Pre-processing the data can be difficult, but, luckily, R's packages provide easy-to-use functions for the most common tasks.

In this video, we'll load and process our data in R. In your R console, let's load the data set tweets.csv with the read.csv function.

But since we're working with text data here, we need one extra argument, which is stringsAsFactors=FALSE.

So we'll call our data set tweets.

And we'll use the read.csv function to read in the data file tweets.csv, but then we'll add the extra argument stringsAsFactors=FALSE.

You'll always need to add this extra argument when working on a text analytics problem so that the text is read in properly.

Now let's take a look at the structure of our data with the str function.

We can see that we have 1,181 observations of two variables, the text of the tweet, called Tweet, and the average sentiment score, called Avg for average.

The tweet texts are real tweets that we found on the internet directed to Apple with a few cleaned up words.

We're more interested in being able to detect the tweets with clear negative sentiment, so let's define a new variable in our data set tweets called Negative.

And we'll set this equal to as.factor(tweets$Avg = -1).

This will set tweets$Negative equal to true if the average sentiment score is less than or equal to negative 1 and will set tweets$Negative equal to false if the average sentiment score is greater than negative 1.

Let's look at a table of this new variable, Negative.

We can see that 182 of the 1,181 tweets, or about 15%, are negative.

Now to pre-process our text data so that we can use the bag of words approach, we'll be using the tm text mining package.

We'll need to install and load two packages to do this.

First, let's install the package tm, and go ahead and select a CRAN mirror near you.

As soon as that package is done installing and you're back at the blinking cursor, go ahead and load that package with the library command.

Then we also need to install the package snowballC.

This package helps us use the tm package.

And go ahead and load the snowball package as well.

One of the concepts introduced by the tm package is that of a corpus.

A corpus is a collection of documents.

We'll need to convert our tweets to a corpus for pre-processing.

tm can create a corpus in many different ways, but we'll create it from the tweet column of our data frame using two functions, corpus and vector source.

We'll call our corpus "corpus" and then use the corpus and the vector source functions called on our tweets variable of our tweets data set.

So that's tweets$Tweet.

We can check that this has worked by typing corpus and seeing that our corpus has 1,181 text documents.

And we can check that the documents match our tweets by using double brackets.

So type corpus[[1]].

This shows us the first tweet in our corpus.

Now we're ready to start pre-processing our data.

Pre-processing is easy in tm.

Each operation, like stemming or removing stop words, can be done with one line in R, where we use the tm_map function.

Let's try it out by changing all of the text in our tweets to lowercase.

To do that, we'll replace our corpus with the output of the tm_map function, where the first argument is the name

of our corpus and the second argument is what we want to do.

In this case, tolower.

tolower is a standard function in R, and this is like when we pass mean to the tapply function.

We're passing the tm_map function a function to use on our corpus.

Let's see what that did by looking at our first tweet again.

Go ahead and hit the up arrow twice to get back to corpuscorpus{[1] and now we can see that all of our letters are lowercase.

Now let's remove all punctuation.

This is done in a very similar way, except this time we give the argument removePunctuation instead of tolower.

Hit the up arrow twice, and in the tm_map function, delete tolower, and type removePunctuation.

Let's see what this did to our first tweet again.

Now the comma after "say", the exclamation point after "received", and the @ symbols before "Apple" are all gone.

Now we want to remove the stop words in our tweets.

tm provides a list of stop words for the English language.

We can check it out by typing stopwords("english") [1:10].

We see that these are words like "I," "me," "my," "myself," et cetera.

Removing words can be done with the removeWords argument to the tm_map function, but we need one extra argument this time-- what the stop words are that we want to remove.

We'll remove all of these English stop words, but we'll also remove the word "apple" since all of these tweets have the word "apple" and it probably won't be very useful in our prediction problem.

So go ahead and hit the up arrow to get back to the tm_map function, delete removePunctuation and, instead, type removeWords.

Then we need to add one extra argument, c("apple").

This is us removing the word "apple." And then stopwords("english").

So this will remove the word "apple" and all of the English stop words.

Let's take a look at our first tweet again to see what happened.

Now we can see that we have significantly fewer words, only the words that are not stop words.

Lastly, we want to stem our document with the stem document argument.

Go ahead and scroll back up to the removePunctuation, delete removePunctuation, and type stemDocument.

If you hit Enter and then look at the first tweet again, we can see that this took off the ending of "customer," "service," "received," and "appstore." In the next video, we'll investigate our corpus and prepare it for our prediction problem.