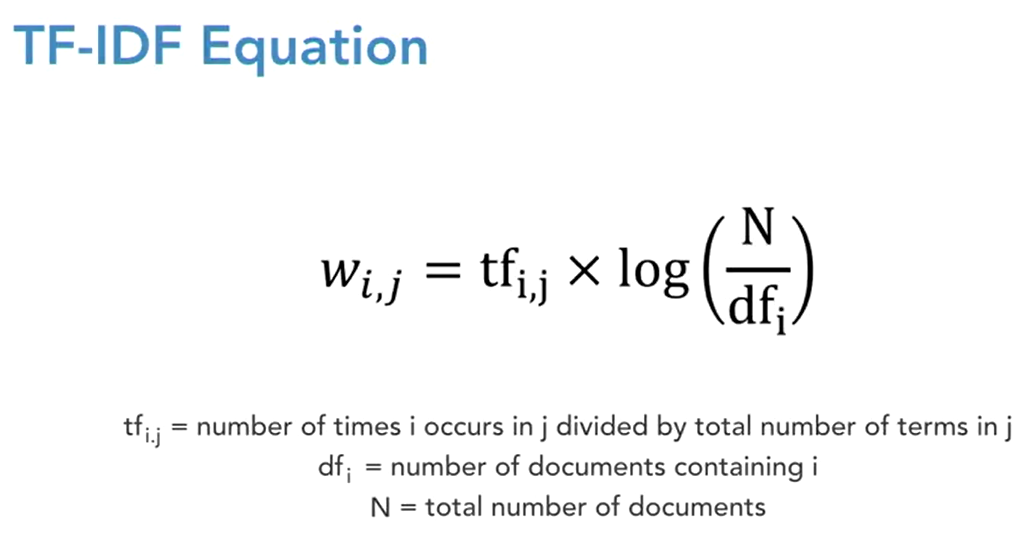
*Start a new Python project folder … do not reuse the previous workspace … As with any keyboard-driven console-like environment, developing muscle -memory for the common commands is also part of the learning curve.*

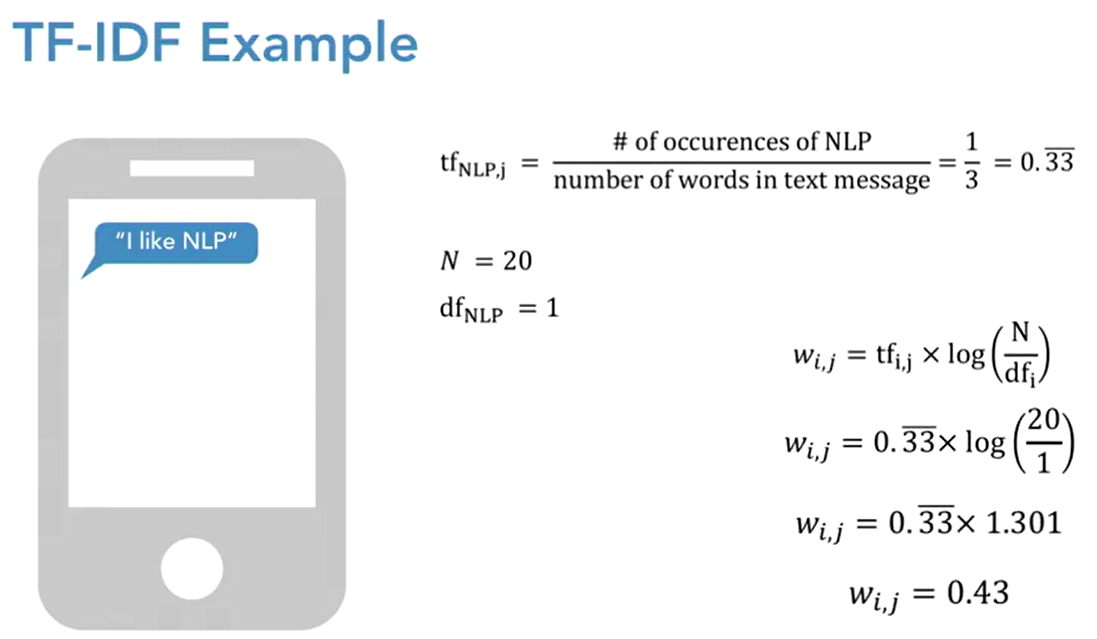
Term Frequency - Inverse Document Frequency, often referred to as TF-IDF. This is the 3rd method of vectorizing. These are the three most popular vectorizing methods.



TF-IDF creates a document term matrix, where there’s still one row per text message and the columns still represent single unique terms. But instead of the cells representing the count, the cells represent a weighting that’s meant to identify how important a word is to an individual text message. This formula lays out how this weighting is determined. It may look a little bit intimidating, but it’s actually quite simple. You start with this TF term, which is just the number of times that term I occurs in text message J, divided by the number of terms in text message J. It’s just the percent of terms in this given text message that are this specific word.

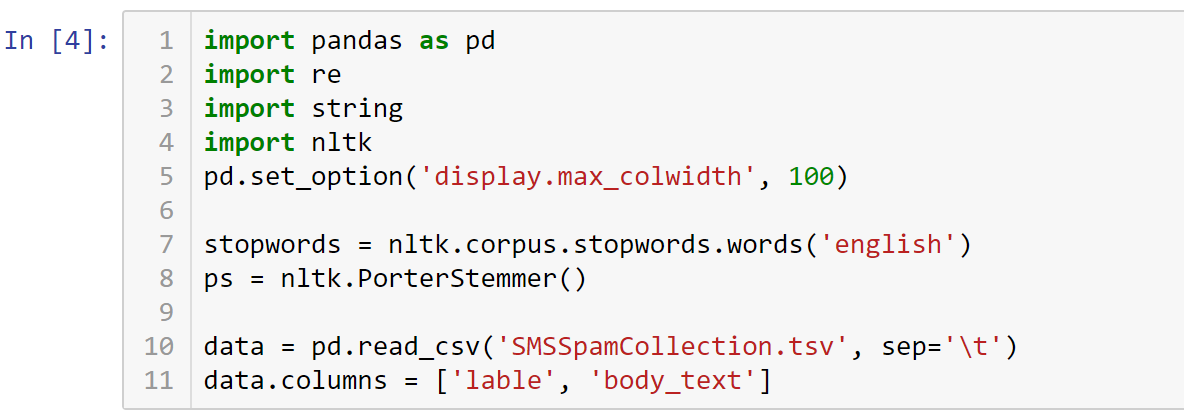
So for instance, if we use “I like NLP,” and the word we’re focused on is NLP, then this term would be 1 divided by 3, or 0.33. Then the second part of this equation measures how frequently this word occurs across all other text messages. It calculates the number of text messages in the data set divided by the number of text messages that this word appears in. That takes the log of all of that. Let’s just say that we have 20 text messages, so that’s going to represent N in this case, and only one of those contains NLP.

The second part of this equation would then be log of 20 divided by 1. As this fraction inside the log gets larger, the log of that fraction also gets larger. Now let’s say that you have 40 text messages instead of 20, but NLP still only occurs in one of them, so the denominator here will still only be 1. Now this fraction is 40 over 1. The term NLP is less frequent, and this term collectively is going to be larger.

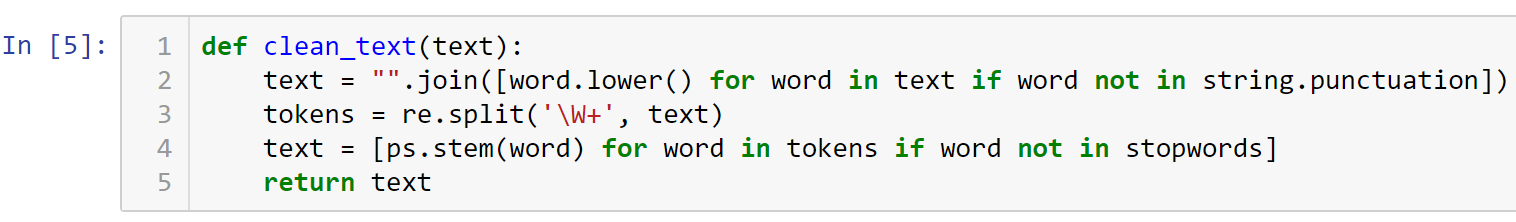


Basically, all this says is that the rarer the word is, the higher that this value’s going to be. If a word occurs very frequently within a particular text message, so that’s TF, but very infrequently elsewhere, that’s going to be the second term. Then a very large number will be assigned, and it’ll be assumed to be very important to differentiating that text message from others. In summary, this method helps you pull out important but seldom-used words.

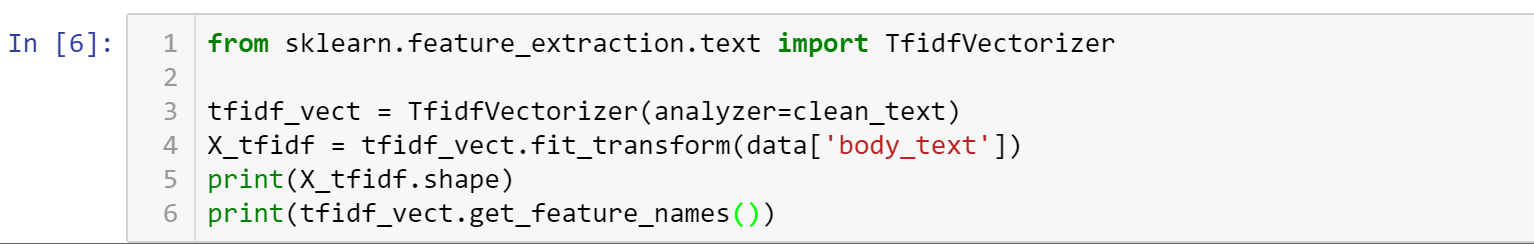
Enough on definitions . . . let’s jump into the code.

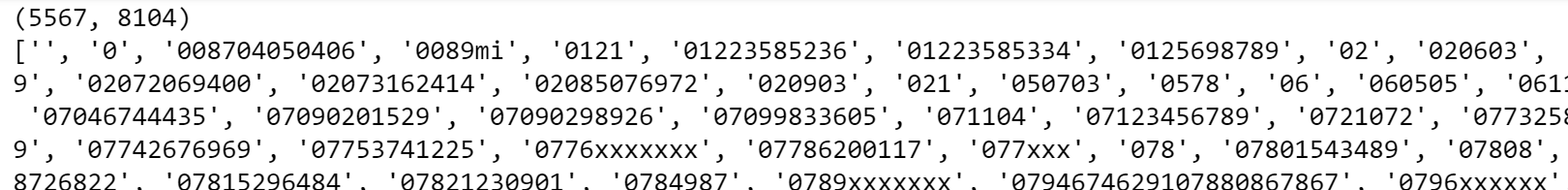


Create function to remove punctuations, tokenize, remove stopwords, and stem.

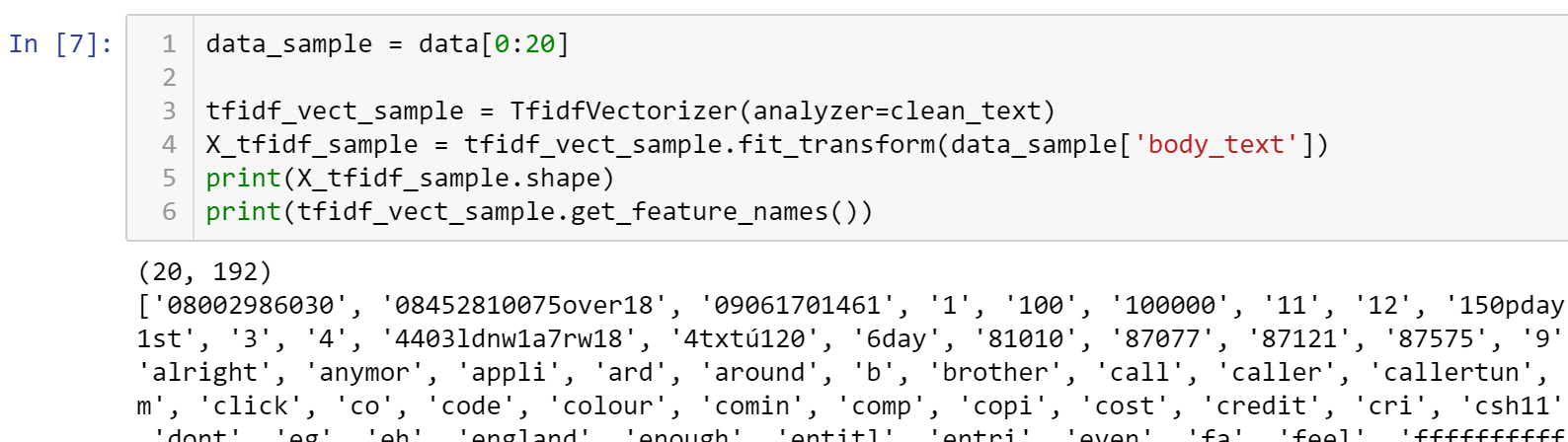


Now apply TfidfVectorizer

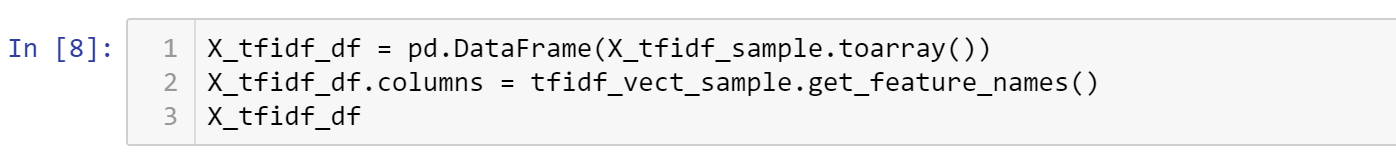


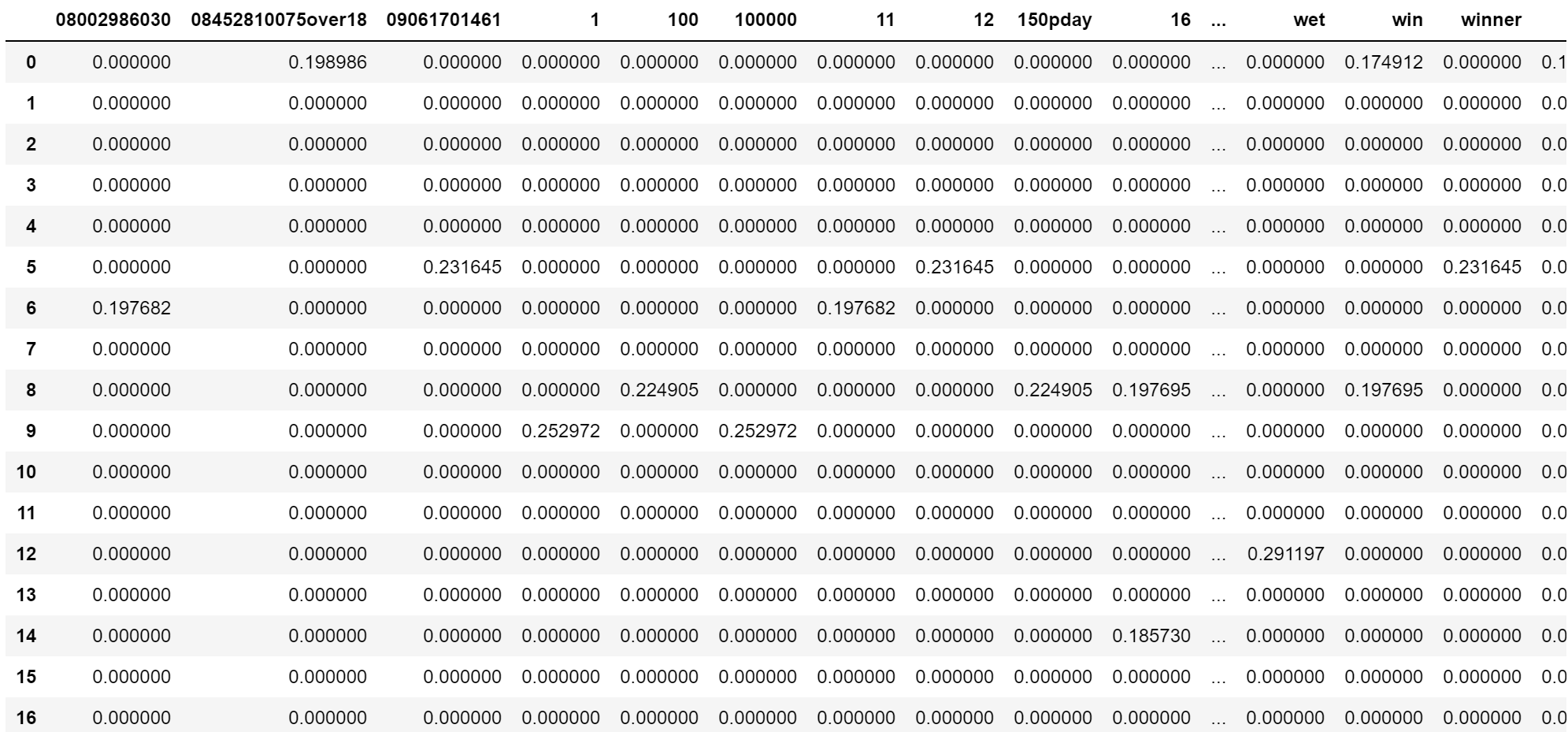


The shape is similar to the other vectorizer with 5,567 rows and 8,104 columns. So let’s create a smaller sample out of this data in order to make meaning of this model.



Creating a DataFrame to see the output since the raw output is a sparse matrix. A sparse matrix is a matrix which most entries are 0. In the interest of efficient storage, a sparse matrix will be stored y only storing the locations of the non-zero elements.





Note that the values of the cells are decimals instead of integers. They are the percentages. For example 0.231645 (or 23%) is likely more important than 0.197682. What that means is, either text “12” occurs more frequently in the 5th text message than “11” does in the 6th text message, or it means “12” occurs less frequently across all the other text messages than “11” does across all the other text messages.

In summary, we created this false choice here, indicating that there are three different ways to vectorize. These are all very closely related, though, and some can actually be used together. TF-IDF is basically a count vectorizer that includes some consideration for the length of the document, and also how common the word is across other text messages. And then n-grams is just used within either of these two methods to look for groups of adjacent words instead of just looking for single terms. They are all just slight modifications of each other, and typically you’ll test different vectorization methods depending on your problem, and then you let the results determine which one you use.

* All submissions should be separate from other exercises and quests. Please do not lump all your answers into one document and re-using that same workspace to gain multiple points. Thanks.
* Place your name at the bottom of your code, download your Python program in html format, and submit your work in Canvas.