the topics here are 'camera', 'pictures', 'card', 'manual', 'digital'. And each of them has a weight attached to it which is probability that this camera, word 'camera' is in this topic. This makes a lot sense because camera is, you know, part of the, it's a... it’s a

camera. So, you will expect camera there. So, we can ignore camera in a little bit. In fact, you might want to take it out of our topic list completely, and add it as a stop word to this thing, so this is 'pictures', 'card', 'manual' and 'digital' right! So, it...it tells us– let me go back to three actually, because that's a little bit clearer I think, so this is about pictures, this is just a very general camera thing.
This is about the user of the camera I guess. This is about the battery life of the camera or the battery of the camera. It tells you something about the battery. So, the topics here are pictures, battery, 'picture' 'use', 'picture' 'card', 'picture' 'use', 'pix', and we probably should be stemming the words as well, because if you stem then, then this pics and pictures will probably come in the same thing here. But, we have 'battery' and 'use' as two reasonable topics over here. But, the point being that you need to go back and look at the topics and try to make sense of them. So, how do we match topics to documents? Well the idea roughly here is that what you do is you let's say you have a 1000 different documents, and you have a new document coming in, you, what you want to do really is you want to take your initial set of 1000 documents, split it up into a training and test sample, take the training sample, look at that for possible topics, apply those, you know, do some analysis of the club you've been doing, and figure out where the topics are, apply those to the, take the documents in the test sample and run them through the topic analyzer to see, given the trained set of topics, what topics show up inside the test one? and see if that makes sense. I mean, you have to do some work up front, right! And then, what you want to be able to do is that as new documents come in, you want to be able to say what are the main topics that this new document has using the analysis you've done before? Obviously, you know, you'll have a domain, and the domain has bunch of topics that are of interest to you. And, some of them will be, some documents will—let's say, you have ten topics that are of interest. Some documents will concentrate on topic 1 or 2 and others will concentrate on 5 and 6 and, you know etc, etc, right! So, it's not a question of taking one single document and deciding what topics it has, but it really is a question of taking a large number of topics and deciding what are the range of topics that they cover, and then when you get a new document, you can figure out which particular topic or topics that new document addresses.

Video 15 (4:35): Topic Modeling Part 2

So, what you want to do is—and here we're using sentences at documents. So, it's probably less than ideal, but what we can do is we can take our documents here and sort them by topic 'Id' here, and we give it one document here 'corpus[0]'. So, 'corpus[0]' is the first sentence in our document, right! Is that right? Yeah, the first sentence where the 'corpus' is a sentence by sentence, and it says that the first sentence is focused on topic number 4, which was camera picture use. That's really what it is. And, let's say 'corpus[1]' is on number '1', which is camera, pictures, card, 'corpus[2]' is on '4', which is camera, picture, use, 'corpus[3]', is again on '0', camera, pictures, card, or, you know what, yeah...yeah. So, we are giving a minimum probability here '0', which says that you don't want to include anything that has a probability of less than '0'. So typically, you want to do better than that and that's really trying to be sort by the probability, which is this and we get sorted by '0.727' etc. So, it tells you which topics are working for that. We will take a better example down the road but for now, let's just move on to this and let's

draw word—a word cloud, which can also help us a little bit. So, we...we look at our topics, they are all listed with these words over here. So, what we can do is, we can draw—for any given topic, we can actually draw a word cloud and see, you know, there are many words here, right! So, if I do here 'num_words' equals '30', then I get lots of words here. So, I can actually take all these words, weight them by their probabilities, and use that in a word cloud, right! So, a weighted word cloud based on probabilities here. So, that's what this does over here. So, I can use a word cloud example and we run this and draw the word cloud. This is for topic number '2'. It will tell us that the word cloud is the only thing we're doing here is we are expanding it so that we get the frequency of the words and the topic based on its probability, right! So, probability here, that's what we do there. So, this tells us that topic number '2', the words there are quality, settings, use, digital, pics, camera.
If you look at topic number '0', it tells us camera, manual, digital, this is more about the camera itself, right! So, it's camera, manual, features of the camera versus use of the camera. So, we have two different topics over there, which we can roughly sort of intrude from this set of words over there. So, let's look at presidential addresses to see what sort of topics emerge from there. And, what we can do is we can take each document and make that, you know, look at the topics per document rather than looking at documents per sentence. So, what we do here is, we take all our files in 'inaugural. fileids ()' and each document there then becomes a document in our 'corpus'. And, we get a 'corpus' by address which is similar to what we did before, which contains all the...the addresses that we have for the presidents and each one of them is a document there. We can see the length of this thing here. If I look at length, that's '58', which is the number of—in presidential inaugurations we've had so far. So, we create the model and do the same thing we done before and run this. It's going to take a little bit of a longer time because there's a lot of data but not that long I guess. and we get these topics here. So, this tells us some roughly what topics we have, course, things, knowledge, and, you know, I guess we need someone well-versed in presidential inaugurations to figure out what these topics mean. But we are not going to worry about that, we'll just assume that we have these 9—'20' topics—possible topics.

Video 16 (6:54): Topic Modeling Part 3

And, we can now compare the presidential addresses by topic. So, what we can do is we could take Washington's first inaugural address, and it tells us that it's mostly '19', topic '19', which is 'power', 'right', 'years', 'protection', 'experience', 'future', 'subject', 'trust', 'state', 'support', whatever that is. We can take Trump's topic here, and that's '57', it's the last one. And that's '11'. So, his topics—most important topic is 'world', 'power', 'years', 'children', 'things', 'future', 'change', 'right', 'moment', 'place'. Interesting! And, you know, I guess what I'm trying to say is I don't know what these topics mean. We need an expert in presidential addresses to figure out what these topics are. Let's take—Obama would be '55', right! '55', '56', '57', would be his first—oh, he also has '11'. So, Obama and Trump have pretty much similar topics in their addresses in the first address. And we're on the first one, but the second one goes to '8', which is 'journey', 'requires', 'belief', 'carry', 'created', 'years', 'meaning', 'capacity', 'lasting', 'person'. No idea what that is. But that's roughly what you want to do, right! So, you have a bunch of documents, you figure out what these topics are—and you really want to give names to these topics.

They have to have some kind of intuitive sense that makes sense to you in your domain. And then, you can look at any new document that comes in and attach to it the topic that is most likely to address what that person's saying. Actually, here we have two different topics, '8' and '11'. So, Obama didn't change that much. '11' was his first one and '11' and '8' are the second one. So, we have zero point five one and zero point four eight. You want to look at the probability numbers. So that's roughly what you want to do there. And you can draw a word could for any of them. So, let's take '11', for example, which is we saw was sort of important, and draw a thing for that. And while it's drawing, we can move on here. 'World', 'change', 'future', 'forward', 'power', 'friends'. Okay, that's the stuff there. I think another thing we probably need to do is to remove a lot of these words from it. I think, like, 'world' may be the right, but 'friends', 'things', these kinds of words we should probably remove from that. So, in the example that I set up here, I've actually removed some words, I think. Let me see here. Yeah, I have a list of words that are removed, but you can remove more or better or, you know, I really need domain knowledge for how we deal with this stuff here.
The last thing we want to look at is the similarity of documents. It's very similar to LDA here. And what we want to do is I'm going to do a very simple, quick example because this class has become really long here. We have a list of documents community, le monde, amigos, heights. We have all the text for that, and we want to compute the corpus and the dictionary just like we did before. So, we got all that stuff. So, we run that. And now what you want to do is we want to use the genism similarities module from genism because of similarities module. We give it a new document. And I'm going to give this document here which has a random review that I pulled out of the...the database. And this is a review for a restaurant called 'French Roast', which is a different restaurant in Manhattan. It's not near Columbia. And it is—I believe, I'm not 100 percent sure, but I think it's run by the same company that runs le monde. So, I've just taken that as an example, right! And what we want to do then is we want to see whether it's—what is it similar to? So, if I run this, I created this—So I get this similarities from the lsi similarity corpus here, which is our corpus. I run the lsi model. And then I get the similarities, and that's what this does here. And if I need to get 'sims', which are—I'll show you what they look like. They look like this. So, this tells us that the document—this text that I have given here which is my document is most similar to number 1 by '0.98'. Pretty similar to number 0, '0.95'. So, this is le monde. Number 1, this is le monde. Number 0 is community. We can see that here. So, it's community, le monde, amigos, and heights, right! So French Roast is very similar to le monde, most similar to le monde, also, pretty similar to community. Less similar to heights and amigos, which is makes sense because it's not, you know, a Mexican restaurant. So that's roughly what you want to do here. And we can try a second document here, which is called Mexican Festival restaurant, which is a Mexican restaurant nearby. And we run the theme analysis for that. And it tells us it's most similar to amigos, which makes sense, Mexican. Heights, which makes sense because it's a—heights is also, you know, a kind of Mexican restaurant. But it's also kind of similar to le monde and not as similar to community, right!
But we see that this makes sense somewhat. Because the most similar that it is giving the highest similarity rating is giving us to Mexican restaurants, which is exactly what we want. So that's it for text mining. And we've seen a lot of stuff here today. I want you to think about these things because what we learned from this is that we can do a lot with text. We can find sentiments, we can figure out whether it's saying positive or negative things in document. We can find out whether they're saying

positive or negative things about things inside the document. We can summarize the text and give a brief summary of what the text does, which is very useful when you have to read hundreds of documents, long documents, and you really want to get only the gist of what they're saying. You can figure out what they're most—what any arriving document is most similar to in a set of documents, which is very helpful when you want to find something that's similar to it. Because if it's similar, then it will be saying similar things. You can find the topics that a document covers and the topics you can use to figure out what, you know, a new arriving document is talking about and essentially whether it's worth looking at. Because reality is that we get lots of information. Information is coming in the form of text nonstop in our news feeds and all kinds of stuff. And we don't have the human capacity to read them all. So, what you want to be able to do is you want to be able to filter them. You want to filter them by topic, by sentiment, by similarity, by all these kinds of things, and that what feeds back into making life, you know, easier for us in an information sense. So, text mining is really this umbrella for all these grabbing information out of text kind of things.