Tidy Lyrical Analysis

An introduction to Tidytext through geniusR

Josiah Parry

2018/07/15

Loading [MathJax]/jax/output/CommonHTML/jax.js

Talk Overview

- Who I am
- What does it mean to be tidy
- `geniusR`
- `tidytext`
- Tidytext Analysis With geniusR
- Give it a go!
    - Workshopping tidy text analysis

Loading [MathJax]/jax/output/CommonHTML/jax.js

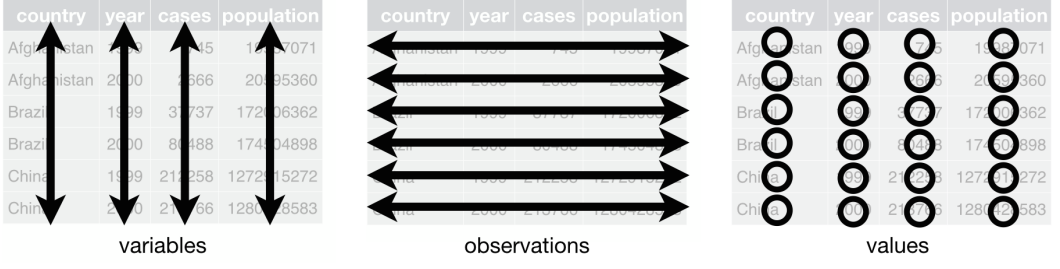
Some things about me

- Plymouth State '18, Sociology, GIS, Math
- Esspresso connoisseur
- NH Data Manager @ NextGen America #youthvote
  - Remember to vote Nov. 6th!
- MS Urban Informatics at Northeastern in September

Loading [MathJax]/jax/output/CommonHTML/jax.js

What does it mean to be tidy?

- I will refer you to this beautiful paper by Hadley Wickham
- The short of it is that:
  - Each variable is a column
  - Each observation is a row
  - Each type of observational unit is a table
- Does anyone have a definition?



Loading [MathJax]/jax/output/CommonHTML/jax.js

What does it mean to be tidy

Here we have a vector of text.

```
lyrics <- c("Where the black clouds never lingered",
            "The sunlight spread like honey",
            "Through my sister's tiny hands",
            "But while picking sour apples")

lyrics
```

```
## [1] "Where the black clouds never lingered"
## [2] "The sunlight spread like honey"
## [3] "Through my sister's tiny hands"
## [4] "But while picking sour apples"
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Packages!

```
library(tidyverse)
library(tidytext)
#devtools::install_github("josiahparry/geniusR")
library(geniusR)
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Tidy Song Lyrics

- Tidy data works in a table structure.
- We now have a tidy table where our unit is a line in a song.
- Ready for tidytext analysis
- Tidy text data like this is what inspired me to write `geniusR`.

```
lyrics_df <- tibble(lyric = lyrics,
                    line = 1:4)

lyrics_df
```

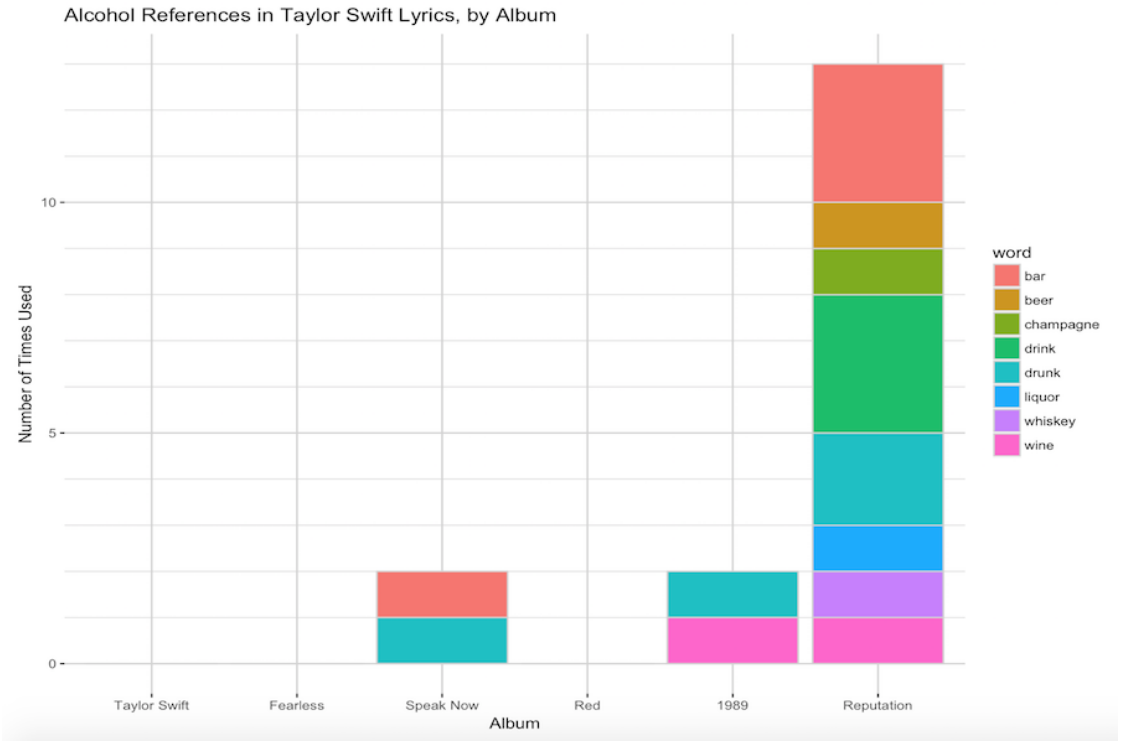
```
## # A tibble: 4 x 2
##   lyric                                 line
##   <chr>                                <int>
## 1 Where the black clouds never lingered     1
## 2 The sunlight spread like honey            2
## 3 Through my sister's tiny hands            3
## 4 But while picking sour apples             4
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

geniusR

Loading [MathJax]/jax/output/CommonHTML/jax.js

Taylor Swift Analysis



Loading [MathJax]/jax/output/CommonHTML/jax.js

geniusR

- Inspired by:
  - Text Mining with R
  - & Kendrick Lamar's pulitizer prize winning DAMN.
- Problem: No way I'm copying & pasting 14 songs' lyrics from the internet
- Solution: Spend way too long making a package to do just that.

Loading [MathJax]/jax/output/CommonHTML/jax.js

Guiding Principles

- ALL DATA SHOULD BE **TIDY** FROM THE GET GO
- Functions should be `%>%`-able, or `map()`-able
- Functions should be simple to understand and use

Note: if you feel a function acts otherwise, please submit an issue 😄

Loading [MathJax]/jax/output/CommonHTML/jax.js

`geniusR` functionality

This package serves 1 function:

- functionally provide you with song lyrics which are ready for tidy text analysis.

3 Key Functions:

- `genius_lyrics()`: gets a single song
- `genius_album()`: gets a whole album
- `add_genius()`: add lyrics to a tibble with a column of artist name and a column with album, or track names (one or the other)

Loading [MathJax]/jax/output/CommonHTML/jax.js

genius_lyrics()

```
anonanimal <- genius_lyrics(artist = "Andrew Bird", song = "Anonanimal")

head(anonanimal)
```

```
## # A tibble: 6 x 3
##   track_title lyric                                                line
##   <chr>       <chr>                                               <int>
## 1 Anonanimal  I see a sea anemone                                     1
## 2 Anonanimal  The enemy                                              2
## 3 Anonanimal  See a sea anemone                                      3
## 4 Anonanimal  And that'll be the end of me                           4
## 5 Anonanimal  While the vicious fish was caught unawares in the ten…     5
## 6 Anonanimal  Underneath her tender gills                            6
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

genius_album()

This will be the most useful as it returns whole albums in one call.

```
channel_orange <- genius_album("Frank Ocean", "channel ORANGE")
```

```
## Joining, by = c("track_title", "track_n", "track_url")
```

```
channel_orange
```

```
## # A tibble: 729 x 4
##    track_title      track_n lyric                                    line
##    <chr>              <int> <chr>                                   <int>
##  1 Start                  1 They look like twins                        1
##  2 Start                  1 That was embarrassing                       2
##  3 Thinkin Bout You       2 A tornado flew around my room before yo…    1
##  4 Thinkin Bout You       2 Excuse the mess it made, it usually doe…    2
##  5 Thinkin Bout You       2 Southern California, much like Arizona      3
##  6 Thinkin Bout You       2 My eyes don't shed tears, but boy, they…    4
##  7 Thinkin Bout You       2 I'm thinkin' 'bout you (Ooh no, no, no)     5
##  8 Thinkin Bout You       2 I've been thinkin' bout you (You know, …    6
##  9 Thinkin Bout You       2 I've been thinkin' bout you, do you thi…    7
## 10 Thinkin Bout You       2 Do ya, do ya?                               8
## # ... with 719 more rows
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

What songs are in the album?

```
channel_orange %>%
  distinct(track_title)
```

```
## # A tibble: 17 x 1
##    track_title
##    <chr>
##  1 Start
##  2 Thinkin Bout You
##  3 Fertilizer
##  4 Sierra Leone
##  5 Sweet Life
##  6 Not Just Money (Ft. Rosie Watson)
##  7 Super Rich Kids (Ft. Earl Sweatshirt)
##  8 Pilot Jones
##  9 Crack Rock
## 10 Pyramids
## 11 Lost
## 12 White (Album Version) (Ft. John Mayer)
## 13 Monks
## 14 Bad Religion
## 15 Pink Matter (Ft. André 3000)
## 16 Forrest Gump
## 17 End/Golden Girl (Ft. Tyler, The Creator)
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Which song is the longest (lyrically)?

```
channel_orange %>%
  count(track_title) %>%
  arrange(-n)
```

```
## # A tibble: 17 x 2
##    track_title                                    n
##    <chr>                                      <int>
##  1 End/Golden Girl (Ft. Tyler, The Creator)    86
##  2 Super Rich Kids (Ft. Earl Sweatshirt)       84
##  3 Pyramids                                    83
##  4 Monks                                       60
##  5 Pink Matter (Ft. André 3000)                57
##  6 Lost                                        54
##  7 Bad Religion                                50
##  8 Sweet Life                                  49
##  9 Forrest Gump                                45
## 10 Crack Rock                                  39
## 11 Pilot Jones                                 33
## 12 Thinkin Bout You                            33
## 13 Sierra Leone                                29
## 14 Not Just Money (Ft. Rosie Watson)           19
## 15 Fertilizer                                   5
## 16 Start                                        2
## 17 White (Album Version) (Ft. John Mayer)      1
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Let's get tidy (tokenization)

- Individual unit of speech or writing.
- `unigram == word`

```
orange_words <- channel_orange %>%
  unnest_tokens(word, lyric)

orange_words
```

```
## # A tibble: 4,605 x 4
##    track_title      track_n  line word
##    <chr>              <int> <int> <chr>
##  1 Start                  1     1 they
##  2 Start                  1     1 look
##  3 Start                  1     1 like
##  4 Start                  1     1 twins
##  5 Start                  1     2 that
##  6 Start                  1     2 was
##  7 Start                  1     2 embarrassing
##  8 Thinkin Bout You       2     1 a
##  9 Thinkin Bout You       2     1 tornado
## 10 Thinkin Bout You       2     1 flew
## # ... with 4,595 more rows
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Most common words?

```
orange_words %>%
  count(word) %>%
  arrange(-n)
```

```
## # A tibble: 1,110 x 2
##    word      n
##    <chr> <int>
##  1 the     215
##  2 you     152
##  3 i       112
##  4 a        94
##  5 my       83
##  6 and      77
##  7 to       75
##  8 in       68
##  9 me       57
## 10 love     48
## # ... with 1,100 more rows
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

"the" & *"a"* are uniformative!

- These are *stop words*: words with little meaning
- We can remove them before an analysis to focus on *"important"* words

```
get_stopwords()
```

```
## # A tibble: 175 x 2
##    word      lexicon
##    <chr>     <chr>
##  1 i         snowball
##  2 me        snowball
##  3 my        snowball
##  4 myself    snowball
##  5 we        snowball
##  6 our       snowball
##  7 ours      snowball
##  8 ourselves snowball
##  9 you       snowball
## 10 your      snowball
## # ... with 165 more rows
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Removing Stop Words

We can focus our analysis on word that carry more meaning

```
orange_counts <- orange_words %>%
  anti_join(get_stopwords()) %>%
  count(word) %>%
  arrange(-n)
```

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

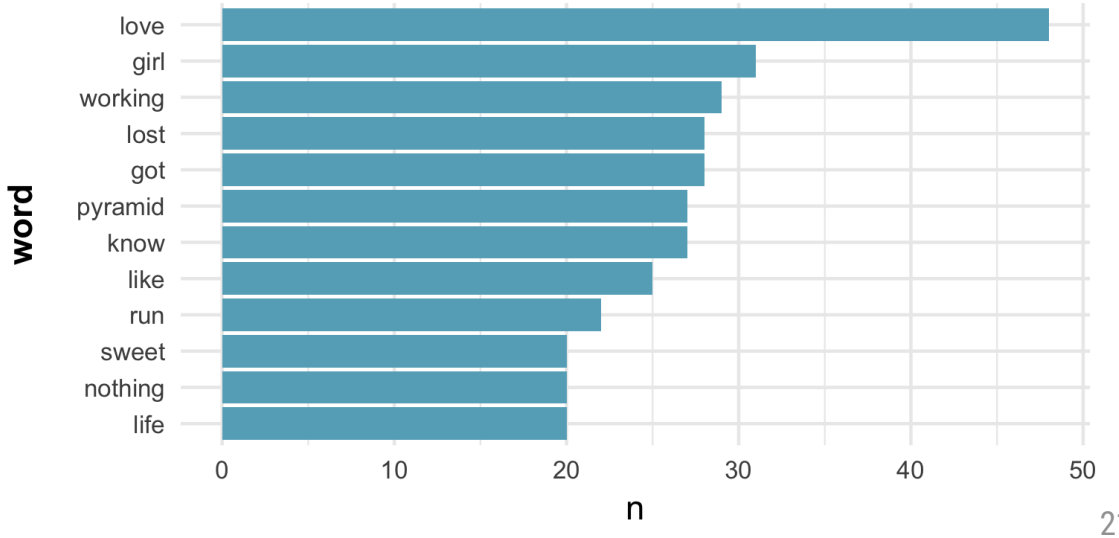
```
orange_counts
```

```
## # A tibble: 981 x 2
##    word        n
##    <chr>   <int>
##  1 love       48
##  2 girl       31
##  3 working    29
##  4 got        28
##  5 lost       28
##  6 know       27
##  7 pyramid    27
##  8 like       25
##  9 run        22
## 10 life       20
## # ... with 971 more rows
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Plot Common Words

```
orange_counts %>%
  top_n(10, n) %>%
  mutate(word = fct_reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  coord_flip() +
  geom_col() +
  my_theme()
```



Loading [MathJax]/jax/output/CommonHTML/jax.js

Sentiment

- Words have meaning and associations
- Sentiment is a feeling or emotion
- There are many attempts to quantify this in words

```
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
```

```
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
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Sentiment lexicons

- Lexicons don't take into account qualifiers (i.e. no, not, etc.)

```
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
```

```
get_sentiments("loughran")
```

```
## # A tibble: 4,149 x 2
##    word         sentiment
##    <chr>        <chr>
##  1 abandon      negative
##  2 abandoned    negative
##  3 abandoning   negative
##  4 abandonment  negative
##  5 abandonments negative
##  6 abandons     negative
##  7 abdicated    negative
##  8 abdicates    negative
##  9 abdicating   negative
## 10 abdication   negative
## # ... with 4,139 more rows
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Song Sentiment

- Evaluate a song as the sum of the sentiment of individual words (tokens)
- Obtain sentiment by joining sentiment to song lyrics

```
song_sentiment <- orange_words %>%
  inner_join(get_sentiments("bing")) %>%
  group_by(track_n) %>%
  count(sentiment, word)
```

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

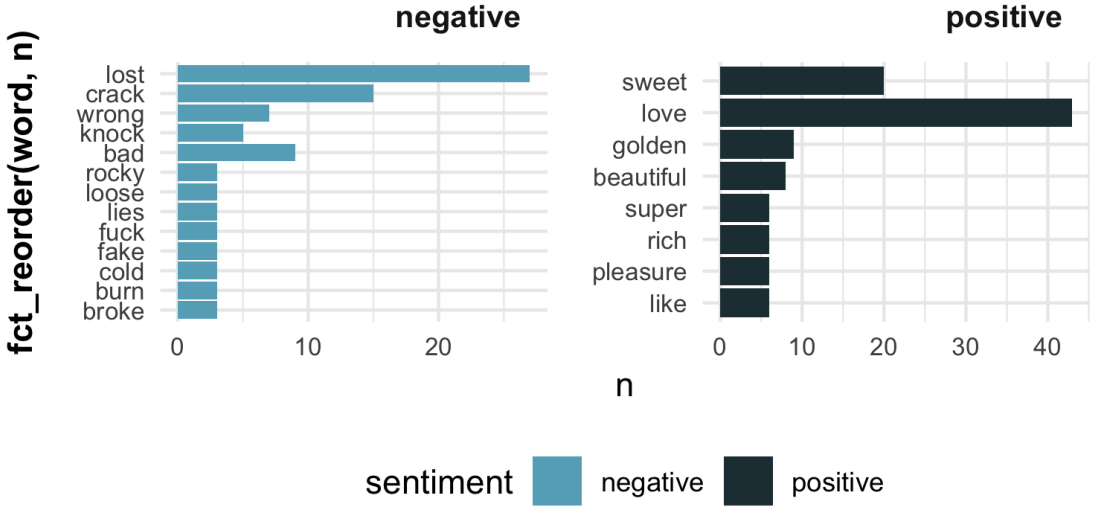
```
song_sentiment
```

```
## # A tibble: 193 x 4
## # Groups:   track_n [16]
##    track_n sentiment word            n
##      <int> <chr>     <chr>       <int>
##  1       1 negative  embarrassing     1
##  2       1 positive  like             1
##  3       2 negative  excuse           1
##  4       2 negative  lying            1
##  5       2 negative  mess             1
##  6       2 positive  cool             1
##  7       2 positive  cute             1
##  8       2 positive  enough           1
##  9       2 positive  like             2
## 10       2 positive  love             1
## # ... with 183 more rows
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Visualizing Song Sentiment

```
song_sentiment %>%
  group_by(sentiment) %>%
  top_n(10, n) %>%
  ggplot(aes(fct_reorder(word, n), n, fill = sentiment)) +
  geom_col() + coord_flip() +
  facet_wrap(~sentiment, scales = "free") + my_theme()
```



Loading [MathJax]/jax/output/CommonHTML/jax.js

Calculating Overall Song Sentiment

- Count of total positive & negative words per song

```
orange_words %>%
  inner_join(get_sentiments("bing")) %>% # again, identify our sentiment via join
  count(track_n, sentiment) # Count pos/neg words by song
```

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

## # A tibble: 31 x 3
##    track_n sentiment     n
##      <int> <chr>     <int>
##  1       1 negative      1
##  2       1 positive      1
##  3       2 negative      3
##  4       2 positive      6
##  5       3 negative      1
##  6       4 negative      2
##  7       4 positive      7
##  8       5 negative      4
##  9       5 positive     37
## 10       6 negative      2
## # ... with 21 more rows
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Calculating Overall Song Sentiment

- sometimes you need to make a mess to get clean

```
orange_words %>%
  inner_join(get_sentiments("bing")) %>% # again, identify our sentiment via join
  count(track_n, sentiment) %>% # Count pos/neg words by song
  spread(sentiment, n, fill = 0) # Create pos/neg count columns
```

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

## # A tibble: 16 x 3
##    track_n negative positive
##      <int>    <dbl>    <dbl>
##  1       1       1.       1.
##  2       2       3.       6.
##  3       3       1.       0.
##  4       4       2.       7.
##  5       5       4.      37.
##  6       6       2.       5.
##  7       7      26.      44.
##  8       8       5.       1.
##  9       9      33.       2.
## 10      10      12.      20.
## 11      11      39.      20.
## 12      13       5.      22.
## 13      14       9.      23.
## 14      15      20.      19.
## 15      16       6.       3.
## 16      17      13.      28.
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Calculating Overall Song Sentiment

```
orange_sentiment <- orange_words %>%
  inner_join(get_sentiments("bing")) %>% # again, identify our sentiment via join
  count(track_n, sentiment) %>% # Count pos/neg words by song
  spread(sentiment, n, fill = 0) %>% # Create pos/neg count columns
  mutate(sentiment = positive - negative)  # calculate sentiment
```

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

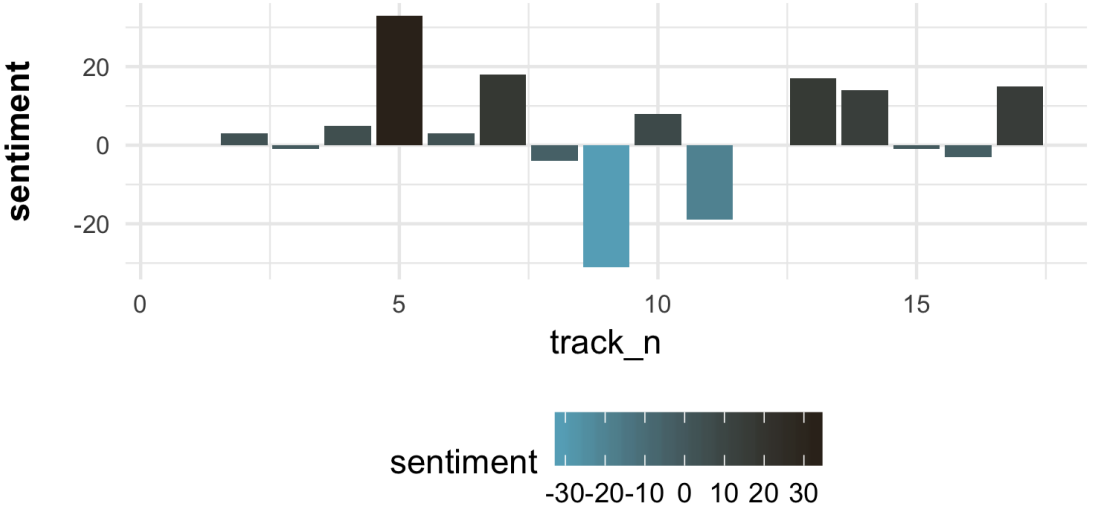
```
head(orange_sentiment)
```

```
## # A tibble: 6 x 4
##   track_n negative positive sentiment
##     <int>    <dbl>    <dbl>     <dbl>
## 1       1       1.       1.        0.
## 2       2       3.       6.        3.
## 3       3       1.       0.       -1.
## 4       4       2.       7.        5.
## 5       5       4.      37.       33.
## 6       6       2.       5.        3.
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Visualize Sentiment Over an Album

```
orange_sentiment %>%
  ggplot(aes(track_n, sentiment, fill = sentiment)) +
  geom_col() +
  my_theme()
```



Loading [MathJax]/jax/output/CommonHTML/jax.js

Comparing Artists: add_genius()

- Used to build on a data frame with artist and album/track information

```
albums <- tribble(~artist, ~album,
                  "Beyonce", "Lemonade",
                  "Andrew Bird", "Noble Beast",
                  "Drive By Truckers", "American Band") %>%
  add_genius(artist, album, "album")

head(albums)
```

```
## # A tibble: 6 x 6
##   artist  album    track_title       track_n lyric                    line
##   <chr>   <chr>    <chr>               <int> <chr>                   <int>
## 1 Beyonce Lemonade PRAY YOU CATCH ME       1 You can taste the dish…     1
## 2 Beyonce Lemonade PRAY YOU CATCH ME       1 It's all over your bre…     2
## 3 Beyonce Lemonade PRAY YOU CATCH ME       1 But even that's a test      3
## 4 Beyonce Lemonade PRAY YOU CATCH ME       1 Constantly aware of it…     4
## 5 Beyonce Lemonade PRAY YOU CATCH ME       1 My lonely ear pressed …     5
## 6 Beyonce Lemonade PRAY YOU CATCH ME       1 Pray to catch you whis…     6
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Which album has the most lines?

```
albums %>%
  unnest_tokens(word, lyric) %>%
  count(artist) %>%
  arrange(-n)
```

```
## # A tibble: 3 x 2
##   artist               n
##   <chr>            <int>
## 1 Beyonce           4598
## 2 Drive By Truckers  2822
## 3 Andrew Bird        1959
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Most Common Words? (w/ stop words)

```
albums %>%
  unnest_tokens(word, lyric) %>%
  count(artist, word) %>%
  arrange(-n) %>%
  head()
```

```
## # A tibble: 6 x 3
##   artist            word      n
##   <chr>             <chr> <int>
## 1 Beyonce           i       236
## 2 Beyonce           you     233
## 3 Drive By Truckers the     178
## 4 Andrew Bird       the     116
## 5 Beyonce           me      102
## 6 Beyonce           and     100
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Most Common Words? (w/out stop words)

```
album_words <- albums %>%
  unnest_tokens(word, lyric) %>%
  anti_join(get_stopwords()) %>%
  count(artist, word) %>%
  arrange(-n)
```

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

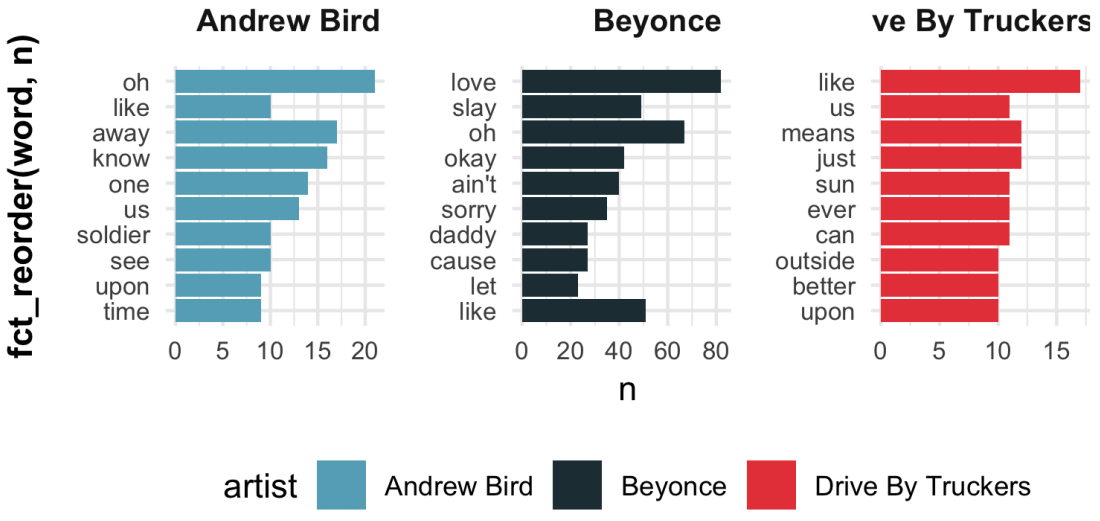
```
head(album_words)
```

```
## # A tibble: 6 x 3
##   artist  word      n
##   <chr>   <chr> <int>
## 1 Beyonce love     82
## 2 Beyonce oh       67
## 3 Beyonce like     51
## 4 Beyonce slay     49
## 5 Beyonce okay     42
## 6 Beyonce ain't    40
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Visualizing Most Common Words

```
album_words %>%
  group_by(artist) %>%
  top_n(10, n) %>%
  ggplot(aes(fct_reorder(word, n), n, fill = artist)) +
  geom_col() + my_theme() +
  coord_flip() + facet_wrap(~artist, scales = "free")
```



Loading [MathJax]/jax/output/CommonHTML/jax.js

Sentiment!

```
album_sentiment <- albums %>%
  unnest_tokens(word, lyric) %>%
  anti_join(get_stopwords()) %>%
  inner_join(get_sentiments("bing"))
```

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

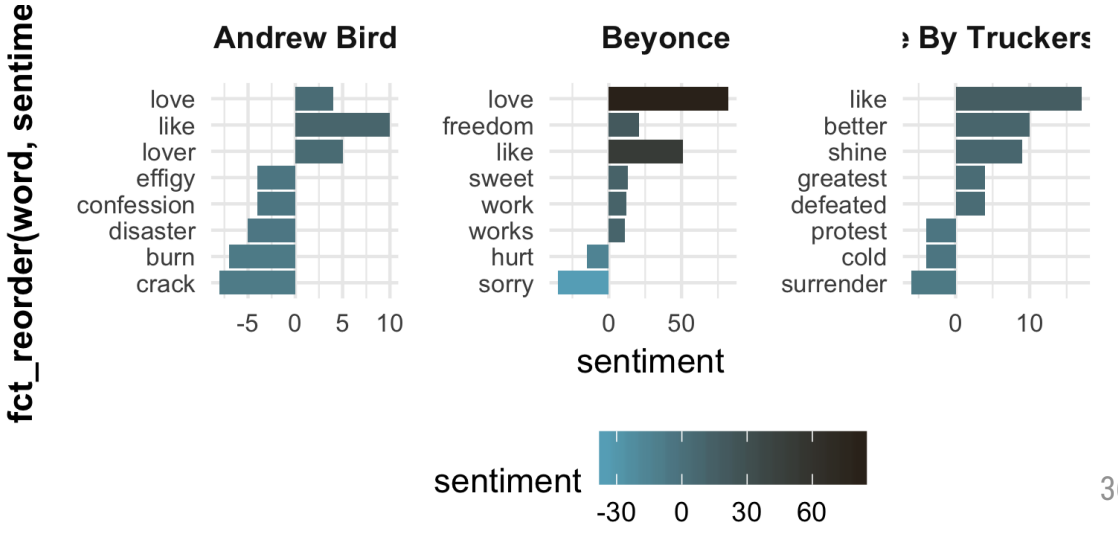
```
album_sentiment
```

```
## # A tibble: 890 x 7
##    artist  album    track_title       track_n  line word       sentiment
##    <chr>   <chr>    <chr>               <int> <int> <chr>      <chr>
##  1 Beyonce Lemonade PRAY YOU CATCH ME       1     1 dishonesty negative
##  2 Beyonce Lemonade PRAY YOU CATCH ME       1     5 lonely     negative
##  3 Beyonce Lemonade PRAY YOU CATCH ME       1    13 hurt       negative
##  4 Beyonce Lemonade PRAY YOU CATCH ME       1    13 like       positive
##  5 Beyonce Lemonade PRAY YOU CATCH ME       1    13 smile      positive
##  6 Beyonce Lemonade PRAY YOU CATCH ME       1    15 concern    negative
##  7 Beyonce Lemonade PRAY YOU CATCH ME       1    15 ease       positive
##  8 Beyonce Lemonade PRAY YOU CATCH ME       1    24 love       positive
##  9 Beyonce Lemonade HOLD UP                 2     1 love       positive
## 10 Beyonce Lemonade HOLD UP                 2     1 like       positive
## # ... with 880 more rows
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Visualizing Sentiment

```
album_sentiment %>%
  count(artist, sentiment, word) %>%
  spread(sentiment, n, fill = 0) %>% # Create pos/neg count columns
  mutate(sentiment = positive - negative) %>%  # calculate sentiment
  group_by(artist) %>%
  top_n(8, abs(sentiment)) %>%
  ggplot(aes(fct_reorder(word, sentiment), sentiment, fill = sentiment)) +
  geom_col() + my_theme() +
  coord_flip() +  facet_wrap(~artist, scales = "free")
```



Loading [MathJax]/jax/output/CommonHTML/jax.js

Documents

- Term frequency
  - Frequent words are upweighted
- Inverse document frequency
  - less frequent words are given more weight

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

- `tf-idf` is about comparing **documents** within a **collection**.
- Identifies tokens that are important, but not too common!

Loading [MathJax]/jax/output/CommonHTML/jax.js

Finding `tf-idf`

- `bind_tf_idf()`: does all of the hard work calculating `tf`, `idf`, & `tf_idf`
  - 4 arguments
  - `tbl`: the `%>%` takes care of it
  - `term`: Our token (word)
  - `document`: In this case, artist
  - `n`: The count of the word

Loading [MathJax]/jax/output/CommonHTML/jax.js

Finding `tf-idf`

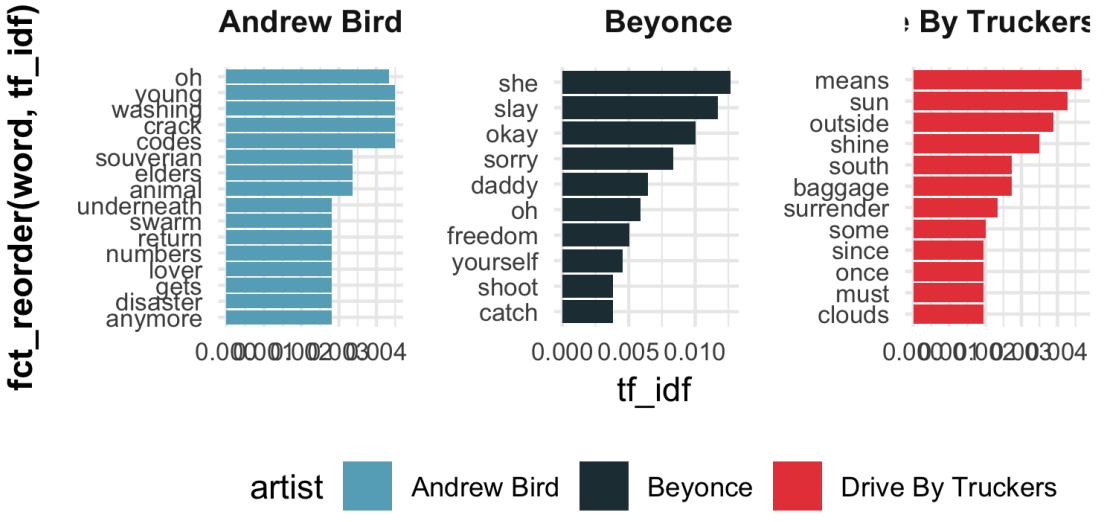
```
album_tf_idf <- albums %>%
  unnest_tokens(word, lyric) %>%
  count(artist, word) %>%
  bind_tf_idf(word, artist, n) %>%
  arrange(-tf_idf)

head(album_tf_idf)
```

```
## # A tibble: 6 x 6
##   artist  word      n      tf   idf  tf_idf
##   <chr>   <chr> <int>   <dbl> <dbl>   <dbl>
## 1 Beyonce she      53 0.0115  1.10  0.0127
## 2 Beyonce slay     49 0.0107  1.10  0.0117
## 3 Beyonce okay     42 0.00913 1.10  0.0100
## 4 Beyonce sorry    35 0.00761 1.10  0.00836
## 5 Beyonce daddy    27 0.00587 1.10  0.00645
## 6 Beyonce oh       67 0.0146  0.405 0.00591
```


Loading [MathJax]/jax/output/CommonHTML/jax.js

Visualizing `tf-idf`

```
album_tf_idf %>%
  group_by(artist) %>%
  top_n(10, tf_idf) %>%
  ggplot(aes(fct_reorder(word, tf_idf), tf_idf, fill = artist)) +
  geom_col() + my_theme() +
  coord_flip() + facet_wrap(~artist, scales = "free")
```



Loading [MathJax]/jax/output/CommonHTML/jax.js

Give it a try!

Loading [MathJax]/jax/output/CommonHTML/jax.js

Process

- Create a tibble with artist & album name
- `%>%` to `add_genius()`
- Tokenize
- Remove Stop Words
- Sentiment Analysis
  - add sentiment
  - count
  - spread & mutate
  - plot!
- `tf-idf`
  - count
  - `bind_tf_idf()`
  - plot!

Loading [MathJax]/jax/output/CommonHTML/jax.js

Loading [MathJax]/jax/output/CommonHTML/jax.js