Lesson 15 - Data Analytics - Tidy Text

Dr. Jeffrey Strickland

9/15/2018

Load necessary libraries

```r
    if(!require(tm)) install.packages("tm")

    if(!require(tidytext)) install.packages("tidytext")

    if(!require(tidyverse)) install.packages("tidyverse")

    if(!require(stringr)) install.packages("stringr")

    if(!require(dplyr)) install.packages("dplyr")

    if(!require(tidyr)) install.packages("tidyr")

    if(!require(janeaustenr)) install.packages("janeaustenr")

    if(!require(purrr)) install.packages("purrr")

    if(!require(sentimentr)) install.packages("sentimentr")

    if(!require(readr)) install.packages("readr")

    if(!require(wordcloud)) install.packages("wordcloud")

    if(!require(lubridate)) install.packages("lubridate")

    if(!require(ggplot2)) install.packages("ggplot2")

    if(!require(ggraph)) install.packages("ggraph")

    if(!require(igraph)) install.packages("igraph")

    if(!require(plotrix)) install.packages("plotrix")
```

Tidy Text

As already covered, tidy text is a format useful for several typse of text mining problems, including sentiment and topic analysis. Soem advantages of the tidy text format are: - keeps one token (typically a word) in each row - keeps each variable (such as a document or chapter) in a column. - when your data is tidy, you can use a common set of tools for exploring and visualizing them

This frees you from struggling to get your data into the right format for each task and lets you focus on the questions you want to ask.

Step 1: put the text into a data frame

- the `c()` function returns a vector (a one dimensional array)
- `Paste0()` concatenates strings without spaces
- the `data_frame()` function is used for storing data tables
- The `map()` function transforms the text by applying a function to each element and returning a vector the same length
- `read_lines()` reads up to `n_max` lines from a file. … `read_lines_raw()` produces a list of raw vectors
- the Mutate function adds new variables and preserves existing

```
quantum_words<-
 data_frame(file =
   
paste0("C:\\Users\\jeff\\Documents\\VIT_Course_Material\\Data_Analytics_2018\
\data\\",
   c("quantum_phaith.txt",
     "quantum_hope.txt",
     "quantum_love.txt"))) %>%
   mutate(text = map(file, read_lines))
```

Step 2: Unnest the tibble

Tibbles are new data frames that keep the features we like and drops the features that are now frustrating (i.e. converting character vectors to factors). - unnest() the tibble - remove the lines that are LaTeX crude - compute a line number with the mutate function

```
quantum_words <- quantum_words %>%
   unnest() %>%
   mutate(line_number = 1:n(),
      file =
      str_sub(basename(file), 1, -5))
```

Step 3: Delete words and LaTex

- `str_detect()` detects the presence or absence of a pattern and returns a logical vector
- ! is the logical operator for negation

Quantum Words

```
quantum_words <- quantum_words %>%
  unnest() %>%
  filter(text != "%!TEX root = ind.tex") %>%
  filter(!str_detect(text, "^(\\\\[A-Z,a-z])"), text != "")
```

Quantum Tokens

```
quantum_words <- quantum_words %>%
  unnest_tokens(word, text) %>%
  filter(!str_detect(word, "[0-9]"),
         word != "fismanreview",
         word != "multicolumn",
         word != "p",
         word != "_i",
         word != "al",
         word != "tabular",
         word != "ref",
         word != "cite",
         !str_detect(word, "[a-z]_"),
         !str_detect(word, ":"),
         word != "bar",
         word != "emph",
         !str_detect(word, "textless"))
quantum_words
```

```
## # A tibble: 158,482 x 3
##    file           line_number word
##    <chr>                <int> <chr>
##  1 quantum_phaith           1 quantum
##  2 quantum_phaith           1 phaith
##  3 quantum_phaith           2 preface
##  4 quantum_phaith           3 obviously
##  5 quantum_phaith           3 either
##  6 quantum_phaith           3 i
##  7 quantum_phaith           3 do
##  8 quantum_phaith           3 not
##  9 quantum_phaith           3 know
## 10 quantum_phaith           3 how
## # ... with 158,472 more rows
```

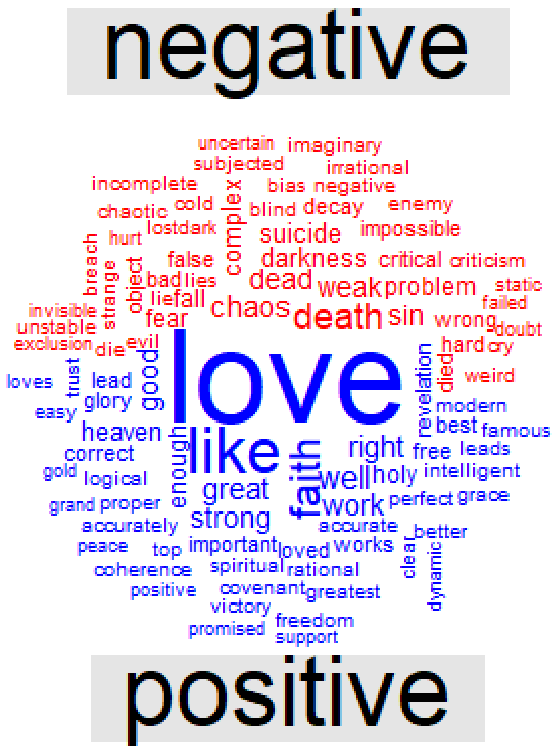
Quantum Cloud

```
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
quantum_words %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("red", "blue"),
                   max.words = 100)
```

```
## Joining, by = "word"
```



Lexicon Exploration

The tidytext package contains several sentiment lexicons in the sentiments dataset. The three general-purpose lexicons are:

- AFINN from Finn Årup Nielsen
- bing from Bing Liu and collaborators
- nrc from Saif Mohammad and Peter Turney

All three of these lexicons are based on unigrams, i.e., single words.

```
library(tidytext)
sentiments

## # A tibble: 27,314 x 4
##    word         sentiment lexicon score
##    <chr>        <chr>     <chr>   <int>
##  1 abacus       trust     nrc        NA
##  2 abandon      fear      nrc        NA
##  3 abandon      negative  nrc        NA
##  4 abandon      sadness   nrc        NA
##  5 abandoned    anger     nrc        NA
##  6 abandoned    fear      nrc        NA
##  7 abandoned    negative  nrc        NA
##  8 abandoned    sadness   nrc        NA
##  9 abandonment anger     nrc         NA
## 10 abandonment fear      nrc         NA
## # ... with 27,304 more rows
```

NRC Lexicon

These lexicons contain many English words and the words are assigned scores for positive/negative sentiment, and also possibly emotions like joy, anger, sadness, and so forth. The nrc lexicon: - Includes 13,901 words - Categorizes words in a binary fashion (“yes”/“no”) into categories of: - Positive - Negative - Anger - Fear - Joy - Sadness - Surprise - Trust

```
get_sentiments("nrc")

## # A tibble: 13,901 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 abacus      trust
##  2 abandon     fear
##  3 abandon     negative
##  4 abandon     sadness
##  5 abandoned   anger
##  6 abandoned   fear
##  7 abandoned   negative
##  8 abandoned   sadness
##  9 abandonment anger
## 10 abandonment fear
## # ... with 13,891 more rows
```

AFINN Lexicon

- Includes 2,476 words
- The AFINN lexicon assigns words with a score that runs between -5 and 5 with
  - negative scores indicating negative sentiment
  - positive scores indicating positive sentiment.

```
get_sentiments("afinn")

## # A tibble: 2,476 x 2
##    word       score
##    <chr>      <int>
##  1 abandon       -2
##  2 abandoned     -2
##  3 abandons      -2
##  4 abducted      -2
##  5 abduction     -2
##  6 abductions    -2
##  7 abhor         -3
##  8 abhorred      -3
##  9 abhorrent     -3
## 10 abhors        -3
## # ... with 2,466 more rows
```

Bing Lexicon

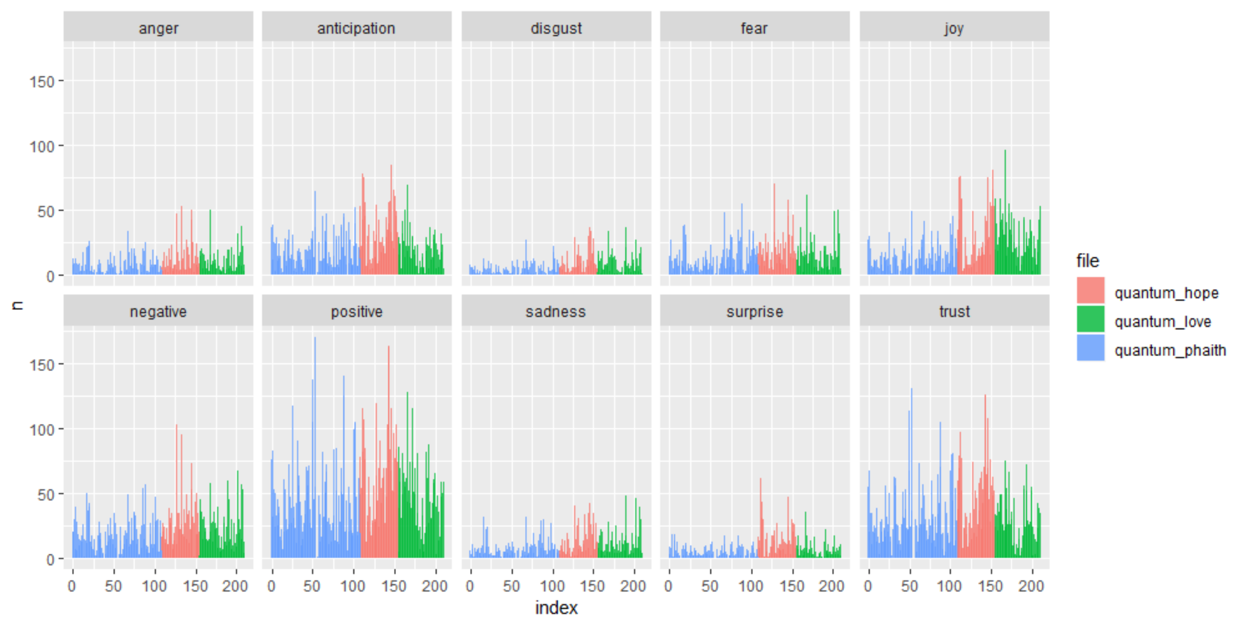
- Includes 13,901 words
- The nrc lexicon categorizes words in a binary fashion (“yes”/“no”) into categories of:
  - Positive
  - Negative
  - Anger
  - Fear
  - Joy
  - Sadness
  - Surprise
  - Trust

```
get_sentiments("bing")

## # A tibble: 6,788 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 2-faced     negative
##  2 2-faces     negative
##  3 a+          positive
##  4 abnormal    negative
##  5 abolish     negative
##  6 abominable  negative
##  7 abominably  negative
##  8 abominate   negative
##  9 abomination negative
## 10 abort       negative
## # ... with 6,778 more rows
```

Getting Sentiments with NRC

Using the nrc lexicon, let’s see how the emotions of my words change between Quantum Phaith and Quantum Love.

```
quantum_words %>%
  inner_join(get_sentiments("nrc")) %>%
  group_by(index = line_number %/% 25, file, sentiment) %>%
  summarize(n = n()) %>%
  ggplot(aes(x = index, y = n, fill = file)) +
  geom_bar(stat = "identity", alpha = 0.8) +
  facet_wrap(~ sentiment, ncol = 5)

## Joining, by = "word"
```



Based on the plot, it looks like Quantum Love is mode "sentimental, as the sentiment scores are a little higher than for Quantum Phaith. Not that Quantum Phaith is much more scientific.

Tidy packages

tidyverse and tidytext work well together, so I loaded both. The stringr package is useful for filtering out the LaTeX specific code and also for dropping words that have numbers in them (like jefferson1776 as a reference or 0.05).

Put/convert ind texts to a data frame

```
ind_words <- tibble(file = paste0("~/VIT_University/tidy_text/",
    c("Indian_Philosophy_Part_I.txt", "Indian_Philosophy_Part_II.txt"))) %>%
    mutate(text = map(file, read_lines))
ind_words

## # A tibble: 2 x 2
##   file                                                 text
##   <chr>                                                <list>
## 1 ~/VIT_University/tidy_text/Indian_Philosophy_Part_I.txt  <chr [13,715]>
## 2 ~/VIT_University/tidy_text/Indian_Philosophy_Part_II.txt <chr [19,984]>
```

Unnest the Tibbles

The resulting tibble has a variable file that is the name of the file that created that row and a list-column of the text of that file. We want to unnest() that tibble, remove the lines that are LaTeX crude and compute a line number.

```
ind_words <- ind_words %>%
  unnest() %>%
  filter(text != "%!TEX root = ind.tex") %>%
  filter(!str_detect(text, "^(\\\\[A-Z,a-z])"),
         text != "") %>%
  mutate(line_number = 1:n(),
         file = str_sub(basename(file), 1, -5))
```

Now we have a tibble with file giving us the chapter, text giving us the line of text from the text files and line_number giving a counter of the number of lines since the start of the ind texts.

Tokenize the Text

Now we want to tokenize (strip each word of any formatting and reduce down to the root word, if possible). This is easy with unnest_tokens(). I played around with the results and came up with some other words that needed to be deleted (stats terms like ci or p, LaTeX terms like _i or tabular and references/numbers.

```
ind_words <- ind_words %>%
  unnest_tokens(word, text) %>%
  filter(!str_detect(word, "[0-9]"),
         word != "fismanreview",
         word != "multicolumn",
         word != "p",
         word != "_i",
         word != "ci",
         word != "al",
         word != "tabular",
         word != "t",
         word != "ref",
         word != "cite",
         !str_detect(word, "[a-z]_"),
         !str_detect(word, ":"),
         word != "bar",
         word != "emph",
         !str_detect(word, "textless"))
ind_words
```

```
## # A tibble: 250,178 x 3
##    file                         line_number word
##    <chr>                              <int> <chr>
##  1 Indian_Philosophy_Part_I               1 indian
##  2 Indian_Philosophy_Part_I               1 philosophy
##  3 Indian_Philosophy_Part_I               1 part
##  4 Indian_Philosophy_Part_I               1 i
##  5 Indian_Philosophy_Part_I               2 chapter
##  6 Indian_Philosophy_Part_I               2 i
##  7 Indian_Philosophy_Part_I               3 introduction
##  8 Indian_Philosophy_Part_I               4 general
##  9 Indian_Philosophy_Part_I               4 characteristics
## 10 Indian_Philosophy_Part_I          4 of
## # ... with 250,168 more rows
```

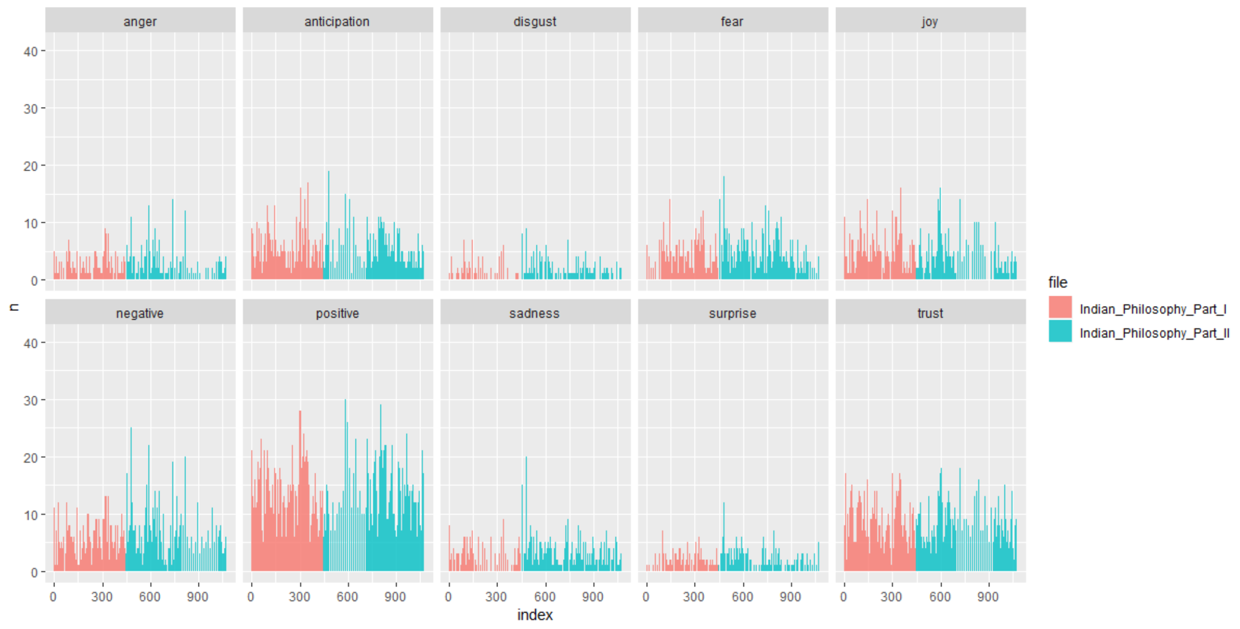
Sentiment Computation

Now to compute the sentiment using the words written per line in the ind texts. tidytext comes with three sentiment lexicons, affin, bing and nrc. affin provides a score ranging from -5 (very negative) to +5 (very positive) for 2,476 words. bing provides a label of “negative” or “positive” for 6,788 words. nrc provides a label (anger, anticipation, disgust, fear, joy, negative, positive, sadness, surprise or trust) for 13,901 words. None of these account for negation (“I’m not sad” is a negative sentiment, not a positive one).

Getting Sentiments with NRC

Using the nrc lexicon, let’s see how the emotions of my words change over the three ind texts.

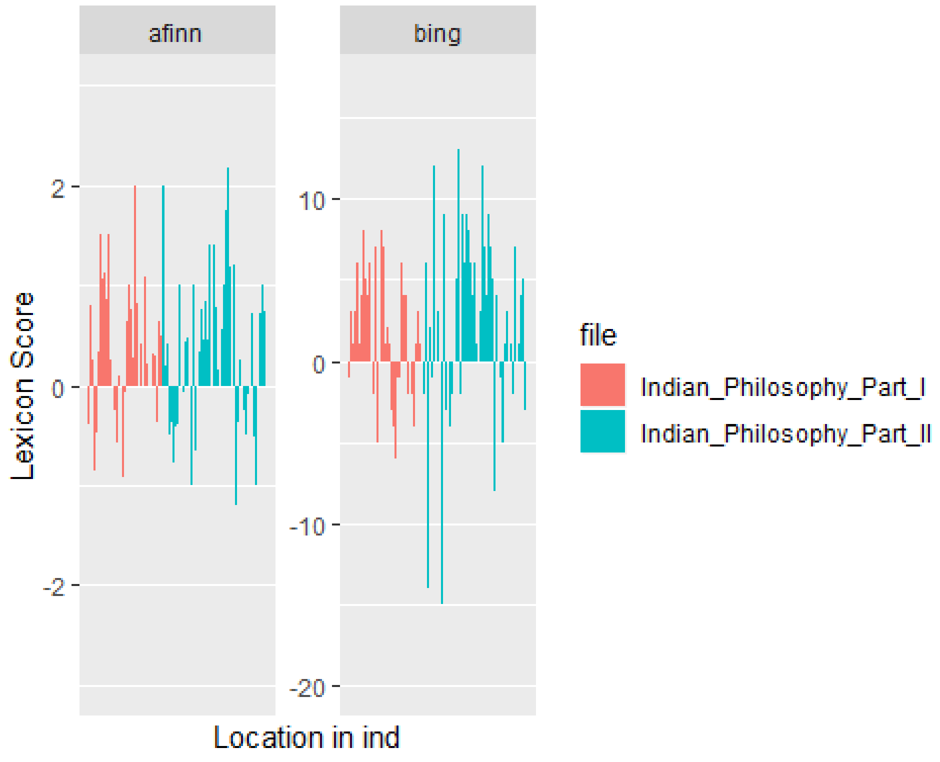
```
ind_words %>%
  inner_join(get_sentiments("nrc")) %>%
  group_by(index = line_number %/% 25, file, sentiment) %>%
  summarize(n = n()) %>%
  ggplot(aes(x = index, y = n, fill = file)) +
  geom_bar(stat = "identity", alpha = 0.8) +
  facet_wrap(~ sentiment, ncol = 5)

## Joining, by = "word"
```



Getting Sentiments with Bing

All three texts are more positive than negative, and all three representedd trust fairly well. It looks like “disgust” and “sadness” are minimized. We can use the bing and afinn lexicons to look at how the sentiment of the words changed over the course of the thesis.

```
ind_words %>%
  left_join(get_sentiments("bing")) %>%
  left_join(get_sentiments("afinn")) %>%
  group_by(index = line_number %/% 25, file) %>%
  summarize(afinn = mean(score, na.rm = TRUE),
            bing = sum(sentiment == "positive", na.rm = TRUE) - sum(sentiment
== "negative", na.rm = TRUE)) %>%
  gather(lexicon, lexicon_score, afinn, bing) %>%
  ggplot(aes(x = index, y = lexicon_score, fill = file)) +
    geom_bar(stat = "identity") +
    facet_wrap(~ lexicon, scale = "free_y") +
    scale_x_continuous("Location in ind", breaks = NULL) +
    scale_y_continuous("Lexicon Score")

## Joining, by = "word"
## Joining, by = "word"

## Warning: Removed 2 rows containing missing values (position_stack).
```



```
ind_words
```

```
## # A tibble: 250,178 x 3
##    file                         line_number word
##    <chr>                              <int> <chr>
##  1 Indian_Philosophy_Part_I               1 indian
##  2 Indian_Philosophy_Part_I               1 philosophy
##  3 Indian_Philosophy_Part_I               1 part
##  4 Indian_Philosophy_Part_I               1 i
##  5 Indian_Philosophy_Part_I               2 chapter
##  6 Indian_Philosophy_Part_I               2 i
##  7 Indian_Philosophy_Part_I               3 introduction
##  8 Indian_Philosophy_Part_I               4 general
##  9 Indian_Philosophy_Part_I               4 characteristics
## 10 Indian_Philosophy_Part_I               4 of
## # ... with 250,168 more rows
```

Lexicon Comparison

Looking at the two lexicon's scoring of my books, the affin lexicon seems a little more stable if we assume local correlation of sentiments is likely. The scores show that all three text are much more positive than negative.

Scoring the Text

Filter for negative words

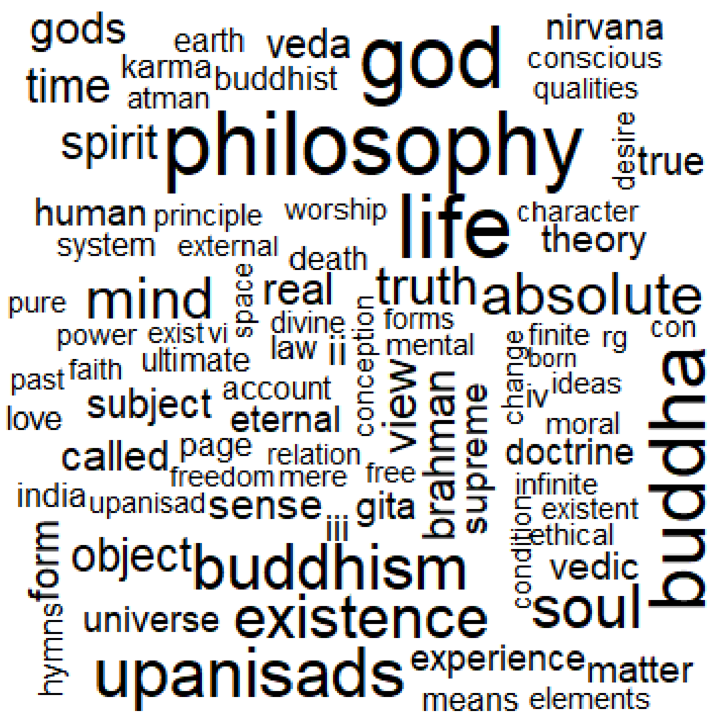
```
bingnegative <- get_sentiments("bing") %>%
    filter(sentiment == "negative")
```

Get a word count

```
wordcounts <- ind_words %>%
    group_by(index = line_number %/% 25, file) %>%
    summarize(words = n())
wordcounts

## # A tibble: 1,081 x 3
## # Groups:   index [?]
##    index file                      words
##    <dbl> <chr>                     <int>
##  1     0 Indian_Philosophy_Part_I    187
##  2     1 Indian_Philosophy_Part_I    240
##  3     2 Indian_Philosophy_Part_I    251
##  4     3 Indian_Philosophy_Part_I    227
##  5     4 Indian_Philosophy_Part_I    215
##  6     5 Indian_Philosophy_Part_I    240
##  7     6 Indian_Philosophy_Part_I    241
##  8     7 Indian_Philosophy_Part_I    239
##  9     8 Indian_Philosophy_Part_I    228
## 10     9 Indian_Philosophy_Part_I    221
## # ... with 1,071 more rows
```

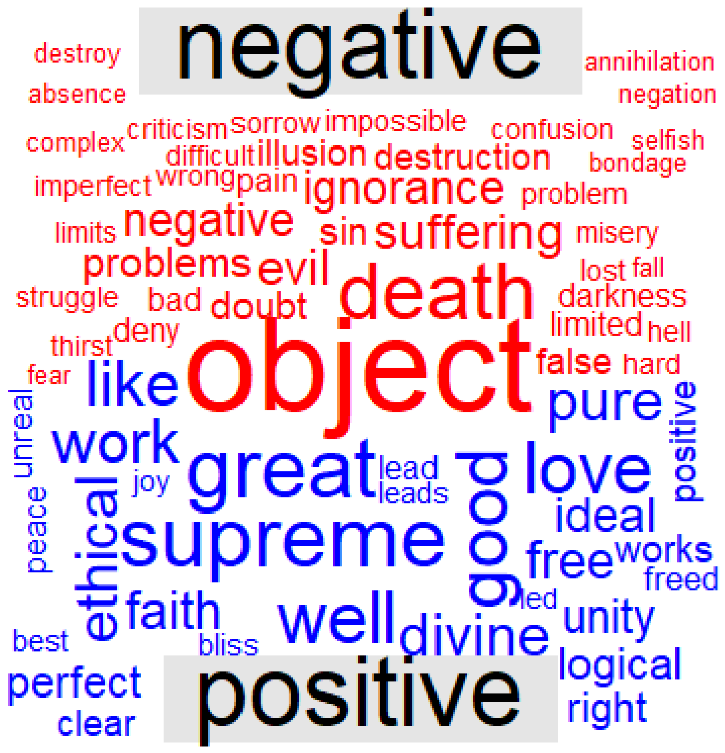
Build a cloud chart

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



Build a contrasting cloud chart.

```
library(reshape2)
ind_words %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("red", "blue"),
                   max.words = 100)

## Joining, by = "word"
```



Document Comparison

```
readings <-
read.csv("C:\\Users\\jeff\\Documents\\VIT_Course_Material\\Data_Analytics_201
8\\data\\readings.csv")
head(readings,5)
```

```
##                  book
## 1 Indian Philosophy
## 2 Indian Philosophy
## 3 Indian Philosophy
## 4 Indian Philosophy
## 5 Indian Philosophy
##                                                                    text
## 1                                                     INDIAN PHILOSOPHY
## 2                                                       By Radhakrishnan
## 3                                          MUIRHEAD LIBRARY OF PHILOSOPHY
## 4   An admirable statement of the aims of the Library of Philosophy was
## 5 provided by the first editor, the late Professor H J Muirhead, in his
```

Analyzing word and document frequency: tf-idf

A central question in text mining and natural language processing is how to quantify what a document is about. Can we do this by looking at the words that make up the document? One measure of how important a word may be is its term frequency (tf), how frequently a word occurs in a document, as we examined in Chapter 1. There are words in a document, however, that occur many times but may not be important; in English, these are probably words like “the”, “is”, “of”, and so forth. We might take the approach of adding words like these to a list of stop words and removing them before analysis, but it is possible that some of these words might be more important in some documents than others. A list of stop words is not a very sophisticated approach to adjusting term frequency for commonly used words.

Another approach is to look at a term’s inverse document frequency (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. This can be combined with term frequency to calculate a term’s tf-idf (the two quantities multiplied together), the frequency of a term adjusted for how rarely it is used.

The statistic tf-idf is intended to measure how important a word is to a document in a collection (or corpus) of documents, for example, to one novel in a collection of novels or to one website in a collection of websites.

Term frequency

Let’s start by looking at a corpus of foue books and examine first term frequency, then tf-idf. We can start just by using dplyr verbs such as group_by() and join(). What are the most commonly used words in our book corpus? (Let’s also calculate the total words in each novel here, for later use.)

First, we load and tokenize the book corpus called :readings," as well as remove stopwords.

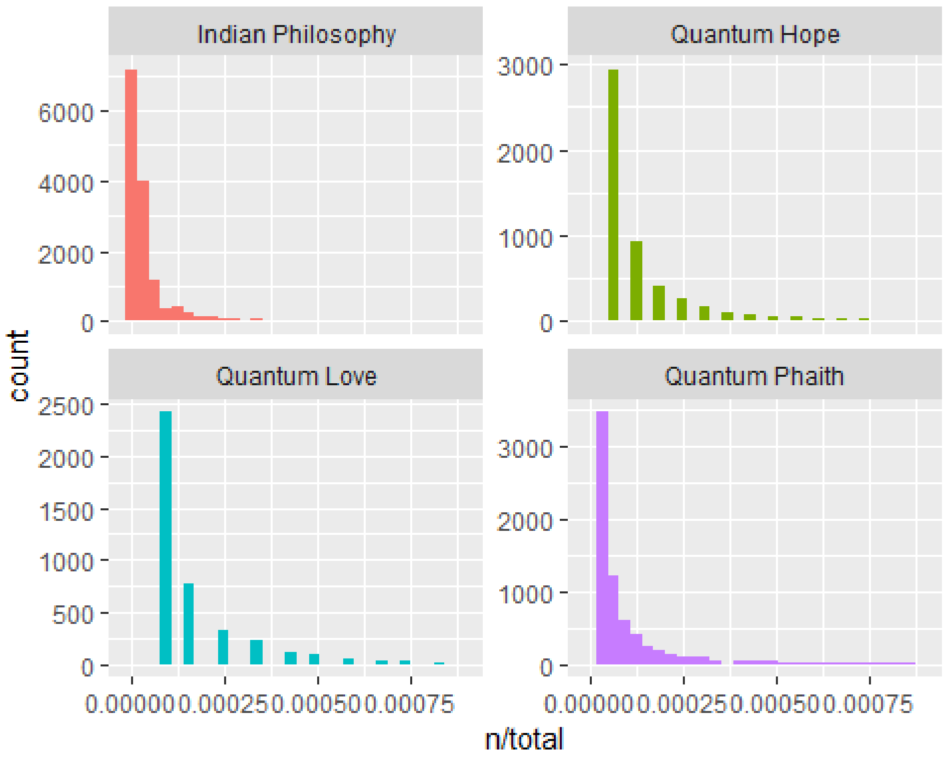
```
readings <-
read.csv("C:\\Users\\jeff\\Documents\\VIT_Course_Material\\Data_Analytics_201
8\\data\\readings.csv",stringsAsFactor=FALSE)

readings<-readings %>% group_by(book) %>% mutate(ln=row_number())%>%
unnest_tokens(word,text) %>% count(book, word, sort = TRUE) %>%ungroup()

my_stops <- c("https", "http", stopwords("en"))
readings <- readings %>%
  filter(!word %in% stop_words$word,
         !word %in% my_stops,
         !word %in% str_remove_all(stop_words$word, "'"))
head(readings,5)

## # A tibble: 5 x 3
##   book               word       n
##   <chr>              <chr>  <int>
## 1 Indian Philosophy world   1073
## 2 Indian Philosophy life     693
## 3 Indian Philosophy god      660
## 4 Quantum Love      love     604
## 5 Indian Philosophy buddha   566
```

Get Term Frequencies

```
total_words <- readings %>%
  group_by(book) %>%
  summarize(total = sum(n))

book_words <- left_join(readings, total_words)

## Joining, by = "book"

book_words

## # A tibble: 31,228 x 4
##    book               word          n total
##    <chr>              <chr>     <int> <int>
##  1 Indian Philosophy world      1073 93145
##  2 Indian Philosophy life        693 93145
##  3 Indian Philosophy god         660 93145
##  4 Quantum Love      love        604 12077
##  5 Indian Philosophy buddha      566 93145
##  6 Indian Philosophy reality     538 93145
##  7 Indian Philosophy nature      451 93145
##  8 Indian Philosophy soul        443 93145
##  9 Indian Philosophy upanisads   439 93145
## 10 Indian Philosophy existence   437 93145
## # ... with 31,218 more rows
```

There is one row in this book_words data frame for each word-book combination; n is the number of times that word is used in that book and total is the total words in that book. The usual suspects are here with the highest n, “the”, “and”, “to”, and so forth. The plot shows the distribution of n/total for each novel, the number of times a word appears in a novel divided by the total number of terms (words) in that book. This is exactly what “term frequency” is.

```
library(ggplot2)
ggplot(book_words, aes(n/total, fill = book)) +
  geom_histogram(show.legend = FALSE) +
  xlim(NA, 0.0009) +
  facet_wrap(~book, ncol = 2, scales = "free_y")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 584 rows containing non-finite values (stat_bin).
```



Zipf’s law

Distributions like those shown in the previous plot are typical in language. In fact, those types of long-tailed distributions are so common in any given corpus of natural language (like a book, or a lot of text from a website, or spoken words) that the relationship between the frequency that a word is used and its rank has been the subject of study; a classic version of this relationship is called Zipf’s law, after George Zipf, a 20th century American linguist.

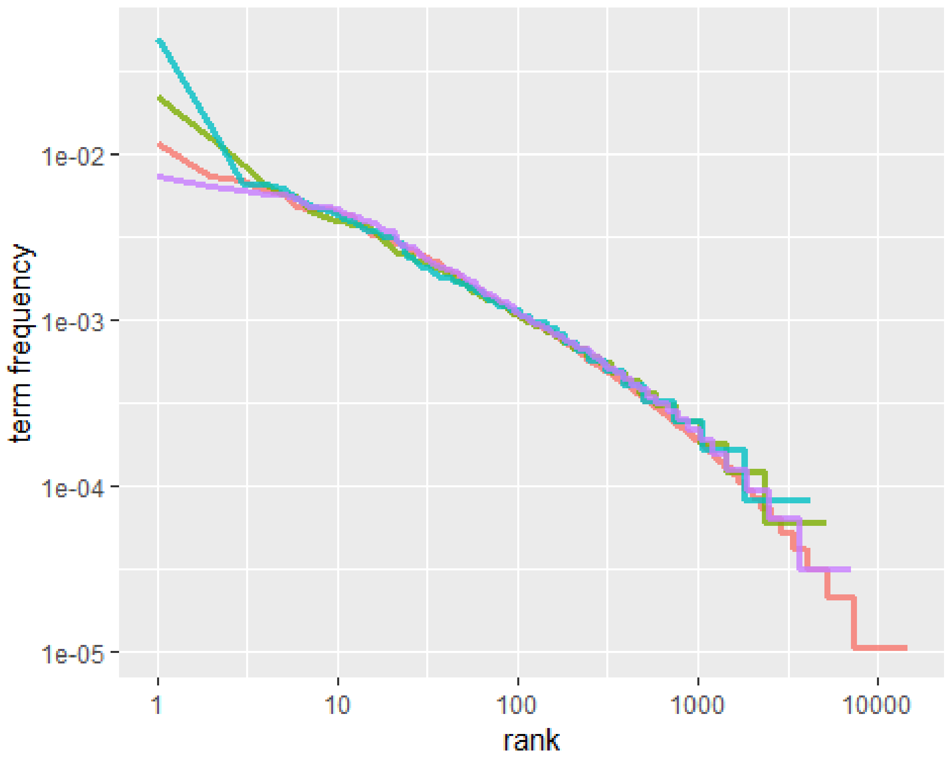
Zipf’s law states that the frequency that a word appears is inversely proportional to its rank.

Since we have the data frame we used to plot term frequency, we can examine Zipf’s law for our book collect with just a few lines of dplyr functions.

```
freq_by_rank <- book_words %>%
  group_by(book) %>%
  mutate(rank = row_number(),
         `term frequency` = n/total)
freq_by_rank

## # A tibble: 31,228 x 6
## # Groups:   book [4]
##    book              word          n total  rank `term frequency`
##    <chr>             <chr>     <int> <int> <int>            <dbl>
##  1 Indian Philosophy world      1073 93145     1           0.0115
##  2 Indian Philosophy life        693 93145     2           0.00744
##  3 Indian Philosophy god         660 93145     3           0.00709
##  4 Quantum Love      love        604 12077     1           0.0500
##  5 Indian Philosophy buddha      566 93145     4           0.00608
##  6 Indian Philosophy reality     538 93145     5           0.00578
##  7 Indian Philosophy nature      451 93145     6           0.00484
##  8 Indian Philosophy soul        443 93145     7           0.00476
##  9 Indian Philosophy upanisads   439 93145     8           0.00471
## 10 Indian Philosophy existence   437 93145     9           0.00469
## # ... with 31,218 more rows
```

The rank column here tells us the rank of each word within the frequency table; the table was already ordered by n so we could use row_number() to find the rank. Then, we can calculate the term frequency in the same way we did before. Zipf's law is often visualized by plotting rank on the x-axis and term frequency on the y-axis, on logarithmic scales. Plotting this way, an inversely proportional relationship will have a constant, negative slope.

```
freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = book)) +
  geom_line(size = 1.1, alpha = 0.8, show.legend = FALSE) +
  scale_x_log10() +
  scale_y_log10()
```



Notice that the plot is in log-log coordinates. We see that all four of our books are similar to each other, and that the relationship between rank and frequency does have negative slope.

It is not quite constant, though; perhaps we could view this as a broken power law with, say, three sections. Let's see what the exponent of the power law is for the middle section of the rank range.

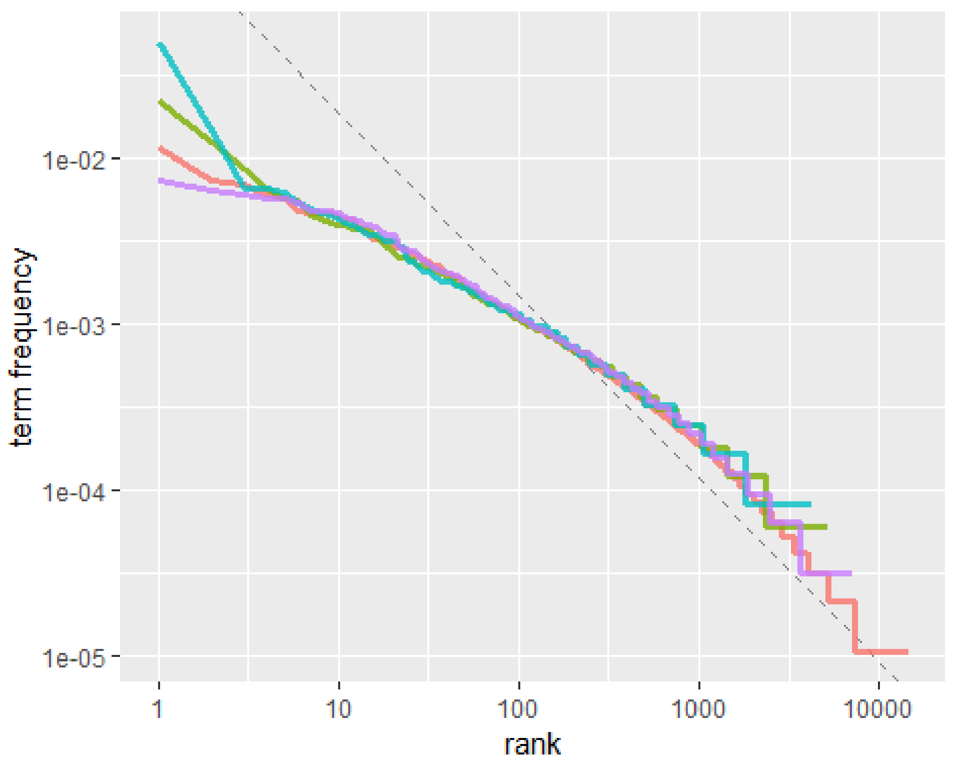
```
rank_subset <- freq_by_rank %>%
  filter(rank < 500,
         rank > 10)
lm(log10(`term frequency`) ~ log10(rank), data = rank_subset)

##
## Call:
## lm(formula = log10(`term frequency`) ~ log10(rank), data = rank_subset)
##
## Coefficients:
## (Intercept)  log10(rank)
##     -1.6528      -0.6521
```

Classic versions of Zipf's law have

$$frequency = \propto \frac{1}{rank}$$

and we have in fact gotten a slope close to -1 here. Let's plot this fitted power law with the data in Figure 3.3 to see how it looks

```
freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = book)) +
  geom_abline(intercept = -0.62, slope = -1.1, color = "gray50", linetype =
2) +
  geom_line(size = 1.1, alpha = 0.8, show.legend = FALSE) +
  scale_x_log10() +
  scale_y_log10()
```



We have found a result close to the classic version of Zipf's law for the corpus of four "philosophy" books. The deviations we see here at high rank are not uncommon for many kinds of language; a corpus of language often contains fewer rare words than predicted by a single power law. The deviations at low rank are more unusual. Our books use a lower percentage of the most common words than many collections of language. This kind of analysis could be extended to compare authors, or to compare any other collections of text; it can be implemented simply using tidy data principles.

The bind_tf_idf function

The idea of tf-idf is to find the important words for the content of each document by decreasing the weight for commonly used words and increasing the weight for words that are not used very much in a collection or corpus of documents, in this case, the group of books as a whole. Calculating tf-idf attempts to find the words that are important (i.e., common) in a text, but not too common. Let's do that now.

The bind_tf_idf function in the tidytext package takes a tidy text dataset as input with one row per token (term), per document. One column (word here) contains the terms/tokens, one column contains the documents (book in this case), and the last necessary column contains the counts, how many times each document contains each term (n in this example). We calculated a total for each book for our explorations in previous sections, but it is not necessary for the bind_tf_idf function; the table only needs to contain all the words in each document.

```r
book_words <- book_words %>%
  bind_tf_idf(word, book, n)
book_words
```

```
## # A tibble: 31,228 x 7
##    book              word          n total      tf   idf  tf_idf
##    <chr>             <chr>     <int> <int>   <dbl> <dbl>   <dbl>
##  1 Indian Philosophy world      1073 93145 0.0115   0    0
##  2 Indian Philosophy life        693 93145 0.00744  0    0
##  3 Indian Philosophy god         660 93145 0.00709  0    0
##  4 Quantum Love      love        604 12077 0.0500   0    0
##  5 Indian Philosophy buddha      566 93145 0.00608  1.39 0.00842
##  6 Indian Philosophy reality     538 93145 0.00578  0    0
##  7 Indian Philosophy nature      451 93145 0.00484  0    0
##  8 Indian Philosophy soul        443 93145 0.00476  0    0
##  9 Indian Philosophy upanisads   439 93145 0.00471  1.39 0.00653
## 10 Indian Philosophy existence   437 93145 0.00469  0    0
## # ... with 31,218 more rows
```

Notice that idf and thus tf-idf are zero for these extremely common words. These are all words that appear in all all four books, so the idf term (which will then be the natural log of 1) is zero. The inverse document frequency (and thus tf-idf) is very low (near zero) for words that occur in many of the documents in a collection; this is how this approach decreases the weight for common words. The inverse document frequency will be a higher number for words that occur in fewer of the documents in the collection.

Let’s look at terms with high tf-idf in book collection.

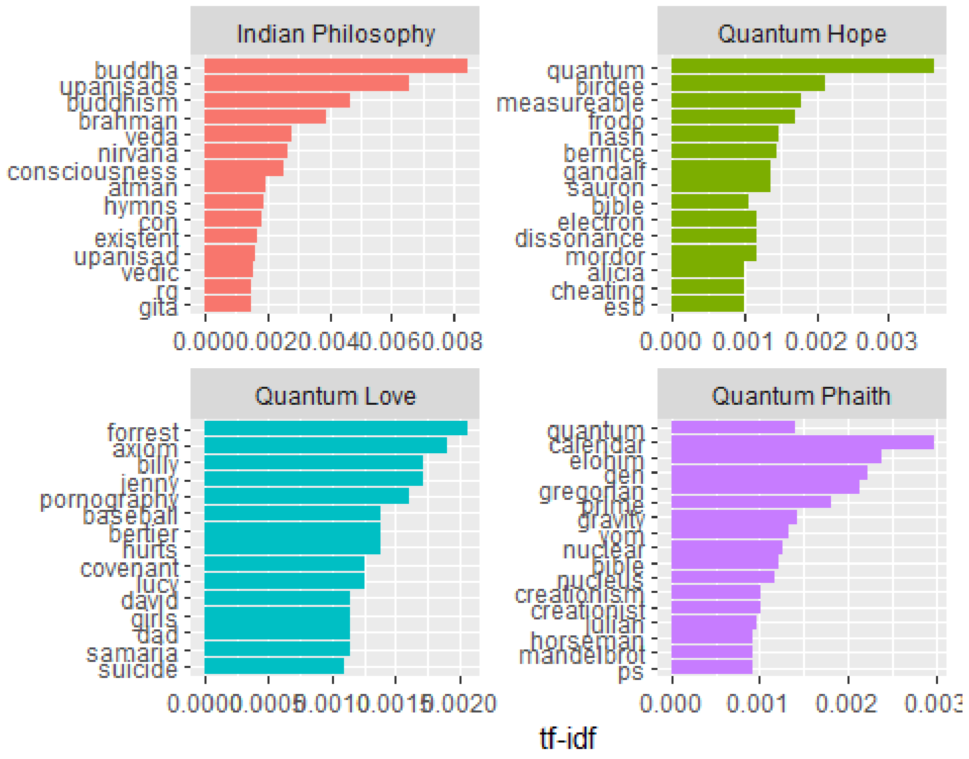
```r
book_words %>%
  select(-total) %>%
  arrange(desc(tf_idf))
```

```
## # A tibble: 31,228 x 6
##    book              word              n      tf   idf  tf_idf
##    <chr>             <chr>         <int>   <dbl> <dbl>   <dbl>
##  1 Indian Philosophy buddha          566 0.00608 1.39  0.00842
##  2 Indian Philosophy upanisads       439 0.00471 1.39  0.00653
##  3 Indian Philosophy buddhism        313 0.00336 1.39  0.00466
##  4 Indian Philosophy brahman         260 0.00279 1.39  0.00387
##  5 Quantum Hope      quantum         205 0.0126  0.288 0.00362
##  6 Quantum Phaith    calendar        134 0.00429 0.693 0.00297
##  7 Indian Philosophy veda            186 0.00200 1.39  0.00277
##  8 Indian Philosophy nirvana         180 0.00193 1.39  0.00268
##  9 Indian Philosophy consciousness   344 0.00369 0.693 0.00256
## 10 Quantum Phaith    elohim           54 0.00173 1.39  0.00240
## # ... with 31,218 more rows
```

Here we see all proper nouns, names that are in fact important in these novels. None of them occur in all of novels, and they are important, characteristic words for each text within the corpus of books.

We wil examine a visualization for these high tf-idf words in the next plot.

```
book_words %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(book) %>%
  top_n(15) %>%
  ungroup %>%
  ggplot(aes(word, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 2, scales = "free") +
  coord_flip()

## Selecting by tf_idf
```



Still all proper nouns in Figure 3.4! These words are, as measured by tf-idf, the most important to each novel and most readers would likely agree. What measuring tf-idf has done here is show us that our authors used similar language across their four books. This is the point of tf-idf; it identifies words that are important to one document within a collection of documents.

Summary

Using term frequency and inverse document frequency allows us to find words that are characteristic for one document within a collection of documents, whether that document is a novel or physics text or webpage. Exploring term frequency on its own can give us insight into how language is used in a collection of natural language, and dplyr verbs like `count()` and `rank()` give us tools to reason about term frequency. The tidytext package uses an implementation of tf-idf consistent with tidy data principles that enables us to see how different words are important in documents within a collection or corpus of documents.