Introduction to Text Mining

Kylie Ariel Bemis

11/2/2018

Mining text data in R using `tidytext`

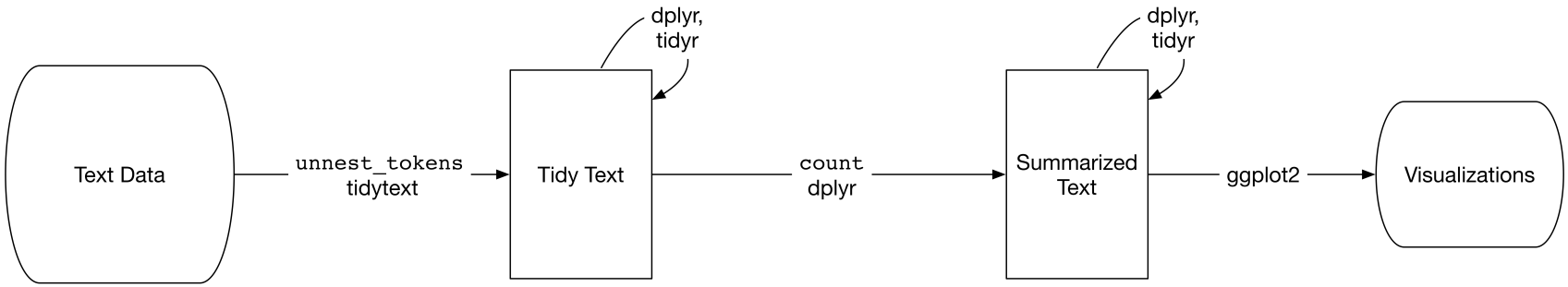


Figure 1: *Text Mining with R* by Julia Silge and David Robinson

Book freely available at http://tidytextmining.com.

Text data

Typically, text data is not tidy.

Often, text data is not even tabular or structured at all.

Most commonly, raw text data comes in two forms:

- **String:** Text data (e.g., emails, tweets, books) can be stored as strings and is most often first encountered and read into memory as strings, i.e., a character vector in R.
- **Corpus:** A collection of text data, typically stored as strings, where each object is distinct in some way (i.e., each string is a specific book or email or tweet), sometimes annotated with additional metadata

To analyze text data, we first need to put it into some sort of structured, tabular format.

Tidying text data

To create a structured dataset from unstructured text data, first we must identify the basic unit of text that we need to analyze.

A **token** (or term) is a meaningful unit of text that we are interested in using for analysis; we apply **tokenization** to split the text into tokens.

Most commonly, tokens are:

- A **word** is a distinct, meaningful element of text. In English, words are easy to identify, but in other languages, word tokenization can be a more difficult process.
- An **n-gram** is a contiguous sequence of words in a text. E.g., a 2-gram or "bigram" is two consecutive words, and a 3-gram or "trigram" is three consecutive words.

Tokenization example

```
library(tokenizers)
text <- "The quick brown fox jumps over the lazy dog"
tokenize_words(text)
```

```
## [[1]]
## [1] "the"   "quick" "brown" "fox"   "jumps" "over"  "the"
## [8] "lazy"  "dog"
```

```
tokenize_ngrams(text, n=2)
```

```
## [[1]]
## [1] "the quick"   "quick brown" "brown fox"   "fox jumps"
## [5] "jumps over"  "over the"    "the lazy"    "lazy dog"
```

Tidying text data (cont'd)

After a text document has been tokenized, there are two common formats for working with the text data:

- The **tidy text** format follows the idea of "tidy data" and formats text data as "one-token-per-row". This is the format we will focus on now.
  - Each token is a row
  - Each variable is a column
  - Each value is in a cell
- A **document-term matrix** is a sparse matrix format used commonly in machine learning where each *unique* token (or term) is a column.
  - Each document is a row
  - Each term is a column
  - Each value is a count or frequency

These formats are each useful for different purposes. We will focus on the tidy format, because it is compatible with all of the `tidyverse` functions we have learned so far.

Tidy text format

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

```
## # A tibble: 20 x 2
##     line word
##    <int> <chr>
##  1     1 because
##  2     1 i
##  3     1 could
##  4     1 not
##  5     1 stop
##  6     1 for
##  7     1 death
##  8     2 he
##  9     2 kindly
## 10     2 stopped
## # ... with 10 more rows
```

Document-term matrix

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

```
##   because i could not stop for death he kindly stopped me
## 1       1 1     1   1    1   1     1  0       0       0  0
## 2       0 0     0   0    0   1     0  1       1       1  1
## 3       0 0     0   0    0   0     0  0       0       0  0
## 4       1 0     0   0    0   0     0  0       0       0  0
##   the carriage held but just ourselves immortality
## 1   0        0    0   0    0         0           0
## 2   0        0    0   0    0         0           0
## 3   1        1    1   1    1         1           0
## 4   0        0    0   0    0         0           1
```

Tokenizing with `unnest_tokens()`

Given a data frame with a column of strings, `unnest_tokens()` will tokenize the strings and return a new data frame in tidy text format.

```
unnest_tokens(data, output, input, token, ...)
```

- The first argument is the data
- The `output` is the name of a new column that will be created with the tokens
- The `input` is the name of the column with the strings to be tokenized
- The `token` is the type of tokenizer to use (e.g., word or n-gram)

unnest_tokens() example

```
library(tidytext)
poem <- tibble(line=1:4, text=emily)
tidy_poem <- unnest_tokens(poem, word, text)
print(tidy_poem, n=10)
```

```
## # A tibble: 20 x 2
##     line word
##    <int> <chr>
##  1     1 because
##  2     1 i
##  3     1 could
##  4     1 not
##  5     1 stop
##  6     1 for
##  7     1 death
##  8     2 he
##  9     2 kindly
## 10     2 stopped
## # ... with 10 more rows
```

unnest_tokens() example (cont'd)

```
library(tidytext)
poem <- tibble(line=1:4, text=emily)
tidy_poem <- unnest_tokens(poem, bigram, text, token="ngrams", n=2)
print(tidy_poem, n=10)
```

```
## # A tibble: 16 x 2
##     line bigram
##    <int> <chr>
##  1     1 because i
##  2     1 i could
##  3     1 could not
##  4     1 not stop
##  5     1 stop for
##  6     1 for death
##  7     2 he kindly
##  8     2 kindly stopped
##  9     2 stopped for
## 10     2 for me
## # ... with 6 more rows
```

Importing text data into R

In order to analyze text data, we must first import it into R as strings. We can do this simply with the `readLines()` function.

```
library(tidytext)
text <- readLines("prideprejudice.txt")
head(text, n=10)
```

```
##  [1] "PRIDE AND PREJUDICE"
##  [2] ""
##  [3] "By Jane Austen"
##  [4] ""
##  [5] ""
##  [6] ""
##  [7] "Chapter 1"
##  [8] ""
##  [9] ""
## [10] "It is a truth universally acknowledged, that a single man in p
```

Preparing text data for tidying

In order to use the `unnest_tokens()` function, we need to put the string data in a data frame.

```
prideprejudice <- tibble(line=1:length(text), text=text)
prideprejudice
```

```
## # A tibble: 13,030 x 2
##     line text
##    <int> <chr>
##  1     1 PRIDE AND PREJUDICE
##  2     2 ""
##  3     3 By Jane Austen
##  4     4 ""
##  5     5 ""
##  6     6 ""
##  7     7 Chapter 1
##  8     8 ""
##  9     9 ""
## 10    10 It is a truth universally acknowledged, that a si~
## # ... with 13,020 more rows
```

Tidying the text data

```
tidy_pride <- prideprejudice %>%
  unnest_tokens(word, text)

tidy_pride
```

```
## # A tibble: 122,204 x 2
##     line word
##    <int> <chr>
##  1     1 pride
##  2     1 and
##  3     1 prejudice
##  4     3 by
##  5     3 jane
##  6     3 austen
##  7     7 chapter
##  8     7 1
##  9    10 it
## 10    10 is
## # ... with 122,194 more rows
```

Check most common words

```
tidy_pride %>% count(word, sort=TRUE)
```

```
## # A tibble: 6,538 x 2
##    word      n
##    <chr> <int>
##  1 the    4331
##  2 to     4162
##  3 of     3610
##  4 and    3585
##  5 her    2203
##  6 i      2065
##  7 a      1954
##  8 in     1880
##  9 was    1843
## 10 she    1695
## # ... with 6,528 more rows
```

Not very useful.

Remove stop words

Stop words are words that are not useful for analysis.

```
stop_words
```

```
## # A tibble: 1,149 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
##  7 accordingly SMART
##  8 across      SMART
##  9 actually    SMART
## 10 after       SMART
## # ... with 1,139 more rows
```

Check most common words again

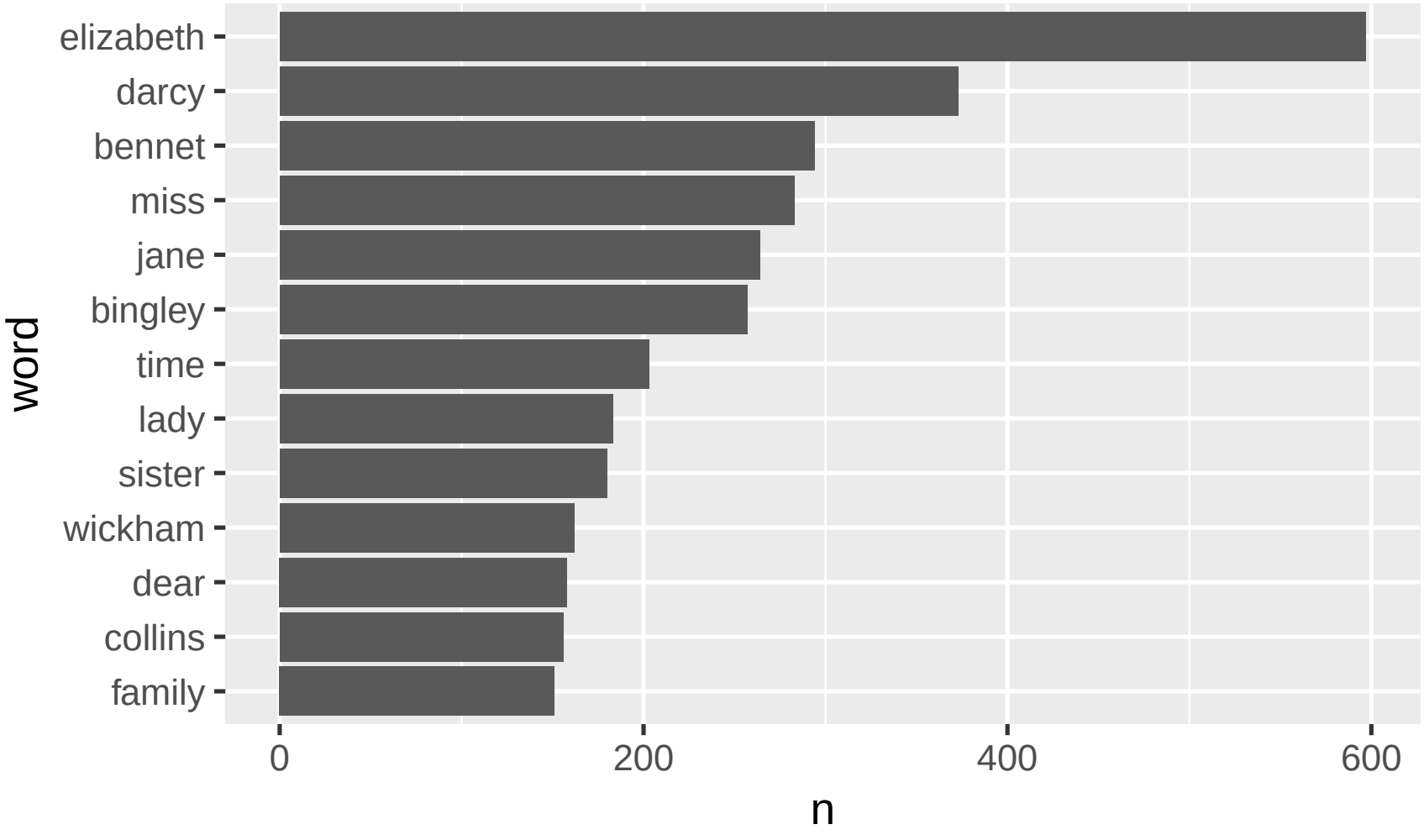
```
pride_stop <- tibble(word=c("elizabeth", "darcy", "bennet", "jan

tidy_pride %>%
  anti_join(stop_words, by="word") %>%
  anti_join(pride_stop, by="word") %>%
  count(word, sort=TRUE)
```

```
## # A tibble: 6,013 x 2
##    word         n
##    <chr>    <int>
##  1 miss       283
##  2 time       203
##  3 lady       183
##  4 sister     180
##  5 wickham    162
##  6 dear       158
##  7 collins    156
##  8 family     151
##  9 day        140
## 10 lydia      133
```

Visualize the most common words

```
tidy_pride %>%
  anti_join(stop_words, by="word") %>%
  count(word, sort=TRUE) %>%
  filter(n > 150) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(x=word, y=n)) +
  geom_col() +
  coord_flip()
```

What does `reorder()` do here?


elizabeth
darcy
bennet
miss
jane
bingley
time
lady
sister
wickham
dear
collins
family
word
0
200
400
600
n

Analyzing all Jane Austen novels

Now that we know how to import and tidy raw text data, we will use data from the `janeaustenr` package, which includes the full text from all six of Jane Austen's completed, published novels.

```
library(janeaustenr)
austen_books()
```

```
## # A tibble: 73,422 x 2
##    text                  book
##  * <chr>                 <fct>
##  1 SENSE AND SENSIBILITY Sense & Sensibility
##  2 ""                    Sense & Sensibility
##  3 by Jane Austen        Sense & Sensibility
##  4 ""                    Sense & Sensibility
##  5 (1811)                Sense & Sensibility
##  6 ""                    Sense & Sensibility
##  7 ""                    Sense & Sensibility
##  8 ""                    Sense & Sensibility
##  9 ""                    Sense & Sensibility
## 10 CHAPTER 1             Sense & Sensibility
## # ... with 73,412 more rows
```

Tidying the works of Jane Austen

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

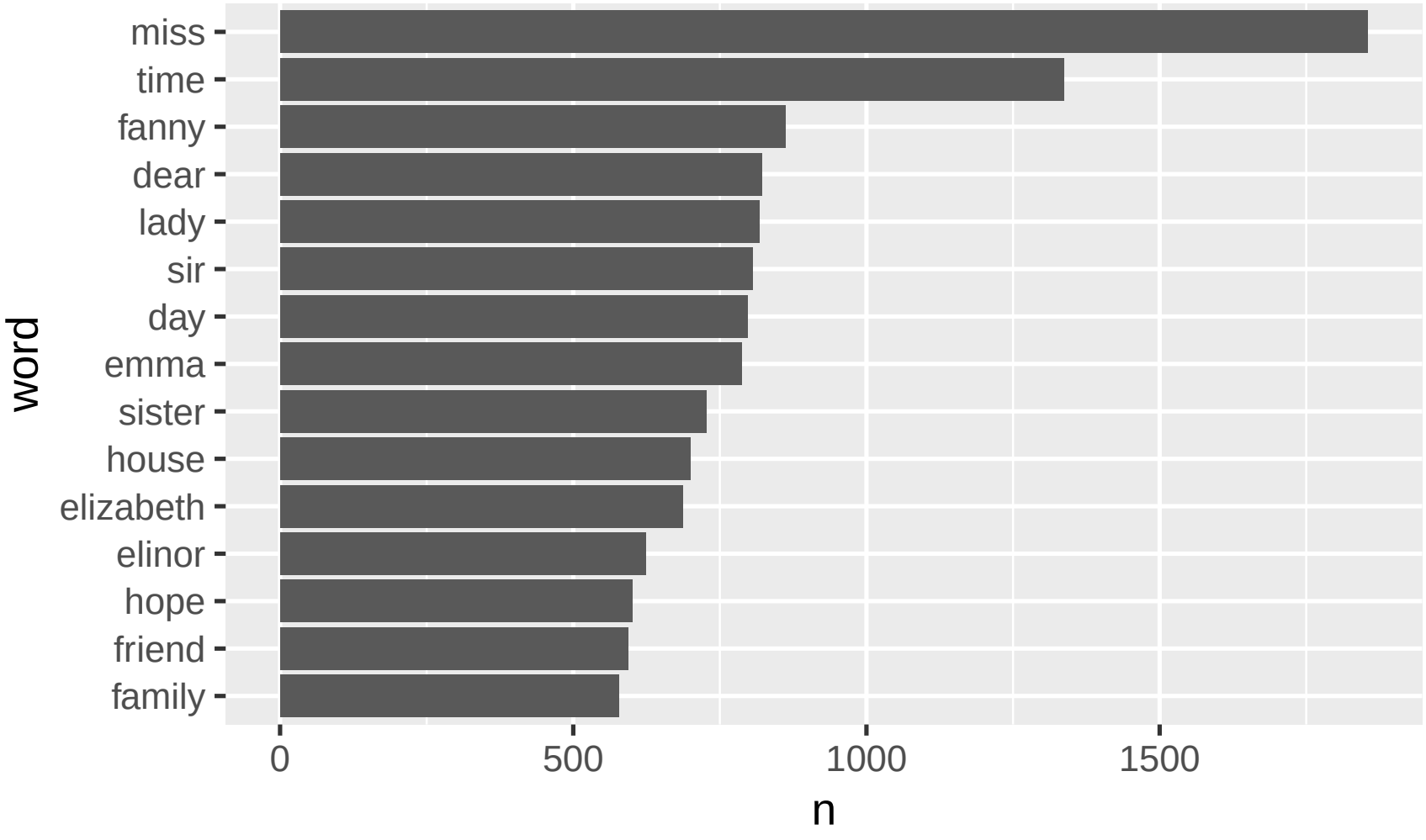
```
tidy_austen
```

```
## # 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
```

Visualize the most common words in Jane Austen

```
tidy_austen %>%
  anti_join(stop_words, by="word") %>%
  count(word, sort=TRUE) %>%
  top_n(15) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(x=word, y=n)) +
  geom_col() +
  coord_flip()
```

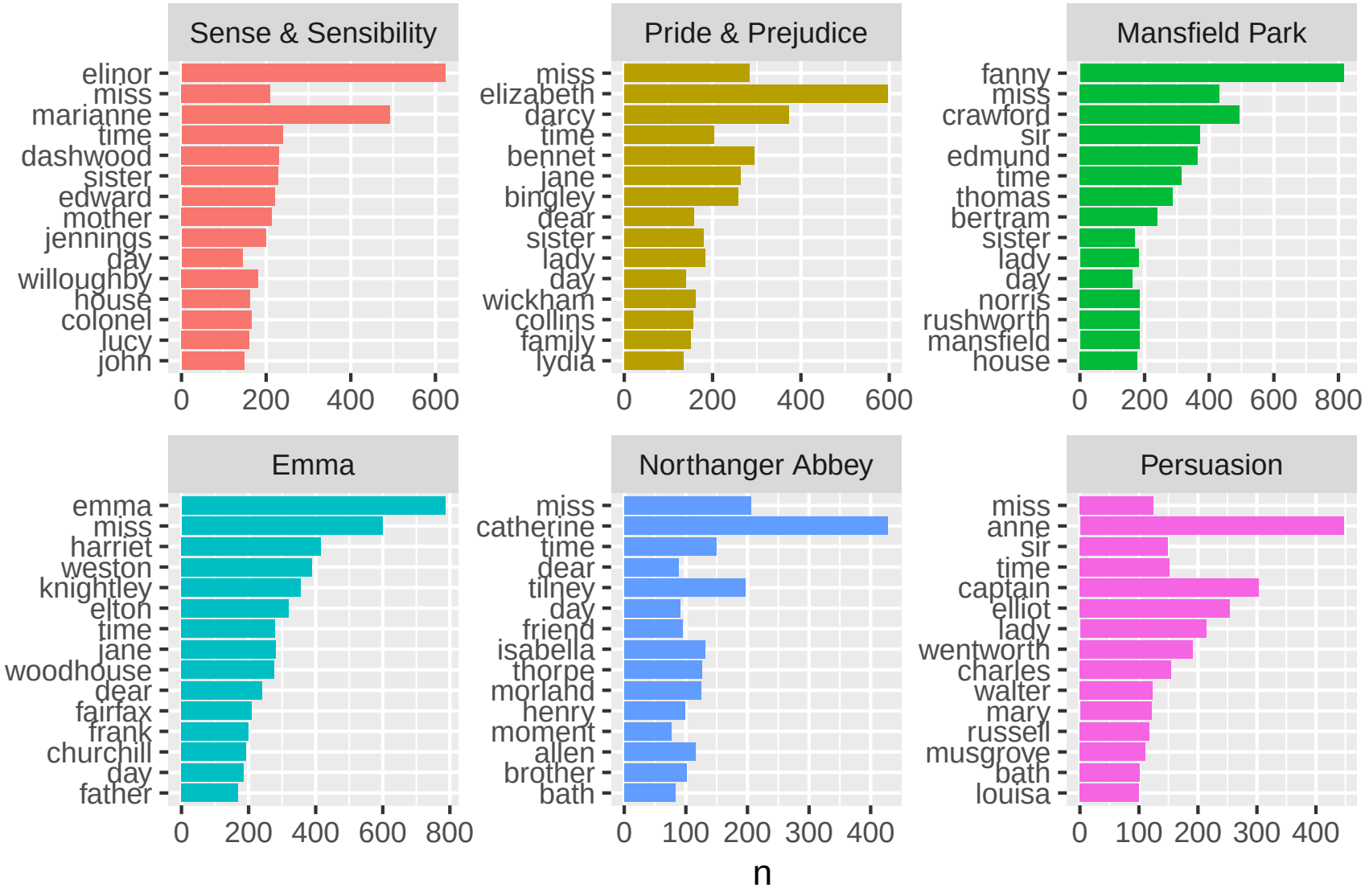
```
## Selecting by n
```



Visualize the most common words by novel

```
tidy_austen %>%
  anti_join(stop_words, by="word") %>%
  count(book, word, sort=TRUE) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(book) %>%
  top_n(15) %>%
  ungroup() %>%
  ggplot(aes(word, n, fill = book)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "n") +
  facet_wrap(~book, ncol = 3, scales = "free") +
  coord_flip()
```

```
## Selecting by n
```



Analyze word frequency

When analyzing a corpus, we may want to identify words that are the most important to each document. However, looking at frequency alone may not help, since words that appear very commonly across all documents will be ranked highly, yet are not very important to any particular document.

Instead, we could give a weighting to words that are used most frequently in some documents, and not as much in others. The *inverse document frequency* (idf) does this by downweighting words that are very common across all documents in a corpus.

$$idf(\text{term}) = \ln\left(\frac{n_{\text{documents}}}{n_{\text{documents containing term}}}\right)$$

The *idf* can be combined with the *tf*, or term frequency (by multiplying them together) to calculate the *tf-idf*, which measures the importance of a term to a particular document in a corpus.

Although it has no basis in statistics or information theory, the *tf-idf* is a commonly used heuristic measure.

Calculate term frequency

```
austen_words <- tidy_austen %>%
  count(book, word, sort=TRUE)

austen_totals <- austen_words %>%
  group_by(book) %>%
  summarise(total = sum(n))

austen_words <- austen_words %>%
  left_join(austen_totals) %>%
  mutate(tf = n / total)
```

```
## Joining, by = "book"
```

```
austen_words
```

```
## # A tibble: 40,379 x 5
##    book                word      n  total     tf
##    <fct>               <chr> <int>  <int>  <dbl>
##  1 Mansfield Park      the    6206 160460 0.0387
##  2 Mansfield Park      to     5475 160460 0.0341
##  3 Mansfield Park      and    5438 160460 0.0339
##  4 Emma                to     5239 160996 0.0325
##  5 Emma                the    5201 160996 0.0323
##  6 Emma                and    4896 160996 0.0304
##  7 Mansfield Park      of     4778 160460 0.0298
##  8 Pride & Prejudice   the    4331 122204 0.0354
##  9 Emma                of     4291 160996 0.0267
## 10 Pride & Prejudice   to     4162 122204 0.0341
## # ... with 40,369 more rows
```

Calculate inverse document frequency

```
austen_words <- austen_words %>%
  mutate(n_doc = n_distinct(book)) %>%
  group_by(word) %>%
  mutate(idf = log(n_doc / n()),
         tf_idf = tf * idf) %>%
  select(-n_doc, -total) %>%
  ungroup()
```

```
austen_words
```

```
## # A tibble: 40,379 x 6
##    book                word      n     tf   idf tf_idf
##    <fct>               <chr> <int>  <dbl> <dbl>  <dbl>
##  1 Mansfield Park      the    6206 0.0387     0      0
##  2 Mansfield Park      to     5475 0.0341     0      0
##  3 Mansfield Park      and    5438 0.0339     0      0
##  4 Emma                to     5239 0.0325     0      0
##  5 Emma                the    5201 0.0323     0      0
##  6 Emma                and    4896 0.0304     0      0
##  7 Mansfield Park      of     4778 0.0298     0      0
##  8 Pride & Prejudice   the    4331 0.0354     0      0
##  9 Emma                of     4291 0.0267     0      0
## 10 Pride & Prejudice   to     4162 0.0341     0      0
## # ... with 40,369 more rows
```

```
austen_words %>% 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
```

Calculating tf-idf with `bind_tf_idf()`

Alternatively, the *tf-idf* for each term can be calculated easily by using the `bind_tf_idf()` function provided by the `tidytext` package.

`bind_tf_idf(data, term, document, n)`

- The first argument is the data
- The following arguments give the name of the columns that identify the term, document, and counts

```
austen_words_tf <- tidy_austen %>%
  count(book, word, sort=TRUE) %>%
  bind_tf_idf(word, book, n)
```

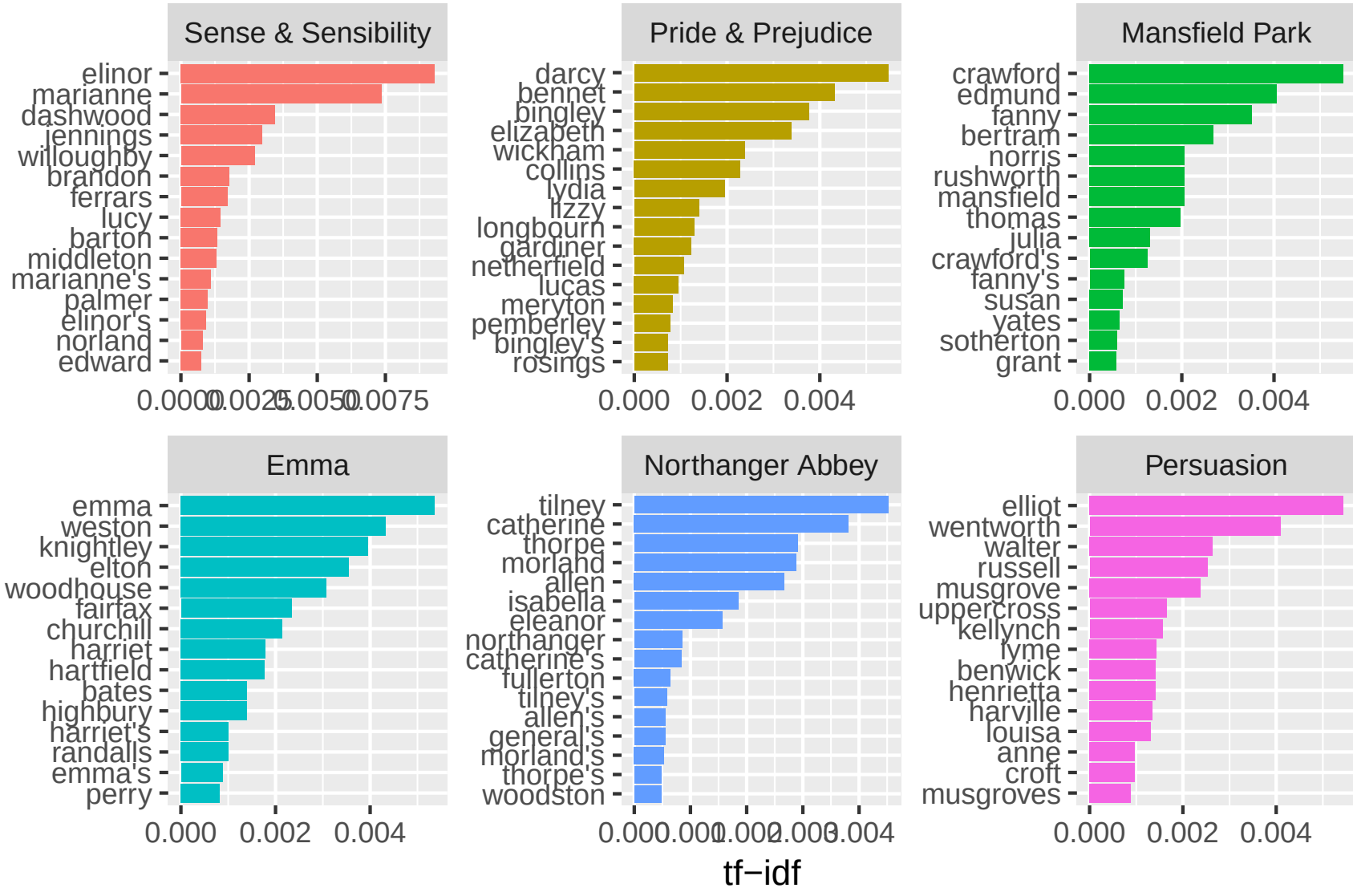
```
austen_words_tf %>% arrange(desc(tf_idf))
```

```
## # 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
```

Visualizing the tf-idf of Jane Austen novels

```
austen_words %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(book) %>%
  top_n(15) %>%
  ungroup() %>%
  ggplot(aes(word, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 3, scales = "free") +
  coord_flip()
```

```
## Selecting by tf_idf
```



Analyzing n-grams

So far we have looked at the importance of individual words, but not the relationships between them. We can investigate relationships between words (e.g., which words tend to immediately follow others, or what words tend to co-occur in a corpus) by tokenizing on n-grams. We will focus on $n = 2$, or bigrams.

```
austen_bigrams <- austen_books() %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)
```

```
austen_bigrams
```

```
## # A tibble: 725,049 x 2
##    book                bigram
##    <fct>               <chr>
##  1 Sense & Sensibility sense and
##  2 Sense & Sensibility and sensibility
##  3 Sense & Sensibility sensibility by
##  4 Sense & Sensibility by jane
##  5 Sense & Sensibility jane austen
##  6 Sense & Sensibility austen 1811
##  7 Sense & Sensibility 1811 chapter
##  8 Sense & Sensibility chapter 1
##  9 Sense & Sensibility 1 the
## 10 Sense & Sensibility the family
## # ... with 725,039 more rows
```

Check most common bigrams in Jane Austen

```
austen_bigrams %>% count(bigram, sort=TRUE)
```

```
## # A tibble: 211,236 x 2
##    bigram       n
##    <chr>    <int>
##  1 of the    3017
##  2 to be     2787
##  3 in the    2368
##  4 it was    1781
##  5 i am      1545
##  6 she had   1472
##  7 of her    1445
##  8 to the    1387
##  9 she was   1377
## 10 had been  1299
## # ... with 211,226 more rows
```

Separate bigrams and remove stop words

```
austen_bigrams <- austen_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")

austen_bigrams <- austen_bigrams %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)
```

Check most common bigrams again

```
austen_bigrams %>% count(word1, word2, sort=TRUE)
```

```
## # A tibble: 33,421 x 3
##    word1   word2          n
##    <chr>   <chr>      <int>
##  1 sir     thomas       287
##  2 miss    crawford     215
##  3 captain wentworth    170
##  4 miss    woodhouse    162
##  5 frank   churchill    132
##  6 lady    russell      118
##  7 lady    bertram      114
##  8 sir     walter       113
##  9 miss    fairfax      109
## 10 colonel brandon      108
## # ... with 33,411 more rows
```

Check most common use of “miss”

```
austen_bigrams %>%
  filter(word1 == "miss") %>%
  count(word1, word2, sort=TRUE)
```

```
## # A tibble: 121 x 3
##    word1 word2          n
##    <chr> <chr>      <int>
##  1 miss  crawford     215
##  2 miss  woodhouse    162
##  3 miss  fairfax      109
##  4 miss  bates        103
##  5 miss  tilney        82
##  6 miss  bingley       72
##  7 miss  dashwood      65
##  8 miss  bennet        60
##  9 miss  morland       55
## 10 miss  smith         52
## # ... with 111 more rows
```

Calculate tf-idf for bigrams

```
austen_bigrams_tf <- austen_bigrams %>%
  unite(bigram, word1, word2, sep = " ") %>%
  count(book, bigram) %>%
  bind_tf_idf(bigram, book, n) %>%
  arrange(desc(tf_idf))
```

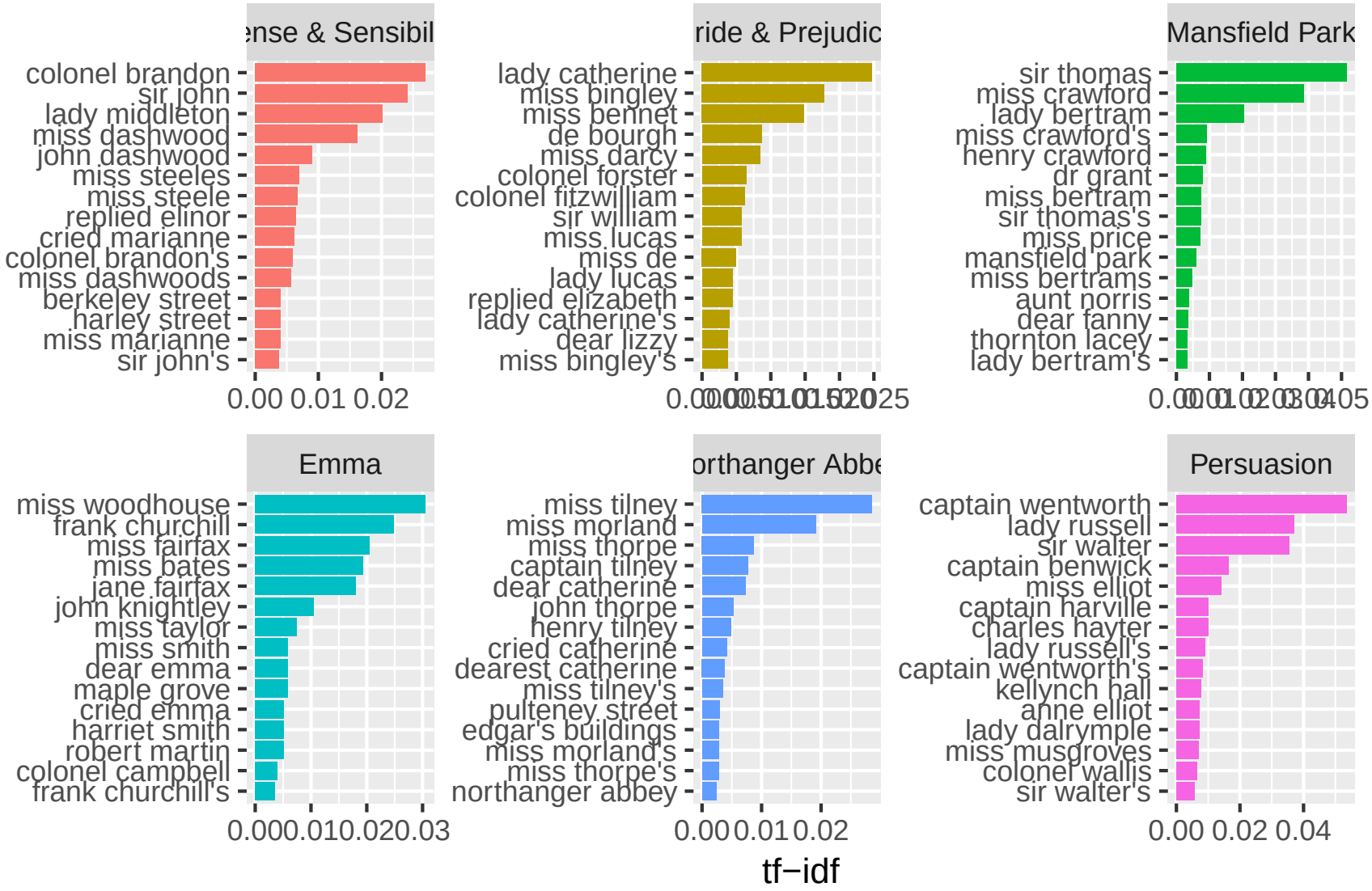
```
austen_bigrams_tf
```

```
## # A tibble: 36,217 x 6
##    book                bigram            n     tf   idf tf_idf
##    <fct>               <chr>         <int>  <dbl> <dbl>  <dbl>
##  1 Persuasion          captain went~   170 0.0299  1.79 0.0535
##  2 Mansfield Park      sir thomas      287 0.0287  1.79 0.0515
##  3 Mansfield Park      miss crawford   215 0.0215  1.79 0.0386
##  4 Persuasion          lady russell    118 0.0207  1.79 0.0371
##  5 Persuasion          sir walter      113 0.0198  1.79 0.0356
##  6 Emma                miss woodhou~   162 0.0170  1.79 0.0305
##  7 Northanger Abbey    miss tilney      82 0.0159  1.79 0.0286
##  8 Sense & Sensibi~    colonel bran~   108 0.0150  1.79 0.0269
##  9 Emma                frank church~   132 0.0139  1.79 0.0248
## 10 Pride & Prejudi~    lady catheri~   100 0.0138  1.79 0.0247
## # ... with 36,207 more rows
```

Visualize bigrams by tf-idf

```
austen_bigrams_tf %>%
  arrange(desc(tf_idf)) %>%
  mutate(bigram = factor(bigram, levels = rev(unique(bigram)))) %>%
  group_by(book) %>%
  top_n(15) %>%
  ungroup() %>%
  ggplot(aes(bigram, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 3, scales = "free") +
  coord_flip()
```

```
## Selecting by tf_idf
```



Analyzing bigrams with graphs

Rather than visualize the most common bigrams by their counts or term frequencies, we can instead visualize them as a graph. Here, we use "graph" not in the sense of visualization, but to refer to a network of connected nodes.

A graph can be constructed from a tidy object by considering the *edges* between nodes as the observations (rows), and the variables (columns) are:

- **from** the node that the edge is coming from
- **to** the node that the edge is going toward
- **weight** the weight of the edge

In our case, each bigram represents an edge in the network, each word is a node, and the weights are the bigram counts.

We can use the `igraph` package to construct the graph.

Creating a graph from bigrams

```
library(igraph)
austen_graph <- austen_bigrams %>%
  count(word1, word2, sort=TRUE) %>%
  filter(n > 20) %>%
  graph_from_data_frame()

austen_graph
```

```
## IGRAPH 77e17f6 DN-- 91 77 --
## + attr: name (v/c), n (e/n)
## + edges from 77e17f6 (vertex names):
##  [1] sir     ->thomas    miss    ->crawford
##  [3] captain ->wentworth miss    ->woodhouse
##  [5] frank   ->churchill lady    ->russell
##  [7] lady    ->bertram   sir     ->walter
##  [9] miss    ->fairfax   colonel ->brandon
## [11] miss    ->bates     lady    ->catherine
## [13] sir     ->john      jane    ->fairfax
## [15] miss    ->tilney    lady    ->middleton
## + ... omitted several edges
```

Visualizing n-gram graphs

We can use the ggraph package to visualize the graph of bigrams that we just made.

```
library(ggraph)
```

```
set.seed(1)
ggraph(austen_graph) +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```

```
## Using `nicely` as default layout
```