In the previous video, we preprocessed our data, and we're now ready to extract the word frequencies to be used in our prediction problem.

The tm package provides a function called DocumentTermMatrix that generates a matrix where the rows correspond to documents, in our case tweets, and the columns correspond to words in those tweets.

The values in the matrix are the number of times that word appears in each argument.

Let's go ahead and generate this matrix and call it "frequencies." So we'll use the DocumentTermMatrix function calls on our corpus that we created in the previous video.

Let's take a look at our matrix by typing frequencies.

We can see that there are 3,289 terms or words in our matrix and 1,181 documents or tweets after preprocessing.

Let's see what this matrix looks like using the inspect function.

So type inspect(frequencies[1000:1005, 505:515]).

In this range we see that the word "cheer" appears in the tweet 1005, but "cheap" doesn't appear in any of these tweets.

This data is what we call sparse.

This means that there are many zeros in our matrix.

We can look at what the most popular terms are, or words, with the function findFreqTerms.

We want to call this on our matrix frequencies, and then we want to give an argument lowFreq, which is equal to the minimum number of times a term must appear to be displayed.

Let's type 20.

We see here 56 different words.

So out of the 3,289 words in our matrix, only 56 words appear at least 20 times in our tweets.

This means that we probably have a lot of terms that will be pretty useless for our prediction model.

The number of terms is an issue for two main reasons.

One is computational.

More terms means more independent variables, which usually means it takes longer to build our models.

The other is in building models, as we mentioned before, the ratio of independent variables to observations will affect how good the model will generalize.

So let's remove some terms that don't appear very often.

We'll call the output sparse, and we'll use the removeSparseTerms(frequencies, 0.98).

The sparsity threshold works as follows.

If we say 0.98, this means to only keep terms that appear in 2% or more of the tweets.

If we say 0.99, that means to only keep terms that appear in 1% or more of the tweets.

If we say 0.995, that means to only keep terms that appear in 0.5% or more of the tweets, about six or more tweets.

We'll go ahead and use this sparsity threshold.

If you type sparse, you can see that there's only 309 terms in our sparse matrix.

This is only about 9% of the previous count of 3,289.

Now let's convert the sparse matrix into a data frame that we'll be able to use for our predictive models.

We'll call it tweetsSparse and use the as.data.frame function called on the as.matrix function called on our matrixsparse.

This convert sparse to a data frame called tweetsSparse.

Since R struggles with variable names that start with a number, and we probably have some words here that start with a number, let's run the make names function to make sure all of our words are appropriate variable names.

To do this type COLnames and then in parentheses the name of our data frame, tweetsSparse equals make.names, and then in parentheses again colnames(tweetsSparse = make.names(colnames(tweetsSparse)).

This will just convert our variable names to make sure they're all appropriate names before we build our predictive models.

You should do this each time you've built a data frame using text analytics.

Now let's add our dependent variable to this data set.

We'll call it tweetsSparse$Negative = tweets$Negative.

Lastly, let's split our data into a training set and a testing set, putting 70% of the data in the training set.

First we'll have to load the library catools so that we can use the sample split function.

Then let's set the seed to 123 and create our split using sample.split where a dependent variable is tweetsSparse$Negative.

And then our split ratio will be 0.7.

We'll put 70% of the data in the training set.

Then let's just use subset to create a treating set called trainSparse, which will take a subset of the whole data set tweetsSparse, but always take the observations for which split==TRUE.

And we'll create our test set, testSparse, again using subset to take the observations of tweetsSparse, but this time for which split==FALSE.

Our data is now ready, and we can build our predictive model.

In the next video, we'll use CART and logistic regression to predict negative sentiment.