# Installing and loading the required packages

install\_or\_load\_pack <- function(pack){

create.pkg <- pack[!(pack %in% installed.packages()[, "Package"])] if (length(create.pkg))

install.packages(create.pkg, dependencies = TRUE) sapply(pack, require, character.only = TRUE)

}

packages <- c("ggplot2", "data.table", "wordcloud", "tm", "wordcloud2","scales", "tidytext", "devtools", "twitteR", "caret", "magrittr", "RColorBrewer", "tidytext", "ggdendro","tidyr", "topicmodels", "SnowballC", "gtools")

install\_or\_load\_pack(packages)

Here I specify that I want to load the main packages for dealing with sentiment and discourse analysis in

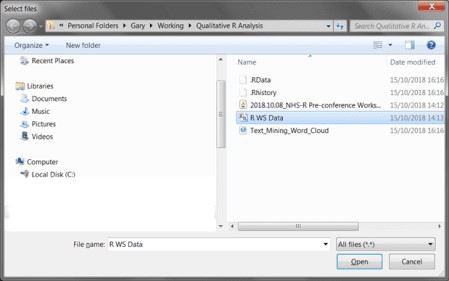
R. Libraries such as **tm**, **wordcloud** and **wordcloud2** are loaded for working with this type of data.

# Choosing the file to import

The file we have to import is a prepared csv file and instead of hard coding the path to load the file from I simply use:

path <- choose\_files()

This is a special function which allows you to open a dialog UI from R:



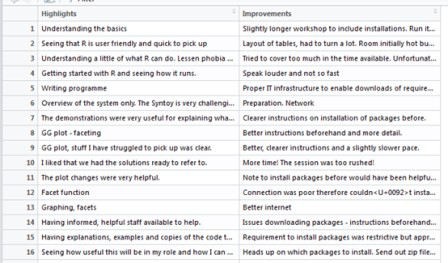
From this dialog I select the csv file I want to be imported. Once I have selected the csv and hit open, the path variable will be filled with the location of the file to work with.

# Creating the R data frame

To create the data frame I can now pass the variable **path** to the read\_csv command:

workshop\_sentiment <- read\_csv(path, col\_names = T)

This will read the textual data from the workshops in to a data frame with 2 columns. The first relates to what the attendees enjoyed about the workshop and the second relates to improvements that can be made:



# Seperating the master data frame

The master data frame now needs to be separated into two separate data frames, as text analysis requires one column with the number of rows for each sentence, as demonstrated. Here I use **magrittr** to divide this into two new data frames:

ws\_highlights <- workshop\_sentiment %>% slice(1)

#Copy for improvements

ws\_improvements <- workshop\_sentiment %>% slice(2)

The **ws\_highlights** data frame uses the first column and the **ws\_improvements** data frame uses the second.

## Function to create textual corpus

As I want to replicate this for highlights and improvements – I have created a function that could be replicated with any text analysis to create what is known as a text corpus (see: https://cran.r-project. org/web/packages/tm/vignettes/tm.pdf) this creates a series of documents, in our case sentences.

corpus\_tm <- function(x){

corpus\_tm <- Corpus(VectorSource(x))

}

This function allows you to pass any data frame to the function and creates a corpus for each data frame you pass to the function. The data frame would be passed to the x parameter.

The **VectorSource()** function creates an element for each part of the corpus.

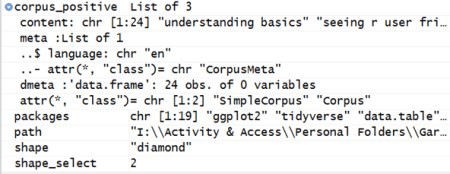
## Create Corpus for Highlights and Improvements data frame

Now the function has been created, I can simpily pass the two separate data frames I created before to

create two corpuses:

corpus\_positive <- corpus\_tm(ws\_highlights$Highlights) corpus\_improvements <- corpus\_tm(ws\_improvements$Improvements)

The code block above shows that I create a corpus for the positive (highlights) data frame and an improvements corpus. This will display as hereunder in your environment:



## Function to clean data in the corpus

The most common cleaning task of working with text data is to remove things like punctuation, common English words, etc. This is something I have to repeat multiple times when dealing with discourse analysis:

clean\_corpus <- function(corpus\_to\_use){ corpus\_to\_use %>%

tm\_map(removePunctuation) %>% tm\_map(stripWhitespace) %>%

tm\_map(content\_transformer(function(x) iconv(x, to='UTF-8', sub='byte')))

%>%

tm\_map(removeNumbers) %>% tm\_map(removeWords, stopwords("en")) %>% tm\_map(content\_transformer(tolower)) %>%

tm\_map(removeWords, c("etc","ie", "eg", stopwords("english")))

}

The parameter here takes the corpus object previously created and uses the corpus passed to:

Remove punctuation

Strip out whitespace between each text item, as the VectorSource has stripped out each word from each sentence in the data frame

Change the underlying formatting of the text to UTF-8 Remove numbers

Remove common English word (stop words) Change the case to lower case

Remove a custom vector of words to adjust for things like e.g., i.e., etc.

To clean the corpus objects I simply pass the original corpus objects back through this function to perform cleaning:

corpus\_positive <- clean\_corpus(corpus\_positive) corpus\_improvements <- clean\_corpus(corpus\_improvements)

Inspection of one of the data frames confirms that this has successfully been cleaned:

## Create TermDocumentMatrix to attain frequent terms

The term document matrix can be used with the corpus to identify frequent terms by classification on a matrix. However, more code is needed to do this:

find\_freq\_terms\_fun <- function(corpus\_in){ doc\_term\_mat <- TermDocumentMatrix(corpus\_in)

freq\_terms <- findFreqTerms(doc\_term\_mat)[1:max(doc\_term\_mat$nrow)] terms\_grouped <- doc\_term\_mat[freq\_terms,] %>%

as.matrix() %>% rowSums() %>%

data.frame(Term=freq\_terms, Frequency = .) %>% arrange(desc(Frequency)) %>% mutate(prop\_term\_to\_total\_terms=Frequency/nrow(.))

return(data.frame(terms\_grouped))

}

This function needs explanation. The function uses as a single parameter the corpus that you need to pass in, then a variable is created to create the **doc\_term\_mat** which uses the tm TermDocumentMatrix.

Next, I use the **findFreqTerms** function to iterate from the first entry to the maximum number of rows in the matrix. These are the powerhouses of the function, as they highlight how many times a word has been used in a sentence across all the rows of text.

The **terms\_grouped** variable then slices the term matrix with the frequent terms, this is converted to a matrix, sum of each row are calculated i.e. the number of times the word appears. Then, a data frame is created of the terms in the function with the headings *term* and *Frequency*.

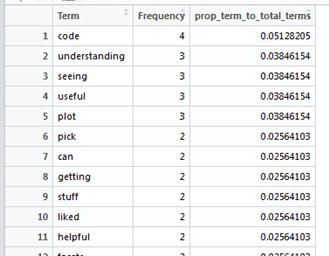
Next, we use the power of **DPLYR** to use arrange by the frequency descending and to add a mutated column to the data frame to calculate the proportion of that specific term over all terms.

The **return(data.frame(terms\_group))** then forces R to return the results of the function.

I then pass my data frames (highlights and improvements) to the function I have just created to see if this method works:

positive\_freq\_terms <- data.frame(find\_freq\_terms\_fun(corpus\_positive)) improvement\_freq\_terms <- data.frame(find\_freq\_terms\_ fun(corpus\_improvements))

These will be built as data frames and can be viewed in R Studio’s Data environment window:



This has worked just as expected. You could now use **ggplot2** to produce a bar chart / pareto chart of the terms.

## Create a Word Cloud with the wordcloud2 package

R has a **wordcloud** package that produces relatively nice looking word clouds,

but **wordcloud2** surpasses this in terms of visualisation. To use this function is easy now I have the frequent terms data frame – using the highlights data frame this can be implemented by using the below syntax:

wordcloud2(positive\_freq\_terms[,1:2], shape="pentagon",

color="random-dark")

To use the function I pass the data frame and use the term and frequency fields only to use the visualisation. There are a number of options and these can be accessed by using

the **help(“wordcloud2”)** function. Here I use the shape and colour parameters to set the display of the word cloud:



This can be exported in the viewer window by using the Export function:



This word cloud relates to the pre-workshop before the conference in 2018. I personally thought the NHS-R conference was amazing and I was honoured to have a spot to speak amongst so many other brilliant R users.

R is so versatile – every day is like a school day when you are learning it, but what a journey.