working_with_text

First, install the packages you will likely need to conduct the text analysis.

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(tidytext)
library(stringr)
```

Text analysis

In this tutorial we are going to start working with text. We will take the tidy approach to text analysis because it is fairly intuitive in that it treats text as data frames for analysis. Furthermore, the tibble data frames created in tidy are small and don't require as much processing power.

The text converted into a data frame can be converted to a variety of formats that allow it to be analyzed by other popular r programs such as quanteda. While the examples are different and we won't cover all of the topics in the book you can find much of this material in **Text Mining with R: A Tidy Approach** listed in the syllabus

This image from **Text Mining with R: A Tidy Approach** demonstrates the general overall process for text analysis using tidyverse:

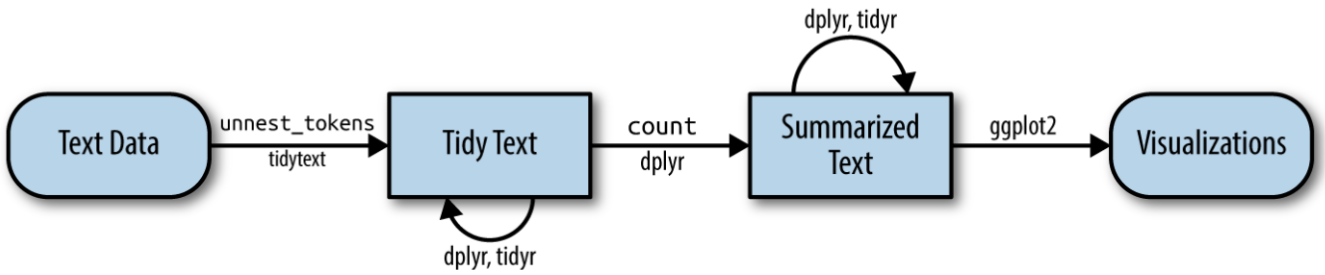


Figure 1.1: A flowchart of a typical text analysis using tidy data principles. This chapter shows how to summarize and visualize text using these tools.

A Corpus

For our analysis we need text. The raw text is the corpus that we work on. The corpus that we are interested in varies depending on our research objectives. For example, a corpus can contain a newspaper articles, multiple articles, paragraphs from articles, tweets about a particular topic or from a group of people, text from pdf documents that we have collected etc. How we put together the relevant corpus depends on the objective of our analysis.

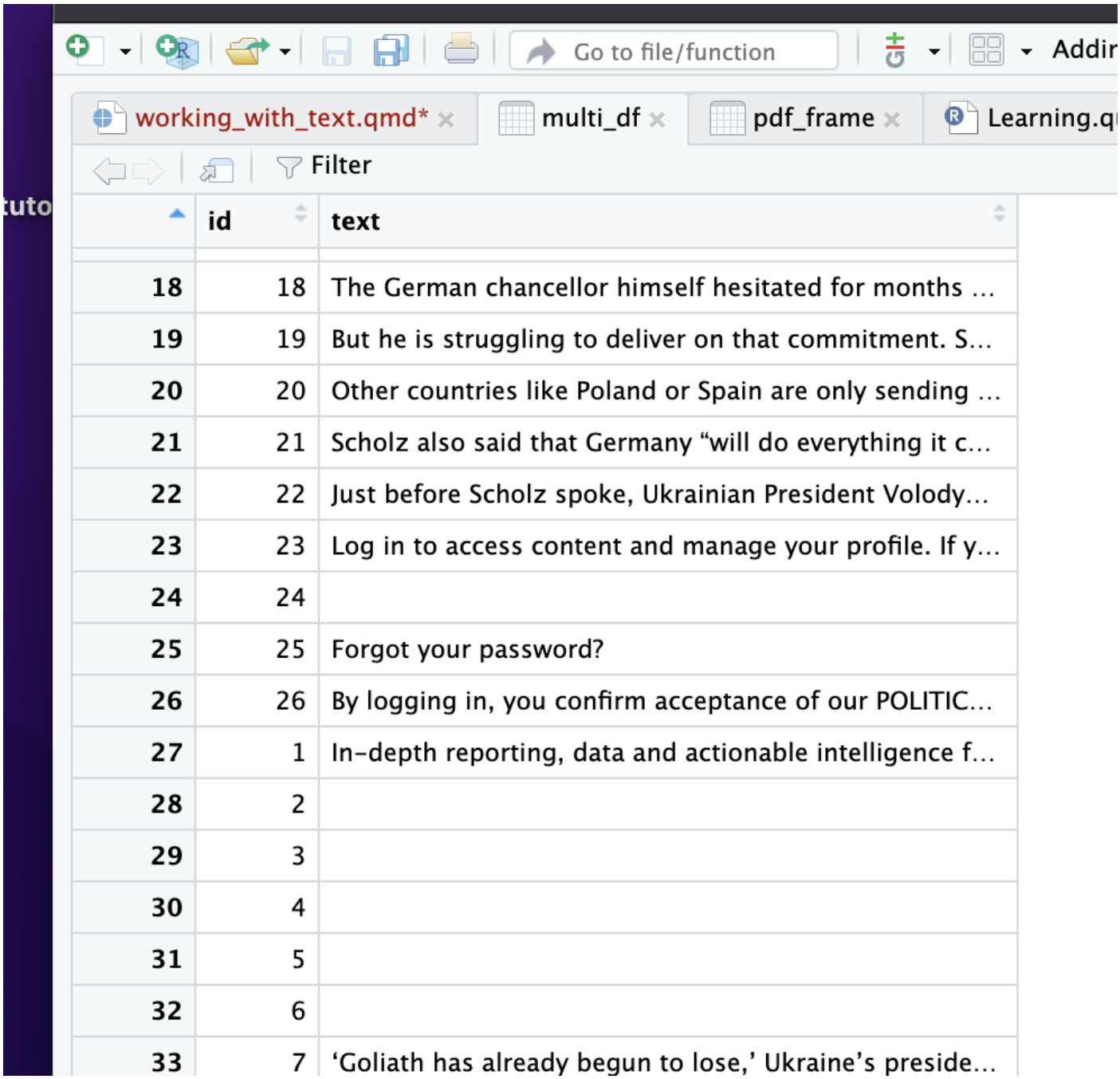
Today we will work with some data that has already been provided to us. The first is text from newspaper articles that were previously scraped and the paragraphs collected into a dataframe.

The first step is to load the data. It was saved as an R Object, so we can load it the following way:

```
load("multiple_url_paragraphs.RData")
```

This loads the R data object into our working environment to allow us to begin working with the data. As you can see, it's loaded as, multi_df.

The multi_df data consists of a number of newspaper paragraphs collected from the web and an id that identifies the number of the paragraph in a given article. The id numbers repeat for each article-paragraph.



	id	text
18	18	The German chancellor himself hesitated for months ...
19	19	But he is struggling to deliver on that commitment. S...
20	20	Other countries like Poland or Spain are only sending ...
21	21	Scholz also said that Germany “will do everything it c...
22	22	Just before Scholz spoke, Ukrainian President Volody...
23	23	Log in to access content and manage your profile. If y...
24	24	
25	25	Forgot your password?
26	26	By logging in, you confirm acceptance of our POLITIC...
27	1	In-depth reporting, data and actionable intelligence f...
28	2	
29	3	
30	4	
31	5	
32	6	
33	7	‘Goliath has already begun to lose,’ Ukraine’s preside...

Cleaning

To work with the unmodified corpus data we need to render it into a format that we can use for text analysis. Specifically, the data we need to turn it into a tokenized cleaned dataframe.

For this tutorial, we will be using the tidyverse text analysis method because it is very intuitive so begin by loading the tidytext package and installing it if not installed yet. This will be one of the core packages needed for our work.

```
# install.packages("tidytext")
library(tidytext)
```

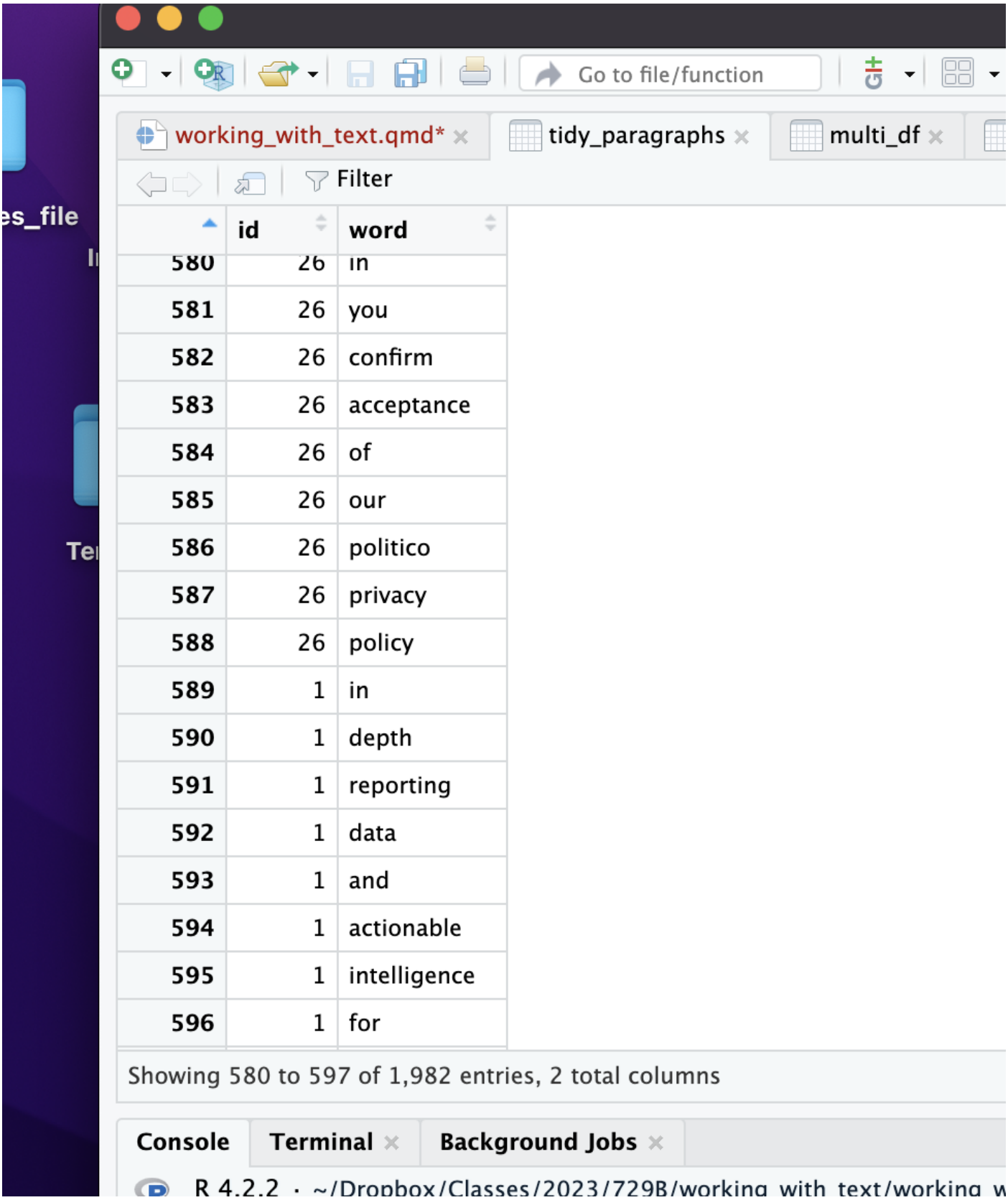
Now we can begin to work with the data.

First we want to transform the corpus into something more usable usable by tokenizing the text.

By tokenizing the corpus we are separating the paragraphs and sentences into words instead of the format they are currently in.

```
tidy_paragraphs <- multi_df %>%
  unnest_tokens(word, text)
```

If we look at tidy_paragraphs, we can see that the text is broken down by word and noted by the paragraph it is found in. The paragraph id repeats for each article-paragraph-word that we downloaded that contains text.



Now we want to begin cleaning the corpus by removing unnecessary words. These are words that convey no meaningful information but appear frequently, usually referred to as “stop words” such as but, and, if, then, and similar words.

```
tidy_paragraphs <- tidy_paragraphs %>%
  anti_join(stop_words)
```

```
Joining with `by = join_by(word)`
```

This uses the “stop words” dataset from tidyverse to remove the most common stop words.

We can then look at the top words in our articles using the following function:

```
tidy_paragraphs %>%
  count(word, sort = TRUE)
```

This shows that the top words in these articles, with stop words removed, are Munich, western, Ukraine, war, countries, and Germany.

You can also clean words/spaces/numbers that aren’t stop words. This can be especially useful if you accidentally include advertisements or other unnecessary data in your scraping.

```
tidy_paragraphs <-tidy_paragraphs %>%
  filter(!word %in% c("Politico","reporting")) #Add in the word you'd like to remove here
```

You can also clean out any numbers with the following code.

```
tidy_paragraphs <-tidy_paragraphs %>%
          filter(!grepl("[0-9]+", word))
#filters out non-grepl (string) digits from 0-9 in any form in the word column
```

Cleaning is a lengthy and involved process that differs between every project and depends on the data you are using. Take your time cleaning the data.

Analyzing the Data

Next we want to analyze the data.

Wordcloud Visualization

One common way to analyze the data is to visualize them.

For example, the most commonly used words can be visualized using a word cloud. Two packages, devtools and wordcloud2, are needed to do this.

```
library(wordcloud2)
library(devtools)
```

```
Loading required package: usethis
```

A wordcloud displays the words in the corpus by size according to how often the words occur in the corpus. To show this we first must count the words.

Words that occur more frequently will be larger and the ones that occur less frequently will be smaller.

```
t_para <- tidy_paragraphs %>%
  count(word, sort = TRUE)
```

Note that the new object (t_para) counts words in the entire tokenized corpus we fed to it (tidy_paragraphs) and did NOT treat each article as a separate entity. You can make a very simple wordcloud using the wordcloud2() function.

```
wordcloud2(t_para)
```

This then outputs a simple wordcloud of the most commonly use words in our articles.

Word sequences N-grams

Alternatively we may want to establish relationships between words. For example, by seeing how often word X is followed by word Y, we can build a model of the relationships between them.

To do this we go back to the un-tokenized corpus multi_df.

We can then use the function unnest_tokens as used earlier in order to tokenize into consecutive sequences of words (as opposed to single words), called n-grams.

We do this by adding the token = “ngrams” option to unnest_tokens(), and setting n to the number of words we wish to capture in each n-gram. When we set n to 2, we are examining pairs of two consecutive words, often called “bigrams”:

```
tidy_bigrams <- multi_df %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  filter(!is.na(bigram)) #NOTE here we are filtering out any words that are not a part of
```

Our usual tidy tools apply equally well to n-gram analysis. We can examine the most common bigrams using dplyr’s count():

```
tidy_bigrams %>%
  count(bigram, sort = TRUE)
```

It then shows the pairs of words that frequently appear together. Looking at it, you can see that the words are often two stop words and thus, uninteresting. We can clean the data in order to filter the stop words and focus on the interesting text.

This is a useful time to use tidyr’s separate(), which splits a column into multiple based on a delimiter. This lets us separate it into two columns, “word1” and “word2”, at which point we can remove cases (rows) where either is a stop-word.

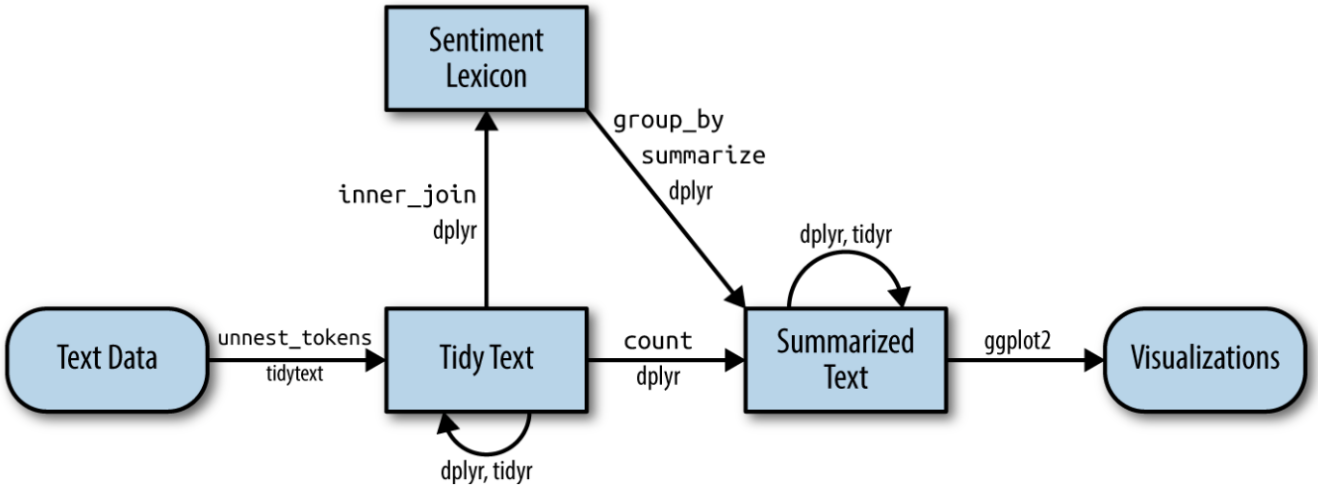
```
library(tidyr)

bigrams_separated <- tidy_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")

bigrams_filtered <- bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)
```

Now we can see the more interesting results:

	word1	word2	n
1	central	europe	4
2	central	europeans	4
3	leopard	2	4
4	munich	security	4
5	security	conference	4
6	access	content	3
7	actionable	intelligence	3
8	confirm	acceptance	3
9	depth	reporting	3
10	german	chancellor	3

Sentiment Analysis

When human readers approach a text, we use our understanding of the emotional intent of words to infer whether a section of text is positive or negative, or perhaps characterized by some other more nuanced emotion like surprise or disgust. We can use the tools of text mining to approach the emotional content of text programmatically, as shown below:



There alrerady exist quite a few existing dictionaries and methods for sentiment analysis that make our work easier. The tidytext package provides access to several ready made sentiment lexicons.

All three of these lexicons are based on unigrams, i.e., single words. These lexicons contain many English words and the words are assigned scores for positive/negative sentiment, and also possibly

emotions like joy, anger, or sadness.

The function get_sentiments() allows us to get specific sentiment lexicons with the appropriate measures for each one.

The following code is an example of how to use an existing sentiment dictionary to analyze the text we worked with earlier. In order to make the analysis more interesting, we can separate the analysis by document. To do this, we will use data scraped from some pdf files.

```
load("pdf_frame1.RData")

pdf_frame1=pdf_frame1%>%
  select("document", "term")
```

Looking at this dataframe you see that it contains 2 column 1 for the document number, one for the document-word.

First, rename the “term” column in pdf_frame1 to word so it’s easier to join with the sentiment dictionary.

```
colnames(pdf_frame1)[2] ="word"
```

Now, run this code to join the sentiment dictionary with the scraped text and derive a sentiment score. This uses the “bing” sentiment dictionary, which comes with the tidyverse packages:

```
multi_doc_sentiment <- pdf_frame1 %>%
  inner_join(get_sentiments("bing"), by="word") %>%
  count(document, sentiment) %>%
  pivot_wider(names_from = sentiment, values_from = n, values_fill = 0) %>%
  mutate(sentiment = positive - negative)
```

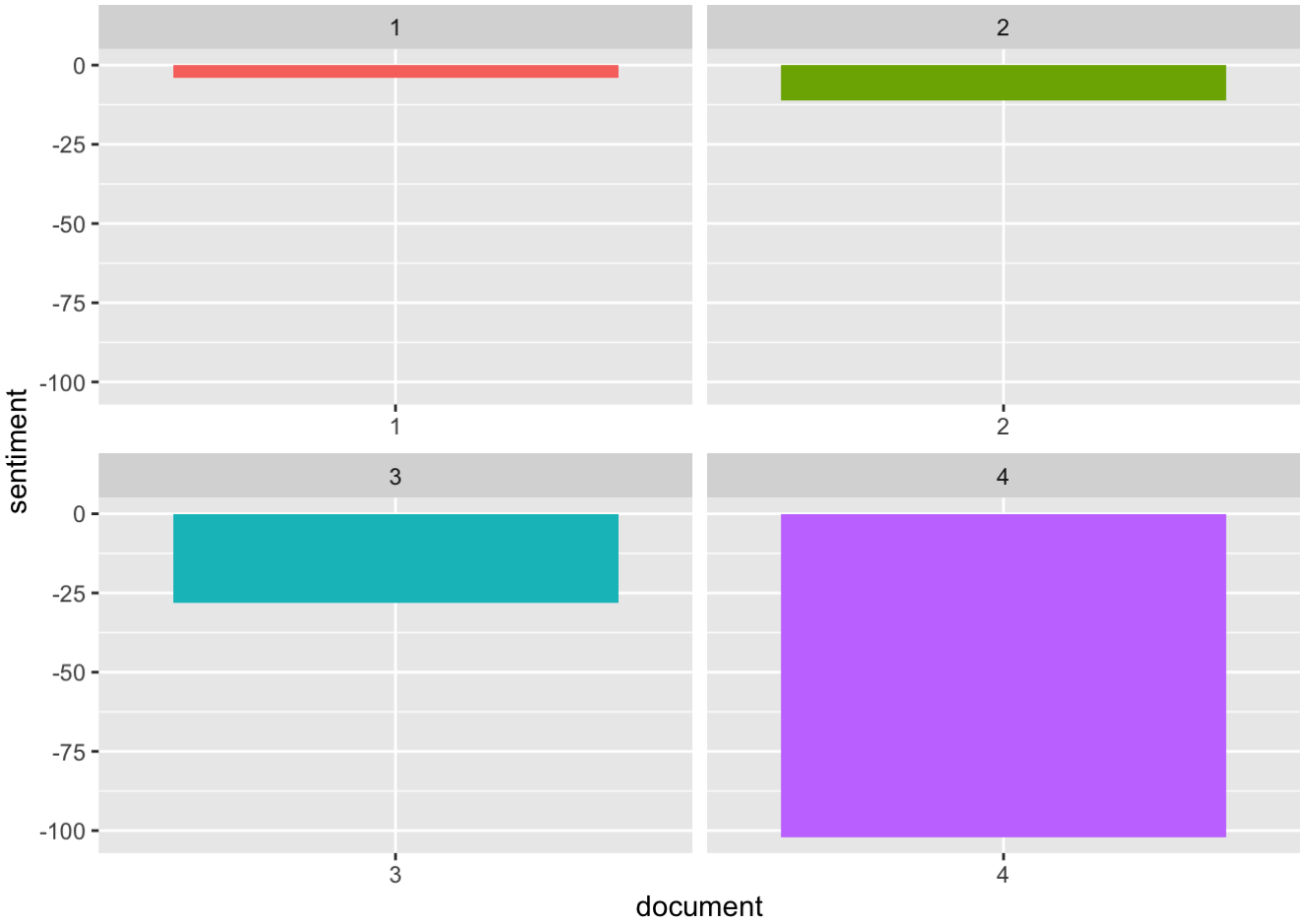
Note count(document, sentiment) asks the program to count sentiment words by document and not in the whole corpus. This is useful for comparing one document to the next.

Then the overall sentiment of the document is counted (positive vs. negative words)

Now we can compare the overall sentiments between documents data in the , just as an example of what you can do with sentiment analysis. This can be done via ggplot2.

```
library(ggplot2)

p2 <- ggplot(multi_doc_sentiment, aes(document, sentiment, fill = document)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~document, ncol = 2, scales = "free_x")
```

```
p2
```



As shown in these graphs, document 4 is significantly more negative in sentiment than the other three documents.

This is not altogether surprising as the first 3 documents are judicial opinions whereas the 4th document is a Security council statement on the situation in the Yugoslav republic.

Other packages and options

Text analysis has gained significant steam and multiple packages have been developed to analyze sentiment and other features of text. Some of the interesting and popular packages out there include: Quanteda and many others. Some, including quanteda are discussed in the book we based this tutorial largely on **Text Mining with R**, some are genenal purpose others satisfy a specific requirement. We encourage you to read more if you are interested in exploring this further.

Furthermore, when you develop your own research you may want to develop your own dictionary for text depending on what you are looking for. For example, the AMAR project (see ilcss) developed dictionaries to detect ethnic minority groups and a variety of activity they engage in in news paper text. We will learn more about this March 7th.