Ads project1

Cindy Xu UNI:cx2199

2/3/2018

Section 1: Check and install needed packages. Load the libraries and functions.

```
packages.used<-c("ggplot2","dplyr","tibble","tidyr","stringr", "tidytext","to
picmodels","wordcloud",
                 "ggridges",'igraph','ggraph','sentimentr',"devtools",
                 'exploratory',"ldatuning",'CTM','purrr','stm','corpus'
                 ,'tm','quanteda')
# check packages that need to be installed.
packages.needed <- setdiff(packages.used, intersect(installed.packages()[,1],
packages.used))

# install additional packages
if(length(packages.needed) > 0) {
  install.packages(packages.needed, dependencies = TRUE, repos = 'http://cran
.us.r-project.org')
}

#devtools::install_github("exploratory-io/exploratory_func")

library(ggplot2)
library(dplyr)
library(tibble)
library(tidyr)
library(stringr)
library(tidytext)
library(topicmodels)
library(RColorBrewer)
library(wordcloud)
library(ggridges)
library(igraph)
library(ggraph)
library(sentimentr)
library(syuzhet)
library(broom)
library(urltools)
library(exploratory)
library(ldatuning)
```

```
library(purrr)
library(CTM)
library(stm)
library(corpus)
library(tm)
library(quanteda)
source("../libs/multiplot.R")
```

Section 2: Read in the data

The following code assumes that the dataset `spooky.csv` lives in a `data` folder (and that we are inside a `doc` folder).

Step 1: Using spooky

```
spooky<-read.csv('../data/spooky.csv',as.is=T)
```

An overview of the data structure and content

Let's first remind ourselves of the structure of the data.

```
dim<-dim(spooky)
dim

## [1] 19579     3

head(spooky)

##        id
## 1 id26305
## 2 id17569
## 3 id11008
## 4 id27763
## 5 id12958
## 6 id22965
##
text
## 1
This process, however, afforded me no means of ascertaining the dimensions of
my dungeon; as I might make its circuit, and return to the point whence I set
out, without being aware of the fact; so perfectly uniform seemed the wall.
## 2
It never once occurred to me that the fumbling might be a mere mistake.
## 3
In his left hand was a gold snuff box, from which, as he capered down the hil
l, cutting all manner of fantastic steps, he took snuff incessantly with an a
ir of the greatest possible self satisfaction.
## 4
How lovely is spring As we looked from Windsor Terrace on the sixteen fertile
counties spread beneath, speckled by happy cottages and wealthier towns, all
```

```
looked as in former years, heart cheering and fair.
## 5
Finding nothing else, not even gold, the Superintendent abandoned his attempt
s; but a perplexed look occasionally steals over his countenance as he sits t
hinking at his desk.
## 6 A youth passed in solitude, my best years spent under your gentle and fe
minine fosterage, has so refined the groundwork of my character that I cannot
overcome an intense distaste to the usual brutality exercised on board ship:
I have never believed it to be necessary, and when I heard of a mariner equal
ly noted for his kindliness of heart and the respect and obedience paid to hi
m by his crew, I felt myself peculiarly fortunate in being able to secure his
services.
##   author
## 1    EAP
## 2    HPL
## 3    EAP
## 4    MWS
## 5    HPL
## 6    MWS
```

```
summary(spooky)
```

```
##       id                  text                author
##  Length:19579       Length:19579       Length:19579
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
```

```
sum(is.na(spooky))
```

```
## [1] 0
```

```
spooky$author<-as.factor(spooky$author)
unique(spooky$author)
```

```
## [1] EAP HPL MWS
## Levels: EAP HPL MWS
```

When we look into spooky data set, it is a 19579 rows and 3 columns dataset. Each row correspoding a unique id number, an excerpt of texts, and author name. Addtionally, there are no missing values. There are three authors, Like `HPL` is Lovecraft, `MWS` is Shelly, and `EAP` is Poe.

Step 2: Data Processing

1: Punctuation – typical sentence structure. Clauses they have. Number of commas or semicolons.

```
str_count(spooky,',')
```

```
## [1] 19578 57798 19578
```

```
str_count(spooky,';' )
```

```
## [1]    0 5159    0
```

Poe used commas 19578 times, Lovecraft used commas 57798 times, Shelly used commas 19578 times. Poe used semicolons 0 times, Lovecraft used semicolons 5159 times, Shelly used semicolons 0 times.

2: He/she

```
str_count(spooky,'He')
```

```
## [1]    0 1251    0
```

```
str_count(spooky,'he')
```

```
## [1]     0 61490     0
```

```
str_count(spooky,'She')
```

```
## [1]   0 320   0
```

```
str_count(spooky,'she')
```

```
## [1]    0 2028    0
```

Poe and Shelly did not use he/she, Lovecraft used he more than she. I would say Lovecraft may have more people in her story.

Step 3: Data Cleaning

1: Drop all punctuation and transform all words into lower case.

```
spooky_wrd<-unnest_tokens(spooky,word,text)
head(spooky_wrd)
```

```
##          id author     word
## 1   id26305    EAP     this
## 1.1 id26305    EAP  process
## 1.2 id26305    EAP  however
## 1.3 id26305    EAP afforded
## 1.4 id26305    EAP       me
## 1.5 id26305    EAP       no
```

2: Bi-grams, n-grams

If we wanna get relationships between words, we use n-grams. So far we've considered words as individual units, and considered their relationships to sentiments or to documents. However, many interesting text analyses are based on the relationships between words, whether examining which words tend to follow others immediately. we'll explore some of the methods tidytext offers for calculating and visualizing relationships between words in your text dataset. This includes the token = "ngrams" argument, which tokenizes by pairs of adjacent words rather than by individual ones.

Tokenizing by n-gram

We've been using the unnest_tokens function to tokenize by word, or sometimes by sentence, which is useful for the kinds of sentiment and frequency analyses we've been doing so far. But we can also use the function to tokenize into consecutive sequences of words, called n-grams. By seeing how often word X is followed by word Y, we can then build a model of the relationships between them. 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"

```
# Make a table with one word per row and remove `stop words` (i.e. the common
words).
bigrams<-unnest_tokens(spooky,bigram, text, token = "ngrams", n = 2)
head(bigrams)

##        id author       bigram
## 1 id00001    MWS    idris was
## 2 id00001    MWS     was well
## 3 id00001    MWS well content
## 4 id00001    MWS content with
## 5 id00001    MWS    with this
## 6 id00001    MWS this resolve

bigrams_HPL<-unnest_tokens(spooky[spooky$author=='HPL',],bigram, text, token
= "ngrams", n = 2)
head(bigrams_HPL)

##        id author       bigram
## 1 id00002    HPL        i was
## 2 id00002    HPL    was faint
## 3 id00002    HPL   faint even
## 4 id00002    HPL even fainter
## 5 id00002    HPL fainter than
## 6 id00002    HPL     than the
```

```
bigrams_MWS<-unnest_tokens(spooky[spooky$author=='MWS',],bigram, text, token
= "ngrams", n = 2)
head(bigrams_MWS)

##        id author       bigram
## 1 id00001    MWS    idris was
## 2 id00001    MWS     was well
## 3 id00001    MWS well content
## 4 id00001    MWS content with
## 5 id00001    MWS    with this
## 6 id00001    MWS this resolve

bigrams_EAP<-unnest_tokens(spooky[spooky$author=='EAP',],bigram, text, token
= "ngrams", n = 2)
head(bigrams_EAP)

##        id author    bigram
## 1 id00003    EAP above all
## 2 id00003    EAP     all i
## 3 id00003    EAP    i burn
## 4 id00003    EAP   burn to
## 5 id00003    EAP   to know
## 6 id00003    EAP  know the
```

This data structure is still a variation of the tidy text format. It is structured as one-token-per-row (with extra metadata, such as author, still preserved), but each token now represents a bigram. From these tables, we can get phrases for each author. Then we can analyze the relationship between these phrases.

(1): Counting and filtering n-grams

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

```
bigrams_count<-count(bigrams,bigram,sort=T)
head(bigrams_count)

## # A tibble: 6 x 2
##   bigram       n
##   <chr>    <int>
## 1 of the    5581
## 2 in the    2743
## 3 to the    1847
## 4 and the   1343
## 5 it was    1037
## 6 from the  1036

bigrams_EAP_count<-count(bigrams_EAP,bigram,sort=T)
head(bigrams_EAP_count)
```

```
## # A tibble: 6 x 2
##   bigram      n
##   <chr>   <int>
## 1 of the   2877
## 2 in the   1237
## 3 to the    823
## 4 of a      530
## 5 to be     431
## 6 and the   428
```

```
bigrams_MWS_count<-count(bigrams_MWS,bigram,sort=T)
head(bigrams_MWS_count)
```

```
## # A tibble: 6 x 2
##   bigram      n
##   <chr>   <int>
## 1 of the   1217
## 2 in the    605
## 3 to the    534
## 4 and the   412
## 5 of my     359
## 6 on the    356
```

```
bigrams_HPL_count<-count(bigrams_HPL,bigram,sort=T)
head(bigrams_HPL_count)
```

```
## # A tibble: 6 x 2
##   bigram       n
##   <chr>    <int>
## 1 of the    1487
## 2 in the     901
## 3 and the    503
## 4 to the     490
## 5 on the     428
## 6 from the   350
```

As one might expect, a lot of the most common bigrams are pairs of common (uninteresting) words, such as of the and in the: what we call “stop-words” . 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 where either is a stop-word. From these tables, we can learn the frequency of these common words. Like for different authors, ‘of the’, ‘in the’, etc, these phrases appear frequently. We can treat these phrases as redundancies. We’ re more interested in the text without these phrases, then we can focus on main materials and it is easy for us to analyze the differences between each author.

```
bigrams_separated<-separate(bigrams,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=T)
head(bigram_counts)
```

```
## # A tibble: 6 x 3
##   word1  word2           n
##   <chr>  <chr>       <int>
## 1 lord   raymond        27
## 2 fellow creatures      22
## 3 ha     ha             22
## 4 main   compartment    21
## 5 madame lalande        20
## 6 chess  player         18
```

```
bigrams_HPL_separated<-separate(bigrams_HPL,bigram,c("word1", "word2"),sep =
" ")
bigrams_HPL_filtered<-bigrams_HPL_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_HPL_counts<-bigrams_HPL_filtered %>%
  count(word1,word2,sort=T)
head(bigram_HPL_counts)
```

```
## # A tibble: 6 x 3
##   word1   word2        n
##   <chr>   <chr>    <int>
## 1 heh     heh         17
## 2 shunned house       16
## 3 tempest mountain    14
## 4 brown   jenkin      13
## 5 herbert west        13
## 6 yog     sothoth     12
```

```
bigrams_MWS_separated<-separate(bigrams_MWS,bigram,c("word1", "word2"),sep =
" ")
bigrams_MWS_filtered<-bigrams_MWS_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_MWS_counts<-bigrams_MWS_filtered %>%
  count(word1,word2,sort=T)
head(bigram_MWS_counts)
```

```
## # A tibble: 6 x 3
##   word1   word2           n
```

```
##   <chr>   <chr>       <int>
## 1 lord    raymond        27
## 2 fellow  creatures      22
## 3 native  country        14
## 4 natural philosophy     10
## 5 poor    girl           10
## 6 human   race            9
```

```r
bigrams_EAP_separated<-separate(bigrams_EAP,bigram,c("word1", "word2"),sep =
" ")
bigrams_EAP_filtered<-bigrams_EAP_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_EAP_counts<-bigrams_EAP_filtered %>%
  count(word1,word2,sort=T)
head(bigram_EAP_counts)
```

```
## # A tibble: 6 x 3
##   word1  word2           n
##   <chr>  <chr>       <int>
## 1 ha     ha             22
## 2 main   compartment    21
## 3 madame lalande        20
## 4 chess  player         18
## 5 left   arm            13
## 6 tea    pot            13
```

After we get rid of stop words. We can see that these phrases are the most common pairs in spooky data set.

In other analyses, we may want to work with the recombined words. tidyr's unite() function is the inverse of separate(), and lets us recombine the columns into one. Thus, "separate/filter/count/unite" let us find the most common bigrams not containing stop-words.

```r
bigrams_united<-bigrams_filtered %>%
  unite(bigram, word1, word2, sep = " ")
head(bigrams_united)
```

```
##        id author            bigram
## 1 id00002    HPL hateful modernity
## 2 id00002    HPL     accursed city
## 3 id00003    EAP       dark valley
## 4 id00004    EAP unusual clearness
## 5 id00004    EAP  necessarily lost
## 6 id00004    EAP         lost sight
```

(2): Analyzing bigrams

A bigram can also be treated as a term in a document in the same way that we treated individual words. For example, we can look at the tf-idf of bigrams across spooky dataset.

TF stands for term frequency or how often a word appears in a text and it is what is studied above in the word cloud. IDF stands for inverse document frequncy, and it is a way to pay more attention to words that are rare within the entire set of text data that is more sophisticated than simply removing stop words. Multiplying these two values together calculates a term's tf-idf, which is the frequency of a term adjusted for how rarely it is used. We'll use tf-idf as a heuristic index to indicate how frequently a certain author uses a word relative to the frequency that all the authors use the word. Therefore we will find words that are characteristic for a specific author, a good thing to have if we are interested in solving the author identification problem.

```
#get rid of stop words
spooky_wrd <- anti_join(spooky_wrd, stop_words, by = "word")
frequency<-count(spooky_wrd,author,word)
tf_idf<-bind_tf_idf(frequency,word,author,n)
head(tf_idf)
```

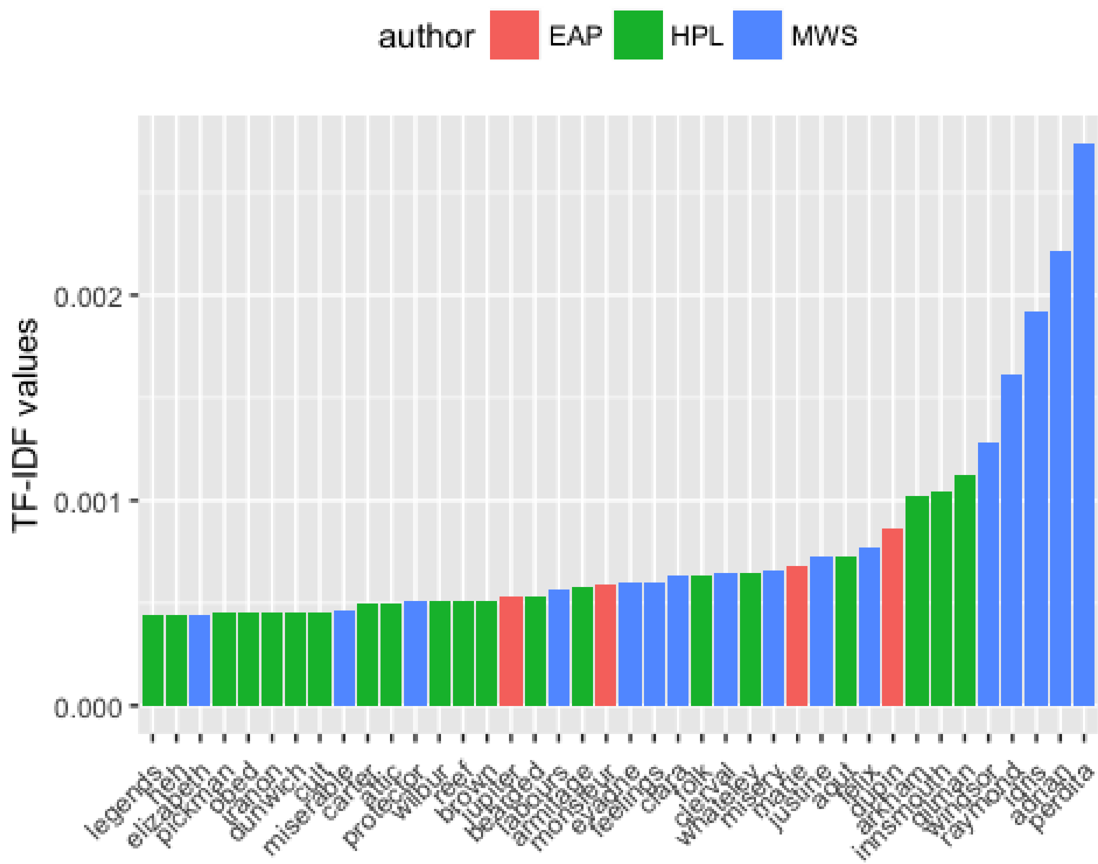
```
## # A tibble: 6 x 6
##   author word        n        tf   idf    tf_idf
##   <chr>  <chr>   <int>     <dbl> <dbl>     <dbl>
## 1 EAP    à           9 0.000124  1.10  0.000136
## 2 EAP    a.m         3 0.0000412 0.405 0.0000167
## 3 EAP    aaem        1 0.0000137 1.10  0.0000151
## 4 EAP    ab          1 0.0000137 1.10  0.0000151
## 5 EAP    aback       2 0.0000275 1.10  0.0000302
## 6 EAP    abandon     7 0.0000961 0     0
```

```
tail(tf_idf)
```

```
## # A tibble: 6 x 6
##   author word         n        tf   idf     tf_idf
##   <chr>  <chr>    <int>     <dbl> <dbl>      <dbl>
## 1 MWS    youth's      1 0.0000160 0.405 0.00000649
## 2 MWS    youthful    10 0.000160  0     0
## 3 MWS    youths       2 0.0000320 0.405 0.0000130
## 4 MWS    zaimi        2 0.0000320 1.10  0.0000352
## 5 MWS    zeal         7 0.000112  0     0
## 6 MWS    zest         3 0.0000480 0     0
```

```
tf_idf<-arrange(tf_idf,desc(tf_idf))
tf_idf<-mutate(tf_idf, word = factor(word,levels= rev(unique(word))))

# Grab the top fourty tf_idf scores in all the words
tf_idf_40<- top_n(tf_idf,40,tf_idf)

ggplot(tf_idf_40) +
```

```
  geom_col(aes(word,tf_idf,fill = author)) +
  labs(x = NULL, y = "TF-IDF values") +
  theme(legend.position ="top",axis.text.x= element_text(angle=45,hjust=1,vju
st=0.9))
```

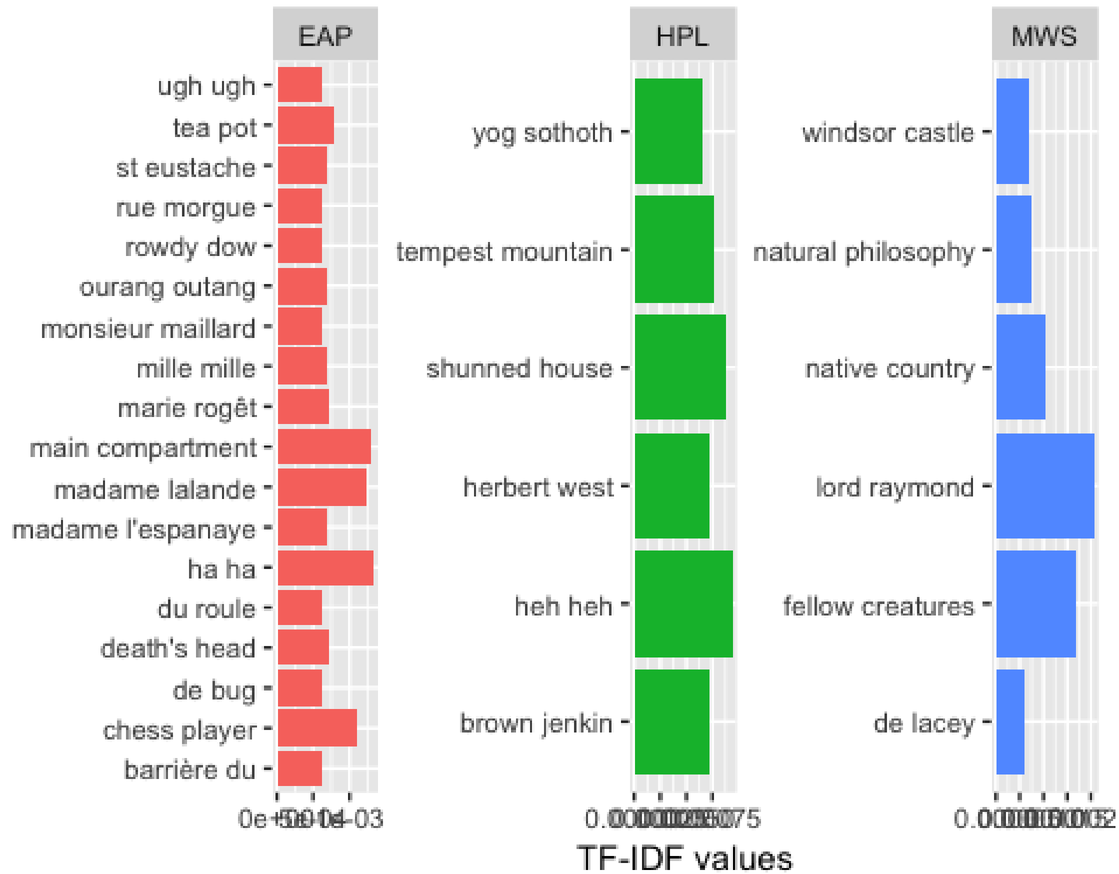


Note that in the above, many of the words recognized by their tf-idf scores are names. This makes sense – if we see text referencing Raymond, Idris, or Perdita, we know almost for sure that MWS is the author. But some non-names stand out. EAP often uses “monsieur” and “jupiter” while HPL uses the words “bearded” and “attic” more frequently than the others. We can also look at the most characteristic terms per author.

Then we can look at the tf-idf of bigrams across spooky datasts.

```
bigram_tf_idf<-bigrams_united %>%
  count(author,bigram) %>%
  bind_tf_idf(bigram,author,n) %>%
  arrange(desc(tf_idf))
bigram_tf_idf_30<-head(bigram_tf_idf,30)
ggplot(bigram_tf_idf_30) +
  geom_col(aes(bigram,tf_idf, fill = author)) +
  labs(x = NULL, y = "bigram_tf_idf") +
  theme(legend.position = "none") +
  facet_wrap(~ author,ncol =3,scales="free")+
```

```
  coord_flip() +
  labs(y = "TF-IDF values")
```



From this plot, I would say EAP prefer more colorful phrases in order to avoid repeating in his story. HLP and MWS, phrases in their stories are in less variation.

There are advantages and disadvantages to examining the tf-idf of bigrams rather than individual words. Pairs of consecutive words might capture structure that isn't present when one is just counting single words, and may provide context that makes tokens more understandable. However, the per-bigram counts are also sparser: a typical two-word pair is rarer than either of its component words.

Section 3: Sentiment Analysis

Step1: Word level

1: Using bigrams to provide context in sentiment analysis

Our sentiment analysis approach in simply counted the appearance of positive or negative words, according to a reference lexicon. One of the problems with this approach is that a word's context can matter nearly as much as its presence. For example, the words "happy" and “like” will be counted as positive, even in a sentence like “I’m not happy and I don’t like it!”

Now that we have the data organized into bigrams, it’s easy to tell how often words are preceded by a word like “not”:

```
bigrams_separated %>%
  filter(word1 == "not") %>%
  count(word1, word2, sort = TRUE)
```

```
## # A tibble: 946 x 3
##    word1 word2     n
##    <chr> <chr> <int>
##  1 not   to      139
##  2 not   be      131
##  3 not   the     103
##  4 not   a        88
##  5 not   have     72
##  6 not   only     66
##  7 not   in       57
##  8 not   so       57
##  9 not   even     44
## 10 not   been     37
## # ... with 936 more rows
```

By performing sentiment analysis on the bigram data, we can examine how often sentiment-associated words are preceded by “not” or other negating words. We could use this to ignore or even reverse their contribution to the sentiment score.

Let’s use the AFINN lexicon for sentiment analysis, which you may recall gives a numeric sentiment score for each word, with positive or negative numbers indicating the direction of the sentiment.

```
AFINN<-get_sentiments("afinn")
```

We can then examine the most frequent words that were preceded by “not” and were associated with a sentiment.

```
not_words<-bigrams_separated %>%
  filter(word1 == "not") %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word2, score, sort = TRUE) %>%
  ungroup()
not_words
```

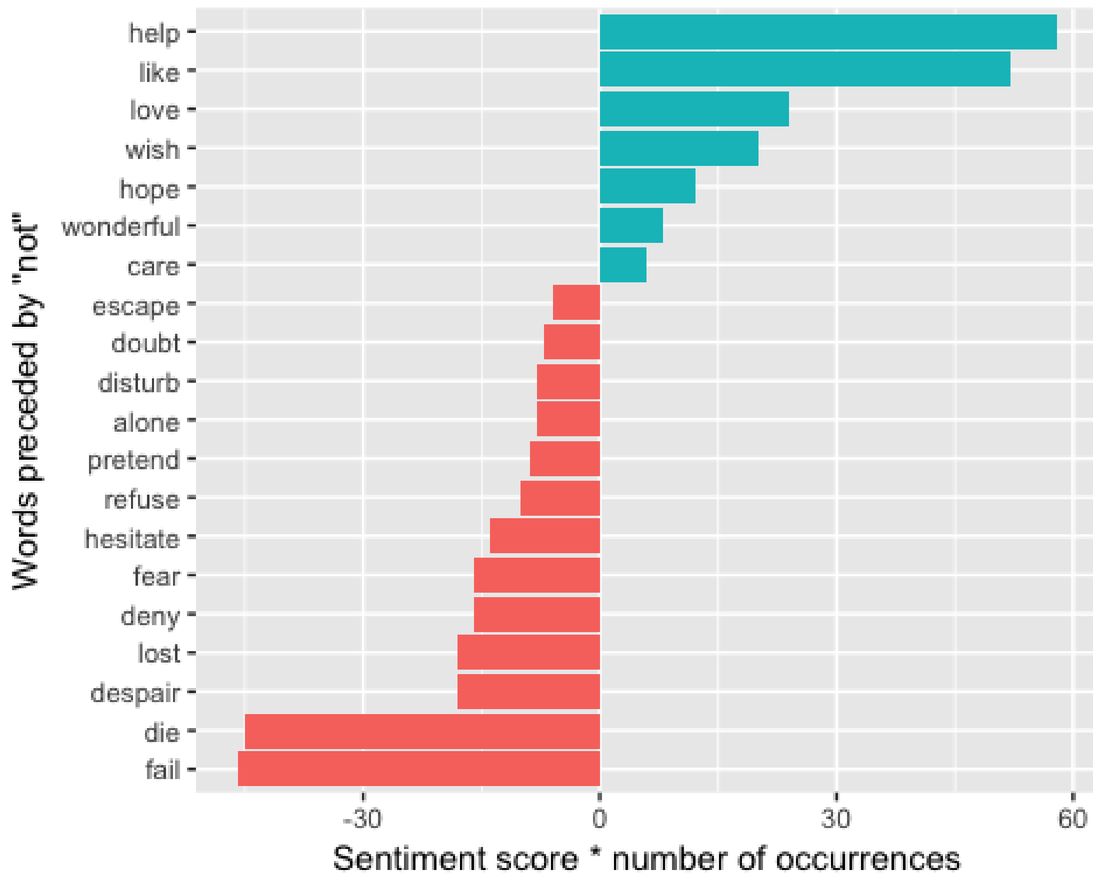
```
## # A tibble: 158 x 3
##    word2   score     n
##    <chr>   <int> <int>
##  1 help        2    29
##  2 like        2    26
##  3 fail       -2    23
```

```
## 4 wish 1 20
## 5 die -3 15
## 6 pretend -1 9
## 7 deny -2 8
## 8 fear -2 8
## 9 love 3 8
## 10 doubt -1 7
## # ... with 148 more rows
```

For example, the most common sentiment-associated word to follow “not” was “help”, which would normally have a (positive) score of 2.

It’s worth asking which words contributed the most in the “wrong” direction. To compute that, we can multiply their score by the number of times they appear (so that a word with a score of +3 occurring 10 times has as much impact as a word with a sentiment score of +1 occurring 30 times). We visualize the result with a bar plot.

```
not_words %>%
  mutate(contribution = n * score) %>%
  arrange(desc(abs(contribution))) %>%
  head(20) %>%
  mutate(word2 = reorder(word2, contribution)) %>%
  ggplot(aes(word2, n * score, fill = n * score > 0)) +
  geom_col(show.legend = FALSE) +
  xlab("Words preceded by \"not\"") +
  ylab("Sentiment score * number of occurrences") +
  coord_flip()
```



The 20 words preceded by 'not' that had the greatest contribution to sentiment scores, in either a positive or negative direction. The bigrams "not help" and "not like" were overwhelmingly the largest causes of misidentification, making the text seem much more positive than it is. But we can see phrases like "not fail" and "not die" sometimes suggest text is more negative than it is.

"Not" isn't the only term that provides some context for the following word. We could pick four common words (or more) that negate the subsequent term, and use the same joining and counting approach to examine all of them at once.

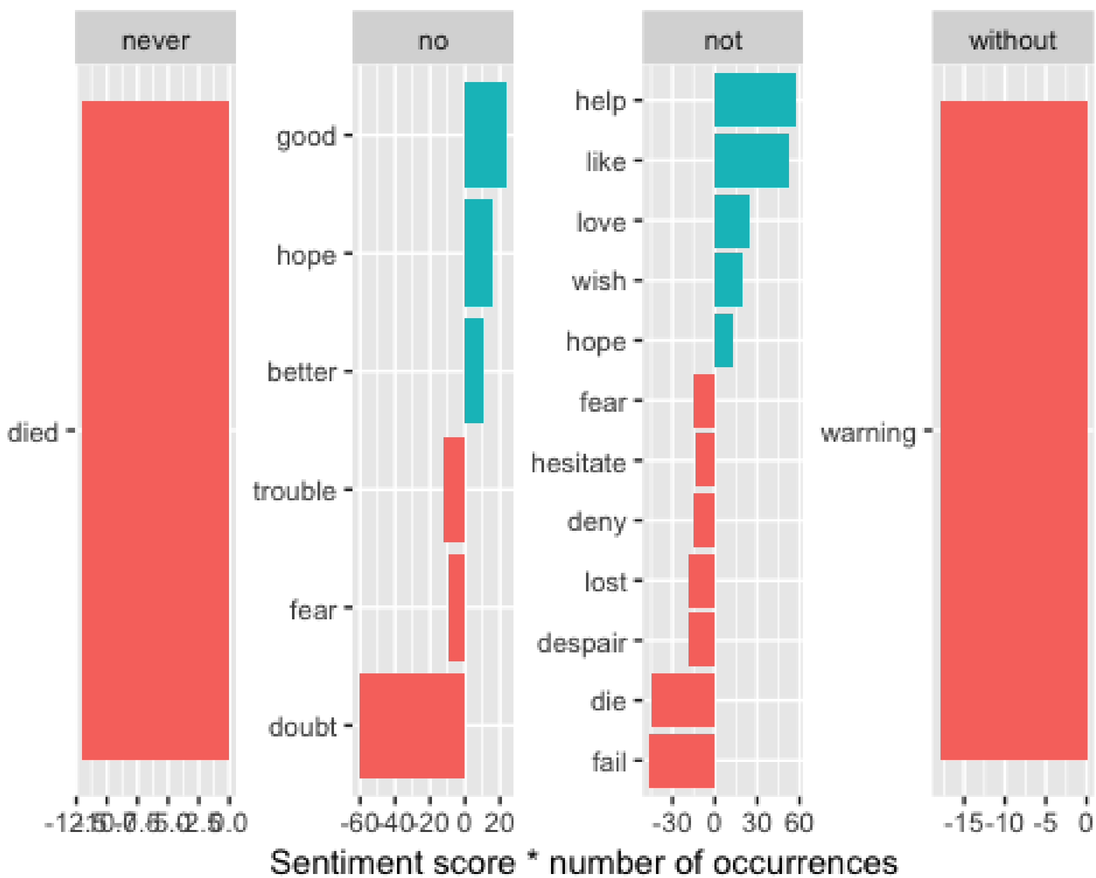
```
negation_words <- c("not", "no", "never", "without")

negated_words<-bigrams_separated %>%
  filter(word1 %in% negation_words) %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word1, word2, score, sort = TRUE)
head(negated_words)

## # A tibble: 6 x 4
##   word1 word2 score     n
##   <chr> <chr> <int> <int>
```

```
## 1 no    doubt    -1    60
## 2 not   help      2    29
## 3 not   like      2    26
## 4 not   fail     -2    23
## 5 not   wish      1    20
## 6 not   die      -3    15
```

```
negated_words$word1<-as.factor(negated_words$word1)
unique(negated_words$word1)
```

```
## [1] no      not     never   without
## Levels: never no not without
```

```
negated_words %>%
  mutate(contribution = n * score) %>%
  arrange(desc(abs(contribution))) %>%
  head(20) %>%
  mutate(word2 = reorder(word2, contribution)) %>%
  ggplot(aes(word2, n * score, fill = n * score > 0)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "Sentiment score * number of occurrences")+
  facet_wrap(~ word1,ncol =4,scales="free")+
  coord_flip()
```



"not doubt" and "not help" are the two most common examples, we can also see pairings such as "no hope" and "never forget." We can see how these negate phrases make text more positive or more negative than it is.

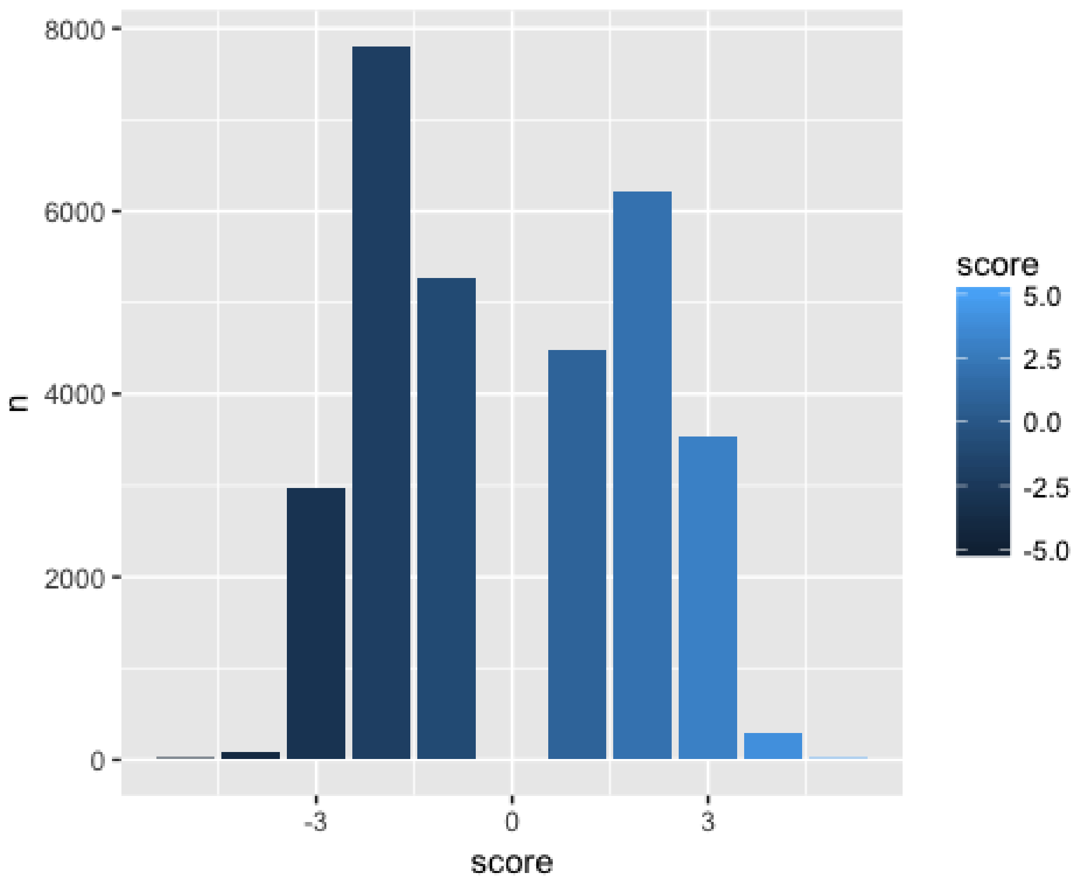
2: Compare Afinn, Bing with NRC

```
# Keep words that have been classified within the NRC lexicon.
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

sentiments_afinn <- inner_join(spooky_wrd, get_sentiments('afinn'), by = "wor
d")
head(sentiments_afinn)

##        id author      word score
## 1 id26305    EAP        no    -1
## 2 id26305    EAP perfectly     3
## 3 id17569    HPL   mistake    -2
## 4 id11008    EAP   cutting    -1
## 5 id11008    EAP fantastic     4
## 6 id11008    EAP  greatest     3

count(sentiments_afinn, score)

## # A tibble: 10 x 2
##    score     n
##    <int> <int>
##  1    -5    12
##  2    -4    98
##  3    -3  2961
##  4    -2  7810
##  5    -1  5280
##  6     1  4479
##  7     2  6220
##  8     3  3529
##  9     4   291
## 10     5     9
```

```r
count(sentiments_afinn, author, score)
```

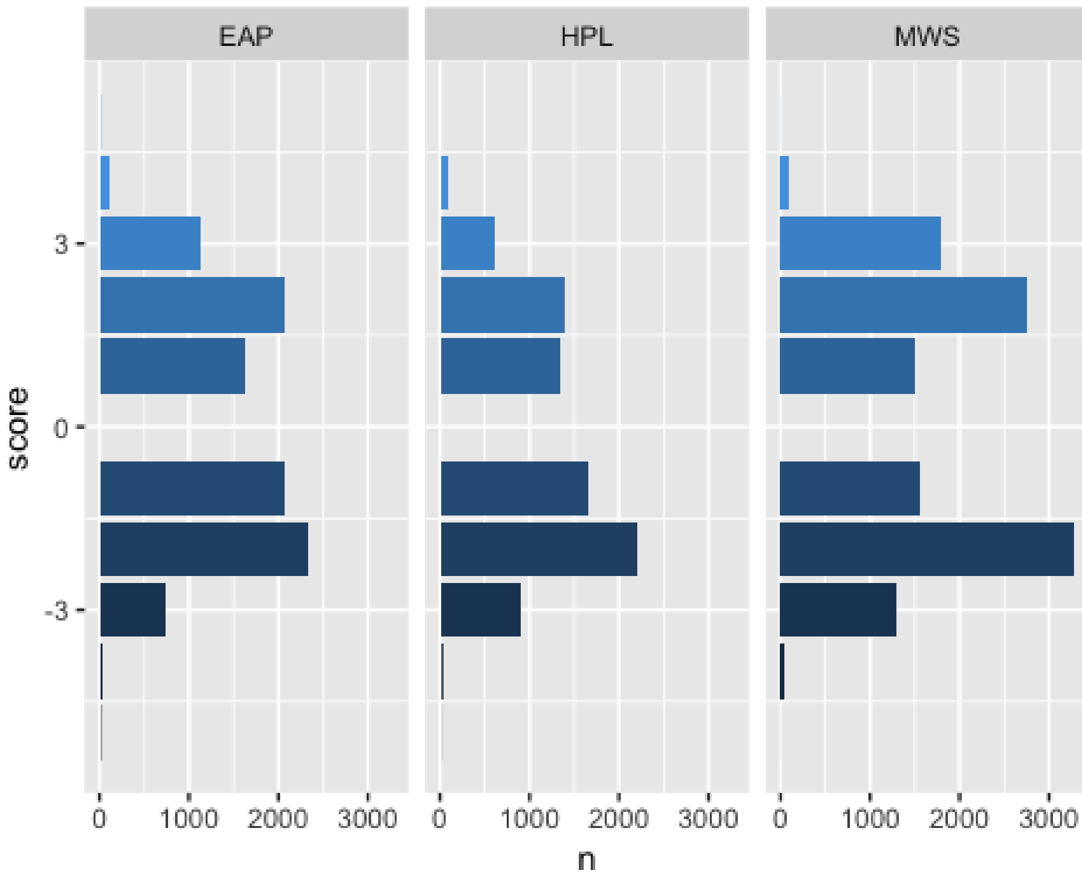
```
## # A tibble: 29 x 3
##    author score     n
##    <chr>  <int> <int>
##  1 EAP       -5     9
##  2 EAP       -4    28
##  3 EAP       -3   751
##  4 EAP       -2  2321
##  5 EAP       -1  2072
##  6 EAP        1  1639
##  7 EAP        2  2071
##  8 EAP        3  1121
##  9 EAP        4   113
## 10 EAP        5     7
## # ... with 19 more rows
```

```r
ggplot(count(sentiments_afinn, score)) +
  geom_col(aes(score, n, fill = score))
```




```r
ggplot(count(sentiments_afinn, author, score)) +
  geom_col(aes(score, n, fill = score)) +
  facet_wrap(~ author) +
```

```r
  coord_flip() +
  theme(legend.position = "none")
```




```r
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
```

```r
sentiments_bing<- inner_join(spooky_wrd, get_sentiments('bing'), by = "word")
head(sentiments_bing)
```

```
##         id author        word sentiment
## 1 id26305    EAP     dungeon  negative
## 2 id26305    EAP   perfectly  positive
## 3 id17569    HPL     mistake  negative
## 4 id11008    EAP        gold  positive
## 5 id11008    EAP   fantastic  positive
## 6 id11008    EAP incessantly  negative
```

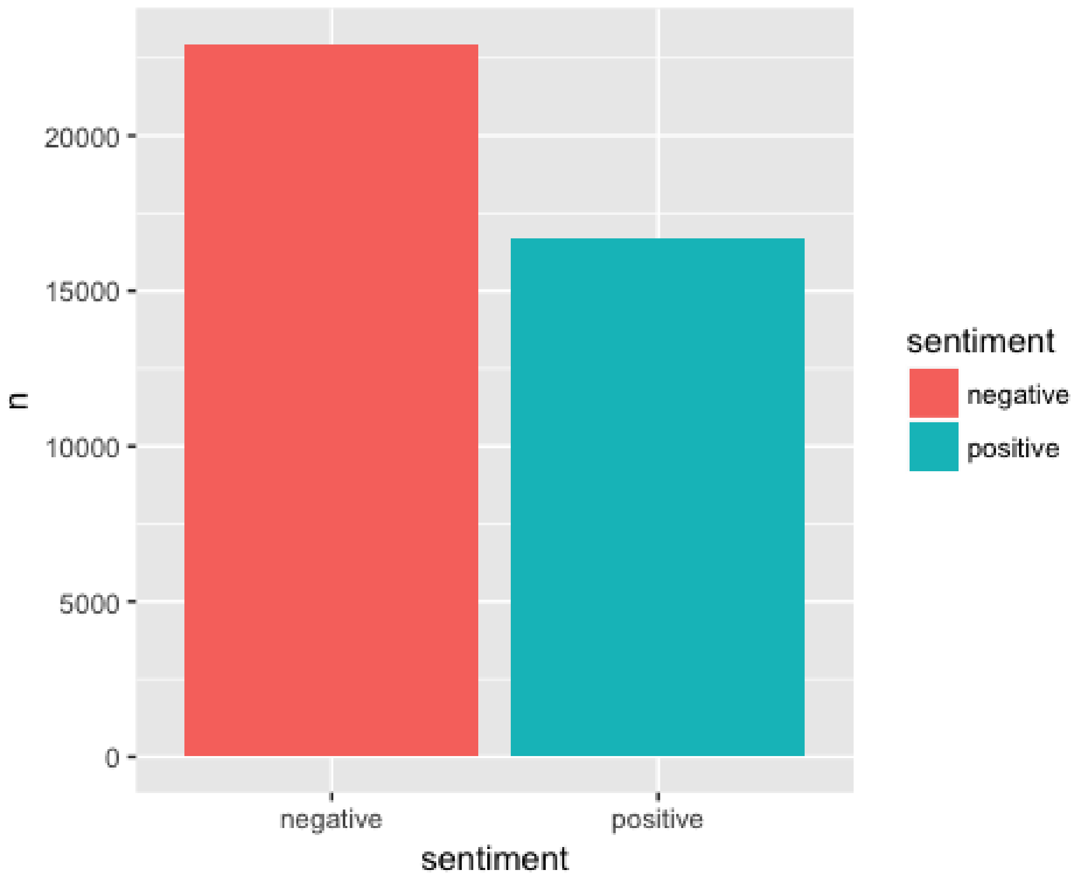
```
count(sentiments_bing,sentiment)
```

```
## # A tibble: 2 x 2
##   sentiment     n
##   <chr>     <int>
## 1 negative  22958
## 2 positive  16674
```

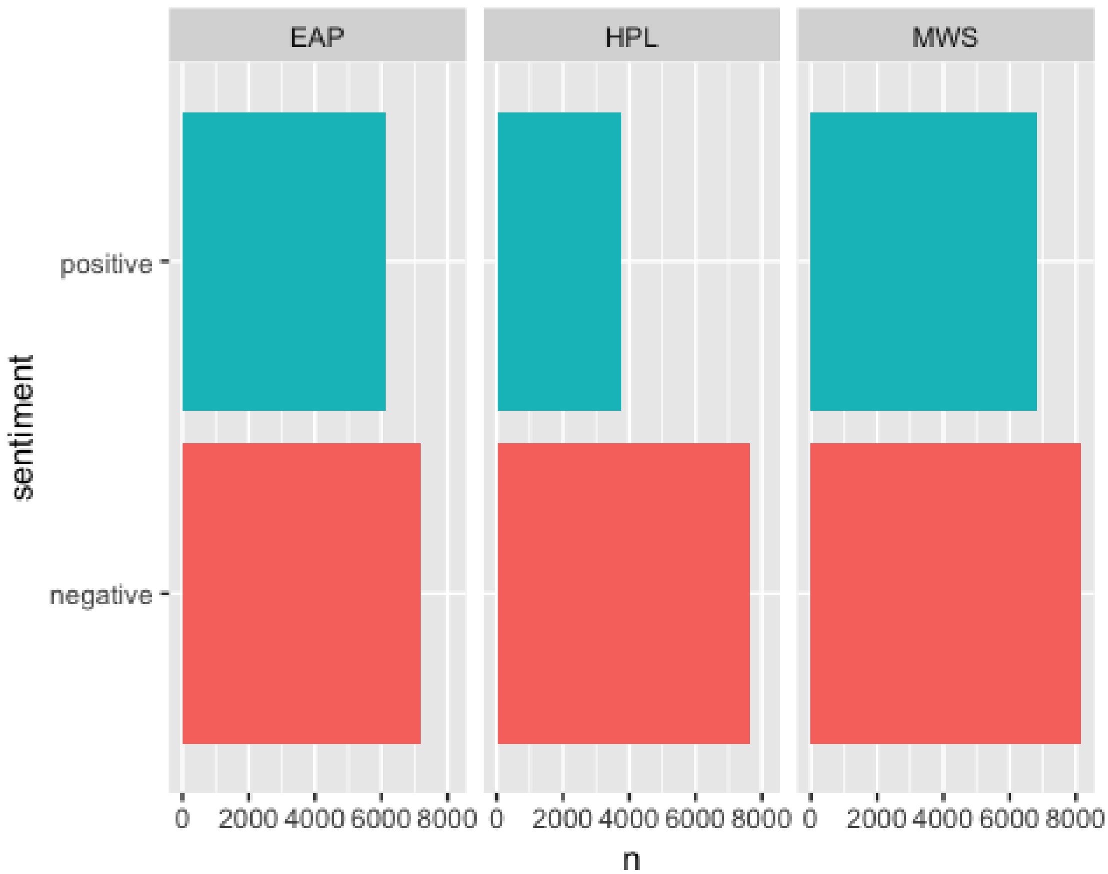
```
count(sentiments_bing,author,sentiment)
```

```
## # A tibble: 6 x 3
##   author sentiment     n
##   <chr>  <chr>     <int>
## 1 EAP    negative   7203
## 2 EAP    positive   6144
## 3 HPL    negative   7605
## 4 HPL    positive   3731
## 5 MWS    negative   8150
## 6 MWS    positive   6799
```

```
ggplot(count(sentiments_bing,sentiment)) +
  geom_col(aes(sentiment, n, fill = sentiment))
```




```r
ggplot(count(sentiments_bing, author, sentiment)) +
  geom_col(aes(sentiment, n, fill = sentiment)) +
  facet_wrap(~ author) +
  coord_flip() +
  theme(legend.position = "none")
```



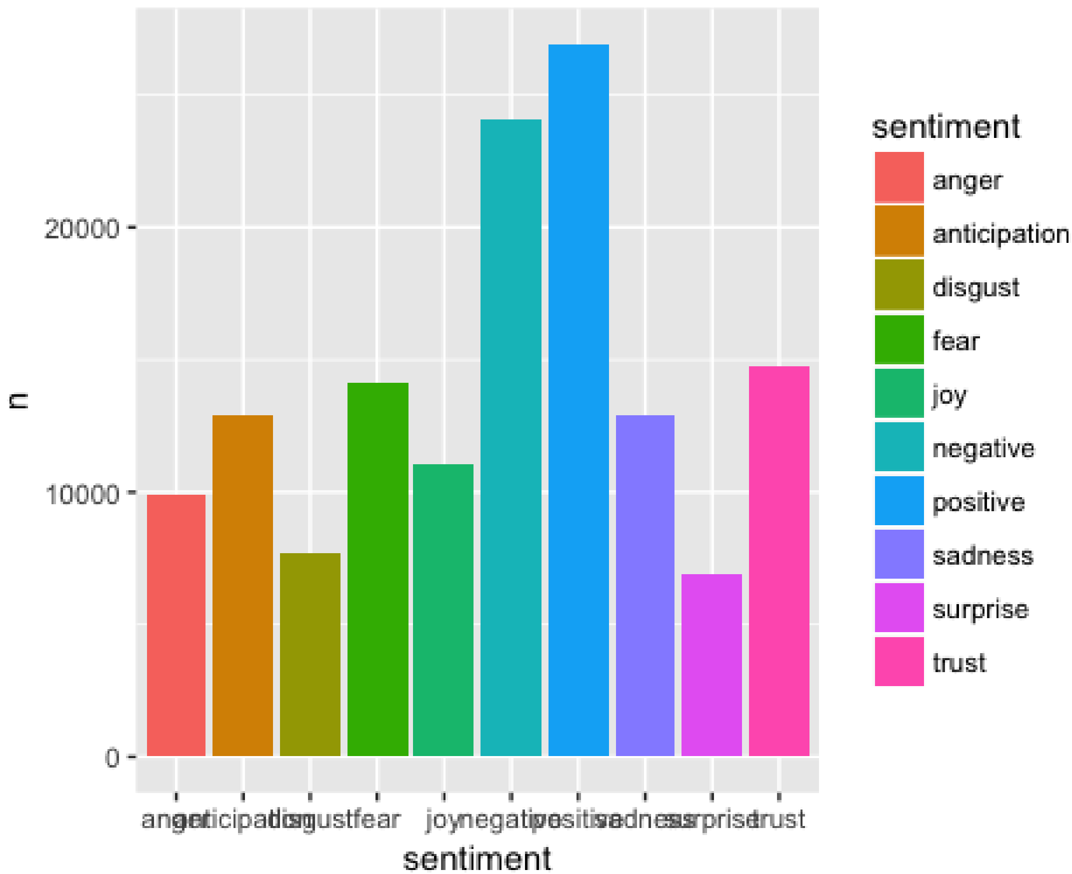

```
get_sentiments('nrc')

## # 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

sentiments <- inner_join(spooky_wrd, get_sentiments('nrc'), by = "word")

count(sentiments, sentiment)

## # A tibble: 10 x 2
##    sentiment        n
##    <chr>        <int>
```

```
## 1 anger         9869
## 2 anticipation 12912
## 3 disgust       7731
## 4 fear         14096
## 5 joy          11077
## 6 negative     24084
## 7 positive     26934
## 8 sadness      12896
## 9 surprise      6903
## 10 trust       14777
```

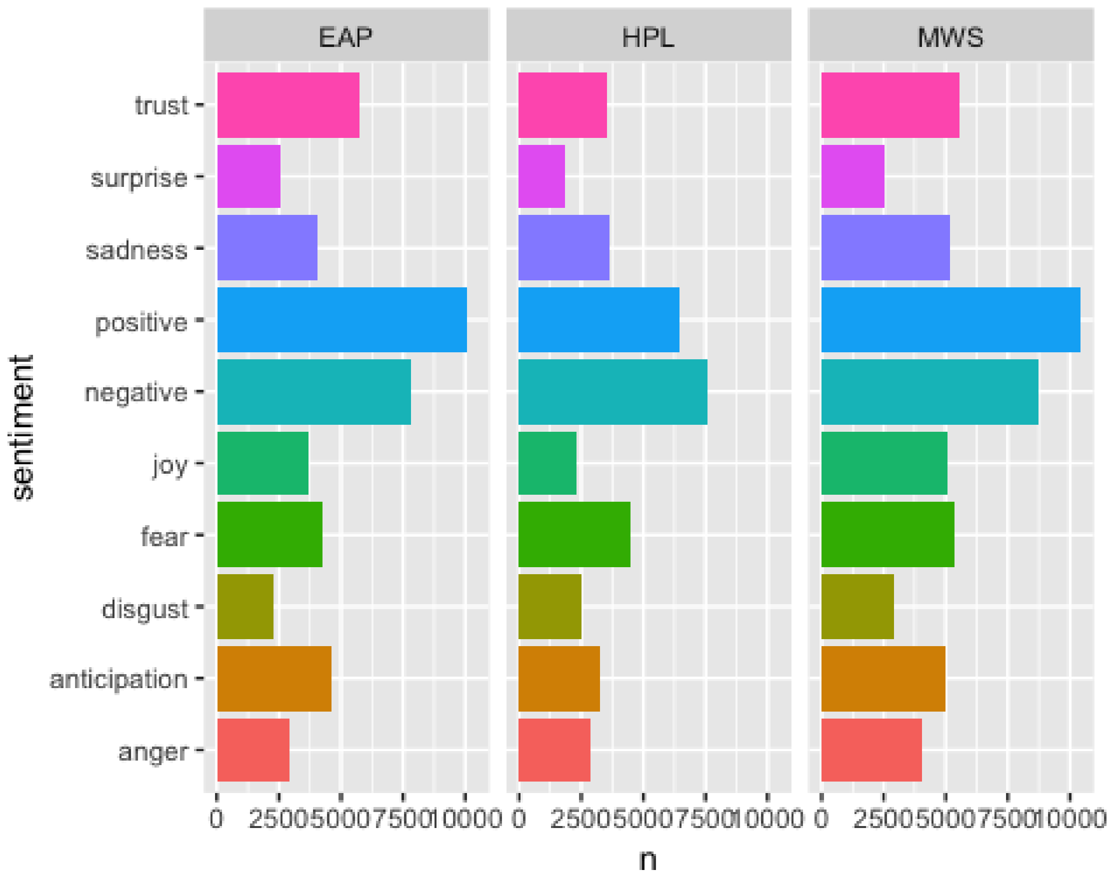
```r
count(sentiments, author, sentiment)
```

```
## # A tibble: 30 x 3
##    author sentiment        n
##    <chr>  <chr>        <int>
##  1 EAP    anger         2962
##  2 EAP    anticipation  4656
##  3 EAP    disgust       2273
##  4 EAP    fear          4287
##  5 EAP    joy           3652
##  6 EAP    negative      7833
##  7 EAP    positive     10083
##  8 EAP    sadness       4045
##  9 EAP    surprise      2538
## 10 EAP    trust         5739
## # ... with 20 more rows
```

```r
ggplot(count(sentiments, sentiment)) +
  geom_col(aes(sentiment, n, fill = sentiment))
```




```r
ggplot(count(sentiments, author, sentiment)) +
  geom_col(aes(sentiment, n, fill = sentiment)) +
  facet_wrap(~ author) +
  coord_flip() +
  theme(legend.position = "none")
```



Based on afinn, we see the whole text contains more negative words, like score=-5. And Shelly uses more extreme words than others.

Based on bing, we learn three authors are all negative and Shelly has more negative emotions than other two.

Based on nrc, we get the number of words for different emotions for whole text and for different authors. In this case, it seems like, if we put emotions into more categories, positive is more than other emotions. Then we can compare three different methods for sentiment analysis.

We then use spread() so that we have negative and positive sentiment in separate columns, and lastly calculate a net sentiment (positive - negative). And Now we can plot these sentiment scores across the plot trajectory of each author. Notice that we are plotting against the index on the x-axis that keeps track of text.

```
head(spooky_wrd)
```

```
##            id author     word
## 1   id26305    EAP     this
## 1.1 id26305    EAP  process
## 1.2 id26305    EAP  however
```

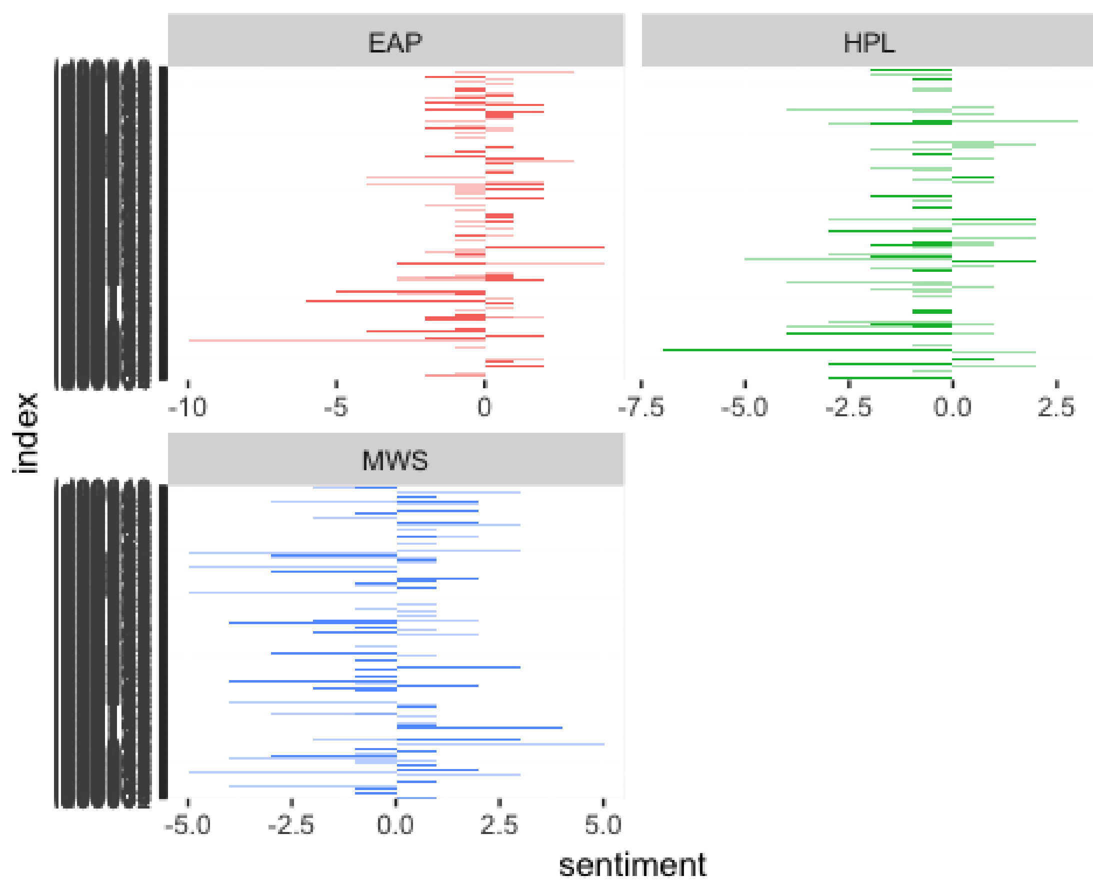
```
## 1.3 id26305     EAP afforded
## 1.4 id26305     EAP        me
## 1.5 id26305     EAP        no
```

```
specialsentiment<-spooky_wrd%>%inner_join(get_sentiments("bing")) %>%
  count(index=id, author, sentiment)%>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

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

```
specialsentiment_300<-head(specialsentiment,300)
```

```
ggplot(specialsentiment_300, aes(index , sentiment, fill = author)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~author, ncol = 2, scales = "free_x")+
  coord_flip()
```



We can see how the plot of each author changes toward more positive or negative sentiment over the trajectory of the story. Like, MWS is more emotional than others with high variation for trajectory of the story.

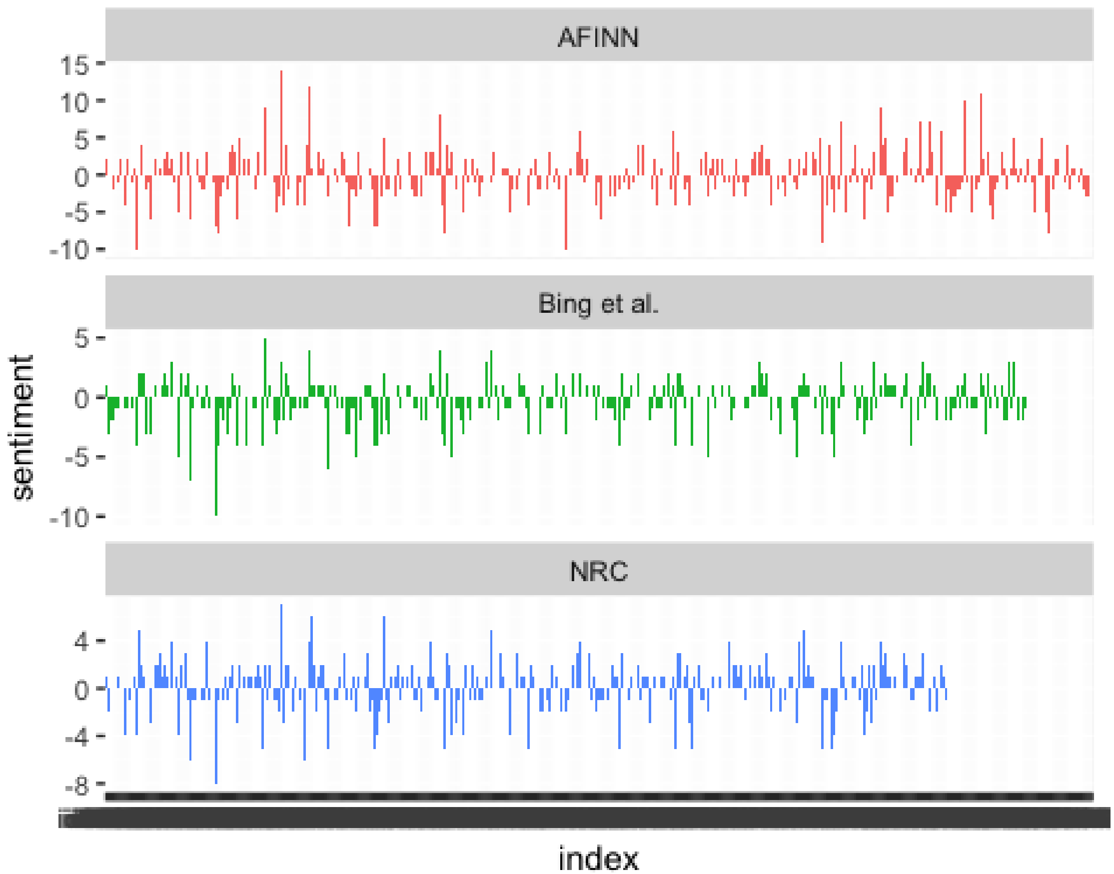
3: Comparing the three sentiment dictionaries

With several options for sentiment lexicons, you might want some more information on which one is appropriate for your purposes. Let's use all three sentiment lexicons and examine how the sentiment changes across the author.

```
afinn_method<-spooky_wrd%>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(index =id) %>%
  summarise(sentiment = sum(score)) %>%
  mutate(method = "AFINN")

## Joining, by = "word"

afinn_300<-head(afinn_method,300)
bing_method<-spooky_wrd%>%
  inner_join(get_sentiments("bing")) %>%
  mutate(method = "Bing et al.") %>%
  count(method, index =id, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)

## Joining, by = "word"

bing_300<-head(bing_method,300)
nrc_method<-spooky_wrd%>%
  inner_join(get_sentiments("nrc") %>%
               filter(sentiment %in% c("positive","negative"))%>%
               mutate(method = "NRC")) %>%
  count(method, index =id, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)

## Joining, by = "word"

nrc_300<-head(nrc_method,300)
```

We now have an estimate of the net sentiment (positive - negative) in each chunk of the text for each sentiment lexicon. Let's bind them together and visualize them.

```
bind_rows(afinn_300,
          bing_300, nrc_300) %>%
  ggplot(aes(index, sentiment, fill = method)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y")
```



The three different lexicons for calculating sentiment give results that are different in an absolute sense but have similar relative trajectories through the author. We see similar dips and peaks in sentiment at about the same places in the author, but the absolute values are significantly different. The AFINN lexicon gives the largest absolute values, with high positive values. The lexicon from Bing et al. has lower absolute values and seems to label larger blocks of contiguous positive or negative text. The NRC results are shifted higher relative to the other two, labeling the text more positively, but detects similar relative changes in the text. Sentiment appears to find longer stretches of similar text, but all three agree roughly on the overall trends in the sentiment through a narrative arc.

Step2: Do sentiment analysis at sentense level

```
spooky<-read.csv('../data/spooky.csv',as.is=T)
spooky.sentense<-spooky%>%
  mutate(sentiment = get_sentiment(text))

## Warning in split_warn(text.var, "sentiment_by", ...): Each time
## `sentiment_by` is run it has to do sentence boundary disambiguation when
## a raw `character` vector is passed to `text.var`. This may be costly of
## time and memory. It is highly recommended that the user first runs the raw
## `character` vector through the `get_sentences` function.
```

```r
count(spooky.sentense, sentiment)
```

```
## # A tibble: 8,704 x 2
##    sentiment     n
##        <dbl> <int>
##  1     -2.42     1
##  2     -2.15     1
##  3     -1.92     1
##  4     -1.66     1
##  5     -1.65     1
##  6     -1.59     1
##  7     -1.56     1
##  8     -1.53     1
##  9     -1.48     1
## 10     -1.48     1
## # ... with 8,694 more rows
```

```r
count(spooky.sentense, author, sentiment)
```

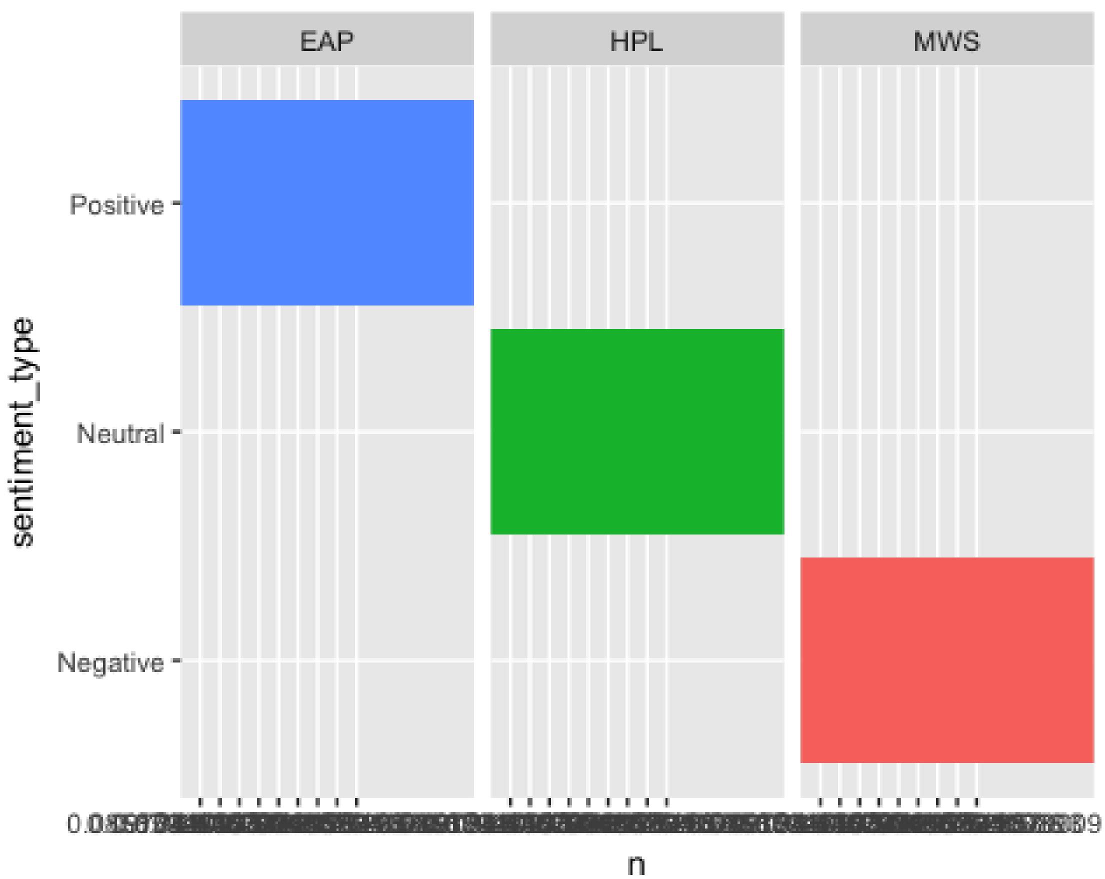
```
## # A tibble: 11,370 x 3
##    author sentiment     n
##    <chr>      <dbl> <int>
##  1 EAP        -2.42     1
##  2 EAP        -2.15     1
##  3 EAP        -1.66     1
##  4 EAP        -1.56     1
##  5 EAP        -1.45     1
##  6 EAP        -1.42     1
##  7 EAP        -1.40     1
##  8 EAP        -1.35     1
##  9 EAP        -1.33     1
## 10 EAP        -1.32     1
## # ... with 11,360 more rows
```

```r
spooky.sentense.data<-spooky.sentense %>%
  mutate(sentiment_type = if_else(sentiment >0, "Positive", if_else(sentiment
<0, "Negative", "Neutral")))%>%
  select(sentiment, sentiment_type,text,author)
order.spooky.sentense<-spooky.sentense.data[order(spooky.sentense.data$sentim
ent),]
positive.rate<-sum(spooky.sentense.data$sentiment_type=='Positive')/nrow(spoo
ky.sentense.data)
positive.rate
```

```
## [1] 0.4305634
```

```r
count.whole.table<-count(spooky.sentense.data%>%group_by(author))
interger.EAP<-as.integer(count.whole.table[count.whole.table$author=='EAP',]$
n)
interger.HPL<-as.integer(count.whole.table[count.whole.table$author=='HPL',]$
n)
```

```
interger.MWS<-as.integer(count.whole.table[count.whole.table$author=='MWS',]$
n)
count.table<-count(spooky.sentense.data%>%group_by(sentiment_type, author))
frequency.EAP<-count.table[count.table$author=='EAP',]$n/
  as.integer(count.whole.table[count.whole.table$author=='EAP',]$n)
frequency.HPL<-count.table[count.table$author=='HPL',]$n/
  as.integer(count.whole.table[count.whole.table$author=='HPL',]$n)
frequency.MWS<-count.table[count.table$author=='MWS',]$n/
  as.integer(count.whole.table[count.whole.table$author=='MWS',]$n)
n<-c(frequency.MWS,frequency.HPL,frequency.EAP)
author<-c('MWS','MWS','MWS','HPL','HPL','HPL','EAP','EAP','EAP')
sentiment_type<-c('Negative','Negative','Negative','Neutral','Neutral','Neutr
al',
                  'Positive','Positive','Positive')
frequency.table<-as.data.frame(cbind(sentiment_type,author,n))
ggplot(frequency.table)+geom_col(aes(sentiment_type, n, fill = sentiment_type
)) +
  facet_wrap(~ author) +
  coord_flip() +
  theme(legend.position = "none")
```



Proportion of sentences are ‘postive’ is 43% in the whole text, and for each author, Poe has more positive sentenses, Lovecraft has more neutral sentences and Shelly has more negative sentenses.

Section 4: Topic Models

We use the `topicmodels` package for this analysis. Since the `topicmodels` package doesn’t use the `tidytext` framework, we first convert our `spooky_wrd` dataframe into a document term matrix (DTM) matrix using `tidytext` tools.

```
# Counts how many times each word appears in each sentence
sent_wrd_freqs <- count(spooky_wrd, id, word)
head(sent_wrd_freqs)
```

```
## # A tibble: 6 x 3
##   id      word        n
##   <chr>   <chr>   <int>
## 1 id00001 content     1
## 2 id00001 idris       1
## 3 id00001 mine        1
## 4 id00001 resolve     1
## 5 id00002 accursed    1
## 6 id00002 city        1
```

```
spooky_wrd <- anti_join(spooky_wrd, stop_words, by = "word")
# Creates a DTM matrix
spooky_wrd_tm <- cast_dtm(sent_wrd_freqs, id, word, n)
spooky_wrd_tm
```

```
## <<DocumentTermMatrix (documents: 19467, terms: 24941)>>
## Non-/sparse entries: 193944/485332503
## Sparsity           : 100%
## Maximal term length: 19
## Weighting          : term frequency (tf)
```

```
length(unique(spooky_wrd$id))
```

```
## [1] 19467
```

```
length(unique(spooky_wrd$word))
```

```
## [1] 24941
```

The matrix `spooky_wrd_tm` is a sparse matrix with 19467 rows, corresponding to the 19467 ids (or originally, sentences) in the `spooky_wrd` dataframe, and 24941 columns corresponding to the total number of unique words in the `spooky_wrd` dataframe. So each row of `spooky_wrd_tm` corresponds to one of the original sentences. The value of the matrix at a certain position is then the number of occurences of that word (determined by the column) in this specific sentence (determined by the row). Since most sentence/word pairings don’t occur, the matrix is sparse meaning there are many zeros.

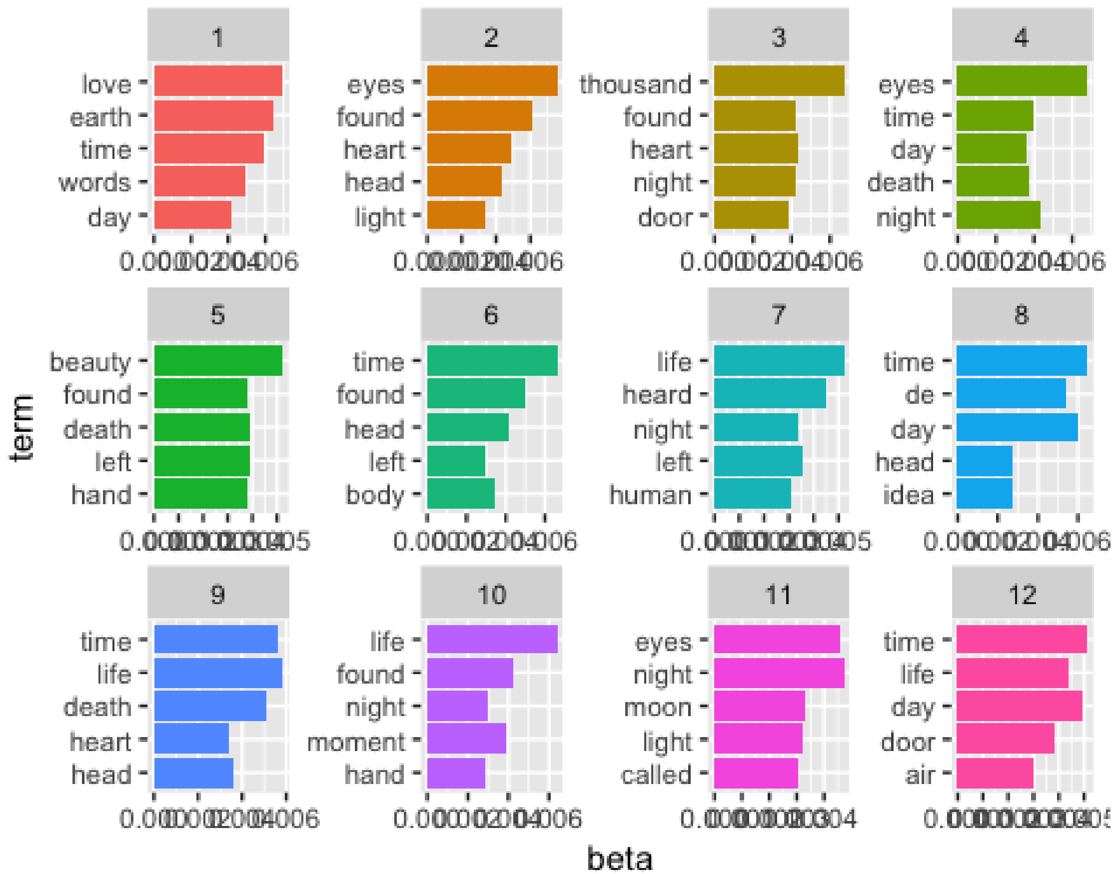
step1: Determine how many topics to use

Pick 6, 12 # of topics and to see if there are any dupilicate topics or not.

```
spooky_wrd_lda    <- LDA(spooky_wrd_tm, k = 12, control = list(seed = 1234))
spooky_wrd_topics <- tidy(spooky_wrd_lda, matrix = "beta")
spooky_wrd_topics
```

```
## # A tibble: 299,292 x 3
##    topic term         beta
##    <int> <chr>       <dbl>
##  1     1 content 0.000144
##  2     2 content 0.000230
##  3     3 content 0.000285
##  4     4 content 0.0000162
##  5     5 content 0.000213
##  6     6 content 0.0000758
##  7     7 content 0.0000736
##  8     8 content 0.000323
##  9     9 content 0.0000845
## 10    10 content 0.0000201
## # ... with 299,282 more rows
```

```
spooky_wrd_topics_5 <- ungroup(top_n(group_by(spooky_wrd_topics, topic), 5, b
eta))
spooky_wrd_topics_5 <- arrange(spooky_wrd_topics_5, topic, -beta)
spooky_wrd_topics_5 <- mutate(spooky_wrd_topics_5, term = reorder(term, beta)
)

ggplot(spooky_wrd_topics_5) +
  geom_col(aes(term, beta, fill = factor(topic)), show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free", ncol = 4) +
  coord_flip()
```

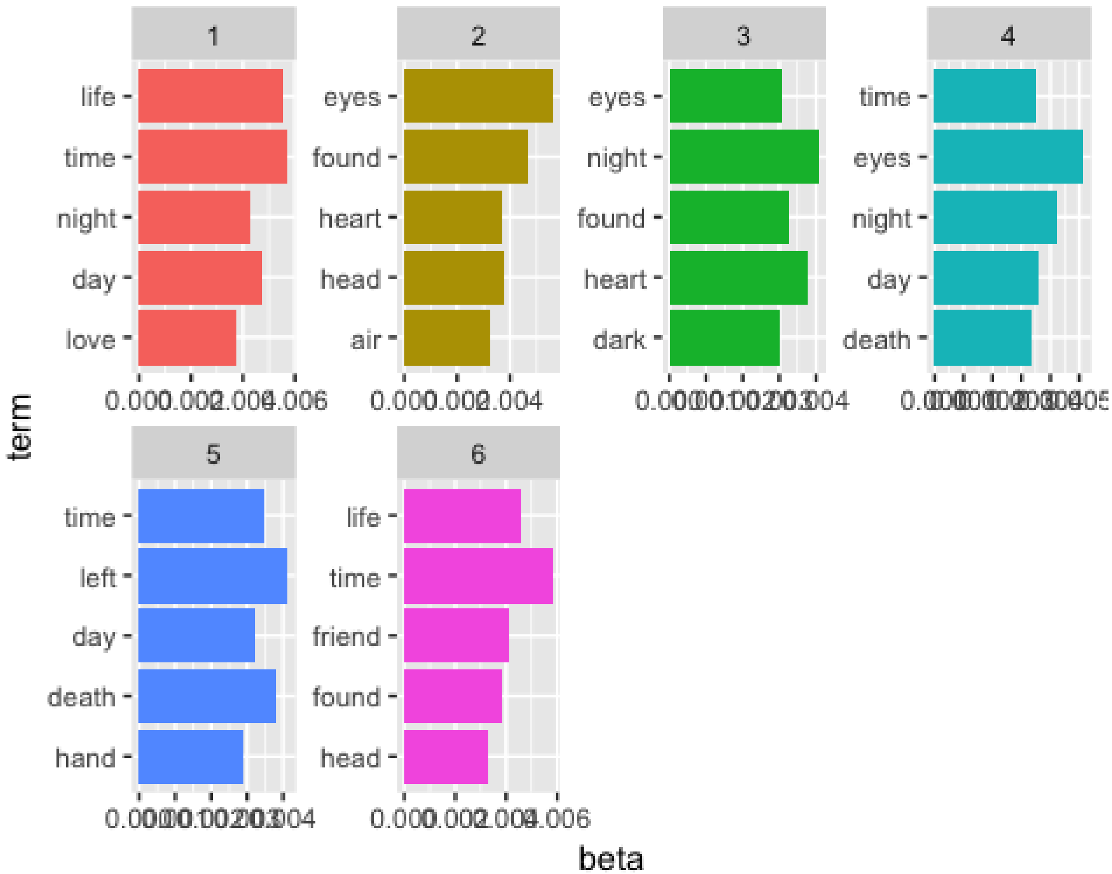



```
spooky_wrd_lda_6<-LDA(spooky_wrd_tm,k=6, control = list(seed = 1234))
spooky_wrd_6_topics <- tidy(spooky_wrd_lda_6, matrix = "beta")
spooky_wrd_6_topics
```

```
## # A tibble: 149,646 x 3
##    topic term         beta
##    <int> <chr>       <dbl>
##  1     1 content 0.000198
##  2     2 content 0.000316
##  3     3 content 0.000359
##  4     4 content 0.0000223
##  5     5 content 0.0000294
##  6     6 content 0.000107
##  7     1 idris   0.000499
##  8     2 idris   0.000562
##  9     3 idris   0.000467
## 10     4 idris   0.000406
## # ... with 149,636 more rows
```

```
spooky_wrd_6_topics_5 <- ungroup(top_n(group_by(spooky_wrd_6_topics, topic),
5, beta))
spooky_wrd_6_topics_5 <- arrange(spooky_wrd_6_topics_5, topic, -beta)
spooky_wrd_6_topics_5 <- mutate(spooky_wrd_6_topics_5, term = reorder(term, b
```

```
eta))
ggplot(spooky_wrd_6_topics_5) +
  geom_col(aes(term, beta, fill = factor(topic)), show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free", ncol = 4) +
  coord_flip()
```



Compare 6, 12 topic, I would suggest 6, because when u pick 12, there are some kind of of duplicated.

In the above, we see that the first topic is characterized by words like "love", "life", and "time" while the third topic includes the word "eyes", and the fifth topic the word "death". Note that the words "life" and "time" appear in many topics. This is the advantage to topic modelling as opposed to clustering when using natural language -- often a word may be likely to appear in documents characterized by multiple topics.

step2: Visualizing author topics

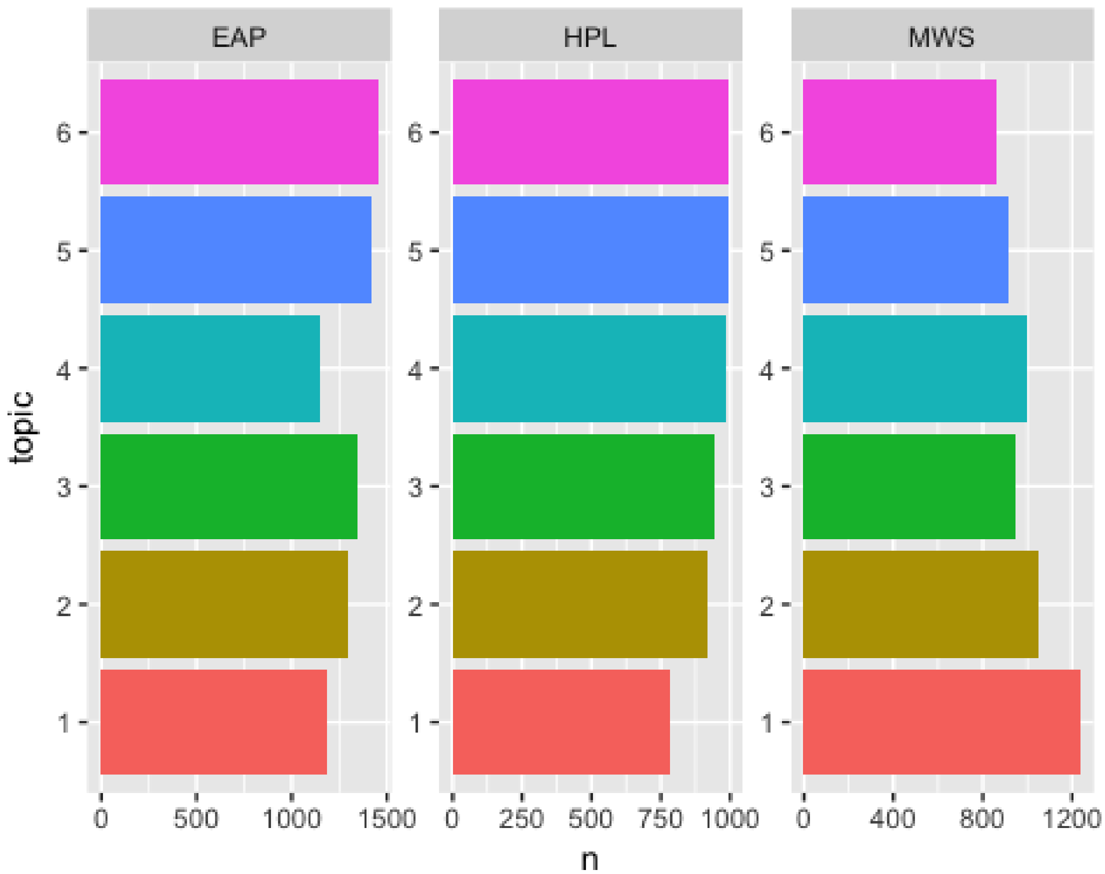
```
spooky_wrd_docs <- tidy(spooky_wrd_lda_6, matrix = "gamma")
head(spooky_wrd_docs)

## # A tibble: 6 x 3
##   document topic gamma
##   <chr>    <int> <dbl>
## 1 id00001      1 0.166
```

```
## 2 id00002      1 0.167
## 3 id00003      1 0.165
## 4 id00004      1 0.166
## 5 id00005      1 0.165
## 6 id00006      1 0.169
```

```r
author_topics <- left_join(spooky_wrd_docs, spooky, by = c("document" = "id")
)
author_topics <- select(author_topics, -text)
author_topics$topic <- as.factor(author_topics$topic)
# Chooses the top topic per sentence
author_topics <- ungroup(top_n(group_by(author_topics, document), 1, gamma))

# Counts the number of sentences represented by each topic per author
author_topics <- ungroup(count(group_by(author_topics, author, topic)))
author_topics
```

```
## # A tibble: 18 x 3
##    author topic     n
##    <chr>  <fct> <int>
##  1 EAP    1      1187
##  2 EAP    2      1298
##  3 EAP    3      1348
##  4 EAP    4      1142
##  5 EAP    5      1414
##  6 EAP    6      1450
##  7 HPL    1       781
##  8 HPL    2       917
##  9 HPL    3       941
## 10 HPL    4       981
## 11 HPL    5       995
## 12 HPL    6       994
## 13 MWS    1      1236
## 14 MWS    2      1047
## 15 MWS    3       947
## 16 MWS    4      1003
## 17 MWS    5       919
## 18 MWS    6       867
```

```r
ggplot(author_topics) +
  geom_col(aes(topic, n, fill = factor(topic)), show.legend = FALSE) +
  facet_wrap(~ author, scales = "free", ncol = 4) +
  coord_flip()
```



From plot, we learn different author focus on different topics. And combine 5 top words for each topic, we can get theme for each author. Like, EAP focuses on topic 6 which is about life, time and friends. HPL focuses on topic 6 and 5 which are about life, time and death. MWS focuses on topic 1 which are about life, time and love. I would say the theme of EAP and HPL are similar, and the theme of MWS is different from others.