In this project, the problem is to process a large set of documents (50) to understand how text analytics works. And we need to install the package of NLP and tm.

install.packages("NLP")

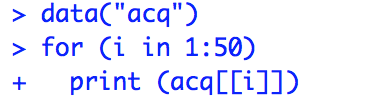
install.packages("tm")

library(NLP)#Required package for package tm

library(tm)#Package for function acq

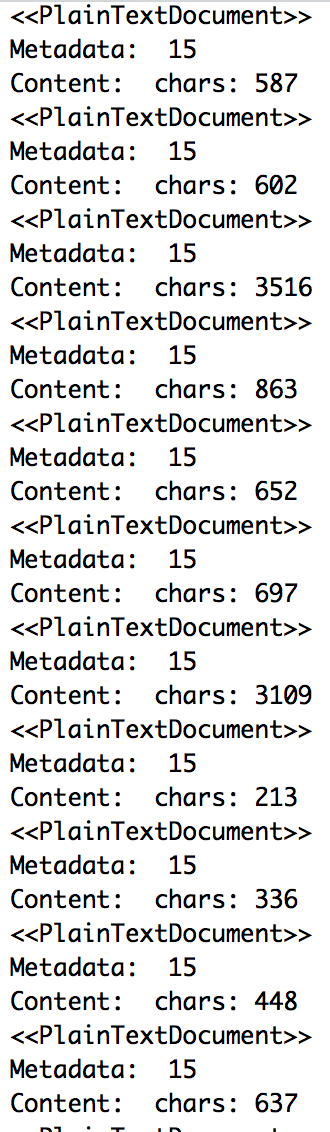
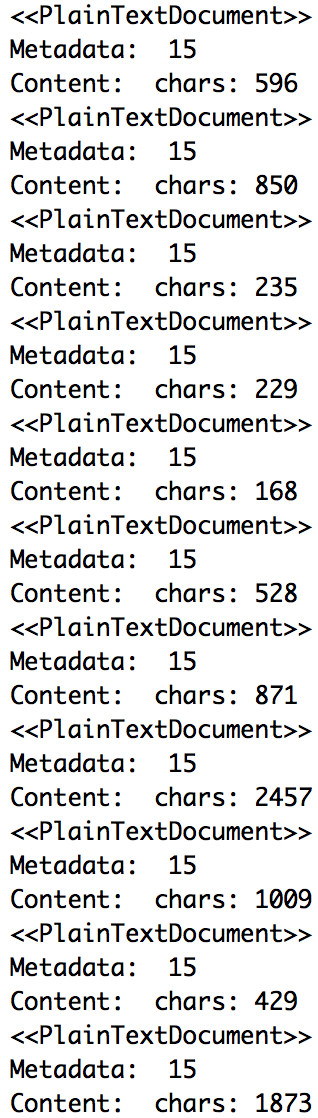
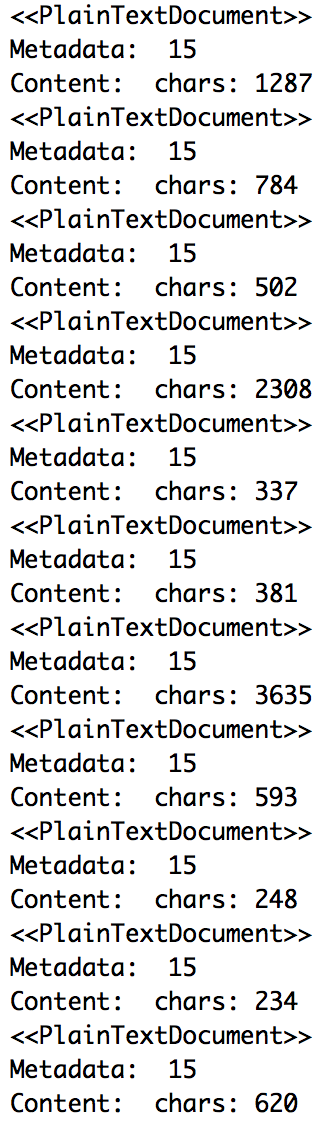
1. **Try the functions in lecture 7**

Load the data from acq first



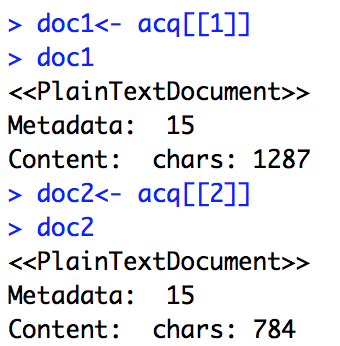
* 1. **inspect()**

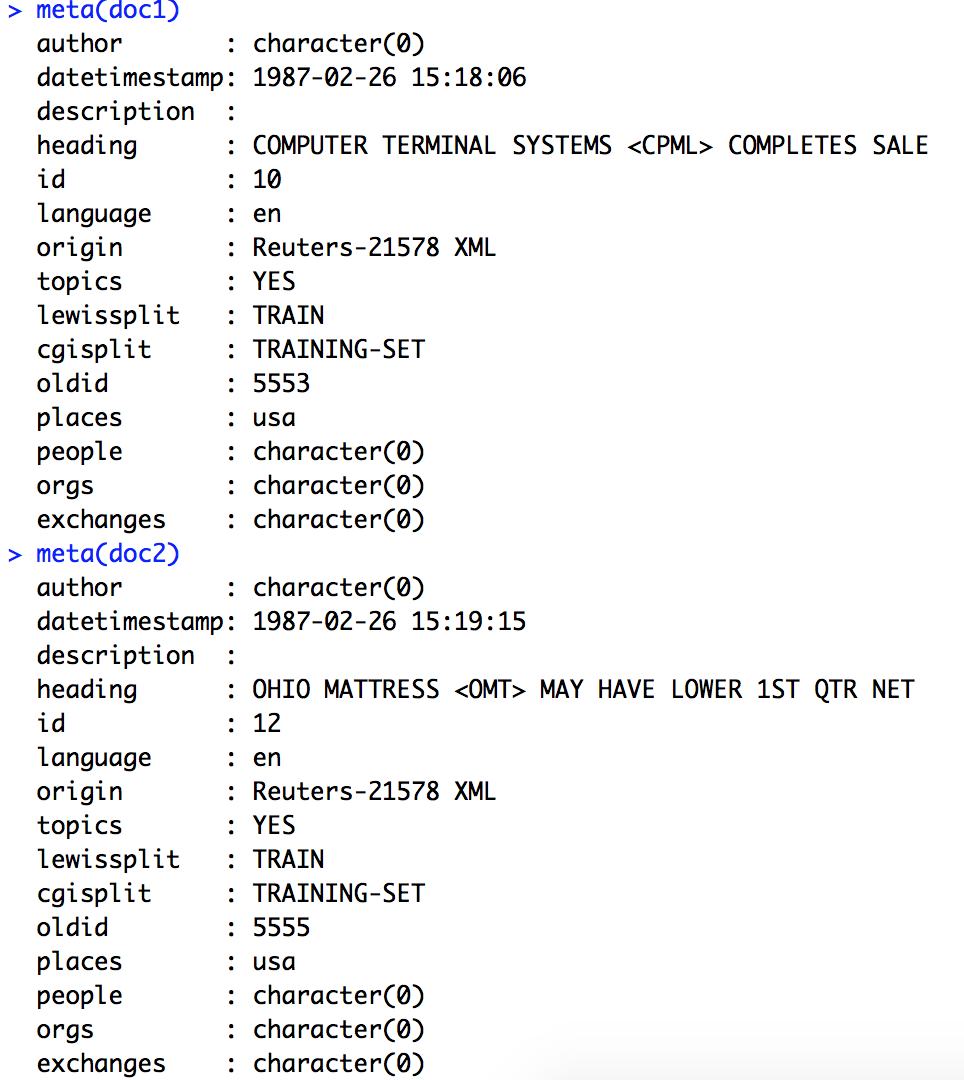
We used insepct() function to display detailed information on a corpus or a term-document matrix. By using inspect (SATdtm[a:b, c:d]), we are able to inspect part or the whole matrix. For example, such calling inspects the first 10 terms (by the order of ASCII) and their appearance in the 50 documents.



* 1. **Extract the single document to see the detail information.**

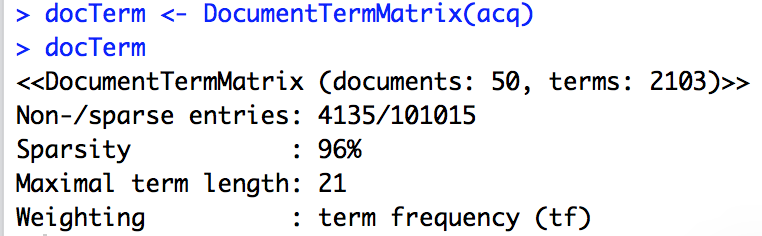
Use meta() function to return a named or empty list of tag/value pairs if no tag is given (default), or the value for the given tag.



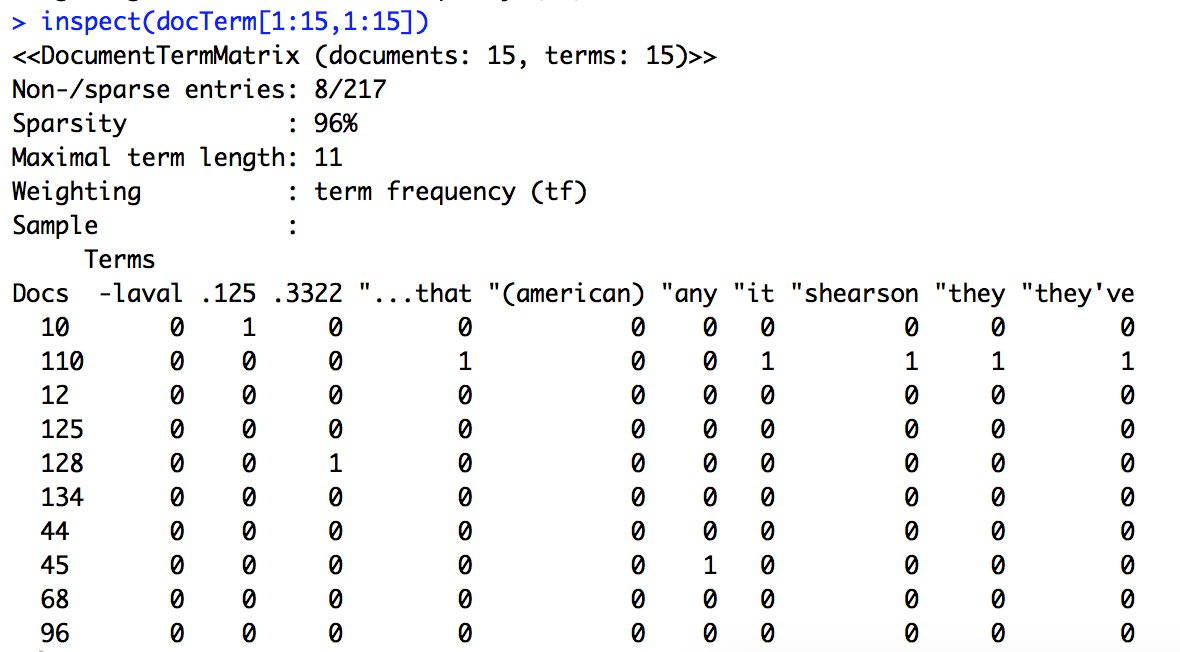


**1.3 DocumentTermMatrix**

By using this function, we are able to have a brief impression of the 50 documents: there are 2013 terms in these documents, making up a 50×2013 term matrix. There are 4135 non-zero elements and 101015 zero in this matrix, making the sparsity of the matrix to be 96% (101015/ (4135+101015)). The longest term is 21 letters in length.

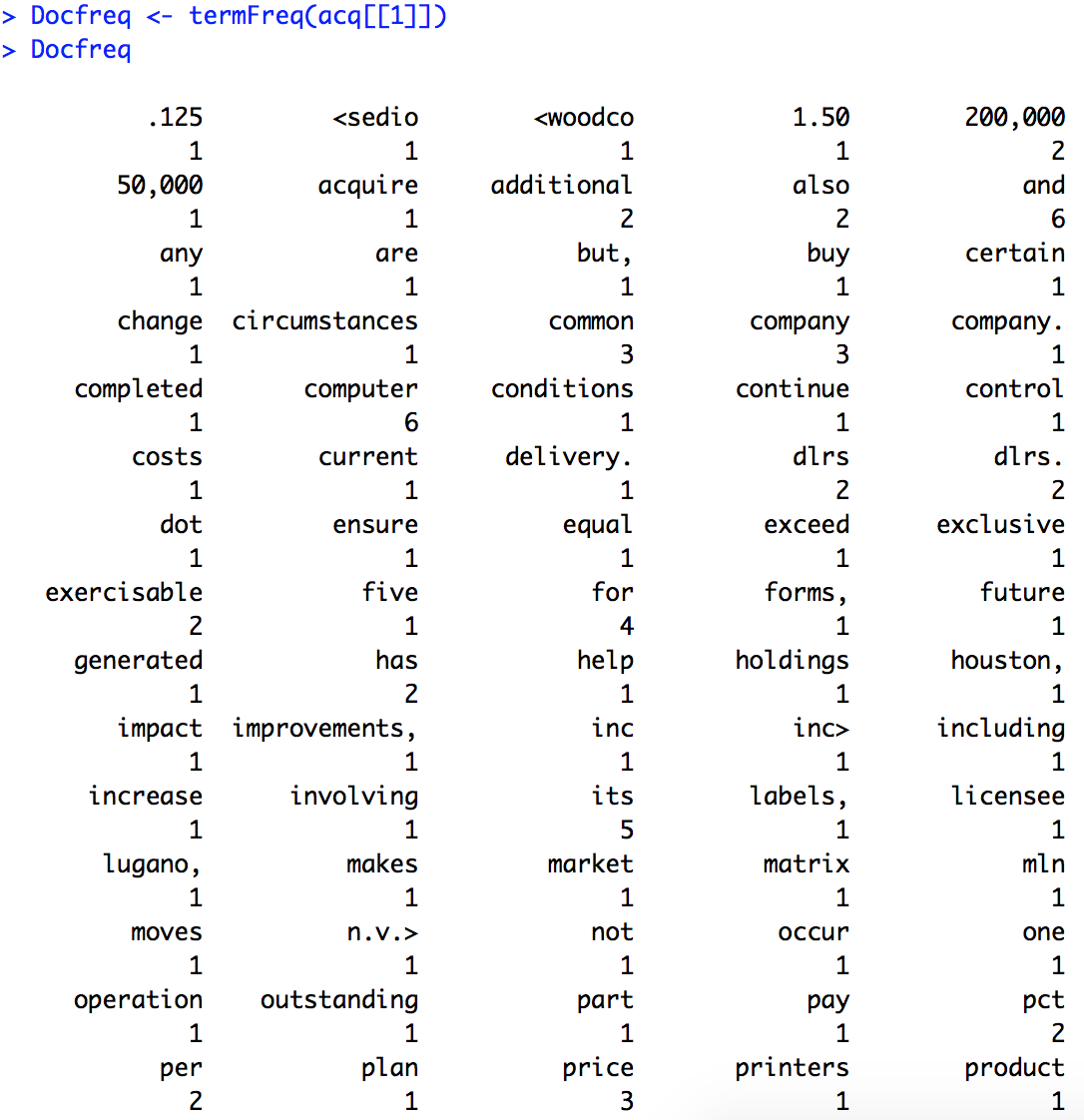


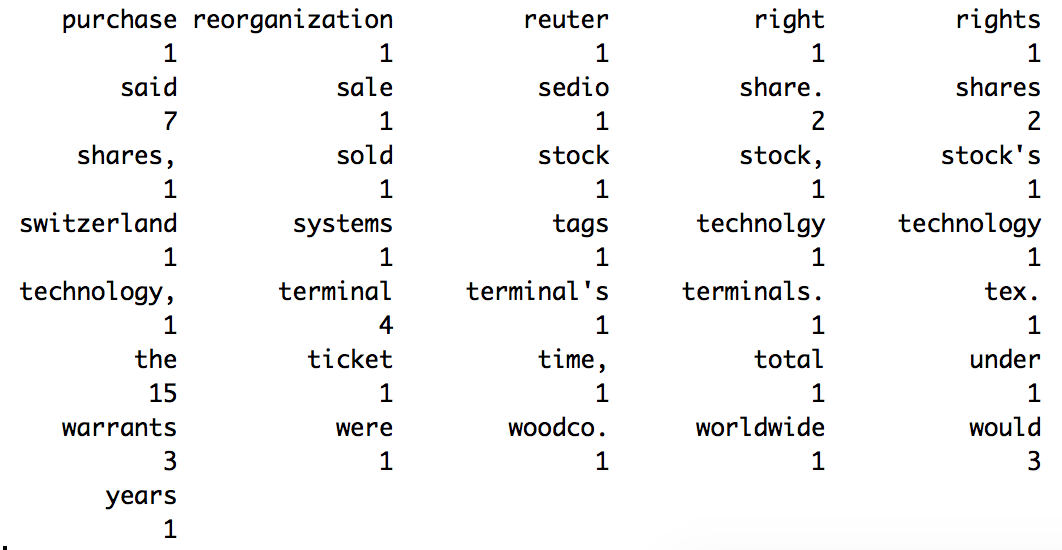
**1.4 inspect() document term matrix**



**1.5 Document Term Frequenccy**

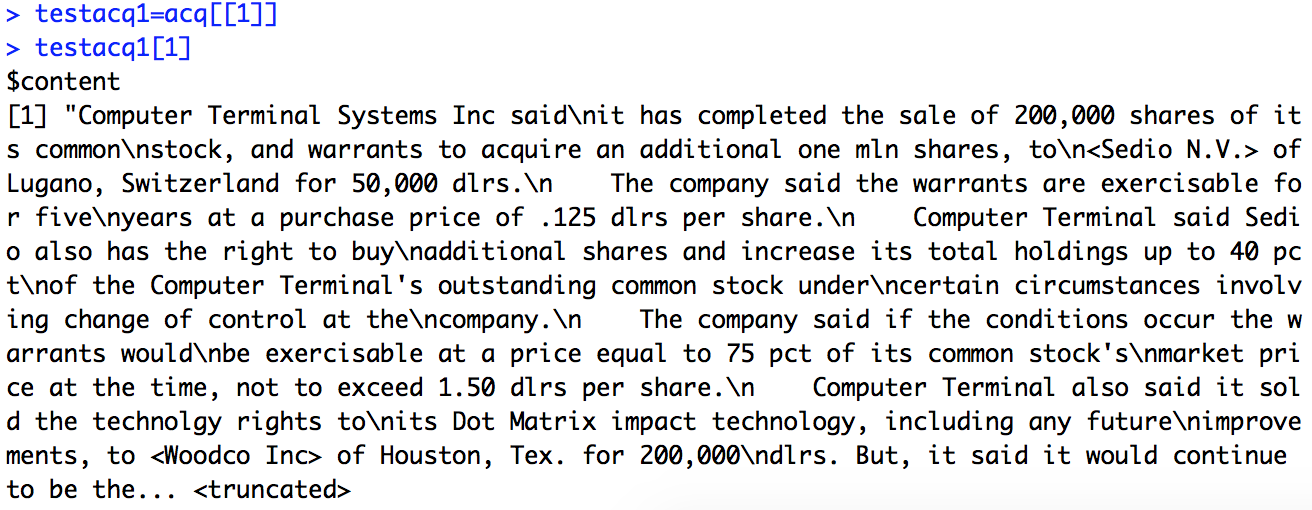
We used termFrq() on the first document to show the word frequency.





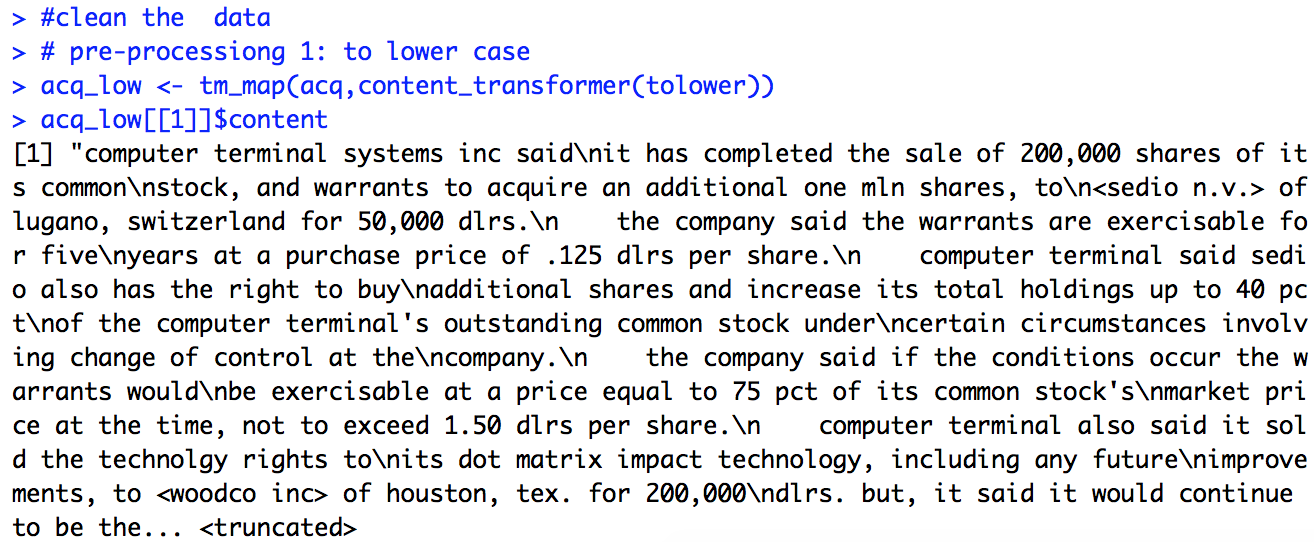
**1.6 Corpus Management**

There are multiple choices of functions in the tm package to conduct corpus management on the documents. By combining tm\_map, content\_transformer function and some other commands, we can do some adjustments on the documents: Since the acq data itself is much too large to be checked the actual implementation of the function, we only see the original words in acq[[1]], to test the result of these function, which is as follows:



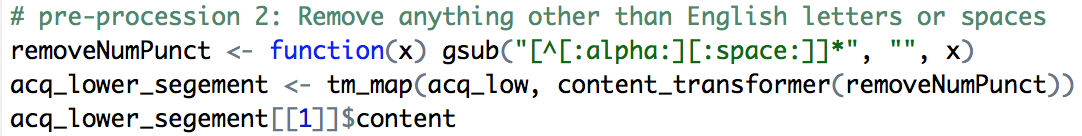
**1.6.1 Convert the corpus to lower case**

As we can see, the first letter of the words such as "Computer", "Terminal, and "System" converted into lower cases. tm\_map() is the function to apply transformation functions (also denoted as mappings) to corpora.



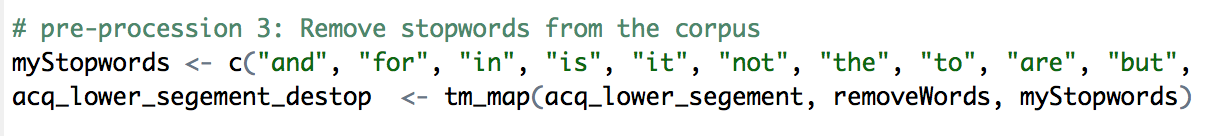
**1.6.2 Remove anything other than English letters or spaces**

The whole puncutation such as periods, commas, and numbers was deleted by in the acq\_low function, which is compared to the acqLow function.

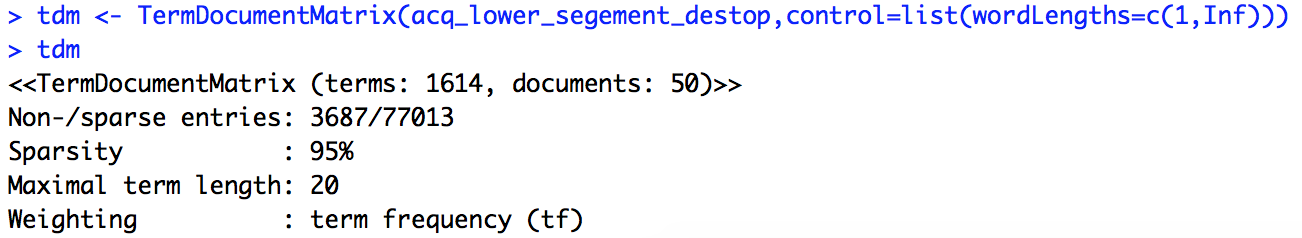


**1.6.3 Remove stopwords from the corpus**

Stopwords are a stop list of 25 semantically non-selective words, including a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with. Comparing to acq\_lower\_segement, the acq\_lower\_segement \_destop deleted all English stopwords in the sentences.

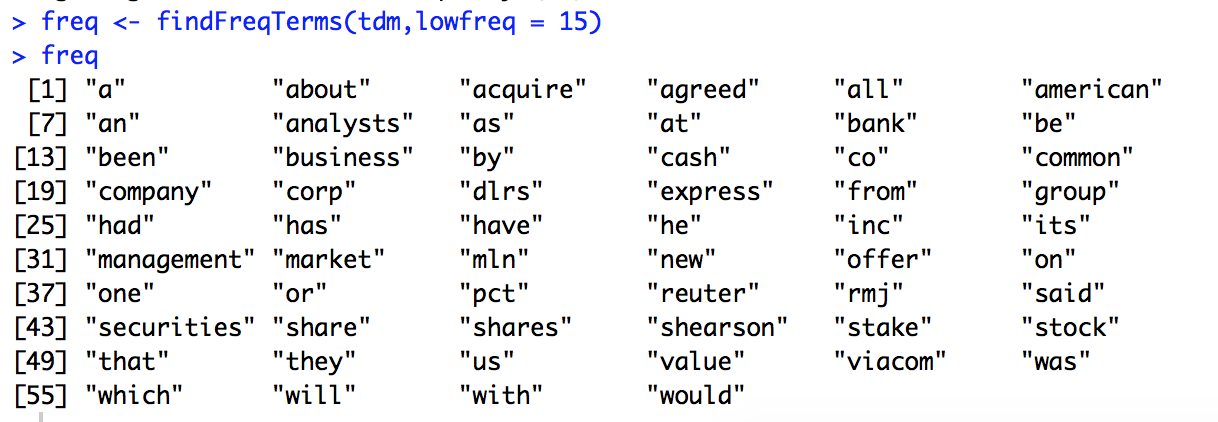


Compare with previous results and show the comparison:

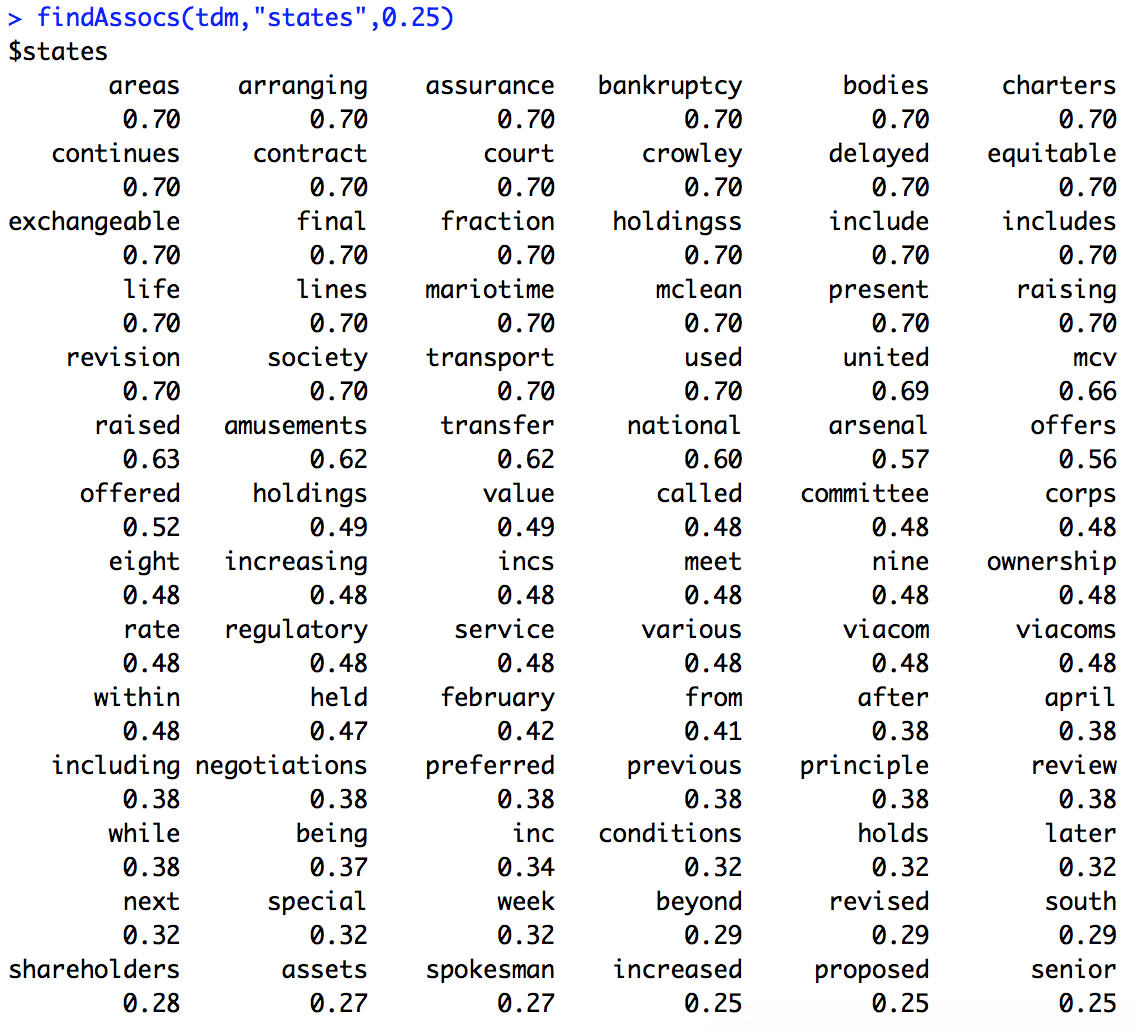


**1.7 Find frequent words**

We try to find the terms have a frequency of 15 or higher. The following list shows the terms which appeared upper and equal than 15 times in acq.

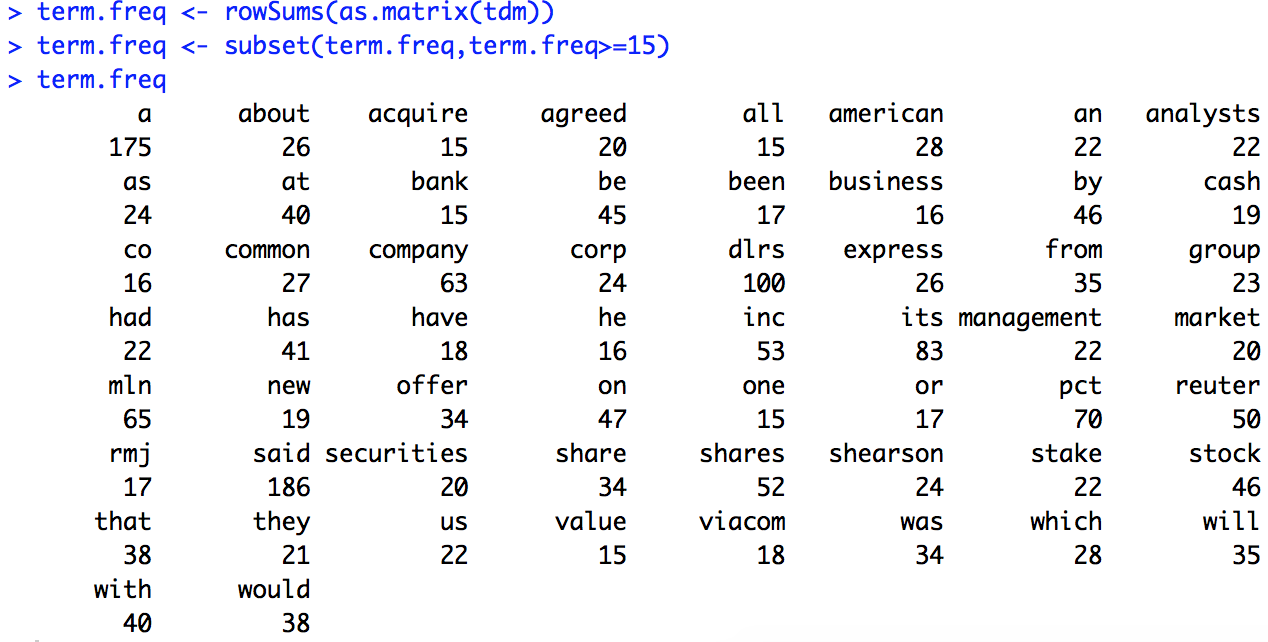


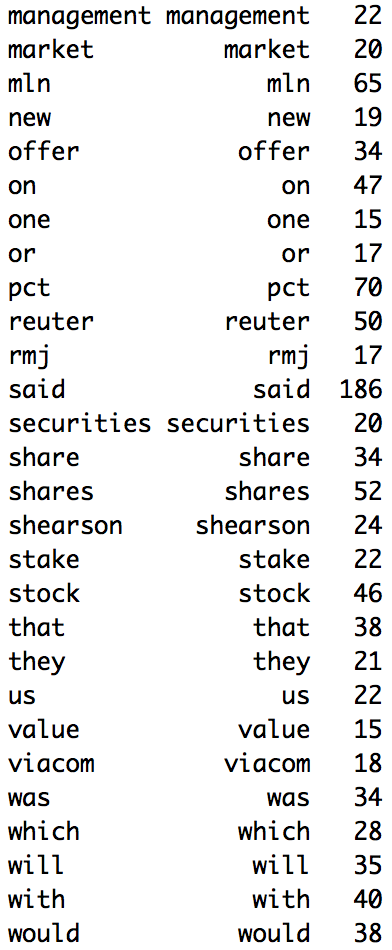
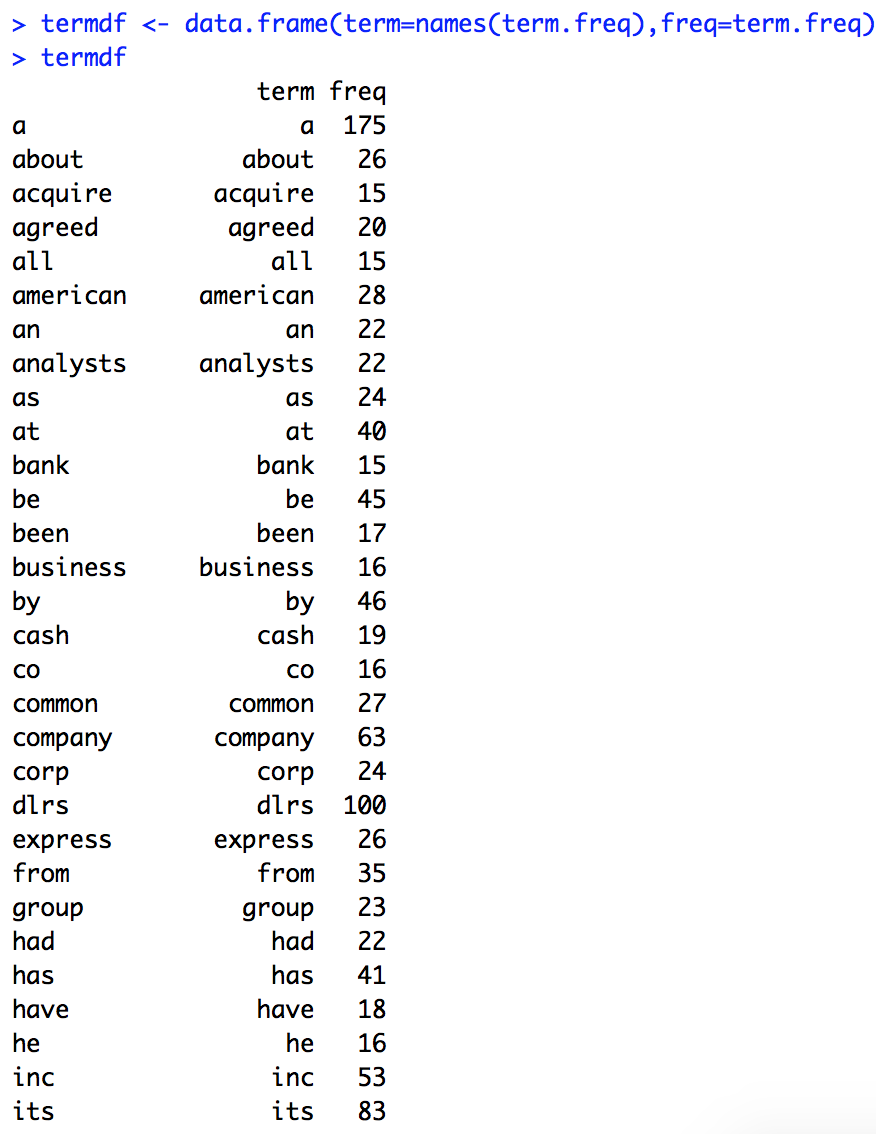
**1.8 Find words associated with "states"**



**1.9 Term Frequency**

We use this function to list the actual frequency of terms with 15 or higher frequency by forming row sums for data frames and returning subsets of data frames, which meet conditions.



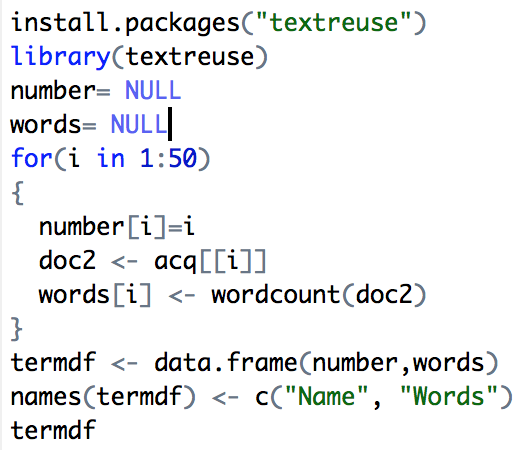


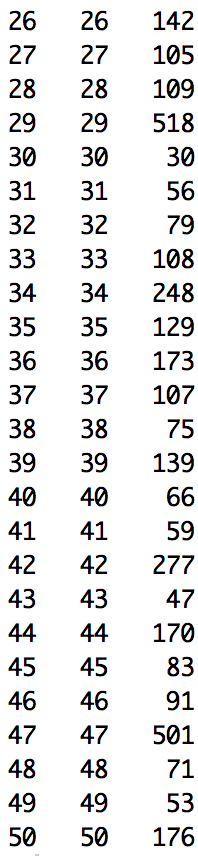
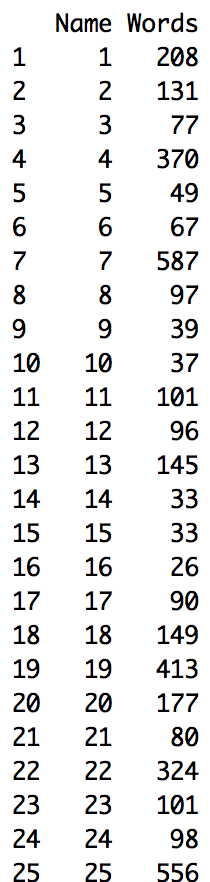
1. **Find the 15 longest documents (in number of words)**

This step we need to use package of textreuse

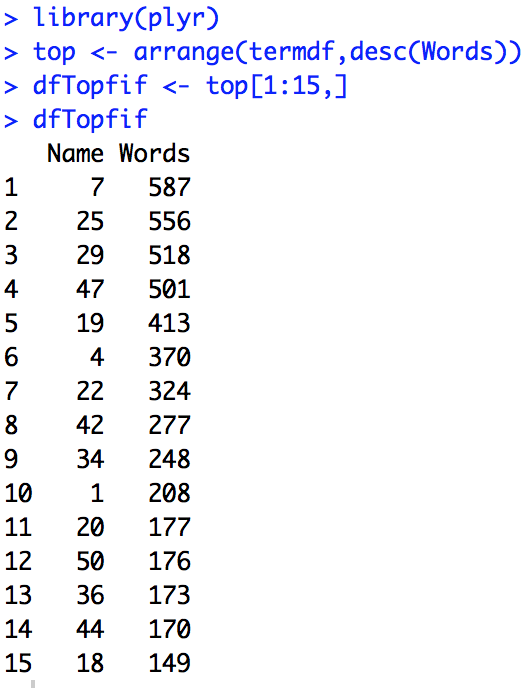
* 1. **We use a loop to count words in every document.**

We record the documents number as the name. Codes and results are as follow:





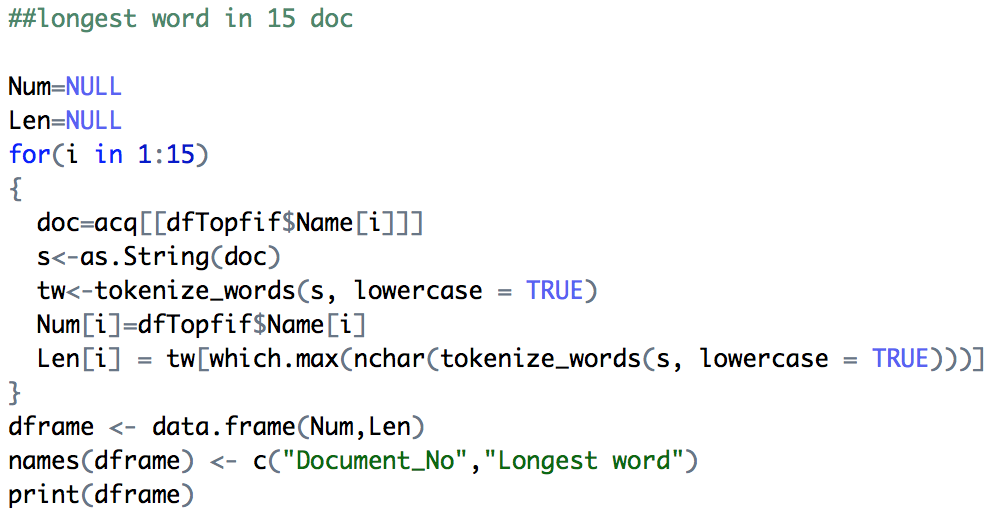
**2.2 we use arrange() to get the data frame by descending order, so the top 15 documents are larger than others.**



1. **Prior to removing the punctuation, find the longest word and longest sentence in each document from the 15 largest documents.**

**3.1 Find the longest word**

We use a loop to operate each document. We get the content for each document and split the whole text into words, and then we find the longest word and record it.

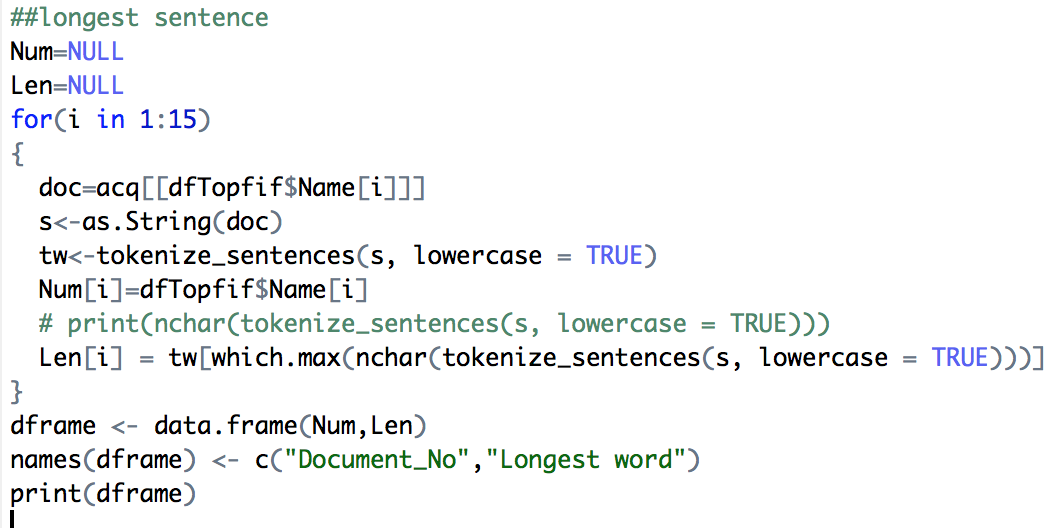


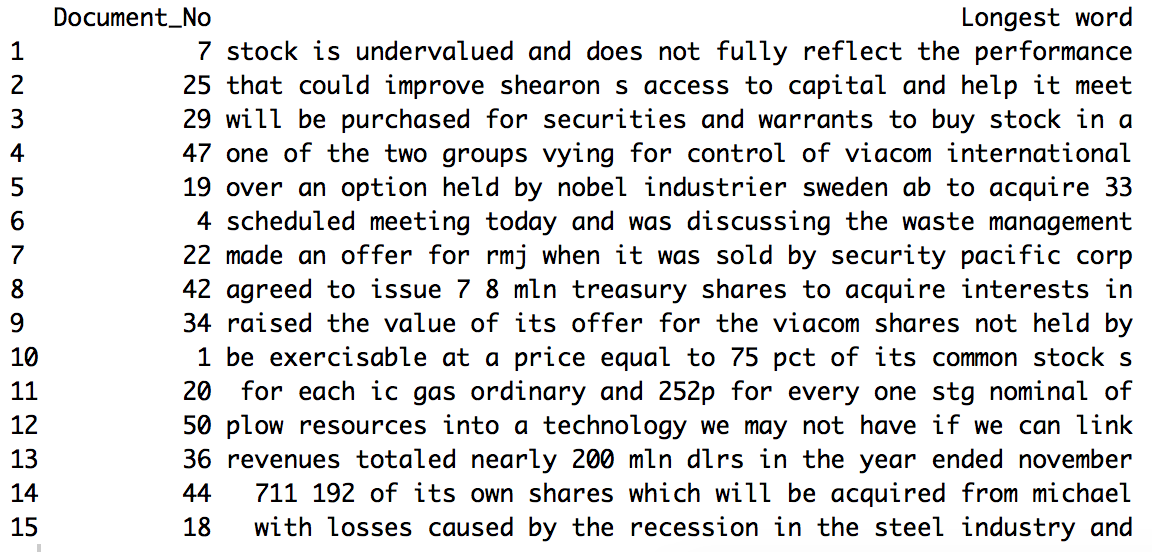
The result is:



**3.2 Find longest sentence**

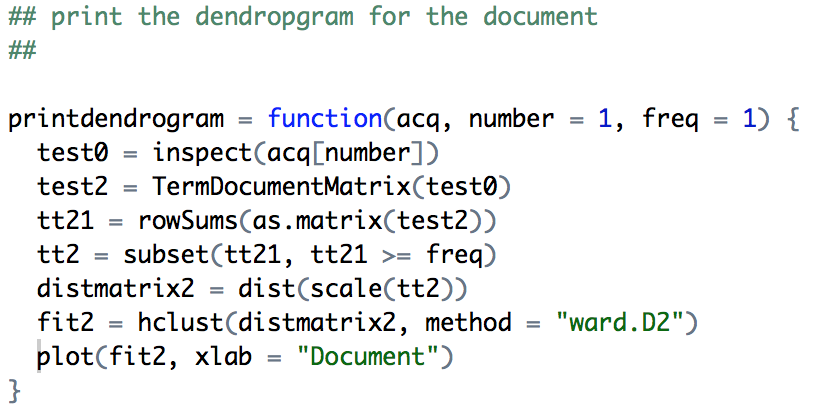
Similarly with finding the longest word, we use a loop to operate each document. We get the content for each document and split the whole text into words, and then we find the longest word and record it.

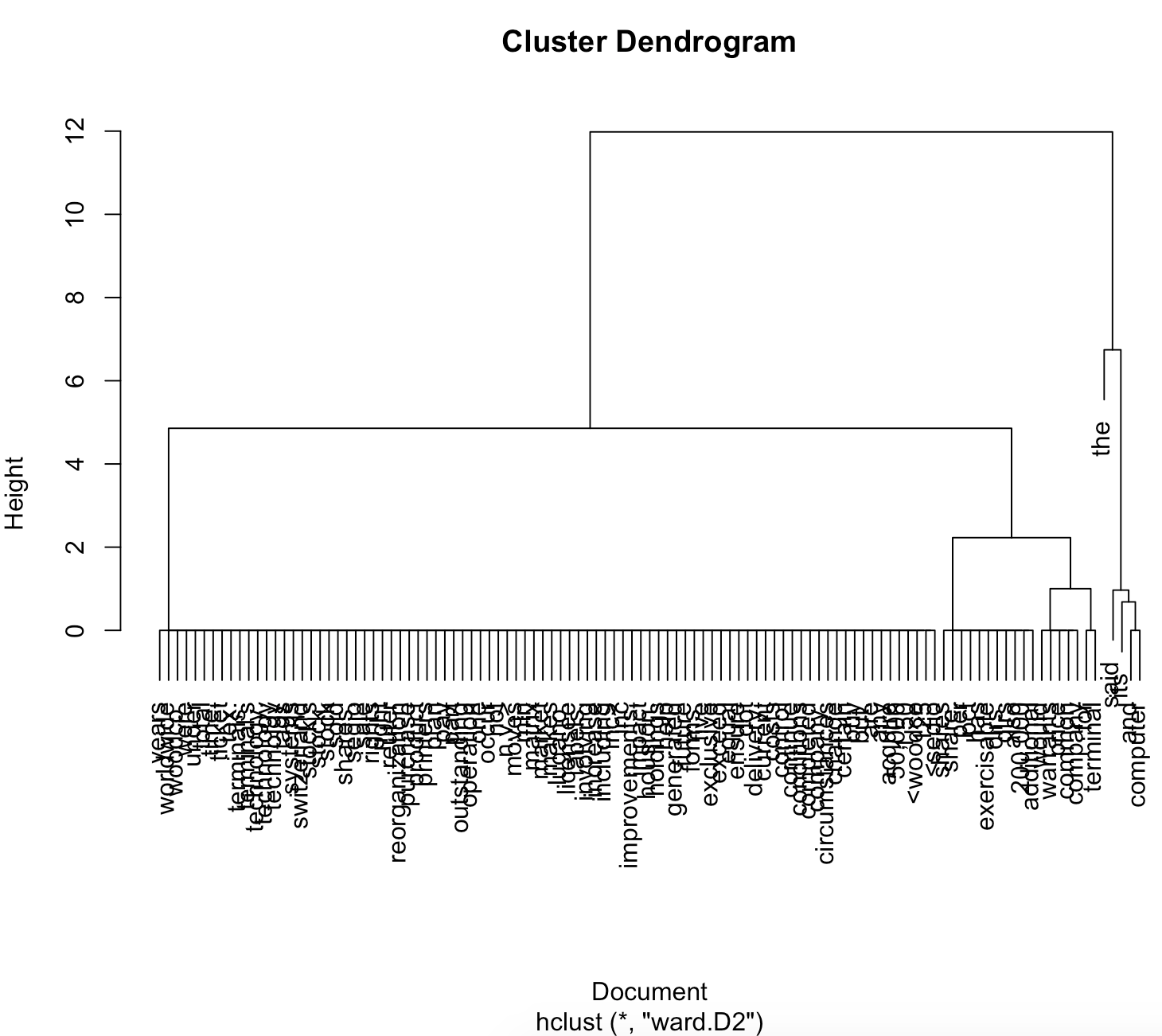




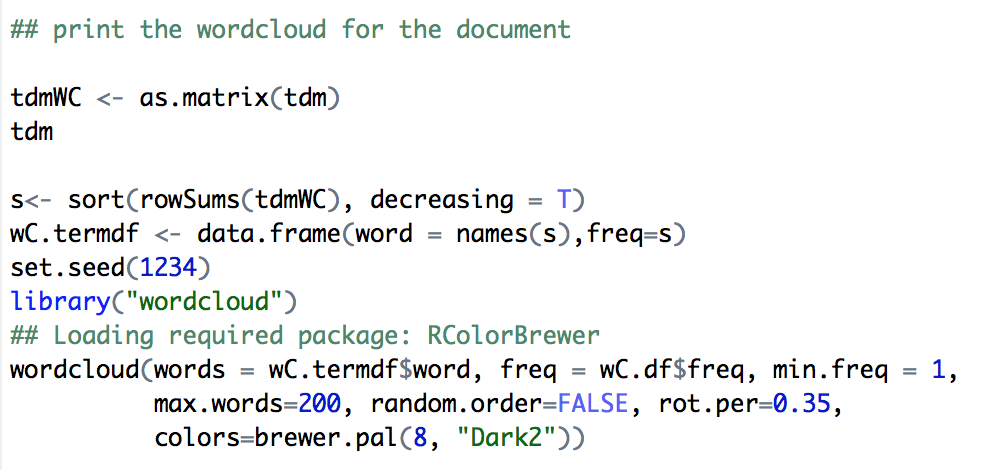
1. **Display the dendrogram and the WordCloud.**

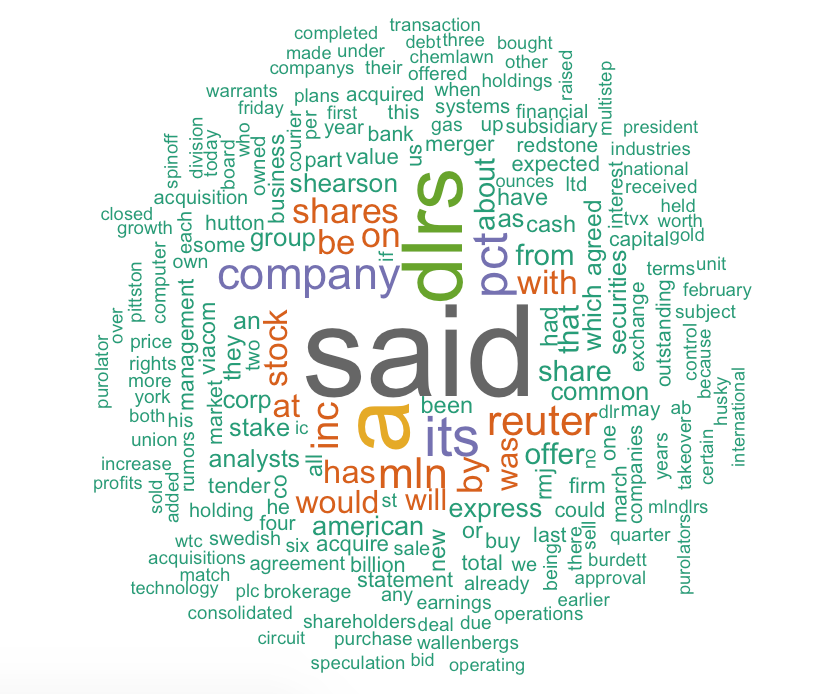
**4.1 Display the dendrogram**





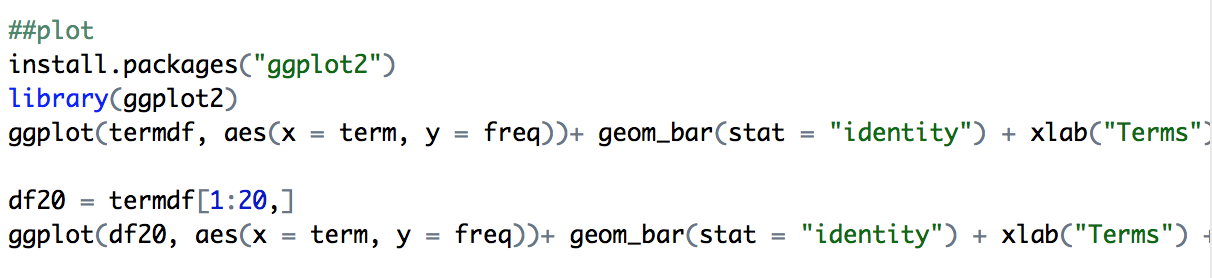
**4.2 Display the WordCloud**

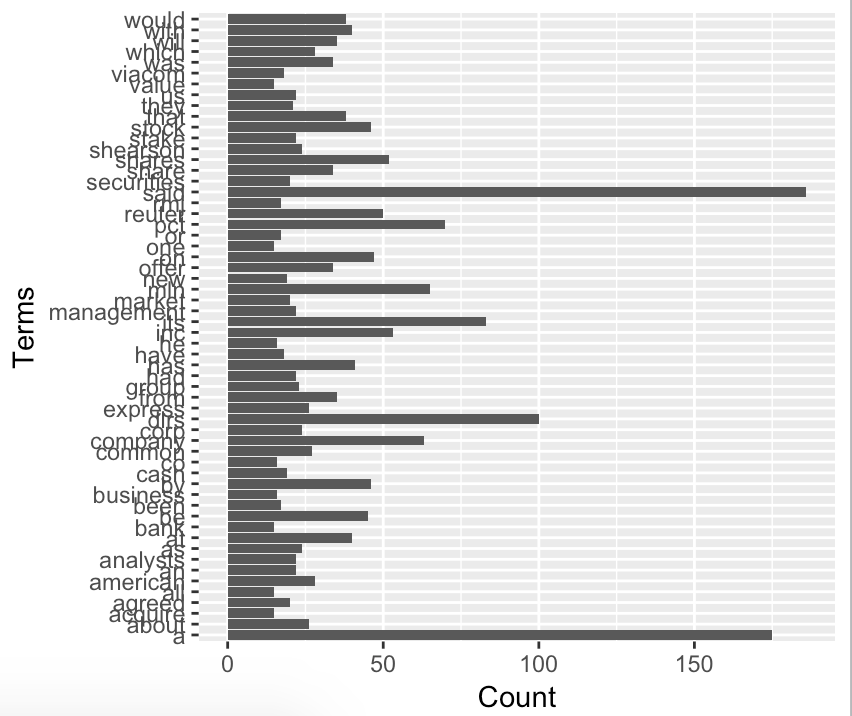




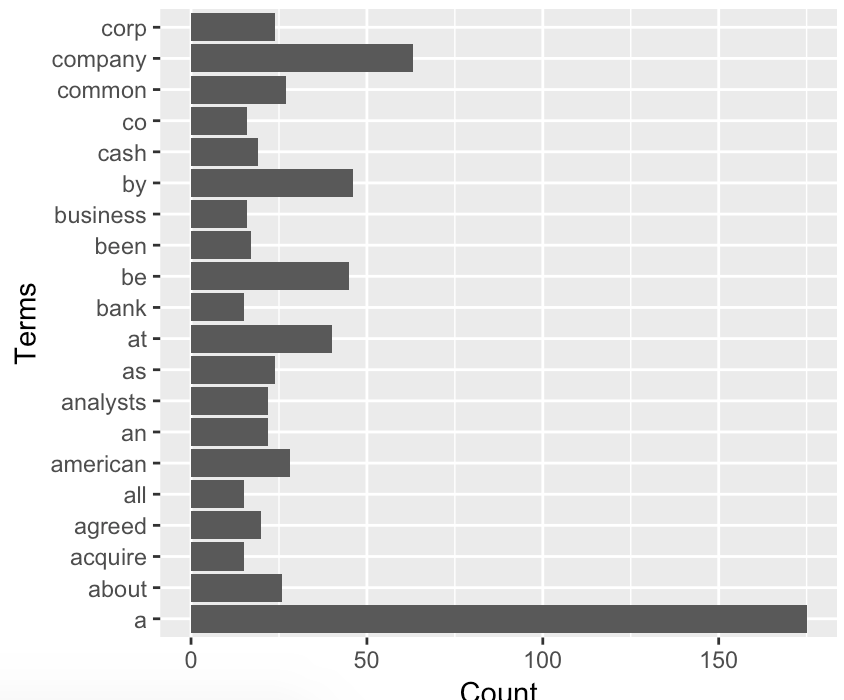
**4.3 Plot other graph to better analysis**

We used the ggplot2() function to plot the term frequency by the term. We could visualize the frequency in a better way.

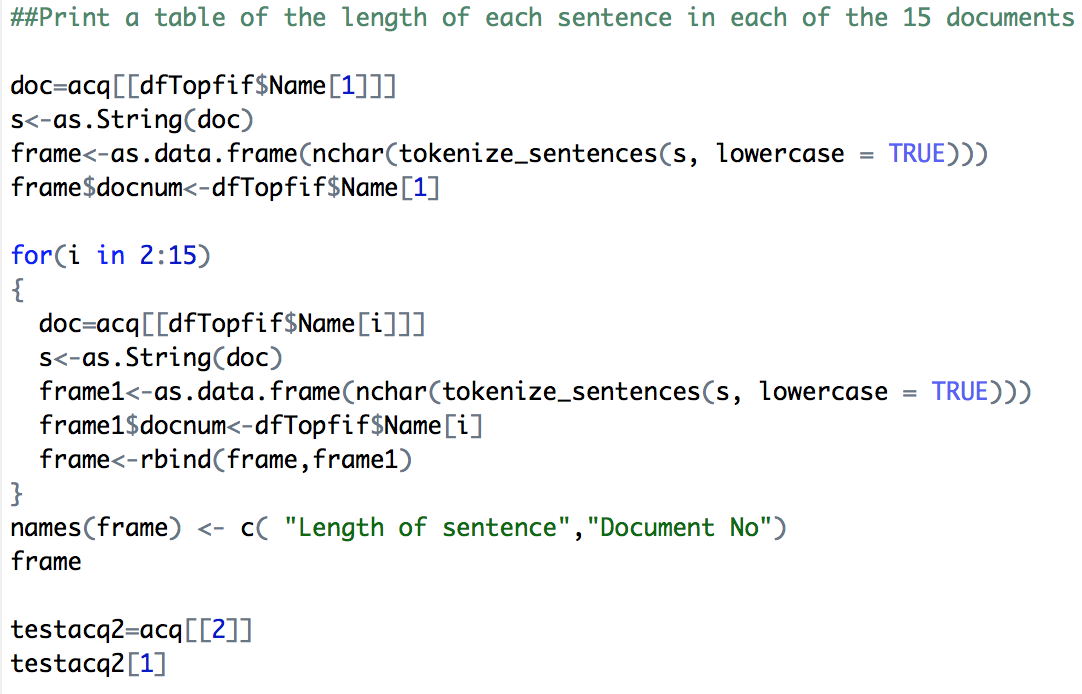


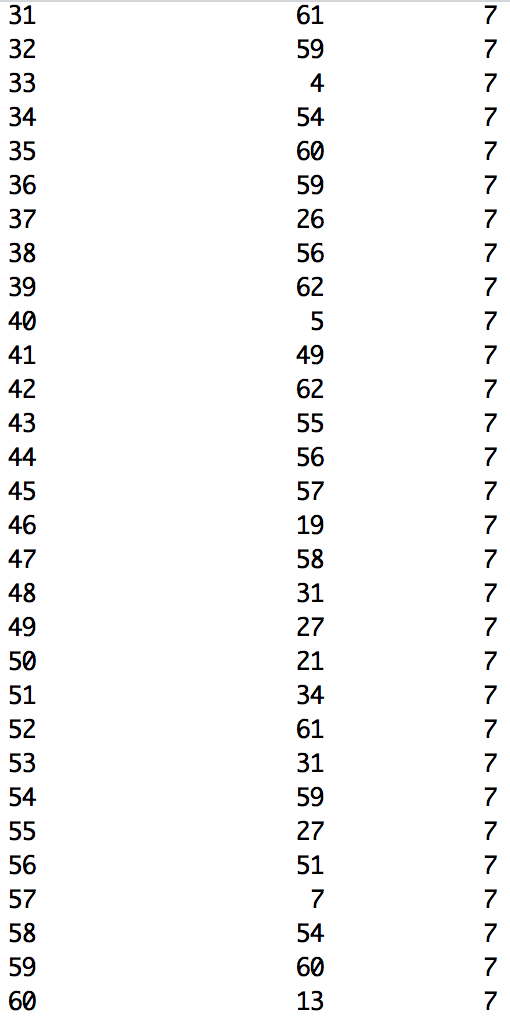
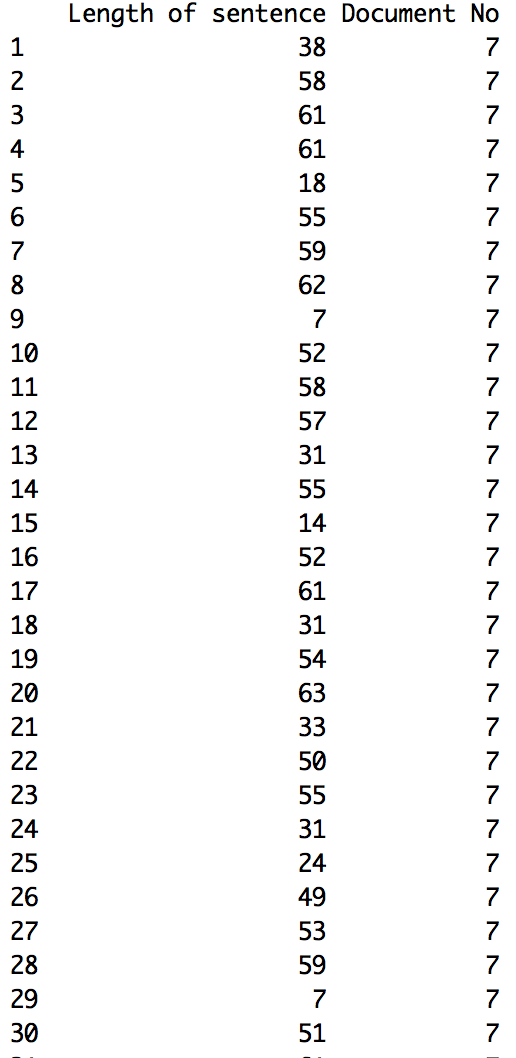


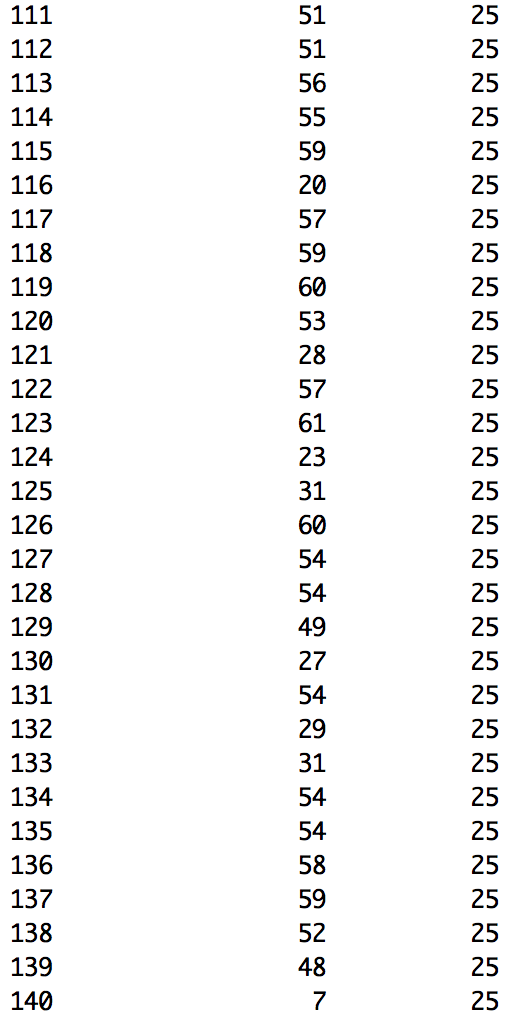
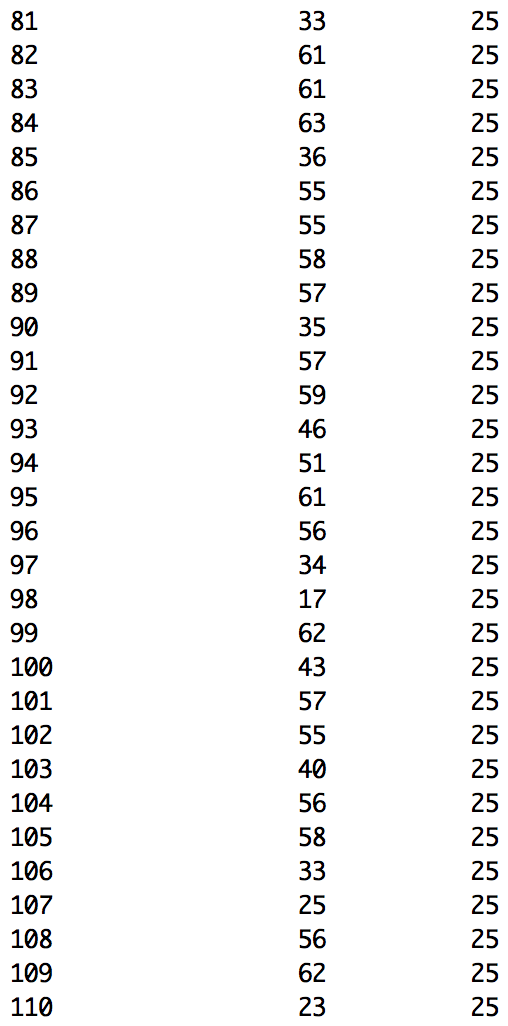
But, we want to focus on the first 20 frequent terms:

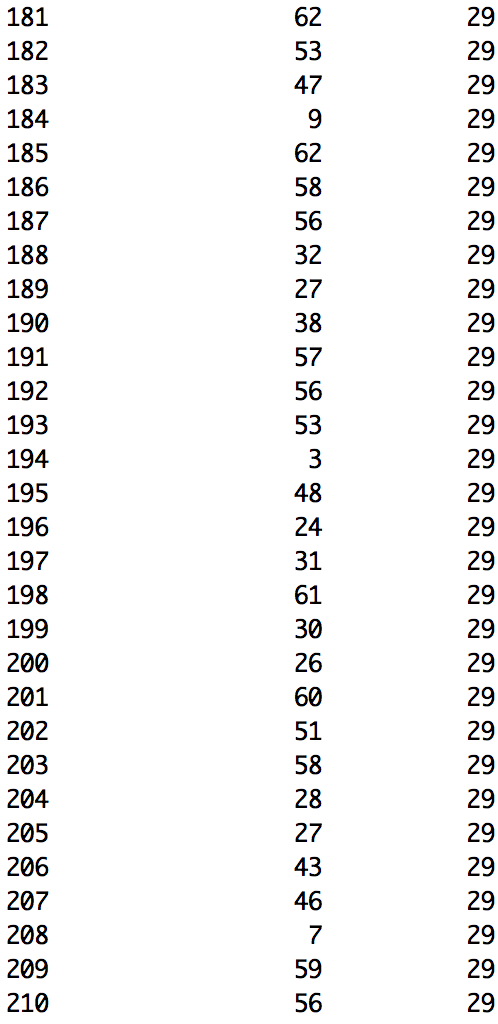
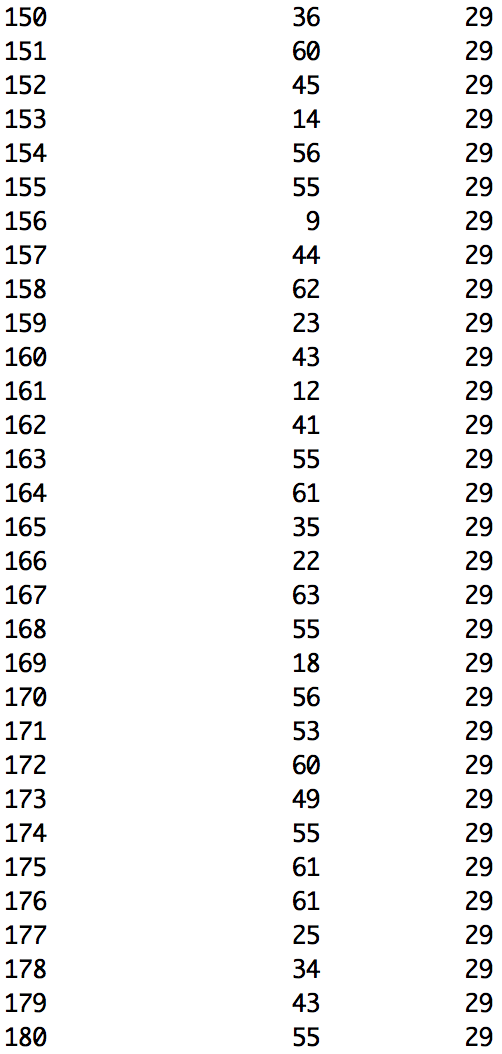


1. **Print a table of the length of each sentence in each of the 15 documents.**







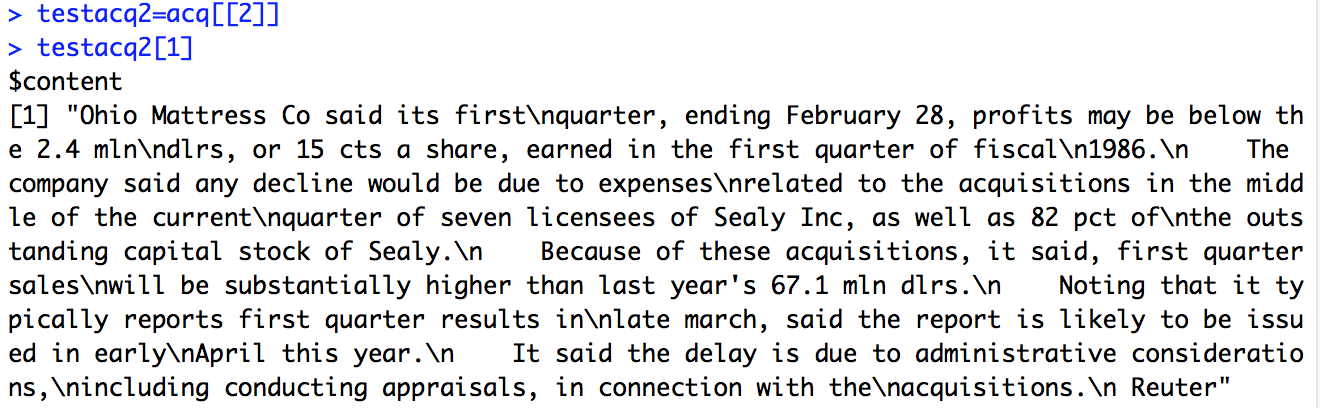


1. **For each sentence of each document, remove the punctuation. Display the sentences.**

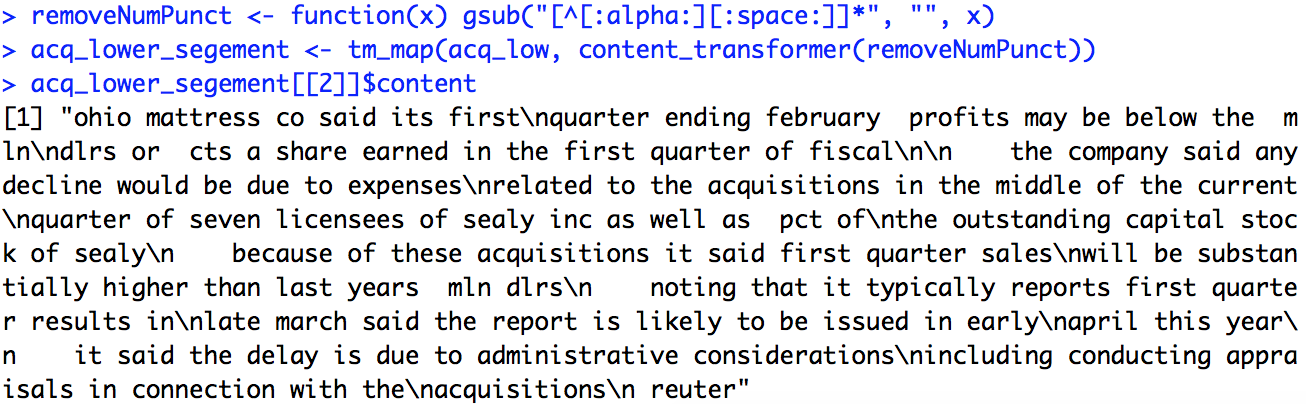
We have already done in the part 1: clean the data.

We can take document 2 as an example.

The original acq[2] is:



after we remove all the punctuation, the result is:



1. **For each word print its part of speech using the Wordnet package.**
2. **Analyze word frequency using functions from package zipfR.**

In this part, we need to install package zipfR and package language. The functions in the zipfR package include N(), V() and Vm() which takes a frequency spectrum object as its input. So we first tokenized each document into words and then converted it into frequency spectrum using a function text2spc.fnc() function of languageR package.

