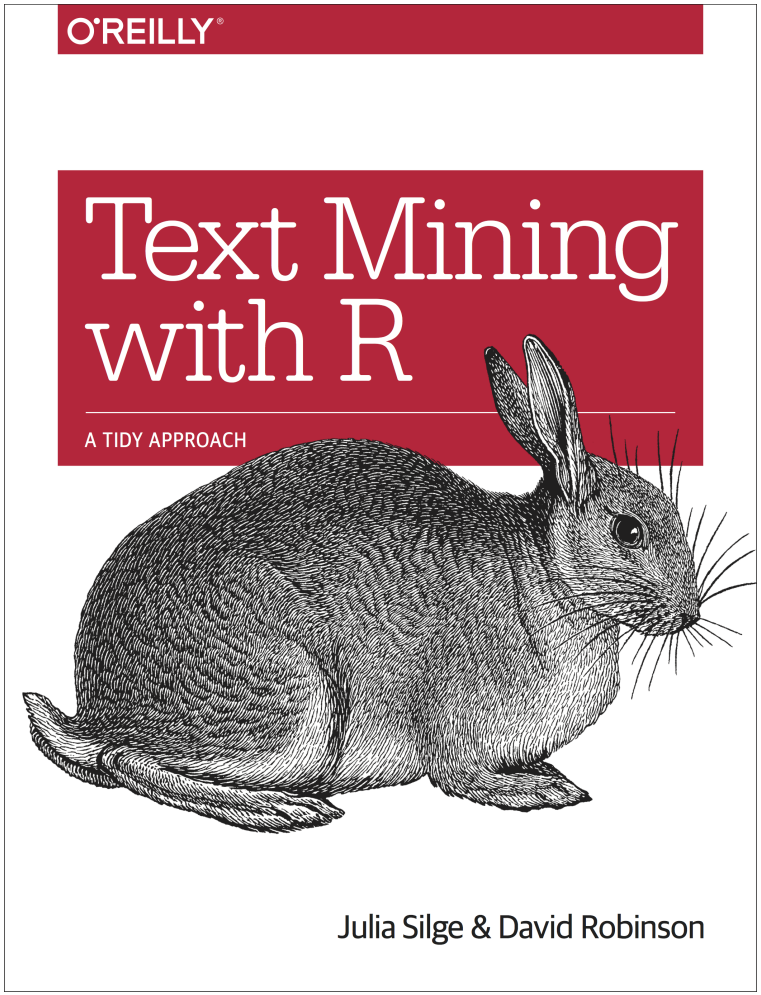
Socio-Informatics 348

Text Analysis
Relations between Words & Format Conversion

Dr Lisa Martin

Department of Information Science
Stellenbosch University

Today's Reading



Text Mining with R, Chapters 4 & 5

Moving Beyond Individual Words

Why Word Relationships Matter

So far: Words as individual units

- Relationships to sentiments
- Relationships to documents

New focus: Relationships between words

- Which words tend to follow others immediately?
- Which words tend to co-occur within documents?

New tools:

- `ggraph`: network visualizations
- `widyr`: pairwise correlations in tidy data

Tokenizing by N-gram

N-gram: Consecutive sequence of n words

Types:

- Bigram (n=2): pairs of consecutive words
- Trigram (n=3): sequences of three words
- Generally: n consecutive words

Use case: Build models of relationships by examining how often word X is followed by word Y

Creating Bigrams with unnest_tokens()

```
library(dplyr)
library(tidytext)
library(janeaustenr)

austen_bigrams <- austen_books() %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  filter(!is.na(bigram))

austen_bigrams
#> # A tibble: 662,792 × 2
#>    book                bigram
#>    <fct>               <chr>
#>  1 Sense & Sensibility sense and
#>  2 Sense & Sensibility and sensibility
#>  3 Sense & Sensibility by jane
#>  4 Sense & Sensibility jane austen
#>  5 Sense & Sensibility chapter 1
#>  6 Sense & Sensibility the family
```

Most Common Bigrams in Jane Austen

Problem: Most common bigrams are pairs of stop-words

```
austen_bigrams %>%
  count(bigram, sort = TRUE)
#> # A tibble: 193,212 × 2
#>    bigram       n
#>    <chr>    <int>
#>  1 of the    2853
#>  2 to be     2670
#>  3 in the    2221
#>  4 it was    1691
#>  5 i am      1485
#>  6 she had   1405
#>  7 of her    1363
#>  8 to the    1315
```

Solution: Filter out stop-words

Filtering Stop-words from Bigrams

```
library(tidyr)

bigrams_separated <- austen_bigrams %>%
  separate(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 = TRUE)

bigram_counts
#> # A tibble: 28,971 × 3
#>    word1   word2         n
#>    <chr>   <chr>     <int>
#>  1 sir     thomas      266
#>  2 miss    crawford    196
#>  3 captain wentworth   143
```

Result: Names and titles dominate ("sir thomas", "miss crawford")

unite() to Recombine Bigrams

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

bigrams_united
#> # A tibble: 38,910 × 2
#>    book                bigram
#>    <fct>               <chr>
#>  1 Sense & Sensibility jane austen
#>  2 Sense & Sensibility chapter 1
#>  3 Sense & Sensibility norland park
#>  4 Sense & Sensibility surrounding acquaintance
#>  5 Sense & Sensibility late owner
#>  6 Sense & Sensibility advanced age
#>  7 Sense & Sensibility constant companion
#>  8 Sense & Sensibility happened ten
#>  9 Sense & Sensibility henry dashwood
#> 10 Sense & Sensibility norland estate
#> # i 38,900 more rows
```

Extending to Trigrams

```
austen_books() %>%
  unnest_tokens(trigram, text, token = "ngrams", n = 3) %>%
  filter(!is.na(trigram)) %>%
  separate(trigram, c("word1", "word2", "word3"), sep = " ") %>%
  filter(!word1 %in% stop_words$word,
         !word2 %in% stop_words$word,
         !word3 %in% stop_words$word) %>%
  count(word1, word2, word3, sort = TRUE)
#> # A tibble: 6,139 × 4
#>    word1     word2     word3         n
#>    <chr>     <chr>     <chr>     <int>
#>  1 dear      miss      woodhouse    20
#>  2 miss      de        bourgh       17
#>  3 lady      catherine de           11
#>  4 poor      miss      taylor       11
#>  5 sir       walter    elliot       10
#>  6 catherine de        bourgh        9
#>  7 dear      sir       thomas        8
#>  8 replied   miss      crawford      7
#>  9 sir       william   lucas         7
#> 10 ten       thousand  pounds        7
#> # ℹ 6,129 more rows
```

Example Analysis: Streets in Austen

Bigrams enable domain-specific queries

```
bigrams_filtered %>%
  filter(word2 == "street") %>%
  count(book, word1, sort = TRUE)
#> # A tibble: 33 × 3
#>    book                word1          n
#>    <fct>               <chr>      <int>
#>  1 Sense & Sensibility harley         16
#>  2 Sense & Sensibility berkeley       15
#>  3 Northanger Abbey    milsom         10
#>  4 Northanger Abbey    pulteney       10
#>  5 Mansfield Park      wimpole         9
#>  6 Pride & Prejudice   gracechurch     8
#>  7 Persuasion          milsom          5
#>  8 Sense & Sensibility bond            4
#>  9 Sense & Sensibility conduit         4
#> 10 Persuasion          rivers          4
#> # ℹ 23 more rows
```

Bigram TF-IDF

```
bigram_tf_idf <- bigrams_united %>%
  count(book, bigram) %>%
  bind_tf_idf(bigram, book, n) %>%
  arrange(desc(tf_idf))
```

```
bigram_tf_idf
#> # A tibble: 31,388 × 6
#>    book                bigram            n     tf   idf tf_idf
#>    <fct>               <chr>         <int>  <dbl> <dbl>  <dbl>
#>  1 Mansfield Park      sir thomas      266 0.0304  1.79 0.0546
#>  2 Persuasion          captain wentworth 143 0.0290  1.79 0.0519
#>  3 Mansfield Park      miss crawford   196 0.0224  1.79 0.0402
#>  4 Persuasion          lady russell    110 0.0223  1.79 0.0399
#>  5 Persuasion          sir walter      108 0.0219  1.79 0.0392
#>  6 Emma                miss woodhouse  143 0.0173  1.79 0.0309
#>  7 Northanger Abbey    miss tilney      74 0.0165  1.79 0.0295
#>  8 Sense & Sensibility colonel brandon  96 0.0155  1.79 0.0278
#>  9 Sense & Sensibility sir john         94 0.0152  1.79 0.0273
#> 10 Pride & Prejudice   lady catherine   87 0.0139  1.79 0.0248
#> # i 31,378 more rows
```

Bigrams and Sentiment Context

Problem with word-level sentiment: Context matters!

Example: "I'm not happy and I don't like it!"

- Word-level: "happy" (positive), "like" (positive)
- Reality: Sentence is negative

Solution: Use bigrams to detect negation
Common negation words: "not", "no", "never", "without"

Finding Negated Sentiment Words

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

not_words
#> # A tibble: 229 × 3
#>    word2   value     n
#>    <chr>   <dbl> <int>
#>  1 like        2    95
#>  2 help        2    77
#>  3 want        1    41
#>  4 wish        1    39
#>  5 allow       1    30
#>  6 care        2    21
#>  7 sorry      -1    20
#>  8 leave      -1    17
#>  9 pretend    -1    17
#> 10 worth       2    17
#> # ℹ 219 more rows
```

Finding Negated Sentiment Words

Most common sentiment words after "not":

- "like" (normally +2) - 95 times
- "help" (normally +2) - 77 times
- "want" (normally +1) - 41 times

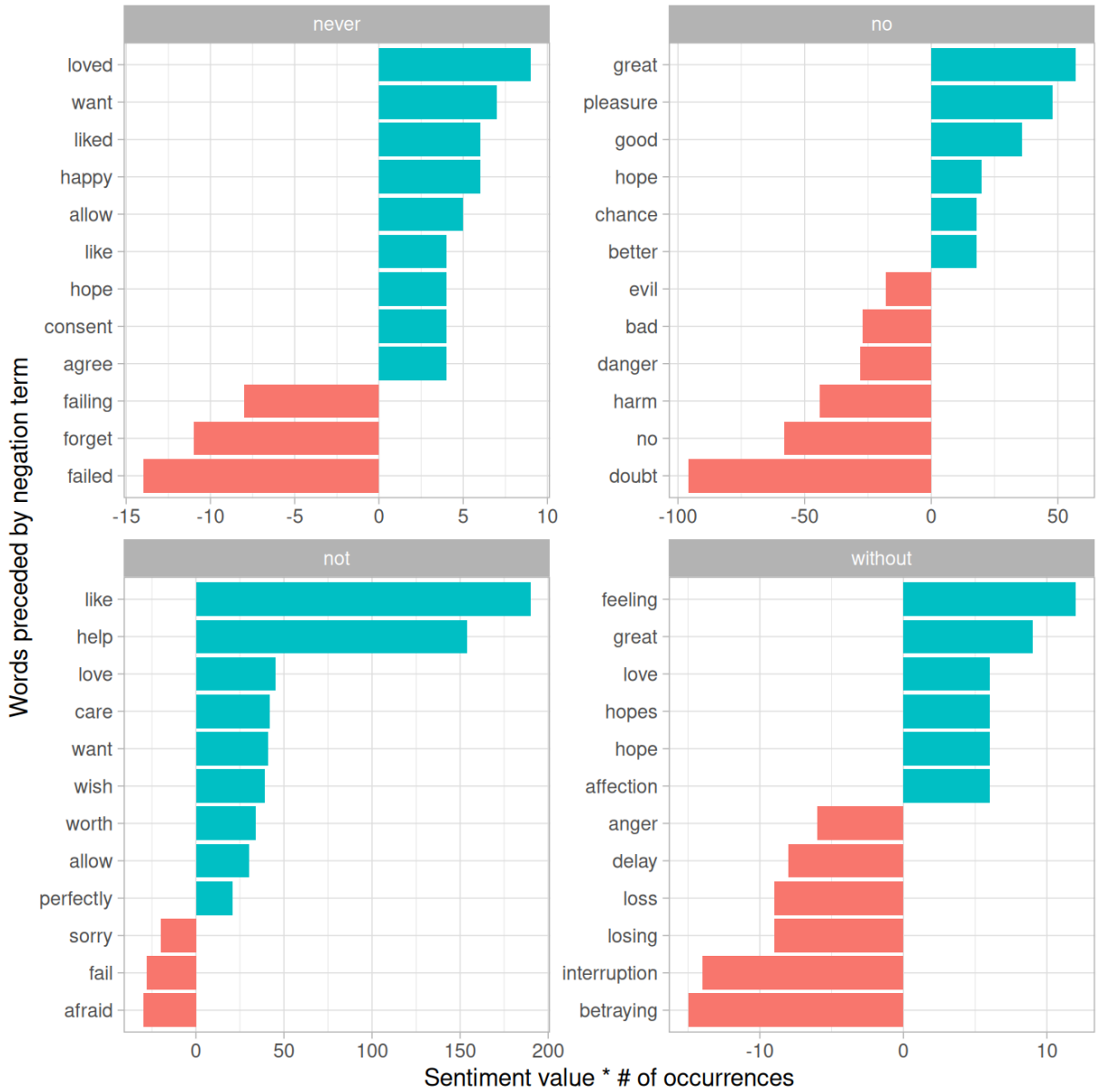
These contribute in the "wrong" direction to sentiment scores!

Biggest misidentifications:

- "not like" and "not help": make text seem more positive
- "not afraid" and "not fail": make text seem more negative

```
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, value, sort = TRUE)
```

Finding Negated Sentiment Words



Visualising Bigram Networks

Goal: Visualise all relationships simultaneously

Approach: Network graph with:

- **Nodes:** Words
- **Edges:** Connections between words that appear together
- **Weight:** Frequency of co-occurrence

Tools:

- `igraph`: create network objects
- `ggraph`: visualise with ggplot2 grammar

Creating a Bigram Network

`bigram_counts` from earlier (`word1, word2, n`)

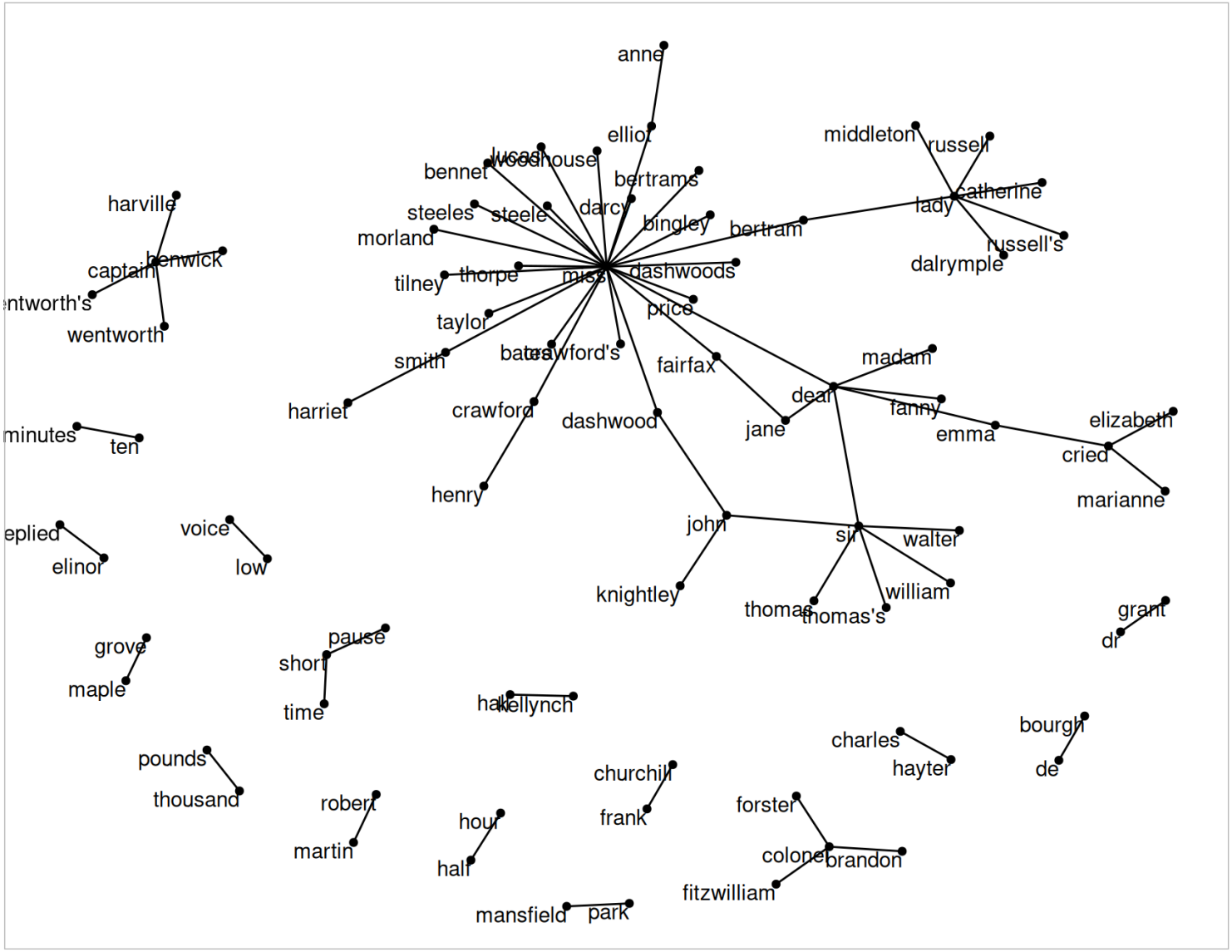
```
# filter for only relatively common combinations
bigram_graph <- bigram_counts %>%
  filter(n > 20) %>%
  graph_from_data_frame()

bigram_graph
#> IGRAPH c1d46ae DN-- 85 70 --
#> + attr: name (v/c), n (e/n)
#> + edges from c1d46ae (vertex names):
#>  [1] sir     ->thomas     miss    ->crawford   captain ->wentworth
#>  [4] miss    ->woodhouse  frank   ->churchill  lady    ->russell
#>  [7] sir     ->walter     lady    ->bertram    miss    ->fairfax
#> [10] colonel ->brandon    sir     ->john       miss    ->bates
#> [13] jane    ->fairfax    lady    ->catherine  lady    ->middleton
#> [16] miss    ->tilney     miss    ->bingley    thousand->pounds
#> [19] miss    ->dashwood   dear    ->miss       miss    ->bennet
#> [22] miss    ->morland    captain ->benwick    miss    ->smith
#> + ... omitted several edges
```

Creating a Bigram Network

```
library(ggraph)
set.seed(2017)

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```

`layout = "fr"` specifies the Fruchterman–Reingold force-directed layout, which positions nodes so that connected nodes are close together, while minimizing edge crossings.

Creating a Bigram Network



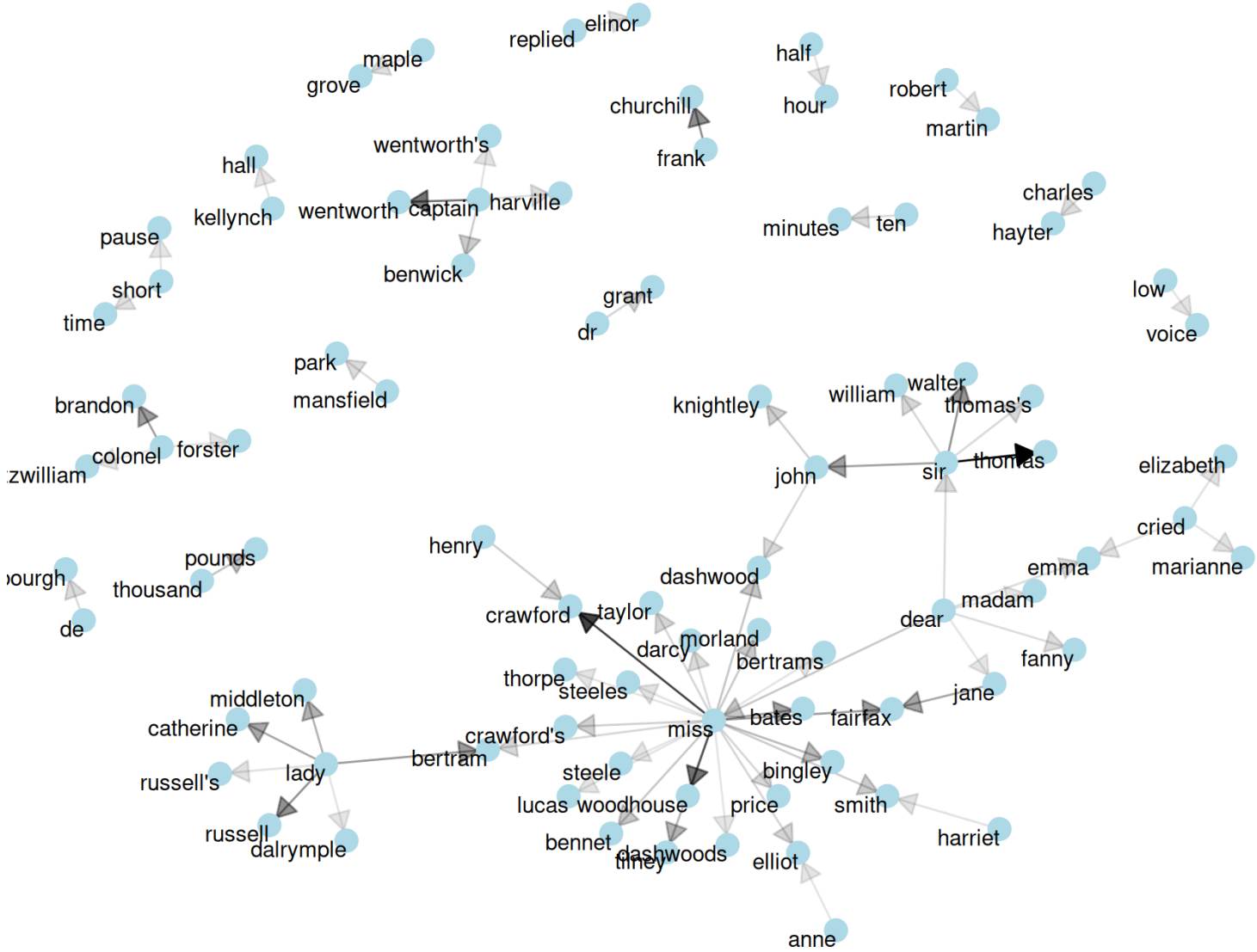
Improving Network Visualisations

Enhancements:

- `edge_alpha`: transparency based on frequency
- `arrow`: add directionality
- `end_cap`: prevent arrows from overlapping nodes
- `theme_void()`: clean network theme

```
set.seed(2020)

a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void()
```

Improving Network Visualisations



Counting Word Pairs with widyr

Different question: What about words that co-occur but aren't adjacent?

Challenge: Comparing between rows in tidy data

Solution: `widyr` package
Philosophy: "Widen data, perform operation, re-tidy"

1. Cast tidy data to wide matrix
2. Perform operation (count, correlate)
3. Return to tidy format

Pairwise Word Counting

Divide text into sections, count word co-occurrences:

```
austen_section_words <- austen_books() %>%
  filter(book == "Pride & Prejudice") %>%
  mutate(section = row_number() %/% 10) %>%
  filter(section > 0) %>%
  unnest_tokens(word, text) %>%
  filter(!word %in% stop_words$word)

austen_section_words
#> # A tibble: 37,240 × 3
#>    book              section word
#>    <fct>               <dbl> <chr>
#>  1 Pride & Prejudice       1 truth
#>  2 Pride & Prejudice       1 universally
#>  3 Pride & Prejudice       1 acknowledged
#>  4 Pride & Prejudice       1 single
#>  5 Pride & Prejudice       1 possession
#>  6 Pride & Prejudice       1 fortune
#>  7 Pride & Prejudice       1 wife
#>  8 Pride & Prejudice       1 feelings
```

Most Common Word Pairs

Most common pairs are main character names:

```
library(widyr)

# count words co-occuring within sections
word_pairs <- austen_section_words %>%
  pairwise_count(word, section, sort = TRUE)

word_pairs
#> # A tibble: 796,008 × 3
#>    item1     item2         n
#>    <chr>     <chr>     <dbl>
#>  1 darcy     elizabeth   144
#>  2 elizabeth darcy       144
#>  3 miss      elizabeth   110
#>  4 elizabeth miss        110
#>  5 elizabeth jane        106
```

Most Common Word Pairs

Note: Results are symmetric (both "elizabeth/darcy" and "darcy/elizabeth")

```
word_pairs %>%
  filter(item1 == "darcy")
#> # A tibble: 2,930 × 3
#>    item1 item2         n
#>    <chr> <chr>     <dbl>
#>  1 darcy elizabeth   144
#>  2 darcy miss         92
#>  3 darcy bingley      86
#>  4 darcy jane         46
#>  5 darcy bennet       45
#>  6 darcy sister       45
#>  7 darcy time         41
#>  8 darcy lady         38
#>  9 darcy friend       37
#> 10 darcy wickham      37
#> # ℹ 2,920 more rows
```

From Counting to Correlation

Problem: Most common pairs are also individually common words

Better metric: Correlation

- How often words appear together
- Relative to how often they appear separately

Phi coefficient: Correlation measure for binary data

$$\phi = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{n_{1\cdot}n_{0\cdot}n_{\cdot 0}n_{\cdot 1}}}$$

Where n_{11} = both appear, n_{00} = neither appears, etc.

Using pairwise_cor()

```
# we need to filter for at least relatively common words first
word_cors <- austen_section_words %>%
  group_by(word) %>%
  filter(n() >= 20) %>%
  pairwise_cor(word, section, sort = TRUE)

word_cors
#> # A tibble: 154,842 × 3
#>    item1     item2     correlation
#>    <chr>     <chr>           <dbl>
#>  1 bourgh    de              0.951
#>  2 de        bourgh          0.951
#>  3 pounds    thousand        0.701
#>  4 thousand  pounds          0.701
#>  5 william   sir             0.664
#>  6 sir       william         0.664
#>  7 catherine lady            0.663
#>  8 lady      catherine       0.663
#>  9 forster   colonel         0.622
#> 10 colonel   forster         0.622
#> # ℹ 154,832 more rows
```

Using pairwise_cor()

Filter to find words commonly used together:

```
word_cors %>%
  filter(item1 == "pounds")
#> # A tibble: 393 × 3
#>    item1  item2     correlation
#>    <chr>  <chr>           <dbl>
#>  1 pounds thousand       0.701
#>  2 pounds ten            0.231
#>  3 pounds fortune        0.164
#>  4 pounds settled        0.149
#>  5 pounds wickham's      0.142
#>  6 pounds children       0.129
#>  7 pounds mother's       0.119
#>  8 pounds believed       0.0932
#>  9 pounds estate         0.0890
#> 10 pounds ready          0.0860
#> # i 383 more rows
```

Visualise Correlations as Networks

Same approach as bigrams:

- Use `ggraph` for visualisation
- Nodes = words
- Edges = correlations above threshold
- Edge transparency = correlation strength

Key difference: Relationships are *symmetric* (no arrows)

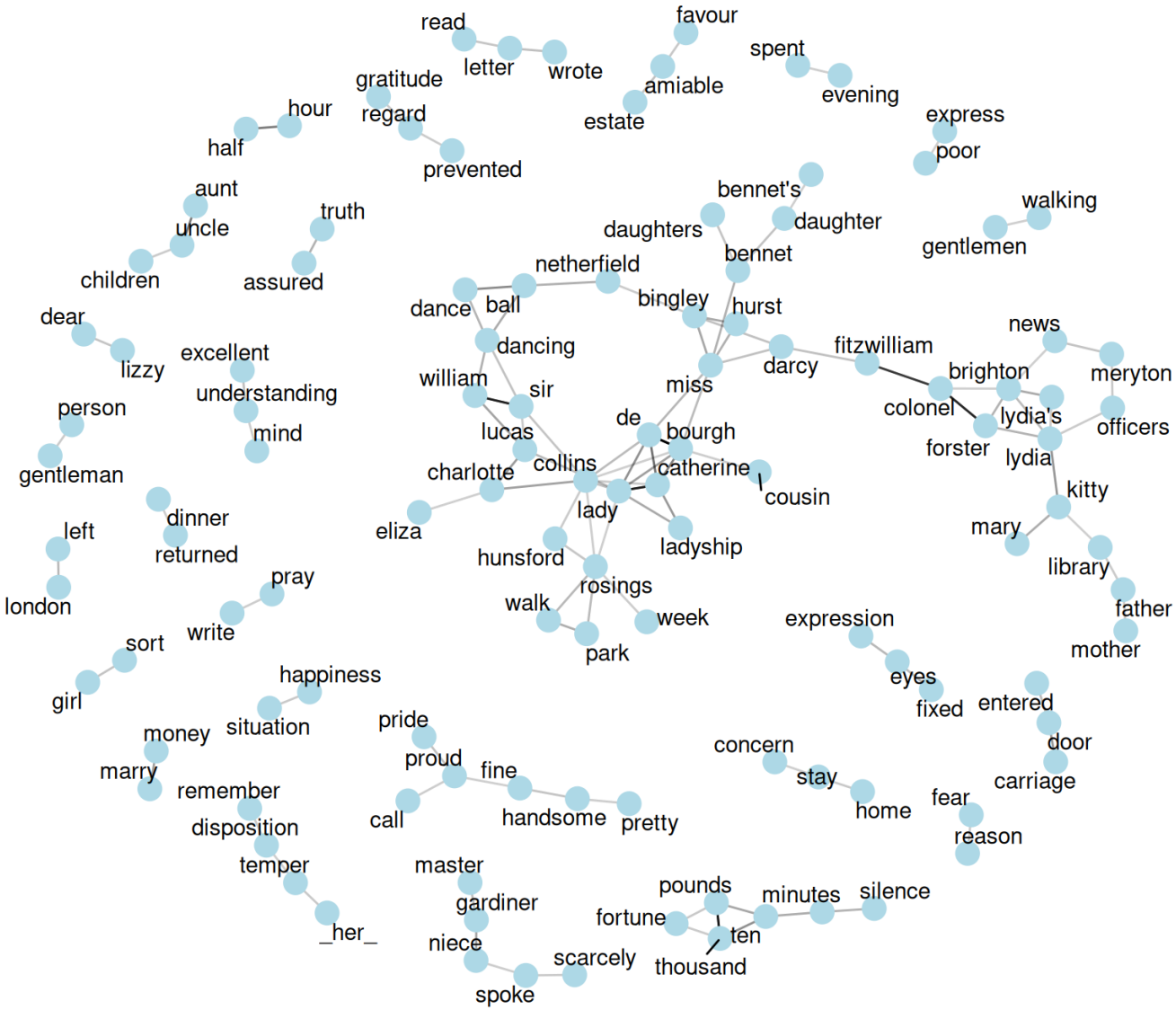
Reveals:

- Name/title pairings
- Proximity words: “walk” and “park”, “dance” and “ball”
- Thematic clusters in the text

Visualise Correlations as Networks

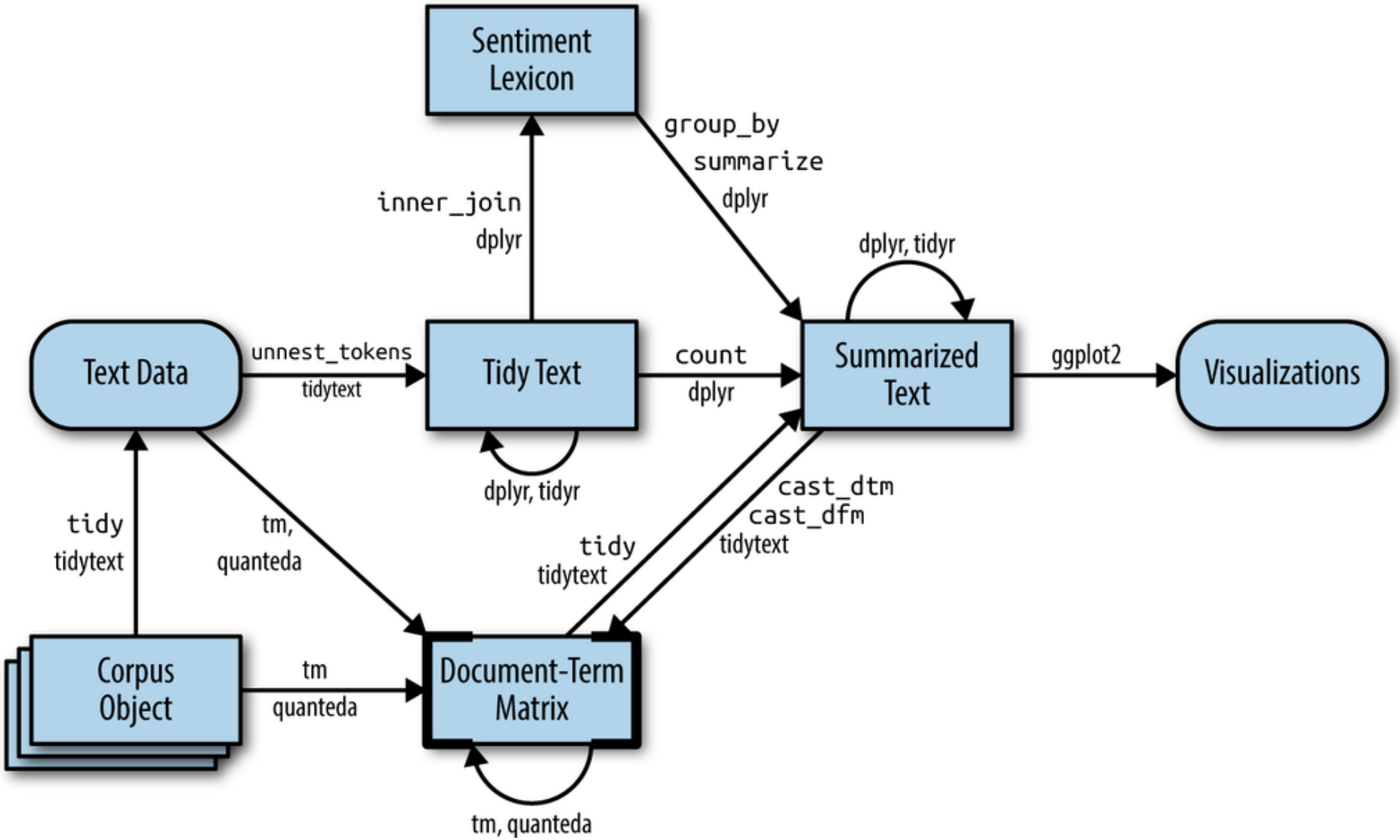
```
set.seed(2016)

word_cors %>%
  filter(correlation > .15) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = correlation), show.legend = FALSE) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void()
```

Visualise Correlations as Networks



Bridging Tidy and Traditional Text Mining

Converting to and from Non-Tidy Formats



Why Format Conversion Matters

Tidy text format: One-token-per-document-per-row

- Compatible with dplyr, tidyr, ggplot2
- Great for exploration and visualisation

Challenge: Most NLP packages use different formats

- Document-term matrices (DTM)
- Corpus objects
- Sparse matrices

Solution: Conversion "glue" between formats

Document-Term Matrix (DTM)

Structure:

- Rows = documents
- Columns = terms
- Values = term counts (or other weights)

Properties:

- Usually sparse (most values are zero)
- Efficient storage implementations
- Common input/output for text mining packages

Comparable to: Tidy data after `count()` or `group_by()`/`summarize()`

Document-Term Matrix (DTM)

```
library(tm)

data("AssociatedPress", package = "topicmodels")
AssociatedPress
#> <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
#> Non-/sparse entries: 302031/23220327
#> Sparsity           : 99%
#> Maximal term length: 18
#> Weighting          : term frequency (tf)
```

Converting DTM to Tidy Format

Function: `tidy()` from broom package

```
library(dplyr)
library(tidytext)

ap_td <- tidy(AssociatedPress)
ap_td
#> # A tibble: 302,031 × 3
#>    document term       count
#>       <int> <chr>      <dbl>
#>  1        1 adding         1
#>  2        1 adult          2
#>  3        1 ago            1
#>  4        1 alcohol        1
#>  5        1 allegedly      1
#>  6        1 allen          1
#>  7        1 apparently     2
#>  8        1 appeared       1
```

Document-Feature Matrix (quanteda)

Alternative DTM implementation: `dfm` from quanteda

```
data("data_corpus_inaugural", package = "quanteda")
inaug_dfm <- data_corpus_inaugural %>%
  quanteda::tokens() %>%
  quanteda::dfm(verbose = FALSE)
inaug_dfm
#> Document-feature matrix of: 59 documents, 9,437 features (91.84% sparse) a
```

Document-Feature Matrix (quanteda)

`tidy()` works the same way on different DTM formats!

```
inaug_td <- tidy(inaug_dfm)
inaug_td
#> # A tibble: 45,452 × 3
#>    document        term            count
#>    <chr>           <chr>           <dbl>
#>  1 1789-Washington fellow-citizens     1
#>  2 1797-Adams      fellow-citizens     3
#>  3 1801-Jefferson  fellow-citizens     2
#>  4 1809-Madison    fellow-citizens     1
#>  5 1813-Madison    fellow-citizens     1
#>  6 1817-Monroe     fellow-citizens     5
#>  7 1821-Monroe     fellow-citizens     1
#>  8 1841-Harrison   fellow-citizens    11
#>  9 1845-Polk       fellow-citizens     1
#> 10 1849-Taylor     fellow-citizens     1
#> # ℹ 45,442 more rows
```

Tidying Corpus Objects

Corpus: Collection of documents with metadata
Structure:

- List-like object
- Each item contains text + metadata
- Metadata: ID, date, author, language, etc.

```
data("acq")
acq
#> <<VCorpus>>
#> Metadata:  corpus specific: 0, document level (indexed): 0
#> Content:  documents: 50

# first document
acq[[1]]
#> <<PlainTextDocument>>
#> Metadata:  15
#> Content:  chars: 1287
```

Tidying Corpus Objects

```
acq_td <- tidy(acq)
acq_td
#> # A tibble: 50 × 16
#>    author   datetimestamp       description heading id    language origin
#>    <chr>    <dttm>              <chr>       <chr>   <chr> <chr>    <chr>
#>  1 <NA>     1987-02-26 15:18:06 ""          COMPUT… 10    en       Reute…
#>  2 <NA>     1987-02-26 15:19:15 ""          OHIO M… 12    en       Reute…
#>  3 <NA>     1987-02-26 15:49:56 ""          MCLEAN… 44    en       Reute…
#>  4 By Cal … 1987-02-26 15:51:17 ""          CHEMLA… 45    en       Reute…
#>  5 <NA>     1987-02-26 16:08:33 ""          <COFAB… 68    en       Reute…
#>  6 <NA>     1987-02-26 16:32:37 ""          INVEST… 96    en       Reute…
#>  7 By Patt… 1987-02-26 16:43:13 ""          AMERIC… 110   en       Reute…
#>  8 <NA>     1987-02-26 16:59:25 ""          HONG K… 125   en       Reute…
#>  9 <NA>     1987-02-26 17:01:28 ""          LIEBER… 128   en       Reute…
#> 10 <NA>     1987-02-26 17:08:27 ""          GULF A… 134   en       Reute…
#> # i 40 more rows
#> # i 8 more variables: lewissplit <chr>, cgisplit <chr>, oldid <chr>,
#> #   places <named list>, people <lgl>, orgs <lgl>, exchanges <lgl>, text <
```

Tidying Corpus Objects

```
acq_tokens <- acq_td %>%
  select(-places) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word")

# most common words
acq_tokens %>%
  count(word, sort = TRUE)
#> # A tibble: 1,559 × 2
#>    word         n
#>    <chr>    <int>
#>  1 dlrs       100
#>  2 pct         70
#>  3 mln         65
#>  4 company     63
#>  5 shares      52
#>  6 reuter      50
```