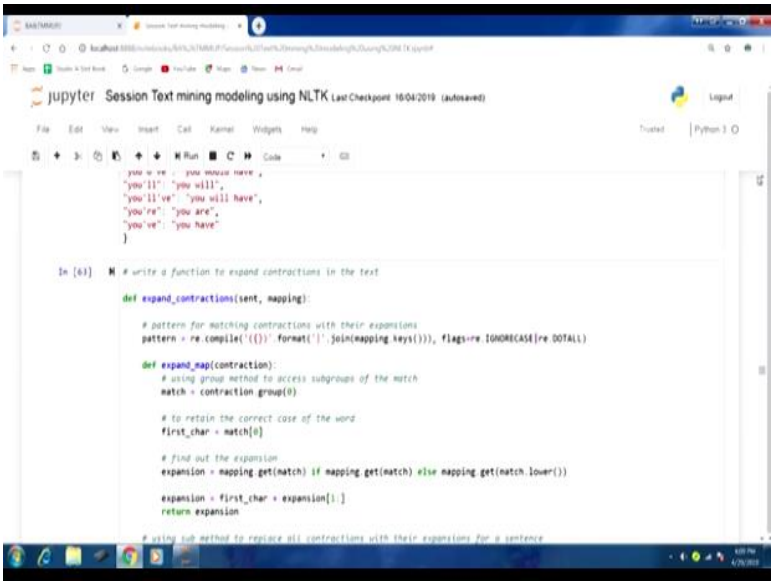
Business Analytics &Text Mining Modeling Using python
Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology Roorkee

Lecture-39
Text Mining and Modeling-Part II

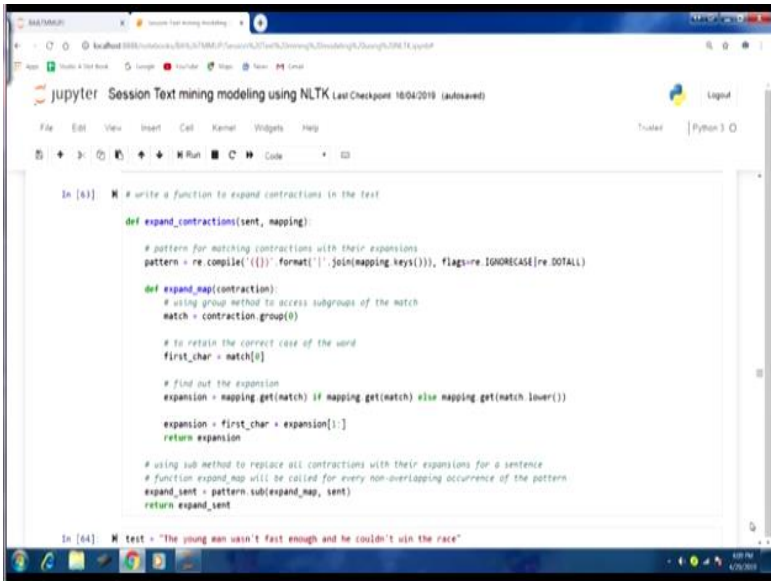
Welcome to the course business analytics and text mining modeling using python. So, in previous few lectures we have been discussing some of the steps that are required to process the text convert you know transform the unstructured text into a form where we can you know do our text mining modeling. So, specifically we were talking about expanding contractions, so we talked about that aspect, we talked about the mapping you know contraction mapping that we discussed in the previous lecture.

(Refer Slide Time: 00:52)



So, let us go through quickly briefly this and then we will continue.

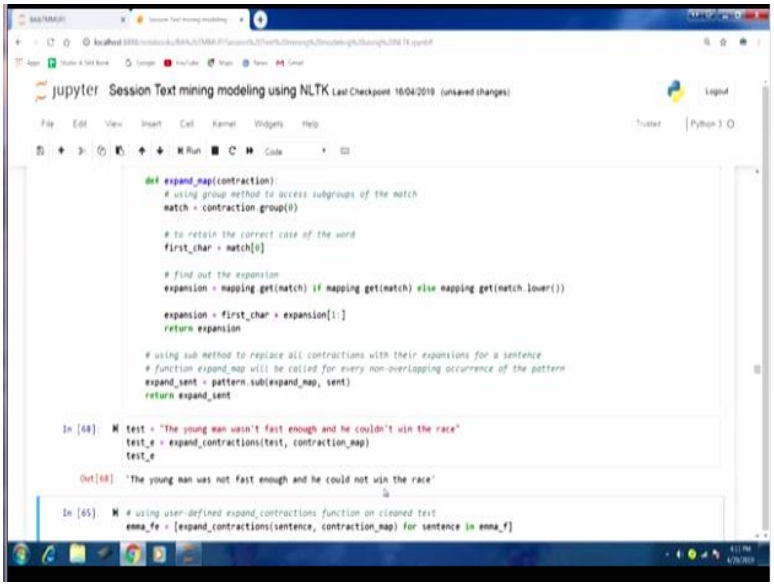
(Refer Slide Time: 01:00)



So, here we are defining a function that can be use to expand contractions in the text, so you can see define expand underscore contraction and any sentence you know text and then mapping. So, mapping that we discussed before we have a dict object there, so we can use that to perform of our mapping here. So, here we are again using pattern for matching contractions with their expansion, so you can see that you know this pattern here.

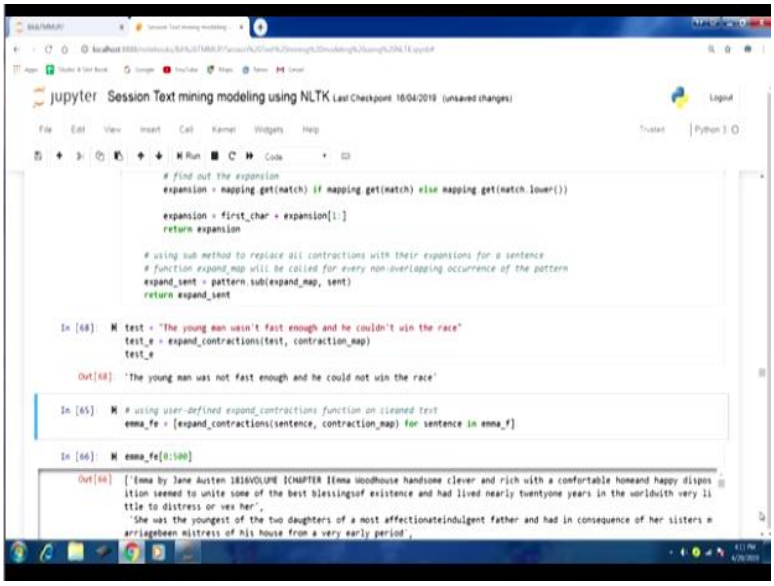
And we will be using this pattern to actually find out all the contractions with their expansions for a sentence, you can look at this part of the code. So, here I am calling you know this using the pattern dot sub I am finding out all the occurrences here you know expanded underscore map and this particular this function is being called which will actually sense essentially it will you know do the mapping, so this part we have discussed in the previous lecture, so this is the function that we define to perform this.

(Refer Slide Time: 02:09)



Now once we through this then we can apply this expand underscore contraction in one of the text. So, I have taken this sample text here you can see the young man was not fast enough and he could not win the race. So, was not and could not are the 2 contractions that are there and let us use our function and let us see whether they are you know converted into their expansion form, so let me run this and you can see in the output that you know was not has been converted in to was not and could not has you know has been converted into could not, so you can see the function is working quite well.

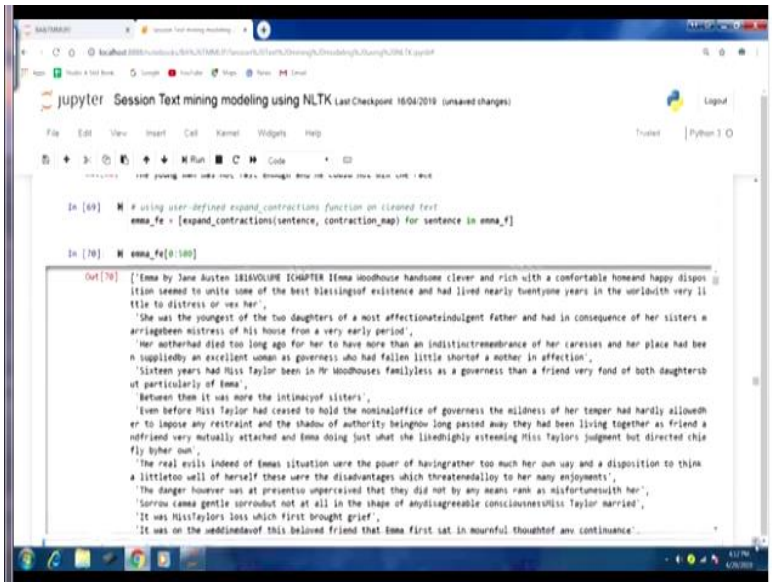
(Refer Slide Time: 02:42)



Now the same thing we can apply in our this you know this text document that is Emma file that we have. So, we can use this user define expand underscore contraction function on clean text M

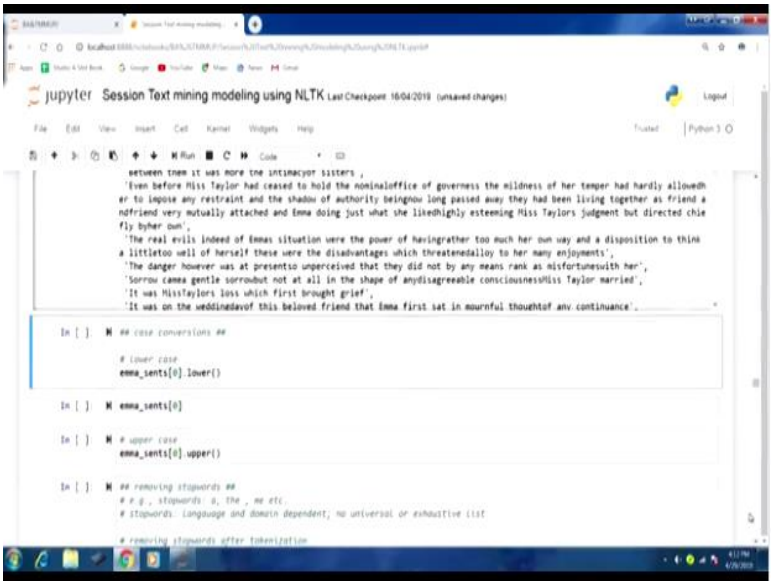
underscore F that we had cleaned in previous lectures.

(Refer Slide Time: 03:00)



So, if I run this and you can see now we have Emma underscore fe after this expansion process. And if you will have a look at the first you know 500 you in the sentences here, so many places you would we you know if you compare this output with the am Emma sentence output.

(Refer Slide Time: 03:19)

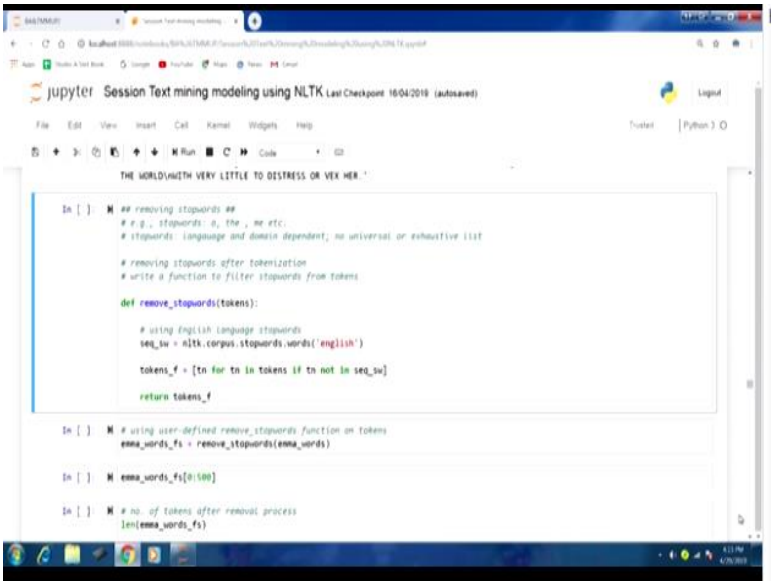


Then you would be able to identify that many places where you know contractions were present, now they have been map to expansions. So, in this fashion in one go we can actually perform this, now let us move to the next aspect that is case convergence. So, sometimes if you might require you know certain case convergence to be performed. So, for that we can just take the you

know we have taken this example if we want lower cases.

So, we can type like this M underscore sense and 0 just for an example you know in the first sentence and dot lower, so we are calling this lower method and it will convert all the you know text into lower case.

(Refer Slide Time: 04:01)



If I run this will have this you know convert into lower case, this is we can compare, similarly for upper also similarly you can see all the text can be converted into the upper case at as well. Let us move forward now I will talk about another aspect that is removing stop words, so what we mean by stop word essentially it is about the terms like a dha, dhe, me. So, these words which are not you know really meaningful in the text analytics context in the test you know classification you know context.

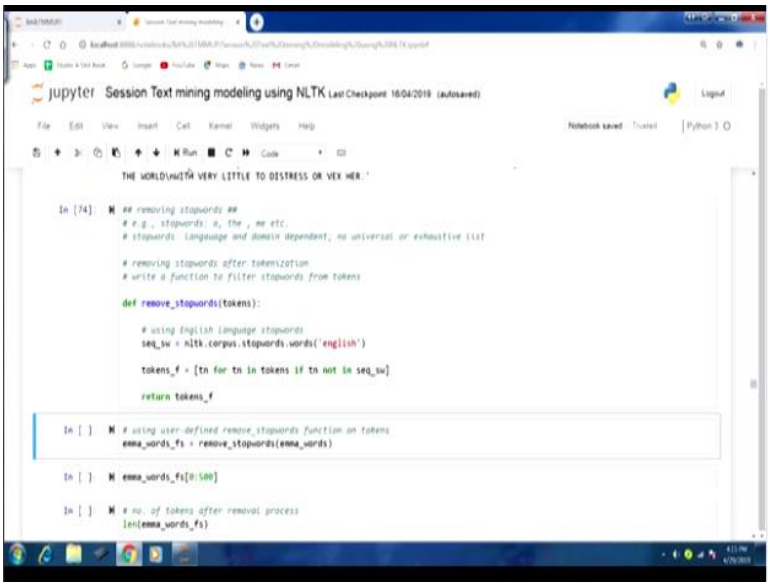
So, we might prefer to remove these terms and you know because we would like to later on I will discuss will like to identify relevant features, relevant vectors which will help us in our text classification problems. So, therefore some of these stop words which are not going to add much in that context we would like to get rid of these terms. So, stop words again the regional language and domain dependent, so depending on the you know language English typically you know these stop words differ from language to language and also domain you know.

So, certain domains, certain words might be meaningful and certain you know and another domain they might lose you know importance there. So, however we do not have any universal or exhaustive list in this case still we have for English language will use this NLTK you know module. Here we have a list of stop words, so that will be using to get rid of these words from our corpus.

So, typically removing stop words after tokenization because it makes the process slightly easier for us. So, we will talk about writing a function to filter stop words from tokens you can see now we are defining the function deff remove and of course stop words tokens will be passing tokens here and then we will use English language stop words here. So, you can see we are creating this sequence underscore SW this list of the stop words and NLTK dig it dot corpus dot stop words dot words and English.

So, we will have the list here and then we are running this you know list comprehension tokens underscore F. So, we are filtering these tough words, so that is why underscore F, so tn, so here you can see tn for tn in tokens if tn not in sequence underscore sw. So, if a particular token is not already present in the list of stop words then you know will have that and our filtered tokens list, so this is the functions.

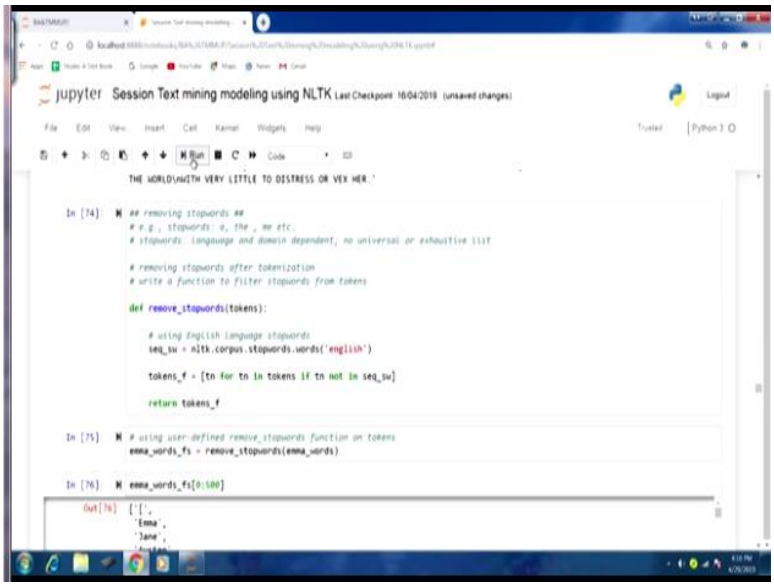
(Refer Slide Time: 06:53)



So, if I run this and now we can apply this particular function in our Emma underscore words, so

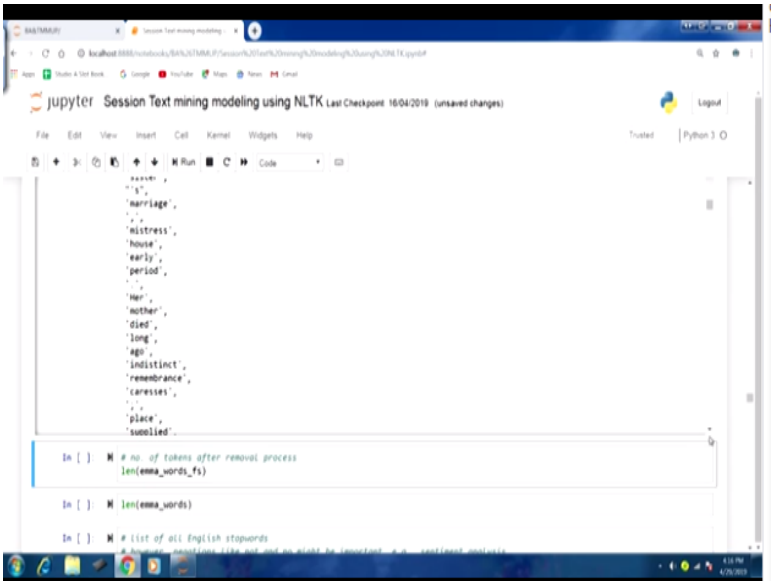
will be using this user-defined remove underscore stopper function on tokens, remove underscore stoppers, M underscore words, these tokens we have already created.

(Refer Slide Time: 07:10)



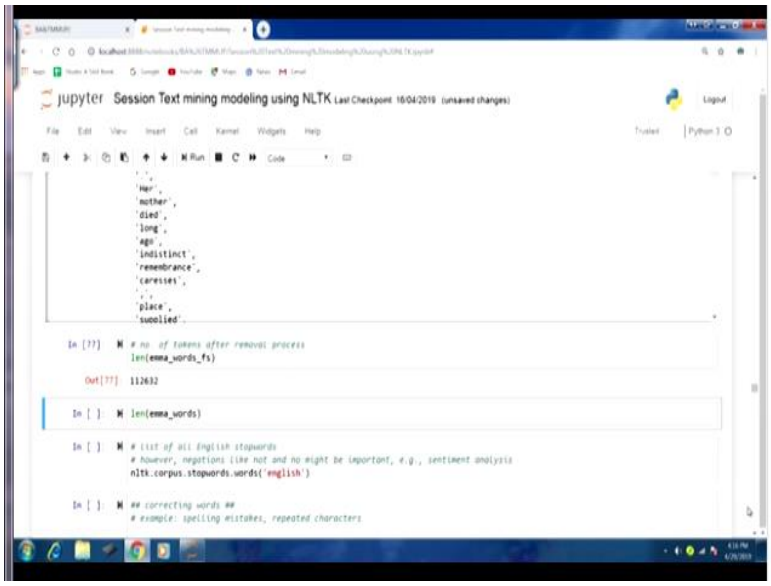
So, if I run this we will have Emma underscore words underscore FS let us have a look.

(Refer Slide Time: 07:16)



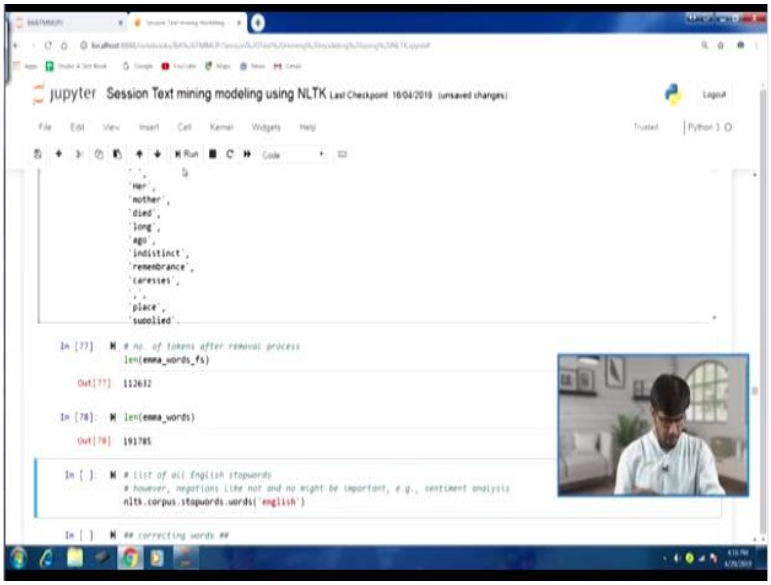
And in the output if I scroll through this you can see if you look at Emma, Jane, Austen 1816 and you would see that all the you know stop words a and these kind of terms are gone. If I scroll through this you would see only meaningful words have remained and you know so all others are gone. So, let us have a look at the number of tokens after removal process, so we can use this learn function.

(Refer Slide Time: 07:51)



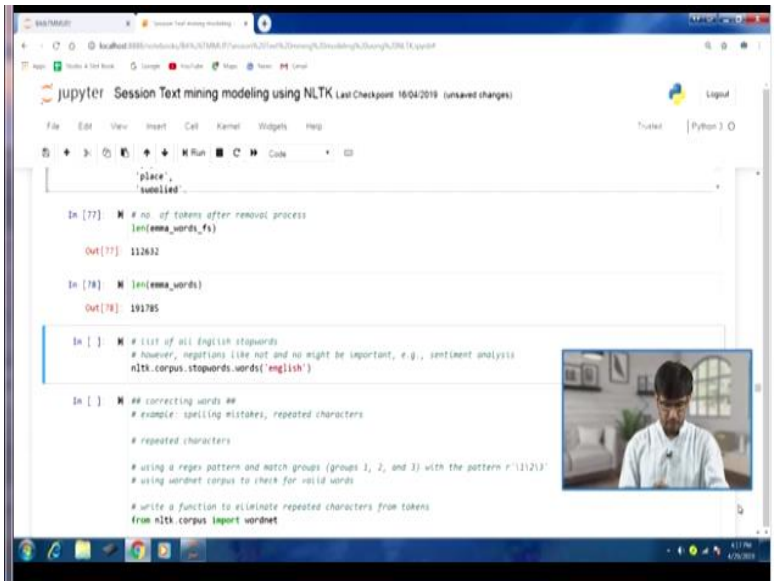
And you can see the number of words after this stop word removal process we have one lakh twelve thousand six hundred thirty-two.

(Refer Slide Time: 07:58)



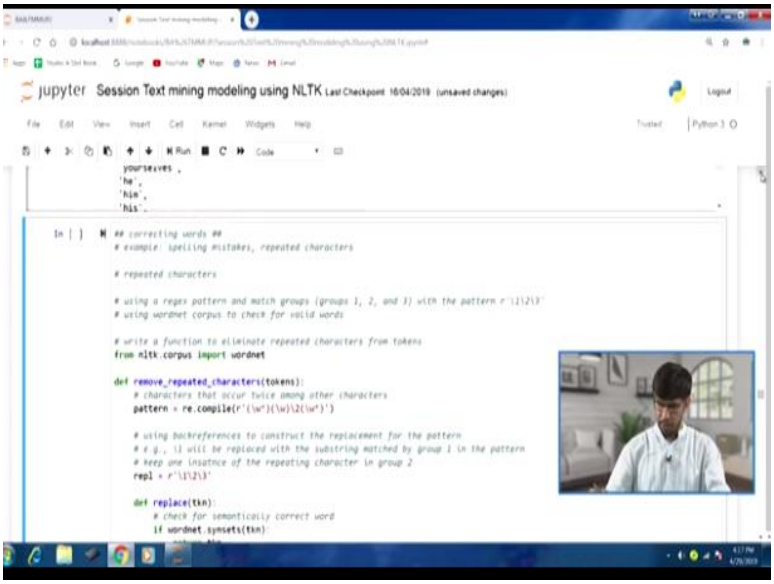
And if we look at the original list here, so you can see my one lakh ninety one thousand, so approximately about you know eighty thousand words we have lost just by removing stop words, this is the reduction in size that we have been able to perform.

(Refer Slide Time: 08:17)



Now if you are interested in having a look at the list of all English stop words, so you know you can again print this list NLTK dot corpus dot stop words dot words English.

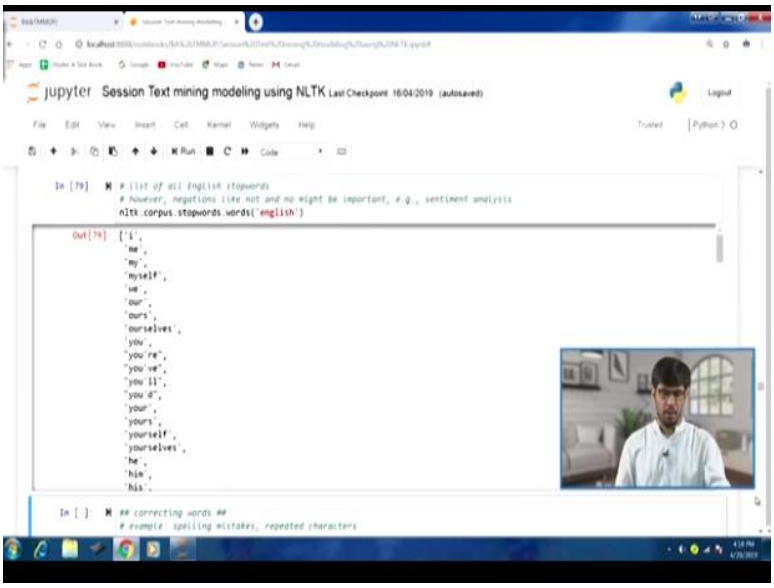
(Refer Slide Time: 08:30)



If I run this yeah and if I go back to the output you can see these are these could be some of these stop words these are some of the listed stop words for English language you can see I, me, my, myself, we, our, hours. So, these are some of the words which are not which might not be meaningful for our text and analytics context or text classification problems. So, these are the stopper that we typically would like to remove, so the same thing I mentioned in the comment part also.

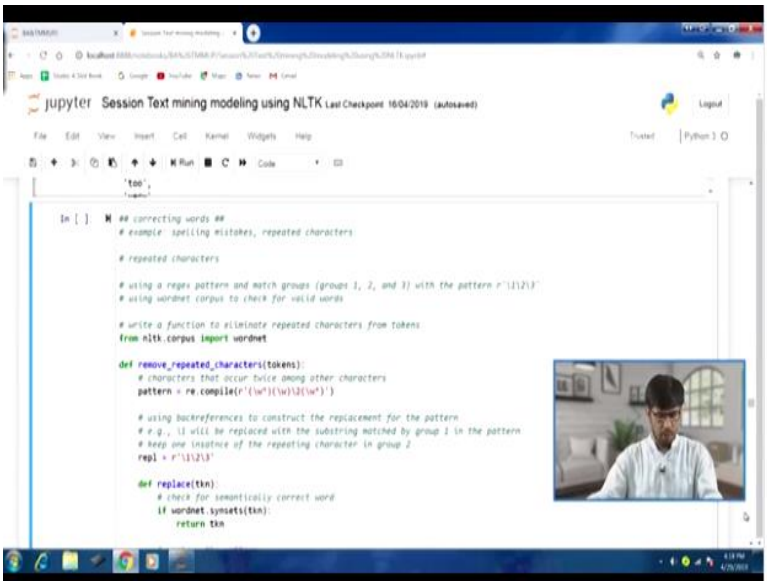
So, however there might be certain exceptions also for example negations like not and no they might be important sometimes, specifically we are applying if we are you know doing sentiment analysis. So, there sentiment analysis would also involve you know we will have negation aspect.

(Refer Slide Time: 09:23)



So, therefore these words might become important there, so we will have to really see you know whether you know we can really remove you know these you know stop words whether we can afford no and not in these stop words .So, right scroll through we might encounter however if that is the case you can see here no, nor, not here. So, they are right now in the stop words but if we are performing sentiment analysis probably will be better off you know not removing stop words. Because he might lose no were not or we can prepare our own list, own stop word list.

(Refer Slide Time: 10:03)



Let us move forward, so another aspect that we might encounter in processing text and transforming text is correcting words. So, there could be spelling mistakes, there could be you know repeated characters sometimes. So, especially informal writing you know if somebody is writing in a social media platform or in the comment section of e-commerce website they might write you know who emphasize you know certain words they might in a repeat certain characters.

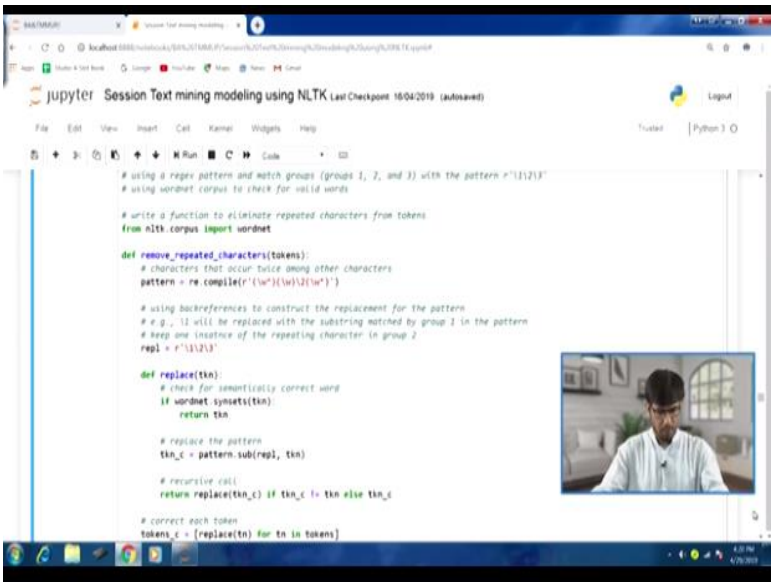
So, for our you know text a analytics purpose if we want to remove those you know we want to you know correct those words, so how we can perform that .So, let us talk about this, so we will pick up the repeated characters scenario, so here we will be using a rejects pattern and you know different match groups 1, 2, 3. So, you know pattern like R you know in within single quotes slash1, slash2, slash3 this is essentially we can use different groups in a pattern.

And we construct another pattern you know that this would be really useful for us in repeated character scenario. So, let us have a look at the function here, now here it will also be important for us to check the check for valid words. So, you know for many of these words for example repeated character for example finally and there we have Y repeating many times or L repeating many times.

So, where do we stop while we are trying to remove some of those repeated characters, so for

that we will take we will have to take the help of word net corpus to actually check whether you know in that removal process whether we have at least valid word you know case.

(Refer Slide Time: 12:00)



So, that will have to perform here, so let us have a look at the definition and definition of this function remove underscore, peter underscore characters. So, we are passing tokens here, so now will be in the pattern that we are going to use here is this characters that occurred twice among other characters. So, if you look at the pattern we have used parentheses to identify different groups here.

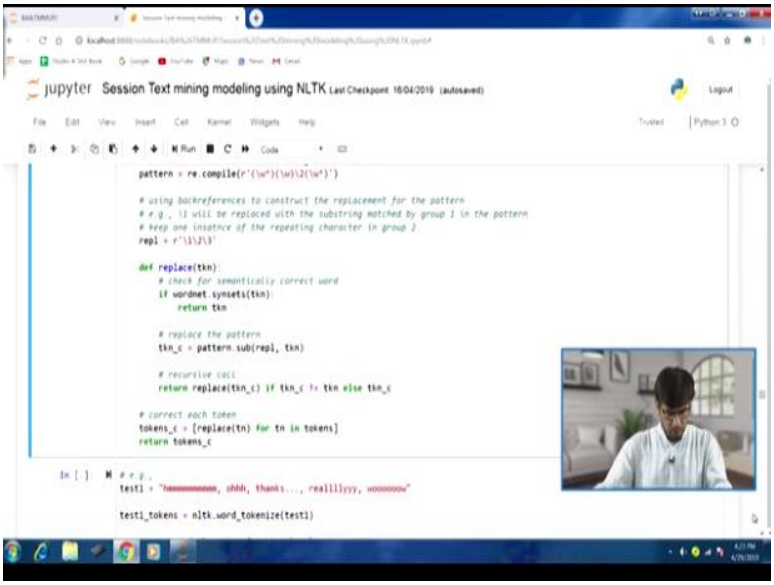
And you can see the bit the second group you know this is slash slash 2 indicating that you know occurring twice. Now these are you know are back differences, so will be using back references to construct the replacement for the pattern. So, one thing is to match a particular you know these kind of repeat words with repeating characters. The another thing is to replace them all eliminate the repeating characters.

So, here when we say slash 1, so slash 1 will be replaced with the substring matched by group 1 in the pattern. So, slash 1 is actually a back reference to the group 1, slash 2 is a back referenced to the group 2. So, when in the in the first line pattern Li dot compiled there you can see slash 2, it is actually referring to it as a back reference to a group 2 which is nothing but slash w, so the same what is repeating here.

So, that is why in the comment section we say characters that occurred twice, so in this in this fashion we have defined and that would be used for matching and then for the replacement. So, if you look at the you know pattern that we are using for replacement but keep one instance of the repeating character in group 2. So, you can see REPL this is our replacement pattern or within single quotes slash 1 that is group 1, then slash 2 that means just you know one instance of that you know that repeating character and then slash 3.

So, slash 1 and slash 3 they are back references the first and the third group they are being kept intact and slash 2 is just once, so just one occurrence we will like to keep here. So, this is how the pattern will use to match and the placement REPL will use to actually replace the matching occurrences.

(Refer Slide Time: 14:19)



So, now let us have a look at the code you know more detail, so if you come down to this part correct each tokens. So, here you can see we are using a list comprehension we are calling replace function for TN in tokens. So, for each token in the list of tokens that we might be passing will be calling this replace you know function here. So, let us have a look at replace now.

So, once we receive a token will check in the world net whether that is syntactically correct word or not, so it might be so. So, we would like to stop at that point of time, so if world net dot sin

sets and in the begin the balance is TK and token then will return token. If it is true then we will get an token that means we have arrived at a syntactically correct word if not then we will like to replace the pattern.

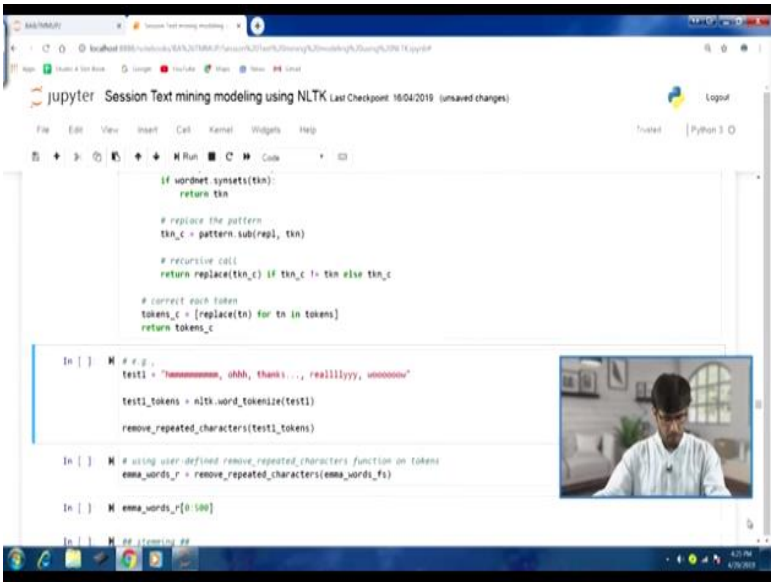
So, TK and underscore C and we are calling you know using this rejects object pattern dot sub and REPL will be use to replace. So, all the you know matching occurrences of the pattern they are going to be replaced with REPL in that particular token TK. Now then we are making a recursive call here, so you can see if token underscore C is not equal to token, in that case you know we are again calling replace TK and underscore C.

And if happens to be if it you know happens to be you know equal to then will written TK and underscore C itself. So, in this fashion recursively because 1 character might be repeated many times, so once we remove you know let us say it is repeating 4 times or we removed once. Then again we will you know we will check whether it is you know equal or not if it is you know not equal then again we will call replace.

In that fashion we will you know go on in this fashion till we you know arrive at a situation where in a recursive call the first if that we have Warner dot sin sets there, we are able to reach the syntactically correct word. So, somewhere this written will help us these decent syntactically correct word and that is the place where this if TK and underscore C not equal to token would actually become you know false and the else part would be written, so it will end this you know record C loop there.

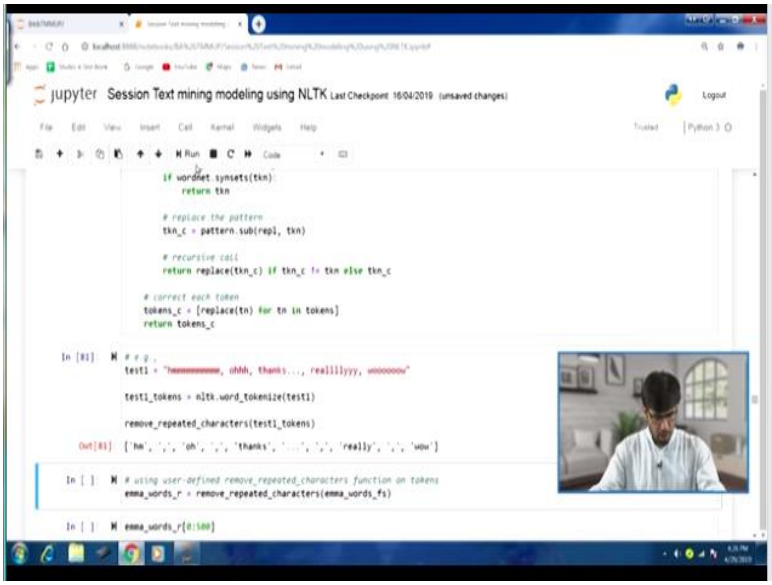
So, this is how this code will work to actually remove the repeated characters, so let me run this.

(Refer Slide Time: 17:01)



And let us have a let us take an example here, so here you can see test 1, this is the you know sentence this we are taking you can see all the words they are having repeated character. And then we are calling our you know we are tokenizing this you know sentence here, you are calling word underscore tokenize, then we are passing you know this token to our user our you know function that we have just defined remove underscore repeated characters.

(Refer Slide Time: 17:36)

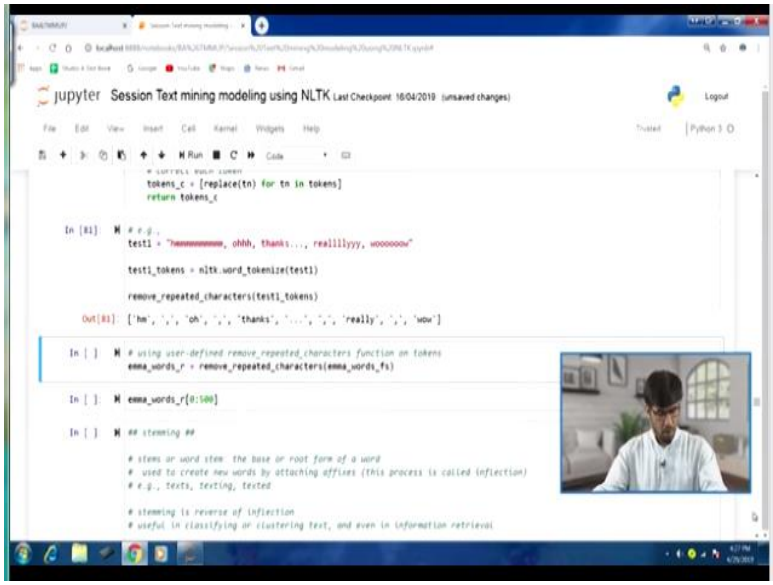


So, if I run this and you can have a look at the output 81 you can see you know HM there is because we would not reach a syntactically correct word there. So, only will this you know function will be called recursive call will keep on happening till we are left with just one M. In the second case O triple H will be left with OH because you know that is also you know that you

know again same example it might not be syntactically correct word.

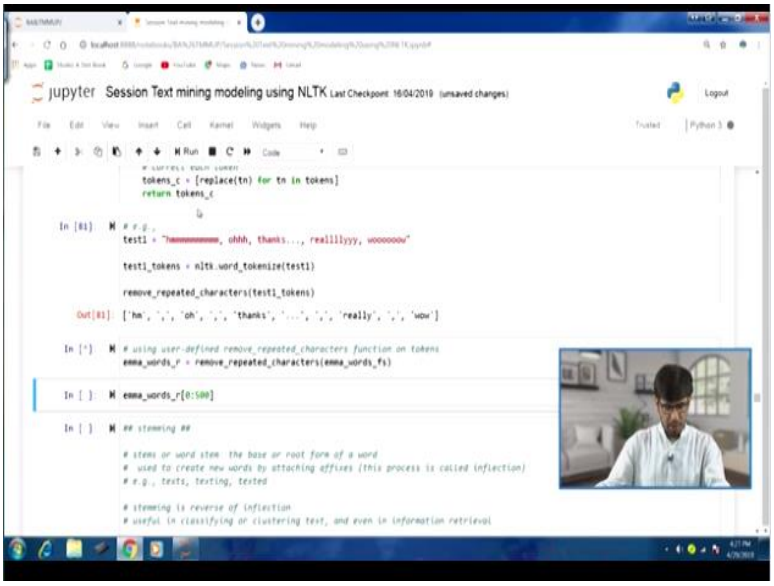
Then third one thanks, you can see there dot is repeating thrice, so that is gone and thanks is syntactically correct words, so it will stop there. Similarly really you can see you know 2 Ys and you know two L's are gone because only then will be able to reach the syntactically correct word similarly wall also, there also you know those in extra Os are gone. So, in this fashion our you know you can see that the function that we have justified is working quite well in terms of removing repeated characters.

(Refer Slide Time: 18:43)



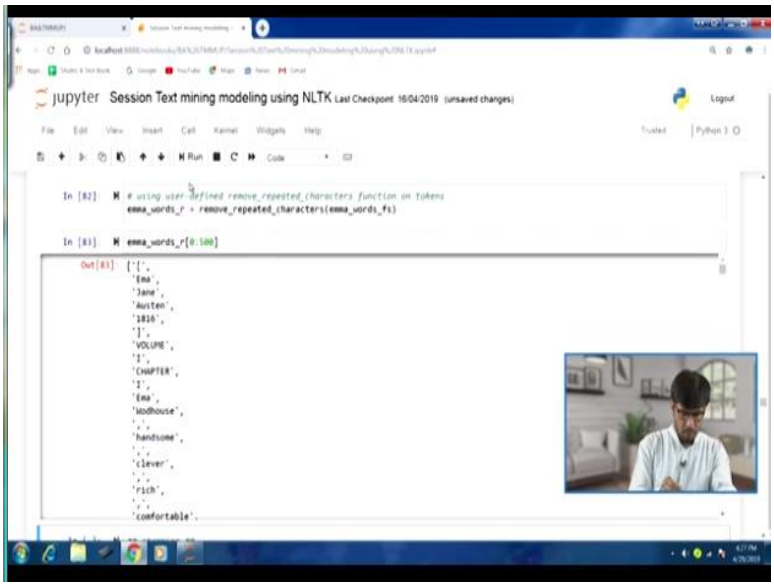
Now let us talk about let us apply this remove underscore repeated characters on Emma underscore words underscore FS as well.

(Refer Slide Time: 18:54)



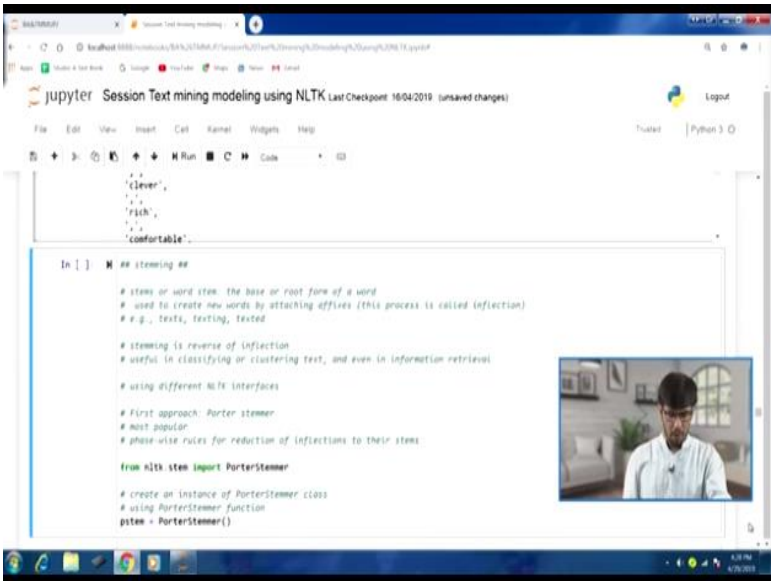
So, let me run this and if there are any such instances in this particular you know text corpus then those would also be corrected.

(Refer Slide Time: 19:07)



But because Emma was part of formal writing, so I do not expect this to happen in that case but anyway we can call this function.

(Refer Slide Time: 19:16)

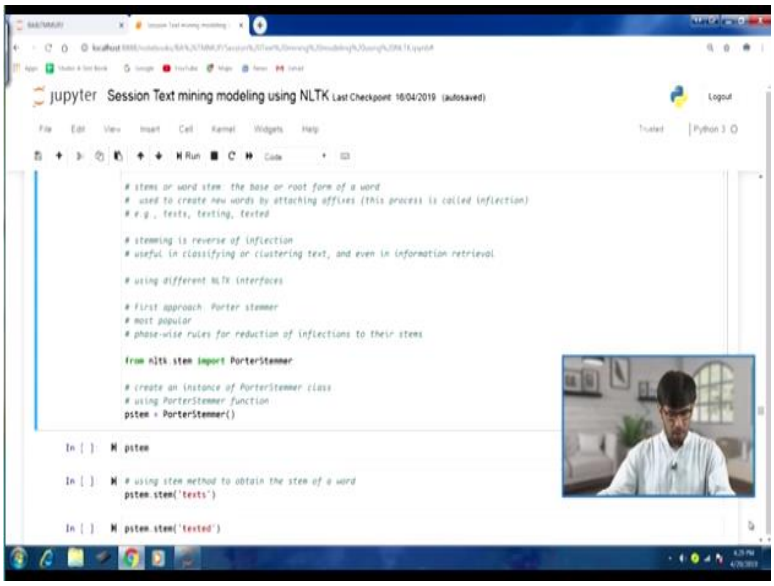


So, with this let us move onto the next aspect that is very important for text you know processing that is a stemming. So, what is stem or word you know or word stem, so essentially what we mean by this process systeming is we would like to reach the base or root form of a word. Because in you know there could be multiple variants of a root form of a word.

For example text we might in writing we might have texts, texting, texted, so we would like to just keep the root form of the word. So, essentially when we say stem or word stem we are essentially referring to that particular root form of the word. So, this might this kind of processing we are able to perform this kind of stemming in our text corpus, it will really help in our text analytics and text you know classification context, we will talk this in more detail.

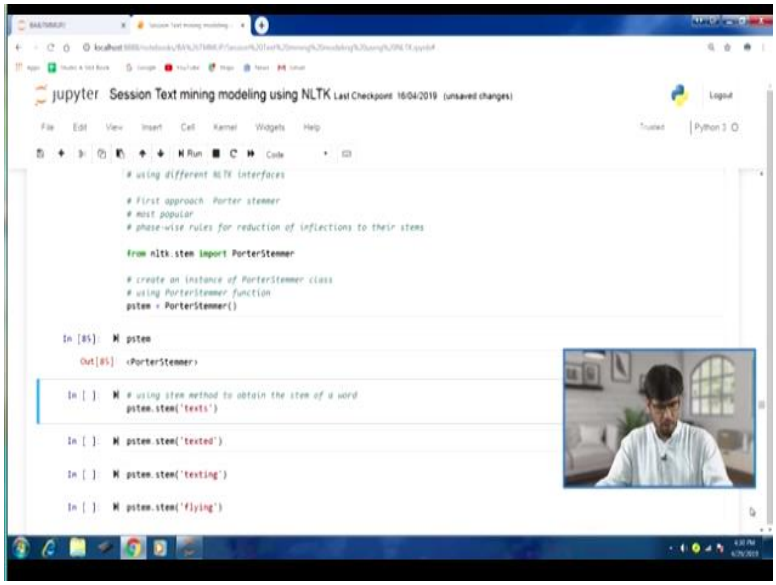
So, stemming is all stemming can also be considered as a reverse of inflection, so as we said useful in classifying or clustering text and even in sometimes in information retrieval. There are so many available you know an NLTK interfaces for this, so we will talk about the first approach which is porter stemmer, porter algorithm this is the most popular approach. So, in this one we perform phase wise you know rules, we use phase wise rules for reduction of inflection to their stems.

(Refer Slide Time: 20:53)



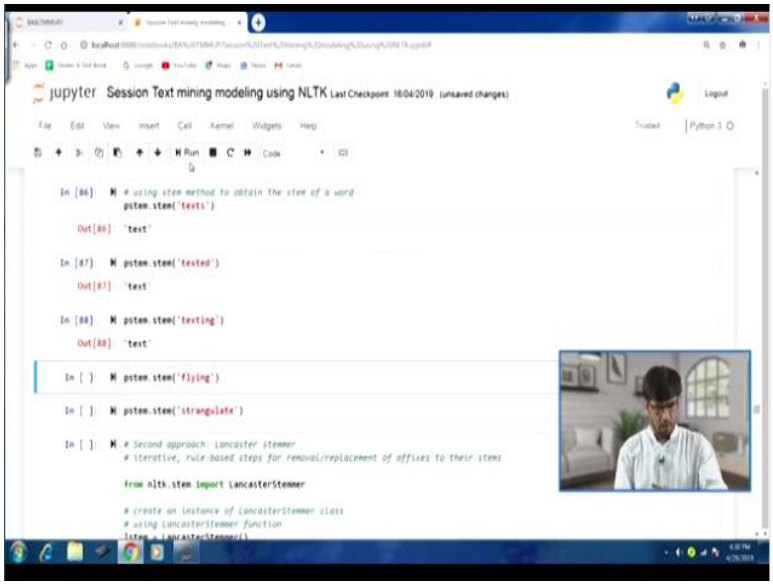
So, let us have a look at the code here, so for this we will have to import this from this and LD hit order NLTK dot stem module will have to import this class porter stemmer and then will define an instance of this porter stemmer class and let us run this.

(Refer Slide Time: 21:12)



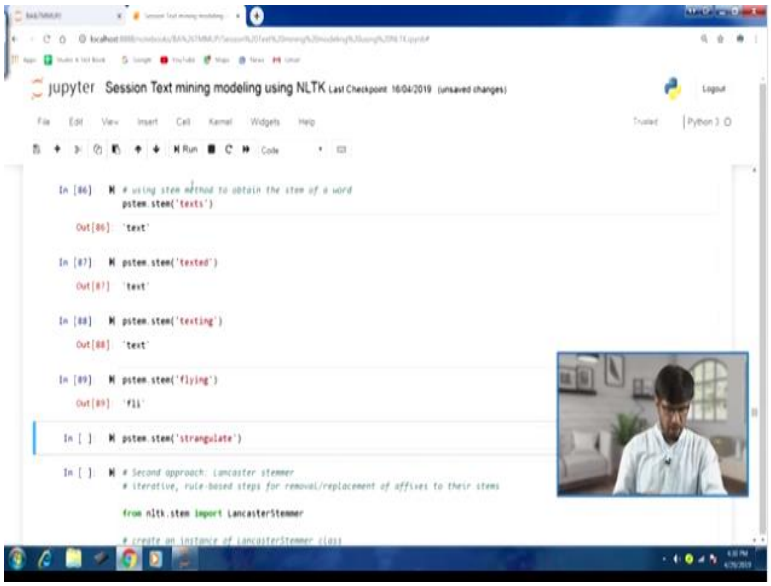
If you look at the PSTM you can see this is porter stemmer instance of porter stemmer class. Now we can call this you know stem method using this instance and pass on the you know pass on the examples here for example text or texted or texting and it will give us the you know route form of the word.

(Refer Slide Time: 21:36)



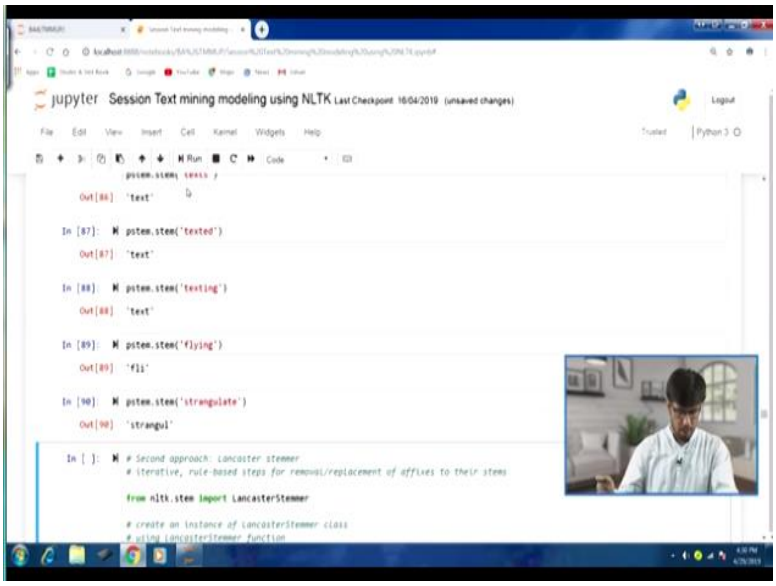
So, if I run this you can see texted to get text texting again you will get text, now if I take an example like flying.

(Refer Slide Time: 21:45)



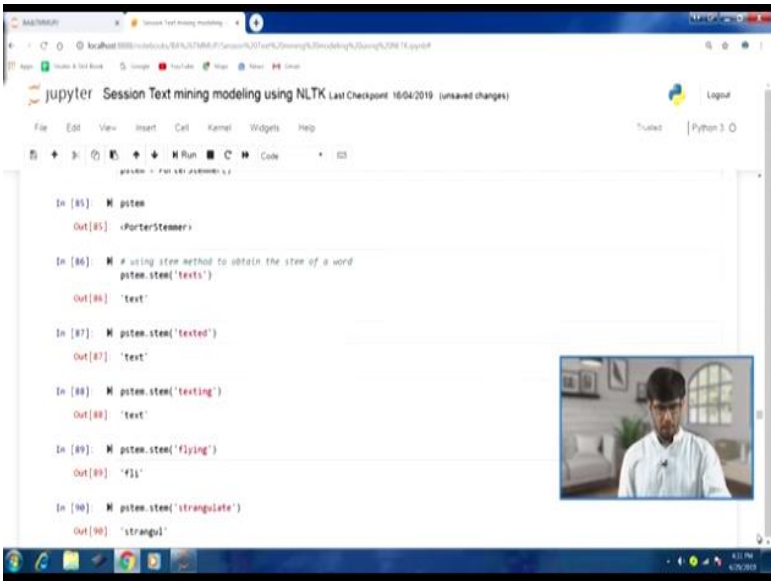
So, if I run this we get FLI and you know for strangulate.

(Refer Slide Time: 21:53)



If I run this you can see strangle, so in this fashion you can see you know different forms of a word would be there in writing in a pub would be there as part of a text. So, how we can use stemming algorithm suppose apple in this case we use Porter's method. So, how we can use these methods to actually reach arrive at a root form of the word and what it will how it will help us in text and analytics.

(Refer Slide Time: 22:20)

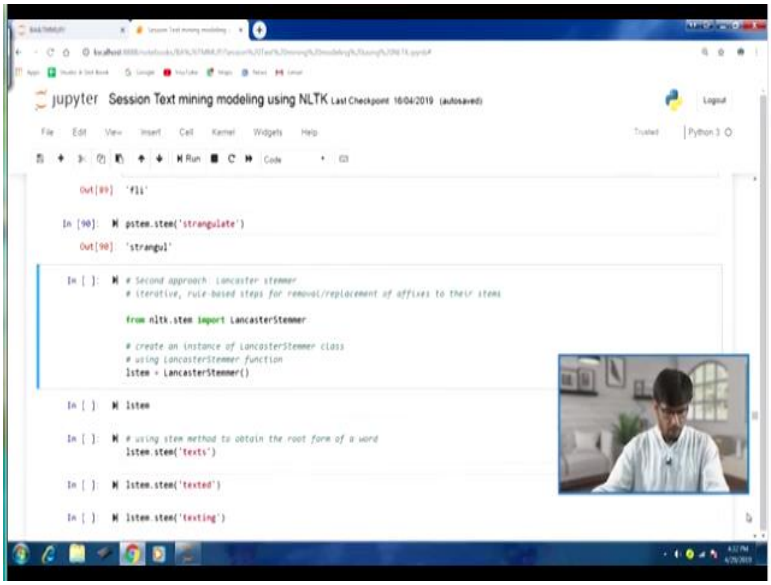


In that sense that if see if in your text if you have you know if you have words if you have variants like text, texture, texting and all of those instances would be converted into text. So, therefore frequency of text will increase and that will be really helpful in our you know factorization steps or later on you know when we apply a machine learning algorithms, so

because the frequency will increase.

So, and you would see that you know as we have talked about that in our table tabular layout you know will be counting you know for a particular term will be counting how many times it has appear in a particular document. So, that information is important now in this we perform stemming, so it will really help us in terms of having a you know fair distribution of you know root words or you know stems.

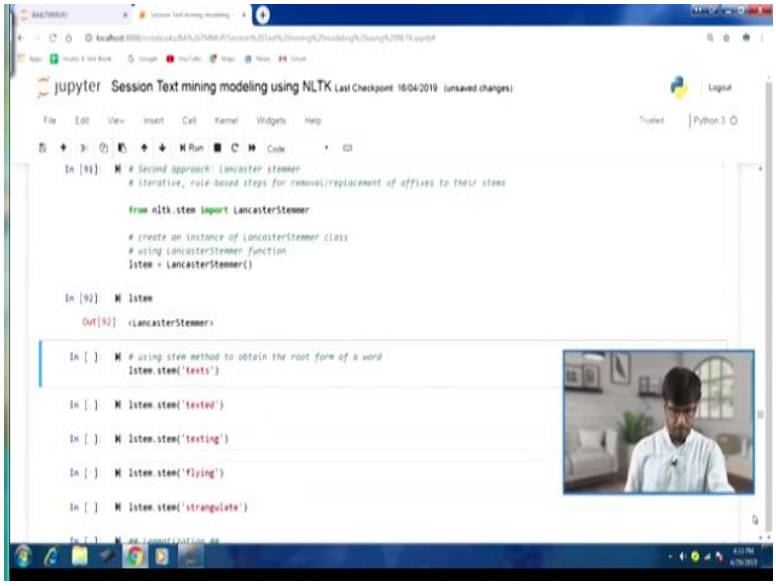
(Refer Slide Time: 23:18)



So, it will statistically it will help us in terms of distribution of words and therefore you know application of machine learning algorithms will also be you know useful. Now let us talk about the second approach this is the Lancaster's stemmer, so this is the iterative and follows rule-based steps for removal or replacement of affixes to their stems. So, you know when we talk about root form of root form of a word or stem.

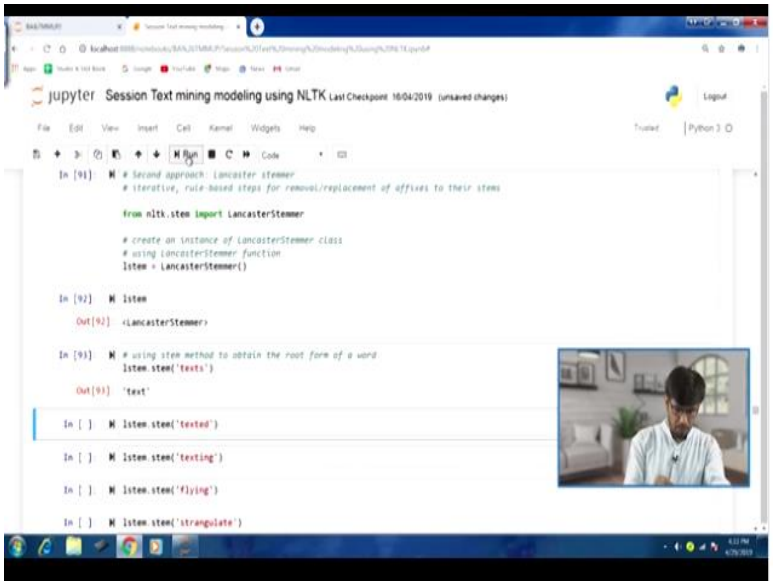
So, there are going to be affixes there prefixes or suffixes and different variants of that words would be there in the writing. So, this particular you know algorithm you know is based on removal or replacement of these affixes. So, let us import this Lancaster stemmer class and we will be using an instance of this class.

(Refer Slide Time: 24:17)



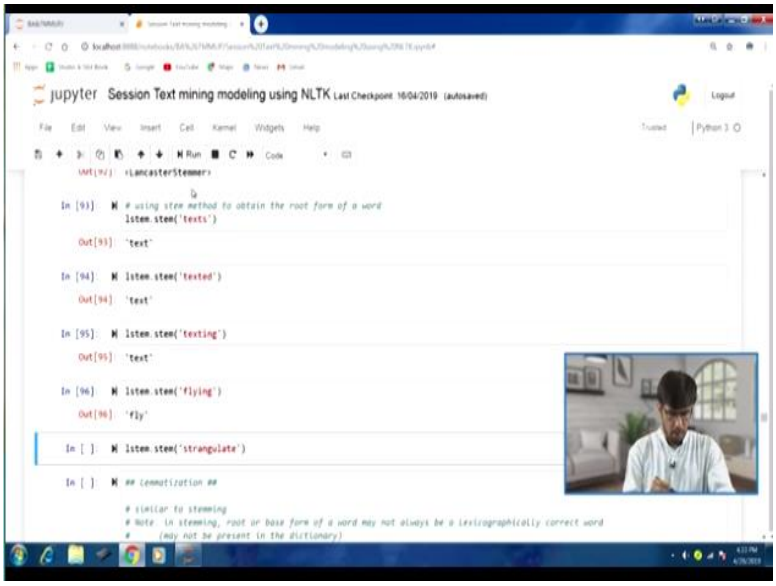
So, let me run this and you can have a look at the class Lancaster stemmer this is an instance of that. Now will be again calling stem method of this particular class and will be passing on the same example same words we will be using here again and we will try to find out the root here.

(Refer Slide Time: 24:32)



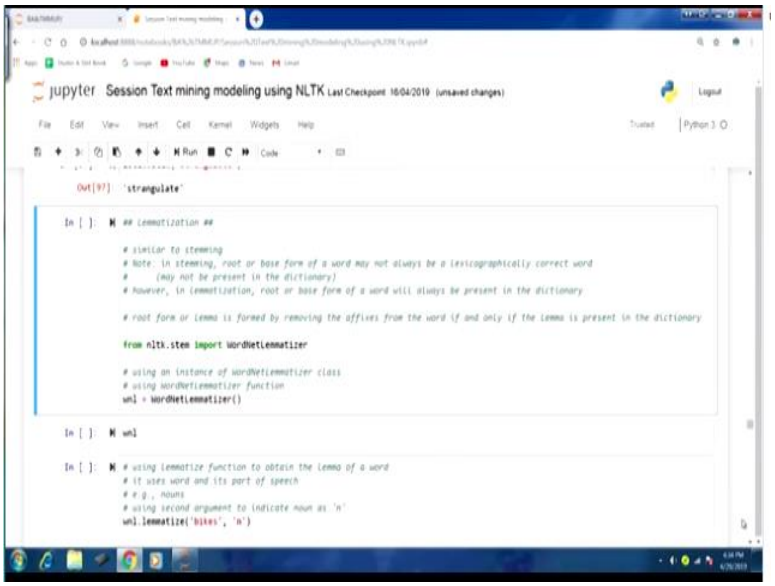
So, you can see same output but texted also same output for texting also same output like you know Porter algorithm, how we run it for flying you can see we got fly here in the porters we got FLI.

(Refer Slide Time: 24:40)



So, you can see different you know stemmer different stemming algorithm they might give you different output depending on what kind of you know steps they follow.

(Refer Slide Time: 25:00)



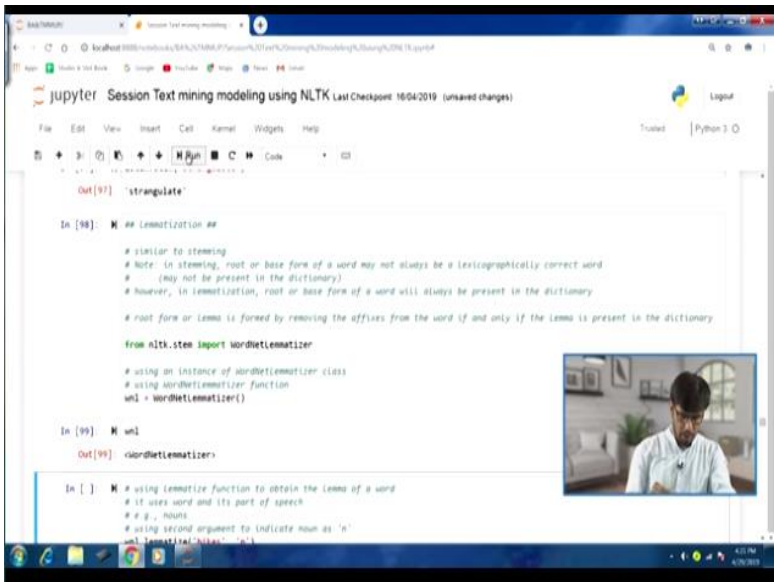
If I run this strangulate on this, so for this I think it is the again they there also difference in terms of Porter algorithm, we got a strangle, here we got a strangulated itself. So, you can see these 2 different algorithms they are working slightly differently when it comes to you know stemming. Now let us move to you know another aspect that is lambda lemmatization, so lambda lemmatization is quite similar to is to stemming.

And in stemming typically root or base form of a word you know that we typically obtain that

may not always be like lexicographically correct word. So, we are just trying to arrive at a stem or root form of a word which might not be present in the dictionary in another words. But when we perform lemmatization it is the root or base form of word you know that will also be present in dictionary that is taken, so slight difference is there.

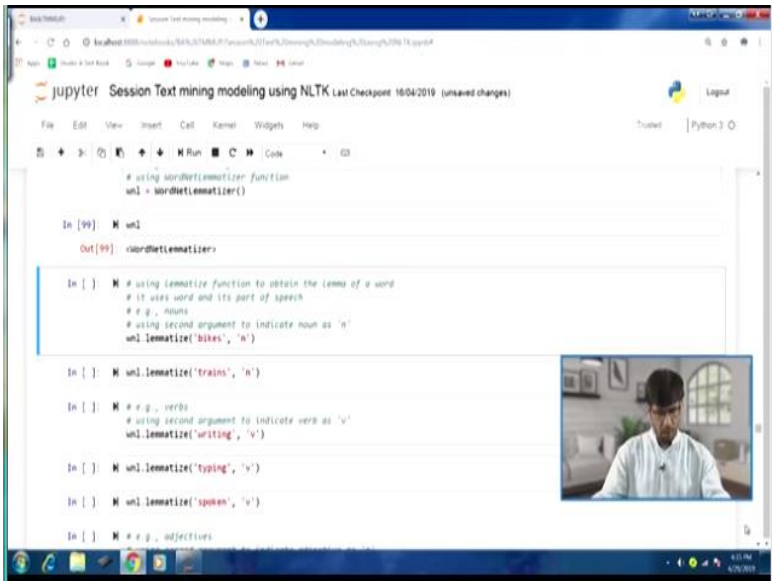
So, root form of or lemma is formed by removing the affixes from the word if and only if lemma is present in the dictionary. So, there in stemming the stem might not be present in the dictionary however in lemmatization the lemma that we obtain the root form of the word that we obtain it has to be present in the dictionary. So, for this will be using again and NLTKdot stem module and word net lemmatizer, this is the class that we will be using, so we will define an instance of this class.

(Refer Slide Time: 26:33)



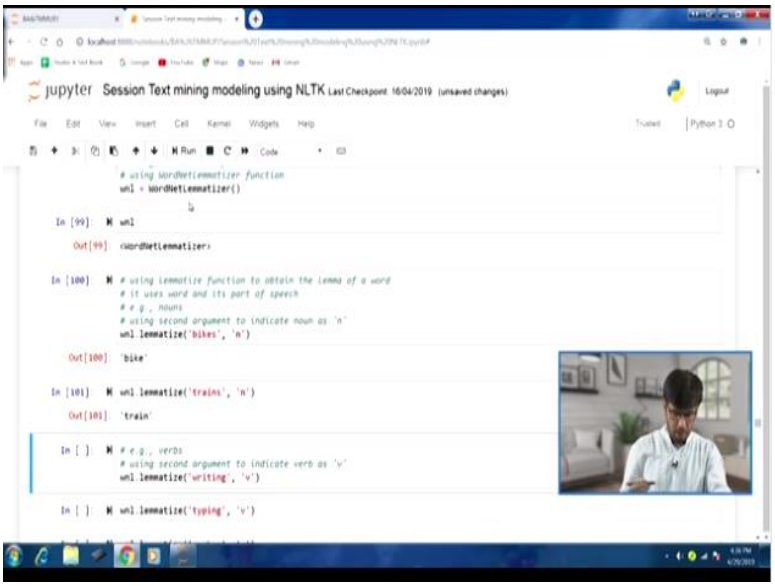
So, let me run this and will have WNL, this is an instance of word net lemmatizer class as you can see.

(Refer Slide Time: 26:40)



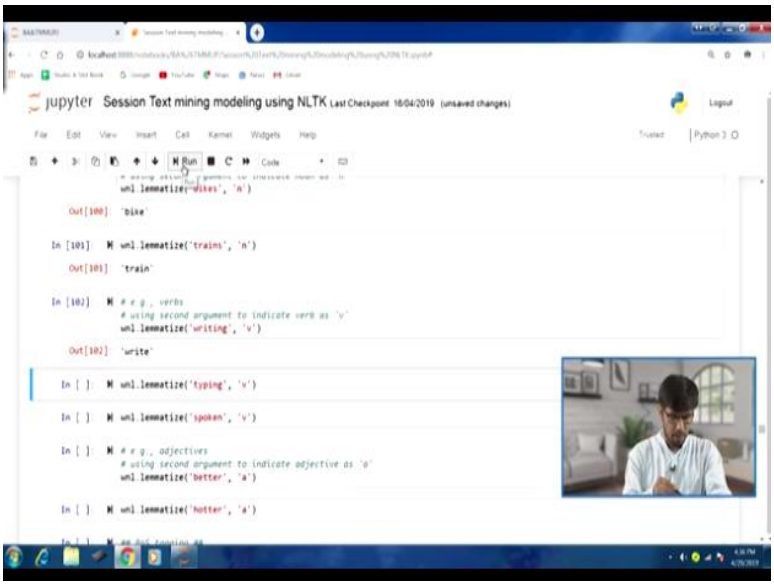
Now what we will do we will take few examples, so we will use the limit lemmatize function to obtain lemma of a word here and you know it uses word and it is part of his speech. So, for example if we you know let us take the example of you know some nouns. So, second argument is typically use to indicate noun, so that is as n in single codes you can see WNL dot lemmatize within parentheses first argument is the word that we would like you lemmatize here bikes and comma and the part of a speech it is a noun**.**

(Refer Slide Time: 27:18)



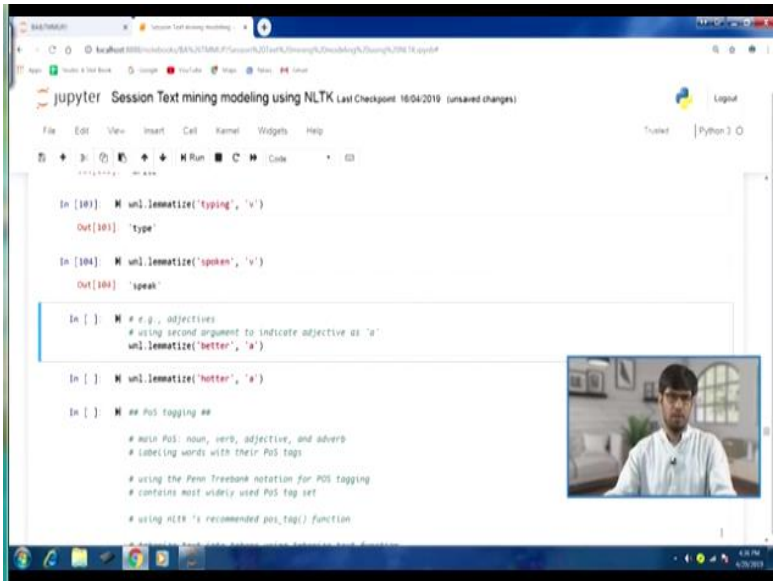
So, let me run this you can see bikes has become pike similarly trains it has become train.

(Refer Slide Time: 27:27)



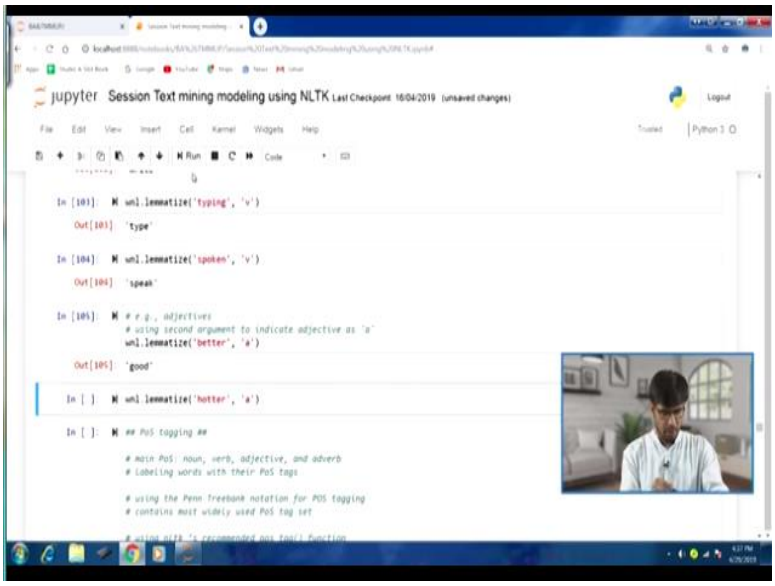
So, you can see part of a speech is also is required to be passed on here because that will help in terms of identifying the you know root form of the word in the dictionary. Now let us take example of a few word, so again in the second argument instead of you know now will indicate v-twin you know to denote word here. So, writing will become write, typing will become type and a spoken will become a speak as you can see here.

(Refer Slide Time: 27:56)



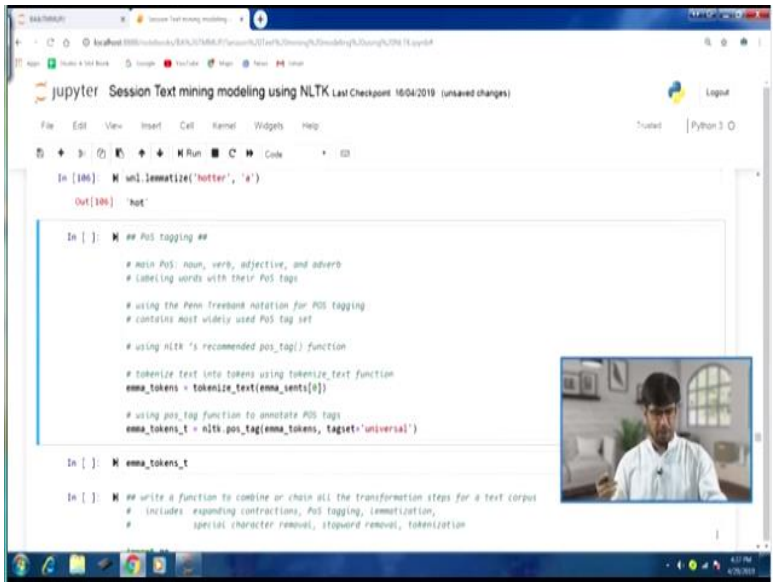
So, you can see how lemmatization is slightly different from you know stem, now let us take another example this term adjective so better and A to indicate that this is an adjective.

(Refer Slide Time: 28:13)



So, if I run this you can see good and hotter we got hot.

(Refer Slide Time: 28:16)



So, in this fashion you can see that how lemmatizer is working here, now let us move to the another aspect here that is you know POS tagging. So, we just learned about you know that in lambda lemmatization that whenever we are lemmatizing a word or text or a particular token, we also indicate part of speech. So, how do we tag you know all the tokens with their corresponding part of a speech so for that we have will talk about this POS tagging aspect here.

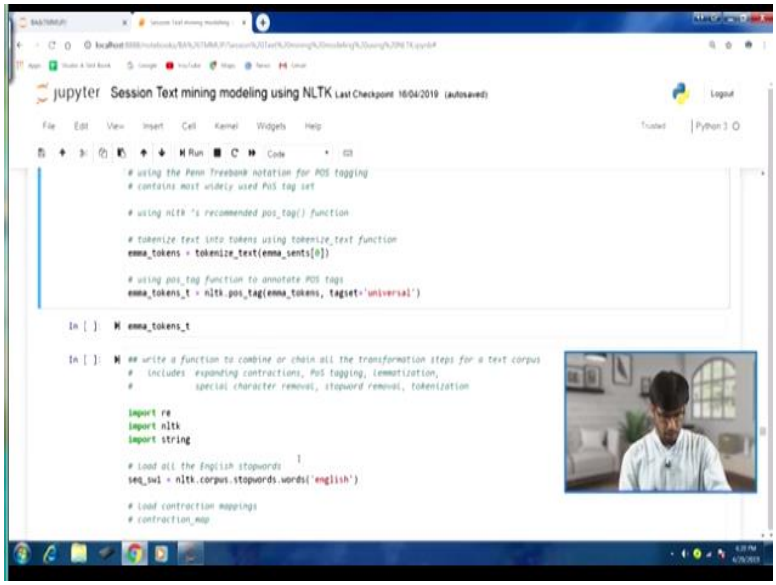
So, main part of a speech that are typically used in writing and here also noun, verb, adjective and adverb. So, it is important for us to label words with their POS tags and typically pantry

bank notation you know is used for you know he POS tagging and most it contains the most widely used pure stack set. So, will be using this analogy NLTKs POS underscore tag function to perform part of a speech tagging.

So, let us take an example, so we will use tokenize underscore text function here and we will pass on Emma underscore sense, it will you know perform tokenization here and then will call this NLTK dot. So, NLTK dot POS underscore tag, so first argument we need to pass on tokens there and tag set also something you know different tax sets are there or you can just not specify this, so universal is one example that can be used.

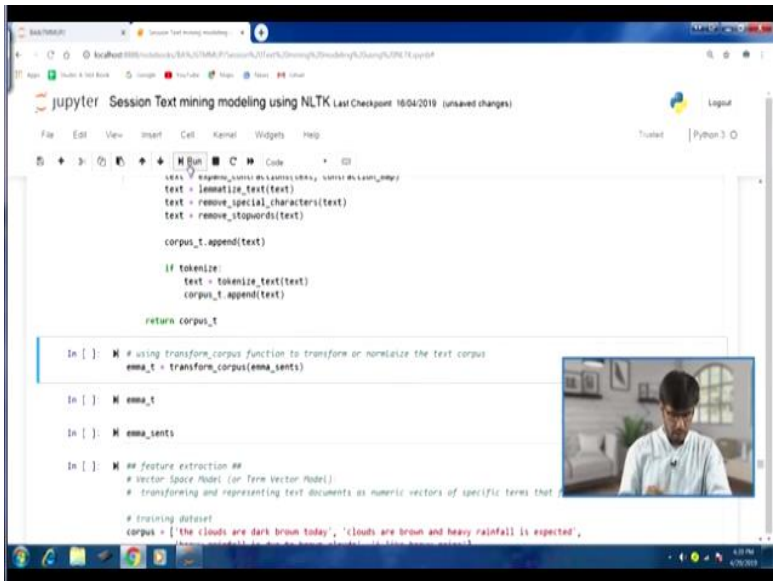
So, NLTK so in that code you can see that we first organize token, so we need to talk tokenize, for this I have written a function tokenize underscore text in the next section.

(Refer Slide Time: 30:00)



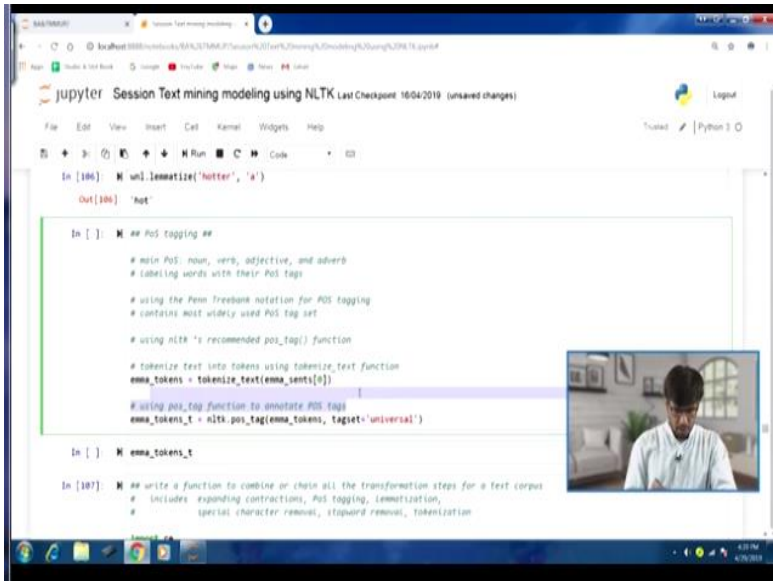
So, here I will just also run another part of the code right now, so that we are able to access that this particular function here.

(Refer Slide Time: 30:14)



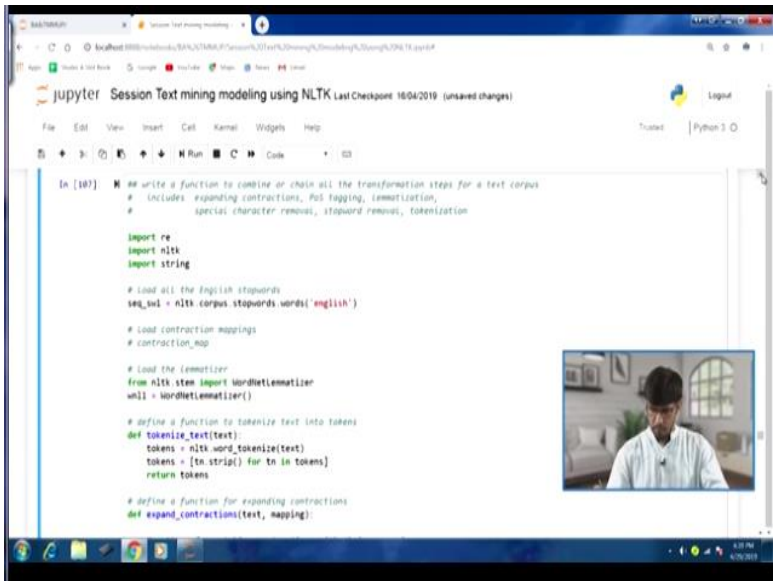
So, I will run this part as well and I will go back we will come back to this part and discuss in detail but because since I am using this function, so let me go back and use it here for POS tagging.

(Refer Slide Time: 30:28)

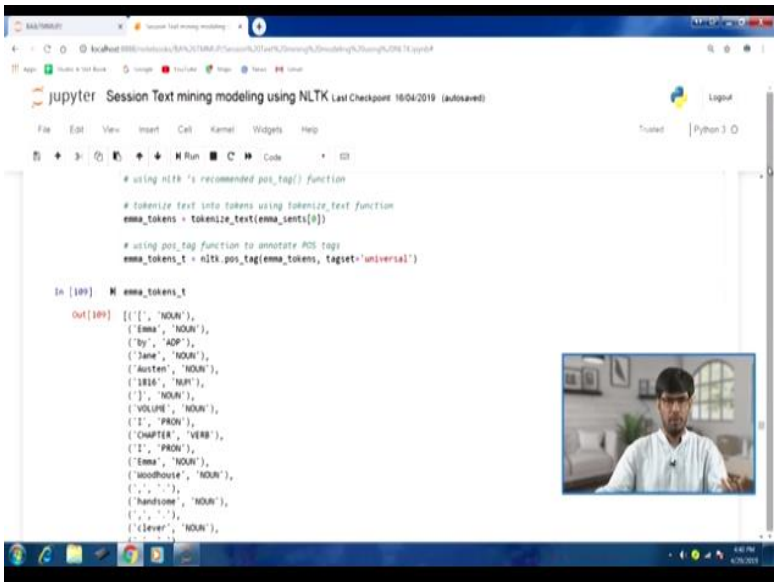


So, if I run this and let us have a look at the tokens here.

(Refer Slide Time: 30:34)



(Refer Slide Time: 30:39)



So, you can see in the output here in the output 109 you can see Emma noun by ADP you know all those tokens they have been tagged with their corresponding part of a speech. So, you can clearly see here, so you can see how first we tokenized the text and those tokens were passed in the POS underscore tag function to perform the POS tagging and now you can see the output. So, we would like to stop here and in the next lecture we will move on to the you know some of the next aspect of you know transforming unstructured text, thank you.

Keywords: contraction, structured and unstructured text, tokens, reduction, inflation etc.