Ch.3 Text Mining

1.1 tidytext 패키지의 unnest tokens 함수

```
text <- c('Because I could not stop for Death -',
          'He kindly stopped for me -',
          'The Carriage held but just Ourselves -',
          'and Immortality')

text_df <- tibble(line = 1:4, text = text)
text_df
```

```
## # A tibble: 4 x 2
##    line text
##   <int> <chr>
## 1     1 Because I could not stop for Death -
## 2     2 He kindly stopped for me -
## 3     3 The Carriage held but just Ourselves -
## 4     4 and Immortality
```

```
text_df %>%
  unnest_tokens(word, text) %>%
  head(5)
```

```
## # A tibble: 5 x 2
##    line word
##   <int> <chr>
## 1     1 because
## 2     1 i
## 3     1 could
## 4     1 not
## 5     1 stop
```

줄 번호와 단어가 들어가 있는 text_df를 활용한다.

tidytext 패키지의 unnest_tokens 함수를 사용하면, Token 단위로 쪼갤 수 있다.

즉, 위 예시의 경우에는 text_df에 있는 text 변수를 Token으로 쪼개서 word로 만든다는 것이다.

< unnest_tokens 함수 >

1. Tokenization의 기본값은 single words이다.
2. 위 예시에서 line과 같은 설정되지 않은 다른 변수들은 그대로 유지된다.
3. Punctuation(구두점)은 제거된다.
4. 자동으로 소문자로 만들어서 비교하거나, 합치는데 용이하도록 한다.
    - to_lower = FALSE 함수를 통해 이 기본값을 무시할 수 있다.

1-2. Tidying the works of Jane Austen

```
austen_books() %>%
  select(book) %>%
  unique()
```

```
## # A tibble: 6 x 1
##   book
##   <fct>
## 1 Sense & Sensibility
## 2 Pride & Prejudice
## 3 Mansfield Park
## 4 Emma
## 5 Northanger Abbey
## 6 Persuasion
```

janeaustenr 패키지에서 Jane Austen의 책을 austen_books()로 불러온다.

책들 중에서 unique 한 것들만 추려서 확인하면, 총 6개의 책이 나온다.

```
original_books <- austen_books() %>%
                    group_by(book) %>%
                    mutate(linenumber = row_number(),
                           chapter = cumsum(str_detect(text, regex('^chapter [\\divxlc]',
                                                                   ignore_case = T)))) %>%
                    ungroup()

head(original_books)
```

```
## # A tibble: 6 x 4
##   text                    book                linenumber chapter
##   <chr>                   <fct>                    <int>   <int>
## 1 "SENSE AND SENSIBILITY" Sense & Sensibility          1       0
## 2 ""                      Sense & Sensibility          2       0
## 3 "by Jane Austen"        Sense & Sensibility          3       0
## 4 ""                      Sense & Sensibility          4       0
## 5 "(1811)"                Sense & Sensibility          5       0
## 6 ""                      Sense & Sensibility          6       0
```

austen_books에서 book으로 group_by를 하고, 줄 번호(linenumber)를 만들어준다.

그리고 정규표현식(regex)을 이용하여 chapter들을 구분해주도록 한다.

ungroup 함수는 기존의 austen_books에서 'Groups: book [6]'을 제거해주는 것이다.

이렇게 추려진 original_books 데이터는 아직 tidy text라고 말할 수는 없다.

그 이유는 공백이 있고, 단어 단위로 Tokenization이 되어 있지 않기 때문이다.

```
tidy_books <- original_books %>%
                unnest_tokens(word, text)

tidy_books
```

```
## # A tibble: 725,055 x 4
##    book                linenumber chapter word
##    <fct>                    <int>   <int> <chr>
##  1 Sense & Sensibility          1       0 sense
##  2 Sense & Sensibility          1       0 and
##  3 Sense & Sensibility          1       0 sensibility
##  4 Sense & Sensibility          3       0 by
##  5 Sense & Sensibility          3       0 jane
##  6 Sense & Sensibility          3       0 austen
##  7 Sense & Sensibility          5       0 1811
##  8 Sense & Sensibility         10       1 chapter
##  9 Sense & Sensibility         10       1 1
## 10 Sense & Sensibility         13       1 the
## # ... with 725,045 more rows
```

그리고 나서 unnest_tokens 함수로 tokenization을 수행하도록 한다.
이렇게 되면, text 변수를 제외하고 기존의 열들은 그대로 유지되는 것을 확인할 수 있다.
또한 text 변수는 single word로 구분되며, 공백들은 자동으로 사라지고, 소문자로 변환된다.
이를 통해 나오는 최종 결과는 총 725,055개의 단어라는 것을 알 수 있다.

1-3. Removing stop-words

```
data(stop_words)
tidy_books <- tidy_books %>%
                anti_join(stop_words)

dim(tidy_books)
```

```
## [1] 217609      4
```

data(stop_words)를 불러오면, be동사나 전치사, the, a, an과 같은 단어를 불러올 수 있다.
그리고 나서, tidy_books 데이터에서 anti_join으로 stopwords를 제거해주도록 한다.
이를 통해 기존에 725,055개의 데이터가 217,609로 줄어든 것을 확인할 수 있다.

```
tidy_books %>%
  count(word, sort = T)
```

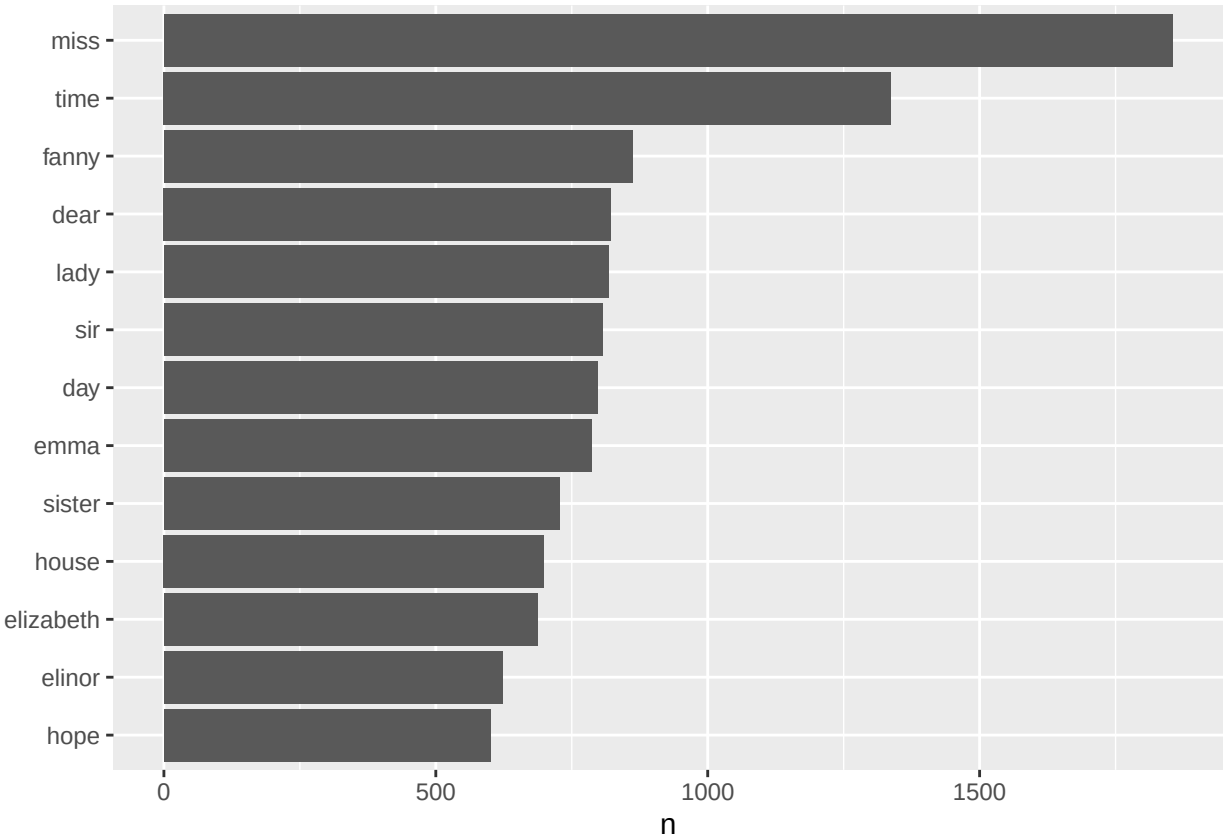
```
## # A tibble: 13,914 x 2
##    word       n
##    <chr>  <int>
##  1 miss    1855
##  2 time    1337
##  3 fanny    862
##  4 dear     822
##  5 lady     817
##  6 sir      806
##  7 day      797
##  8 emma     787
```

```
##  9 sister   727
## 10 house    699
## # ... with 13,904 more rows
```

최종적으로 stopwords를 제거한 단어들 중에서 unique한 단어들의 개수를 확인하도록 한다.
결과를 확인하면, miss, time, fancy 등의 단어가 가장 많이 사용된 것을 볼 수 있다.
전체 unique한 단어는 총 13,914개라는 것도 확인할 수 있다.

1-4. Visualization

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



가장 많이 사용된 단어들 중에서 600번 이상 사용된 단어만 추려서 시각화를 해본다.
단어들의 개수가 많은 것으로 정렬하여 그래프를 그려보도록 한다.

2-1. The gutenbergr package

```
hgwells <- gutenberg_download(c(35, 36, 5230, 159),
                              mirror = "http://mirrors.xmission.com/gutenberg/")
tidy_hgwells <- hgwells %>%
                  unnest_tokens(word, text) %>%
                  anti_join(stop_words)

tidy_hgwells %>%
  count(word, sort = T) %>%
  head(6)
```

```
## # A tibble: 6 x 2
##   word       n
##   <chr>  <int>
## 1 time     461
## 2 people   302
## 3 door     260
## 4 heard    249
## 5 black    232
## 6 stood    229
```

gutenberg_download를 통해서 일부의 책들을 다운 받는다.

그리고 1번 과정에서 했던 것처럼 tokenizations을 수행하고, stopwords를 제거한다.

그 후, unique한 단어들의 빈도 수를 출력해보도록 한다.

이 경우에는 총 11,830개의 단어가 사용되었고, time, people, door 등이 가장 많이 사용됐다.

```
bronte <- gutenberg_download(c(1260, 768, 969, 9182, 767),
                             mirror = "http://mirrors.xmission.com/gutenberg/")
tidy_bronte <- bronte %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)

tidy_bronte %>%
  count(word, sort = T) %>%
  head(6)
```

```
## # A tibble: 6 x 2
##   word      n
##   <chr> <int>
## 1 time   1064
## 2 miss    854
## 3 day     826
## 4 hand    767
## 5 eyes    713
## 6 don't   666
```

gutenberg_download를 통해서 다른 책들을 다운 받고, 위 과정을 똑같이 수행한다.

2-2. Comparison with frequent words

```
frequency <- bind_rows(mutate(tidy_bronte, author = 'Bronte Sisters'),
                       mutate(tidy_hgwells, author = 'H.G. Wells'),
                       mutate(tidy_books, author = 'Jane Austen')) %>%
              mutate(word = str_extract(word, "[a-z']+")) %>%
              count(author, word) %>%
              group_by(author) %>%
              mutate(proportion = n / sum(n)) %>%
              select(-n) %>%
              spread(author, proportion)

head(frequency, 5)
```

```
## # A tibble: 5 x 4
##   word    `Bronte Sisters` `H.G. Wells` `Jane Austen`
##   <chr>              <dbl>        <dbl>         <dbl>
## 1 a             0.0000587    0.0000148    0.00000919
## 2 a'n't        NA            NA            0.00000460
## 3 aback         0.00000391   0.0000148   NA
## 4 abaht         0.00000391  NA           NA
## 5 abandon       0.0000313    0.0000148   NA
```

각 작가들의 단어 빈도 수에 대한 비율을 비교해보려고 한다.

작가와 각 단어들을 가지고 와서, word의 빈도 수를 만들도록 한다.

그리고 나서 작가들 별로 전체 단어에 대한 비율을 계산하여 출력하도록 한다.

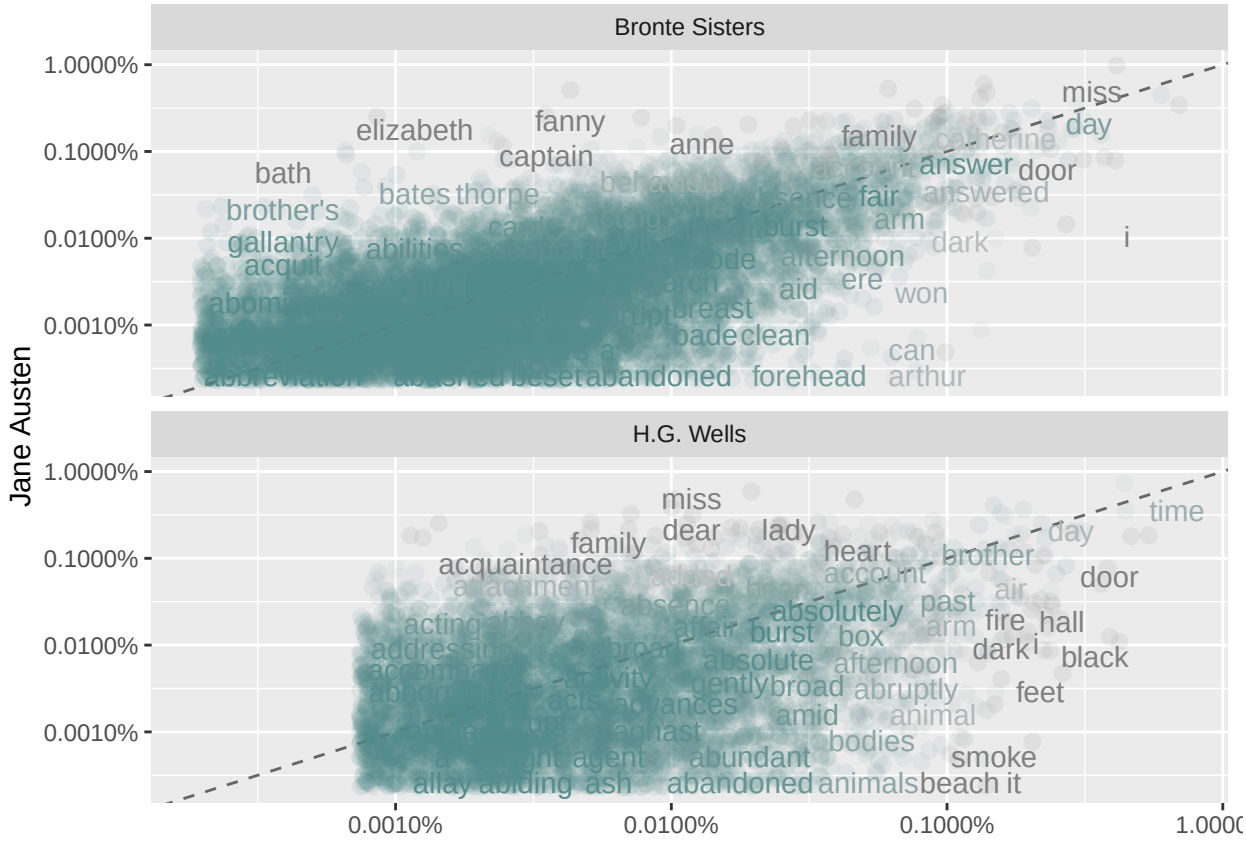
```
frequency <- frequency %>%
              gather(author, proportion, 'Bronte Sisters':'H.G. Wells')

frequency %>%
  arrange(word)
```

```
## # A tibble: 57,252 x 4
##    word    `Jane Austen` author          proportion
##    <chr>           <dbl> <chr>                <dbl>
##  1 a          0.00000919 Bronte Sisters  0.0000587
##  2 a          0.00000919 H.G. Wells      0.0000148
##  3 a'n't      0.00000460 Bronte Sisters NA
##  4 a'n't      0.00000460 H.G. Wells     NA
##  5 aback     NA          Bronte Sisters  0.00000391
##  6 aback     NA          H.G. Wells      0.0000148
##  7 abaht     NA          Bronte Sisters  0.00000391
##  8 abaht     NA          H.G. Wells     NA
##  9 abandon   NA          Bronte Sisters  0.0000313
## 10 abandon   NA          H.G. Wells      0.0000148
## # ... with 57,242 more rows
```

위에서 출력된 결과에서 마지막에 2명의 작가에 대하여 gather를 해주도록 한다.

이를 통해 Jane Austen 작가에 대해 Bronte와 H.G Wells의 단어 비율을 확인할 수 있도록 한다.

```
ggplot(frequency, aes(x = proportion, y = `Jane Austen`,
                      color = abs(`Jane Austen` - proportion))) +
  geom_abline(color = 'gray40', lty = 2) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.3, height = 0.3) +
  geom_text(aes(label = word), check_overlap = T, vjust = 1.5) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  scale_color_gradient(limits = c(0, 0.001), low = 'darkslategray4', high = 'gray75') +
  facet_wrap(~author, nrow = 2) +
  theme(legend.position = 'none') +
  labs(y = 'Jane Austen', x = NULL)
```



Jane Austen 작가에 대해 다른 두 명의 작가들의 단어 빈도수에 대한 시각화를 수행해보도록 한다.

X축은 각 작가들의 단어에 대한 비율, Y축은 Jane Austen 작가의 단어 비율이다.

회귀선을 통해 Austen 작가와 Bronte, Wells 각 작가가 함께 많이 사용한 단어들을 확인할 수 있다.

miss, door 등의 단어가 작가들이 가장 자주 사용한 단어들로 확인할 수 있다.

```
cor.test(data = frequency[frequency$author == 'Bronte Sisters', ], ~ proportion + `Jane Austen`)
```

```
##
##  Pearson's product-moment correlation
##
## data:  proportion and Jane Austen
## t = 111.09, df = 10345, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7286568 0.7462330
## sample estimates:
##       cor
## 0.7375698
```

```
cor.test(data = frequency[frequency$author == 'H.G. Wells', ], ~ proportion + `Jane Austen`)
```

```
##
##  Pearson's product-moment correlation
##
## data:  proportion and Jane Austen
## t = 36.083, df = 6046, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.3999815 0.4414612
## sample estimates:
##       cor
## 0.4209414
```

Austen과 Bronte, Wells 작가의 단어 사용 빈도수에 대한 피어슨 상관관계를 확인해보도록 한다.
가장 먼저 Bronte 작가와 단어의 상관계수를 보면, 0.76의 높은 수치를 보인다.
이는 두 작가들이 사용하는 단어의 빈도수가 매우 유사함을 보인다고 할 수 있다.
반면, Wells 작가는 상관계수가 0.42로 Bronte 작가보다는 조금 떨어지는 수치이다.
그럼에도 불구하고, 약한 상관관계가 있다고 이야기할 수 있겠다.

3-1. Sentiment Analysis

감성분석은 Text 데이터를 Tokenization을 수행하고, 감성 어휘 목록(Sentiment Lexicon)과 inner_join 한다. 그리고 각 단어들의 감성 Label이 무엇인지 확인하여 시각화 등으로 결과물을 도출할 수 있다.

Sentiment lexicon은 3가지 종류가 있는데, **afinn, bing, nrc**가 있다.

- afinn: 단어는 약 2,400개 정도 있으며, -5부터 5까지 부정과 긍정을 나눈다.
- bing: 단어는 약 6,700개 정도 있으며, 단순히 긍정과 부정으로 나눈다.
- nrc: 단어는 약 13,000개 정도 있으며, 긍정과 부정 이외에 fear, anger, sadness 등이 있다.

```
head(get_sentiments('afinn'), 5)
```

```
## # A tibble: 5 x 2
##   word       value
##   <chr>      <dbl>
## 1 abandon       -2
## 2 abandoned     -2
## 3 abandons      -2
## 4 abducted      -2
## 5 abduction     -2
```

```
head(get_sentiments('bing'), 5)
```

```
## # A tibble: 5 x 2
##   word       sentiment
##   <chr>      <chr>
## 1 2-faces    negative
## 2 abnormal   negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
```

```
head(get_sentiments('nrc'), 5)
```

```
## # A tibble: 5 x 2
##   word      sentiment
##   <chr>     <chr>
## 1 abacus    trust
## 2 abandon   fear
## 3 abandon   negative
## 4 abandon   sadness
## 5 abandoned anger
```

```
tidy_books <- austen_books() %>%
                group_by(book) %>%
                mutate(linenumber = row_number(),
                       chapter = cumsum(str_detect(text,
                                                   regex("^chapter [\\divxlc]",
                                                         ignore_case = T)))) %>%
                ungroup() %>%
                unnest_tokens(word, text)

head(tidy_books, 5)
```

```
## # A tibble: 5 x 4
##   book                linenumber chapter word
##   <fct>                    <int>   <int> <chr>
## 1 Sense & Sensibility          1       0 sense
## 2 Sense & Sensibility          1       0 and
## 3 Sense & Sensibility          1       0 sensibility
## 4 Sense & Sensibility          3       0 by
## 5 Sense & Sensibility          3       0 jane
```

austen_books에서 책끼리 묶은 후, 줄번호와 chapter를 토큰화된 단어와 함께 만들어주도록 한다.
따라서 어떤 책에서 줄번호와 chapter를 통해 어떤 단어가 나왔는지 확인할 수 있다.

```
nrc_joy <- get_sentiments('nrc') %>%
            filter(sentiment == 'joy')

tidy_books %>%
  filter(book == 'Emma') %>%
  inner_join(nrc_joy) %>%
  count(word, sort = T)
```

```
## # A tibble: 301 x 2
##    word          n
##    <chr>     <int>
##  1 good        359
##  2 friend      166
##  3 hope        143
##  4 happy       125
##  5 love        117
##  6 deal         92
##  7 found        92
##  8 present      89
##  9 kind         82
## 10 happiness    76
## # ... with 291 more rows
```

nrc 사전에서 joy 감성만 filter 하여 nrc_joy를 만들어준다.
그리고 위에서 만들었던 tidy_books에서 'Emma' 책에 대해 joy 단어만 추려서 빈도 수를 확인한다.
가장 많이 사용된 joy 단어는 good, friend, hope, happy 등이라는 것을 알 수 있다.

3-2. Sentiment trend over section - 80 lines

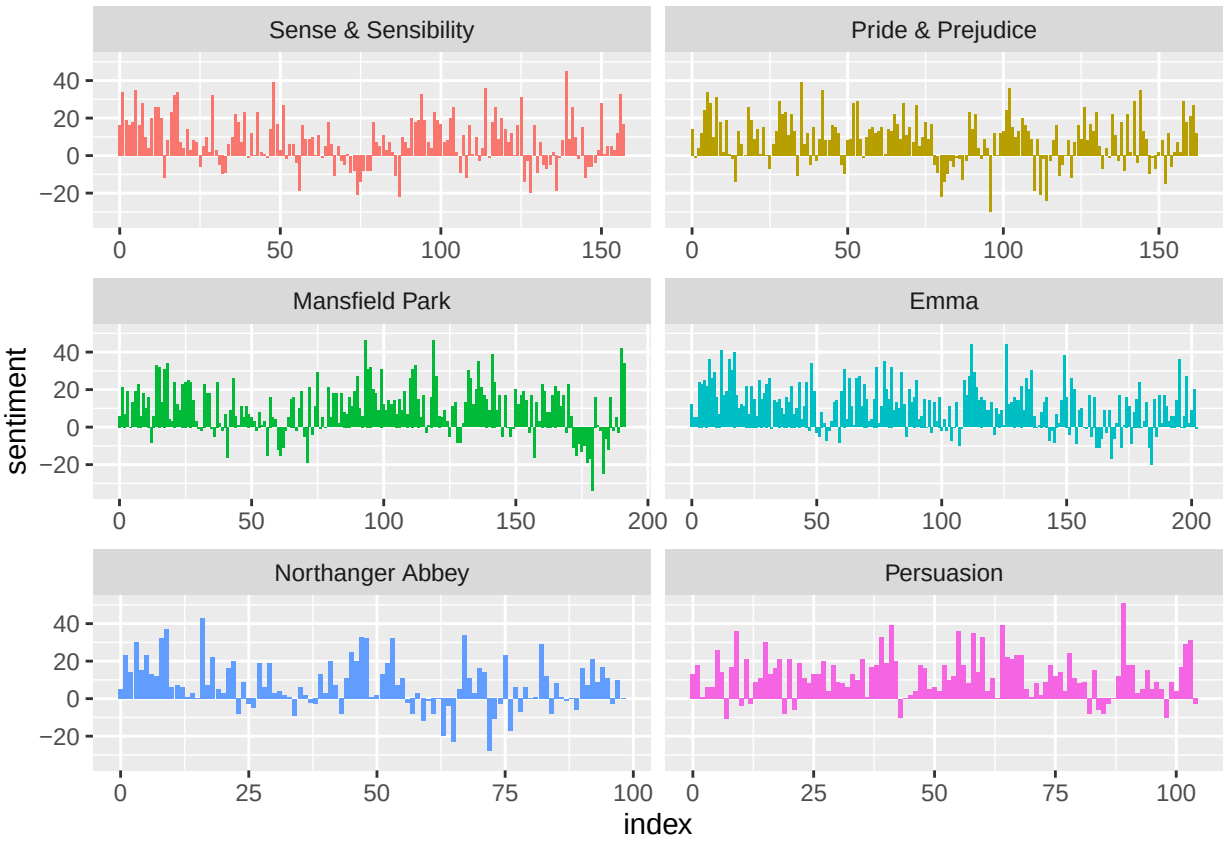
```
jane_austen_sentiment <- tidy_books %>%
                         inner_join(get_sentiments('bing')) %>%
                         count(book, index = linenumber %/% 80, sentiment) %>%
                         spread(sentiment, n, fill = 0) %>%
                         mutate(sentiment = positive - negative)

head(jane_austen_sentiment, 3)
```

```
## # A tibble: 3 x 5
##   book                index negative positive sentiment
##   <fct>               <dbl>    <dbl>    <dbl>     <dbl>
## 1 Sense & Sensibility     0       16       32        16
## 2 Sense & Sensibility     1       19       53        34
## 3 Sense & Sensibility     2       12       31        19
```

그렇다면, 80줄씩 읽어서 bing 사전을 통해 긍정과 부정의 빈도수를 확인해보자.

그리고 spread를 통해 긍정과 부정을 새로운 열로 만든 후, 감성 점수를 만들어준다.

```
ggplot(jane_austen_sentiment, aes(index, sentiment, fill = book)) +
  geom_col(show.legend = F) +
  facet_wrap(~book, ncol = 2, scales = 'free_x')
```



이를 시각화로 표현하면, 각 책 별로 80줄씩 다음과 같은 감성점수를 확인할 수 있다.

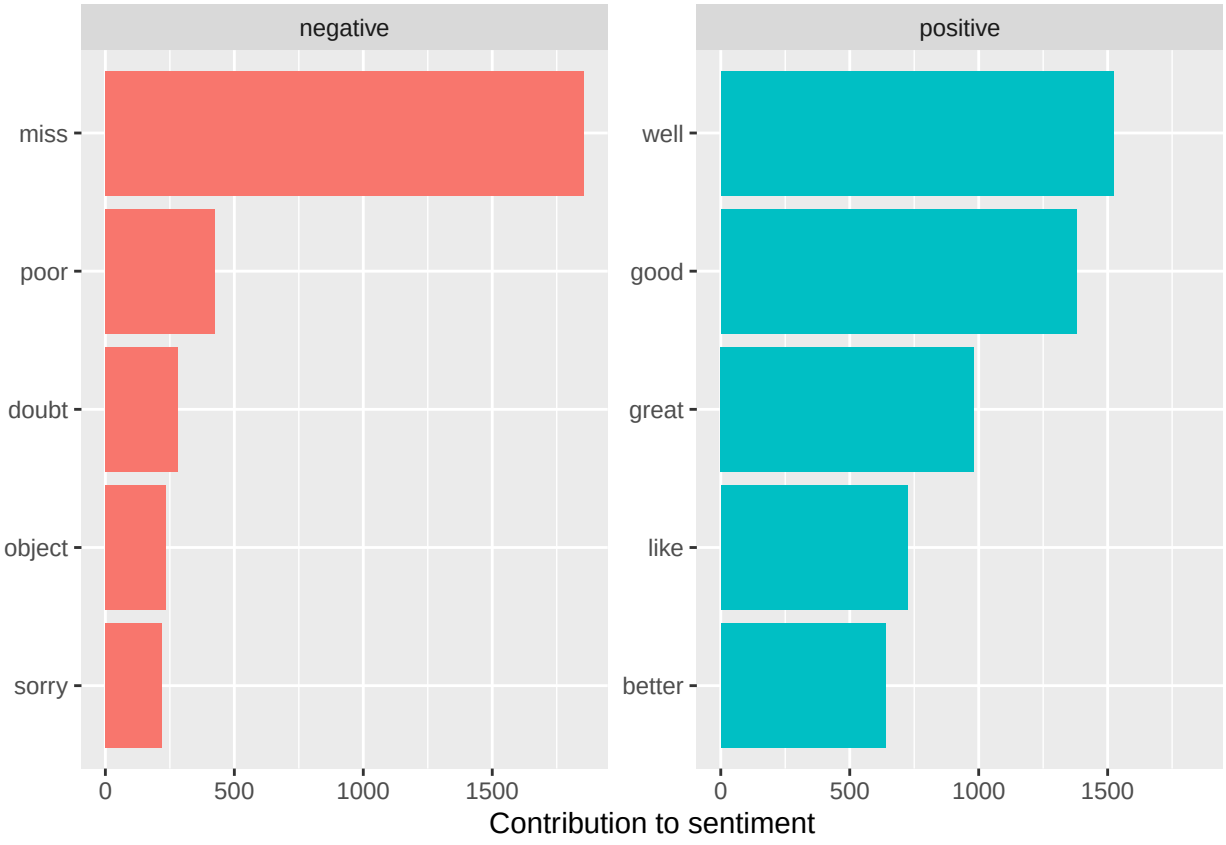
3-3. Most common positive, negative words

```
bing_word_counts <- tidy_books %>%
                      inner_join(get_sentiments('bing')) %>%
                      count(word, sentiment, sort = T)

head(bing_word_counts, 3)
```

```
## # A tibble: 3 x 3
##   word  sentiment     n
##   <chr> <chr>     <int>
## 1 miss  negative   1855
## 2 well  positive   1523
## 3 good  positive   1380
```

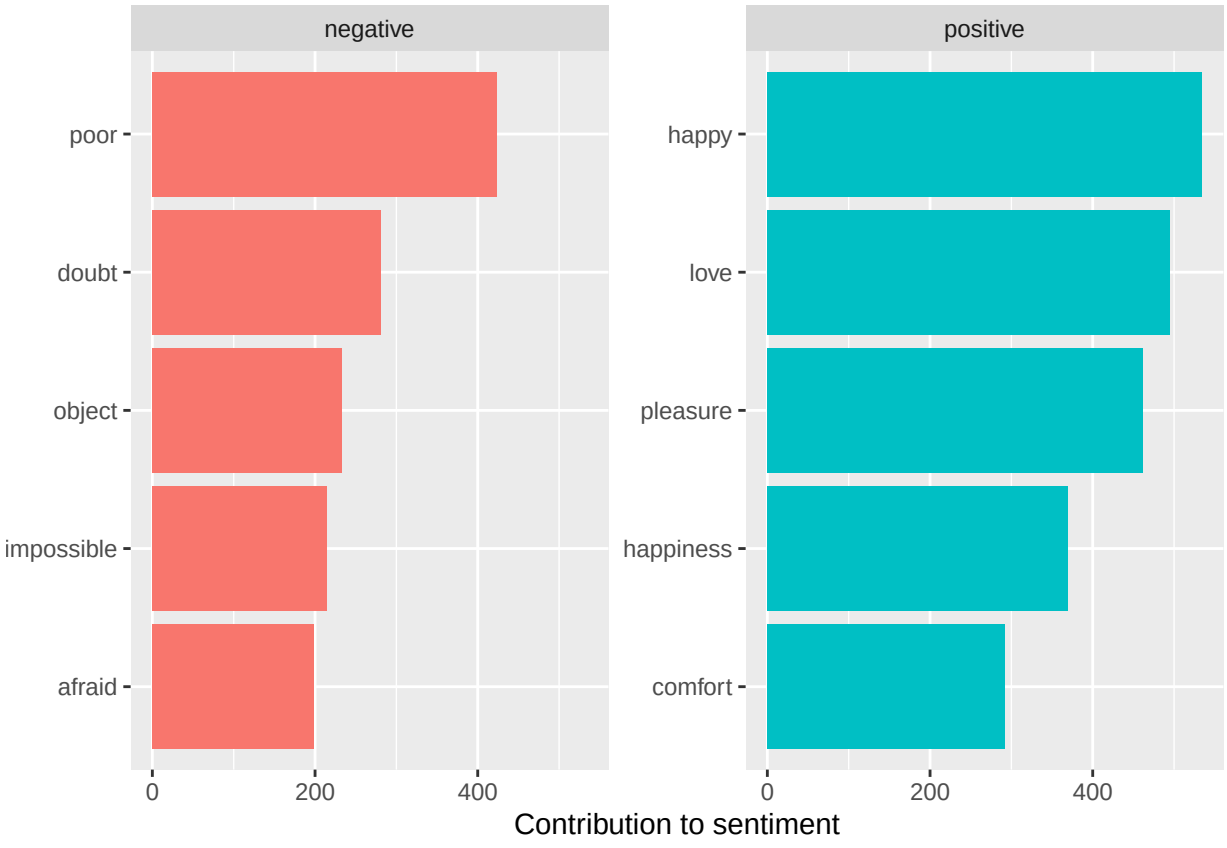
tidy_books의 word에 대하여 bing 사전에 있는 lexicon으로 일치 여부를 파악해 긍/부정을 파악한다. 그 결과 miss라는 부정이 가장 많았고, well, good, great 등의 긍정이 나오는 것을 볼 수 있다.

```
bing_word_counts %>%
  group_by(sentiment) %>% top_n(5) %>% ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col(show.legend = F) +
  labs(y = 'Contribution to sentiment', x = NULL) +
  facet_wrap(~sentiment, scales = 'free_y') + coord_flip()
```



각 감정별로 상위 5개 단어를 막대그래프로 시각화해본다면 다음과 같다.
여기서 주목해야 하는 것은 부정의 가장 높았던 'miss'에 대해 생각해볼 필요가 있다는 것이다.
'miss' 라는 단어가 그리움이라는 부정일 수도 있지만, 잃어버렸거나, 여성의 호칭에도 함께 사용된다.
감성 분석에 있어서 다양한 의미를 갖는 같은 단어에 대한 접근이 되지 않는 한계가 있다.
따라서 이러한 경우에는 아래의 예시와 같이 stopwords에 추가하는 등의 과정이 필요하다.

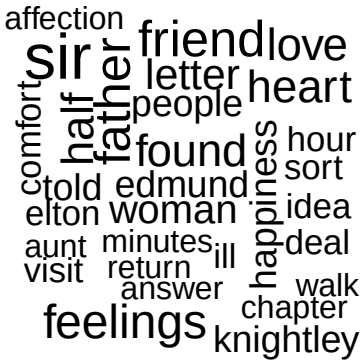
```
custom_stop_words <- bind_rows(data_frame(word = c('miss'), lexicon = c('custom')),
                               stop_words)

bing_word_counts %>%
  anti_join(custom_stop_words) %>%
  group_by(sentiment) %>%  top_n(5) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col(show.legend = F) +
  labs(y = 'Contribution to sentiment', x = NULL) +
  facet_wrap(~sentiment, scales = 'free_y') + coord_flip()
```



따라서 기존의 stop_words에 대하여 miss를 custom으로 추가하여 custom_stop_words를 만든다.
그리고 custom_stop_words를 anti_join 하여 다시 시각화를 하면 위와 같은 결과를 얻을 수 있다.
'miss'와 같이 애매한 단어를 제외하여 부정에서는 'poor'가 가장 높게 나오는 것을 볼 수 있다.

3-4. Wordcloud

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



tidy_books 데이터에 대하여 stop_words를 제거하고, 각 단어의 개수를 확인한다.

그리고 wordcloud 패키지에 있는 해당 함수를 통해 워드클라우드를 만들어보도록 한다.

```
tidy_books %>%
  inner_join(get_sentiments('bing')) %>%
  count(word, sentiment, sort = T) %>%
  acast(word ~ sentiment, value.var = 'n', fill = 0) %>%
  comparison.cloud(colors = c('darkred', 'darkcyan'),
                   max.words = 100)
```

위 시각화는 너무 간소하므로 긍정과 부정을 비교하는 형태로도 만들어볼 수 있다.

acast 함수는 reshape2 패키지의 함수로 형태는 spread와 비슷하지만, list로 결과를 반환한다.

3-5. The most negative chapters in each of Austen's novels

```
bingnegative <- get_sentiments('bing') %>% filter(sentiment == 'negative')
wordcounts <- tidy_books %>%
                group_by(book, chapter) %>%
                summarize(words = n())

head(wordcounts, 5)
```

```
## # A tibble: 5 x 3
## # Groups:   book [1]
##   book                chapter words
##   <fct>                 <int> <int>
## 1 Sense & Sensibility       0     7
## 2 Sense & Sensibility       1  1571
## 3 Sense & Sensibility       2  1970
## 4 Sense & Sensibility       3  1538
## 5 Sense & Sensibility       4  1952
```

bing 사전으로부터 negative 단어만 추려서 bingnegative를 만든다.

그리고 tidy_books에서 book과 chapter에 대해 단어들의 합을 구해주도록 한다.

```
tidy_books %>%
  semi_join(bingnegative) %>%
  group_by(book, chapter) %>%
  summarize(negativewords = n()) %>%
  left_join(wordcounts, by = c('book', 'chapter')) %>%
  mutate(ratio = negativewords / words) %>%
  filter(chapter != 0) %>%
  top_n(1) %>%
  ungroup()
```

```
## # A tibble: 6 x 5
##   book                chapter negativewords words  ratio
##   <fct>                 <int>         <int> <int>  <dbl>
## 1 Sense & Sensibility      43           161  3405 0.0473
## 2 Pride & Prejudice        34           111  2104 0.0528
## 3 Mansfield Park           46           173  3685 0.0469
## 4 Emma                     15           151  3340 0.0452
## 5 Northanger Abbey         21           149  2982 0.0500
## 6 Persuasion                4            62  1807 0.0343
```

그 다음으로는, tidy_books에 대해 bingnegative를 semi_join 해준다.

semi_join은 inner_join과 유사하지만, 테이블을 합치지 않고 첫 행을 반환해주는 역할만 수행한다.

그 후, 각 chapter 별로 부정 단어의 개수를 구하고, wordcounts와 join 해준다.

마지막으로, 전체 단어들에 대해 부정 단어들의 비율(ratio)을 구하고, 각 책 별로 상위 1개를 출력한다.

4-1. Analyzing word and document frequency: TF-IDF

TF-IDF 라는 개념은 다음과 같다.

- TF(Term Frequency): 어떤 단어가 특정 문서에서 얼마나 많이 쓰였는지를 뜻하는 것.
  - 해당 문서에서 확인하고자 하는 단어의 빈도 수 / 해당 문서에서 나오는 전체 단어의 수
- IDF(Inverse Document Frequency): 특정 문서에서 집중적으로 나오는 단어는 어떤 것인지.
  - ln(전체 문서의 수 / 특정 단어를 포함하고 있는 문서의 수)

일반적으로, 어떤 문서를 볼 때 많이 등장하는 단어가 그 문서에서 중요하다고 생각한다. 이게 TF 개념이다.
실제 텍스트 마이닝에서도 이 개념은 기초적으로 단어에 대해 가중치를 할당하는 방법이라고 할 수 있다.
DF는 Document Frequecy로 특정 단어가 문서에 등장한 횟수인데, 이를 전체 문서의 수와 나누면 IDF이다.
최종적으로 TF-IDF = TF*IDF. 값이 클수록 다른 문서에는 적고, 해당 문서에서 자주 등장한다는 것.
한편, BOW(Bag of Words) 라는 개념도 있는데, 이는 TF와 비슷한 개념이기도 하다.
이는 문서 안에서 단어가 몇 번 존재하는지 개수를 세는 것이라고 할 수 있다.
예를 들어, I have a pen, I have a apple pen이라는 문장이 있다고 해보자.
이때 BOW를 I:2, have:2, a:2, apple:1, pen:2 이라고 표현 할 수 있는 것이다.

< 참고 자료 >

- https://wikidocs.net/31698
- https://data-traveler.tistory.com/33

```
book_words <- austen_books() %>%
  unnest_tokens(word, text) %>%
  count(book, word, sort = T) %>%
  ungroup()

head(book_words) # 책에서 특정 단어의 빈도수 계산
```

```
## # A tibble: 6 x 3
##   book           word      n
##   <fct>          <chr> <int>
## 1 Mansfield Park the    6206
## 2 Mansfield Park to     5475
## 3 Mansfield Park and    5438
## 4 Emma           to     5239
## 5 Emma           the    5201
## 6 Emma           and    4896
```

Jane Austen의 책에서 단어의 빈도수를 확인해보도록 한다.
그래서 unnest_tokens 후, count 함수를 통해서 특정 책에서 단어들이 몇 번 나왔는지 확인한다.

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

head(total_words) # 책 별로 등장한 모든 단어의 수 계산
```

```
## # A tibble: 6 x 2
##   book                total
##   <fct>               <int>
## 1 Sense & Sensibility 119957
## 2 Pride & Prejudice   122204
## 3 Mansfield Park      160460
## 4 Emma                160996
## 5 Northanger Abbey     77780
## 6 Persuasion           83658
```

또한 위에서 만든 book_words를 책 별로 묶어서 단어의 수를 계산하도록 한다.

이를 통해 Emma는 약 16만개가, Northanger Abbey는 약 7만개 정도가 나오는 것을 볼 수 있다.

```
book_words <- left_join(book_words, total_words)
book_words
```

```
## # A tibble: 40,379 x 4
##    book              word      n  total
##    <fct>             <chr> <int>  <int>
##  1 Mansfield Park    the    6206 160460
##  2 Mansfield Park    to     5475 160460
##  3 Mansfield Park    and    5438 160460
##  4 Emma              to     5239 160996
##  5 Emma              the    5201 160996
##  6 Emma              and    4896 160996
##  7 Mansfield Park    of     4778 160460
##  8 Pride & Prejudice the    4331 122204
##  9 Emma              of     4291 160996
## 10 Pride & Prejudice to     4162 122204
## # ... with 40,369 more rows
```

이렇게 만든 book_words와 total_words를 left join 하여 결과를 보면 다음과 같다.

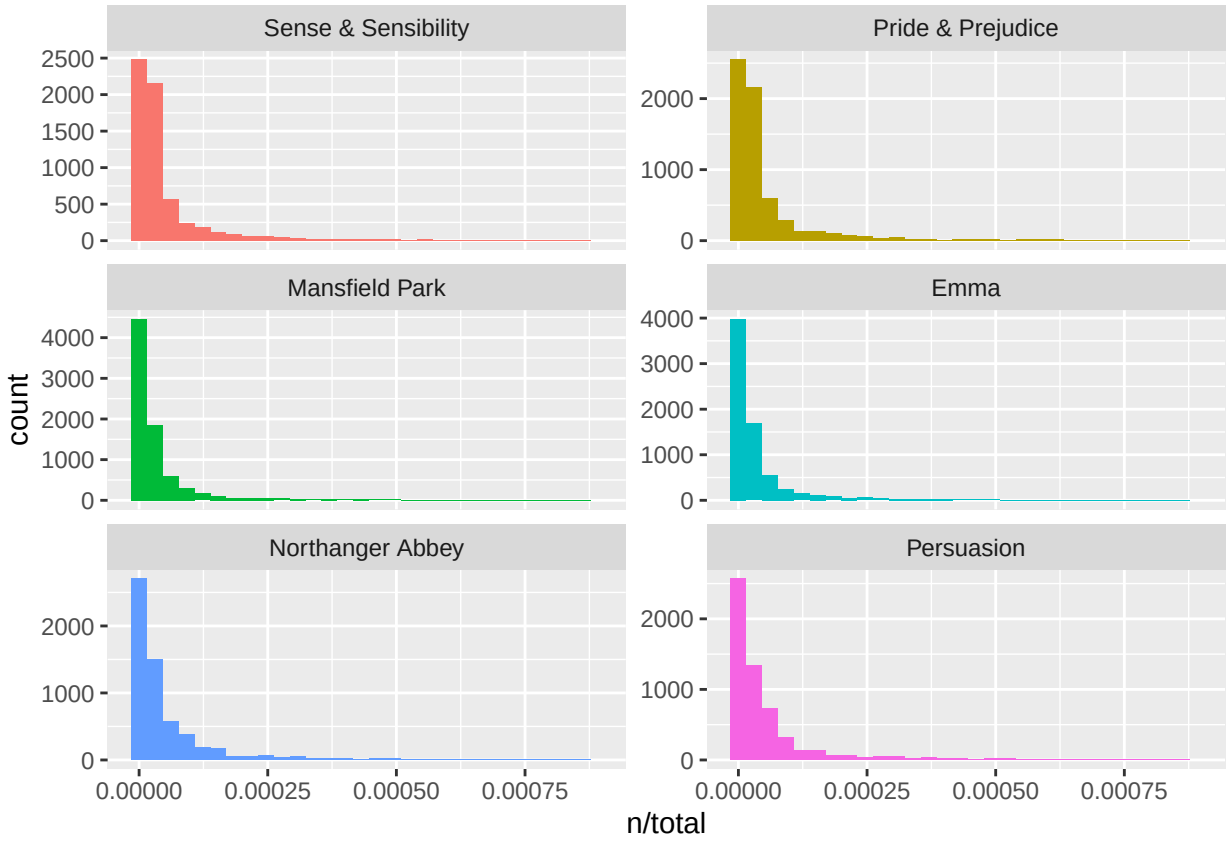
어떤 책에서 특정 단어가 몇 번 나왔고, 전체 단어의 수는 몇 개인지를 확인할 수 있다.

n이 의미하는 것이, 책에서 나온 특정 단어의 빈도 수이고, total이 전체 단어의 수이므로,

향후 n / total을 통해서 Term Frequency 값을 계산 할 수 있을 것이다.

- TF(Term Frequency): 어떤 단어가 특정 문서에서 얼마나 많이 쓰였는지를 뜻하는 것.
- 해당 문서에서 확인하고자 하는 단어의 빈도 수 / 해당 문서에서 나오는 전체 단어의 수

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



book_words에서 n/total이 의미하는 것이 Term Frequency이고, 이를 시각화 해보았다.

히스토그램으로 표현하여 TF 값에 대한 빈도 수를 확인해볼 수 있다.

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

head(freq_by_rank)
```

```
## # A tibble: 6 x 6
## # Groups:   book [2]
##   book           word      n  total  rank `term frequency`
##   <fct>          <chr> <int>  <int> <int>            <dbl>
## 1 Mansfield Park the     6206 160460     1           0.0387
## 2 Mansfield Park to     5475 160460     2           0.0341
## 3 Mansfield Park and    5438 160460     3           0.0339
## 4 Emma           to     5239 160996     1           0.0325
## 5 Emma           the    5201 160996     2           0.0323
## 6 Emma           and    4896 160996     3           0.0304
```

그렇다면, Term Frequency 라는 새로운 변수를 만들어보고자 한다.

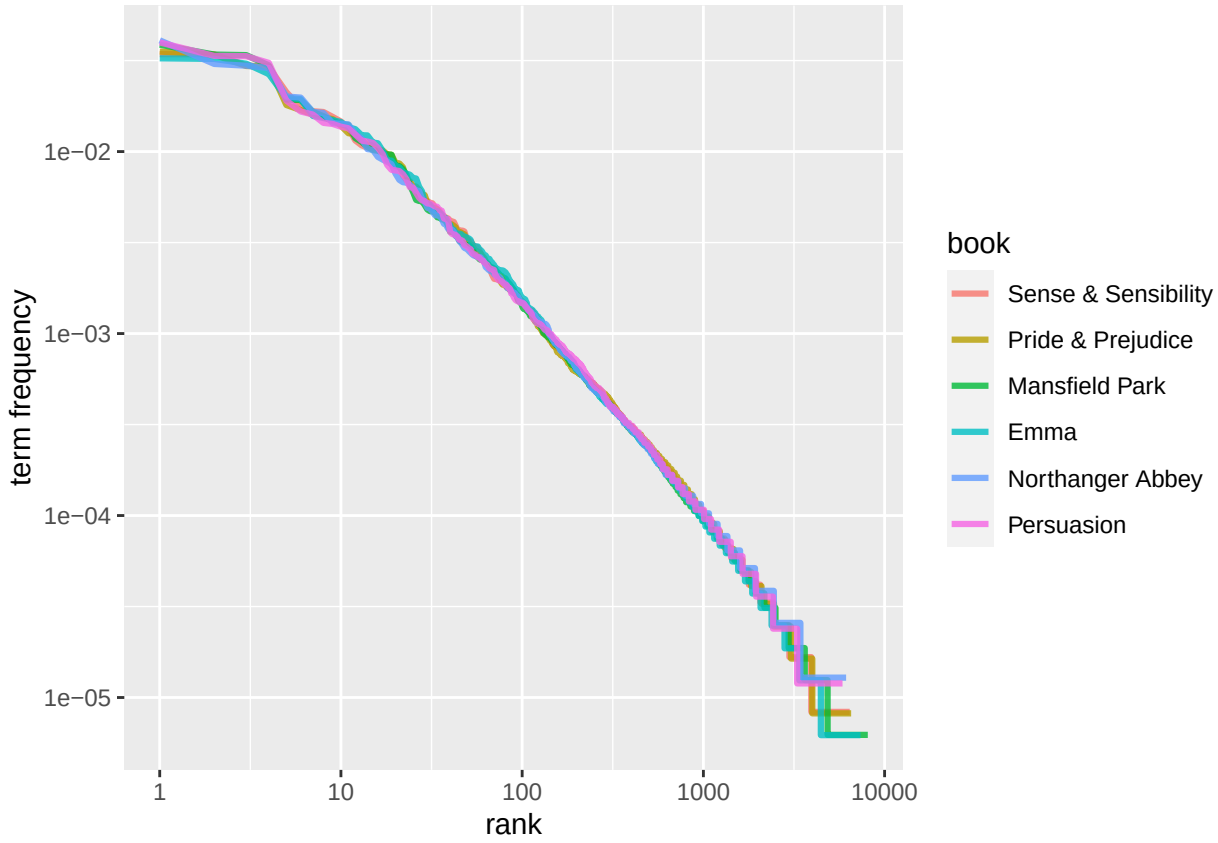
이를 위해서 book_words에 대해 책 별로 묶은 것으로, rank를 매겨주도록 한다.

그 후에 n / total을 통해서 TF 값을 계산하도록 한다.

이를 통해 특정 책과 단어에 대해 rank와 TF를 확인할 수 있다.

4-2. Zipf's law

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



지프의 법칙은 텍스트의 통계적 성질에 관한 법칙 중 하나이다.

등장하는 단어의 빈도를 조사해 크기 순으로 정렬하면, 그 빈도수와 순위는 반비례 관계에 있다는 것.

즉, k번째로 많이 등장한 단어의 빈도가 1번째로 많은 어구의 빈도의 1/k가 되는 법칙이다.

4-3. bind tf idf

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

head(book_words)
```

```
## # A tibble: 6 x 7
##   book           word      n  total     tf   idf tf_idf
##   <fct>          <chr> <int>  <int>  <dbl> <dbl>  <dbl>
## 1 Mansfield Park the    6206 160460 0.0387     0      0
## 2 Mansfield Park to     5475 160460 0.0341     0      0
## 3 Mansfield Park and    5438 160460 0.0339     0      0
## 4 Emma           to     5239 160996 0.0325     0      0
## 5 Emma           the    5201 160996 0.0323     0      0
## 6 Emma           and    4896 160996 0.0304     0      0
```

tidytext 패키지 안에 있는 bind_tf_idf 함수를 활용하면, tf와 idf 모두를 쉽게 계산할 수 있다.

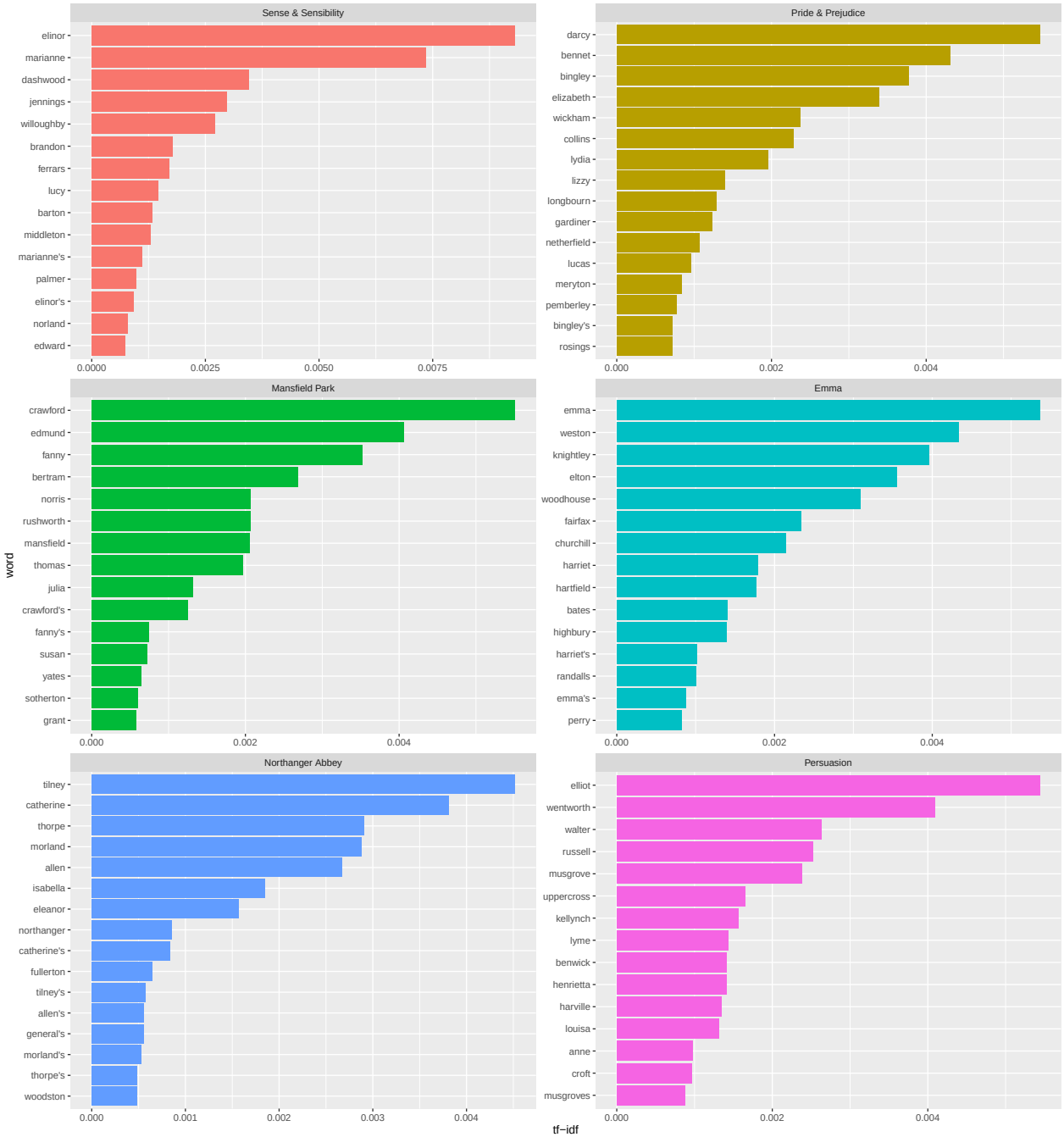
파라미터는 데이터프레임, term, document, n이다. 따라서 word와 book, n을 넣어준다.

이를 통해 결과를 확인해보면, tf와 idf 그리고 tf-idf가 만들어진 것을 볼 수 있다.

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

```
## # A tibble: 40,379 x 6
##    book                word          n      tf   idf  tf_idf
##    <fct>               <chr>     <int>   <dbl> <dbl>   <dbl>
##  1 Sense & Sensibility elinor      623 0.00519  1.79 0.00931
##  2 Sense & Sensibility marianne    492 0.00410  1.79 0.00735
##  3 Mansfield Park      crawford    493 0.00307  1.79 0.00551
##  4 Pride & Prejudice   darcy       373 0.00305  1.79 0.00547
##  5 Persuasion          elliot      254 0.00304  1.79 0.00544
##  6 Emma                emma        786 0.00488  1.10 0.00536
##  7 Northanger Abbey    tilney      196 0.00252  1.79 0.00452
##  8 Emma                weston      389 0.00242  1.79 0.00433
##  9 Pride & Prejudice   bennet      294 0.00241  1.79 0.00431
## 10 Persuasion          wentworth   191 0.00228  1.79 0.00409
## # ... with 40,369 more rows
```

```
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 = F) + labs(X = NULL, y = 'tf-idf') +
  facet_wrap(~book, ncol = 2, scales = 'free') + coord_flip()
```



이를 시각화 하면, 다음과 같은 결과를 얻을 수 있다.

각 책 별로 tf-idf 값이 높은 단어들과 그 값을 확인할 수 있다.

5-1. Topic Modeling - LDA, Gibbs Sampling

```
data('AssociatedPress')

ap_lda <- LDA(AssociatedPress, k = 2, control = list(seed = 1234))
ap_topics <- tidy(ap_lda, matrix = 'beta')
ap_topics
```

```
## # A tibble: 20,946 x 3
##    topic term           beta
##    <int> <chr>         <dbl>
##  1     1 aaron      1.69e-12
##  2     2 aaron      3.90e- 5
##  3     1 abandon    2.65e- 5
##  4     2 abandon    3.99e- 5
##  5     1 abandoned  1.39e- 4
##  6     2 abandoned  5.88e- 5
##  7     1 abandoning 2.45e-33
##  8     2 abandoning 2.34e- 5
##  9     1 abbott     2.13e- 6
## 10     2 abbott     2.97e- 5
## # ... with 20,936 more rows
```

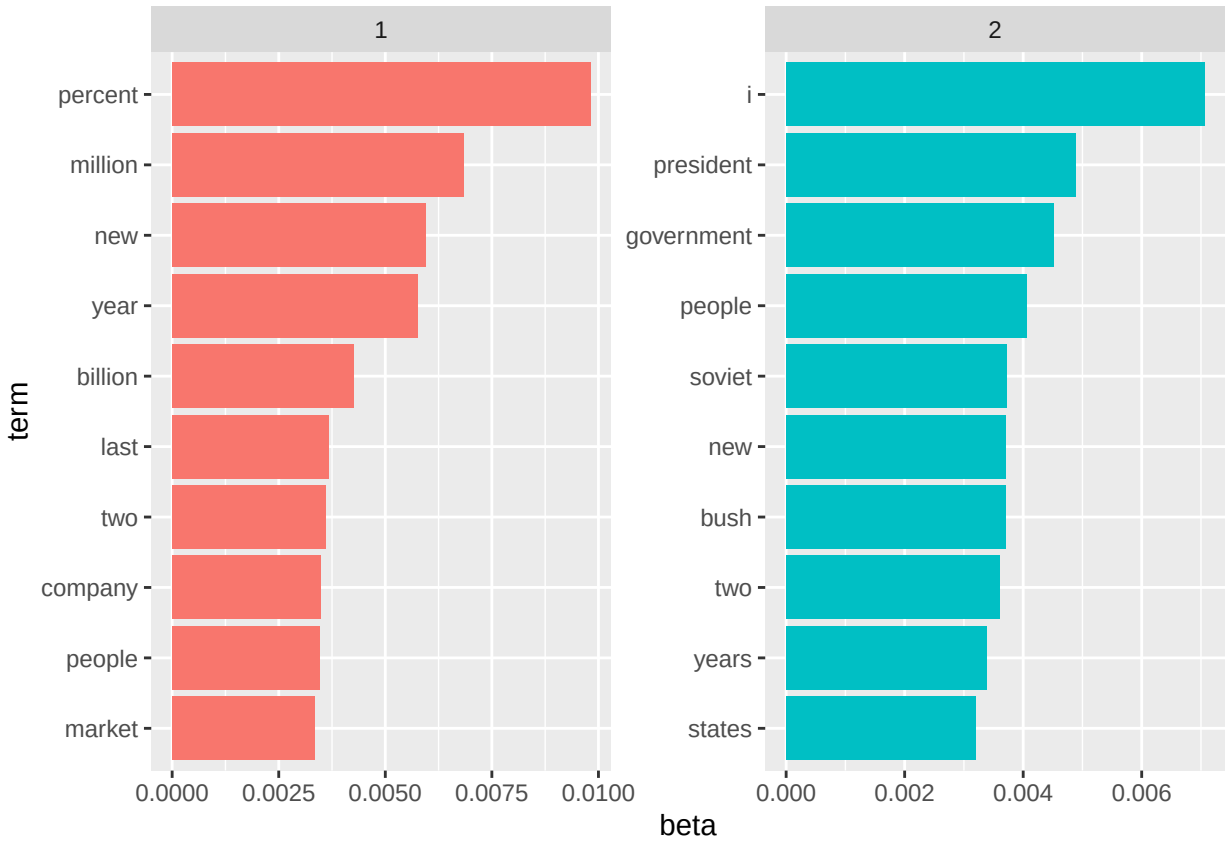
가장 먼저 AssociatedPress를 통해 2,246개의 뉴스 기사를 가지고 오도록 한다.

그리고 topicmodels 패키지에 있는 LDA 함수를 통해서 k = 2로 하여 2개의 Topic을 추리도록 한다.

그 후 tidy 함수를 통해서 결과를 확인해보도록 한다.

5-2. Word-topic probabilities

```
ap_top_terms <- ap_topics %>%
  group_by(topic) %>%
  top_n(10, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

ap_top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(term, beta, fill = factor(topic))) +
  geom_col(show.legend = F) +
  facet_wrap(~topic, scales = 'free') +
  coord_flip() +
  scale_x_reordered()
```



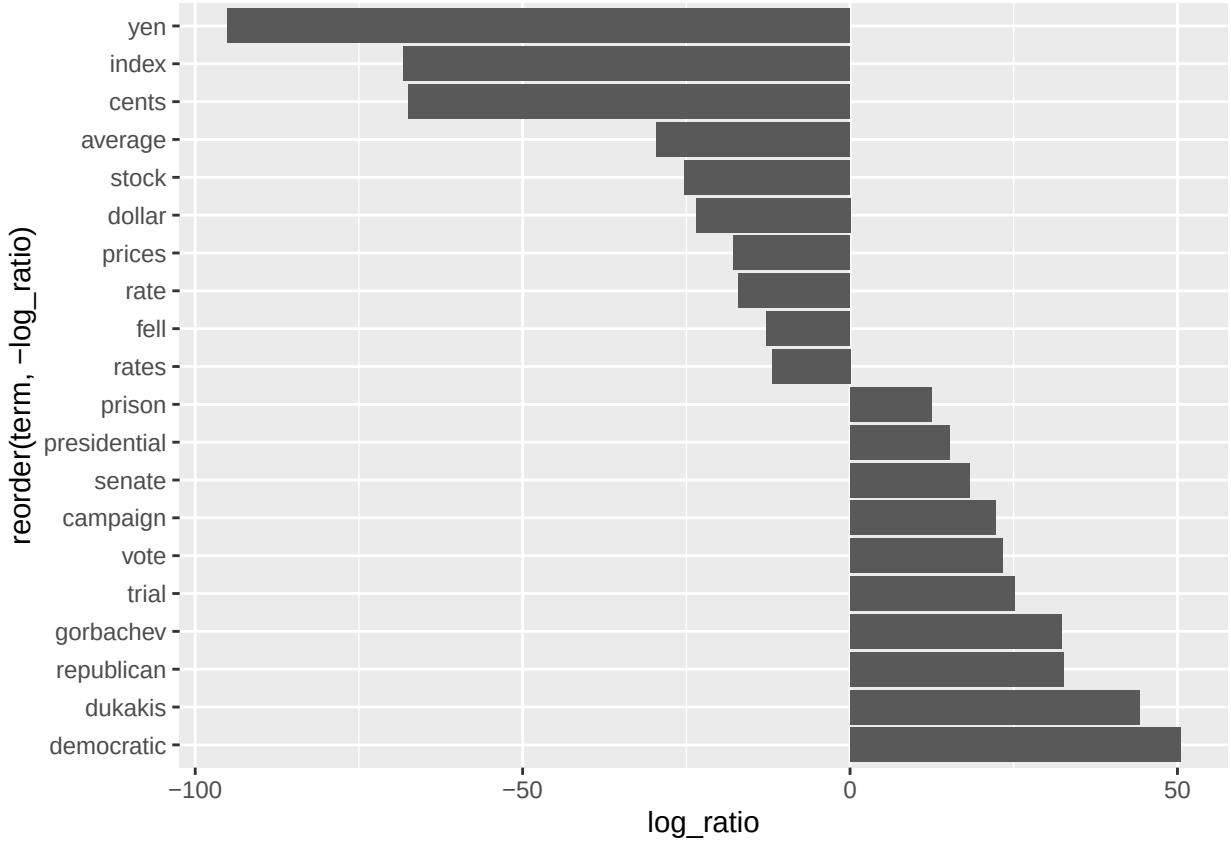
각 Topic 별로 상위 점수를 가지고 있는 단어와 그 수치를 확인해보도록 한다.

그래서 ap_topics 데이터에 대해 topic으로 묶고, beta의 값이 큰 10개로 시각화를 만든다.

이를 통해 결과를 보면, Topic 1은 경제 부분으로 묶였고, 2는 정치로 묶은 것을 추론할 수 있다.

5-3. Terms with greatest difference in beta

```
beta_spread <- ap_topics %>%
  mutate(topic = paste0('topic', topic)) %>%
  spread(topic, beta) %>%
  filter(topic1 > .001 | topic2 > .001) %>%
  mutate(log_ratio = log2(topic2 / topic1))

bind_rows(beta_spread %>% arrange(desc(log_ratio)) %>% head(10),
          beta_spread %>% arrange(desc(log_ratio)) %>% tail(10)) %>%
  arrange(desc(log_ratio)) %>%
  ggplot(aes(x = reorder(term, -log_ratio), y = log_ratio)) +
  geom_bar(stat = 'identity', show.legend = FALSE) +
  coord_flip()
```



다음은 log ratio를 통해서 각 Topic에 대해 단어가 얼마나 나왔는지를 확인한다.
topic2와 topic1에 대하여 밑이 2인 log를 취하여 log_ratio 라는 변수를 만든다.
이는 topic2가 topic1에 대해 상대적으로 작은 경우에는 음수, 그 반대이면 양수가 나온다.
최종적으로, log_ratio의 상위, 하위 10개를 출력하여 시각화를 해본다.

5-4. Document-topic prabability

```
ap_documents <- tidy(ap_lda, matrix = 'gamma')
ap_documents
```

```
## # A tibble: 4,492 x 3
##    document topic    gamma
##       <int> <int>    <dbl>
##  1        1     1 0.248
##  2        2     1 0.362
##  3        3     1 0.527
##  4        4     1 0.357
##  5        5     1 0.181
##  6        6     1 0.000588
##  7        7     1 0.773
##  8        8     1 0.00445
##  9        9     1 0.967
## 10       10     1 0.147
## # ... with 4,482 more rows
```