# Sentiment analysis in R - Fast & Dirty: Polarity Scoring

Digital Humanities DSC 105 Spring 2023

Marcus Birkenkrahe

April 20, 2023

## README

- This lecture closely follows the DataCamp lesson "Sentiment analysis in R" by Ted Kwartler and the book "Text mining in Practice" by the same author.
- Download and open the practice file `10_polarity_practice.org` from GitHub to code along. You can also download it with `M-x eww~` using the URL `bit.ly/tm_polarity`.
- What you will learn:
  1. What is sentiment analysis
  2. What is polarity
  3. How to visualize polarity
  4. What are subjectivity lexicons
  5. What is Zipf's law
- Featured R packages:
  1. Sentiment scoring with `qdap::polarity`
  2. The archived `sentiment` analysis package
  3. Sentiment scoring with `tidytext`

## Free online tools

- I tested a free sentiment analysis tool, text2data.com with one of my own tweets:

## What is sentiment analysis?


Figure 1: "Two hearts that beat as one" - US Pres. Cleveland w/bride (1886)

- **Sentiment analysis** is the process of extracting an author's emotional intent from text.
- What are the challenges of sentiment analysis as defined this way?
    1. the human condition knows many different emotional states
    2. emotional states aren't always (ever?) clearly separated
    3. product reviews e.g. contain mixed emotions
    4. strong analyst or modeling bias
    5. why would you want to extract an author's emotion?
- A reductionist's fantasy? Plutchik (2001) believed that there are only 8 "evolutionary" created emotions:
    1. anger

2. fear
3. sadness
4. disgust
5. surprise
6. anticipation
7. trust
8. joy

- Plutchik's wheel of emotions is one of many frameworks:

## Scoring sentiment as polarity

- **Polarity** of a document is reduction of sentiment to positive, neutral or negative:
- Positive surprise: "I just found out I won the lottery!"
- Negative surprise: "I was just robbed at gunpoint!"
- Sentiment scoring does not always have a tangible value and the insights are not actionable: it is less valuable to say "that was negative" than "that was negative because of so-and-so."
- `qdap::polarity` is based on **subjectivity lexicons**, a list of words associated with emotional states:

## Subjectivity lexicons

- The lexicon used by `polarity` contains ca. 6,800 labeled words[1]
- It is important to understand the validity and size of any subjectivity lexicons used in polarity scoring (errors, bias).
- Simple approach: add up positive words in a passage and subtract the number of negative words. The net result is the sentiment score.
- Examples:

[1] Another popular (with 8,000 terms a little larger) lexicon is the MPQA lexicon from the University of Pittsburgh.

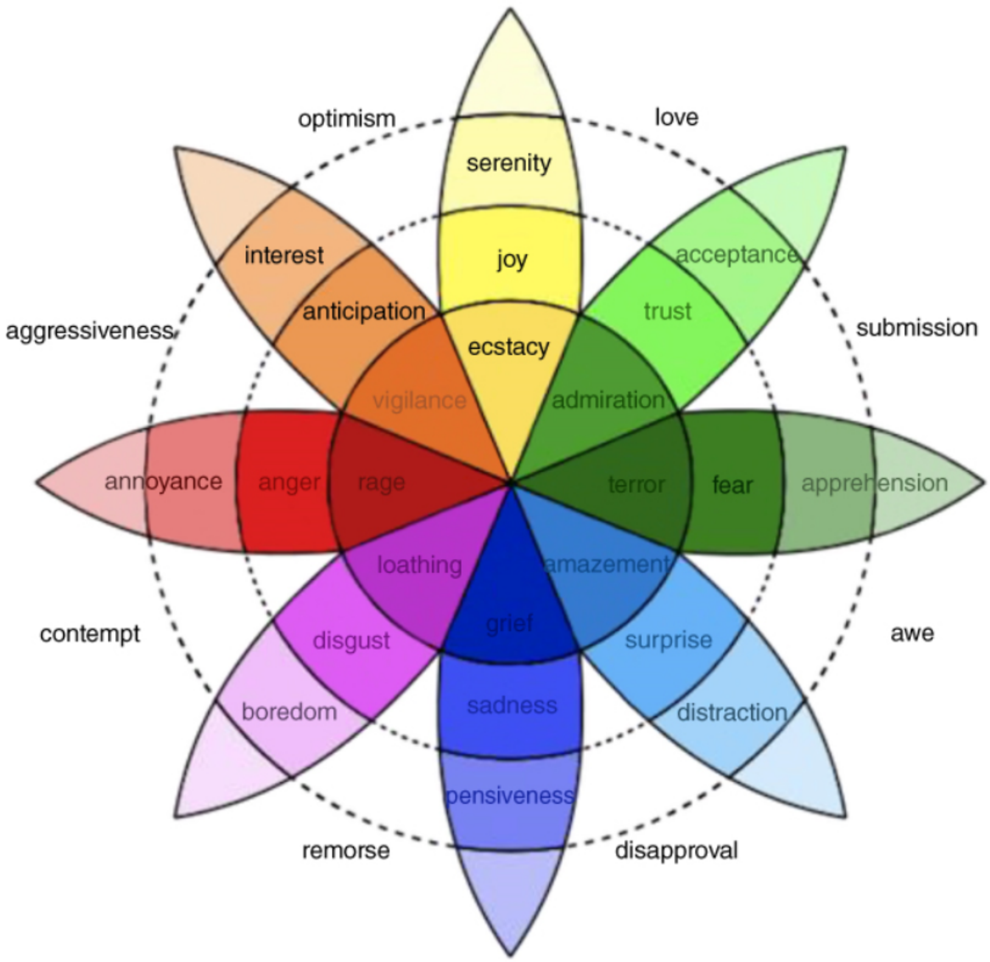



Figure 2: Plutchik's Wheel of Emotions

| Type | Length | Word | Part of Speech | Stemmed y/n | Polarity |
|---|---|---|---|---|---|
| Weak | 1 | Abundant | Adj | N | Positive |
| Weak | 1 | Abundance | Noun | N | Positive |
| Strong | 1 | Accede | Verb | Y | Positive |
| Weak | 1 | Accept | Verb | Y | Positive |
| ... | ... | ... | ... | ... | ... |
| Strong | 1 | Mar | Verb | Y | Negative |
| Weak | 1 | Marginal | Adj | N | Negative |
| Weak | 1 | Marginally | Adv | N | Negative |
| Strong | 1 | Martyrdom | Noun | N | Negative |
| ... | ... | ... | ... | ... | ... |

Figure 3: Excerpt from U Pittsburgh's MQPA Subjectivity Lexicon

| SENTENCE | POLARITY |
|---|---|
| "Sentiment analysis in R is *good* yet *challenging*." | 0 |
| "Sentiment analysis in Python is *not good*." | -1 |
| "Sentiment analysis in R is *very good*." | +2 |

- Challenges:
  1. word groups can shift the score - like "not" or "very"
  2. word choice is not universal - cp. "wicked" in UK vs. US
  3. social media terms are not in lexicons - e.g. "lol"
- Functions ought to allow you to change the subjectivity lexicon (like adding to the `stopwords` dictionary when cleaning data).
- Here are some others because it may be necessary to swap lexica:

## Context clusters

- The function `polarity` creates a *context cluster* around each word found in the lexicon: "The DataCamp sentiment course is very GOOD for learning." Only the word "GOOD" is found in the lexicon.

| Name | Description |
| --- | --- |
| `dodds_sentiment` | Mechanical Turk Sentiment Words |
| `hash_emoticons` | Translations of basic punctuation emoticons :) |
| `hash_sentiment_huliu` | U of IL @CHI Polarity (+/-) word research |
| `hash_sentiment_jockers` | A lexicon inherited from `library(syuzhet)` |
| `hash_sentiment_nrc` | 5468 words crowdsourced scoring between -1 & 1 |

Figure 4: Different subjectivity lexicons in use

- The cluster contains the four words before and the two words after the word found - this means that removing stopwords affects the score.
- The evaluation of the sentence yields:

| Term | Class | Word Count |
| --- | --- | --- |
| Very | Amplifier | 1 |
| Good | Polarized Term/Positive | 1 |
| All other words | Neutral | 7 |

Figure 5: context cluster example

- The cluster words are classified according to different categories:

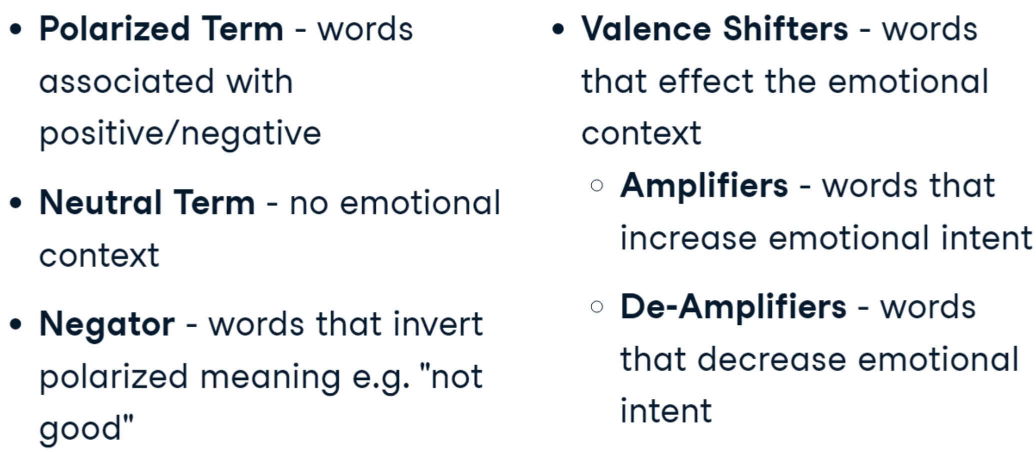



Figure 6: context cluster example

## Subjectivity lexicons in `qdap` and `tidytext`

- `qdap::polarity` uses a lexicon from `hash_sentiment_huliu`, a `data.table` dataset (Hu/Liu, 2004).
- The `tidytext` package has a `sentiments tibble` with 3 lexicons:
    1. NRC - words according to 8 emotions and positive/negative
    2. Bing - words labelled positive or negative
    3. AFINN - words scored from -5 to +5

```
library(tidytext)
ls('package:tidytext')
```

```
Warning message:
package 'tidytext' was built under R version 4.2.3
 [1] "augment"                    "bind_tf_idf"
 [3] "cast_dfm"                   "cast_dtm"
 [5] "cast_sparse"                "cast_tdm"
 [7] "get_sentiments"             "get_stopwords"
 [9] "glance"                     "nma_words"
[11] "parts_of_speech"            "reorder_func"
[13] "reorder_within"             "scale_x_reordered"
```

```
[15] "scale_y_reordered"         "sentiments"
[17] "stop_words"                "tidy"
[19] "unnest_character_shingles" "unnest_characters"
[21] "unnest_lines"              "unnest_ngrams"
[23] "unnest_paragraphs"         "unnest_ptb"
[25] "unnest_regex"              "unnest_sentences"
[27] "unnest_skip_ngrams"        "unnest_tokens"
[29] "unnest_tweets"
```

- To look at the lexicons you must:
  1. install the `textdata` package
  2. run `get_sentiments` with the lexicon as argument
  3. as an example: NRC (link)

```
library(tidytext)
library(textdata)
## you must first download the lexicon with get_sentiment("nrc")
nrc <- get_sentiments("nrc")
str(nrc)
head(nrc)
tail(nrc)
```

```


Attaching package: 'textdata'

The following object is masked from 'package:httr':

    cache_info

Warning message:
package 'textdata' was built under R version 4.2.3
tibble [13,872 × 2] (S3: tbl_df/tbl/data.frame)
 $ word     : chr [1:13872] "abacus" "abandon" "abandon" "abandon" ...
 $ sentiment: chr [1:13872] "trust" "fear" "negative" "sadness" ...
# A tibble: 6 × 2
  word      sentiment
  <chr>     <chr>
1 abacus    trust
```

```
2 abandon   fear
3 abandon   negative
4 abandon   sadness
5 abandoned anger
6 abandoned fear
# A tibble: 6 × 2
  word    sentiment
  <chr>   <chr>
1 zealous trust
2 zest    anticipation
3 zest    joy
4 zest    positive
5 zest    trust
6 zip     negative
```

## Scoring in `qdap::polarity`

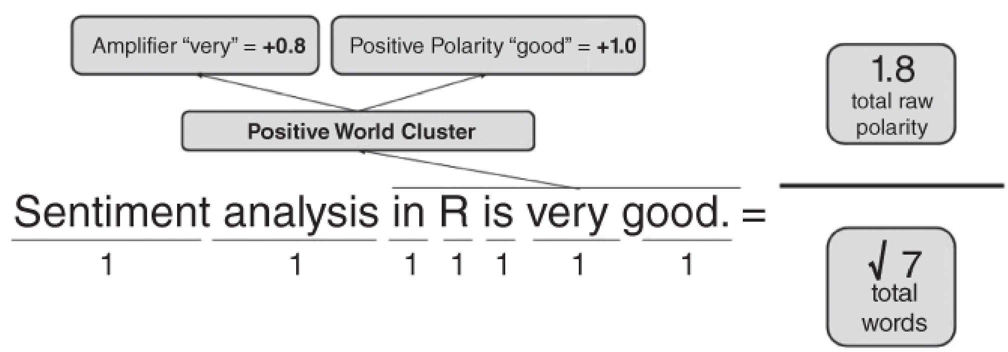



Figure 7: qdap's polarity function equals 0.68 on this sentence

1. The function **`polarity`** scans for positive and negative words
2. When a polarity word is found, a cluster of terms is created, including the four preceding and the two following words.
3. Within the cluster, neutral/positive/negative words are counted as 0, 1 and -1 respectively.
4. The remaining non-neutral and non-polarity words are *valence shifters*: they give a weight to amplify/detract from polar words.

5. The weight is 0.8 - amplifiers add 0.8, detractors subtract 0.8.

6. All values in the 7-word cluster are summed to created total polarity with amplification or negation effects.

7. The total polarity is divided by the square root of all words in the cluster to measure the density of the keywords (more closely packed keywords have a greater impact on the overall polarity).

```
1.8/sqrt(7)
```

```
[1] 0.6803361
```

8. Some of the arguments of the `polarity` function should make sense now:

```
library(qdap)
args(polarity)
```

```
function (text.var, grouping.var = NULL, polarity.frame = qdapDictionaries::key.po
    constrain = FALSE, negators = qdapDictionaries::negation.words,
    amplifiers = qdapDictionaries::amplification.words, deamplifiers = qdapDiction
    question.weight = 0, amplifier.weight = 0.8, n.before = 4,
    n.after = 2, rm.incomplete = FALSE, digits = 3, ...)
NULL
```

## Visualize polarity

- We'll store statements from a conversation among four people as a data frame. We'll apply `polarity` to all sentences for an "average sentiment", and then we'll `plot` the whole conversation.

- Create data frame with a `person` and a `text` column:

```
text_df <- data.frame(
  person=c("Nick", "Jonathan", "Martijn","Nicole",
           "Nick", "Jonathan", "Martijn", "Nicole"),
  text=c("DataCamp courses are the best",
         "I like talking to students",
         "Other online data science curricula are boring.",
```

```
         "What is for lunch?",
         "DataCamp has lots of great content!",
         "Students are passionate and are excited to learn",
         "Other data science curriculum is hard to learn and difficult to understa
         "I think the food here is good."))
```

- Remove punctuation (otherwise `polarity` will complain about it):

```
library(tm)
text_df$text <- removePunctuation(text_df$text)
```

- Examine the text data:

```
text_df
str(text_df)
```

```
    person
1     Nick
2 Jonathan
3  Martijn
4   Nicole
5     Nick
6 Jonathan
7  Martijn
8   Nicole
text
1                                          DataCamp courses are the best
2                                            I like talking to students
3                             Other online data science curricula are boring
4                                                     What is for lunch
5                                      DataCamp has lots of great content
6                            Students are passionate and are excited to learn
7 Other data science curriculum is hard to learn and difficult to understand
8                                          I think the food here is good
'data.frame': 8 obs. of  2 variables:
 $ person: chr  "Nick" "Jonathan" "Martijn" "Nicole" ...
 $ text  : chr  "DataCamp courses are the best" "I like talking to students" "Othe
```

- Approximate the sentiment (polarity) of text by grouping variables:

```
library(qdap)
args(polarity)

function (text.var, grouping.var = NULL, polarity.frame = qdapDictionaries::key.po
    constrain = FALSE, negators = qdapDictionaries::negation.words,
    amplifiers = qdapDictionaries::amplification.words, deamplifiers = qdapDiction
    question.weight = 0, amplifier.weight = 0.8, n.before = 4,
    n.after = 2, rm.incomplete = FALSE, digits = 3, ...)
NULL
```

- The function does not require a vector:

```
polarity("Malcolm X is not a student at all.")

  all total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 all               1           8            0          NA                 NA
```

- Compute polarity on the `text`:

```
polarity(text.var=text_df$text)

  all total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 all               8          54        0.179       0.452              0.396
```

- Group by the `person` column and save the result:

```
polarity(text.var=text_df$text,
         grouping.var=text_df$person) -> datacamp_conversation
datacamp_conversation

    person total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 Jonathan               2          13        0.577       0.184              3.141
2  Martijn               2          19       -0.478       0.141             -3.388
3     Nick               2          11        0.428       0.028             15.524
4   Nicole               2          11        0.189       0.267              0.707
```

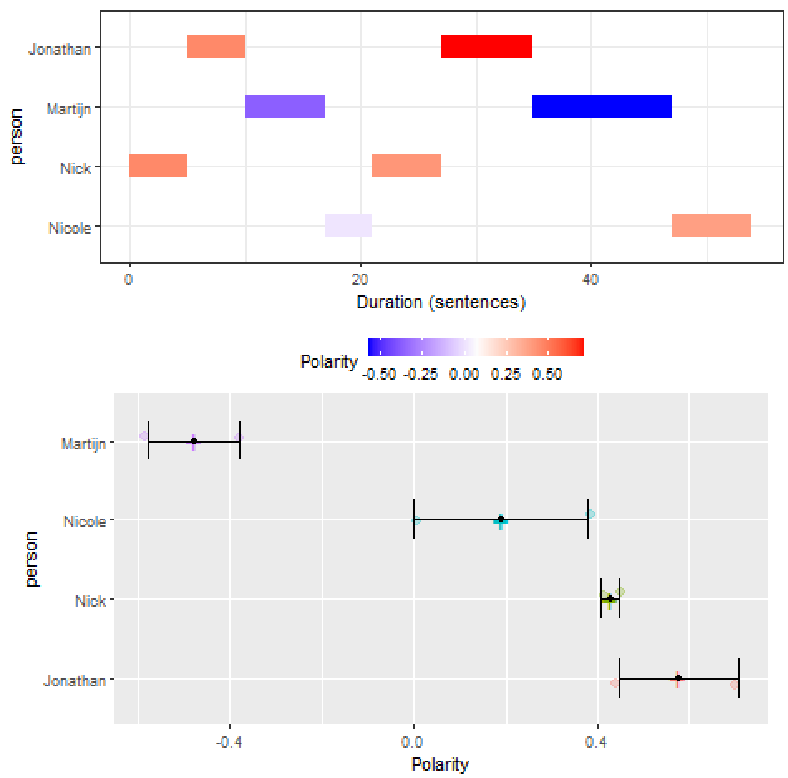
- Apply `qdap::counts` to print the specific emotional words that were found:

```
library(qdap)
counts(datacamp_conversation)

    person wc polarity           pos.words       neg.words
1     Nick  5    0.447                best               -
2 Jonathan  5    0.447                like               -
3  Martijn  7   -0.378                   -          boring
4   Nicole  4    0.000                   -               -
5     Nick  6    0.408               great               -
6 Jonathan  8    0.707 passionate, excited               -
7  Martijn 12   -0.577                   - hard, difficult Other data science curr
8   Nicole  7    0.378                good               -
```

- Plot the `datacamp_conversation` with `plot` (which has a suitable method for polarity output already):

```
plot(datacamp_conversation)
```



- Let's interpret the plot:

  1. Lower plot: Martijn is negative, all the others are positive, Jonathan most of all, Nicole less.
  2. Upper plot: shows the timeline - looks as if Martijn was getting more negative, and Jonathan more positive as time went on.
  3. If you compare with the original statements you can see that the function did not capture the "Other" in Martijn's sentences: he actually was complimenting DataCamp!

- `plot` is a generic function and contains methods for different polarity scoring data structures:

```
library(qdap)
methods(plot)
```

```
  [1] plot,ANY-method
  [2] plot,color-method
  [3] plot.acf*
  [4] plot.ACF*
  [5] plot.animated_character*
  [6] plot.animated_discourse_map*
  [7] plot.animated_formality*
  [8] plot.animated_lexical_classification*
  [9] plot.animated_polarity*
 [10] plot.augPred*
 [11] plot.automated_readability_index*
 [12] plot.character_table*
 [13] plot.cm_distance*
 [14] plot.cmspans*
 [15] plot.cohesiveBlocks*
 [16] plot.coleman_liau*
 [17] plot.combo_syllable_sum*
 [18] plot.communities*
 [19] plot.compareFits*
 [20] plot.cumulative_animated_formality*
 [21] plot.cumulative_animated_lexical_classification*
 [22] plot.cumulative_animated_polarity*
 [23] plot.cumulative_combo_syllable_sum*
 [24] plot.cumulative_end_mark*
```

```
[25] plot.cumulative_formality*
[26] plot.cumulative_lexical_classification*
[27] plot.cumulative_polarity*
[28] plot.cumulative_syllable_freq*
[29] plot.data.frame*
[30] plot.decomposed.ts*
[31] plot.default
[32] plot.dendrogram*
[33] plot.density*
[34] plot.discourse_map*
[35] plot.diversity*
[36] plot.DocumentTermMatrix*
[37] plot.ecdf
[38] plot.end_mark*
[39] plot.end_mark_by*
[40] plot.end_mark_by_count*
[41] plot.end_mark_by_preprocessed*
[42] plot.end_mark_by_proportion*
[43] plot.end_mark_by_score*
[44] plot.factor*
[45] plot.flesch_kincaid*
[46] plot.formality*
[47] plot.formality_scores*
[48] plot.formula*
[49] plot.freq_terms*
[50] plot.function
[51] plot.gantt*
[52] plot.ggplot*
[53] plot.gls*
[54] plot.gtable*
[55] plot.hcl_palettes*
[56] plot.hclust*
[57] plot.histogram*
[58] plot.HoltWinters*
[59] plot.igraph*
[60] plot.intervals.lmList*
[61] plot.irt
[62] plot.isoreg*
[63] plot.kullback_leibler*
[64] plot.lexical*
```

```
 [65] plot.lexical_classification*
 [66] plot.lexical_classification_preprocessed*
 [67] plot.lexical_classification_score*
 [68] plot.linsear_write*
 [69] plot.linsear_write_count*
 [70] plot.linsear_write_scores*
 [71] plot.lm*
 [72] plot.lme*
 [73] plot.lmList*
 [74] plot.medpolish*
 [75] plot.mlm*
 [76] plot.Network*
 [77] plot.nffGroupedData*
 [78] plot.nfnGroupedData*
 [79] plot.nls*
 [80] plot.nmGroupedData*
 [81] plot.object_pronoun_type*
 [82] plot.pdMat*
 [83] plot.polarity*
 [84] plot.polarity_count*
 [85] plot.polarity_score*
 [86] plot.poly
 [87] plot.poly.parallel
 [88] plot.pos*
 [89] plot.pos_by*
 [90] plot.pos_preprocessed*
 [91] plot.ppr*
 [92] plot.prcomp*
 [93] plot.princomp*
 [94] plot.profile.nls*
 [95] plot.pronoun_type*
 [96] plot.psych
 [97] plot.question_type*
 [98] plot.question_type_preprocessed*
 [99] plot.R6*
[100] plot.ranef.lme*
[101] plot.ranef.lmList*
[102] plot.raster*
[103] plot.readability_count*
[104] plot.readability_score*
```

```
[105] plot.reliability
[106] plot.residuals
[107] plot.rmgantt*
[108] plot.sent_split*
[109] plot.shingle*
[110] plot.simulate.lme*
[111] plot.sir*
[112] plot.SMOG*
[113] plot.spec*
[114] plot.stepfun
[115] plot.stl*
[116] plot.subject_pronoun_type*
[117] plot.sum_cmspans*
[118] plot.sums_gantt*
[119] plot.syllable_freq*
[120] plot.table*
[121] plot.table_count*
[122] plot.table_proportion*
[123] plot.table_score*
[124] plot.termco*
[125] plot.TermDocumentMatrix*
[126] plot.times*
[127] plot.trans*
[128] plot.trellis*
[129] plot.ts
[130] plot.tskernel*
[131] plot.TukeyHSD*
[132] plot.type_token_ratio*
[133] plot.Variogram*
[134] plot.VennDiagram*
[135] plot.weighted_wfm*
[136] plot.wfdf*
[137] plot.wfm*
[138] plot.word_cor*
[139] plot.word_length*
[140] plot.word_position*
[141] plot.word_proximity*
[142] plot.word_stats*
[143] plot.word_stats_counts*
see '?methods' for accessing help and source code
```

## Components of `polarity`

- For later, it might be useful to be able to extract parts of the `polarity` result: look at the structure of `datacamp_conversation`:

```
str(datacamp_conversation)  # display structure of a polarity object
```

```
List of 2
 $ all  :'data.frame': 8 obs. of  6 variables:
  ..$ person   : chr [1:8] "Nick" "Jonathan" "Martijn" "Nicole" ...
  ..$ wc       : int [1:8] 5 5 7 4 6 8 12 7
  ..$ polarity : num [1:8] 0.447 0.447 -0.378 0 0.408 ...
  ..$ pos.words:List of 8
  .. ..$ : chr "best"
  .. ..$ : chr "like"
  .. ..$ : chr "-"
  .. ..$ : chr "-"
  .. ..$ : chr "great"
  .. ..$ : chr [1:2] "passionate" "excited"
  .. ..$ : chr "-"
  .. ..$ : chr "good"
  ..$ neg.words:List of 8
  .. ..$ : chr "-"
  .. ..$ : chr "-"
  .. ..$ : chr "boring"
  .. ..$ : chr "-"
  .. ..$ : chr "-"
  .. ..$ : chr "-"
  .. ..$ : chr [1:2] "hard" "difficult"
  .. ..$ : chr "-"
  ..$ text.var : chr [1:8] "DataCamp courses are the best" "I like talking to stuc
 $ group:'data.frame': 4 obs. of  6 variables:
  ..$ person            : chr [1:4] "Jonathan" "Martijn" "Nick" "Nicole"
  ..$ total.sentences   : int [1:4] 2 2 2 2
  ..$ total.words       : int [1:4] 13 19 11 11
  ..$ ave.polarity      : num [1:4] 0.577 -0.478 0.428 0.189
  ..$ sd.polarity       : num [1:4] 0.1838 0.141 0.0276 0.2673
  ..$ stan.mean.polarity: num [1:4] 3.141 -3.388 15.524 0.707
 - attr(*, "class")= chr [1:2] "polarity" "list"
 - attr(*, "digits")= num 3
```

```
 - attr(*, "constrained")= logi FALSE
 - attr(*, "unconstrained.polarity")= num [1:8] 0.447 0.447 -0.378 0 0.408 ...
```

- Extract the positive and negative words, the text, and the people:

```
unlist(datacamp_conversation$all[["pos.words"]])
unlist(datacamp_conversation$all[["neg.words"]]) -> negative
unlist(datacamp_conversation$all[["text.var"]])
unlist(datacamp_conversation$all[["person"]])
```

```
[1] "best"       "like"       "-"          "-"          "great"      "passionate"
[7] "excited"    "-"          "good"
[1] "DataCamp courses are the best"
[2] "I like talking to students"
[3] "Other online data science curricula are boring"
[4] "What is for lunch"
[5] "DataCamp has lots of great content"
[6] "Students are passionate and are excited to learn"
[7] "Other data science curriculum is hard to learn and difficult to understand"
[8] "I think the food here is good"
[1] "Nick"     "Jonathan" "Martijn"  "Nicole"   "Nick"     "Jonathan" "Martijn"
[8] "Nicole"
```

- Did any of the people find DataCamp "boring"?

```
any(negative=="boring")
```

```
[1] TRUE
```

## Adding terms to the subjectivity lexicon

- The polarity scoring is highly sensitive to the lexicon used. If we already know that our subjects (people who talk or write) are using special words not in the lexicon, then we need to add them.
- Here, we're doing that to show the impact of a few words missing on the polarity scoring with `polarity`.
- To add new terms, define a vector `new.pos`:

```
new.pos <- c('rofl', 'lol')
```

- Load qdap. Its basic subjectivity lexicon is held in a list key.pol - it contains 6779 terms x and their polarity labels y.

```
library(qdap)
str(key.pol)
```

```
Classes 'qdap_hash', 'data.table' and 'data.frame': 6779 obs. of  2 variables:
 $ x: chr  "a plus" "abnormal" "abolish" "abominable" ...
 $ y: num  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
 - attr(*, ".internal.selfref")=<externalptr>
 - attr(*, "sorted")= chr "x"
 - attr(*, "mode")=function (x, ...)
```

- How can you get the list of all dictionaries in qdap? You can see all qdap packages in the search() vector:

```
search()[grep('qdap*',search())]
```

```
[1] "package:qdap"               "package:qdapTools"          "package:qdapRegex"
[4] "package:qdapDictionaries"
```

- Now use data to list the dictionaries in qdapDictionaries:

```
library(qdap)
data(package='qdapDictionaries')
```

```
Warning message:
In file.show(outFile, delete.file = TRUE, title = paste("R", tolower(x$title))) :
  '"c:/PROGRA~1/R/R-42~1.2/bin/pager"' not found
```

- Use the subset(x,pattern) function to retain only the original key.pol terms that have polarity 1 and store them in old.pos:

```
subset(as.data.frame(key.pol), key.pol$y==1) -> old.pos
str(old.pos)
```

```
'data.frame': 2003 obs. of  2 variables:
 $ x: chr  "a plus" "abound" "abounds" "abundance" ...
 $ y: num  1 1 1 1 1 1 1 1 1 1 ...
```

- Add `new.pos` to `old.pos` and create `all.pos`:

```
all.pos <- c(new.pos, old.pos[,1]) # only the terms, not the scores
```

- Proceed accordingly with the negative portion of the subjectivity lexicon. For example to include the terms 'kappa' (used among gamers to express sarcasm) and 'meh' (unenthusiastic):

```
new.neg <- c('kappa','meh')
old.neg <- subset(as.data.frame(key.pol),key.pol$y==-1)
all.neg <- c(new.neg,old.neg[,1])
```

- To compute polarity score, `polarity` uses a sentiment lookup table as a function of vectors of `positives` and `negatives` and their weights:

```
args(sentiment_frame)

function (positives, negatives, pos.weights = 1, neg.weights = -1)
NULL
```

- We need to create a new sentiment data frame `all.polarity` replacing `key.pol` using `sentiment_frame`:

```
all.polarity <- sentiment_frame(all.pos,all.neg,1,-1)  ## this is our new lexicon
str(all.polarity)

Classes 'qdap_hash', 'key', 'data.table' and 'data.frame': 6783 obs. of  2 variabl
 $ x: chr  "a plus" "abnormal" "abolish" "abominable" ...
 $ y: num  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
 - attr(*, ".internal.selfref")=<externalptr>
 - attr(*, "sorted")= chr "x"
 - attr(*, "mode")=function (x, ...)
```

- You can see that there are four more words included:

```
c('rofl','lol','meh','kappa') %in% all.polarity$x
c('rofl','lol','meh','kappa') %in% key.pol$x

[1] TRUE TRUE TRUE TRUE
[1] FALSE FALSE FALSE FALSE
```

## Using the extended subjectivity lexicon

- Consider the sample sentences:

```
foo <- 'ROFL, look at that!'
bar <- 'Whatever you say - Kappa.'
```

- Applying `polarity` returns the polarity scores:

```
polarity(foo, polarity.frame=all.polarity)
```

```
  all total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 all               1           4          0.5          NA                 NA
```

- There is no grouping variable (`all`), one sentence only with 4 words (hence no stats), and the polarity is 0.5 = 1/sqrt(4) because 'ROFL' counts as 1.

- When computing the polarity with the standard lexicon, polarity is zero or neutral, because 'ROFL' was not found in the lexicon:

```
polarity(foo)  # by default, polarity.frame is key.pol
```

```
  all total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 all               1           4            0          NA                 NA
```

- Applying `polarity` and `all.polarity` to the second sentence:

```
polarity(bar,polarity.frame=all.polarity)
polarity(bar)
```

```
  all total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 all               1           4         -0.5          NA                 NA
  all total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 all               1           4            0          NA                 NA
```

- 'Kappa' counts as -1 and -1/sqrt(4) = -0.5, and with the standard lexicon, Kappa is not found and considered neutral.


Figure 8: Sentiments by Viktoriia Vidal (Flickr.com)

## Why do subjectivity lexicons work at all?

- How can such a short list of only several thousand words deliver somewhat accurate sentiment readings?
- An average person has tens of thousands of words in their personal vocabulary, and any list would miss many of these words.
- The number of unique words varies by gender, age, and demography, making a "hit" on one of only 6,800 words very unlikely.

## Zipf's law and the principle of least effort

| RANK | Word | Frequency | Expected |
|---|---|---|---|
| NA | sb | 1,984,423 | |
| 1 | rt | 1,700,564 | 1700564/1=1,700,564 |
| 2 | the | 1,101,899 | 1700564/2=850,282 |
| 3 | to | 588,803 | 1700564/3=566,855 |
| 4 | A | 428,598 | 1700564/4=425,141 |
| 5 | for | 388,390 | 1700564/5=340,113 |

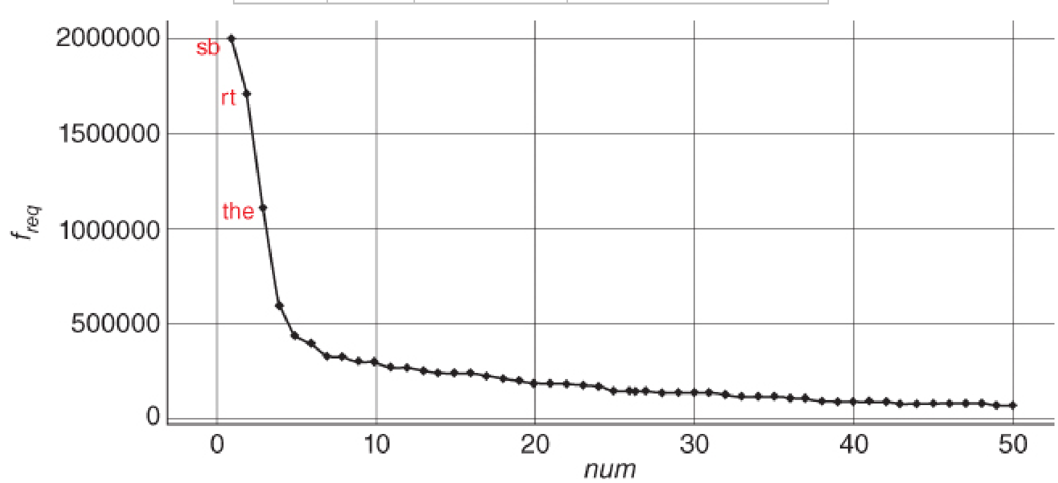



Figure 9: Top 5 terms in word frequency matrix and top 50 from 2.5mio tweets

- **Zipf's law** asserts that any word in a document is **inversely proportional to its rank** when looking at term frequency.

- The most frequent word will occur about twice as often as the 2nd most frequent word, three times as often as the third, and so on.

- Zipf's law outside of linguistics - city populations:

| Rank | City | 2010 Census Population | Actual % | Zipf's Expected % |
|---|---|---|---|---|
| 1 | New York | 8,175,133 | 100% | ... |
| 2 | LA | 3,792,621 | 46% | 50% |
| 3 | Chicago | 2,695,598 | 33% | 33% |
| 4 | Houston | 2,100,263 | 26% | 25% |
| 5 | Philadelphia | 1,526,006 | 19% | 20% |

Figure 10: Zipf's power law for city populations based on rank

- One explanation of Zipf's law is the **principle of least effort**: humans will choose the path of least resistance and minimize effort.

- Once some minimum threshold of understanding has occurred, the effort exerted in searching for meaning will decrease or cease.

- While humans may know many words, they often use only a few thousand distinct terms when communicating to minimize effort.[2]

## Observing Zipf's law for a big data set

- To prove it, let's construct a visual from 3 million tweets mentioning "#sb" (SuperBowl).

- Keep in mind that the visual doesn't follow Zipf's law perfectly, the tweets all mentioned the same hashtag so it is a bit skewed.

[2] I don't find this argument compelling. It seems to me that there are many more factors present when humans communicate - efficiency of expression being only one of them. Rather, this could be an artefact of digital communication - and hence potentially another example where we mistake relationships between humans mediated by machines for true relationships.

- The visual follows a steep decline showing a small lexical diversity among the millions of tweets.

- In this exercise, we use the package `metricsgraphics`. The main function of the package `metricsgraphics` is the `mjs_plot()` function which is the first step in creating a JavaScript plot

- An example `metricsgraphics` workflow is below:

```
## make basic JavaScript plot for mouse-over action
metro_plot <- mjs_plot(data, x = x_axis_name,
                             y = y_axis_name,
                             show_rollover_text = FALSE)
## add lines
metro_plot <- mjs_line(metro_plot)
metro_plot <- mjs_add_line(metro_plot,
                           line_one_values)
## add a legend
metro_plot <- mjs_add_legend(metro_plot,
                             legend = c('names', 'more_names'))
metro_plot
```

- Getting the data and reviewing the top words:

```
sb_words=read.csv("https://docs.google.com/spreadsheets/d/e/2PACX-1vSr1GbdxxFhoZcA
## Examine sb_words
head(sb_words)
str(sb_words) # three columns: word, freq and rank

  word    freq rank
1   sb 1984423    1
2   rt 1700564    2
3  the 1101899    3
4   to  588803    4
5    a  428598    5
6  for  388390    6
'data.frame': 159 obs. of  3 variables:
 $ word: chr  "sb" "rt" "the" "to" ...
 $ freq: int  1984423 1700564 1101899 588803 428598 388390 326464 322154 296673 29
 $ rank: int  1 2 3 4 5 6 7 8 9 10 ...
```

- Create a new column expectations by dividing the largest word frequency, `freq[1]`, by the `rank` column:

```
sb_words$expectations <- sb_words$freq[1]/sb_words$rank
str(sb_words)
```

```
'data.frame': 159 obs. of  4 variables:
 $ word        : chr  "sb" "rt" "the" "to" ...
 $ freq        : int  1984423 1700564 1101899 588803 428598 388390 326464 322154 2
 $ rank        : int  1 2 3 4 5 6 7 8 9 10 ...
 $ expectations: num  1984423 992212 661474 496106 396885 ...
```

- Load `metricsgraphics` (must be installed). Start `sb_plot` using `mjs_plot`, and pass in `data=sb_words` with `x = rank`, `y = freq` and set `show_rollover_text` to `FALSE`:

```
library(metricsgraphics)
sb_plot <- mjs_plot(data=sb_words,
                    x=rank,
                    y=freq,
                    show_rollover_text=FALSE)
```

- Add first line to `sb_plot`:

```
sb_plot <- mjs_line(sb_plot)
```

- Add 2nd line to `sb_plot` with `mjs_add_line()`. Pass in the previous `sb_plot` object and the vector, `expectations`:

```
sb_plot <- mjs_add_line(sb_plot, expectations)
```

- Place a legend on a new `sb_plot` object using `mjs_add_legend`:

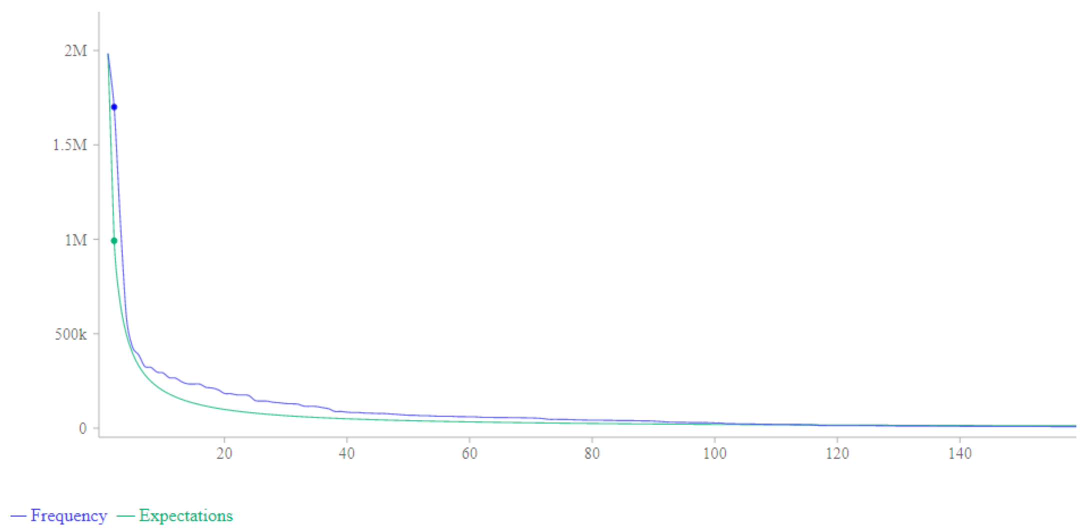
  1. pass in the previous `sb_plot` object
  2. The legend `labels` should consist of "Frequency" and "Expectation":

```
sb_plot <- mjs_add_legend(sb_plot,
                          legend=c("Frequency","Expectation"))
```

- Call `sb_plot` to display the plot. Mouseover a point to simultaneously highlight a `freq` and Expectation point:

```
sb_plot  # This will open an interactive plot in the browser
```

- Screenshot:




## Exercise: manual `polarity` scoring

- `polarity` scans the text to identify words in the lexicon. It then creates a *cluster* around an identified *subjectivity word*. Within the cluster *valence shifters* adjust the score.

- Valence shifters are words that amplify or negate the emotional intent of the subjectivity word. For example, "well known" is positive while "not well known" is negative. Here "not" is a negating term and reverses the emotional intent of "well known." In contrast, "very well known" employs an *amplifier* increasing the positive intent.

- The `polarity` function then calculates a score using subjectivity terms, valence shifters and the total number of words in the passage. This exercise demonstrates a simple polarity calculation.

- Calculate the polarity of the string positive in a new object called pos_score, then print it:

```
positive <- "DataCamp courses are good for learning"
## Calculate polarity of statement
library(qdap)
polarity(positive)

  all total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
1 all               1           6        0.408          NA                 NA
```

- Manually perform the same polarity calculation:

  1. Get a word count object pos_counts by calling counts on the polarity object pos_score, then print it:
  2. All the identified subjectivity words are part of count object's list. Specifically, positive words are in the $pos.words element vector of pos_counts. Print the structure of pos_counts:
  3. Find the number of positive words by calling length on the first member of the $pos_words element of pos_counts and store it in n_good:
  4. Capture the total number of words and assign it to n_words. This value is stored in pos_count as the wc (word count) element:
  5. De-construct the polarity calculation by dividing n_good by sqrt of n_words and save the result as pos_pol. Compare pos_pol to pos_score calculated with polarity earlier using identical:

## Exercise: apply valence shifters

- Of course just positive and negative words aren't enough. In this exercise you will learn about valence shifters which tell you about the author's emotional intent. Previously you applied polarity() to text without valence shifters. In this example you will see amplification and negation words in action.

- Recall that an amplifying word adds 0.8 to a positive word in polarity() so the positive score becomes 1.8. For negative words 0.8 is subtracted so the total becomes -1.8. Then the score is divided by the square root of the total number of words.

- Consider the following example from Frank Sinatra:

  ```
  "It was a very good year"
  ```

  "Good" equals 1 and "very" adds another 0.8. So, 1.8/sqrt(6) results in 0.73 polarity.

- A negating word such as "not" will inverse the subjectivity score. Consider the following example from Bobby McFerrin:

  ```
  "Don't worry Be Happy"
  ```

  ```
  [1] "Don't worry Be Happy"
  ```

  "worry is now 1 due to the negation "don't." Adding the "happy", +1, equals 2. With 4 total words, 2 / sqrt(4) equals a polarity score of 1.

---

1. Load the `conversation` data frame and display its structure:

   ```
   conversation <- data.frame( "student"=c("Martijn","Nick","Nicole"),
                              "text"=c("This restaurant is never bad", "The lunch was
                                       "It was awful I got food poisoning and was ext
   str(conversation)
   ```

   ```
   'data.frame': 3 obs. of  2 variables:
    $ student: chr  "Martijn" "Nick" "Nicole"
    $ text   : chr  "This restaurant is never bad" "The lunch was very good" "It was
   ```

2. Examine the `conversation` data frame by printing it.

3. What context cluster category is "never"?

   Answer: ...

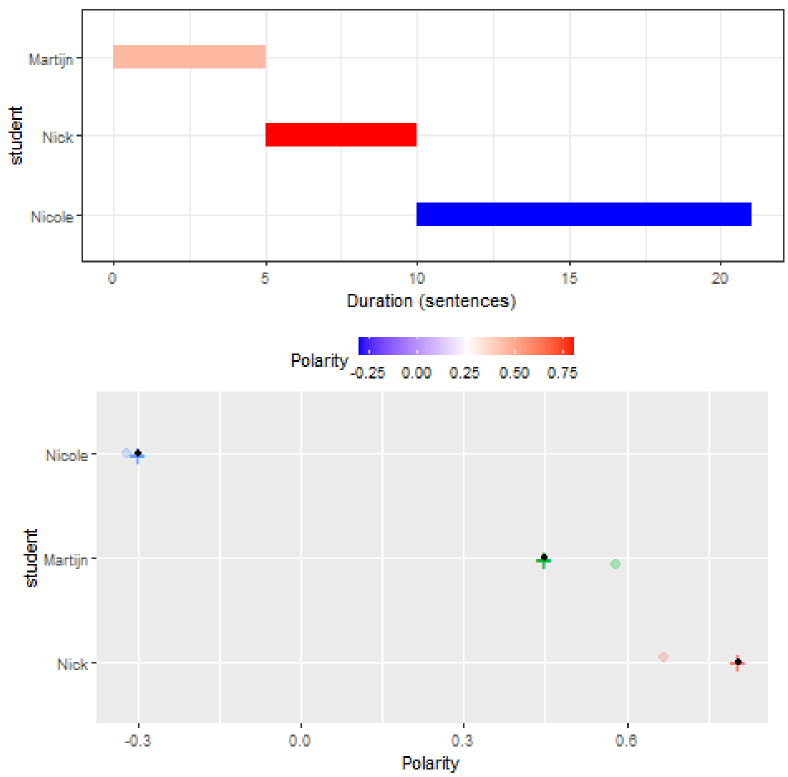
4. Apply `polarity()` to the text column of conversation to calculate the polarity for the entire conversation:

5. Calculate the polarity scores for the `text` by `student` using the `grouping.var` argument, and assign the result to `student_pol`:

6. To see the `student` level results, use `scores()` on `student_pol`:

7. The `counts()` function applied to `student_pol` will print the sentence level polarity for the entire data frame along with lexicon words identified:

8. The polarity object, `student_pol`, can be plotted with `plot()`:




## Exercise: examine and use `qdap`'s lexicon

- Even with Zipf's law in action, you will still need to adjust lexicons to fit the text source (for example twitter versus legal documents) or the author's demographics (teenager versus the elderly). This exercise demonstrates the explicit components of `polarity()` so you can change it if needed.

- In Trey Songz "Lol :)" song there is a lyric "LOL smiley face, LOL smiley face." In the basic `polarity()` function, "LOL" is not defined as positive. However, "LOL" stands for "Laugh Out Loud" and should be positive. As a result, you should adjust the lexicon to fit the text's context which includes pop-culture slang. If your analysis contains text from a specific channel (Twitter's "LOL"), location (Boston's "Wicked Good"), or age group (teenagers' "sick") you will likely have to adjust the lexicon.

- In the first exercise, you are examining the existing word data frame objects so you can change them in the following exercise.

---

1. As a sample text, here are two excerpts from Beyoncé's "Crazy in Love" lyrics for the exercise - run the code and print the data frame's structure:

```
text <- data.frame(
  "speaker"=c("beyonce","jay_z"),
  "words"=c("I know I dont understand Just how your love can do what no one else c
            "They cant figure him out they like hey, is he insane"))
```

2. Print `qdapDictionaries::key.pol` to see a portion of the subjectivity words and values:

3. Examine the predefined `negation.words` to print all the negating terms:

4. Print the amplifiers in `amplification.words` to see the words that add values to the lexicon:

5. Check the `deamplification.words` that reduce the lexicon values:

6. Now, calculate the `polarity` of `text` as follows and save it in `text_pol`:

   (a) Set `text.var` to `text$words`.
   (b) Set `grouping.var` to `text$speaker`.
   (c) Set `polarity.frame` to `key.pol`.
   (d) Set `negators` to `negation.words`.
   (e) Set `amplifiers` to `amplification.words`.
   (f) Set `deamplifiers to deamplification.words`.

7. Print the positive and negative words alongside the `text` with the `all` element of `text_pol`:

8. Why is the polarity of Beyonce's lyrics 0.25, and why is the polarity of Jay Z's lyrics 0?

   Answer:

## Exercise: amplification and negation words

- Here you will adjust the negative words to account for the specific text. You will then compare the basic and custom `polarity()` scores.

- A popular song from Twenty One Pilots is called "Stressed Out" (2015). If you scan the song lyrics, you will observe the song is about youthful nostalgia. Overall, most people would say the polarity is negative. Repeatedly the lyrics mention stress, fears and pretending.

- Let's compare the song lyrics using the default subjectivity lexicon and also a custom one.

- To start, you need to verify the `key.pol` subjectivity lexicon does not already have the term you want to add. One way to check is with `grep`. The pattern matching `grep()` function returns the row containing characters that match a search `pattern`. Here is an example where the column `col` of `df` is searched for "search_pattern":

```
idx <- grep(pattern="search_pattern", x=df$col)
```

- The vector `idx` can now be used to return all elements of `df` that match the pattern as `df[idx, ]`.

- After verifying the slang or new word is not already in the `key.pol` lexicon you need to add it.

---

1. Add the lyrics as a single string from the file `stressed_out.txt` and store it in the vector `stressed_out`, then replace \\ by \ with `gsub` and print the lyrics:

```
stressed_out <- readLines("https://bit.ly/stressed_out_txt")
gsub("\\\\n","\n",stressed_out) -> stressed_out
stressed_out
```

```
[1] "I wish I found some better sounds no one's ever heard\nI wish I had a better
```

2. Compute the default `polarity` score of `stressed_out`:

3. Bonus question: can you show just the value for the polarity? Tip: `polarity(stressed_out)` is a `list` and "polarity" is a member of the `$all` element of that list (you can check that with `str`):

4. Check `key.pol` for any words containing "stress":

    (a) use `grep` to index the data frame by searching in the `x` column

    (b) save the result in `rowindex`

5. Construct a new polarity lexicon `custom_pol` using `sentiment_frame`. This function creates a sentiment lookup table for use with the `polarity.frame` argument of `polarity` (i.e. the lexicon) - check the function's arguments:

6. Pass `positive.words` as `positives` argument to the function `sentiment_frame`, and for the second argument concatenate (with `c`) `negative_words` and the words "stressed" and "turn back". Save the result in `custom_pol`

7. Compute the `polarity` using the `custom_pol` lexicon as `polarity.frame`:

8. You should see that the modified lexicon leads to a more realistic sentiment scoring than the standard lexicon.

## Summary

- **Sentiment analysis** is the process of extracting an author's emotional intent from text. It is challenging because of the complexity of human emotions and potential modeling bias.

- A prominent method is **polarity scoring**, which categorizes sentiment as positive, neutral or negative, and is based on subjectivity lexicons, lists of words labelled with emotional states.

- To estimate polarity, lexicon words have to be found and context has to be established using knowledge of negators, amplifiers and deamplifiers, which **shift the valence**, and the lexicon may have to be extended by words with special meaning.

- Subjectivity lexicons work well despite their small size because of **Zipf's power law**, which says that a word's frequency is inversely proportional to its rank: the most frequent word will occur n times as often as the n-th most frequent word.

## Glossary of code

| COMMAND | MEANING |
|---|---|
| `qdap::polarity` | Polarity scoring function |
| `tm::removePunctuation` | Remove punctuation before analysing |
| `qdap::counts` | Print identified emotional words |
| `plot.polarity` | Generic function that plots polarity |
| `plot.polarity_count` | |
| `plot.polarity_score` | |
| `qdapDictionaries` | Different lexicons in the `qdap` package |
| `key.pol` | Standard polarity scoring lexicon in `qdap` |
| `qdap::sentiment_frame` | Create polarity scoring lexicon |

## References


- Kwartler T (2017). Text mining in R. Wiley.

- Kwartler T (2019). Sentiment Analysis in R. URL: datacamp.com.

- Plutchik, Robert: The Nature of Emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice. In: American Scientist, Vol. 89, No. 4 (JULY-AUGUST 2001), pp. 344-350. URL: harvard.edu.