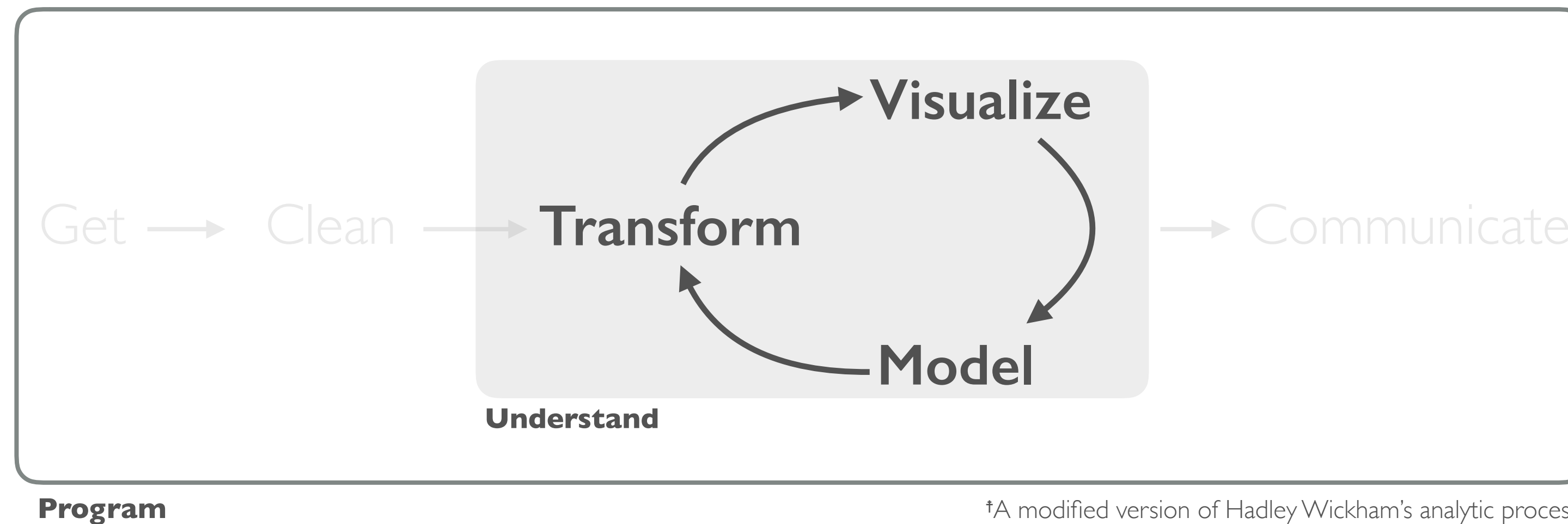


SENTIMENT ANALYSIS



†A modified version of Hadley Wickham's analytic process

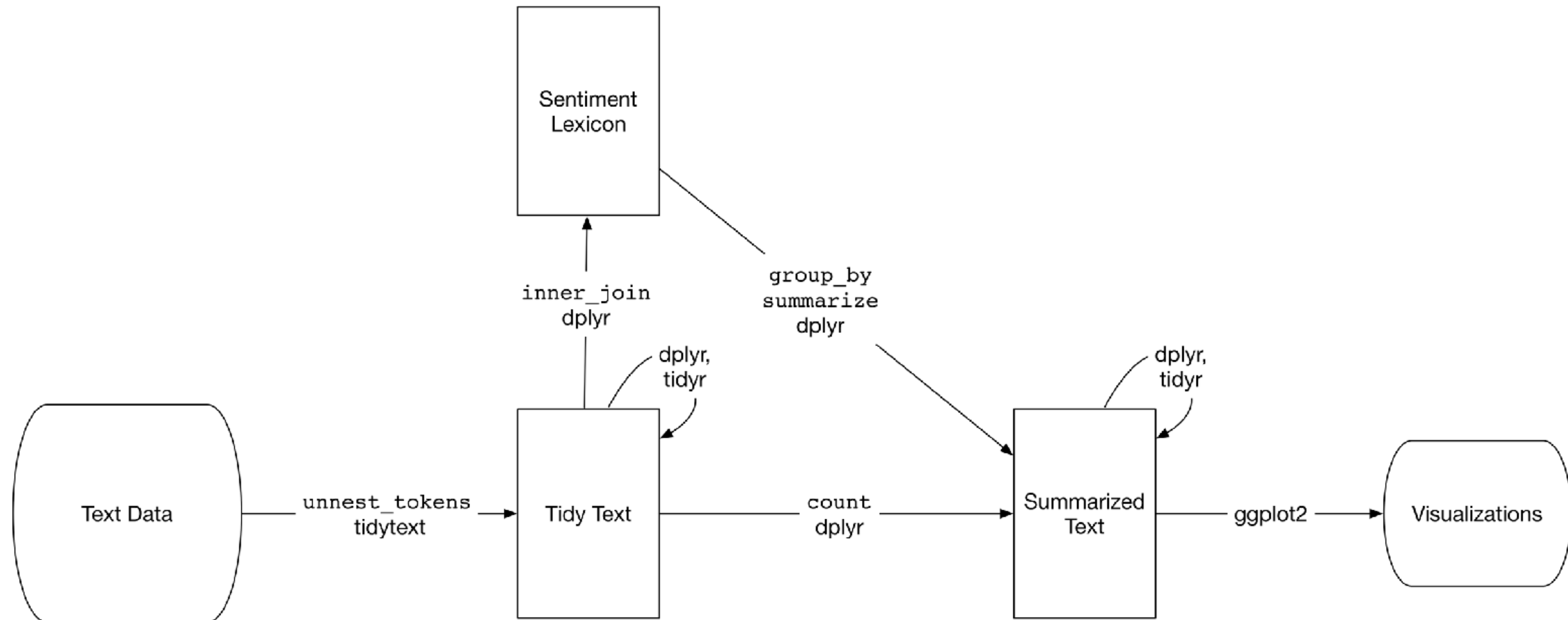
“When human readers approach a text, we use our understanding of the emotional intent of words to infer whether a section of text is positive or negative, or perhaps characterized by some other more nuanced emotion like surprise or disgust.”

— Julia Silge and David Robinson

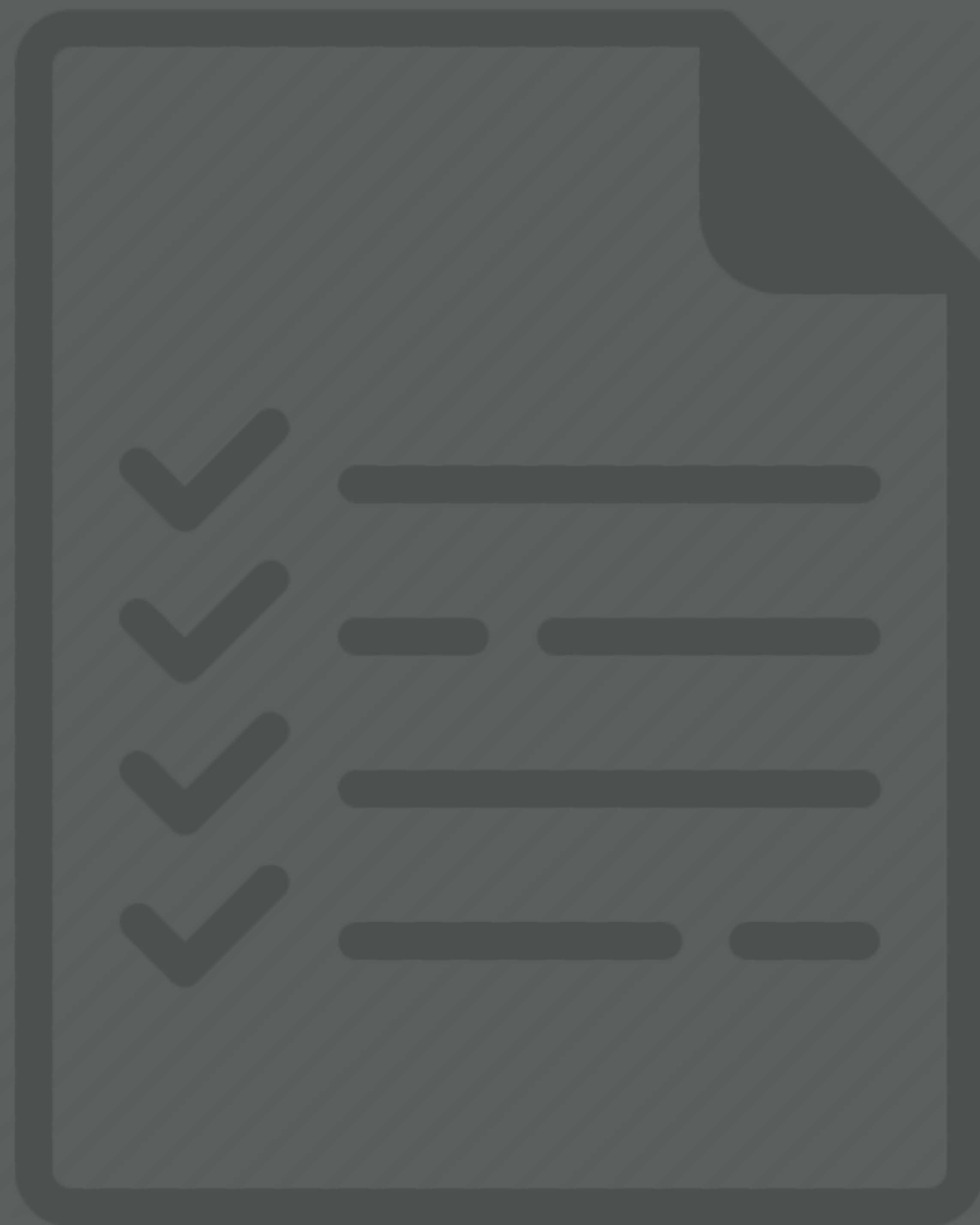
SENTIMENT

- I love this class!
- I hate this teacher, he stinks!
- I feel tired this morning.
- This view is amazing.
- I am not looking forward to the concert.
- There is a book on the desk

SENTIMENT ANALYSIS



PREREQUISITES



PACKAGE PREREQUISITE

```
library(tidyverse)  
library(tidytext)  
library(sentimentr)  
library(magrittr)  
library(harrypotter)
```

SENTIMENT LEXICONS

The good, bad, and the ugly



Negative



Neutral



Positive

SENTIMENT LEXICONS

sentiments

A tibble: 23,165 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 23,155 more rows

- **AFINN** from Finn Årup Nielsen
- **bing** from Bing Liu and collaborators
- **nrc** from Saif Mohammad and Peter Turney
- to see the individual lexicons try
 - `get_sentiments("afinn")`
 - `get_sentiments("bing")`
 - `get_sentiments("nrc")`

SENTIMENT LEXICONS

These lexicons represent different ways to score some contextual words

```
get_sentiments("afinn") %>%  
  count(score)
```

```
# A tibble: 11 x 2
```

```
  score      n  
  <int> <int>
```

1	-5	16
2	-4	43
3	-3	264
4	-2	965
5	-1	309
6	0	1
7	1	208
8	2	448
9	3	172
10	4	45
11	5	5

```
get_sentiments("bing") %>%  
  count(sentiment)
```

```
# A tibble: 2 x 2
```

```
  sentiment      n  
    <chr> <int>
```

1	negative	4782
2	positive	2006

```
get_sentiments("nrc") %>%  
  count(sentiment)
```

```
# A tibble: 10 x 2
```

```
  sentiment      n  
    <chr> <int>
```

1	anger	1247
2	anticipation	839
3	disgust	1058
4	fear	1476
5	joy	689
6	negative	3324
7	positive	2312
8	sadness	1191
9	surprise	534
10	trust	1231

BASIC SENTIMENT ANALYSIS

Finding the good, bad, and the ugly



BASIC SENTIMENT

```
ps_df <- tibble(  
  chapter = seq_along(philosophers_stone),  
  text    = philosophers_stone  
) %>%  
  unnest_tokens(word, text)
```

```
ps_df  
# A tibble: 77,875 x 2  
  chapter word  
    <int> <chr>  
1       1 the  
2       1 boy  
3       1 who  
4       1 lived  
5       1 mr  
6       1 and  
7       1 mrs  
8       1 dursley  
9       1 f
```

- First, let's tidy our text

BASIC SENTIMENT

```
ps_df %>%  
  inner_join(get_sentiments("bing")) %>%  
  count(sentiment, sort = TRUE)  
# A tibble: 2 x 2  
  sentiment      n  
  <chr>      <int>  
1 negative   2335  
2 positive   1986
```

- First, let's tidy our text
- We can get the overall positive vs. negative sentiment with the Bing lexicon.

But what about the most common types of emotions in the text?

BASIC SENTIMENT

```
ps_df %>%
  inner_join(get_sentiments("nrc"))
# A tibble: 18,054 x 3
   chapter word      sentiment
   <int> <chr>    <chr>
1       1 boy      disgust
2       1 boy      negative
3       1 proud    anticipation
4       1 proud    joy
5       1 proud    positive
6       1 proud    trust
7       1 expect  anticipation
8       1 expect  positive
9       1 expect  surprise
10      1 expect  trust
# ... with 18,044 more rows
```

- First, let's tidy our text
- We can get the overall positive vs. negative sentiment with the Bing lexicon.
- We can use the NRC lexicon
- Notice how “boy”, “proud”, “expect” have more than one feeling.

BASIC SENTIMENT

```
ps_df %>%
  inner_join(get_sentiments("nrc")) %>%
  count(sentiment, sort = TRUE)
# A tibble: 10 x 2
#   sentiment      n
#   <chr>      <int>
1 negative    3678
2 positive    2608
3 sadness     2371
4 anger       2239
5 trust       1623
6 anticipation 1439
7 fear        1323
8 joy         1034
9 surprise     873
10 disgust     866
```

- First, let's tidy our text
- We can get the overall positive vs. negative sentiment with the Bing lexicon.
- We can use the NRC lexicon
- Notice how “boy”, “proud”, “expect” have more than one feeling.

YOUR TURN!

*Using the AFINN lexicon, can you rank-order the chapters in *Philosopher's Stone* by sentiment score?*

GRANULARITY

What if we want a finer level of sentiment understanding



FINER GRANULARITY

```
ps_df %>%
  mutate(
    word_count = 1:n(),
    page = word_count %% 275 + 1
  )
# A tibble: 77,875 x 4
  chapter word    word_count page
  <int> <chr>      <int> <dbl>
1      1 the         1  1.00
2      1 boy        2  1.00
3      1 who        3  1.00
4      1 lived       4  1.00
5      1 mr         5  1.00
6      1 and        6  1.00
7      1 mrs       7  1.00
8      1 dursley     8  1.00
9      1 of         9  1.00
10     1 number    10  1.00
```

- We can break up our book by
apprx **page** (250-300 words
per page)

FINER GRANULARITY

```
ps_df %>%
  mutate(
    word_count = 1:n(),
    page = word_count %% 275 + 1
  ) %>%
  inner_join(get_sentiments("bing")) %>%
  count(page, sentiment)
# A tibble: 567 x 3
   page sentiment      n
  <dbl> <chr>      <int>
1  1.00 negative         4
2  1.00 positive         9
3  2.00 negative        10
4  2.00 positive         4
5  3.00 negative         9
6  3.00 positive         1
7  4.00 negative         8
8  4.00 positive         7
```

- We can break up our book by apprx page (250-300 words per page)
- We can then get positive vs negative sentiment by page

FINER GRANULARITY

```
ps_df %>%
  mutate(
    word_count = 1:n(),
    page = word_count %% 275 + 1
  ) %>%
  inner_join(get_sentiments("bing")) %>%
  count(page, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

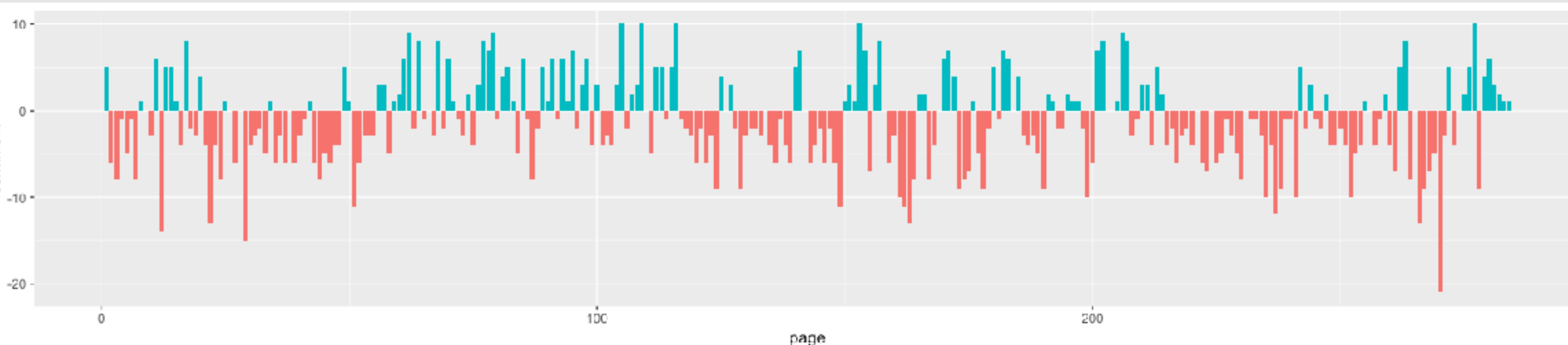
A tibble: 284 x 4

	page	negative	positive	sentiment
	<dbl>	<dbl>	<dbl>	<dbl>
1	1.00	4.00	9.00	5.00
2	2.00	10.0	4.00	-6.00
3	3.00	9.00	1.00	-8.00
4	4.00	8.00	7.00	-1.00
5	5.00	13.0	8.00	-5.00
6	6.00	6.00	5.00	-1.00

- We can break up our book by apprx page (250-300 words per page)
- We can then get positive vs negative sentiment by page
- Then spread our data and compute the net sentiment

FINER GRANULARITY

```
ps_df %>%
  mutate(
    word_count = 1:n(),
    page = word_count %% 275 + 1
  ) %>%
  inner_join(get_sentiments("bing")) %>%
  count(page, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative) %>%
  ggplot(aes(page, sentiment, fill = sentiment > 0)) +
  geom_col(show.legend = FALSE)
```



- We can break up our book by apprx page (250-300 words per page)
- We can then get positive vs negative sentiment by page
- Then spread our data and compute the net sentiment
- And plot the results

YOUR TURN!

Compare the Bing and AFINN sentiment for deathly_hallows. Do they differ?

INFLUENCERS

What is driving sentiment?



DRIVERS OF SENTIMENT

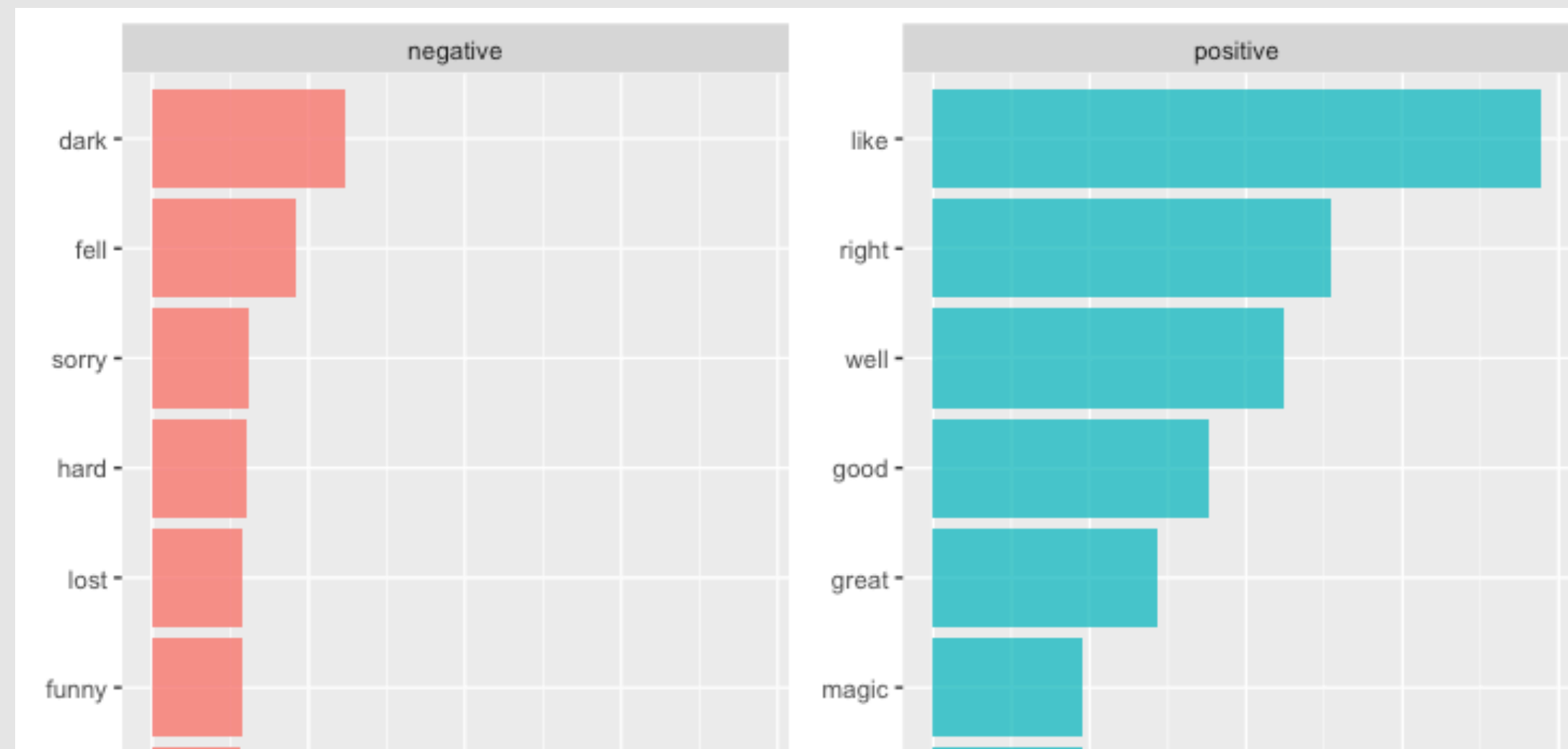
```
ps_df %>%  
  inner_join(get_sentiments("bing")) %>%  
  count(word, sentiment, sort = TRUE)
```

```
# A tibble: 925 x 3  
  word      sentiment      n  
  <chr>    <chr>      <int>  
1 like     positive    194  
2 right    positive    127  
3 well     positive    112  
4 good     positive     88  
5 great    positive     72  
6 dark     negative     62  
7 enough   positive     48  
8 magic    positive     48  
9 fell     negative     46  
10 better  positive     41  
# ... with 915 more rows
```

- To understand what words are driving sentiment we simply count by word and sentiment

DRIVERS OF SENTIMENT

```
ps_df %>%  
  inner_join(get_sentiments("bing")) %>%  
  count(word, sentiment, sort = TRUE) %>%  
  group_by(sentiment) %>%  
  top_n(10) %>%  
  ggplot(aes(reorder(word, n), n, fill = sentiment)) +  
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +  
  facet_wrap(~sentiment, scales = "free_y") +  
  labs(y = "Contribution to sentiment", x = NULL) +  
  coord_flip()
```



- To understand what words are driving sentiment we simply count by word and sentiment
- And then plot the top 10 influential words

RISK OF NEGATION

Getting better context



SENTIMENT

- I haven't been sad in a long time.
- I am extremely happy today.
- It's a good day.
- But suddenly I'm only a little bit happy.
- Then I'm not happy at all.
- In fact, I am now the least happy person on the planet.
- There is no happiness left in me.
- Wait, it's returned!
- I don't feel so bad after all!

OPTION 1: ASSESS AT THE UNIT LEVEL

```
ps_lines <- tibble(
  chapter = seq_along(philosophers_stone),
  text     = philosophers_stone
) %>%
  unnest_tokens(sentence_text, text, token = "sentences") %>%
  mutate(sentence = 1:n()) %>%
  unnest_tokens(word, sentence_text)
```

```
ps_lines %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(sentence) %>%
  summarize(score = sum(score, na.rm = TRUE))
```

```
# A tibble: 2,655 x 2
```

	sentence	score
	<int>	<int>
1	2	7
2	3	-3
3	6	1
4	8	2

- We can unnest at the **sentence** level
- Then we can get the overall sentiment **score** for that sentence.
- However, this still treats each sentence as a bag of words rather than trying to understand the context.

OPTION 2: ASSESS POLARITY AT THE SENTENCE LEVEL

```
ch_sentiment <- tibble(
  chapter = seq_along(philosophers_stone),
  text     = philosophers_stone
) %>%
  sentiment_by(
    get_sentences(text),
    list(chapter)
  )
```



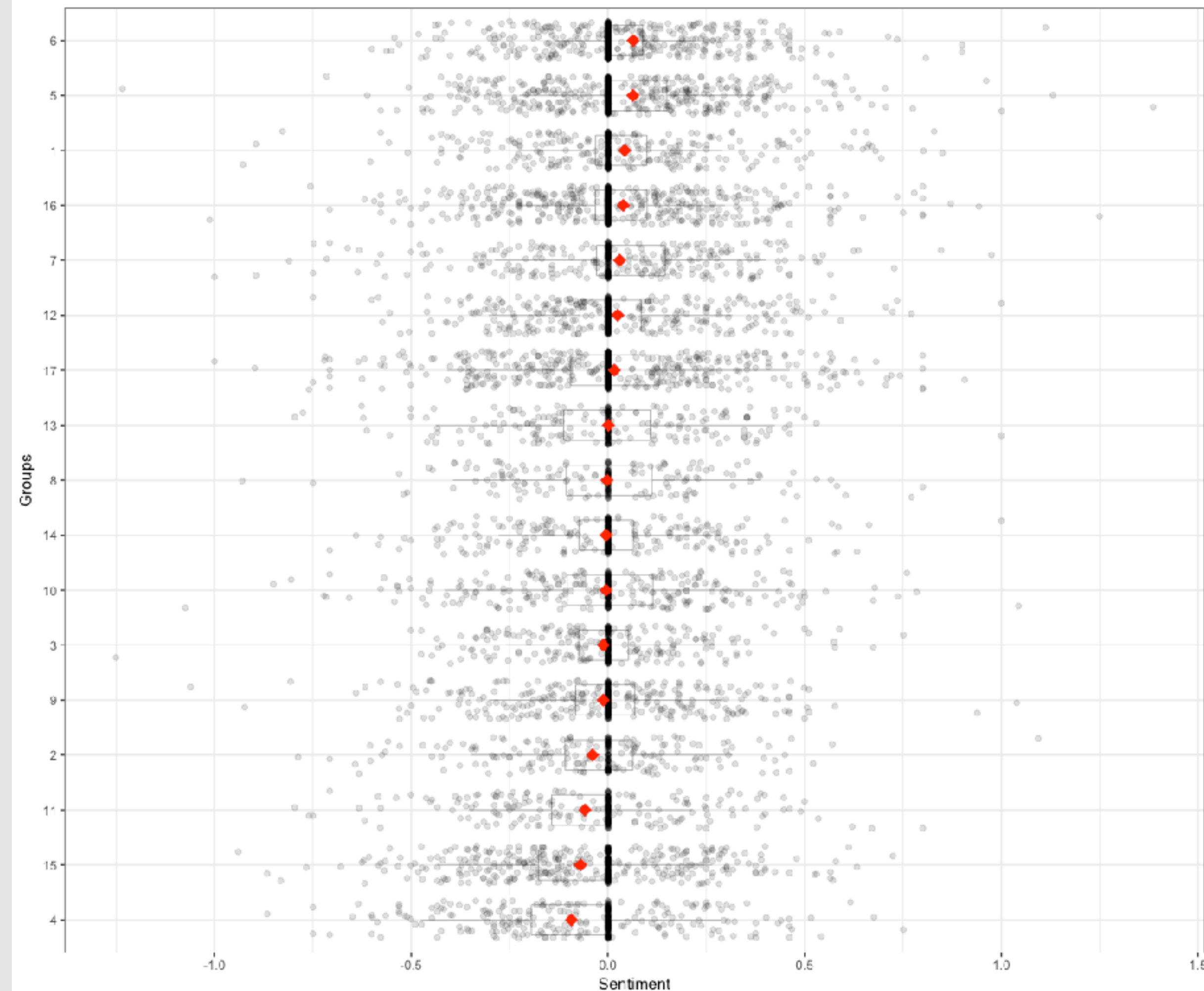
```
ch_sentiment
  chapter word_count      sd ave_sentiment
1:      1      4622 0.2377134  0.042569167
2:      2      3456 0.2292486 -0.039026682
3:      3      3853 0.1980889 -0.011839610
4:      4      3700 0.2420885 -0.092671054
```

- **sentmentr** package provides a method for incorporating weighting for valence shifters (negation and amplifiers/deamplifiers)
- **get_sentence** splits text up by sentence
- **sentiment_by** will assess net sentiment at the sentence level grouped at a certain level (chapters in this example)

OPTION 2: ASSESS POLARITY AT THE SENTENCE LEVEL

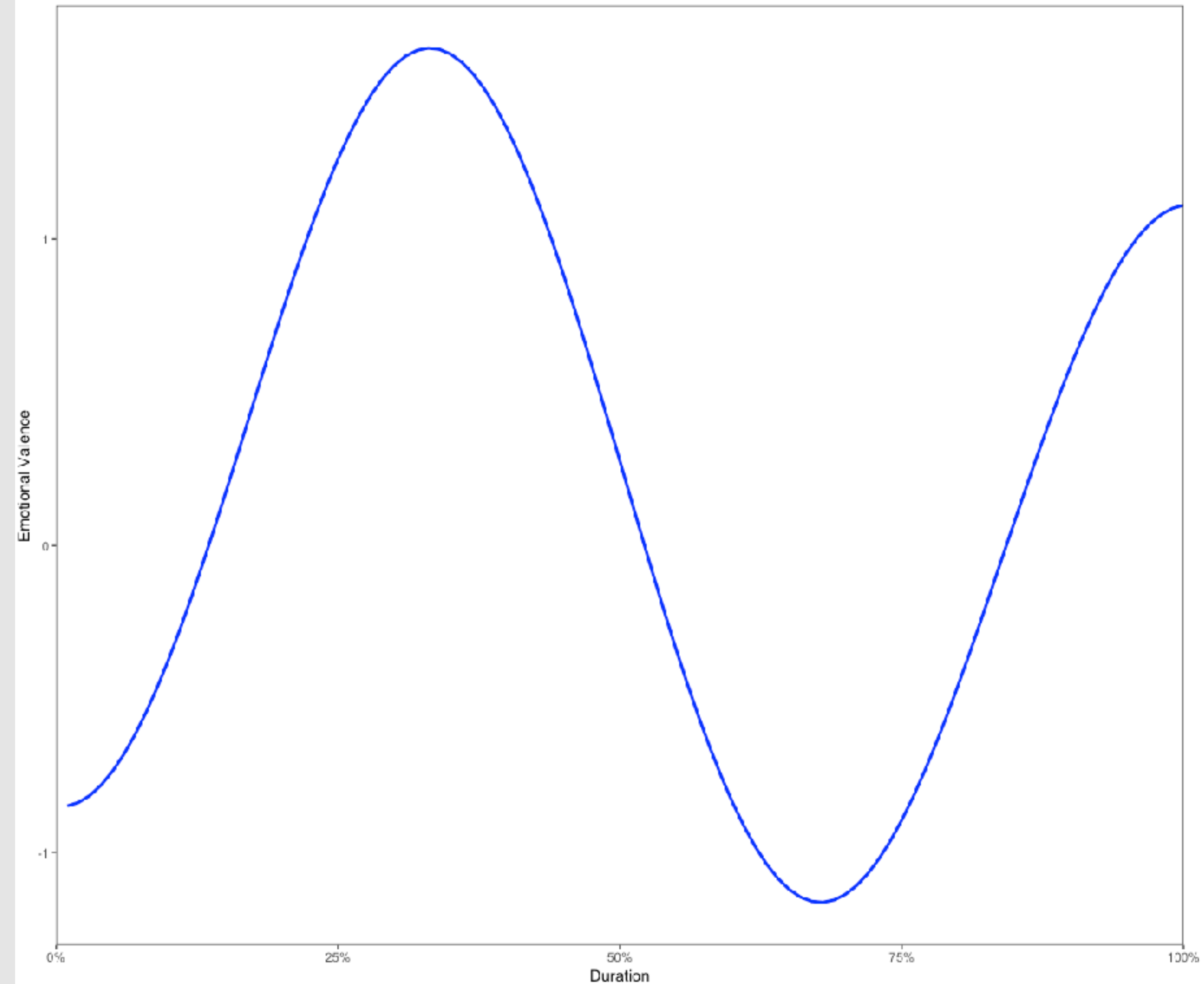
```
plot(ch_sentiment)
```

Chapters in rank order based on average sentiment



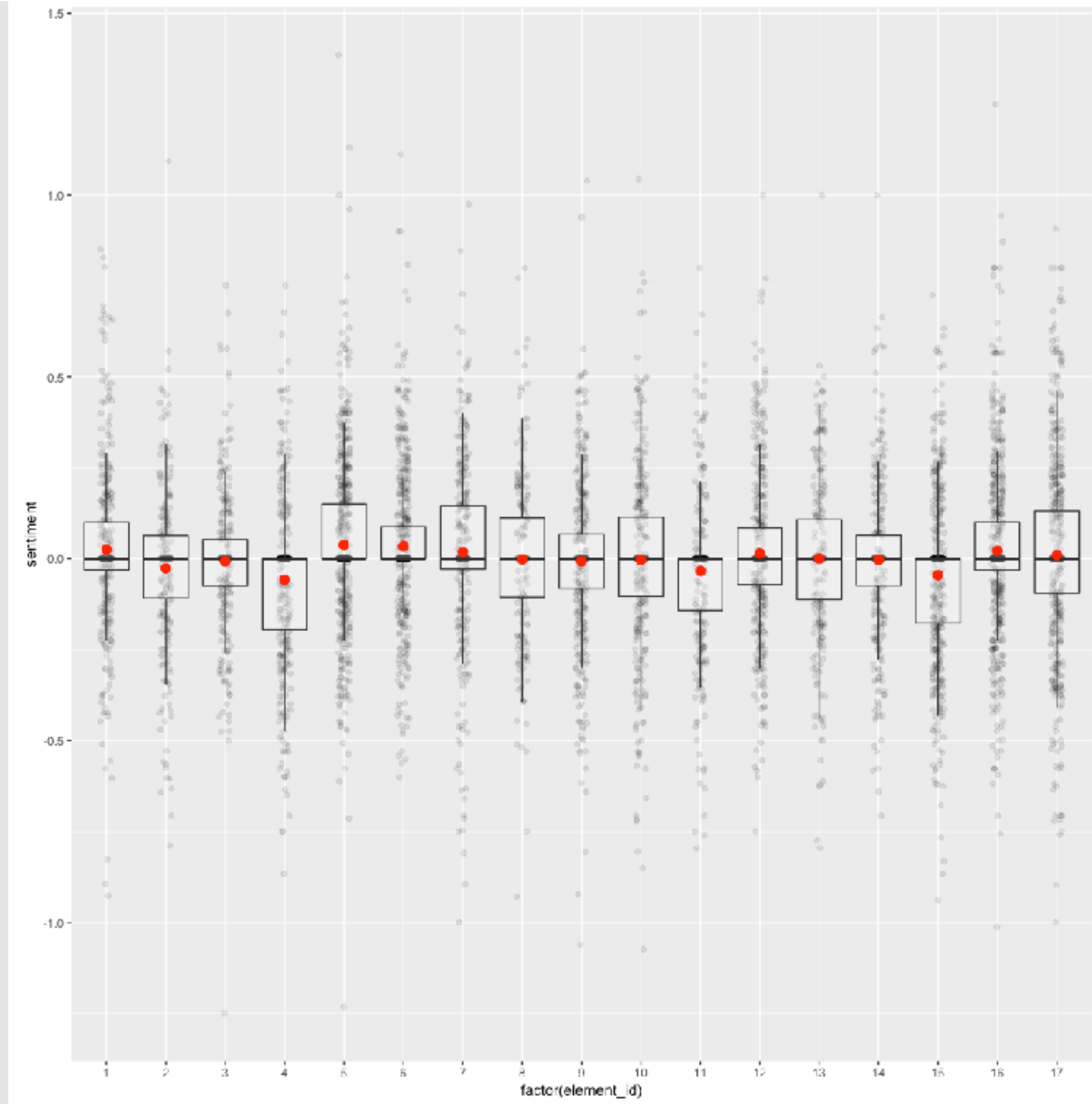
OPTION 2: ASSESS POLARITY AT THE SENTENCE LEVEL

```
plot(ch_sentiment)  
plot(uncombine(ch_sentiment))
```



OPTION 2: ASSESS POLARITY AT THE SENTENCE LEVEL

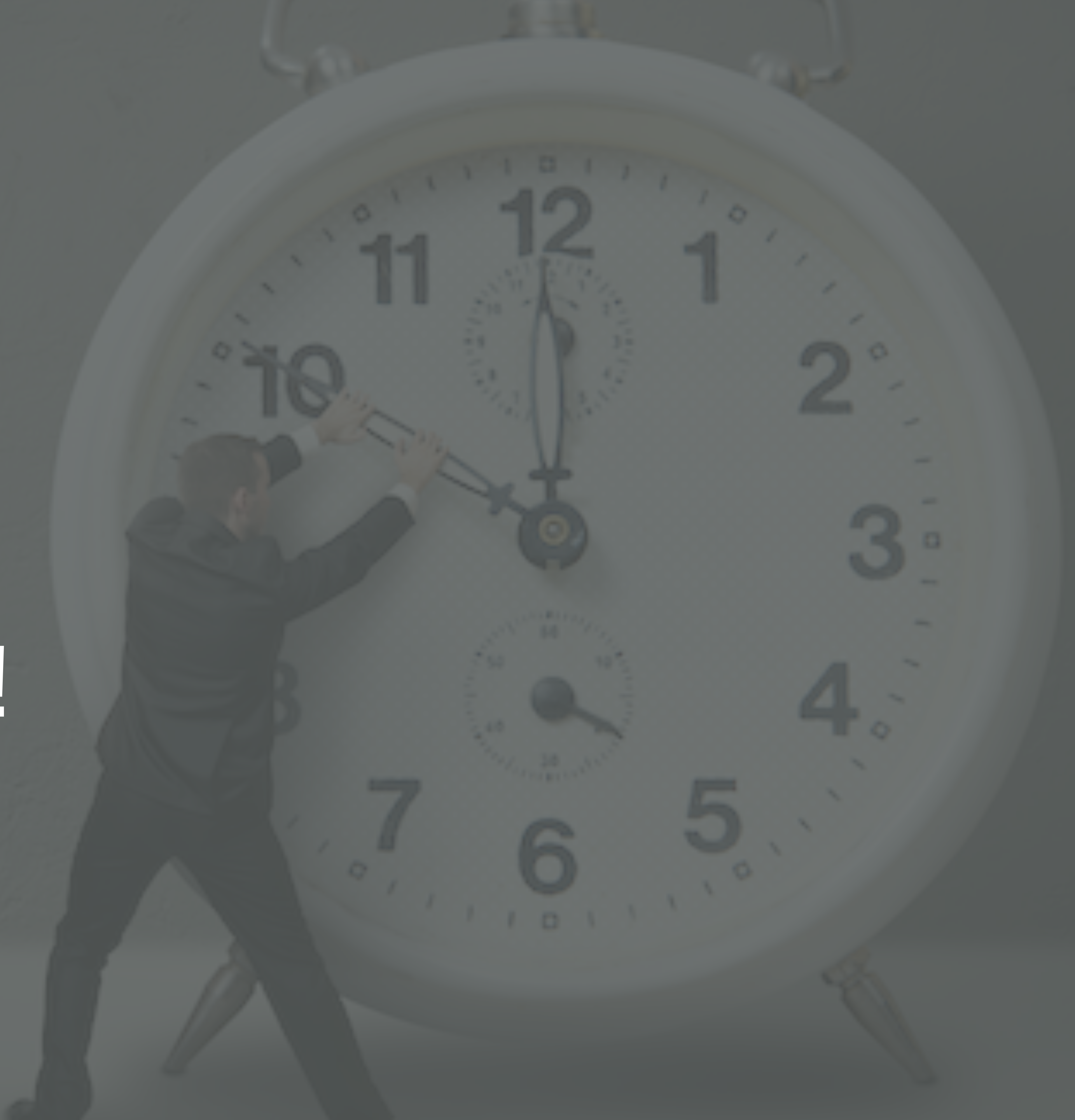
```
plot(ch_sentiment)
plot(uncombine(ch_sentiment))
ch_sentiment %>%
  uncombine() %>%
  ggplot(aes(factor(element_id), sentiment)) +
  geom_jitter(alpha = .1, width = .1, height = 0) +
  geom_boxplot(outlier.shape = NA, alpha = .25) +
  stat_summary(fun.y = "mean", geom = "point", size = 3,
    color = "red")
```



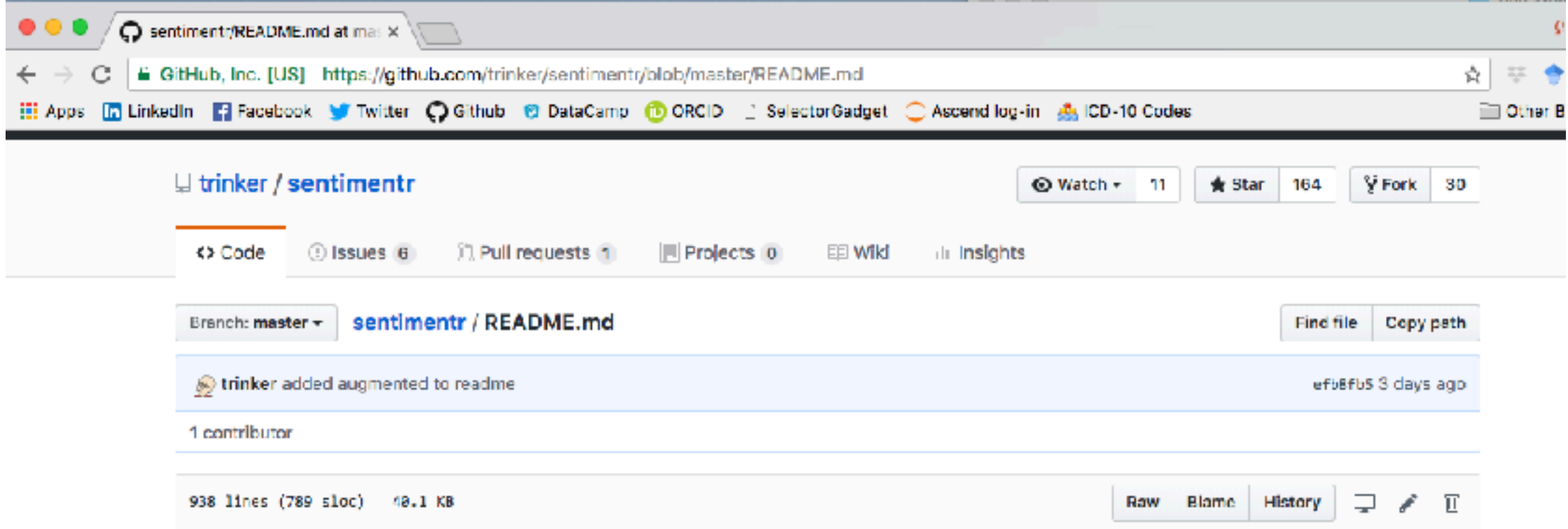
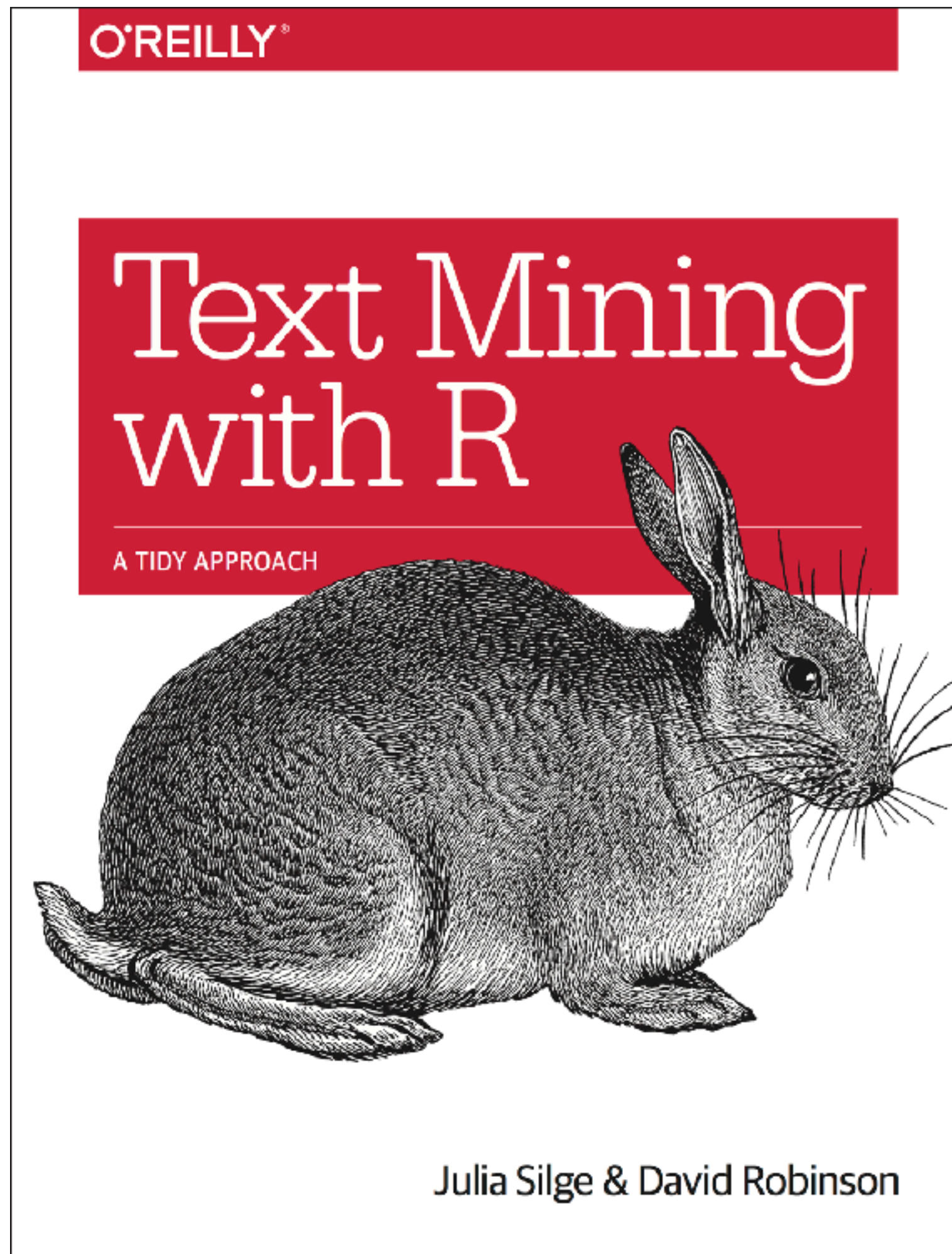
YOUR TURN!

- 1. Rank order the three review files provided in your .R script by overall positive vs. negative sentiment.*
- 2. Identify the top 5 emotions in each file.*
- 3. Within each file can you identify the most negative review?*
- 4. Within each file can you identify the most positive review?*

SO LITTLE TIME!



LEARN MORE



trinker / sentimentr

Watch 11 Star 164 Fork 30

Code Issues 6 Pull requests 1 Projects 0 Wik Insights

Branch: master sentimentr / README.md

Find file Copy path

trinker added augmented to readme ef98fu5 3 days ago


1 contributor

938 lines (789 sloc) 48.1 KB

Raw Blame History

sentimentr

repo status: Active build: passing coverage: 44% DOI: 10.5281/zenodo.222103 downloads: 1687/month



sentimentr is designed to quickly calculate text polarity sentiment at the sentence level and optionally aggregate by rows or grouping variable(s).

sentimentr is a response to my own needs with sentiment detection that were not addressed by the current R tools. My own `polarity` function in the `qdap` package is slower on larger data sets. It is a dictionary lookup approach that tries to incorporate weighting for valence shifters (negation and amplifiers/deamplifiers). Matthew Jockers created the [syuzhet](#) package that utilizes dictionary lookups for the Bing, NRC, and Afinn methods as well as a custom dictionary. He also utilizes a wrapper for the [Stanford coreNLP](#) which uses much more sophisticated analysis. Jocker's dictionary methods are fast but are more prone to error in the case of valence shifters. Jocker's [addressed these critiques](#) explaining that the method is good with regard to analyzing general sentiment in a piece of literature. He points to the accuracy of the Stanford detection as well. In my own work I need better accuracy than a simple dictionary lookup; something that considers valence shifters yet optimizes speed which the Stanford's parser does not. This leads to a trade off of speed vs. accuracy. Simply, sentimentr attempts to balance accuracy and speed.

Table of Contents

- [Why sentimentr](#)
- [Functions](#)
- [The Function](#)

WHAT TO REMEMBER



FUNCTIONS TO REMEMBER

Operator/Function	Description
<code>tidytext::sentiments</code>	Data frame containing common sentiment words established by the lexicons Bing, NRC, and AFINN
<code>tidytext::get_sentiments</code>	Get word list and sentiment for a specific lexicon
<code>sentimentr::get_sentences</code>	Break an observation up into discrete sentences
<code>sentimentr::sentiment_by</code>	Compute sentiment (along with capturing context via valence shifters) at a specified aggregated level (i.e. chapter, document).
<code>sentimentr::uncombine</code>	Disaggregate sentiment to the observation level.