Words, text mining, corpus linguistics

Steven Moran

23 February, 2023

Contents

Setup

We use R [@R] and the following R packages [@tidyverse;@tidytext;@wordcloud].

```
library(tidyverse)
library(tidytext)
library(gutenbergr)
library(wordcloud)
```

Data

We can access books from Gutenberg project via the R gutenbergr package. Who was Gutenberg?

What can the `gutenbergr` package do?

```
?gutenbergr
```

When we look at the help, there are some functions native to this package for browsing the collection of books.

```
gutenberg_metadata
```

```
## # A tibble: 69,199 x 8
##    gutenberg_id title              author guten~1 langu~2 guten~3 rights has_t~4
##           <int> <chr>              <chr>    <int> <chr>   <chr>   <chr>  <lgl>
##  1            1 "The Declaration ~ Jeffe~    1638 en      Politi~ Publi~ TRUE
##  2            2 "The United State~ Unite~       1 en      Politi~ Publi~ TRUE
##  3            3 "John F. Kennedy'~ Kenne~    1666 en      <NA>    Publi~ TRUE
```

```
##  4           4 "Lincoln's Gettys~ Linco~       3 en      US Civ~ Publi~ TRUE
##  5           5 "The United State~ Unite~       1 en      United~ Publi~ TRUE
##  6           6 "Give Me Liberty ~ Henry~       4 en      Americ~ Publi~ TRUE
##  7           7 "The Mayflower Co~ <NA>        NA en      <NA>    Publi~ TRUE
##  8           8 "Abraham Lincoln'~ Linco~       3 en      US Civ~ Publi~ TRUE
##  9           9 "Abraham Lincoln'~ Linco~       3 en      US Civ~ Publi~ TRUE
## 10          10 "The King James V~ <NA>        NA en      Banned~ Publi~ TRUE
## # ... with 69,189 more rows, and abbreviated variable names
## #   1: gutenberg_author_id, 2: language, 3: gutenberg_bookshelf, 4: has_text
```

```
gutenberg_authors
```

```
## # A tibble: 21,323 x 7
##    gutenberg_author_id author              alias birth~1 death~2 wikip~3 aliases
##                  <int> <chr>               <chr>   <int>   <int> <chr>   <chr>
##  1                   1 United States       U.S.~      NA      NA https:~ <NA>
##  2                   3 Lincoln, Abraham    <NA>     1809    1865 https:~ United~
##  3                   4 Henry, Patrick      <NA>     1736    1799 https:~ <NA>
##  4                   5 Adam, Paul          <NA>     1849    1931 https:~ <NA>
##  5                   7 Carroll, Lewis      Dodg~    1832    1898 https:~ <NA>
##  6                   8 United States. Cen~ <NA>       NA      NA https:~ Agency~
##  7                   9 Melville, Herman    Melv~    1819    1891 https:~ <NA>
##  8                  10 Barrie, J. M. (Jam~ <NA>     1860    1937 https:~ Barrie~
##  9                  12 Smith, Joseph, Jr.  Smit~    1805    1844 https:~ <NA>
## 10                  14 Madison, James      Unit~    1751    1836 https:~ <NA>
## # ... with 21,313 more rows, and abbreviated variable names 1: birthdate,
## #   2: deathdate, 3: wikipedia
```

Let's grab Moby Dick by Herman Melville.

On the web, it looks like this:

- https://www.gutenberg.org/files/2701/2701-h/2701-h.htm

Let's download it and have a look. What has the `gutenberg_download()` done?

```
moby_dick <- gutenberg_download(2701)
```

```
## Determining mirror for Project Gutenberg from https://www.gutenberg.org/robot/harvest
```

```
## Using mirror http://aleph.gutenberg.org
```

```
moby_dick
```

```
## # A tibble: 21,932 x 2
##    gutenberg_id text
##           <int> <chr>
##  1         2701 "MOBY-DICK;"
##  2         2701 ""
##  3         2701 "or, THE WHALE."
##  4         2701 ""
##  5         2701 "By Herman Melville"
##  6         2701 ""
##  7         2701 ""
##  8         2701 ""
##  9         2701 "CONTENTS"
## 10         2701 ""
## # ... with 21,922 more rows
```

Now, recall we want to work with tidy data typically. What is so-called tidy data?

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

How can we do that?

One way is that we can use other people's work, i.e., another library that someone has created, to process the data for us. Pro-tip: when you need something done, go look for it on the web before you build it from scratch!

We can use the tidytext package for that.

```
help(tidytext)
```

It has an `unnest_tokens()` function.

```
?unnest_tokens
```

```
tidy_moby_dick <- moby_dick %>%
  unnest_tokens(word, text)
```

Now what does our data structure look like?

```
tidy_moby_dick
```

```
## # A tibble: 216,928 x 2
##    gutenberg_id word
##           <int> <chr>
##  1         2701 moby
##  2         2701 dick
##  3         2701 or
##  4         2701 the
##  5         2701 whale
##  6         2701 by
##  7         2701 herman
##  8         2701 melville
##  9         2701 contents
## 10         2701 etymology
## # ... with 216,918 more rows
```

This took has *tokenized* the data for us, i.e., it has split the raw data (sentences in parapgrahs in chapters in a book) into words.

Tokenization is the process of splitting text into tokens.

This one-token-per-row structure is in contrast to the ways text is often stored, perhaps as strings or in a document-term matrix.

For tidy text mining, the token that is stored in each row is most often a single word, but can also be an n-gram, sentence, or paragraph.

Now compare the first and tokenized data. What's are the differences?

```
moby_dick
```

```
## # A tibble: 21,932 x 2
##    gutenberg_id text
##           <int> <chr>
```

```
##  1         2701 "MOBY-DICK;"
##  2         2701 ""
##  3         2701 "or, THE WHALE."
##  4         2701 ""
##  5         2701 "By Herman Melville"
##  6         2701 ""
##  7         2701 ""
##  8         2701 ""
##  9         2701 "CONTENTS"
## 10         2701 ""
## # ... with 21,922 more rows
```

```
tidy_moby_dick
```

```
## # A tibble: 216,928 x 2
##    gutenberg_id word
##           <int> <chr>
##  1         2701 moby
##  2         2701 dick
##  3         2701 or
##  4         2701 the
##  5         2701 whale
##  6         2701 by
##  7         2701 herman
##  8         2701 melville
##  9         2701 contents
## 10         2701 etymology
## # ... with 216,918 more rows
```

What happened in the `unnest_tokens()` function?

```
moby_dick
```

```
## # A tibble: 21,932 x 2
##    gutenberg_id text
##           <int> <chr>
##  1         2701 "MOBY-DICK;"
##  2         2701 ""
##  3         2701 "or, THE WHALE."
##  4         2701 ""
##  5         2701 "By Herman Melville"
##  6         2701 ""
##  7         2701 ""
##  8         2701 ""
##  9         2701 "CONTENTS"
## 10         2701 ""
## # ... with 21,922 more rows
```

```
moby_dick %>%
  unnest_tokens(word, text)
```

```
## # A tibble: 216,928 x 2
##    gutenberg_id word
##           <int> <chr>
##  1         2701 moby
##  2         2701 dick
##  3         2701 or
```

```
##  4         2701 the
##  5         2701 whale
##  6         2701 by
##  7         2701 herman
##  8         2701 melville
##  9         2701 contents
## 10         2701 etymology
## # ... with 216,918 more rows
```

What else happened?

Hint: bag of words.

Remember, use the power of the force. Help `?` is your friend – in all its forms.

```
?unnest_tokens
help(unnest_tokens)
```

Exploration

What can we do with this "bag of words"?

One thing we can do is ask R to calculate the number words (tokens) in the novel.

```
length(tidy_moby_dick$word)
```

```
## [1] 216928
```

Another thing we can do is is ask R to calculate the number unique words (types) in the novel.

```
length(unique(tidy_moby_dick$word))
```

```
## [1] 17868
```

R's `unique()` function will examine all the values in the character vector (word column) and identify those that are the same and those that are different. By embedding the `unique()` function into the `length()` function, you calculate the number of unique word types from all the word tokens, i.e., all the unique words in Melville's Moby Dick vocabulary.

We can also ask R to `count()` the elements in the column for us.

What do we see here? What do we have to pass to the `count()` function?

```
tidy_moby_dick %>%
  count(word)
```

```
## # A tibble: 17,868 x 2
##    word            n
##    <chr>       <int>
##  1 _a              6
##  2 _advancing_     2
##  3 _ahab           4
##  4 _algerine       1
##  5 _alive_         1
##  6 _all_           1
##  7 _am_            1
##  8 _anglo          1
##  9 _apology        1
## 10 _are_           1
## # ... with 17,858 more rows
```

Most functions have parameters and we can tell them what to do with certain variables or properties. How to know what they are? Ask for help!

```
?count
```

There's a parameter in `sort()` that is by default set to `FALSE`, i.e., if we do not tell the function `count()` explicitly, `count(sort = TRUE)`, it will assume that it should NOT sort the count.

So what happens if we set `sort = TRUE`?

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

```
## # A tibble: 17,868 x 2
##    word      n
##    <chr> <int>
##  1 the   14523
##  2 of     6624
##  3 and    6447
##  4 a      4720
##  5 to     4627
##  6 in     4181
##  7 that   2974
##  8 his    2530
##  9 it     2418
## 10 i      1988
## # ... with 17,858 more rows
```

That's interesting, perhaps we want to save the results into a new data frame that we can call by a new variable name.

```
moby_dick_word_counts <- tidy_moby_dick %>%
  count(word, sort = TRUE)
moby_dick_word_counts
```

```
## # A tibble: 17,868 x 2
##    word      n
##    <chr> <int>
##  1 the   14523
##  2 of     6624
##  3 and    6447
##  4 a      4720
##  5 to     4627
##  6 in     4181
##  7 that   2974
##  8 his    2530
##  9 it     2418
## 10 i      1988
## # ... with 17,858 more rows
```

Now we have a data frame with two columns that includes and two data types:

- word
- n

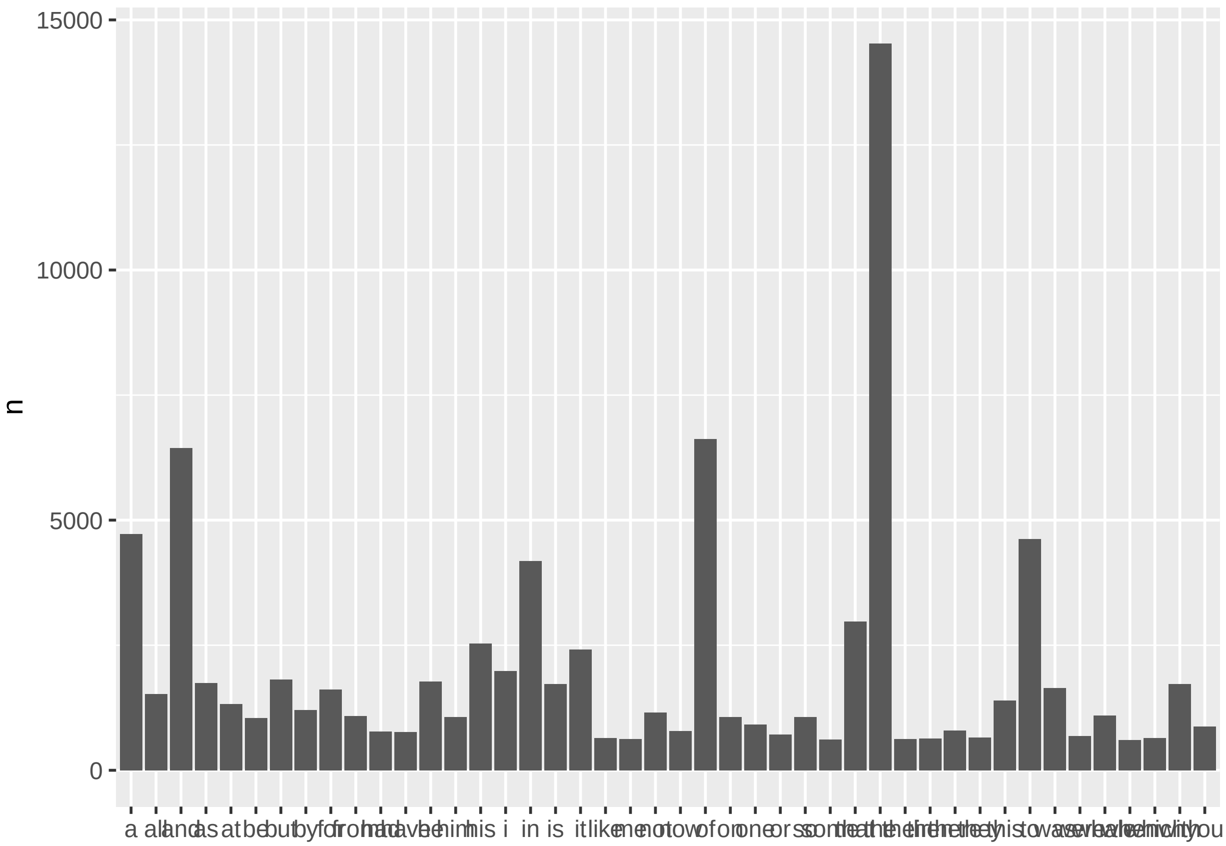
How can we visualize these data?

- https://www.data-to-viz.com

Let's consider the column `n` (a common acronym used for "count"). What kind of data type is it?

What kind of visualizations can we make with one numeric data point?

```
tidy_moby_dick %>%
  count(word, sort = TRUE) %>%
  filter(n > 600) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  xlab(NULL)
```

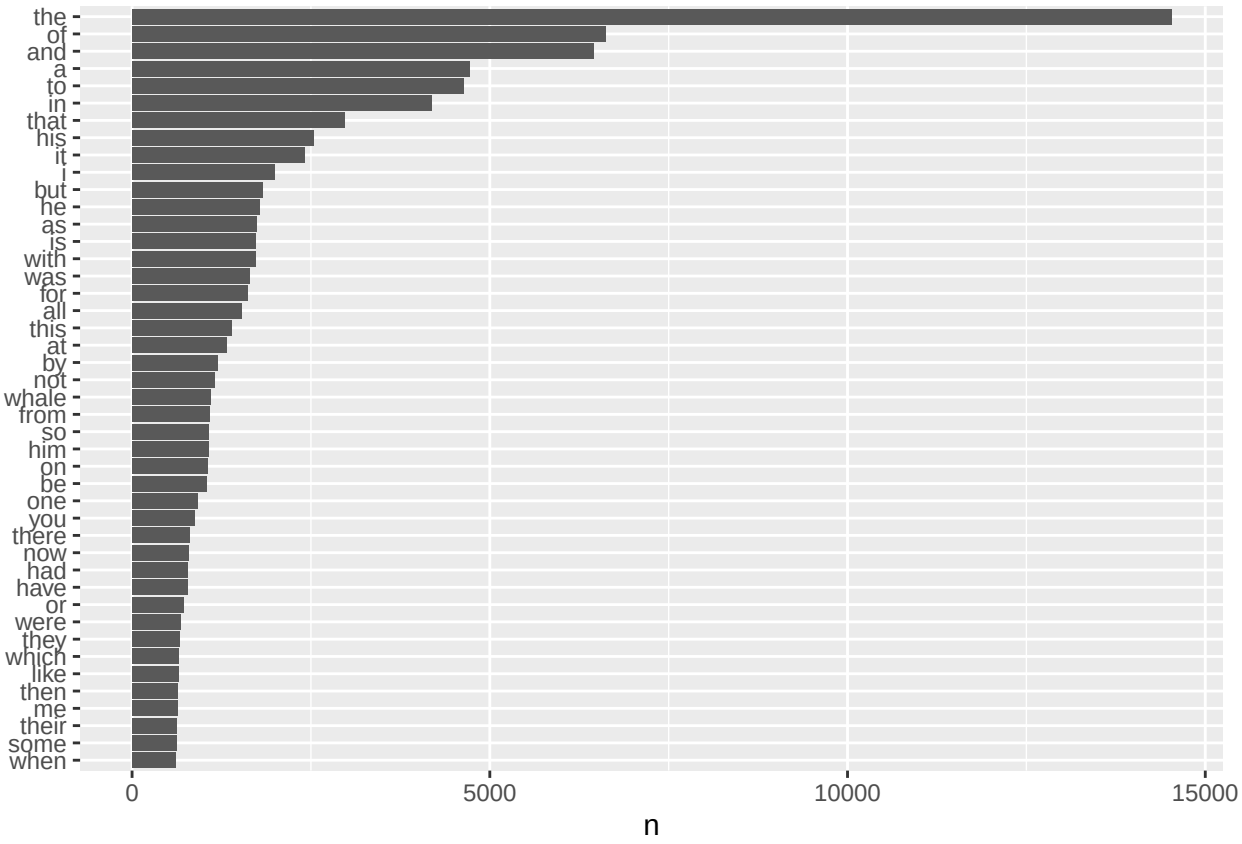


This data visualization – a bar chart aka bar plot or bar graph tells us what?

Is it useful as is?

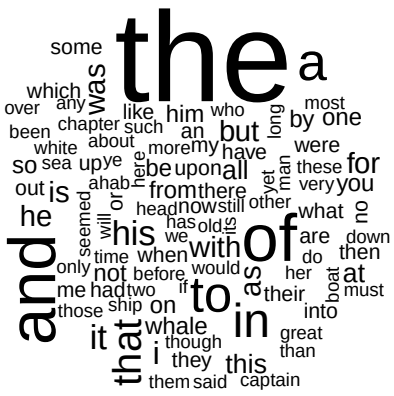
How can we display the information more meaningfully for the reader?

```
tidy_moby_dick %>%
  count(word, sort = TRUE) %>% filter(n > 600) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  xlab(NULL) +
  coord_flip()
```



What's another fun way to visualize one variable data?

```
tidy_moby_dick %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))
```



What if we're intrested in particular words?

How many occurrences of the word `whale` are in the book?

```
moby_dick_word_count <- tidy_moby_dick %>%
  count(word, sort = TRUE)

moby_dick_word_count %>% filter(word == "whale")
```

```
## # A tibble: 1 x 2
##   word      n
##   <chr> <int>
## 1 whale  1096
```

But is this a lot?

The raw count does NOT tell us much in terms of the validity of some hypothesis.

For example, is the word whale *much* more common in Moby Dick than in, say, some other works? How about Jane Austen?

We cannot say, because frequency is a *relative* judgement.

One way to put this number "of whales" into perspective is to express it as a percentage of the whole corpus, i.e., Moby Dick as a novel is over 200k words.

What percentage does the word for whale make out of the entire book, or in corpus linguistics speak, the "corpus"?

```
whale_count <- moby_dick_word_count %>% filter(word == "whale")
```

But is that a lot, relatively speaking, compared to other words?

Let's express the words as a percentage of the whole corpus in which we see the words.

How many words do we have in the whole book / corpus?

```
length(tidy_moby_dick$word)
```

```
## [1] 216928
```

Now we can use the total number of words as our denominator and we can extend our data frame by adding a new column.

```
moby_dick_word_count$frequency <- moby_dick_word_count$n / length(tidy_moby_dick$word) * 100
moby_dick_word_count
```

```
## # A tibble: 17,868 x 3
##    word      n frequency
##    <chr> <int>     <dbl>
##  1 the   14523     6.69
##  2 of     6624     3.05
##  3 and    6447     2.97
##  4 a      4720     2.18
##  5 to     4627     2.13
##  6 in     4181     1.93
##  7 that   2974     1.37
##  8 his    2530     1.17
##  9 it     2418     1.11
## 10 i      1988     0.916
## # ... with 17,858 more rows
```

Now we can again look at how frequent *within* the text a given word is.

```
moby_dick_word_count %>% filter(word == "whale")
```

```
## # A tibble: 1 x 3
##   word      n frequency
##   <chr> <int>     <dbl>
```

```
## 1 whale  1096     0.505
```

```
moby_dick_word_count %>% filter(word == "harpoon")
```

```
## # A tibble: 1 x 3
##   word        n frequency
##   <chr>   <int>     <dbl>
## 1 harpoon    76    0.0350
```

Word type-token ratio (TTR)

The type–token ratio is one of the basic corpus statistics.

Comparing the number of tokens in the text to the number of types of tokens (unique word form) can tell us how large a range of vocabulary is used in the text.

- TTR = (number of types/number of tokens), or
- TTR = (number of types/number of tokens) * 100 (as a percentage)

TTR allows us to measure vocabulary variation between corpora: the closer the result is to 1 (or 100%), the greater the vocabulary variation.

Here's our tidy data.

```
tidy_moby_dick
```

```
## # A tibble: 216,928 x 2
##    gutenberg_id word
##           <int> <chr>
##  1         2701 moby
##  2         2701 dick
##  3         2701 or
##  4         2701 the
##  5         2701 whale
##  6         2701 by
##  7         2701 herman
##  8         2701 melville
##  9         2701 contents
## 10         2701 etymology
## # ... with 216,918 more rows
```

Now let's get the TTR.

```
types <- length(unique(tidy_moby_dick$word))
tokens <- length(tidy_moby_dick$word)
ttr <- types / tokens
ttr
```

```
## [1] 0.08236834
```

So now we can compare two (or more) texts to see which text has a greater range of vocabulary. For example, consider the vocabulary of rappers:

- https://pudding.cool/projects/vocabulary/index.html

Comparing vocabulary between corpora

Now let's compare the vocabulary range of Melville with, say, Jane Austin.

This should get you started.

```
sense_sensibility <- gutenberg_download(161)
sense_sensibility
```

```
## # A tibble: 12,673 x 2
##    gutenberg_id text
##           <int> <chr>
##  1          161 "[Illustration]"
##  2          161 ""
##  3          161 ""
##  4          161 ""
##  5          161 ""
##  6          161 "Sense and Sensibility"
##  7          161 ""
##  8          161 "by Jane Austen"
##  9          161 ""
## 10          161 "(1811)"
## # ... with 12,663 more rows
```

Which text has the greater range in vocabulary in terms of its type-to-token ratio?

But note we can also drill down into a specific words and compare them.

Is Moby Dick all about men?

Let's compare the (relative) frequencies of pronouns between texts for the pronouns "he" and "she".

Start first by calculating their normalized frequencies. Choose an appropriate normalization base.

Next, filter for all relevant pronouns (e.g. with the notation %in% c(...) or other filter options). Hint:

```
tidy_moby_dick %>% filter(word %in% c('he', 'his'))
```

```
## # A tibble: 4,306 x 2
##    gutenberg_id word
##           <int> <chr>
##  1         2701 his
##  2         2701 he
##  3         2701 his
##  4         2701 he
##  5         2701 his
##  6         2701 he
##  7         2701 his
##  8         2701 his
##  9         2701 he
## 10         2701 his
## # ... with 4,296 more rows
```

Compare the pronouns' relative frequencies between texts.

Is there a discrepancy between Moby Dick and other texts?

For comparison, load another text, pre-process it, and calculate the normalized frequency of the two sets of pronouns.

Ngrams

The `unnest_tokens()` function also takes other types tokenizations.

```
moby_dick_bigrams <- moby_dick %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)
moby_dick_bigrams
```

```
## # A tibble: 201,317 x 2
##    gutenberg_id bigram
##           <int> <chr>
##  1         2701 moby dick
##  2         2701 <NA>
##  3         2701 or the
##  4         2701 the whale
##  5         2701 <NA>
##  6         2701 by herman
##  7         2701 herman melville
##  8         2701 <NA>
##  9         2701 <NA>
## 10         2701 <NA>
## # ... with 201,307 more rows
```

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

```
## # A tibble: 107,359 x 2
##    bigram        n
##    <chr>     <int>
##  1 <NA>       3300
##  2 of the     1778
##  3 in the     1115
##  4 to the      701
##  5 from the    410
##  6 of his      355
##  7 and the     351
##  8 on the      340
##  9 the whale   329
## 10 of a        322
## # ... with 107,349 more rows
```

```
moby_dick_bigrams <- moby_dick_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")

moby_dick_bigrams_counts <- moby_dick_bigrams %>% count(word1, word2, sort = TRUE)

moby_dick_bigrams_counts
```

```
## # A tibble: 107,359 x 3
##    word1 word2     n
##    <chr> <chr> <int>
##  1 <NA>  <NA>   3300
##  2 of    the    1778
##  3 in    the    1115
##  4 to    the     701
##  5 from  the     410
##  6 of    his     355
```

```
##  7 and   the     351
##  8 on    the     340
##  9 the   whale   329
## 10 of    a       322
## # ... with 107,349 more rows
```