Lab 9: Text as data

Solutions

October 5, 2021

About this lab

Did you know the Emily Dickinson Museum is located within walking distance of the Amherst College campus? Prof. Horton started a tradition of taking his Data Science students to visit the museum. We won't be continuing that tradition, but I encourage you to take a tour once it opens again (closed for a major restoration project until Spring 2022)!

Today we're going to analyze Dickinson's poetry. The text for her poems were scraped from Wikipedia (the final web scraping code I used is available on Moodle).

Packages

In addition to the familiar **tidyverse** and **janitor** packages, there are three new packages we'll be using:

1. the **tidytext** package, which makes text analysis easier and is consistent with the tools we've been using in the **tidyverse** package;
2. the **wordcloud** package which allows us to visually represent the text data in word clouds; and
3. the **textdata** package which allows us to access lexicons for sentiment analysis.

Make sure you load each package in the `setup` code chunk above.

The data

We'll be working with the set of Emily Dickinson's poems we scraped from Wikipedia, available in the file "dickinson-poems.txt" (although it's a txt file, we can load this file using `read_csv()`). We'll remove the cases where the poem text is missing.

```
poems <- read_csv("dickinson-poems.txt") %>%
  filter(text != "Missing")
```

A note on filepaths and your working directory

When you knit an Rmd file, it searches for files relative to where the Rmd file is located. Here are four use cases for reading in a file called "my-data.csv" that you can generalize to other situations:

1. The dataset is in the same folder as the Rmd file:

```
mydata <- read_csv("my-data.csv")
```

2. The dataset is in a subfolder called "data" within the folder the Rmd file is in:

```
mydata <- read_csv("data/my-data.csv")
```

3. The dataset is up a folder relative to the folder the Rmd file is in (e.g., the data are in a folder called "my-project" and the Rmd file is in "my-project/code"):

```
mydata <- read_csv("../my-data.csv")
```

4. The dataset is in one subfolder and the Rmd file is in another subfolder of the same directory (e.g., the data has path "my-project/data" and the Rmd file has path "my-project/code"):

```
mydata <- read_csv("../data/my-data.csv")
```

When you are coding interactively, you'll want to make sure your working directory is the same as the folder your Rmd file is in so that the filepaths you specify work. You can check your working directory in the console by typing `getwd()` and hitting enter. If it doesn't match, one way to set your working directory in RStudio is to navigate to the folder in the **Files** pane, click on the **More** gear menu in that pane, and choose **Set As Working Directory**.

Part 1 **Tidying text** In this part of the lab, we'll work through pre-processing a text using the **tidytext** package.

1.1 **Tokenizing** Tokenizing a text is the process of splitting the text from it's full form and splitting it into smaller units (e.g., sentences, lines, words, etc.). We do this with the `unnest_tokens()` function from the **tidytext** package, which takes on two main arguments: `output` and `input`. `output` creates a new variable that will hold the smaller units of text, and `input` identifies the variable in your dataframe that holds the full text. In the process, we get a long version of the dataset. Run the code below and view the `poems_words_all` dataset and compare it to the `poems` dataset to see these changes.

```
poems_words_all <- poems %>%
  unnest_tokens(output = word, input = text)
```

1.2 **Alternative tokens** The default unit for tokens is a word, but you can specify the `token =` option to tokenize the text by other functions, such as "characters", "ngrams" (n words that occur together), "sentences", or "lines", among other options. Try one or more of these alternative options, using the help as a guide, and see how it changes the output.

```
poems_ngrams <- poems %>%
  unnest_tokens(output = bigram, input = text,
                token = "ngrams", n = 2)

# 389,220 characters!
poems_characters <- poems %>%
  unnest_tokens(output = character, input = text,
                token = "characters")
```

1.3 **Removing stop words** Many commonly used words like "the", "if", and "or" don't provide any insight into the text and are not useful for analysis. These are called *stop words* and are typically removed from a body of text before analysis. The **tidytext** package provides a dataframe with stop words from three differnt lexicons (SMART, snowball, and onix). We can use this `stop_words` dataset and `anti_join()` to remove all the stop words from our `poems_words_all` dataset.

```
data(stop_words)

# First, take a look at the stop_words dataset
head(stop_words)
```

```
# A tibble: 6 x 2
  word      lexicon
  <chr>     <chr>
1 a         SMART
2 a's       SMART
3 able      SMART
4 about     SMART
```

```
     5 above     SMART
     6 according SMART
```

```
tail(stop_words)
```

```
     # A tibble: 6 x 2
       word     lexicon
       <chr>    <chr>
     1 you      onix
     2 young    onix
     3 younger  onix
     4 youngest onix
     5 your     onix
     6 yours    onix
```

```
stop_words %>%
  count(lexicon)
```

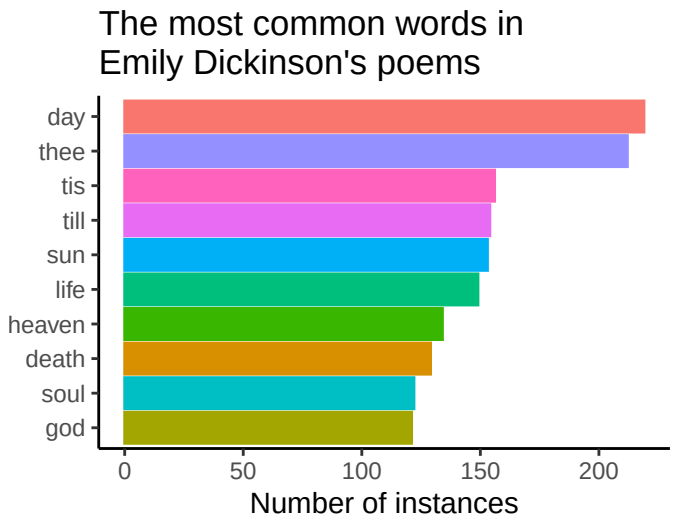
```
     # A tibble: 3 x 2
       lexicon      n
       <chr>    <int>
     1 onix       404
     2 SMART      571
     3 snowball   174
```

```
# Create new dataset since we are removing words
poems_words <- poems_words_all %>%
  anti_join(stop_words, by = "word")

# Explore which stop words were removed
## If you don't want all these words removed, you can modify
## the stop_words dataframe before `anti_join`ing
removed <- poems_words_all %>%
  anti_join(poems_words, by = "word") %>%
  count(word) %>%
  arrange(word)
```

Part 2 **Term frequency** Once our text has been pre-processed, we can use functions we already know and love to create a simple descriptive analysis of the term frequency.

2.1 **Common words plot** Run the code below to create a simple plot of the 10 most common words used by Emily Dickinson. *Note*: The `slice()` function is used to select a subset of rows from a dataframe.

```
poems_words %>%
  count(word, sort = TRUE) %>%
  slice(1:10) %>%
  ggplot(aes(x = reorder(word, n), y = n,
             color = word, fill = word)) +
  geom_col() +
  coord_flip() +
  guides(color = "none", fill = "none") +
  labs(x = NULL,
       y = "Number of instances",
       title = "The most common words in\nEmily Dickinson's poems")
```

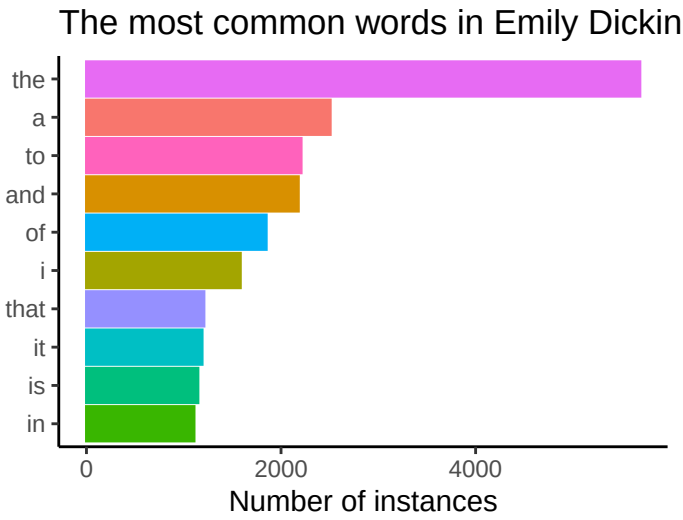


2.2 Run the same code but using the `poems_words_all` dataset. What do you notice about this graphic, and what does this tell us about the utility of removing stop words before analysis?

All the top ten words are stop words!

```
poems_words_all %>%
  count(word, sort = TRUE) %>%
  slice(1:10) %>%
  ggplot(aes(x = reorder(word, n), y = n,
             color = word, fill = word)) +
  geom_col() +
  coord_flip() +
  labs(x = NULL,
       y = "Number of instances",
       title = "The most common words in Emily Dickinson's poems") +
```

```
  guides(color = "none", fill = "none")
```



2.3 To recap, it really only took 4 lines of code to get from our scraped dataset to a dataset formatted for plotting word frequencies:

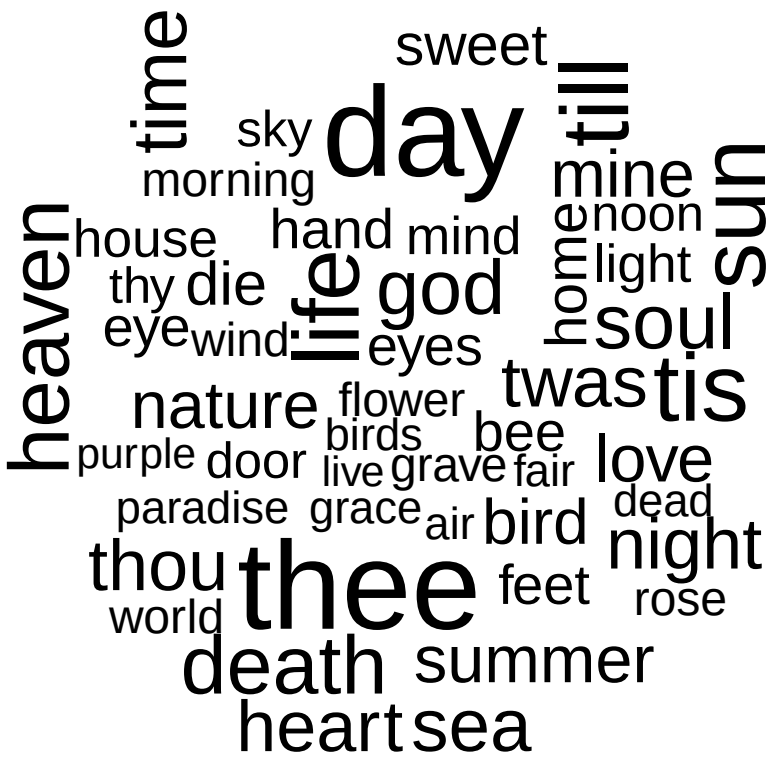
```
word_frequencies <- poems %>%
  unnest_tokens(output = word, input = text) %>%
  anti_join(stop_words, by = "word") %>%
  count(word, sort = TRUE)
```

Part 3 **Word clouds** Word clouds can be used as a quick visualization of the prevalence of words in a corpus.

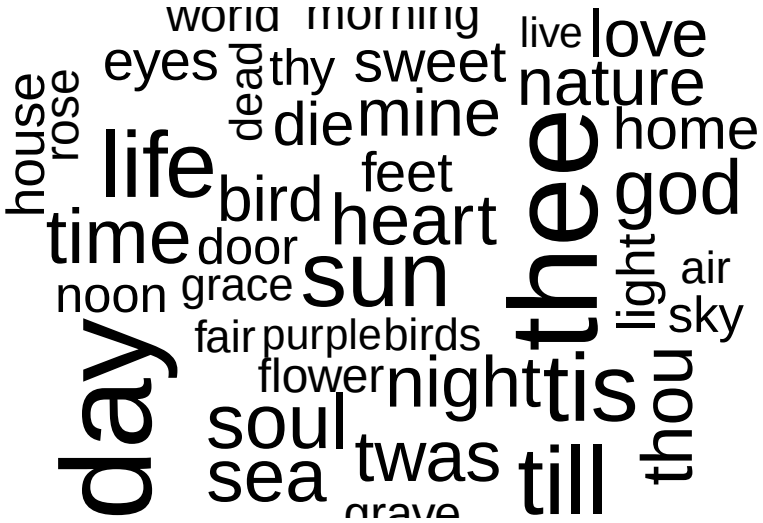
3.1 **Basic word cloud** We can get a bare-bones word cloud using the `wordcloud()` function from the **wordcloud** package.

Note: if you get an error "Error in plot.new() : figure margins too large" or a message "[word] could not be fit on page. It will not be plotted.", try re-adjusting the size of the plotting pane.

```
# Word cloud will rearrange each time unless seed is set
set.seed(53)

# Using pipes
word_frequencies %>%
  with(wordcloud(words = word, freq = n, max.words = 50))
```




```
# Using base R to reference variables directly
wordcloud(words = word_frequencies$word,
          freq = word_frequencies$n,
          max.words = 50)
```

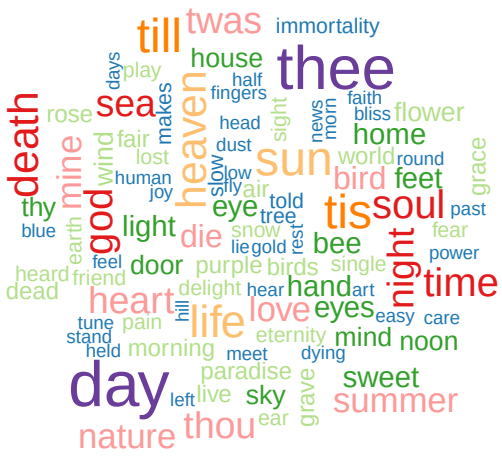


3.2 **Custom word cloud** We can customize the word cloud by mapping the size and color of words to their frequency.

```
# choose color palette from color brewer
mypal <- brewer.pal(10, "Paired")

wordcloud(words = word_frequencies$word,
          freq = word_frequencies$n,
          min.freq = 20,
          max.words = 50,
          # plot the words in a random order
          random.order = TRUE,
          # specify the range of the size of the words
          scale = c(2, 0.3),
          # specify proportion of words with 90 degree rotation
          rot.per = 0.15,
          # colors words from least to most frequent
          colors = mypal,
          # font family
          family = "sans")
```

3.3 **Your turn!** Create your own word cloud with 100 words.

```
wordcloud(words = word_frequencies$word,
          freq = word_frequencies$n,
          max.words = 100,
          # plot the words in a random order
          random.order = TRUE,
          # specify the range of the size of the words
          scale = c(2, 0.3),
          # specify proportion of words with 90 degree rotation
          rot.per = 0.15,
          # colors words from least to most frequent
          colors = mypal,
          # font family
          family="sans")
```



Part 4 **Term frequency-inverse document frequency (tf-idf)** 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 corpus.

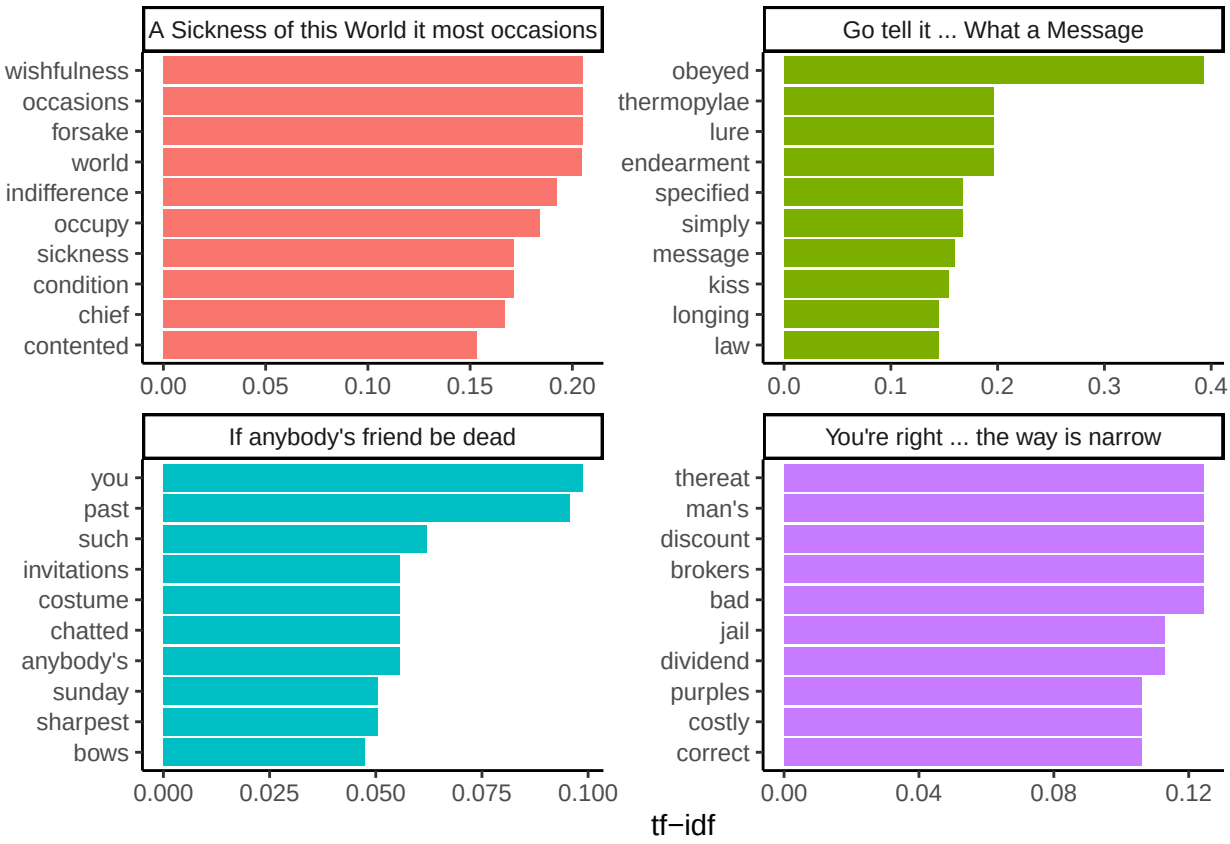
4.1 **Computing term frequency statistics** The `bind_tf_idf()` function will compute the term frequency (tf), inverse document frequency (idf), and the tf-idf statistics for us. It requires a dataset with one row per word per poem, meaning we need a variable to indicate which poem the word comes from (`title`, in our tokenized dataset), a variable to indicate the word (`word` in the tokenized dataset), and a third variable to indicate the number of times that word appears in that specific poem. *This time, we do not need to remove stop words. Why not?*

The tf-idf takes into account word frequency. For words like "a", "the" and "or" that occur very frequently, their inverse document frequency will be very low and therefore their tf-idf will be very low.

```
word_freqs_by_poem <- poems %>%
  unnest_tokens(output = word, input = text) %>%
  group_by(title) %>%
  count(word)

poems_tfidf <- word_freqs_by_poem %>%
  bind_tf_idf(term = word,
              document = title,
              n = n)
```

4.2 **Visualizing tf-idf** We can visualize the words with the highest 10 tf-idf values for a subset of the poems using the code below.

```
set.seed(32)
poems_subset <- sample(poems$title, size = 4)

top_tfidf <- poems_tfidf %>%
  filter(title %in% poems_subset) %>%
  arrange(desc(tf_idf)) %>%
  group_by(title) %>%
  slice(1:10) %>%
  ungroup()

ggplot(data = top_tfidf, aes(x = reorder(word, tf_idf), y = tf_idf,
                             fill = title)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ title, ncol = 2, scales = "free") +
  coord_flip() +
  labs(x = NULL, y = "tf-idf")
```


A Sickness of this World it most occasions
wishfulness
occasions
forsake
world
indifference
occupy
sickness
condition
chief
contented
0.00
0.05
0.10
0.15
0.20
Go tell it ... What a Message
obeyed
thermopylae
lure
endearment
specified
simply
message
kiss
longing
law
0.0
0.1
0.2
0.3
0.4
If anybody's friend be dead
you
past
such
invitations
costume
chatted
anybody's
sunday
sharpest
bows
0.000
0.025
0.050
0.075
0.100
You're right ... the way is narrow
thereat
man's
discount
brokers
bad
jail
dividend
purples
costly
correct
0.00
0.04
0.08
0.12
tf−idf

Part 5 **Sentiment analysis** What is sentiment analysis? From Text Mining with R (Silge & Robinson 2019): *"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... One way to analyze the sentiment of a text is to consider the text as a combination of its individual words and the sentiment content of the whole text as the sum of the sentiment content of the individual words. This isn't the only way to approach sentiment analysis, but it is an often-used approach."* There are different lexicons that can be used to classify the sentiment of text. Today, we'll compare two different lexicons that are both based on unigrams, the AFINN lexicon and the NRC lexicon.

5.1 The AFINN lexicon (Nielsen 2011) assigns words a score from -5 (negative sentiment) to +5 (positive sentiment). Check out the AFINN lexicon using the code below. What do you think of the scores? What is the rating for the word "slick"? Does "slick" always have a positive connotation (can you think of a sentence where "slick" has a negative connotation)?

There are only 5 words that get assigned a value of 5 (most positive) and 16 words that get assigned a value of -5 (most negative). The majority of words are assigned values between -2 and 2, which probably reflects the fact that their sentiment can change depending on context, rather than that they're truly that neutral in every context.

Slick has rating +2. This is interesting because "slick" is sometimes used to refer to somebody who is trying to get away with something.

```
# Type "Yes" to download if prompted
afinn_lexicon <- get_sentiments("afinn")

afinn_lexicon %>%
  filter(word == "slick")
```

```
# A tibble: 1 x 2
  word  value
  <chr> <dbl>
1 slick     2
```

5.2 Use the `get_sentiments()` function to create a dataframe `nrc_lexicon` that holds the NRC Word-Emotion Association lexicon (Mohammad 2010). The NRC lexicon catergoizes words as yes/no for the following sentiment categories: positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. What does each row in this dataset represent? (Hint: it's *not* the same as the `afinn_lexicon` dataset.)

Each row in this lexicon captures a different sentiment that a word could be.

```
nrc_lexicon <- get_sentiments("nrc")

head(nrc_lexicon)
```

```
# A tibble: 6 x 2
  word      sentiment
  <chr>     <chr>
```

```
      1 abacus    trust
      2 abandon   fear
      3 abandon   negative
      4 abandon   sadness
      5 abandoned anger
      6 abandoned fear
```

5.3 User (and Consumer!) Beware: Do you see any issues in applying these lexicons (developed fairly recently) to the Emily Dickinson poems?

The meaning and sentiment of a particular word may have changed over the years, such that the poet's intended sentiment is not how it would be classified today. In addition, many words may not be included in the lexicons because of the differences in language used in the 1800s and in the 2000s. (Note that many words used today are also not included in some of the lexicons because they are neutral, having neither a particularly positive or negative sentiment).

5.4 The lexicons are based on unigrams. Do you see any disadvantages of basing the sentiment on single words?

It does not account for qualifiers (e.g. "not good" only counts "good"; "somewhat happy" only counts "happy"). A text with many qualifiers could be rated incorrectly as being mostly positive or negative because it does not consider the negating.

5.5 We can calculate how many words used in the poems are not found in the lexicons using the code below. List a few words that are not in the NRC lexicon that appear in the poems. What proportion of unigrams observed within this corpora of Dickinson poems are *not* scored by the NRC lexicon?

About 78% of words in Dickinson's poems are not scored by the NRC lexicon (yikes!). Missed words includes words like "soul", "heaven" and "brave", which was surprising to me.

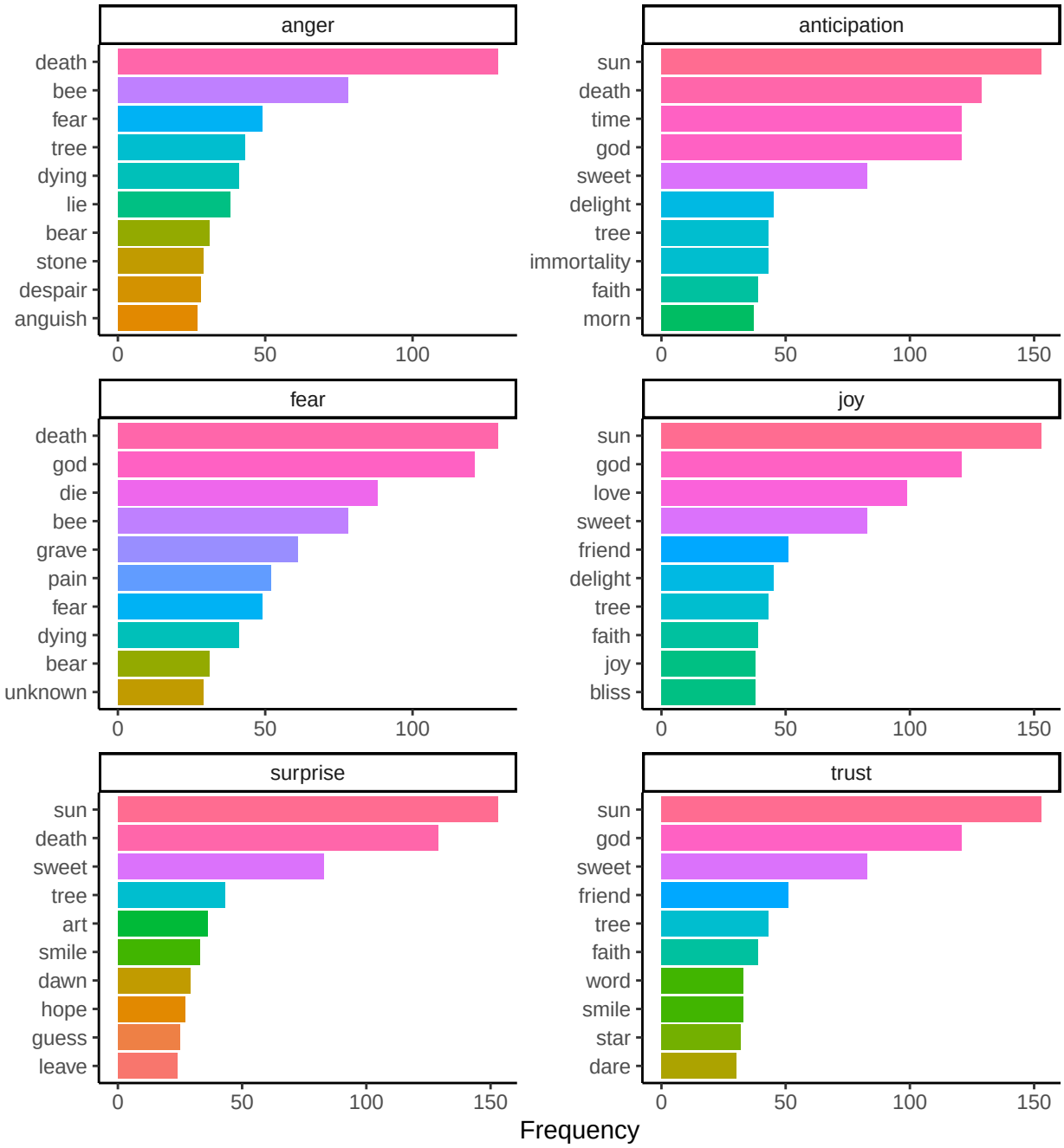
```
nrc_missed_words <- word_frequencies %>%
  anti_join(nrc_lexicon, by = "word")

nrow(nrc_missed_words)/nrow(word_frequencies)
```

```
      [1] 0.7815143
```

5.6 With these (rather important!) drawbacks in mind, let's go ahead and view the top words by sentiment classified by the NRC lexicon. That is, create a figure of the top 10 words under each sentiment, facetted by sentiment, for the following sentiments: anger, anticipation, fear, joy, surprise, and trust. You can use code given in earlier chunks to guide you.

```
nrc_poems <- word_frequencies %>%
  inner_join(nrc_lexicon, by = "word") %>%
  filter(sentiment %in% c("anger", "anticipation",
                          "fear", "joy",
                          "surprise", "trust")) %>%
  arrange(sentiment, desc(n)) %>%
```

```
  group_by(sentiment) %>%
  slice(1:10)


ggplot(data = nrc_poems, aes(x = reorder(word,n), y = n,
                             fill = as.factor(n))) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  facet_wrap(~ sentiment, ncol = 2, scales = "free") +
  labs(x = NULL, y = "Frequency")
```



5.7 How might you summarize the sentiment of this corpus using the AFINN lexicon?

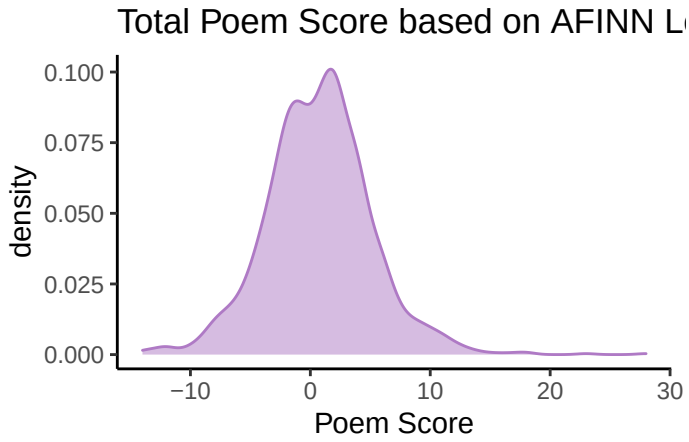
Based on the AFINN lexicon, the collection of Dickinson's poems are positive on the whole, with a total score of 1,111, indicating more positive-valued words than negative-valued words overall.

Alternatively, one could compute a total score for each poem, and then consider the distribution of scores across poems. In this case, across the 1,772 poems, half actually have a total score less than 1 (so would lean neutral to negative). About 25% have a total score of 3 or more so lean positive. It seems like some particularly positive poems are driving the overall score to lean positive.

```
# Overall score
poems_words_all %>%
  inner_join(afinn_lexicon, by = "word") %>%
  summarize(total_score = sum(value))
```

```
# A tibble: 1 x 1
  total_score
        <dbl>
1        1111
```

```
# Scores by poem
afinn_by_poem <- poems_words_all %>%
  inner_join(afinn_lexicon, by = "word") %>%
  group_by(title) %>%
  summarize(poem_score = sum(value)) %>%
  arrange(poem_score)

ggplot(data = afinn_by_poem, aes(x = poem_score)) +
  geom_density(color = "#AF7AC5",
               fill = "#AF7AC5",
               alpha = 0.5) +
  labs(title = "Total Poem Score based on AFINN Lexicon",
       x = "Poem Score")
```




```
mosaic::favstats(data = afinn_by_poem, ~ poem_score)
```

```
 min Q1 median Q3 max      mean       sd    n missing
 -14 -2      1  3  28 0.7177003 4.425322 1548       0
```

References

AFINN Lexicon

Nielsen, FA. A new ANEW: Evaluation of a word list for sentiment analysis in microblogs. Proceedings of the ESWC2011 Workshop on 'Making Sense of Microposts': Big things come in small packages 718 in CEUR Workshop Proceedings 93-98. 2011 May. http://arxiv.org/abs/1103.2903.

NRC Lexicon

Mohammad S, Turney P. Crowdsourcing a Word-Emotion Association Lexicon. *Computational Intelligence*. 2013;29(3):436-465.

Mohammad S, Turney P. Emotions Evoked by Common Words and Phrases: Using Mechanical Turk to Create an Emotion Lexicon. In Proceedings of the NAACL-HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text, June 2010, LA, California. http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm

Text Mining with R

Silge J, Robinson D (2016). "tidytext: Text Mining and Analysis Using Tidy Data Principles in R." *JOSS*, *1*(3). doi: 10.21105/joss.00037.

Silge J, Robinson D (2017). Text Mining with R: A Tidy Approach. O'Reilly Media Inc. Sebastopol, CA. https://www.tidytextmining.com/index.html