Package ‘qdap’

May 15, 2018

Type Package

Title Bridging the Gap Between Qualitative Data and Quantitative Analysis

Version 2.3.0

Maintainer Tyler Rinker <tyler.rinker@gmail.com>

Depends R (>= 3.1.0), qdapDictionaries (>= 1.0.2), qdapRegex (>= 0.1.2), qdapTools (>= 1.3.1), RColorBrewer

Imports chron, dplyr (>= 0.3), gdata, gender (>= 0.5.1), ggplot2 (>= 2.1.0), grid, gridExtra, igraph, methods, NLP, openNLP (>= 0.2-1), parallel, plotrix, RCurl, reports, reshape2, scales, stringdist, tidyR, tm (>= 0.7.2), tools, venneuler, wordcloud, xlsx, XML

Suggests koRpus, knitr, lda, proxy, stringi, SnowballC, testthat

LazyData TRUE

VignetteBuilder knitr

Description Automates many of the tasks associated with quantitative discourse analysis of transcripts containing discourse including frequency counts of sentence types, words, sentences, turns of talk, syllables and other assorted analysis tasks. The package provides parsing tools for preparing transcript data. Many functions enable the user to aggregate data by any number of grouping variables, providing analysis and seamless integration with other R packages that undertake higher level analysis and visualization of text. This affords the user a more efficient and targeted analysis. ‘qdap’ is designed for transcript analysis, however, many functions are applicable to other areas of Text Mining/ Natural Language Processing.

License GPL-2

URL http://trinker.github.com/qdap/

BugReports http://github.com/trinker/qdap/issues

RoxygenNote 6.0.1

NeedsCompilation no

Author Bryan Goodrich [ctb],
Dason Kurkiewicz [ctb],
Tyler Rinker [aut, cre]
R topics documented:

+ .Network .......................................... 10
add_incomplete .................................... 11
add_s ................................................ 12
adjacency_matrix .................................. 12
all_words ......................................... 13
Animate ............................................. 15
Animate.character ................................ 15
Animate.discourse_map ............................ 17
Animate.formality ............................... 18
Animate.gantt .................................... 19
Animate.gantt_plot ................................ 20
Animate.lexical_classification ................ 20
Animate.polarity ................................ 22
as.tdm .............................................. 23
automated_readability_index .................... 32
bag_o_words .................................... 36
beg2char ........................................ 37
blank2NA ......................................... 38
bracketX ......................................... 39
build_qdap_vignette ............................. 41
capitalizer ......................................... 42
check_spelling .................................. 43
cHECK_spelling_interactive.character .... 46
cHECK_spelling_interactive.check_spelling 47
cHECK_spelling_interactive.factor .......... 48
cHECK_text ...................................... 49
chunker .......................................... 51
clean .............................................. 53
cm_2long .......................................... 53
cm_code.blank .................................. 55
cm_code.combine ................................ 57
cm_code.exclude ................................ 59
cm_code.overlap ................................ 60
cm_code.transform ................................ 62
cm_combine.dummy ............................... 64
cm_df.fill ....................................... 65
cm_df.temp ...................................... 67
cm_df.transcript ................................ 68
cm_df2long ...................................... 69
cm_distance ...................................... 71
cm_dummy2long .................................. 74
cm_long2dummy .................................. 75
R topics documented:

- cm_range.temp ........................................ 77
- cm_range2long ........................................ 78
- cm_time.temp ........................................ 79
- cm_time2long ........................................ 80
- colcomb2class ....................................... 82
- colSplit ............................................. 83
- colsplit2df ........................................... 84
- comma_spacer ......................................... 86
- common ................................................ 86
- common.list .......................................... 87
- condense ............................................. 87
- counts ................................................ 88
- counts.automated_readability_index .............. 89
- counts.character_table ................................ 89
- counts.coleman_liau .................................. 90
- counts.end_mark_by ................................... 90
- counts.flesch_kincaid ................................ 91
- counts.formality ...................................... 91
- counts.fry ............................................ 92
- counts.linsear_write .................................. 92
- counts.object_pronoun_type ....................... 93
- counts.polarity ....................................... 93
- counts.pos ............................................ 94
- counts.pos_by ......................................... 94
- counts.pronoun_type .................................. 95
- counts.question_type .................................. 95
- counts.SMOG .......................................... 96
- counts.subject_pronoun_type ...................... 96
- counts.termco ......................................... 97
- counts.word_length ................................... 97
- counts.word_position .................................. 98
- counts.word_stats .................................... 98
- cumulative ........................................... 99
- DATA .................................................. 100
- DATA_SPLIT ......................................... 101
- DATA2 ................................................. 101
- dir_map ................................................ 102
- dispersion_plot ...................................... 103
- Dissimilarity ......................................... 107
- dist_tab ............................................... 110
- diversity ............................................ 112
- duplicates ............................................ 114
- end_inc ............................................... 115
- end_mark ............................................ 116
- env.syl ............................................... 117
- exclude ............................................... 119
- Filter.all_words ...................................... 120
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>formality</td>
<td>124</td>
</tr>
<tr>
<td>freq_terms</td>
<td>129</td>
</tr>
<tr>
<td>gantt</td>
<td>131</td>
</tr>
<tr>
<td>gantt_plot</td>
<td>133</td>
</tr>
<tr>
<td>gantt_rep</td>
<td>135</td>
</tr>
<tr>
<td>gantt_wrap</td>
<td>137</td>
</tr>
<tr>
<td>gradient_cloud</td>
<td>139</td>
</tr>
<tr>
<td>hamlet</td>
<td>141</td>
</tr>
<tr>
<td>htruncdf</td>
<td>142</td>
</tr>
<tr>
<td>imperative</td>
<td>144</td>
</tr>
<tr>
<td>incomplete_replace</td>
<td>145</td>
</tr>
<tr>
<td>inspect_text</td>
<td>146</td>
</tr>
<tr>
<td>is.global</td>
<td>147</td>
</tr>
<tr>
<td>key_merge</td>
<td>148</td>
</tr>
<tr>
<td>kullback_leibler</td>
<td>149</td>
</tr>
<tr>
<td>left_just</td>
<td>150</td>
</tr>
<tr>
<td>lexical_classification</td>
<td>151</td>
</tr>
<tr>
<td>mcsv_r</td>
<td>158</td>
</tr>
<tr>
<td>mraja1</td>
<td>160</td>
</tr>
<tr>
<td>mraja1spl</td>
<td>161</td>
</tr>
<tr>
<td>multisub</td>
<td>161</td>
</tr>
<tr>
<td>multiscale</td>
<td>163</td>
</tr>
<tr>
<td>NAer</td>
<td>164</td>
</tr>
<tr>
<td>name2sex</td>
<td>165</td>
</tr>
<tr>
<td>Network</td>
<td>166</td>
</tr>
<tr>
<td>Network.formality</td>
<td>166</td>
</tr>
<tr>
<td>Network.lexical_classification</td>
<td>167</td>
</tr>
<tr>
<td>Network.polarity</td>
<td>168</td>
</tr>
<tr>
<td>new_project</td>
<td>168</td>
</tr>
<tr>
<td>ngrams</td>
<td>170</td>
</tr>
<tr>
<td>object_pronoun_type</td>
<td>172</td>
</tr>
<tr>
<td>outlier_detect</td>
<td>173</td>
</tr>
<tr>
<td>outlier_labeler</td>
<td>174</td>
</tr>
<tr>
<td>paste2</td>
<td>175</td>
</tr>
<tr>
<td>phrase_net</td>
<td>177</td>
</tr>
<tr>
<td>plot.animated_character</td>
<td>178</td>
</tr>
<tr>
<td>plot.animated_discourse_map</td>
<td>179</td>
</tr>
<tr>
<td>plot.animated_formality</td>
<td>179</td>
</tr>
<tr>
<td>plot.animated_lexical_classification</td>
<td>180</td>
</tr>
<tr>
<td>plot.animated_polarity</td>
<td>180</td>
</tr>
<tr>
<td>plot.automated_readability_index</td>
<td>181</td>
</tr>
<tr>
<td>plot.character_table</td>
<td>181</td>
</tr>
<tr>
<td>plot.cmspans</td>
<td>182</td>
</tr>
<tr>
<td>plot.cm_distance</td>
<td>182</td>
</tr>
<tr>
<td>plot.colesman_liau</td>
<td>183</td>
</tr>
<tr>
<td>plot.combo_syllable_sum</td>
<td>184</td>
</tr>
<tr>
<td>plot.cumulative_animated_formality</td>
<td>184</td>
</tr>
<tr>
<td>plot.cumulative_animated_lexical_classification</td>
<td>185</td>
</tr>
</tbody>
</table>
topics documented:

plot.cumulative_animated_polarity ........................................ 185
plot.cumulative_combo_syllable_sum ........................................ 186
plot.cumulative_end_mark .................................................... 186
plot.cumulative_formality .................................................... 187
plot.cumulative_lexical_classification ................................... 187
plot.cumulative_polarity ..................................................... 188
plot.cumulative_syllable_freq .............................................. 188
plot.discourse_map ............................................................. 189
plot.diversity ................................................................. 189
plot.end_mark ................................................................. 190
plot.end_mark_by ............................................................... 190
plot.end_mark_by_count ...................................................... 191
plot.end_mark_by_preprocessed ............................................. 191
plot.end_mark_by_proportion ............................................... 192
plot.end_mark_by_score ...................................................... 192
plot.flesch_kincaid ............................................................ 193
plot.formality ................................................................. 193
plot.formality_scores .......................................................... 194
plot.freq_terms ............................................................... 194
plot.gantt ................................................................. 195
plot.kullback_leibler ........................................................... 195
plot.lexical ................................................................. 196
plot.lexical_classification ................................................... 196
plot.lexical_classification_preprocessed ................................ 198
plot.lexical_classification_score .......................................... 198
plot.linsear_write ............................................................ 199
plot.linsear_write_count .................................................... 199
plot.linsear_write_scores .................................................... 200
plot.Network ................................................................. 200
plot.object_pronoun_type ..................................................... 201
plot.polarity ................................................................. 201
plot.polarity_count ........................................................... 203
plot.polarity_score .......................................................... 204
plot.pos ................................................................. 205
plot.pos_by ................................................................. 205
plot.pos_preprocessed ....................................................... 206
plot.pronoun_type ............................................................ 206
plot.question_type ............................................................ 207
plot.question_type_preprocessed .......................................... 207
plot.readability_count ....................................................... 208
plot.readability_score ........................................................ 208
plot.rmgantt ................................................................. 209
plot.sent_split .............................................................. 209
plot.SMOG ................................................................. 210
plot.subject_pronoun_type ................................................... 210
plot.sums_gantt .............................................................. 211
plot.sum_cmspans ............................................................ 211
plot.syllable_freq ........................................................... 212
plot.table_count .............................................. 212
plot.table_proportion ........................................ 213
plot.table_score .............................................. 213
plot.termco .................................................. 214
plot.type_token_ratio ....................................... 214
plot.weighted_wfm ......................................... 215
plot.wfdf .................................................... 215
plot.wfm ..................................................... 216
plot.word_cor ............................................... 216
plot.word_length .......................................... 217
plot.word_position ......................................... 218
plot.word_proximity ........................................ 218
plot.word_stats ............................................. 219
plot.word_stats_counts .................................... 219
polarity ...................................................... 220
pos ............................................................ 230
potential_NA .................................................. 235
preprocessed .................................................. 236
preprocessed.check_spelling_interactive ................. 236
preprocessed.end_mark_by ................................ 237
preprocessed.formality .................................... 237
preprocessed.lexical_classification ....................... 238
preprocessed.object_pronoun_type ......................... 238
preprocessed.pos ........................................... 239
preprocessed.pos_by ...................................... 239
preprocessed.pronoun_type ................................ 240
preprocessed.question_type ................................ 240
preprocessed.subject_pronoun_type ....................... 241
preprocessed.word_position ................................ 241
pres_debates2012 .......................................... 242
pres_debate_raw2012 ....................................... 242
print.adjacency_matrix .................................... 243
print.all_words ............................................ 243
print.animated_character ................................ 244
print.animated_discourse_map ............................. 244
print.animated_formality ................................ 245
print.animated_lexical_classification ................. 245
print.animated_polarity .................................. 246
print.automated_readability_index ....................... 247
print.boolean_qdap ........................................ 248
print.character_table .................................... 248
print.check_spelling ...................................... 249
print.check_spelling_interactive ....................... 249
print.check_text ........................................... 250
print.cm_distance ......................................... 250
print.coleman_liau ....................................... 251
print.colsplit2df ......................................... 251
print.combo_syllable_sum ................................ 252
print.cumulative_animated_formality ........................................ 252
print.cumulative_animated_lexical_classification ....................... 253
print.cumulative_animated_polarity ....................................... 253
print.cumulative_combo_syllable_sum ..................................... 254
print.cumulative_end_mark .................................................. 254
print.cumulative_formality ................................................ 255
print.cumulative_lexical_classification .................................. 255
print.cumulative_polarity ................................................... 256
print.cumulative_syllable_freq .......................................... 256
print.discourse_map ........................................................... 257
print.Dissimilarity ......................................................... 257
print.diversity .............................................................. 258
print.end_mark ............................................................... 258
print.end_mark_by ............................................................ 259
print.end_mark_by_preprocessed ........................................... 259
print.flesch_kincaid ......................................................... 260
print.formality ............................................................... 260
print.formality_scores ....................................................... 261
print.fry ........................................................................ 261
print.inspect_text ........................................................... 262
print.kullback_leibler ........................................................ 262
print.lexical_classification .................................................. 263
print.lexical_classification_by ............................................. 263
print.lexical_classification_preprocessed ................................ 264
print.lexical_classification_score ....................................... 264
print.linsear_write .......................................................... 265
print.linsear_write_count .................................................. 265
print.linsear_write_scores ................................................ 266
print.Network ............................................................... 266
print.ngrams ................................................................. 267
print.object_pronoun_type .................................................. 268
print.phrase_net ............................................................. 268
print.polarity ............................................................... 269
print.polarity_count ........................................................ 269
print.polarity_score ........................................................ 270
print.polysyllable_sum ...................................................... 270
print.pos ................................................................. 271
print.pos_by ............................................................... 271
print.pos_preprocessed ...................................................... 272
print.pronoun_type .......................................................... 272
print.qdapProj ............................................................. 273
print.qdap_context .......................................................... 273
print.question_type .......................................................... 274
print.question_type_preprocessed ........................................ 274
print.readability_count ...................................................... 275
print.readability_score ...................................................... 275
print.sent_split ............................................................. 276
print.SMOG ................................................................. 276
print.subject_pronoun_type ........................................ 277
print.sub_holder .................................................... 277
print.sums_gantt ..................................................... 278
print.sum_cmspans ................................................... 278
print.syllable_sum ................................................... 279
print.table_count ................................................... 279
print.table_proportion ............................................. 280
print.table_score .................................................... 280
print.termco .......................................................... 281
print.trunc ........................................................... 281
print.type_token_ratio .............................................. 282
print.wfm ............................................................. 282
print.wfm_summary ................................................... 283
print.which_misspelled ............................................. 283
print.word_associate ................................................. 284
print.word_cor ........................................................ 284
print.word_length .................................................... 285
print.word_list ........................................................ 285
print.word_position .................................................. 286
print.word_proximity ................................................ 286
print.word_stats ....................................................... 287
print.word_stats_counts ............................................. 287
pronoun_type .......................................................... 288
prop ................................................................. 290
proportions ............................................................ 291
proportions.character_table ........................................ 291
proportions.end_mark_by ............................................ 292
proportions.formality ................................................ 292
proportions.object_pronoun_type .................................. 293
proportions.pos ....................................................... 293
proportions.pos_by ................................................... 294
proportions.pronoun_type ........................................... 294
proportions.question_type .......................................... 295
proportions.subject_pronoun_type ................................ 295
proportions.termco ................................................... 296
proportions.word_length ............................................ 296
proportions.word_position ......................................... 297
qcombine ............................................................. 297
qcv ................................................................. 298
qdap ................................................................. 299
qdap_df ............................................................... 299
qheat ............................................................... 301
qprep ............................................................... 305
qtheme ............................................................. 306
question_type .......................................................... 310
raj ................................................................. 312
raj.act.1 ............................................................ 313
raj.act.1POS .......................................................... 313
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>raj.act.2</td>
<td>314</td>
</tr>
<tr>
<td>raj.act.3</td>
<td>315</td>
</tr>
<tr>
<td>raj.act.4</td>
<td>315</td>
</tr>
<tr>
<td>raj.act.5</td>
<td>316</td>
</tr>
<tr>
<td>raj.demographics</td>
<td>316</td>
</tr>
<tr>
<td>rajPOS</td>
<td>317</td>
</tr>
<tr>
<td>rajSPLIT</td>
<td>317</td>
</tr>
<tr>
<td>random_sent</td>
<td>317</td>
</tr>
<tr>
<td>rank_freq_mplot</td>
<td>319</td>
</tr>
<tr>
<td>raw.time.span</td>
<td>321</td>
</tr>
<tr>
<td>read.transcript</td>
<td>321</td>
</tr>
<tr>
<td>replacer</td>
<td>324</td>
</tr>
<tr>
<td>replace_abbreviation</td>
<td>325</td>
</tr>
<tr>
<td>replace_contraction</td>
<td>326</td>
</tr>
<tr>
<td>replace_number</td>
<td>327</td>
</tr>
<tr>
<td>replace_ordinal</td>
<td>328</td>
</tr>
<tr>
<td>replace_symbol</td>
<td>329</td>
</tr>
<tr>
<td>rm_row</td>
<td>330</td>
</tr>
<tr>
<td>rm_stopwords</td>
<td>331</td>
</tr>
<tr>
<td>sample.time.span</td>
<td>332</td>
</tr>
<tr>
<td>scores</td>
<td>333</td>
</tr>
<tr>
<td>scores.automated_readability_index</td>
<td>334</td>
</tr>
<tr>
<td>scores.character_table</td>
<td>334</td>
</tr>
<tr>
<td>scores.coleman_liu</td>
<td>335</td>
</tr>
<tr>
<td>scores[end_mark_by]</td>
<td>335</td>
</tr>
<tr>
<td>scores.flesch_kincaid</td>
<td>336</td>
</tr>
<tr>
<td>scores.formality</td>
<td>336</td>
</tr>
<tr>
<td>scores.fry</td>
<td>337</td>
</tr>
<tr>
<td>scores.lexical_classification</td>
<td>337</td>
</tr>
<tr>
<td>scores.linsear_write</td>
<td>338</td>
</tr>
<tr>
<td>scores.object_pronoun_type</td>
<td>338</td>
</tr>
<tr>
<td>scores.polarity</td>
<td>339</td>
</tr>
<tr>
<td>scores.pos_by</td>
<td>339</td>
</tr>
<tr>
<td>scores.pronoun_type</td>
<td>340</td>
</tr>
<tr>
<td>scores.question_type</td>
<td>340</td>
</tr>
<tr>
<td>scores.SMOG</td>
<td>341</td>
</tr>
<tr>
<td>scores.subject_pronoun_type</td>
<td>341</td>
</tr>
<tr>
<td>scores.termco</td>
<td>342</td>
</tr>
<tr>
<td>scores.word_length</td>
<td>342</td>
</tr>
<tr>
<td>scores.word_position</td>
<td>343</td>
</tr>
<tr>
<td>scores.word_stats</td>
<td>343</td>
</tr>
<tr>
<td>scrubber</td>
<td>344</td>
</tr>
<tr>
<td>Search</td>
<td>345</td>
</tr>
<tr>
<td>sentiment_frame</td>
<td>347</td>
</tr>
<tr>
<td>sentSplit</td>
<td>347</td>
</tr>
<tr>
<td>space_fill</td>
<td>350</td>
</tr>
<tr>
<td>spaste</td>
<td>351</td>
</tr>
<tr>
<td>speakerSplit</td>
<td>352</td>
</tr>
</tbody>
</table>
Index

+ .Network .................................................. 353
stemmer ...................................................... 353
strip .......................................................... 354
strWrap ......................................................... 355
subject_pronoun_type ................................. 356
summary.cmspans ......................................... 358
summary.wdf ............................................... 360
summary.wfm ............................................... 360
syllable_sum ................................................ 361
synonyms ..................................................... 363
t.DocumentTermMatrix ................................. 364
t.TermDocumentMatrix ................................. 365
termco .......................................................... 365
termco_c ...................................................... 370
Title ........................................................... 371
tot_plot ......................................................... 372
trans_cloud .................................................. 374
trans_context ................................................. 376
trans_venn .................................................... 378
Trim ............................................................ 379
type_token_ratio ............................................. 380
unique_by ...................................................... 381
vertex_apply .................................................. 381
visual .......................................................... 382
visual.discourse_map ..................................... 383
weight .......................................................... 383
wfm .............................................................. 384
word_associate ............................................... 390
word_cor ....................................................... 394
word_count .................................................... 396
word_diff_list ............................................. 400
word_length ................................................. 401
word_list ...................................................... 403
word_network_plot ........................................... 404
word_position ............................................... 406
word_proximity ............................................... 409
word_stats .................................................... 411
%\&% .......................................................... 414

+.Network ...................................................... 353

Add themes to a Network object.

Description

This operator allows you to add themes to a Network object.
add_incomplete

Usage

## S3 method for class 'Network'

Network.obj + x

Arguments

Network.obj  An object of class Network.
x  A component to add to Network.obj

Description

Automatically detect missing endmarks and replace with the | endmark symbol to indicate an incomplete sentence.

Usage

add_incomplete(text.var, endmarks = "[.?!]+$", silent = FALSE)

Arguments

text.var  The text variable.
endmarks  A regular expression to check for endmarks.
silent  logical. If TRUE messages are not printed out.

Value

Returns a vector with missing endmarks replaced with |.

Examples

add_incomplete(
  c("This in a",
     "I am funny!",
     "An ending of sorts!",
     "What do you want?"
  )
)
**add_s**

*Make Plural (or Verb to Singular) Versions of Words*

**Description**

Add -s, -es, or -ies to words.

**Usage**

```r
add_s(x, keep.original = TRUE)
```

**Arguments**

- `x` A vector of words to make plural.
- `keep.original` logical. If TRUE the original words are kept in the return vector.

**Value**

Returns a vector of plural words.

**Examples**

```r
set.seed(10)
add_s(sample(GradyAugmented, 10))
set.seed(10)
add_s(sample(GradyAugmented, 10), FALSE)
```

**adjacency_matrix**

*Takes a Matrix and Generates an Adjacency Matrix*

**Description**

Takes a matrix (wfm) or termco object and generates an adjacency matrix for use with the igraph package.

**Usage**

```r
adjacency_matrix(matrix.obj)
adjmat(matrix.obj)
```

**Arguments**

- `matrix.obj` A matrix object, preferably, of the class “termco” generated from termco, termco_d or termco_c.
Value

Returns list:

- boolean: A Boolean matrix
- adjacency: An adjacency matrix. Diagonals are the total (sum) number of occurrences a variable had
- shared: An adjacency matrix with no diagonal and the upper triangle replaced with NA
- sum: The diagonal of the adjacency matrix; the total (sum) number of occurrences a variable had

See Also

dist

Examples

```r
## Not run:
words <- c("you", "the", "it", "oo")
Terms <- with(DATA, termco(state, list(sex, adult), words))
adjacency_matrix(Terms)

wordLIST <- c("montague", "capulet", "court", "marry")
raj.termco <- with(raj.act.1, termco(dialogue, person, wordLIST))
raj.adjmat <- adjmat(raj.termco)
names(raj.adjmat) # see what's available from the adjacency_matrix object
library(igraph)
g <- graph.adjacency(raj.adjmat$adjacency, weighted=TRUE, mode ="undirected")
g <- simplify(g)
V(g)$label <- V(g)$name
V(g)$degree <- degree(g)
plot(g, layout=layout.auto(g))

## End(Not run)
```

all_words                  Searches Text Column for Words

Description

A convenience function to find words that begin with or contain a letter chunk and returns the frequency counts of the number of occurrences of each word.

Usage

```r
all_words(text.var, begins.with = NULL, contains = NULL,
  alphabetical = TRUE, apostrophe.remove = FALSE, char.keep = char2space,
  char2space = "~~", ...)```
Arguments

- **text.var**: The text variable.
- **begins.with**: This argument takes a word chunk. Default is NULL. Use this if searching for a word beginning with the word chunk.
- **contains**: This argument takes a word chunk. Default is NULL. Use this if searching for a word containing the word chunk.
- **alphabetical**: logical. If TRUE orders rows alphabetically, if FALSE orders the rows by descending frequency.
- **apostrophe.remove**: logical. If TRUE removes apostrophes from the text before examining.
- **char.keep**: A character vector of symbol character (i.e., punctuation) that strip should keep. The default is to strip everything except apostrophes. This enables the use of special characters to be turned into spaces or for characters to be retained.
- **char2space**: A vector of characters to be turned into spaces.
- **...**: Other argument supplied to strip.

Value

Returns a dataframe with frequency counts of words that begin with or contain the provided word chunk.

Note

Cannot provide both begins.with and contains arguments at once. If both begins.with and contains are NULL. all_words returns a frequency count for all words.

See Also

term_match

Examples

```r
# Not run:
x1 <- all_words(raj$dialogue, begins.with="re")
head(x1, 10)
x2 <- all_words(raj$dialogue, "q")
head(x2, 10)
all_words(raj$dialogue, contains="con")
x3 <- all_words(raj$dialogue)
head(x3, 10)
x4 <- all_words(raj$dialogue, contains="the")
head(x4)
x5 <- all_words(raj$dialogue, contains="read")
head(x5)

# Filter by nchar and stopwords
Filter(head(x3), min = 3)
```
## Animate

```
## Keep spaces
all_words(space_fill(DATA$state, c("are you", "can be")))
## End(Not run)
```

### Generic Animate Method

**Description**

Animate select qdap objects.

**Usage**

```
Animate(x, ...)
```

**Arguments**

- `x`: An animatable qdap object (e.g., `discourse_map`).
- `...`: Arguments passed to Animate method of other classes.

**Value**

Returns a plot object.

**See Also**

- `scores`, `counts`, `preprocessed`, `proportions`

### Animate.character

**Animate Character**

**Description**

Animate.character - Animate a `character` object. Typically this function is useful in conjunction with other Animate objects to create complex animations with accompanying text.

**Usage**

```
## S3 method for class 'character'
Animate(x, wc.time = TRUE, time.constant = 2,
   width = 65, coord = c(0, 0.5), just = c(0, 0.5), size = 5,
   color = "black", border.color = NA, ...)
```
Arguments

- **x**: A character object.
- **wc.time**: logical. If TRUE weights duration of frame by word count.
- **time.constant**: A constant to divide the maximum word count by. Time is calculated by \( \text{round}(\exp(\text{WORD COUNT}/(\text{max(WORD COUNT)/time.constant)})) \). Therefore a larger constant will make the difference between the large and small word counts greater.
- **width**: The width to break text at if type = "text".
- **coord**: The x/y coordinate to plot the text.
- **just**: The hjust and vjust values to use for the text.
- **size**: The size to print the text. Can be a vector of length 1 or equal to the length of x.
- **color**: The color to print the text. Can be a vector of length 1 or equal to the length of x.
- **border.color**: The panel.border color (see theme).
- **...**: Other arguments passed to annotate.

Details

character Method for Animate

See Also

theme

Examples

```r
## Not run:
Animate(DATA[['state']])
Animate(DATA[['state']], color="red")
Animate(DATA[['state']], color=RColorBrewer::brewer.pal(11, "Set3"), size=10)
cls <- DATA[['person']] %>% data.frame(levels(DATA[['person']]),
  RColorBrewer::brewer.pal(5, "Set3")
Animate(DATA[['state']], color=cls, size=10, width=30)
cls2 <- DATA[['sex']] %>% data.frame(c("m", "f"),c("lightblue", "pink"))
Animate(DATA[['state']], color=cls2, just=c(.5, .5), coord = c(.5, .5))

## Print method
print(Animate(DATA[['state']]), color=cls2, just=c(.5, .5), coord = c(.5, .5), pause=.25)
Animate(DATA[['state']], color=sample(colors()), nrow(DATA)),
  size=sample(4:13, nrow(DATA), TRUE), width=30, just=c(.5, .5), coord = c(.5, .5))

## End(Not run)
```
Animate.discourse_map

**Discourse Map**

### Description

**Animate.discourse_map** - Animate a discourse discourse_map.

### Usage

```r
## S3 method for class 'discourse_map'
Animate(x, edge.constant, sep = "_",
    current.color = "red", previous.color = "grey50", wc.time = TRUE,
    time.constant = 2, title = NULL, ...)
```

### Arguments

- **x** The discourse_map object.
- **edge.constant** A constant to multiple edge width by.
- **sep** The separator character to use between grouping variables.
- **current.color** The color to make the vector edge as it moves.
- **previous.color** The color to make the already plotted edges.
- **wc.time** Logical. If TRUE weights duration of frame by word count.
- **time.constant** A constant to divide the maximum word count by. Time is calculated by `round(exp(WORD COUNT/(max(WORD COUNT)/time.constant)))`. Therefore a larger constant will make the difference between the large and small word counts greater.
- **title** The title to apply to the animated image(s).
- **...** Ignored

### Details

discourse_map Method for Animate

### Note

The width of edges is based on words counts on that edge until that moment divided by total number of words used until that moment. Thicker edges tend to thin as time passes. The actual duration the current edge stays as the current.color is based on word counts for that particular flow of dialogue divided by total dialogue (words) used.
Description

Animate.formality - Animate a formality object.

Usage

```r
## S3 method for class 'formality'
Animate(x, contextual = "yellow", formal = "red",
        edge.constant, wc.time = TRUE, time.constant = 2, title = NULL,
        digits = 3, current.color = "black", current.speaker.color = NULL,
        non.speaker.color = NA, missing.color = "purple",
        all.color.line = "red", plus.300.color = "grey40",
        under.300.color = "grey88", type = "network", width = 65, coord = c(0,
        0.5), just = c(0, 0.5), ...)  
```

Arguments

- `x`: A formality object.
- `contextual`: The color to use for 0% formality (purely contextual).
- `formal`: The color to use for 100% formality (purely formal).
- `edge.constant`: A constant to multiple edge width by.
- `wc.time`: logical. If TRUE weights duration of frame by word count.
- `time.constant`: A constant to divide the maximum word count by. Time is calculated by `round(exp(WORD COUNT)/(max(WORD COUNT)/time.constant)))`. Therefore a larger constant will make the difference between the large and small word counts greater.
- `title`: The title to apply to the animated image(s).
- `digits`: The number of digits to use in the current turn of talk formality.
- `current.color`: The color to use for the current turn of talk formality.
- `current.speaker.color`: The color for the current speaker.
- `non.speaker.color`: The color for the speakers not currently speaking.
- `missing.color`: The color to use in a network plot for edges corresponding to missing text data. Use `na.omit` before hand to remove the missing values all together.
- `all.color.line`: The color to use for the total discourse formality color line if network = FALSE.
- `plus.300.color`: The bar color to use for grouping variables exceeding 299 words per Heylighen & Dewaele's (2002) minimum word recommendations.
- `under.300.color`: The bar color to use for grouping variables less than 300 words per Heylighen & Dewaele’s (2002) minimum word recommendations.
type: Character string of either "network" (as a network plot), "bar" (as a bar plot), or "text" (as a simple colored text plot).
width: The width to break text at if type = "text".
coord: The x/y coordinate to plot the text if type = "text".
just: The hjust and vjust values to use for the text if type = "text".
... Other arguments passed to discourse_map or annotate if type = "text".

Details

formality Method for Animate

Note

The width of edges is based on words counts on that edge until that moment divided by total number of words used until that moment. Thicker edges tend to thin as time passes. The actual duration the current edge stays as the current.color is based on word counts for that particular flow of dialogue divided by total dialogue (words) used. The edge label is the current formality for that turn of talk (an aggregation of the sub sentences of the current turn of talk). The coloring of the current edge formality is produced at the sentence level, therefore a label may indicate a positive current turn of talk, while the coloring may indicate a negative sentences. Coloring is based on percentage of formal parts of speech (i.e., noun, adjective, preposition, article).

### Animate.gantt

**Gantt Durations**

Description

gantt - Animate discourse from gantt.

Usage

```r
## S3 method for class 'gantt'
Animate(x, wc.time = TRUE, time.constant = 2, colors = NULL, ...)
```

Arguments

- `x`: The gantt object.
- `wc.time`: logical. If TRUE weights duration of frame by word count.
- `time.constant`: A constant to divide the maximum word count by. Time is calculated by `round(exp(WORD COUNT/(max(WORD COUNT)/time.constant)))`. Therefore a larger constant will make the difference between the large and small word counts greater.
- `colors`: An optional character vector of colors to color the Gantt bars. Must be length 1 (repeats the same color) or equal to the levels of the grouping variable.
- `...`: Other arguments passed to gantt_wrap.
Details

gantt Method for Animate

Description

gantt_plot - Animate discourse from gantt_wrap, gantt_plot, or any other Gantt plotting method.

Usage

```r
## S3 method for class 'gantt_plot'
Animate(x, wc.time = TRUE, time.constant = 2,
        colors = NULL, ...)
```

Arguments

- `x` The gantt_plot object.
- `wc.time` logical. If TRUE weights duration of frame by word count.
- `time.constant` A constant to divide the maximum word count by. Time is calculated by `round(exp(WORD COUNT/max(WORD COUNT)/time.constant))`. Therefore a larger constant will make the difference between the large and small word counts greater.
- `colors` An optional character vector of colors to color the Gantt bars. Must be length 1 (repeats the same color) or equal to the levels of the grouping variable.
- `...` ignored

Details

gantt_plot Method for Animate

Description

Animate.lexical_classification - Animate a lexical_classification object.
Usage

## S3 method for class 'lexical_classification'

Animate(x, type = "network",
        content = "red", functional = "yellow", edge.constant, wc.time = TRUE,
        time.constant = 2, title = NULL, digits = 2, current.color = "black",
        current.speaker.color = NULL, non.speaker.color = NA,
        missing.color = "purple", all.color.line = "red", width = 65,
        function.words = qdapDictionaries::function.words, left = "<<",
        right = ">>", coord = c(0, 0.5), just = c(0, 0.5), ...)

Arguments

- **x**: A `lexical_classification` object.
- **type**: Character string of either "network" (as a network plot), "bar" (as a bar plot), or "text" (as a simple colored text plot).
- **content**: The color to use for 100% `lexical_classification` (purely content).
- **functional**: The color to use for 0% `lexical_classification` (purely functional).
- **edge.constant**: A constant to multiple edge width by.
- **wc.time**: logical. If TRUE weights duration of frame by word count.
- **time.constant**: A constant to divide the maximum word count by. Time is calculated by `round(exp(WORD COUNT/(max(WORD COUNT)/time.constant)))`. Therefore a larger constant will make the difference between the large and small word counts greater.
- **title**: The title to apply to the animated image(s).
- **digits**: The number of digits to use in the current turn of talk's content rate.
- **current.color**: The color to use for the current turn of talk's content rate.
- **current.speaker.color**: The color for the current speaker.
- **non.speaker.color**: The color for the speakers not currently speaking.
- **missing.color**: The color to use in a network plot for edges corresponding to missing text data. Use `na.omit` before hand to remove the missing values all together.
- **all.color.line**: The color to use for the total average discourse content rate.
- **width**: The width to break text at if `type = "text"`.
- **function.words**: A vector of function words. Default is `function.words`.
- **left**: A left bound to wrap content words with if `type = "text"`.
- **right**: A right bound to wrap content words with if `type = "text"`.
- **coord**: The x/y coordinate to plot the text if `type = "text"`.
- **just**: The vjust values to use for the text if `type = "text"`.
- **...**: Other arguments passed to `discourse_map` or `annotate` if `type = "text"`.

Details

`lexical_classification` Method for Animate
Note
The width of edges is based on words counts on that edge until that moment divided by total number of words used until that moment. Thicker edges tend to thin as time passes. The actual duration the current edge stays as the current.color is based on word counts for that particular flow of dialogue divided by total dialogue (words) used. The edge label is the current content rate for that turn of talk (an aggregation of the sub sentences of the current turn of talk). The coloring of the current edge content rate is produced at th sentence level, therefor a label may indicate a more content laden current turn of talk, while the coloring may indicate a functional laden average of sentences. Coloring is based on percentage of content words.

Animate.polarity

Description
Animate.polarity - Animate a polarity object.

Usage
```r
## S3 method for class 'polarity'
Animate(x, negative = "blue", positive = "red",
        neutral = "yellow", edge.constant, wc.time = TRUE, time.constant = 2,
        title = NULL, digits = 3, width = 65, current.color = "black",
        current.speaker.color = NULL, non.speaker.color = NA,
        ave.color.line = "red", type = "network", coord = c(0, 0.5),
        just = c(0, 0.5), ...)
```

Arguments

- **x**: A polarity object.
- **negative**: The color to use for negative polarity.
- **positive**: The color to use for positive polarity.
- **neutral**: The color to use for neutral polarity.
- **edge.constant**: A constant to multiple edge width by.
- **wc.time**: logical. If TRUE weights duration of frame by word count.
- **time.constant**: A constant to divide the maximum word count by. Time is calculated by `round(exp(WORD COUNT/(max(WORD COUNT)/time.constant)))`. Therefore a larger constant will make the difference between the large and small word counts greater.
- **title**: The title to apply to the animated image(s).
- **digits**: The number of digits to use in the current turn of talk polarity.
- **width**: The width to break text at if type = "text".
- **current.color**: The color to use for the current turn of talk polarity.
- **current.speaker.color**: The color for the current speaker.
non.speaker.color
The color for the speakers not currently speaking.

ave.color.line
The color to use for the average color line if type = "network".

type
Character string of either "network" (as a network plot), "bar" (as a bar plot), or "text" (as a simple colored text plot).

coord
The x/y coordinate to plot the test if type = "text".

just
The hjust and vjust values to use for the text if type = "text".

... Other arguments passed to discourse_map or annotate if type = "text".

Details

polarity Method for Animate

Note

The width of edges is based on words counts on that edge until that moment divided by total number of words used until that moment. Thicker edges tend to thin as time passes. The actual duration the current edge stays as the current.color is based on word counts for that particular flow of dialogue divided by total dialogue (words) used. The edge label is the current polarity for that turn of talk (an aggregation of the sub sentences of the current turn of talk). The coloring of the current edge polarity is produced at sentence level, therefore a label may indicate a positive current turn of talk, while the coloring may indicate a negative sentences.

as.tdm

Description

as.tdm - Create term document matrices from raw text or wfm for use with other text analysis packages.
as.TermDocumentMatrix - Create document term matrices from raw text or wfm for use with other text analysis packages.
as.dtm - Create document term matrices from raw text or wfm for use with other text analysis packages.
as.DocumentTermMatrix - Create document term matrices from raw text or wfm for use with other text analysis packages.
as.tdm.Corpus - Corpus method for as.tdm used to convert to a DocumentTermMatrix.
as.tdm.default - Default method for as.tdm used to convert to a TermDocumentMatrix.
as.tdm.character - character method for as.tdm used to convert to a TermDocumentMatrix.
as.dtm.Corpus - Corpus method for as.dtm used to convert to a DocumentTermMatrix.
as.dtm.default - Default method for as.dtm used to convert to a DocumentTermMatrix.
as.dtm.character - character method for as.dtm used to convert to a DocumentTermMatrix.
as.tdm.wfm - wfm method for as.tdm used to convert to a `TermDocumentMatrix`.

as.data.frame - Convert a `tm` package Corpus to a `qdap` data.frame.

as.Corpus - Attempts to convert its argument into a `tm` package Corpus.

as.Corpus.default - Default method for as.Corpus used to convert vectors (from a data.frame) to a Corpus.

apply_as_tm - Apply functions intended to be used on the `tm` package's `TermDocumentMatrix` to a `wfm` object.

apply_as_df - Apply a `tm` Corpus as a qdap dataframe. apply_as_df - Apply functions intended to be used on the `qdap` package's `data.frame` + `sentSplit` to a `tm` Corpus object.

as.Corpus.TermDocumentMatrix - TermDocumentMatrix method for as.Corpus used to convert a Corpus.

as.Corpus.DocumentTermMatrix - DocumentTermMatrix method for as.Corpus used to convert a Corpus.

as.Corpus.wfm - wfm method for as.Corpus used to convert a Corpus.

Usage

as.tdm(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

as.TermDocumentMatrix(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

as.dtm(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

as.DocumentTermMatrix(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

## S3 method for class 'Corpus'
as.tdm(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

## Default S3 method:
as.tdm(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

## S3 method for class 'character'
as.tdm(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

## S3 method for class 'Corpus'
as.dtm(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

## Default S3 method:
as.dtm(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

## S3 method for class 'character'
as.dtm(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

## S3 method for class 'wfm'
as.dtm(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

## S3 method for class 'wfm'
as.dtm(text.var, grouping.var = NULL, vowel.check = TRUE, ...)

## S3 method for class 'Corpus'
as.data.frame(x, row.names, optional, ..., doc = "doc_id", text = "text", sent.split = FALSE)
as.Corporus(text.var, grouping.var = NULL, demographic.vars, ...)

## S3 method for class 'sent_split'
as.Corporus(text.var, grouping.var = NULL, demographic.vars, ...)

## Default S3 method:
as.Corporus(text.var, grouping.var = NULL, demographic.vars, ...)

apply_as_tm(wfm.obj, tmfun, ..., to.qdap = TRUE)

apply_as_df(tm.corpus, qdapfun, ..., stopwords = NULL, min = 1, max = Inf, count.apostrophe = TRUE, ignore.case = TRUE)

## S3 method for class 'TermDocumentMatrix'
as.Corporus(text.var, ...)

## S3 method for class 'DocumentTermMatrix'
as.Corporus(text.var, ...)

## S3 method for class 'wfm'
as.Corporus(text.var, ...)

Arguments

- **text.var**: The text variable or a wfm object.
- **grouping.var**: The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- **vowel.check**: logical. Should terms without vowels be remove?
- **x**: A Corpus object.
row.names NULL or a character vector giving the row names for the data frame. Not used in `qdap`; for base generic consistency.

optional logical. If TRUE, setting row names and converting column names is optional. Not used in `qdap`; for base generic consistency.

doc Name for `Corpus` documents.

text Name for `Corpus` text.

sent.split logical. If TRUE the text variable sentences will be split into individual rows.

demographic.vars Additional demographic information about the grouping variables. This is a data.frame, list of equal length vectors, or a single vector corresponding to the grouping variable/text variable. This information will be mapped to the DMetaData in the `Corpus`.

wfm.obj A `wfm` object.

tmfun A function applied to a `TermDocumentMatrix` object.

to.qdap logical. If TRUE should `wfm` try to coerce the output back to a `qdap` object.

tm.corpus A `Corpus` object.

tmfun A `qdap` function that is usually used on text.variable ~ grouping variable.

stopwords A character vector of words to remove from the text. `qdap` has a number of data sets that can be used as stop words including: Top200Words, Top100Words, Top25Words. For the tm package’s traditional English stop words use `tm::stopwords("english")`.

min Minimum word length.

max Maximum word length.

count.apostrophe logical. If TRUE apostrophes are counted as characters.

ignore.case logical. If TRUE stop words will be removed regardless of case.

... Function dependant:
  * `as.tdm` or `as.dtm` - Other arguments passed to `wfm`
  * `apply_as_tm` - Other arguments passed to functions used on a `tm` `TermDocumentMatrix`
  * `as.data.frame` - Other arguments passed to `sentSplit`
  * `as.Corpus` - Other arguments passed to `tm`’s `Corpus`

Details

Produces output that is identical to the tm package’s `TermDocumentMatrix`, `DocumentTermMatrix`, `Corpus` or allows convenient interface between the qdap and tm packages.

Value

`as.tdm` - Returns a `TermDocumentMatrix`.

`as.TermDocumentMatrix` - Returns a `TermDocumentMatrix`.

`as.dtm` - Returns a `DocumentTermMatrix`.

as.data.frame - Converts a Corpus and returns a qdap oriented data.frame.
as.Corpus - Converts a qdap oriented dataframe and returns a Corpus.
apply_as_tm - Applies a tm oriented function to a wfm and attempts to simplify back to a wfm or weight format.
apply_as_df - Returns the output typical of the applied qdap function.

Note
apply_as_df coerces to a dataframe with columns named ‘docs’ and the other named ‘text’.

See Also
DocumentTermMatrix, Corpus, TermDocumentMatrix, as.wfm
Filter

Examples
## Not run:
as.dtm(DATA$state, DATA$person)
as.tdm(DATA$state, DATA$person)

x <- wfm(DATA$state, DATA$person)
as.tdm(x)
as.dtm(x)
library(tm)
plot(as.tdm(x))

pres <- as.tdm(pres_debates2012$dialogue, pres_debates2012$person)
plot(pres, corThreshold = 0.8)
pres
(pres2 <- removeSparseTerms(pres, .3))
plot(pres2, corThreshold = 0.95)

shorts <- all_words(pres_debates2012)[,1][nchar(all_words(
pres_debates2012)[,1]) < 4]

SW <- c(shorts, qdapDictionaries::contractions[, 1],
qdapDictionaries::Top200Words,
"governor", "president", "mister", "obama", "romney")

DocTermMat2 <- with(pres_debates2012, as.dtm(dialogue, list(person, time), stopwords = SW))
DocTermMat2 <- removeSparseTerms(DocTermMat2, 0.95)
(DocTermMat2 <- DocTermMat2[rowSums(as.matrix(DocTermMat2)) > 0,])
plot(DocTermMat2)

## Correspondence Analysis
library(ca)

dat <- pres_debates2012
dat <- dat[dat$person %in% qcv(ROMNEY, OBAMA),]
speech <- stemmer(dat$dialogue)
mytable1 <- with(dat, as.tdm(speech, list(person, time), stopwords = Top25Words))

fit <- ca(as.matrix(mytable1))
summary(fit)
plot(fit)
plot3d.ca(fit, labels = 1)

mytable2 <- with(dat, as.tdm(speech, list(person, time), stopwords = Top200Words))

fit2 <- ca(as.matrix(mytable2))
summary(fit2)
plot(fit2)
plot3d.ca(fit2, labels = 1)

## Topic Models
# Example 1
library(topicmodels); library(tm)

# Generate stop words based on short words, frequent words and contractions
shorts <- all_words(pres_debates2012)[,1][nchar(all_words(pres_debates2012)[,1]) < 4]

SW <- c(shorts, qdapDictionaries::contractions[, 1],
qdapDictionaries::Top200Words,
"governor", "president", "mister", "obama", "romney")

DocTermMat <- with(pres_debates2012, as.dtm(dialogue, person, stopwords = SW))
DocTermMat <- removeSparseTerms(DocTermMat, 0.999)
DocTermMat <- DocTermMat[rowSums(as.matrix(DocTermMat)) > 0,]

lda.model <- LDA(DocTermMat, 5)

(topics <- posterior(lda.model, DocTermMat)$topics)
terms(lda.model, 20)

# Plot the Topics Per Person
topic.dat <- matrix2df(topics, "Person")
colnames(topic.dat)[[-1]] <- paste2(t(terms(lda.model, 20)), sep="", "

library(reshape2)
mtopic <- melt(topic.dat, variable="Topic", value.name="Proportion")
ggplot(mttopic, aes(weight=Proportion, x=Topic, fill=Topic)) +
  geom_bar() +
  coord_flip() +
  facet_grid(Person~.) +
  guides(fill=FALSE)

# Example 2
DocTermMat2 <- with(pres_debates2012, as.dtm(dialogue, list(person, time), stopwords = SW))
DocTermMat2 <- removeSparseTerms(DocTermMat2, 0.95)
DocTermMat2 <- DocTermMat2[rowSums(as.matrix(DocTermMat2)) > 0,]
lda.model2 <- LDA(DocTermMat2, 6)
(topics2 <- posterior(lda.model2, DocTermMat2)$topics)
terms(lda.model2, 20)
qheat(topics2, high="blue", low="yellow", by.col=FALSE)

# Example 3 #
lda.model3 <- LDA(DocTermMat2, 10)
(topics3 <- posterior(lda.model3, DocTermMat2)$topics)
terms(lda.model3, 20)
qheat(topics3, high="blue", low="yellow", by.col=FALSE)

# Plot the Topics Per Person
topic.dat3 <- matrix2df(topics3, "Person&Time")
colnames(topic.dat3)[-1] <- paste2(t(terms(lda.model3, 10)), sep="", "")
topic.dat3 <- colsplit2df(topic.dat3)

library(reshape2)
library(scales)
mtopic3 <- melt(topic.dat3, variable="Topic", value.name="Proportion")
p1 <- ggplot(mtopic3, aes(weight=Proportion, x=Topic, fill=Topic)) +
  geom_bar() +
  coord_flip() +
  facet_grid(Person~Time) +
  guides(fill=FALSE) +
  scale_y_continuous(labels = percent) +
  theme(plot.margin = unit(c(1, 0, .5, .5), "lines")) +
  ylab("Proportion")

mtopic3.b <- mtopic3
mtopic3.b[, "Topic"] <- factor(as.numeric(mtopic3.b[, "Topic"]), levels = 1:10)

mtopic3.b[, "Time"] <- factor(gsub("time ", "", mtopic3.b[, "Time"]))

p2 <- ggplot(mtopic3.b, aes(x=Time, y=Topic, fill=Proportion)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "grey70", high = "red") +
  facet_grid(Person~Time, scales = "free") +
  theme(axis.title.y = element_blank(),
        axis.text.x= element_text(colour="white"),
        axis.ticks.x= element_line(colour="white"),
        axis.ticks.y = element_blank(),
        axis.text.y= element_blank(),
        plot.margin = unit(c(.5, .5, .5, .5), "lines")
    )

library(gridExtra)
gird.arrange(p1, p2, nrow=1, widths = grid::unit(c(.85, .15), "native"))

## tm Matrices to wfm
library(tm)
data(crude)
## A Term Document Matrix Conversion

```r
(tm_in <- TermDocumentMatrix(crude, control = list(stopwords = TRUE)))
converted <- as.wfm(tm_in)
head(converted)
summary(converted)
```

## A Document Term Matrix Conversion

```r
(dtm_in <- DocumentTermMatrix(crude, control = list(stopwords = TRUE)))
summary(as.wfm(dtm_in))
```

## `apply_as_tm` Examples

### Create a wfm

```r
a <- with(DATA, wfm(state, list(sex, adult)))
summary(a)
```

### Apply functions meant for a tm TermDocumentMatrix

```r
out <- apply_as_tm(a, tm:::removeSparseTerms, sparse=0.6)
summary(out)
```

```r
apply_as_tm(a, tm:::findAssocs, "computer", .8)
apply_as_tm(a, tm:::findFreqTerms, 2, 3)
apply_as_tm(a, tm:::Zipf_plot)
apply_as_tm(a, tm:::Heaps_plot)
apply_as_tm(a, tm:::plot.TermDocumentMatrix, corThreshold = 0.4)
```

### Convert tm Corpus to Dataframe

#### A tm Corpus

```r
library(tm)
reut21578 <- system.file("texts", "crude", package = "tm")
reuters <- Corpus(DirSource(reut21578),
                  readerControl = list(reader = readReut21578XML))
```

#### Convert to dataframe

```r
corp_df <- as.data.frame(reuters)
htruncdf(corp_df)
```

```r
z <- as.Corpus(DATA$state, DATA$person,
     demographic=DATA[, qcv(sex, adult, code)])
as.data.frame(z)
```

### Apply a qdap function

```r
out <- formality(corp_df$text, corp_df$docs)
plot(out)
```

### Convert a qdap dataframe to tm package Corpus

```r
(x <- with(DATA2, as.Corpus(state, list(person, class, day))))
```
library(tm)
inspect(x)
inspect_text(x)
class(x)

(y <- with(pres.debates2012, as.Corpus(dialogue, list(person, time))))

## Add demographic info to DMetaData of Corpus
z <- as.Corpus(DATA$state, DATA$person,
    demographic=DATA[, qcv(sex, adult, code)])
lview(z)
lview(as.Corpus(DATA$state, DATA$person,
    demographic=DATA$sex))
lview(as.Corpus(DATA$state, DATA$person,
    demographic=list(DATA$sex, DATA$adult)))

## Apply qdap functions meant for dataframes from sentSplit to tm Corpus
library(tm)
reuters1578 <- system.file("texts", "crude", package = "tm")
reuters <- Corpus(DirSource(reuters1578),
    readerControl = list(reader = readReuter21578XML))
matches <- list(
    oil = qcv(oil, crude),
    money = c("economic", "money")
)
apply.as.df(reuters, word.stats)
apply.as.df(reuters, formality)
apply.as.df(reuters, word_list)
apply.as.df(reuters, polarity)
apply.as.df(reuters, Dissimilarity)
apply.as.df(reuters, diversity)
apply.as.df(reuters, pos_by)
apply.as.df(reuters, flesch_kincaid)
apply.as.df(reuters, trans_venn)
apply.as.df(reuters, gantt_plot)
apply.as.df(reuters, rank_freq_mplot)
apply.as.df(reuters, character_table)

(termco_out <- apply.as.df(reuters, termco, match.list = matches))
plot(termco_out, values = TRUE, high="red")

(wordcor_out <- apply.as.df(reuters, word_cor, word = unlist(matches)))
plot(wordcor_out)

(f_terms <- apply.as.df(reuters, freq_terms, at.least = 3))
plot(f_terms)

apply.as.df(reuters, trans_cloud)
## To use "all" rather than "docs" as "grouping.var"...
```r
apply_as_df(reuters, trans_cloud, grouping.var=NULL,
              target.words=matches, cloud.colors = c("red", "blue", "grey75"))

finds <- apply_as_df(reuters, freq_terms, at.least = 5,
                      top = 5, stopwords = Top100Words)
apply_as_df(reuters, dispersion_plot, match.terms = finds[, 1],
           total.color = NULL)

## Filter for Term Document Matrix/Document Term Matrix
library(tm)
data(crude)

(tdm_in <- TermDocumentMatrix(crude, control = list(stopwords = TRUE)))
Filter(tdm_in, 5)

(dtm_in <- DocumentTermMatrix(crude, control = list(stopwords = TRUE)))
Filter(dtm_in, 5)

## Filter particular words based on max/min values
Filter(dtm_in, 5, 7)
Filter(dtm_in, 4, 4)
Filter(tdm_in, 3, 4)
Filter(tdm_in, 3, 4, stopwords = Top200Words)

## SPECIAL REMOVAL OF TERMS (more flexible consideration of words than wfm)
data <- data.frame(
  person = paste0("person_", 1:5),
  tweets = c("test one two", "two apples","hashtag apple",
             "#apple #tree", "http://microsoft.com")
)

## remove specialty items
data[[2]] <- rm_default(data[[2]], pattern=pastex("@rm_url", "#apple\b"))

myCorp <- tm::tm_map(crude, tm::removeWords, Top200Words)
myCorp %>% as.dtm() %>% tm::inspect()

## End(Not run)
```
SMOG - Apply SMOG Readability to transcript(s) by zero or more grouping variable(s).
flesch_kincaid - Flesch-Kincaid Readability to transcript(s) by zero or more grouping variable(s).
fry - Apply Fry Readability to transcript(s) by zero or more grouping variable(s).
linsear_write - Apply Linsear Write Readability to transcript(s) by zero or more grouping variable(s).

Usage
automated_readability_index(text.var, grouping.var = NULL,
   rm.incomplete = FALSE, ...)

coleman_liau(text.var, grouping.var = NULL, rm.incomplete = FALSE, ...)

SMOG(text.var, grouping.var = NULL, output = "valid",
   rm.incomplete = FALSE, ...)

flesch_kincaid(text.var, grouping.var = NULL, rm.incomplete = FALSE, ...)

fry(text.var, grouping.var = NULL, rm.incomplete = FALSE,
   auto.label = TRUE, grid = FALSE, div.col = "grey85", plot = TRUE, ...)

linsear_write(text.var, grouping.var = NULL, rm.incomplete = FALSE, ...)

Arguments
text.var The text variable.
grouping.var The grouping variables. Default NULL generates one output for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
rm.incomplete logical. If TRUE removes incomplete sentences from the analysis.
output A character vector character string indicating output type. One of "valid" (default and congruent with McLaughlin’s intent) or "all".
auto.label logical. If TRUE labels automatically added. If FALSE the user clicks interactively.
grid logical. If TRUE a micro grid is displayed, similar to Fry’s original depiction, though this may make visualizing more difficult.
div.col The color of the grade level division lines.
plot logical. If TRUE a graph is plotted corresponding to Fry’s graphic representation.
... Other arguments passed to end_inc.

Value
Returns a list of 2 dataframes: (1) Counts and (2) Readability. Counts are the raw scores used to calculate readability score and can be accessed via counts. Readability is the dataframe with the selected readability statistic by grouping variable(s) and can be access via scores. The fry function returns a graphic representation of the readability as the scores returns the information for graphing but not a readability score.
Warning

Many of the indices (e.g., Automated Readability Index) are derived from word difficulty (letters per word) and sentence difficulty (words per sentence). If you have not run the sentSplit function on your data the results may not be accurate.

Fry

The fry function is based on Fry’s formula that randomly samples 3 100 word length passages. If a group(s) in does not contain 300+ words they will not be included in the output.

References


Examples

```r
# Not run:
AR1 <- with(rajSPLIT, automated_readability_index(dialogue, list(person, act)))
ltruncdf(AR1, 15)
scores(AR1)
counts(AR1)
plot(AR1)
plot(counts(AR1))

AR2 <- with(rajSPLIT, automated_readability_index(dialogue, list(sex, fam.aff)))
ltruncdf(AR2, 15)
scores(AR2)
counts(AR2)
plot(AR2)
plot(counts(AR2))

AR3 <- with(rajSPLIT, automated_readability_index(dialogue, person))
ltruncdf(AR3, 15)
scores(AR3)
head(counts(AR3))
plot(AR3)
```
plot(counts(AR3))

CL1 <- with(rajSPLIT, coleman_liau(dialogue, list(person, act)))
ltruncdf(CL1, 20)
head(counts(CL1))
plot(CL1)

CL2 <- with(rajSPLIT, coleman_liau(dialogue, list(sex, fam.aff)))
ltruncdf(CL2)
plot(counts(CL2))

(SM1 <- with(rajSPLIT, SMOG(dialogue, list(person, act))))
plot(counts(SM1))
plot(SM1)

(SM2 <- with(rajSPLIT, SMOG(dialogue, list(sex, fam.aff))))

(FL1 <- with(rajSPLIT, flesch_kincaid(dialogue, list(person, act))))
plot(scores(FL1))
plot(counts(FL1))

(FL2 <- with(rajSPLIT, flesch_kincaid(dialogue, list(sex, fam.aff))))
plot(scores(FL2))
plot(counts(FL2))

FR1 <- with(rajSPLIT, fry(dialogue, list(sex, fam.aff)))
scores(FR1)
plot(scores(FR1))
counts(FR1)
plot(counts(FR1))

FR2 <- with(rajSPLIT, fry(dialogue, person))
scores(FR2)
plot(scores(FR2))
counts(FR2)
plot(counts(FR2))

FR3 <- with(pres_debates2012, fry(dialogue, list(time, person)))
colsplit2df(scores(FR3))
plot(scores(FR3), auto.label = FALSE)
counts(FR3)
plot(counts(FR3))

library(ggplot2)
ggplot(colsplit2df(counts(FR3)), aes(sent.per.100.wrds, syllables.per.100.wrds) + geom_point(aes(fill=person), shape=21, size=3) + facet_grid(person~time))

LW1 <- with(rajSPLIT, linsear_write(dialogue, list(person, act)))
plot(scores(LW1))
plot(counts(LW1))
`bag_o_words`  

**Description**  
`bag_o_words` - Reduces a text column to a bag of words.  
`unbag` - Wrapper for `paste(collapse=" ")` to glue words back into strings.  
`breaker` - Reduces a text column to a bag of words and `qdap` recognized end marks.  
`word_split` - Reduces a text column to a list of vectors of bag of words and `qdap` recognized end marks (i.e., ".", "!", "?", "#", "-").

**Usage**  
`bag_o_words(text.var, apostrophe.remove = FALSE, ...)`  
`unbag(text.var, na.rm = TRUE)`  
`breaker(text.var)`  
`word_split(text.var)`

**Arguments**  
- `text.var` - The text variable.  
- `apostrophe.remove` - logical. If TRUE removes apostrophe’s from the output.  
- `na.rm` - logical. If TRUE NAs are removed before pasting.  
- `...` - Additional arguments passed to `strip`.

**Value**  
Returns a vector of stripped words.  
`unbag` - Returns a string.  
`breaker` - Returns a vector of striped words and `qdap` recognized end marks (i.e., ".", "!", "?", "#", "-").
Examples

```r
## Not run:
bag_o_words("I'm going home!")
bag_o_words("I'm going home!", apostrophe.remove = TRUE)
unbag(bag_o_words("I'm going home!"))

bag_o_words(DATA$state)
by(DATA$state, DATA$person, bag_o_words)
lapply(DATA$state, bag_o_words)

breaker(DATA$state)
by(DATA$state, DATA$person, breaker)
lapply(DATA$state, breaker)
unbag(breaker(DATA$state))

word_split(c(NA, DATA$state))
unbag(word_split(c(NA, DATA$state)))
```

## End(Not run)

---

**beg2char**

*Grab Begin/End of String to Character*

**Description**

beg2char - Grab from beginning of string to a character(s).

char2end - Grab from character(s) to end of string.

**Usage**

```r
beg2char(text.var, char = " ", noc = 1, include = FALSE)
char2end(text.var, char = " ", noc = 1, include = FALSE)
```

**Arguments**

- `text.var`, A character string
- `char` The character from which to grab until/from.
- `noc` Number of times the character appears before the grab.
- `include` logical. If TRUE includes the character in the grab.

**Value**

returns a vector of text with char on/forward removed.

**Author(s)**

Josh O'Brien, Justin Haynes and Tyler Rinker <tyler.rinker@gmail.com>.
References

http://stackoverflow.com/q/15909626/1000343

Examples

## Not run:
x <- c("a_b_c_d", "1_2_3_4", "<_?-_-:"")
beg2char(x, "_")
beg2char(x, "_", 2)
beg2char(x, "_", 3)
beg2char(x, "_", 4)
beg2char(x, "_", 3, include=TRUE)

char2end(x, "_")
char2end(x, "_", 2)
char2end(x, "_", 3)
char2end(x, "_", 4)
char2end(x, "_", 3, include=TRUE)

x2 <- gsub("_", " ", x)
char2end(x2, "_", 2)
beg2char(x2, "_", 2)

x3 <- gsub("_", "\^", x)
char2end(x3, "\^", 2)
beg2char(x3, "\^", 2)

## End(Not run)

---

**blank2NA**  
*Replace Blanks in a dataframe*

**Description**

Replaces blank (empty) cells in a dataframe. Generally, for internal use.

**Usage**

`blank2NA(dataframe, missing = NA)`

**Arguments**

- `dataframe`: A dataframe with blank (empty) cells.
- `missing`: Value to replace empty cells with.

**Value**

Returns a data frame with blank spaces replaced.
**bracketX**

**Bracket Parsing**

**Description**

`bracketX` - Apply bracket removal to character vectors.

`bracketXtract` - Apply bracket extraction to character vectors.

`genX` - Apply general chunk removal to character vectors. A generalized version of `bracketX`.

`genXtract` - Apply general chunk extraction to character vectors. A generalized version of `bracketXtract`.

**Usage**

```r
bracketX(text.var, bracket = "all", missing = NULL, names = FALSE,
         fix.space = TRUE, scrub = fix.space)
```

```r
bracketXtract(text.var, bracket = "all", with = FALSE, merge = TRUE)
```

```r
genX(text.var, left, right, missing = NULL, names = FALSE,
     fix.space = TRUE, scrub = TRUE)
```

```r
genXtract(text.var, left, right, with = FALSE, merge = TRUE)
```

**Arguments**

- `text.var` - The text variable
- `bracket` - The type of bracket (and encased text) to remove. This is one or more of the strings "curly", "square", "round", "angle" and "all". These strings correspond to: {, [, (, < or all four types.
- `missing` - Value to assign to empty cells.
- `names` - logical. If TRUE the sentences are given as the names of the counts.
fix.space  logical. If TRUE extra spaces left behind from an extraction will be eliminated. Additionally, non-space (e.g., "text(no space between text and parenthesis)") is replaced with a single space (e.g., "text (space between text and parenthesis)").

scrub  logical. If TRUE scrubber will clean the text.

merge  logical. If TRUE returns the brackets and the bracketed text.

left  A vector of character or numeric symbols as the left edge to extract.

right  A vector of character or numeric symbols as the right edge to extract.

Value

bracketX - returns a vector of text with brackets removed.

bracketxtract - returns a list of vectors of bracketed text.

genxtract - returns a vector of text with chunks removed.

genX - returns a list of vectors of removed text.

Author(s)

Martin Morgan and Tyler Rinker <tyler.rinker@gmail.com>.

References

http://stackoverflow.com/q/8621066/1000343

See Also

regex

Examples

## Not run:
examp <- structure(list(person = structure(c(1L, 2L, 1L, 3L),
   .Label = c("bob", "greg", "sue"), class = "factor"), text =
   c("I love chicken [unintelligible]!",
   "Me too! (laughter) It's so good. [interrupting]",
   "Yep it's awesome (reading).", "Agreed. (is so much fun)")
), .Names =
c("person", "text"), row.names = c(NA, -4L), class = "data.frame")

examp
bracketX(examp$text, "square")
bracketX(examp$text, "curly")
bracketX(examp$text, c("square", "round"))
bracketX(examp$text)

bracketxtract(examp$text, "square")
bracketXtract(examp$text, "curly")
bracketXtract(examp$text, c("square", "round"))
bracketXtract(examp$text, c("square", "round"), merge = FALSE)
bracketXtract(examp$text)
bracketXtract(examp$text, with = TRUE)
paste2(bracketXtract(examp$text, "curly"), " ")

x <- c("Where is the big dog?",
      "I think he's @running@b with /little cat." )
genXtract(x, c("/", "@a"), c("#", "@b"))

data$state #notice number 1 and 10
genXtract(data$state, c("is", "we"), c("too", "on"))

## End(Not run)

---

**build_qdap_vignette**  
*Replace Temporary Introduction to qdap Vignette*

**Description**

Replaces the temporary (place holder) *Introduction to qdap Vignette* with the actual vignette.

**Usage**

```r
build_qdap_vignette(download.html = FALSE)
```

**Arguments**

- `download.html` logical. If TRUE the file will be downloaded from: [http://trinker.github.io/qdap/vignettes/qdap_vignette.html](http://trinker.github.io/qdap/vignettes/qdap_vignette.html). This

**Value**

Places the (1) HTML, (2) source, & (3) R code for the *Introduction to qdap Vignette* in the user’s ‘R-VERSION/library/qdap/doc’.

**Note**

The **knitr** built HTML approach above takes about 4 minutes. The user may choose the faster approach (< 30 seconds) that downloads the HTML file directly from the Internet (this is for the latest CRAN release of **qdap**). This choice is controlled via the `download.html` argument. The function will ask for the user’s permission before writing the documents. Once the user has run this function `browseVignettes(package = 'qdap')` will allow access to the new vignette files.
capitalizer  

*Capitalize Select Words*

Description

A helper function for `word_list` that allows the user to supply vectors of words to be capitalized.

Usage

```r
capitalizer(text, caps.list = NULL, I.list = TRUE, apostrophe.remove = FALSE)
```

Arguments

- **text**: A vector of words (generally from `bag_o_words` or `breaker`).
- **caps.list**: A list of words to capitalize.
- **I.list**: logical. If `TRUE` capitalizes I words and contractions.
- **apostrophe.remove**: logical, asking if apostrophes have been removed. If `TRUE` will try to insert apostrophe's back into words appropriately.

Value

Returns a vector of capitalized words based on supplied capitalization arguments.

Note

Not intended for general use. Acts as a helper function to several `qdap` functions.

Examples

```r
## Not run:
capitalizer(bag_o_words("i like it but i'm not certain"), "like")
capitalizer(bag_o_words("i like it but i'm not certain"), "like", FALSE)
## End(Not run)
```
Description

check_spelling - Check the spelling for an vector of strings. The function use the following technique:

- Separate the words from a string into a bag of words.
- Look those words up in a dictionary to find words not recognized/found (considered possibly misspelled).
- These misses (possible misspellings) will be what is looked up for suggested replacements.
- Optionally, reduce dictionary by assuming the first letter of the misspelled word is correct (dictionary for this letter only).
- Reduce dictionary by eliminating words outside of the range of number of characters of the misspelled word.
- Use stringdist to find string distances between possible replacements and the misspelled term.
- Select $n$ ($n_suggests$) terms from dictionary that are closest to the misspelled term.

which_misspelled - Check the spelling for a string.

correct - Access the spell corrector function from a "check_spelling_interactive" object for subsequent text character vector spelling corrections.

Usage

check_spelling(text.var, range = 2, assume.first.correct = TRUE, method = "jw", dictionary = qdapDictionaries::GradyAugmented, parallel = TRUE, cores = parallel::detectCores()/2, n.suggests = 8)

which_misspelled(x, suggest = FALSE, range = 2, assume.first.correct = TRUE, dictionary = qdapDictionaries::GradyAugmented, method = "jw", nchar.dictionary = nchar(dictionary), first.char.dictionary = substring(dictionary, 1, 1), n.suggests = 8)

correct(x, ...)

correct(x, ...)
**Arguments**

- **text.var**  
The text variable.

- **range**  
An integer of length 1 to use as a range for number of characters, beyond the number of characters of a word not found in the dictionary, to initially limit dictionary size and thus time to find a suggested replacement term. This may be expanded if no suitable suggestion is returned.

- **assume.first.correct**  
logical. If TRUE it is assumed that the first letter of the misspelled word is correct. This reduces the dictionary size, thus speeding up computation.

- **method**  
Method for distance calculation. The default is "jaccard". It is assumed that smaller measures indicate closer distance. Measures that do not adhere to this assumption will result in incorrect output (see stringdist for details).

- **dictionary**  
A character vector of terms to search for. To reduce overhead it is expected that this dictionary is lower case, unique terms.

- **parallel**  
logical. If TRUE attempts to run the function on multiple cores. Note that this may not mean a speed boost if you have one core or if the data set is smaller as the cluster takes time to create.

- **cores**  
The number of cores to use if parallel = TRUE. Default is half the number of available cores.

- **n.suggests**  
The number of terms to suggest. In the case of a tie (multiple terms have the same distance from misspelled word) all will be provided. Dictionary reduction may result in less than n.suggests suggested terms.

- **x**  
If which.misspelled - A character string. If correct - An object from check_spelling_interactive.

- **suggest**  
logical. If TRUE returns a data.frame with possible suggestions for misspelled words (words not found in the dictionary).

- **nchar.dictionary**  
A vector that corresponds in length and content to dictionary with elements that are the precalculated number of characters for each word in the dictionary.

- **first.char.dictionary**  
A vector that corresponds in length and content to dictionary with elements that are the pre-allotted first characters of each word in the dictionary.

- **click**  
logical. If TRUE the interface is a point and click GUI. If FALSE the interface is command line driven.

- **...**  
ignored

**Value**

- **check_spelling** - Returns a data.frame with row (row number), not.found word.no (number of misspelled word), not.found (a word not found in the dictionary), suggestion (the most likely replacement for the word), and more.suggestions (A list of vectors of up to 10 most likely replacements).

- **which.misspelled** - Returns either a named vector (names are the word number) of possible misspelled words (ifsuggestions = FALSE) or a data.frame with word.no (number of misspelled
word), not found (a word not found in the dictionary), suggestion (the most likely replacement for the word), and more suggestions (A list of vectors of up to 10 most likely replacements).

check_spelling_interactive - Returns a character vector with the corrected text, the replacement list (via an attribute to the character vector), and a function to correct the same spelling errors in subsequent text character vectors.

correct - Returns a function for correcting spelling errors.

Note

A possible misspelled word is defined as not found in the dictionary.

check_spelling_interactive - The user may go back (undo) by pressing "TYPE MY OWN" entering either "1" (not) or "0" (similar to a phone system). The second choice in the "SELECT REPLACEMENT:" will be the original word and is prefixed with "IGNORE:". Press this to keep the original word.

References

http://stackoverflow.com/a/24454727/1000343

See Also

stringdist

Examples

```r
## Not run
x <- "Robots are evil creatures and deserve extermination."
which_misspelled(x, suggest=FALSE)
which_misspelled(x, suggest=TRUE)

check_spelling(DATASET$state)

## browseURL("http://stackoverflow.com/a/24454727/1000343")
terms <- c("accounts", "account", "accounting", "accounting", "acount", "acounts", "accountt")

set.seed(10)
(fake_text <- unlist(lapply(terms, function(x) {
  unbag(sample(c(x, sample(DICTIONARY[[1]], sample(1:5, 1)))),
}))
)

check_spelling(fake_text)

#-------------------------------------
# INTERACTIVE SPELL CHECKING #
#-------------------------------------

## No misspellings found
check_spelling_interactive(DATASET$state)

## character method approach (minimal example)
dat <- DATASET$state; dat[1] <- "I likedd the cokie icekream"
```
(o <- check_spelling_interactive(dat))
preprocessed(o)
fixit <- attributes(o)$correct
fixit(dat)

## character method approach (larger example)
m <- check_spelling_interactive(mraja1spl$dialogue[1:75])
preprocessed(m)
fixit <- attributes(m)$correct
fixit(mraja1spl$dialogue[1:75])

## check_spelling method approach
out <- check_spelling(mraja1spl$dialogue[1:75])
x <- check_spelling_interactive(out))
preprocessed(x)
correct(x)(mraja1spl$dialogue[1:75])
y <- check_spelling_interactive(out, click=FALSE))
preprocessed(y)

## Examine Methods (?stringdist::stringdist)
strings <- c(
  "Robots are evl creatres and deser extrimanitation kream.",
  "I gots me a biggest measrue, tommorrow"
)
meths <- c("osa", "lv", "dl", "hamming", "lcs", "qgram", "cosine", "jaccard", "jw")
stats::setNames(lapply(meths, function(x) check_spelling(strings, method=x)), meths)

## End(Not run)

---

**check_spelling_interactive.character**

**Check Spelling**

**Description**

View character check_spelling_interactive.

**Usage**

## S3 method for class 'character'
check_spelling_interactive(text.var, range = 2,
assume.first.correct = TRUE, click = TRUE, method = "jw",
dictionary = qdapDictionaries::GradyAugmented, parallel = TRUE,
cores = parallel::detectCores()/2, n.suggests = 8, ...)

Arguments

- **text.var**: A character object, specifically a text vector of character strings.
- **range**: An integer of length 1 to use as a range for number of characters, beyond the number of characters of a word not found in the dictionary, to initially limit dictionary size and thus time to find a suggested replacement term. This may be expanded if no suitable suggestion is returned.
- **assume.first.correct**: logical. If TRUE it is assumed that the first letter of the misspelled word is correct. This reduces the dictionary size, thus speeding up computation.
- **click**: logical. If TRUE the interface is a point and click GUI. If FALSE the interface is command line driven.
- **method**: Method for distance calculation. The default is "jaccard". It is assumed that smaller measures indicate closer distance. Measures that do not adhere to this assumption will result in incorrect output (see stringdist for details).
- **dictionary**: A character vector of terms to search for. To reduce overhead it is expected that this dictionary is lower case, unique terms.
- **parallel**: logical. If TRUE attempts to run the function on multiple cores. Note that this may not mean a speed boost if you have one core or if the data set is smaller as the cluster takes time to create.
- **cores**: The number of cores to use if parallel = TRUE. Default is half the number of available cores.
- **n.suggests**: The number of terms to suggest. In the case of a tie (multiple terms have the same distance from misspelled word) all will be provided. Dictionary reduction may result in less than n.suggests suggested terms.
- **...**: ignored

Details

character Method for check_spelling_interactive

check_spelling_interactive.check_spelling

Check Spelling

Description

View check_spelling check_spelling_interactive.

Usage

```r
## S3 method for class 'check_spelling'
check_spelling_interactive(text.var, range = 2,
assume.first.correct = TRUE, click = TRUE, method = "jw",
dictionary = qdapDictionaries::GradyAugmented, parallel = TRUE,
cores = parallel::detectCores()/2, n.suggests = 8, ...)
```
check_spelling_interactive.factor

Arguments

text.var  A check_spelling object.
range  An integer of length 1 to use as a range for number of characters, beyond the
number of characters of a word not found in the dictionary, to initially limit
dictionary size and thus time to find a suggested replacement term. This may
be expanded if no suitable suggestion is returned.
assume.first.correct  logical. If TRUE it is assumed that the first letter of the misspelled word is correct.
This reduces the dictionary size, thus speeding up computation.
click  logical. If TRUE the interface is a point and click GUI. If FALSE the interface is
command line driven.
method  Method for distance calculation. The default is "jaccard". It is assumed that
smaller measures indicate closer distance. Measures that do not adhere to this
assumption will result in incorrect output (see stringdist for details).
dictionary  A character vector of terms to search for. To reduce overhead it is expected that
this dictionary is lower case, unique terms.
parallel  logical. If TRUE attempts to run the function on multiple cores. Note that this
may not mean a speed boost if you have one core or if the data set is smaller as
the cluster takes time to create.
cores  The number of cores to use if parallel = TRUE. Default is half the number of
available cores.
n.suggests  The number of terms to suggest. In the case of a tie (multiple terms have the
same distance from misspelled word) all will be provided. Dictionary reduction
may result in less than n.suggests suggested terms.

Details

check_spelling Method for check_spelling_interactive

Description

View factor check_spelling_interactive.

Usage

## S3 method for class 'factor'
check_spelling_interactive(text.var, range = 2,
assume.first.correct = TRUE, click = TRUE, method = "jw",
dictionary = qdapDictionaries::GradyAugmented, parallel = TRUE,
cores = parallel::detectCores()/2, n.suggests = 8, ...)
Arguments

text.var A factor object, specifically a text vector of factor strings. Note that this method is provided for factors for convenience, ideally the user should supply a character vector rather than factor.

range An integer of length 1 to use as a range for number of characters, beyond the number of characters of a word not found in the dictionary, to initially limit dictionary size and thus time to find a suggested replacement term. This may be expanded if no suitable suggestion is returned.

assume.first.correct logical. If TRUE it is assumed that the first letter of the misspelled word is correct. This reduces the dictionary size, thus speeding up computation.

click logical. If TRUE the interface is a point and click GUI. If FALSE the interface is command line driven.

method Method for distance calculation. The default is "jaccard". It is assumed that smaller measures indicate closer distance. Measures that do not adhere to this assumption will result in incorrect output (see stringdist for details).

dictionary A character vector of terms to search for. To reduce overhead it is expected that this dictionary is lower case, unique terms.

parallel logical. If TRUE attempts to run the function on multiple cores. Note that this may not mean a speed boost if you have one core or if the data set is smaller as the cluster takes time to create.

cores The number of cores to use if parallel = TRUE. Default is half the number of available cores.

n.suggests The number of terms to suggest. In the case of a tie (multiple terms have the same distance from misspelled word) all will be provided. Dictionary reduction may result in less than n.suggests suggested terms.

... ignored

Details

factor Method for check.spelling_interactive

Description

Uncleaned text may result in errors, warnings, and incorrect results in subsequent analysis. check_text checks text for potential problems and suggests possible fixes. Potential text anomalies that are detected include: factors, missing ending punctuation, empty cells, double punctuation, non-space after comma, no alphabetic characters, non-ascii, missing value, and potentially misspelled words.

Usage

check_text(text.var, file = NULL)
Arguments

- **text.var** The text variable.
- **file** A connection, or a character string naming the file to print to. If `NULL` prints to the console. Note that this is assigned as an attribute and passed to `print`.

Value

Returns a list with the following potential text faults reports:

- **non_character** Text that is non-character.
- **missing_ending_punctuation** Text with no endmark at the end of the string.
- **empty** Text that contains an empty element (i.e., ".").
- **double_punctuation** Text that contains two qdap punctuation marks in the same string.
- **non_space_after_comma** Text that contains commas with no space after them.
- **no_alpha** Text that contains string elements with no alphabetic characters.
- **non_ascii** Text that contains non-ASCII characters.
- **missing_value** Text that contains missing values (i.e., `NA`).
- **containing_escaped** Text that contains escaped (see `?Quotes`).
- **containing_digits** Text that contains digits.
- **indicating_incomplete** Text that contains endmarks that are indicative of incomplete/trailing sentences (e.g., `. .`).
- **potentially_misspelled** Text that contains potentially misspelled words.

Note

The output is a list but prints as a pretty formatted output with potential problem elements, the accompanying text, and possible suggestions to fix the text.

See Also

- `check_spelling_interactive`

Examples

```r
## Not run:

x <- c("i like", "i want. that them .", "I am ! that|", ",", "NA, "they.were there", ".", ",", ",?", ",", "I like goud eggs!", "i 4like...", "\\tgreat", "She said \"yes\"")

check_text(x)

print(check_text(x), include.text=FALSE)

y <- c("A valid sentence.", "yet another!")

check_text(y)

## End(Not run)
```
**chunker**

---

### Break Text Into Ordered Word Chunks

**Description**

Some visualizations and algorithms require text to be broken into chunks of ordered words. `chunker` breaks text, optionally by grouping variables, into equal chunks. The chunk size can be specified by giving number of words to be in each chunk or the number of chunks.

**Usage**

```r
chunker(text.var, grouping.var = NULL, n.words, n.chunks, as.string = TRUE, rm.unequal = FALSE)
```

**Arguments**

- **text.var**: The text variable.
- **grouping.var**: The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- **n.words**: An integer specifying the number of words in each chunk (must specify `n.chunks` or `n.words`).
- **n.chunks**: An integer specifying the number of chunks (must specify `n.chunks` or `n.words`).
- **as.string**: logical. If TRUE the chunks are returned as a single string. If FALSE the chunks are returned as a vector of single words.
- **rm.unequal**: logical. If TRUE final chunks that are unequal in length to the other chunks are removed.

**Value**

Returns a list of text chunks.

**Examples**

```r
with(DATA, chunker(state, n.chunks = 10))
with(DATA, chunker(state, n.words = 10))
with(DATA, chunker(state, n.chunks = 10, as.string=FALSE))
with(DATA, chunker(state, n.chunks = 10, rm.unequal=TRUE))
with(DATA, chunker(state, person, n.chunks = 10))
with(DATA, chunker(state, list(sex, adult), n.words = 10))
with(DATA, chunker(state, person, n.words = 10, rm.unequal=TRUE))
```

---

### Bigger data

```r
with(hamlet, chunker(dialogue, person, n.chunks = 10))
with(hamlet, chunker(dialogue, person, n.words = 300))
```

---

### Not run:

- ## with polarity hedonmetrics
dat <- with(pres_debates2012[pres_debates2012$person %in% qcv(Obama, Romney), ],
  chunker(dialogue, list(person, time), n.words = 300))

dat2 <- colsplit2df(list2df(dat, "dialogue", "person&time")[, 2:1])

dat3 <- split(dat2[,-2], dat2$time)
ltruncdf(dat3, 10, 50)
poldat <- lapply(dat3, function(x) with(x, polarity(dialogue, person, constrain = TRUE)))

m <- lapply(poldat, function(x) plot(cumulative(x)))
m <- Map(function(w, x, y, z) {
  w + ggtitle(x) + xlab(y) + ylab(z)
}, m,
paste("Debate", 1:3),
list(NULL, NULL, "Duration (300 Word Segment)",
list(NULL, "Cumulative Average Polarity", NULL))
)

library(gridExtra)
do.call(grid.arrange, m)

# By person
# By person
poldat2 <- Map(function(x, x2){
  scores <- with(counts(x), split(polarity, person))
  setNames(lapply(scores, function(y) {
    y <- list(cumulative_average_polarity = y)
    attributes(y)["constrained"] <- TRUE
    qdap::plot.cumulative_polarity(y) + xlab(NULL) + ylab(x2)
  }), names(scores))
}, poldat, paste("Debate", 1:3))

poldat2[1] <- lapply(poldat2, function(x) {
  x[[2]] <- x[[2]] + ylab(NULL)
  x
})
poldat2[[1]] <- Map(function(x, y) {
  x + ggtitle(y)
},
  poldat2[[1]], qcv(Obama, Romney)
)

library(gridExtra)
do.call(grid.arrange, unlist(poldat2, recursive=FALSE))

# End(Not run)
**clean**  
*Remove Escaped Characters*

**Description**
Preprocess data to remove escaped characters

**Usage**
clean(text.var)

**Arguments**
text.var The text variable

**Value**
Returns a vector of character strings with escaped characters removed.

**Examples**
```r
## Not run:
x <- "I go \r
   to the \tnext line"
x
clean(x)

## End(Not run)
```

---

**cm_2long**  
*A Generic to Long Function*

**Description**
A wrapper for cm_df2long, cm_range2long, and cm_time2long that automatically detects the objects being read and outputs the correct form and class.

**Usage**
cm_2long(..., v.name = "variable", list.var = TRUE, debug = TRUE)
Arguments

- `v.name` An optional name for the column created for the list.var argument.
- `list.var` logical. If TRUE creates a column for the data frame created by each time.list passed to cm_t2l.
- `debug` logical. If TRUE debugging mode is on. `cm_time2long` will return possible errors in time span inputs.
- `...` list object(s) in the form generated by `cm_df.temp`, `cm_range.temp`, or `cm_time.temp`.

Value

Returns a long data.frame of the correct `cm_XXX` classes.

See Also

`cm_df2long`, `cm_range2long`, `cm_time2long`

Examples

```r
## Not run:
## cm_range2long use:
foo <- list(
  person_greg = qcv(terms='7:11, 20:24, 30:33, 49:56'),
  person_researcher = qcv(terms='42:48'),
  person_sally = qcv(terms='25:29, 37:41'),
  person_sam = qcv(terms='1:6, 16:19, 34:36'),
  person_teacher = qcv(terms='12:15'),
  adult_0 = qcv(terms='1:11, 16:41, 49:56'),
  adult_1 = qcv(terms='12:15, 42:48'),
  AA = qcv(terms="1"),
  BB = qcv(terms="1:2, 3:10, 19"),
  CC = qcv(terms="1:9, 100:150")
)

foo2 <- list(
  person_greg = qcv(terms='7:11, 20:24, 30:33, 49:56'),
  person_researcher = qcv(terms='42:48'),
  person_sally = qcv(terms='25:29, 37:41'),
  person_sam = qcv(terms='1:6, 16:19, 34:36'),
  person_teacher = qcv(terms='12:15'),
  adult_0 = qcv(terms='1:11, 16:41, 49:56'),
  adult_1 = qcv(terms='12:15, 42:48'),
  AA = qcv(terms="40"),
  BB = qcv(terms="50:90"),
  CC = qcv(terms="60:90, 100:120, 150"),
  DD = qcv(terms="")
)

cm_2long(foo, foo2, v.name = "time")
## cm_time2long use:
```
Blank Code Transformation

Description

Transform codes with any binary operator combination.

Usage

cm_code.blanc(x2long.obj, combine.code.list, rm.var = NULL, overlap = TRUE)

Arguments

- **x2long.obj**: An object from `cm_range2long`, `cm_time2long` or `cm_df2long`.
- **combine.code.list**: A list of named character vectors of at least two code column names to combine.
- **rm.var**: Name of the repeated measures column.
- **overlap**: logical, integer or character of binary operator + integer. If TRUE finds the overlap. If FALSE finds anywhere any of the codes occur. If integer finds that exact combination of overlaps. If character must be a logical vector c(>, <, <=, >=, ! =) followed by an integer and wrapped with quotes.

Value

Returns a dataframe with transformed occurrences of supplied overlapping codes added.
Note
For most jobs `cm_code.transform` will work. This adds a bit of flexibility in exclusion and partial matching. The code column must be named "code" and your start and end columns must be named "start" and "end".

See Also
`cm_range2long`, `cm_time2long`, `cm_df2long`, `cm_code.overlap`, `cm_code.combine`, `cm_code.exclude`, `cm_code.transform`

Examples
```r
## Not run:
foo <- list(
  AA = qcv(terms="1:10"),
  BB = qcv(terms="1:2, 3:10, 19"),
  CC = qcv(terms="1:3, 5:6")
)

foo2 <- list(
  AA = qcv(terms="4:8"),
  BB = qcv(terms="1:4, 10:12"),
  CC = qcv(terms="1, 11, 15:20"),
  DD = qcv(terms=""
)
)

## Single occurrence version
(x <- cm_range2long(foo))

cm_code.blank(x, combine.code.list = list(ABC=qcv(AA, BB, CC)),
  overlap = "!=")

## Repeated measures version
(z <- cm_range2long(foo, foo2, v.name="time"))

cm_code.blank(z, combine.code.list = list(ABC=qcv(AA, BB, CC)),
  rm.var = "time", overlap = "!=")

cm_code.blank(z, combine.code.list = list(AB=qcv(AA, BB)),
  rm.var = "time", overlap = TRUE)

cm_code.blank(z, combine.code.list = list(AB=qcv(AA, BB)),
  rm.var = "time", overlap = FALSE)

cm_code.blank(z, combine.code.list = list(AB=qcv(AA, BB)),
  rm.var = "time", overlap = ">"
)

cm_code.blank(z, combine.code.list = list(AB=qcv(AA, BB)),
  rm.var = "time", overlap = "=="
)

## Notice `overlap = "==2"` above is identical to `cm_code.overlap`

cm_code.overlap(z, overlap.code.list = list(AB=qcv(AA, BB)),
```
cm_code.combine

Combine Codes

Description

Combine all occurrences of codes into a new code.

Usage

cm_code.combine(x2long.obj, combine.code.list, rm.var = NULL)

Arguments

x2long.obj An object from cm_range2long, cm_time2long or cm_df2long.
combine.code.list A list of named character vectors of at least two code column names to combine
rm.var Name of the repeated measures column.
cm_code.combine

Value

Returns a dataframe with combined occurrences of supplied overlapping codes added.

Note

The code column must be named "code" and your start and end columns must be named "start" and "end".

See Also

cm_range2long, cm_time2long, cm_df2long, cm_code.blank, cm_code.exclude, cm_code.overlap, cm_code.transform

Examples

## Not run:
foo <- list(
  AA = qcv(terms="1:10"),
  BB = qcv(terms="1:2, 3:10, 19"),
  CC = qcv(terms="1:3, 5:6")
)

foo2 <- list(
  AA = qcv(terms="4:8"),
  BB = qcv(terms="1:4, 10:12"),
  CC = qcv(terms="1, 11, 15:20"),
  DD = qcv(terms="")
)

(x <- cm_range2long(foo))
(z <- cm_range2long(foo, foo2, v.name="time"))
cm_code.combine(x, list(AB=qcv(AA, BB)))
cm_code.combine(x, list(ALL=qcv(AA, BB, CC)))
combines <- list(AB=qcv(AA, BB), ABC=qcv(AA, BB, CC))
cm_code.combine(z, combines, rm.var = "time")

#WITH cm_time2long
x <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00"),
  B = qcv(terms = "2.40, 3.01:3.02, 5.01, 6.02:7.00, 9.00, 1.12.00:1.19.01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 17.01")
)

y <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00"),
  B = qcv(terms = "2.40, 3.01:3.02, 5.01, 6.02:7.00, 9.00, 1.12.00:1.19.01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 17.01")
)
```r
dat <- cm_time2long(x, y)
head(dat, 12)
cm_code.combine(dat, list(P=qcv(A, B), Q=qcv(B, C), R=qcv(A, B, C)), "variable")

## End(Not run)
```

## cm_code.exclude

### Exclude Codes

**Description**

Find the occurrences of n codes excluding the nth code. For example, you have times/words coded for a teacher and you also have times/words coded for happiness. You can find all the happiness times excluding the teacher times or vice versa.

**Usage**

```r
cm_code.exclude(x2long.obj, exclude.code.list, rm.var = NULL)
```

**Arguments**

- `x2long.obj` An object from `cm_range2long`, `cm_time2long`, or `cm_df2long`.
- `exclude.code.list` A list of named character vectors of at least two code column names to compare and exclude. The last column name is the one that will be excluded.
- `rm.var` Name of the repeated measures column.

**Value**

Returns a dataframe with n codes excluding the nth code.

**Note**

The code column must be named "code" and your start and end columns must be named "start" and "end".

**See Also**

- `cm_range2long`, `cm_time2long`, `cm_df2long`, `cm_code.blank`, `cm_code.combine`, `cm_code.overlap`, `cm_code.transform`
cm_code.overlap

Find Co-occurrence Between Codes

Examples

```r
## Not run:
foo <- list(
  AA = qcv(terms="1:10"),
  BB = qcv(terms="1:2, 3:10, 19"),
  CC = qcv(terms="1:3, 5:6")
)

foo2 <- list(
  AA = qcv(terms="4:8"),
  BB = qcv(terms="1:4, 10:12"),
  CC = qcv(terms="1, 11, 15:20"),
  DD = qcv(terms="")
)

(x <- cm_range2long(foo))
(z <- cm_range2long(foo, foo2, v.name="time"))
cm_code.exclude(x, list(ABnoC=qcv(AA, BB, CC)))

#WITH cm_time2long
x <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00", 1.12.00:1.19.01"),
  B = qcv(terms = "2.40, 3.01:3.02, 5.01, 6.02:7.00, 9.00, 1.12.00:1.19.01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 17.01")
)

y <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00"),
  B = qcv(terms = "2.40, 3.01:3.02, 5.01, 6.02:7.00, 9.00, 1.12.00:1.19.01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 17.01")
)

dat <- cm_time2long(x, y)
head(dat, 10)
cm_code.exclude(dat, list(P=qcv(A, B), Q=qcv(B, C), R=qcv(A, B, C)),
  rm.var = "variable")

## End(Not run)
```
Description

Combine co-occurrences of codes into a new code.

Usage

```r
cm_code.overlap(x2long.obj, overlap.code.list, rm.var = NULL)
```

Arguments

- `x2long.obj`: An object from `cm_range2long, cm_time2long` or `cm_df2long`.
- `overlap.code.list`: A list of named character vectors of at least two code column names to aggregate co-occurrences.
- `rm.var`: Name of the repeated measures column.

Value

Returns a dataframe with co-occurrences of supplied overlapping codes added.

Note

The code column must be named code and your start and end columns must be named "start" and "end".

See Also

- `cm_range2long, cm_time2long, cm_df2long, cm_code.combine, cm_code.transform`

Examples

```r
## Not run:
foo <- list(
  AA = qcv(terms="1:10"),
  BB = qcv(terms="1:2, 3:10, 19"),
  CC = qcv(terms="1:3, 5:6")
)

foo2 <- list(
  AA = qcv(terms="4:8"),
  BB = qcv(terms="1:4, 10:12"),
  CC = qcv(terms="1, 11, 15:20"),
  DD = qcv(terms="")
)

(x <- cm_range2long(foo))
(z <- cm_range2long(foo, foo2, v.name="time"))
cm_code.overlap(x, list(AB=qcv(AA, BB)))
cm_code.overlap(x, list(ALL=qcv(AA, BB, CC)))
combines <- list(AB=qcv(AA, BB), ABC=qcv(AA, BB, CC))
```
(a <- cm_code.overlap(z, combines, "time"))
plot(a)

#WITH cm_time2long
x <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00"),
  B = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00,
                  1.12.00:1.19.01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 17.01")
)

y <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00"),
  B = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00,
                  1.12.00:1.19.01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 17.01")
)

dat <- cm_time2long(x, y)
head(dat, 10)
out <- cm_code.overlap(dat, list(P=qcv(A, B), Q=qcv(B, C), R=qcv(A, B, C)),
                       rm.var="variable")
head(out, 10)

## End(Not run)

---

**cm_code.transform**

**Transform Codes**

**Description**

Transform co-occurrences and/or combinations of codes into a new code(s).

**Usage**

`cm_code.transform(x2long.obj, overlap.code.list = NULL, combine.code.list = NULL, exclude.code.list = NULL, rm.var = NULL)`

**Arguments**

- `x2long.obj` An object from `cm_range2long, cm_time2long` or `cm_df2long`
- `overlap.code.list` A list of named character vectors of at least two code column names to aggregate co-occurrences.
- `combine.code.list` A list of named character vectors of at least two code column names to combine
exclude.code.list
A list of named character vectors of at least two code column names to compare and exclude. The last column name is the one that will be excluded.

rm.var
Name of the repeated measures column.

Value
Returns a dataframe with overlapping, combined occurrences, and/or exclusion of supplied overlapping codes added.

Note
The code column must be named "code" and your start and end columns must be named "start" and "end".

See Also
cm_range2long, cm_time2long, cm_df2long, cm_code.blank, cm_code.combine, cm_code.exclude, cm_code.overlap

Examples
```r
## Not run:
foo <- list(
  AA = qcv(terms="1:10"),
  BB = qcv(terms="1:2, 3:10, 19"),
  CC = qcv(terms="1:3, 5:6")
)

foo2 <- list(
  AA = qcv(terms="4:8"),
  BB = qcv(terms="1:4, 10:12"),
  CC = qcv(terms="1, 11, 15:20"),
  DD = qcv(terms="")
)

bar1 <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "0.00:3.00, 5.01, 6.02:7.00, 9.00"),
  B = qcv(terms = "2.40, 3.01:3.02, 5.01, 6.02:7.00, 9.00, 1.12.00:1.19.01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 16.25:17.01")
)

(x <- cm_range2long(foo))
(z <- cm_range2long(foo, foo2, v.name="time"))
(dat <- cm_time2long(bar1))

cm_code.transform(x,
  overlap.code.list = list(ABC=qcv(AA, BB, CC)),
  combine.code.list = list(oABC=qcv(AA, BB, CC)),
  exclude.code.list = list(ABnoC=qcv(AA, BB, CC))
```
cm_combine.dummy

Find Co-occurrence Between Dummy Codes

Description

Combine code columns where they co-occur.

Usage

cm_combine.dummy(cm.12d.obj, combine.code, rm.var = "time", overlap = TRUE)

Arguments

- **cm.12d.obj** An object from `cm_long2dummy`.
- **combine.code** A list of named character vectors of at least two code column names to combine
- **rm.var** Name of the repeated measures column. Default is "time".
- **overlap** logical, integer or character of binary operator + integer. If TRUE finds the overlap. If FALSE finds anywhere any of the codes occur. If integer finds that exact combination of overlaps. If character must be a logical vector c(>, <, <=, =>, ==, !=) followed by an integer and wrapped with quotes.

Value

Returns a dataframe with co-occurrences of provided code columns.

See Also

`cm_long2dummy`
Examples

```
## Not run:
foo <- list(
  AA = qcv(terms="1:10"),
  BB = qcv(terms="1:2, 3:10, 19"),
  CC = qcv(terms="1:3, 5:6")
)
foo2 <- list(
  AA = qcv(terms="4:8"),
  BB = qcv(terms="1:4, 10:12"),
  CC = qcv(terms="1, 11, 15:20"),
  DD = qcv(terms="")
)

(x <- cm_range2long(foo))
(D1 <- cm_long2dummy(x))

(z <- cm_range2long(foo, foo2, v.name="time"))
(D2 <- cm_long2dummy(z, "time"))

  cm_combine.dummy(D1, combine.code = list(AB=qcv(AA, BB)))
  cm_combine.dummy(D1, combine.code = list(AB=qcv(AA, BB), overlap="==1")
  cm_combine.dummy(D1, combine.code = list(AB=qcv(AA, BB), overlap="#1")
D1 <- cm_combine.dummy(D1, combine.code = list(AB=qcv(AA, BB), overlap=0)
D1 <- cm_combine.dummy(D1, combine.code = list(ABC=qcv(AAB, BB, CC), overlap=FALSE)

combines <- list(AB=qcv(AA, BB), ABC=qcv(AA, BB, CC))

  cm_combine.dummy(D1, combine.code = combines)
  cm_combine.dummy(D2, combine.code = combines)
```

## End(Not run)

---

**cm_df.fill**

**Range Coding**

**Description**

Allows range coding of words for efficient coding.

**Usage**

```
cm_df.fill(dataframe, ranges, value = 1, text.var = NULL,
            code.vars = NULL, transform = FALSE)
```

**Arguments**

- **dataframe** A dataframe containing a text variable.
- **ranges** A named list of ranges to recode. Names correspond to code names in dataframe.
value  The recode value. Takes a vector of length one or a vector of length equal to the number of code columns.
text.var  The name of the text variable.
code.vars  Optional vector of codes.
transform  logical. If TRUE the words are located across the top of dataframe.

Details
After ranging coding transcripts via (cm_df.temp) or the blank code matrix via (cm_df.transcript), cm_df.fill is used to create a matrix of what codes occurred at what words (a filled code matrix). A list of range codes (word number spans) is fed to cm_df.fill. A single number indicates a single word with that coding scheme whereas the colon is used as a separator that indicates the range of words from x to y are that particular code.

Value
Generates a dummy coded dataframe.

References

See Also
cm_df.temp, cm_df.transcript, cm_df2long

Examples
## Not run:
codes <- qcv(dc, sf, wes, pol, rejk, lk, azx, mmm)
X <- cm_df.temp(DATA, "state", codes)
head(X, 10)

#recommended structure
cds1 <- list(
  dc=c(1:3, 5),
  sf=c(4, 6:9, 11),
  wes=0,
  pol=0,
  rejk=0,
  lk=0,
  azx=1:30,
  mmm=5
)

out1 <- cm_df.fill(X, cds1)
head(out1)

#recommended structure
cm_df.temp

```r
cds2 <- list(
  sf=c(4, 6:9, 11),
  dc=c(1:3, 5),
  azx=1:30,
  mmm=5
)
out2 <- cm_df.fill(X, cds2)
head(out2)

## End(Not run)
```

## Break Transcript Dialogue into Blank Code Matrix

### Description

Breaks transcript dialogue into words while retaining the demographic factors associated with each word. The codes argument provides a matrix of zeros that can serve as a dummy coded matrix of codes per word.

### Usage

```r
cm_df.temp(dataframe, text.var, codes = NULL, file = NULL, transpose = FALSE, strip = FALSE, ...)
```

### Arguments

- **dataframe**: A dataframe containing a text variable.
- **text.var**: The name of the text variable.
- **codes**: Optional list of codes.
- **file**: The name of the file (csv is recommended file type). If NULL no file is written.
- **transpose**: logical. If TRUE transposes the dataframe so that the text is across the top.
- **strip**: logical. If TRUE all punctuation is removed.
- **...**: Other arguments passed to strip.

### Value

Generates a dataframe, and optional csv file, of individual words while maintaining demographic information. If a vector of codes is provided the outcome is a matrix of words used by codes filled with zeros. This dataframe is useful for dummy coded (1=yes code exists; 0-no it does not) representation of data and can be used for visualizations and statistical analysis.

### References

See Also

`cm_range2long, cm_df.transcript, cm_df.fill`

Examples

```r
## Not run:
codes <- qcv(dc, sf, wes, pol, rejk, lk, azx, mmm)
out1 <- cm_df.temp(DATA, "state", codes)
head(out1, 15)
out2 <- cm_df.temp(DATA, "state", codes, transpose = TRUE)
out2[, 1:10]
out3 <- cm_df.temp(raj.act.1, "dialogue", codes)
head(out3, 15)
out4 <- cm_df.temp(raj.act.1, "dialogue", codes, transpose = TRUE)
out4 [, 1:8]

## End(Not run)
```

---

**cm_df.transcript**

*Transcript With Word Number*

Description

Output a transcript with word number/index above for easy input back into qdap after coding.

Usage

```r
cm_df.transcript(text.var, grouping.var, file = NULL, indent = 4,
width = 70, space = 2, ...)
```

Arguments

- `text.var`  The text variable.
- `grouping.var`  The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `file`  A connection, or a character string naming the file to print to (e.g., .doc, .txt).
- `indent`  Number of spaces to indent.
- `width`  Width to output the file (defaults to 70; this is generally a good width and indent for a .docx file).
- `space`  An integer value denoting the vertical spacing between the grouping.var and the numbered text (allow more space for more coding room) in the output of a text file.
- `...`  Other arguments passed to strip.
Value

Returns a transcript by grouping variable with word number above each word. This makes use with `cm_df2long` transfer/usage easier because the researcher has coded on a transcript with the numeric word index already.

Note

It is recommended that the researcher actually codes on the output from this file. The codes can then be transferred to via a list. If a file already exists `cm_df.transcript` will append to that file.

Author(s)

BondedDust (stackoverflow.com), Gavin Simpson and Tyler Rinker <tyler.rinker@gmail.com>

See Also

`cm_df2long`, `cm_df.temp`

Examples

```r
## Not run:
with(DATA, cm_df.transcript(state, person))
with(DATA, cm_df.transcript(state, list(sex, adult)))
# use it with nested variables just to keep track of demographic info
with(DATA, cm_df.transcript(state, list(person, sex, adult)))

# use double tilde "~~" to keep word group as one word
DATA$state <- mgsub("be certain", "be--certain", DATA$state, fixed = TRUE)
with(DATA, cm_df.transcript(state, person))
DATA <- qdap::DATA

## with(mrajar1spl, cm_df.transcript(dialogue, list(person)))
## with(mrajar1spl, cm_df.transcript(dialogue, list(sex, fam.aff, died)))
## with(mrajar1spl, cm_df.transcript(dialogue, list(person), file="foo.doc"))
## library(reports); delete("foo.doc")  #delete the file just created

## End(Not run)
```

---

**cm_df2long**

**Transform Codes to Start-End Durations**

Description

Transforms the range coding structure(s) from `cm_df.temp` (in list format) into a data frame of start and end durations in long format.
Usage

cm_df2long(df.temp.obj, v.name = "variable", list.var = TRUE,
    code.vars = NULL, no.code = NA, add.start.end = TRUE,
    repeat.vars = NULL, rev.code = FALSE)

Arguments

df.temp.obj A character vector of names of object(s) created by cm_df.temp, a list of cm_df.temp
created objects or a data frame created by cm_df.temp.
v.name An optional name for the column created for the list.var argument.
list.var logical. If TRUE creates a column for the data frame created by each time.list.
code.vars A character vector of code variables. If NULL uses all variables from the first
    column after the column named word.num.
no.code The value to assign to no code; default is NA.
add.start.end logical. If TRUE adds a column for start and end times.
repeat.vars A character vector of repeated/stacked variables. If NULL uses all non code.vars
    variables.
rev.code logical. If TRUE reverses the order of code.vars and no.code variables.

Value

Generates a data frame of start and end times for each code.

References

Miles, M. B. & Huberman, A. M. (1994). An expanded sourcebook: Qualitative data analysis. 2nd

See Also

cm_time2long, cm_range2long, cm_df.temp

Examples

## Not run:
codes <- qcv(dc, sf, wes, pol, rejk, lk, azx, mmm)
x1 <- cm_df.temp(DATA, "state", codes)
head(x1)

#empty code matrix
out1 <- cm_df2long(x1, code.vars = codes)
head(out1, 15)

#fill it randomly
x1[7:14] <- lapply(7:14, function(i) sample(0:1, nrow(x1), TRUE))
out2 <- cm_df2long(x1, code.vars = codes)
head(out2, 15)
plot(out2)
cm_distance

## End(Not run)

### cm_distance

#### Distance Matrix Between Codes

**Description**

Generate distance measures to ascertain a mean distance measure between codes.

**Usage**

```r
cm_distance(dataframe, pvals = c(TRUE, FALSE), replications = 1000, parallel = TRUE, extended.output = TRUE, time.var = TRUE, code.var = "code", causal = FALSE, start.var = "start", end.var = "end", cores = detectCores()/2)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataframe</td>
<td>A data frame from the cm_x2long family (cm_range2long; cm_df2long; cm_time2long).</td>
</tr>
<tr>
<td>pvals</td>
<td>A logical vector of length 1 or 2. If element 2 is blank element 1 will be recycled. If the first element is TRUE pvalues will be calculated for the combined (main) output for all repeated measures from simulated resampling of the data. If the second element is TRUE pvalues will be calculated for the individual (extended) repeated measures output from simulated resampling of the data. Default is to calculate pvalues for the main output but not for the extended output. This process involves multiple resampling of the data and is a time consuming process. It may take from a few minutes to days to calculate the pvalues depending on the number of all codes use, number of different codes and number of replications.</td>
</tr>
<tr>
<td>replications</td>
<td>An integer value for the number of replications used in resampling the data if any pvals is TRUE. It is recommended that this value be no lower than 1000. Failure to use enough replications may result in unreliable pvalues.</td>
</tr>
<tr>
<td>parallel</td>
<td>logical. If TRUE runs the cm_distance on multiple cores (if available). This will generally be effective with most data sets, given there are repeated measures, because of the large number of simulations. Default uses 1/2 of the available cores.</td>
</tr>
<tr>
<td>extended.output</td>
<td>logical. If TRUE the information on individual repeated measures is calculated in addition to the aggregated repeated measures results for the main output.</td>
</tr>
<tr>
<td>time.var</td>
<td>An optional variable to split the dataframe by (if you have data that is by various times this must be supplied).</td>
</tr>
<tr>
<td>code.var</td>
<td>The name of the code variable column. Defaults to &quot;codes&quot; as out putted by x2long family.</td>
</tr>
</tbody>
</table>
causal logical. If TRUE measures the distance between x and y given that x must precede y. That is, only those yᵢ that begin after the xᵢ has begun will be considered, as it is assumed that x precedes y. If FALSE x is not assumed to precede y. The closest yᵢ (either its beginning or end) is calculated to xᵢ (either it’s beginning or end).

start.var The name of the start variable column. Defaults to "start" as outputted by x2long family.

end.var The name of the end variable column. Defaults to "end" as outputted by x2long family.

cores An integer value describing the number of cores to use if parallel = TRUE. Default is to use half of the available cores.

Details

Note that row names are the first code and column names are the second comparison code. The values for Code A compared to Code B will not be the same as Code B compared to Code A. This is because, unlike a true distance measure, cm_distance’s matrix is asymmetrical. cm_distance computes the distance by taking each span (start and end) for Code A and comparing it to the nearest start or end for Code B.

Value

An object of the class "cm_distance". This is a list with the following components:

pvals A logical indication of whether pvalues were calculated
replications Integer value of number of replications used
extended.output An optional list of individual repeated measures information
main.output A list of aggregated repeated measures information
adj.alpha An adjusted alpha level (based on α = .05) for the estimated p-values using the upper end of the confidence interval around the p-values

Within the lists of extended.output and list of the main.output are the following items:

mean A distance matrix of average distances between codes
sd A matrix of standard deviations of distances between codes
n A matrix of counts of distances between codes
stan.mean A matrix of standardized values of distances between codes. The closer a value is to zero the closer two codes relate.
pvalue A n optional matrix of simulated pvalues associated with the mean distances

Warning

p-values are estimated and thus subject to error. More replications decreases the error. Use:

\[ p \pm 1.96 \cdot \sqrt{\frac{\alpha(1-\alpha)}{n}} \]

to adjust the confidence in the estimated p-values based on the number of replications.
cm_distance

References

http://stats.stackexchange.com/a/22333/7482

See Also

print.cm_distance

Examples

```r
## Not run:
foo <- list(
  AA = qcv(terms="02:03, 05"),
  BB = qcv(terms="1:2, 3:10"),
  CC = qcv(terms="1:9, 100:150")
)

foo2 <- list(
  AA = qcv(terms="40"),
  BB = qcv(terms="50:90"),
  CC = qcv(terms="60:90, 100:120, 150"),
  DD = qcv(terms="")
)

(dat <- cm_2long(foo, foo2, v.name = "time"))
plot(dat)
(out <- cm_distance(dat, replications=100))
names(out)
names(out$main.output)
out$main.output
out$extended.output
print(out, new.order = c(3, 2, 1))
print(out, new.order = 3:2)
#===============================================

x <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 6.32:7.00, 9.00, 10.00:11.00, 59.56"),
  B = qcv(terms = "3.01:3.02, 5.01, 19.00, 1.12.00:1.19.01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.32:7.00, 9.00, 17.01")
)

(dat <- cm_2long(x))
plot(dat)
(a <- cm_distance(dat, causal=TRUE, replications=100))

## Plotting as a network graph

dataA <- list(
  A = qcv(terms="02:03, 05"),
  B = qcv(terms="1:2, 3:10, 45, 60, 200:206, 250, 289:299, 330"),
  C = qcv(terms="1:9, 47, 62, 100:150, 202, 260, 292:299, 332"),
  D = qcv(terms="10:20, 30, 38:44, 138:145"),
  E = qcv(terms="10:15, 32, 36:43, 132:140"),
  F = qcv(terms="1:2, 3:9, 10:15, 32, 36:43, 45, 60, 132:140, 250, 289:299"),
)


G = qcv(terms="1:2, 3:9, 10:15, 32, 36:43, 45, 60, 132:140, 250, 289:299"),
H = qcv(terms="20, 40, 60, 150, 190, 222, 255, 277"),
I = qcv(terms="20, 40, 60, 150, 190, 222, 255, 277")
)

datB <- list(
    A = qcv(terms="40"),
    B = qcv(terms="50:90, 110, 148, 177, 200:206, 250, 289:299"),
    C = qcv(terms="60:90, 100:120, 150, 201, 244, 292"),
    D = qcv(terms="10:20, 30, 38:44, 138:145"),
    E = qcv(terms="10:15, 32, 36:43, 132:140"),
    F = qcv(terms="10:15, 32, 36:43, 132:140, 148, 177, 200:206, 250, 289:299"),
    I = qcv(terms="20, 40, 60, 150, 190, 222, 255, 277")
)

(datC <- cm_2long(datA, datB, v.name = "time"))
plot(datC)
(out2 <- cm_distance(datC, replications=1250))
plot(out2)
plot(out2, label.cex=2, label.dist=TRUE, digits=5)

## End(Not run)

---

### Description

Convert \texttt{cm\_combine.dummy} Back to Long

#### Usage

\texttt{cm\_dummy2long(cm\_long2dummy\_obj, rm.var = "time")}

#### Arguments

- \texttt{cm\_long2dummy\_obj}
  - An object from \texttt{cm\_combine.dummy}
- \texttt{rm.var}
  - Name of the repeated measures column. Default is "time".

#### Value

Returns a dataframe with co-occurrences of provided code columns.

#### See Also

\texttt{cm\_long2dummy, cm\_combine.dummy}
Examples

```r
## Not run:
foo <- list(
    AA = qcv(terms="1:10"),
    BB = qcv(terms="1:2, 3:10, 19"),
    CC = qcv(terms="1:3, 5:6")
)

foo2 <- list(
    AA = qcv(terms="4:8"),
    BB = qcv(terms="1:4, 10:12"),
    CC = qcv(terms="1, 11, 15:20"),
    DD = qcv(terms="")
)

(x <- cm_range2long(foo))
(out1 <- cm_long2dummy(x))

(z <- cm_range2long(foo, foo2, v.name="time"))
(out2 <- cm_long2dummy(z, "time")
lapply(out2, head)
cm_combine_dummy(out1, combine.code = list(AB=qcv(AA, BB)))

combines <- list(AB=qcv(AA, BB), ABC=qcv(AA, BB, CC))
A <- cm_combine_dummy(out2, combine.code = combines)
head(A, 10)
B <- cm_combine_dummy(out1, combine.code = combines)
head(B, 10)

cm_dummy2long(A)
cm_dummy2long(B)
plot(cm_dummy2long(A))

## End(Not run)
```

---

**cm_long2dummy**

**Stretch and Dummy Code cm_xxx2long**

Description

Stretches and dummy codes a cm_xxx2long dataframe to allow for combining columns.

Usage

```
cm_long2dummy(dataframe, rm.var = NULL, code = "code", start = "start", end = "end")
```
Arguments
dataframe  A dataframe that contains the person variable.
rmNvar     An optional character argument of the name of a repeated measures column.
code       A character argument of the name of a repeated measures column. Default is "code".
start      A character argument of the name of a repeated measures column. Default is "start".
end        A character argument of the name of a repeated measures column. Default is "end".

Value

Returns a dataframe or a list of stretched and dummy coded dataframe(s).

See Also

cm_range2long, cm_time2long, cm_df2long

Examples

```r
## Not run:
foo <- list(
  AA = qcv(terms="1:10"),
  BB = qcv(terms="1:2, 3:10, 19"),
  CC = qcv(terms="1:3, 5:6")
)

foo2 <- list(
  AA = qcv(terms="4:8"),
  BB = qcv(terms="1:4, 10:12"),
  CC = qcv(terms="1, 11, 15:20"),
  DD = qcv(terms="")
)

(x <- cm_range2long(foo))
cm_long2dummy(x)

(z <- cm_range2long(foo, foo2, v.name="time"))
out <- cm_long2dummy(z, "time")
ltruncdf(out)

## End(Not run)
```
Description

Generates a range coding sheet for coding words.

Usage

```
cm_range.temp(codes, text.var = NULL, grouping.var = NULL, file = NULL)
```

Arguments

- `codes`: Character vector of codes.
- `text.var`: The text variable.
- `grouping.var`: The grouping variables. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `file`: A connection, or a character string naming the file to print to (.txt or .doc is recommended).

References


See Also

`cm_time.temp`

Examples

```r
## Not run:
cm_range.temp(qcv(AA, BB, CC))
with(DATA, cm_range.temp(qcv(AA, BB, CC), state, list(person, adult)))
## cm_range.temp(qcv(AA, BB, CC), file = "foo.txt")
## library(reports); delete("foo.txt")
## End(Not run)
```
Transform Codes to Start-End Durations

Description

Transforms the range coding structure(s) from cm_range.temp (in list format) into a data frame of start and end durations in long format.

Usage

cm_range2long(..., v.name = "variable", list.var = TRUE, debug = TRUE, object = NULL)

Arguments

v.name An optional name for the column created for the list.var argument.
list.var logical. If TRUE creates a column for the data frame created by each time.list passed to cm_t2l.
debug logical. If TRUE debugging mode is on. cm_time2long will return possible errors in time span inputs.
object A list of list object(s) generated by cm_time.temp.
... list object(s) in the form generated by cm_time.temp.

Value

Generates a data frame of start and end spans for each code.

References


See Also

cm_df2long, cm_time.temp, cm_df.transcript

Examples

```
## Not run:
foo <- list(
  person_greg = qcv(terms='7:11, 20:24, 30:33, 49:56'),
  person_researcher = qcv(terms='42:48'),
  person_sally = qcv(terms='25:29, 37:41'),
  person_sam = qcv(terms='1:6, 16:19, 34:36'),
  person_teacher = qcv(terms='12:15'),
  adult_0 = qcv(terms='1:11, 16:41, 49:56'),
  adult_1 = qcv(terms='12:15, 42:48'),
)```
**cm_time.temp**

```r
AA = qcv(terms="1"),
BB = qcv(terms="1:2, 3:10, 19"),
CC = qcv(terms="1:9, 100:150")
```

```r
foo2 <- list(
  person_greg = qcv(terms='7:11, 20:24, 30:33, 49:56'),
  person_researcher = qcv(terms='42:48'),
  person_sally = qcv(terms='25:29, 37:41'),
  person_sam = qcv(terms='1:6, 16:19, 34:36'),
  person_teacher = qcv(terms='12:15'),
  adult_0 = qcv(terms='1:11, 16:41, 49:56'),
  adult_1 = qcv(terms='12:15, 42:48'),
  AA = qcv(terms="40"),
  BB = qcv(terms="50:90"),
  CC = qcv(terms="60:90, 100:120, 150"),
  DD = qcv(terms="")
)
```

```r
## General ldots Approach
(dat <- cm_range2long(foo, foo2, v.name = "time"))
plot(dat)

## Specify `object` Approach
cm_range2long(object=list(foo=foo))
cm_range2long(object=list(foo=foo, foo2=foo2), v.name="time")
cm_range2long(object=list(a=foo, b=foo2), v.name="time")

## End(Not run)
```

---

**cm_time.temp**

**Time Span Code Sheet**

---

**Description**

Generates a time span coding sheet and coding format sheet.

**Usage**

```r
cm_time.temp(codes, grouping.var = NULL, start = "00:00", end = NULL,  
file = NULL, coding = FALSE, print = TRUE)
```

**Arguments**

- **codes** List of codes.
- **grouping.var** The grouping variables. Also takes a single grouping variable or a list of 1 or more grouping variables.
- **start** A character string in the form of "00:00" indicating start time (default is ":00").
- **end** A character string in the form of "00:00" indicating end time.
file  A connection, or a character string naming the file to print to (.txt or .doc is recommended).
coding  logical. If TRUE a coding list is provided with the time span coding sheet. coding is ignored if end = NULL.
print  logical. If TRUE the time spans are printed to the console.

References

See Also
  cm_range.temp,

Examples
  ## Not run:
  ## cm_time.temp(qcv(AA, BB, CC), ":30", "7:40", file = "foo.txt")
  ## library(reports); delete("foo.txt")
  cm_time.temp(qcv(AA, BB, CC), ":30", "7:40")
  
  x <- list(
    transcript_time_span = qcv(terms="00:00 - 1:12:00"),
    A = qcv(terms="2.40:3.00, 5.01, 6.52:7.00, 9.00"),
    B = qcv(terms="2.40, 3.01:3.02, 5.01, 6.52:7.00, 9.00, 1.12.00:1.19.01"),
    C = qcv(terms="2.40:3.00, 5.01, 6.52:7.00, 9.00, 17.01")
  )
  cm_time2long(x)
  cm_time.temp(qcv(AA, BB, CC))
  
  ## End(Not run)

---

### Description
Transforms the range coding structure(s) from `cm_time.temp` (in list format) into a data frame of start and end times in long format.

### Usage
```r
cm_time2long(..., v.name = "variable", list.var = TRUE, debug = TRUE, object = NULL)
```
cm_time2long

Arguments

v.name  An optional name for the column created for the list.var argument
list.var logical. If TRUE creates a column for the data frame created by each time.list
         passed to cm_tRl.
debug   logical. If TRUE debugging mode is on. cm_time2long will return possible
         errors in time span inputs.
object  A list of list object(s) generated by cm_time.temp.
...     List object(s) in the form generated by cm_time.temp.

Value

Generates a dataframe of start and end times for each code.

References

Miles, M. B. & Huberman, A. M. (1994). An expanded sourcebook: Qualitative data analysis. 2nd

See Also

cm_df2long, cm_time.temp

Examples

## Not run:
x <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00"),
  B = qcv(terms = "2.40, 3.01;3.02, 5.01, 6.02:7.00,
         9.00, 1.12:00;1.19:01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 17.01")
)
(dat <- cm_time2long(x))
plot(dat)

bar1 <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00"),
  B = qcv(terms = "2.40, 3.01;3.02, 5.01, 6.02:7.00, 9.00,
         1.12:00;1.19:01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 16.25:17.01")
)

bar2 <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00"),
  B = qcv(terms = "2.40, 3.01;3.02, 5.01, 6.02:7.00, 9.00,
         1.12:00;1.19:01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 17.01")
)
## General ldots Approach

cm_time2long(bar1)
cm_time2long(bar1, bar2, v.name="time")

## Specify 'object' Approach

cm_time2long(object=list(bar1=bar1))
cm_time2long(object=list(bar1=bar1, bar2=bar2), v.name="time")
cm_time2long(object=list(a=bar1, b=bar2), v.name="time")

## End(Not run)

---

### colcomb2class

Combine Columns to Class

**Description**

Combine columns from qdap classes or a data.frame.

**Usage**

```r
colcomb2class(dataframe, combined.columns, class = "list", percent = TRUE, 
digits = 2, elim.old = TRUE, zero.replace = 0, override = FALSE)
```

**Arguments**

- **dataframe** A dataframe or qdap class (e.g., termco, question_type, pos_by, character_table).
- **combined.columns** A list of named vectors of the colnames/indexes of the numeric columns to be combined (summed). If a vector is unnamed a name will be assigned.
- **class** The class to assign to the output.
- **percent** logical. If TRUE output given as percent. If FALSE the output is proportion.
- **digits** Integer; number of decimal places to round when printing.
- **elim.old** logical. If TRUE eliminates the columns that are combined together by the named match.list. TRUE outputs the table proportionally (see `prop`).
- **zero.replace** Value to replace 0 values with.
- **override** logical. If TRUE the printing options (e.g., percent, digits, etc.) of the dataframe argument are overrode.

**Value**

Returns a list with raw counts, percents and combined raw and percents.
Examples

```r
## Not run:
## 'termco' example
ml <- list(
  cat1 = c("the ", " a ", " an ",
  cat2 = c(" I ", "good",
    the = c("the", " the ", " the", "the")
)
dat1 <- with(raj.act.1, termco(dialogue, person, ml))
colcomb2class(dat1, list(cats = c("cat1", "cat2")))

## 'question_type' example
dat2 <- question_type(DATA.SPLIT$state, DATA.SPLIT$person)
combs <- list(
  'wh/how' = c("what", "how"),
  oth = c("shall", "implied_do/does/did")
)
colcomb2class(dat2, combs)

## 'pos_by' example
dat3 <- with(DATA, pos_by(state, list(adult, sex)))
colcomb2class(dat3, qcv(DT, EX, FW))

## data.frame example
dat4 <- data.frame(X=LETTERS[1:5], matrix(sample(0:5, 20, TRUE), ncol = 4))
colcomb2class(dat4, list(new = c("X1", "X4")))
## End(Not run)
```

Description

Separates a `paste2` column into separate columns.

Usage

```r
colSplit(column, col.sep = ".", name.sep = "&")
```

Arguments

- **column**: The pasted vector.
- **col.sep**: The column separator used in `paste2`.
- **name.sep**: Name separator used in the column (generally for internal use with `colsplit2df`).
Value
Returns a dataframe of split columns.

See Also
colsplit2df, paste2

Examples
```r
## Not run:
foo1 <- paste2(CO2[, 1:3])
head(foo1, 12)
bar1 <- colSplit(foo1)
head(bar1, 10)

foo2 <- paste2(mtcars[, 1:3], sep="|")
head(foo2, 12)
bar2 <- colSplit(foo2, col.sep = "|")
head(bar2, 10)
## End(Not run)
```

colsplit2df
Wrapper for colSplit that Returns Dataframe(s)

Description
colsplit2df - Wrapper for colSplit that returns a dataframe.
lcolsplits2df - Wrapper for colsplit2df designed for qdap lists that returns a list dataframes.

Usage
colsplit2df(dataframe, splitcols = 1, new.names = NULL, sep = ". ",
             keep.orig = FALSE, name.sep = "&", index.names = FALSE)
lcolsplits2df(qdap.list, keep.orig = FALSE)

Arguments
dataframe A dataframe with a column that has been pasted together.
splitcols The name/index of the column(s) that has been pasted together.
new.names A character vector of new names to assign to the columns (or list of names if multiple columns are being split). Default attempts to extract the original names before the paste.
sep The character(s) that was used in paste2 to paste the columns.
keep.orig logical. If TRUE the original pasted column will be retained as well.
name.sep  The character(s) that was used to paste the column names.
index.names logical. If TRUE names of columns that are duplicated are indexed with c("name.1", "name.2", ... "name.n").
qdap.list  A qdap list object that contains dataframes with a leading paste2 column.

Value
colsplit2df - returns a dataframe with the paste2 column split into new columns.
lcolsplit2df - returns a list of dataframes with the paste2 column split into new columns.

Warning
This will strip the class of the qdap object.

Note
lcolsplit2df is a convenience function that is less flexible than colsplit2df but operates on multiple dataframes at once.

See Also
colSplit, colpaste2df paste2

Examples

## Not run:
CO2$'Plant&Type&Treatment' <- paste2(CO2[, 1:3])
CO2 <- CO2[, -c(1:3)]
head(CO2)
head(colsplit2df(CO2, 3))
head(colsplit2df(CO2, 3, qcv(A, B, C)))
head(colsplit2df(CO2, 3, qcv(A, B, C), keep_orig=TRUE))
head(colsplit2df(CO2, "Plant&Type&Treatment"))
CO2 <- datasets::CO2

(dat <- colpaste2df(head(mtcars), list(1:3), sep = "|"))) colsplit2df(dat, 12, sep = "|")

## Multiple split example
E <- list(
  c(1, 2, 3, 4, 5),
  qcv(mpg, hp),
  c("disp", "am")
)

(dat2 <- colpaste2df(head(mtcars), E, sep="|"))
cols <- c("mpg&cyl&disp&hp&drat", "mpg&hp", "disp&am")
colsplit2df(dat2, cols, sep = "|")

## lcolsplit2df example
(x <- with(DATA.SPLIT, question_type(state, list(sex, adult)))))
Ensure Space After Comma

Description

Adds a space after a comma as strip and many other functions may consider a comma separated string as one word (i.e., "one, two, three" becomes "onetwothree" rather than "one two three").

Usage

comma_spacer(text.var)

Arguments

text.var The text variable.

Value

Returns a vector of strings with commas that have a space after them.

Examples

```r
## Not run:
x <- c("the, dog,went", "I,like, it", "where are you", NA, "why", ",", ",,"f")
comma_spacer(x)
## End(Not run)
```

Find Common Words Between Groups

Description

Find common words between grouping variables (e.g., people).

Usage

```r
common(word.list, overlap = "all", equal.or = "more", ...)
```
Arguments

word.list: A list of named character vectors.
overlap: Minimum/exact amount of overlap.
equal.or: A character vector of c("equal","greater","more","less").
...: In lieu of word.list the user may input n number of character vectors.

Value

Returns a dataframe of all words that match the criteria set by overlap and equal.or.

Description

list Method for common

Usage

```r
## S3 method for class 'list'
common(word.list, overlap = "all", equal.or = "more", ...)  
```
counts

Value
Returns a dataframe with condensed columns that can be wrote to csv/xlsx.

See Also
mcsv_w

Examples
```r
## Not run:
library(qdap)
poldat <- with(DATA.SPLIT, polarity(state, person))
write.csv(x = condense(counts(poldat)), file = "foo.csv")

## End(Not run)
```

counts Generic Counts Method

Description
Access the count dataframes from select qdap outputs.

Usage
```r
counts(x, ...)
```

Arguments
```
x A qdap object (list) with a count dataframe (e.g., fry).
... Arguments passed to counts method of other classes.
```

Value
Returns a data.frame of counts.

See Also
scores, proportions, preprocessed, visual
counts.automated_readability_index

Description

counts.automated_readability_index - View counts from automated_readability_index.

Usage

```
## S3 method for class 'automated_readability_index'
counts(x, ...)
```

Arguments

- `x` The automated_readability_index object.
- `...` ignored automated_readability_index Method for counts.

counts.character_table

Term Counts

Description

View character_table counts.

Usage

```
## S3 method for class 'character_table'
counts(x, ...)
```

Arguments

- `x` The character_table object.
- `...` ignored

Details

character_table Method for counts
counts.coleman_liau  

Readability Measures

Description

counts.coleman_liau - View counts from coleman_liau.

Usage

```r
## S3 method for class 'coleman_liau'
counts(x, ...)
```

Arguments

- `x`  
  The coleman_liau object.
- `...`  
  ignored

Details

coleman_liau Method for counts.

counts.end_mark_by  

Question Counts

Description

View end_mark_by counts.

Usage

```r
## S3 method for class 'end_mark_by'
counts(x, ...)
```

Arguments

- `x`  
  The end_mark_by object.
- `...`  
  ignored

Details

end_mark_by Method for counts
counts.flesch.kincaid  Readability Measures

Description

counts.flesch.kincaid - View counts from flesch.kincaid.

Usage

```r
## S3 method for class 'flesch.kincaid'
counts(x, ...)
```

Arguments

- `x` The flesch.kincaid object.
- `...` ignored

Details

flesch.kincaid Method for counts.

counts.formality  Formality

Description

View formality counts.

Usage

```r
## S3 method for class 'formality'
counts(x, ...)
```

Arguments

- `x` The formality object.
- `...` ignored

Details

formality Method for counts
counts.fry  

**Description**  
counts.fry - View counts from fry.

**Usage**  
```r  
## S3 method for class 'fry'  
counts(x, ...)  
```

**Arguments**  
- `x`  
The fry object.
- `...`  
ignored

**Details**  
fry Method for counts.

counts.linsear_write  

**Description**  
counts.linsear_write - View counts from linsear_write.

**Usage**  
```r  
## S3 method for class 'linsear_write'  
counts(x, ...)  
```

**Arguments**  
- `x`  
The linsear_write object.
- `...`  
ignored

**Details**  
linsear_write Method for counts.
counts.object_pronoun_type

**Description**

View object_pronoun_type counts.

**Usage**

```r
## S3 method for class 'object_pronoun_type'
counts(x, ...)
```

**Arguments**

- `x` The object_pronoun_type object.
- `...` ignored

**Details**

object_pronoun_type Method for counts

---

counts.polarity

**Description**

counts.polarity - View counts from polarity.

**Usage**

```r
## S3 method for class 'polarity'
counts(x, ...)
```

**Arguments**

- `x` The polarity object.
- `...` ignored

**Details**

polarity Method for counts.
counts.pos

**Description**

View pos counts.

**Usage**

```r
## S3 method for class 'pos'
counts(x, ...)
```

**Arguments**

- `x`: The pos object.
- `...`: ignored

**Details**

pos Method for counts

counts.pos_by

**Description**

View pos_by counts.

**Usage**

```r
## S3 method for class 'pos_by'
counts(x, ...)
```

**Arguments**

- `x`: The pos_by object.
- `...`: ignored

**Details**

pos_by Method for counts
**counts.pronoun_type**  
*Question Counts*

**Description**

View pronoun_type counts.

**Usage**

```r
## S3 method for class 'pronoun_type'
counts(x, ...)
```

**Arguments**

- `x`  
The pronoun_type object.
- `...`  
  ignored

**Details**

pronoun_type Method for counts

---

**counts.question_type**  
*Question Counts*

**Description**

View question_type counts.

**Usage**

```r
## S3 method for class 'question_type'
counts(x, ...)
```

**Arguments**

- `x`  
The question_type object.
- `...`  
  ignored

**Details**

question_type Method for counts
counts.SMOG **Readability Measures**

**Description**

Counts.SMOG - View counts from SMOG.

**Usage**

```r
## S3 method for class 'SMOG'
counts(x, ...)
```

**Arguments**

- `x` The SMOG object.
- `...` ignored

**Details**

SMOG Method for counts.

[counts.subject_pronoun_type](#) **Question Counts**

**Description**

View subject_pronoun_type counts.

**Usage**

```r
## S3 method for class 'subject_pronoun_type'
counts(x, ...)
```

**Arguments**

- `x` The subject_pronoun_type object.
- `...` ignored

**Details**

subject_pronoun_type Method for counts
counts.termco  Term Counts

Description
View termco counts.

Usage
## S3 method for class 'termco'
counts(x, ...)

Arguments
x The termco object.
... ignored

Details
termco Method for counts

counts.word_length  Word Length Counts

Description
View word_length counts.

Usage
## S3 method for class 'word_length'
counts(x, ...)

Arguments
x The word_length object.
... ignored

Details
word_length Method for counts
## counts.word_position

### Word Position

**Description**

View word_position counts.

**Usage**

```r
# S3 method for class 'word_position'
counts(x, ...)
```

**Arguments**

- `x`  
  The `word_position` object.
- `...`  
  Ignored

**Details**

`word_position` Method for counts

## counts.word_stats

### Word Stats

**Description**

View word_stats counts.

**Usage**

```r
# S3 method for class 'word_stats'
counts(x, ...)
```

**Arguments**

- `x`  
  The `word_stats` object.
- `...`  
  Ignored

**Details**

`word_stats` Method for counts
**Cumulative Scores**

**Description**

- `cumulative` - Generate rolling/cumulative scores for select `qdap` objects.
- `cumulative.end_mark` - Generate end_mark over time (duration in sentences).
- `cumulative.formality` - Generate formality over time (duration in sentences).
- `cumulative.pos` - Generate formality over time (duration in sentences).
- `cumulative.pos_by` - Generate formality over time (duration in sentences).
- `cumulative.animated_formality` - Generate animated formality over time (duration in sentences).
- `cumulative.lexical_classification` - Generate lexical_classification over time (duration in sentences).
- `cumulative.animated_lexical_classification` - Generate animated lexical_classification over time (duration in sentences).
- `cumulative.polarity` - Generate polarity over time (duration in sentences).
- `cumulative.animated_polarity` - Generate animated polarity over time (duration in sentences).
- `cumulative.syllable_freq` - Generate syllable_freq over time (duration in sentences).
- `cumulative.combo_syllable_sum` - Generate combo_syllable_sum over time (duration in sentences).

**Usage**

```r
cumulative(x, 
```

```r
## S3 method for class 'end_mark'
cumulative(x, 
```

```r
## S3 method for class 'formality'
cumulative(x, 
```

```r
## S3 method for class 'pos'
cumulative(x, 
```

```r
## S3 method for class 'pos_by'
cumulative(x, 
```

```r
## S3 method for class 'animated_formality'
cumulative(x, 
```

```r
## S3 method for class 'lexical_classification'
cumulative(x, 
```
## Arguments

- **x**: A qdap object with an accompanying `cumulative` method.
- **...**: Ignored

---

**Description**

A fictitious dataset useful for small demonstrations.

**Usage**

```r
data(DATA)
```

**Format**

A data frame with 11 rows and 5 variables

**Details**

- **person**: Speaker
- **sex**: Gender
- **adult**: Dummy coded adult (0-no; 1-yes)
- **state**: Statement (dialogue)
- **code**: Dialogue coding scheme
DATA.SPLIT

**Fictitious Split Sentence Classroom Dialogue**

**Description**

A `sentSplit` version of the DATA dataset.

**Usage**

```r
data(DATA.SPLIT)
```

**Format**

A data frame with 15 rows and 8 variables

**Details**

- person. Speaker
- tot. Turn of talk with sub sentences
- TOT. Turn of talk
- sex. Gender
- adult. Dummy coded adult (0-no; 1-yes)
- code. Dialogue coding scheme
- state. Statement (dialogue)
- stem.text. A stemmed version of the text.var

DATA2

**Fictitious Repeated Measures Classroom Dialogue**

**Description**

A repeated measures version of the DATA dataset.

**Usage**

```r
data(DATA2)
```

**Format**

A data frame with 74 rows and 7 variables
Details

- day. Day of observation
- class. Class period/subject of observation
- person. Speaker
- sex. Gender
- adult. Dummy coded adult (0-no; 1-yes)
- state. Statement (dialogue)
- code. Dialogue coding scheme

**dir_map**

Map Transcript Files from a Directory to a Script

**Description**

Generate script text (and optionally output it to the clipboard and/or an external file) that can be used to individually read in every file in a directory and assign it to an object.

**Usage**

```r
dir_map(loc = "DATA/TRANSSCRIPTS/CLEANED_TRNSCRIPTS", obj.prefix = "dat",
        use.path = TRUE, col.names = c("person", "dialogue"), file = NULL,
        copy2clip = interactive())
```

**Arguments**

- **loc**: The path/location of the transcript data files.
- **obj.prefix**: A character string that will be used as the prefix (followed by a unique digit) as the assignment object.
- **use.path**: logical. If TRUE use the actual path to the loc argument. If FALSE, the code may be more portable in that the actual input to loc is supplied to the `read.transcript`
- **col.names**: Supplies a vector of column names to the transcript columns.
- **file**: A connection, or a character string naming the file to print to.
- **copy2clip**: logical. If TRUE attempts to copy the output to the clipboard.

**Details**

Generally, the researcher will want to read in and parse every transcript document separately. The task of writing the script for multiple transcript documents can be tedious. This function is designed to make the process more efficient and less prone to errors.

**Value**

Prints a read in script text to the console, optionally copies the wrapped text to the clipboard on a Mac or Windows machine and optionally prints to an outside file.
discourse_map

Note
skip is set to 0, however, it is likely that this value will need to be changed for each transcript.

See Also
read.transcript

Examples

## Not run:
(DIR <- system.file("extdata/transcripts", package = "qdap"))
dir_map(DIR)

## End(Not run)

discourse_map Discourse Mapping

Description
View the flow of discourse from social actors.

Usage
discourse_map(text.var, grouping.var, edge.constant, sep = ",")

Arguments
text.var The text variable or a "word_stats" object (i.e., the output of a word_stats function).


grouping.var The grouping variables. Also takes a single grouping variable or a list of 1 or more grouping variables.

edge.constant A constant to multiple the edges by. Defaults (if missing) to 2.5 times the number of social actors.

sep The separator character to use between grouping variables.

condense logical. If TRUE sentCombine is used to condense text by grouping variable.

... ignored

Details
For an example of the video generated from the Animate output of discourse_map see: https://www.youtube.com/watch?v=7LcqFZ0DXNo&feature=youtu.be. An HTML output can be viewed: http://trinker.github.io/qdap_examples/animation_dialogue.
Value

Returns a list:

- **raw**
  The dataframe with to and from columns (the edges) + word counts
- **edge_word_count**
  A dataframe of edges and word counts + proportional word count
- **vertex_word_count**
  A dataframe of vertices and word counts + proportional word count
- **plot**
  An *igraph* object

Examples

```r
## Not run:
discourse_map(DA$state, list(DA$person, DA$sex))
x <- with(mraja1, discourse_map(dialogue, person))
x
lview(x)
library(igraph)
plot(visual(x), edge.curved=FALSE)

## Quickly add/remove a title
Title(x) <- "Act 1"
x
Title(x) <- NULL
x

## Augmenting the plot
library(qdapTools)
mygraph <- visual(x)

plot(mygraph, edge.curved=TRUE)

V(mygraph)$sex <- V(mygraph)$name %c% raj.demographics[, 1:2]
V(mygraph)$color <- ifelse(V(mygraph)$sex="f", "pink", "lightblue")

plot(mygraph, edge.curved=TRUE)

V(mygraph)$family <- V(mygraph)$name %c% raj.demographics[, c(1, 3)]
cols <- qcv(blue, red, brown, darkgreen, grey80)
V(mygraph)$label.color <- lookup(V(mygraph)$family, unique(V(mygraph)$family), cols)

plot(mygraph, edge.curved=TRUE)

## Community detection
x <- with(mraja1, discourse_map(dialogue, person))
wc <- walktrap.community(visual(x))
colors <- grDevices::rainbow(max(membership(wc)))
plot(x, vertex.color=colors[membership(wc)])

## Repeated Measures (BASIC EXAMPLE)
```
## First merge data and map to discourse per act
## to separate networks

dat <- key_merge(raj, raj.demographics)
list_dat <- split(dat, dat$act)
plot_dat <- lapply(list_dat, function(x) plot(x, discourse_map(dialogue, person)))

opar <- par()
par(mfrow=c(3, 2), mar=c(0, 0, 3, 0))

lapply(seq_along(plot_dat), function(i){
  plot(plot_dat[[i]])
  graphics::mtext(paste("Act", names(plot_dat)[i]), side=3)
})

## Repeated Measures (EXTENDED EXAMPLE)
##
# fam_key <- data.frame(fam=unique(raj.demographics$fam.aff),
#   cols=qcv(blue, grey10, red, orange),
#   stringsAsFactors = FALSE)

par(mfrow=c(3, 2), mar=c(0, 1, 3, 1))
lapply(seq_along(plot_dat), function(i){
  THE_PLOT <- visual(plot_dat[[i]])

  V(THE_PLOT)$sex <- V(THE_PLOT)$name %/% raj.demographics[, 1:2]
  V(THE_PLOT)$color <- ifelse(V(THE_PLOT)$sex=="f", "pink", "lightblue")
  V(THE_PLOT)$family <- V(THE_PLOT)$name %c% raj.demographics[, c(1, 3)]
  V(THE_PLOT)$label.color <- lookup(V(THE_PLOT)$family, fam_key)

  plot(THE_PLOT, edge.curved=TRUE)
  graphics::mtext(paste("Act", names(plot_dat)[i]), side=3)
})
frame()
bords <- rep("black", 7)
bords[3] <- "white"
legend(.29, .95, c("Female", "Male", NA, as.character(fam_key[, 1])),
      fill=c("pink", "lightblue", NA, fam_key[, 2]), border=bords, cex=1.5)

## Reset graphics margins
par(mar=opar)

## ANIMATION
##
test <- discourse_map(DATA$state, list(DATA$person))

## Very quick, hard to see
Animate(test)
```r
pdf("test.pdf")
  par(mar=c(0, 0, 1, 0))
  Animate(test, title="Test Plot")
dev.off()

## Animate it
##--------
library(animation)
library(igraph)

loc <- reports::folder(animation_dialogue)
ans <- Animate(test)

## Set up the plotting function
oopt <- animation::ani.options(interval = 0.1)

FUN <- function() {
  lapply(seq_along(ans), function(i) {
    par(mar=c(0, 0, 1, 0))
    set.seed(10)
    plot.igraph(ans[[i]], edge.curved=TRUE, layout=layout.circle)
    graphics::mtext("Discourse Map", side=3)
    animation::ani.pause()
  })
}

## Detect OS

type <- if(.Platform$OS.type == "windows") shell else system
saveGIF(FUN(), interval = 0.1, outdir = loc, cmd.fun = type)

saveVideo(FUN(), video.name = "discourse_map.avi", interval = 0.1, outdir = loc)

saveLatex(FUN(), autoplay = TRUE, loop = FALSE, latex.filename = "tester.tex",
  caption = "animated dialogue", outdir = loc, ani.type = "pdf",
  ani.dev = "pdf", ani.width = 5, ani.height = 5.5, interval = 0.1)

saveHTML(FUN(), autoplay = FALSE, loop = TRUE, verbose = FALSE,
  outdir = file.path(loc, "new"), single.opts =
  "'controls': ['first', 'previous', 'play', 'next', 'last', 'loop', 'speed'], 'delayMin': 0")

## More Elaborate Layout

test2 <- with(mrajal, discourse_map(dialogue, person))

loc2 <- reports::folder(animation_dialogue2)
ans2 <- Animate(test2)

## Set up the plotting function
oopt <- animation::ani.options(interval = 0.1)

FUN3 <- function() {
  lapply(seq_along(ans2), function(i) {
    par(mar=c(0, 0, 1, 0))
    set.seed(10)
  })
}
```
dispersion_plot

Lexical Dispersion Plot

Description

Generate a lexical dispersion plot of terms.

Usage

```
dispersion_plot(text.var, match.terms = NULL, grouping.var = NULL, rm.vars = NULL, color = "blue", bg.color = "grey90", horiz.color = "grey85", total.color = "black", symbol = "|", title = "Lexical Dispersion Plot", rev.factor = TRUE, wrap = "", xlab = "Dialogue (Words)", ylab = NULL, size = 4, plot = TRUE, char2space = "~~", apostrophe.remove = FALSE, scales = "free", space = "free", ...)
```

Arguments

- `text.var` - The text variable.
- `match.terms` - A vector of quoted terms or a named list of quoted terms. If the latter terms will be combined into a single unified theme named according to the list names. Note that terms within the vectors of the list cannot be duplicated.
- `grouping.var` - The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `rm.vars` - The repeated measures variables. Default NULL generates one facet for all text. Also takes a single repeated measures variable or a list of 1 or more grouping variables.
- `color` - The color of the word symbols.
- `bg.color` - The background color.
- `horiz.color` - The color of the horizontal tracking stripe. Use `horiz.color = bg.color` to eliminate.
total.color The color to use for summary 'all' group. If NULL totals are dropped.
symbol The word symbol. Default is "|".
title Title of the plot
rev.factor logical. If TRUE reverses the plot order of the factors.
wrap a character to wrap around the words (enables the reader to visualize spaces). Default is "'", use "" to remove.
xlab The x label.
ylab The y label.
size The size of the plotting symbol.
plot logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knitr, sweave, etc. to add additional plot layers.
char2space A vector of characters to be turned into spaces.
apostrophe.remove logical. If TRUE removes apostrophes from the output.
scales Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y")
space If "fixed", the default, all panels have the same size. If "free_y" their height will be proportional to the length of the y scale; if "free_x" their width will be proportional to the length of the x scale; or if "free" both height and width will vary.
... Other argument supplied to strip.

Value
Plots a dispersion plot and invisibly returns the ggplot2 object.

Note
The match.terms is character sensitive. Spacing is an important way to grab specific words and requires careful thought. Using "read" will find the words "bread", "read" "reading", and "ready". If you want to search for just the word "read" you'd supply a vector of c(" read ", " reads ", " reading ", " reader ").

See Also
term_match

Examples

## Not run:
term_match(raj$d dialogue, c(" love ", "love", " night ", "night"))
dispersion_plot(raj$d dialogue, c(" love ", "love", " night ", "night"))
dispersion_plot(raj$d dialogue, c("love", "night"), rm vars = raj$act)
with(raj$SPLIT, dispersion_plot(dialogue, c("love", "night"),
    grouping.var = list(fam.aff, sex), rm vars = act))
## With grouping variables

```r
with(rajSPLIT, dispersion_plot(dialogue, c("love", "night"),
    grouping.var = sex, rm.vars = act))
```

## Drop total with `total.color = NULL`

```r
with(rajSPLIT, dispersion_plot(dialogue, c("love", "night"),
    grouping.var = sex, rm.vars = act, total.color = NULL))
```

## Change color scheme

```r
with(rajSPLIT, dispersion_plot(dialogue, c("love", "night"),
    bg.color = "black", grouping.var = list(fam.aff, sex),
    color = "yellow", total.color = "white", horiz.color="grey20"))
```

## Use `word_list`

### Presidential debates by all

```r
wrds <- word_list(pres_debates2012$dialogue, stopwords = Top200Words)
wrd2 <- c("governor="-romney ",
    "wrds2[-c(3, 12)]"
)
with(pres_debates2012, dispersion_plot(dialogue, wrds2, rm.vars = time))
```

### Presidential debates by person

```r
dat <- pres_debates2012
dat <- dat[dat$person %in% qcv(ROMNEY, OBAMA), ]
```

```r
wordlist2 <- c("tax", "health", "rich", "america", "truth",
    "money", "cost", "governnor", "president", "we ",
    "job", "i ", "you ", "because ", "our ", "years ")
```

```r
with(dat, dispersion_plot(dialogue, wordlist2, total.color = NULL,
    bg.color = "white", grouping.var = person, rm.vars = time,
    color = "black", horiz.color="grey80")
```

```r
wordlist2 <- c("i'd ", "i'll ", "i'm ", "i've ", "i ",
    "we'd ", "we'll ", "we're ", "we've ", "we ",
    "you'd ", "you'll ", "you're ", "you've ", "you ", "your ",
    "he'd ", "he'll ", "he's ", "he ")
```

```r
with(dat, dispersion_plot(dialogue, wordlist2,
    bg.color = "black", grouping.var = person, rm.vars = time,
    color = "yellow", total.color = NULL, horiz.color="grey20")
```

```r
with(dat, dispersion_plot(dialogue, wordlist2,
    bg.color = "black", grouping.var = person, rm.vars = time,
    color = "red", total.color = "white", horiz.color="grey20")
```

### 'match.terms' as a named list

```r
wordlist3 <- list(
    I = c("i'd ", "i'll ", "i'm ", "i've ", "i " ),
    we = c("we'd ", "we'll ", "we're ", "we've ", "we " ),
    you = c("you'd ", "you'll ", "you're ", "you've ", "you ", "your "),
    he = c("he'd ", "he'll ", "he's ", "he ")
)
```
with(dat, dispersion_plot(dialogue, wordlist3,
    bg.color = "grey80", grouping.var = person, rm.vars = time,
    color = "blue", total.color = "grey40", horiz.color="grey20"))

colsplit2df(scores(with(dat, termco(dialogue, list(time, person), wordlist3))))

## Extras:
## Reverse facets

x <- with(pres_debates2012, dispersion_plot(dialogue, wrds2, rm.vars = time))

## function to reverse ggplot2 facets
rev_facet <- function(x) {
    names(x$facet)[1:2] <- names(x$facet)[2:1]
    print(x)
}

rev_facet(x)

## Discourse Markers: See...
## In D. Schiffrin, D. Tannen, & H. E. Hamilton (Eds.), The handbook of
discourse analysis (pp. 54-75). Malden, MA: Blackwell Publishing.

discoure_markers <- list(
    response_cries = c(" oh ", " ah ", " aha ", " ouch ", " yuk "),
    back_channels = c(" uh-huh ", " uuhuh ", " yeah "),
    summons = " hey ",
    justification = " because "
)

(markers <- with(pres_debates2012,
    termco(dialogue, list(person, time), discoure_markers))
))
plot(markers, high="red")

with(pres_debates2012,
    termco(dialogue, list(person, time), discoure_markers, elim.old = FALSE)
)

with(pres_debates2012,
    dispersion_plot(dialogue, unlist(discoure_markers), person, time)
)

## End(Not run)
**Dissimilarity**

**Description**

Uses the distance function to calculate dissimilarity statistics by grouping variables.

**Usage**

```
Dissimilarity(text.var, grouping.var = NULL, method = "prop",
             diag = FALSE, upper = FALSE, p = 2, ...)  
```

**Arguments**

- `text.var` A text variable or word frequency matrix object.
- `grouping.var` The grouping variables. Default `NULL` generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `method` Distance methods (see `dist` function). If "prop" (the default) the result is 1 - "binary".
- `diag` logical. If TRUE returns the diagonals of the matrix. If method = "prop" diagonals will not be returned.
- `upper` logical. If TRUE returns the upper triangle of the matrix.
- `p` The power of the Minkowski distance.
- `...` Other arguments passed to `wfm`.

**Value**

Returns a matrix of dissimilarity values (the agreement between text).

**See Also**

- `dist`

**Examples**

```r
## Not run:
with(DATA, Dissimilarity(state, list(sex, adult)))
with(DATA, Dissimilarity(state, person, diag = TRUE))

## Clustering: Dendrogram
(x <- with(pres.debates2012, Dissimilarity(dialogue, list(person, time))))
fit <- hclust(x)
plot(fit)
## draw dendrogram with red borders around the 3 clusters
rect.hclust(fit, k=3, border=c("red", "purple", "seagreen"))

## Clustering: Dendrogram with p.values
library(pvclust)
wfm.mod <- with(pres.debates2012, wfm(dialogue, list(person, time)))
fit <- suppressMessages(pvclust(wfm.mod, method.hclust="ward",
                               method.dist="euclidean"))
plot(fit)
pvrect(fit, alpha=.95)
```
## Multidimensional Scaling

Based on blog post from Bodong Chen

http://bodongchen.com/blog/?p=301

### Fit it: 2-D

(diss <- with(pres.debates2012, Dissimilarity(dialogue, list(person, time),
   method = "euclidean")))

fit <- cmdscale(diss, eig = TRUE, k = 2)

### Plot it 2-D

points <- data.frame(x = fit$points[, 1], y = fit$points[, 2])

ggplot(points, aes(x = x, y = y)) +
  geom_point(data = points, aes(x = x, y = y, color = rownames(points))) +
  geom_text(data = points, aes(x = x, y = y - 0.2, label = rownames(points)))

### Fit it: 3-D

library(scatterplot3d)

fit <- cmdscale(diss, eig = TRUE, k = 3)

points <- data.frame(colSplit(names(Fit$points[, 1])))

library(qdapTools)

points$colors <- points$X1 %>% data.frame(levels(points$X1),
  qc(v(yellow, yellow, blue, yellow, red, yellow))

points$shape <- points$X2 %>% data.frame(levels(points$X2), c(15, 17, 19))

### Plot it: 3-D

scatterplot3d(fit$points[, 1], fit$points[, 2], fit$points[, 3],
  color = points$colors, pch = points$shape,
  main = "Semantic Space Scaled to 3D", xlab = "x", ylab = "y",
  zlab = "z", type = "h")

legend("bottomright", title="Person",
  qc(Obama, Romney, Other), fill=qc(blue, red, yellow))

legend("topleft", paste("Time", 1:3), pch=c(15, 17, 19))

### Compare to Cosine Similarity

cos_sim <- function(x, y) x * y / sqrt(x*x * y*y)

mat <- matrix(rbinom(500, 0:1, .45), ncol=10)

v_outer(mat, cos_sim)

v_outer(with(DATA, wfm(state, person)), cos_sim)

with(DATA, Dissimilarity(state, person))

### End(Not run)
**dist_tab**

**Description**

Generates a distribution table for vectors, matrices and dataframes.

**Usage**

```r
dist_tab(dataframe, breaks = NULL, digits = 2, ...)
```

**Arguments**

- `dataframe`: A vector or data.frame object.
- `breaks`: Either a numeric vector of two or more cut points or a single number (greater than or equal to 2) giving the number of intervals into which x is to be cut.
- `digits`: Integer indicating the number of decimal places (round) or significant digits (signif.) to be used. Negative values are allowed
- `...`: Other variables passed to `cut`.

**Value**

Returns a list of data frames (or singular data frame for a vector) of frequencies, cumulative frequencies, percentages and cumulative percentages for each interval.

**See Also**

`cut`

**Examples**

```r
## Not run:
dist_tab(rnorm(10000), 10)
dist_tab(sample(c("red", "blue", "gray"), 100, T), right = FALSE)
dist_tab(CO2, 4)

out1 <- dist_tab(mtcars[, 1:3])
ltruncdf(out1, 4)

out2 <- dist_tab(mtcars[, 1:3], 4)
ltruncdf(out2, 4)

wdst <- with(mraja1spl, word_stats(dialogue, list(sex, fam.aff, died)))
out3 <- dist_tab(wdst$gts[1:4])
ltruncdf(out3, 4)
```

## End(Not run)
Diversity Statistics

Description

Transcript apply diversity/richness indices.

Usage

diversity(text.var, grouping.var = NULL)

Arguments

text.var The text variable.

grouping.var The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.

Details

These are the formulas used to calculate the indices:

Shannon index:

\[ H_1(X) = -\sum_{i=1}^{R} p_i \log p_i \]


Simpson index:

\[ D = \frac{\sum_{i=1}^{R} p_i n_i (n_i - 1)}{N(N-1)} \]


Collision entropy:

\[ H_2(X) = -\log \sum_{i=1}^{n} p_i^2 \]


Berger Parker index:

\[ D_{BP} = \frac{N_{max}}{N} \]

Brillouin index:

\[ H_B = \frac{\ln(N!) - \sum \ln(n_i)!}{N} \]


Value

Returns a dataframe of various diversity related indices for Shannon, collision, Berger Parker and Brillouin.

References

http://arxiv.org/abs/physics/0512106

Examples

```r
## Not run:
div.mod <- with(mraja1spl, diversity(dialogue, list(sex, died, fam.aff)))
colsplit2df(div.mod)
plot(div.mod, high = "red", low = "yellow")
plot(div.mod, high = "red", low = "yellow", values = TRUE)
## End(Not run)
```

---

duplicates

Find Duplicated Words in a Text String

Description

Find duplicated word/word chunks in a string. Intended for internal use.

Usage

duplicates(string, threshold = 1)

Arguments

- `string`: A character string.
- `threshold`: An integer of the minimal number of repeats.

Value

Returns a vector of all duplicated words/chunks.
Examples

```r
## Not run:
duplicates(DATA$state)
duplicates(DATA$state[1])

## End(Not run)
```

---

**end_inc**

*Test for Incomplete Sentences*

**Description**

Test for incomplete sentences and optionally remove them.

**Usage**

```r
end_inc(dataframe, text.var, warning.report = TRUE, which.mode = FALSE)
```

**Arguments**

- `dataframe`: A dataframe that contains the person and text variable.
- `text.var`: A character string of the text variable.
- `which.mode`: logical. If TRUE outputs two logical vectors: ‘NOT’ (logical test of not being an incomplete sentence) and ‘INC’ (logical test of being an incomplete sentence)

**Value**

Generates a dataframe with incomplete sentences removed.

**Examples**

```r
## Not run:
dat <- sentSplit(DATA, "state", stem.col = FALSE)
dat$state[c(2, 5)] <- paste(strip(dat$state[c(2, 5)]), "|")
end_inc(dat, "state")
end_inc(dat, "state", warning.report = FALSE)
end_inc(dat, "state", which.mode = TRUE)

## End(Not run)
```
**end_mark**

<table>
<thead>
<tr>
<th>end_mark</th>
<th>Sentence End Marks</th>
</tr>
</thead>
</table>

**Description**

end_mark - Grab the sentence end marks for a transcript. This can be useful to categorize based on sentence type.

demark_by - Grab the sentence end marks for a transcript by grouping variable(s).

**Usage**

```r
end_mark(text.var, missing.end.mark = " ", missing.text = NA,
          other.endmarks = NULL)

end_mark_by(text.var, grouping.var, digits = 3, percent = FALSE,
            zero.replace = 0, ...)
```

**Arguments**

- **text.var**  
The text variable.
- **missing.end.mark**  
A value to use for sentences with missing endmarks.
- **missing.text**  
A value to use for sentences with missing (NA) text.
- **other.endmarks**  
Other 1-2 character endmarks to search for.
- **grouping.var**  
The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- **digits**  
Integer; number of decimal places to round when printing.
- **percent**  
logical. If TRUE output given as percent. If FALSE the output is proportion.
- **zero.replace**  
Value to replace 0 values with.
- **...**  
Other arguments passed to end_mark.

**Value**

Returns a character vector of qdap end marks for each sentence. End marks include:

- "."  
  Declarative sentence.
- "?"  
  Question sentence.
- "!"  
  Exclamatory sentence.
- "|"  
  Incomplete sentence.
- "*."  
  Imperative-declarative sentence.
- "*?"  
  Imperative-question sentence (unlikely to occur)
- "*!"  
  Imperative-exclamatory sentence.
- "*|"  
  Imperative-incomplete sentence.
- "no.em"  
  No end mark.
- "blank"  
  Empty cell/NA.
Examples

```r
## Not run:
end_mark(DATA.SPLIT$state)
end_mark(mraja$split$dialogue)
table(end_mark(mraja$split$dialogue))
plot(end_mark(mraja$split$dialogue))
ques <- mraja$split[end_mark(mraja$split$dialogue) == "?",] # grab questions
htrcdf(ques)
non.ques <- mraja$split[end_mark(mraja$split$dialogue) != "?",] # non questions
htrcdf(non.ques, 20)
ques.per <- mraja$split[end_mark(mraja$split$dialogue) %in% c(".", "?"),] # grab ? and .
htrcdf(ques.per, 20)

(x_by <- end_mark_by(DATA.SPLIT$state, DATA.SPLIT$person))
scores(x_by)
counts(x_by)
proportions(x_by)
preprocessed(x_by)
plot(scores(x_by))
plot(counts(x_by))
plot(proportions(x_by))
plot(preprocessed(x_by))

===============================================
## End Marks Over Time Examples ##
===============================================

## EXAMPLE 1
sentpres <- lapply(with(pres_debates2012, split(dialogue, time)), function(x) {
  end_mark(x)
})
sentplots <- lapply(seq_along(sentpres), function(i) {
  m <- plot(cumulative(sentpres[[i]]))
  if (i != 2) m <- m + ylab("")
  if (i != 3) m <- m + xlab(NULL)
  m + ggtitle(paste("Debate", i))
})
library(grid)
library(gridExtra)
do.call(grid.arrange, sentplots)

## EXAMPLE 2
sentraj <- lapply(with(rajSPLIT, split(dialogue, act)), function(x) {
  end_mark(x)
})
sentplots2 <- lapply(seq_along(sentraj), function(i) {
  m <- plot(cumulative(sentraj[[i]]))
  if (i != 2) m <- m + ylab("")
  if (i != 3) m <- m + xlab(NULL)
  act <- qcv(I, II, III, IV, V)
```

env.syl

```r
m + ggtitle(paste("Act", act[i]))
}

## ggplot2 function to extract legend
g_legend <- function(a_gplot)
  tmp <- ggplot_gtable(ggplot_build(a_gplot))
  leg <- which(sapply(tmp[["grobs"]], function(x) x[["name"]]) == "guide-box")
  legend <- tmp[["grobs"]][[leg]]
  legend

## remove legends from plots
sentplots3 <- lapply(sentplots2, function(x){
  x + theme(legend.position="none") + xlab(NULL) + ylab(NULL)
})
sentplots3[[6]] <- g_legend(sentplots2[[1]])
do.call(grid.arrange, sentplots3)

## End(Not run)
```

env.syl

**Syllable Lookup Environment**

**Description**
A dataset containing a syllable lookup environment (see DICTIONARY).

**Usage**

```r
data(env.syl)
```

**Format**
A environment with the DICTIONARY data set.

**Details**
For internal use.

**References**

UCI Machine Learning Repository website
**Exclude Elements From a Vector**

**Description**

`exclude` - Quickly exclude words from a word list

- `exclude.wfm` - wfm method for `exclude`.
- `exclude.list` - list method for `exclude`.
- `exclude.default` - default method for `exclude`.

%.ex% - Binary operator version of `exclude`.

**Usage**

```r
exclude(word.list, ...)

## S3 method for class 'TermDocumentMatrix'
exclude(word.list, ...)

## S3 method for class 'DocumentTermMatrix'
exclude(word.list, ...)

## S3 method for class 'wfm'
exclude(word.list, ...)

## S3 method for class 'list'
exclude(word.list, ...)

## Default S3 method:
exclude(word.list, ...)

word.list %.ex% ...
```

**Arguments**

- **word.list**
  A list/vector of words/terms, a `wfm`, `DocumentTermMatrix`, or `TermDocumentMatrix` to exclude from.

- **...**
  A vector (character/numeric) if element(s) to be excluded from the `word.list`.

**Value**

Returns a vector with the excluded terms removed.
**Examples**

```r
## Not run:
exclude(1:10, 3, 4)
exclude(1:10, 3:4)
Top25Words
exclude(Top25Words, qcv(the, of, and))
exclude(Top25Words, "the", "of", "an")

# Using with term_match and termco
terms <- term_match(DATA$state, qcv(th), FALSE)
exclude(terms, "truth")
# All together
termino(DATA$state, DATA$person, exclude(term_match(DATA$state, qcv(th),
FALSE), "truth"))

MTCH.LST <- exclude(term_match(DATA$state, qcv(th, i)), qcv(truth, stinks))
termino(DATA$state, DATA$person, MTCH.LST)

## Works with wfm
dat <- wfm(DATA$state, DATA$person)
the.no <- term_match(DATA$state, c("the", "no"))
exclude(dat, unlist(the.no))

## Works with tm's TermDocumentMatrix/DocumentTermMatrix
dat2 <- as.dtm(DATA$state, DATA$person)
out.dtm <- exclude(dat2, unlist(the.no))
tm::inspect(out.dtm)

dat3 <- as.tdm(DATA$state, DATA$person)
out.tdm <- exclude(dat3, unlist(the.no))
tm::inspect(out.tdm)

## End(Not run)
```

---

**Description**

Filter.all_words - Filter words from a all_words that meet max/min word length criteria.

Filter.TermDocumentMatrix - Filter words from a TermDocumentMatrix vector that meet max/min word length criteria.

Filter.DocumentTermMatrix - Filter words from a DocumentTermMatrix that meet max/min word length criteria.

Filter - Filter words from various objects that meet max/min word length criteria.

Filter.wfm - Filter words from a wfm that meet max/min word length criteria.

Filter.character - Filter words from a character vector that meet max/min word length criteria.
Filter.fwl - Filter words from a fwl that meet max/min word length criteria.
Filter.fswl - Filter words from a fswl that meet max/min word length criteria.
Filter.rfswl - Filter words from a rfswl that meet max/min word length criteria.

Usage

```r
## S3 method for class 'all_words'
Filter(x, min = 1, max = Inf, count.apostrophe = TRUE,
       stopwords = NULL, ignore.case = TRUE, ...)
```

```r
## S3 method for class 'TermDocumentMatrix'
Filter(x, min = 1, max = Inf,
       count.apostrophe = TRUE, stopwords = NULL, ignore.case = TRUE, ...)
```

```r
## S3 method for class 'DocumentTermMatrix'
Filter(x, min = 1, max = Inf,
       count.apostrophe = TRUE, stopwords = NULL, ignore.case = TRUE, ...)
```

```r
## S3 method for class 'wfm'
Filter(x, min = 1, max = Inf, count.apostrophe = TRUE,
       stopwords = NULL, ...)
```

```r
## S3 method for class 'character'
Filter(x, min = 1, max = Inf, count.apostrophe = TRUE,
       stopwords = NULL, ignore.case = TRUE, ...)
```

```r
## S3 method for class 'fwl'
Filter(x, min = 1, max = Inf, count.apostrophe = TRUE,
       stopwords = NULL, ignore.case = TRUE, ...)
```

```r
## S3 method for class 'fswl'
Filter(x, min = 1, max = Inf, count.apostrophe = TRUE,
       stopwords = NULL, ignore.case = TRUE, ...)
```

```r
## S3 method for class 'rfswl'
Filter(x, min = 1, max = Inf, count.apostrophe = TRUE,
       stopwords = NULL, ignore.case = TRUE, ...)
```

Arguments

- **x**: A filterable object (e.g., `wfm`, `character`).
- **min**: Minimum word length.
- **max**: Maximum word length.
- **count.apostrophe**: logical. If TRUE apostrophes are counted as characters.
Filter.all_words

stopwords A vector of stop words to remove.
ignore.case logical. If TRUE stopwords will be removed regardless of case (ignored if used on a wfm).
... Other arguments passed to specific Filter methods.

Details

all_words Method for Filter
TermDocumentMatrix Method for Filter
DocumentTermMatrix Method for Filter
class Method for Filter
fwl Method for Filter
fswl Method for Filter
rfswl Method for Filter

Value

Filter.all_words - Returns a matrix of the class "all_words".
Filter.TermDocumentMatrix - Returns a matrix of the class "TermDocumentMatrix".
Filter.DocumentTermMatrix - Returns a matrix of the class "DocumentTermMatrix".
Filter - Returns a matrix of the class "wfm".
Filter.character - Returns a vector of the class "character".
Filter.wfm - Returns a matrix of the class "wfm".
Filter.fwl - Returns a matrix of the class "fwl".
Filter.fswl - Returns a matrix of the class "fswl".
Filter.rfswl - Returns a matrix of the class "rfswl".

Note

The name and idea behind this function is inspired by the dplyr package’s filter function and has a similar meaning in that you are grabbing rows (or elements) meeting a particular criteria.

Examples

## Not run:
Filter(with(DATA, wfm(state, list(sex, adult)))))
with(DATA, wfm(state, list(sex, adult)))

## Filter particular words based on max/min values in wfm
v <- with(DATA, wfm(state, list(sex, adult)))
Filter(v, 5)
Filter(v, 5, count.apostrophe = FALSE)
Filter(v, 5, 7)
Filter(v, 4, 4)
Filter(v, 3, 4)
formality

Filter(v, 3, 4, stopwords = Top25Words)

## Filter works on character strings too...
x <- c("Raptors don't like robots!", "I'd pay $500.00 to rid them."")
Filter(x, 3)
Filter(x, 4)
Filter(x, 4, count.apostrophe = FALSE)
Filter(x, 4, count.apostrophe = FALSE, stopwords="raptors")
Filter(x, 4, stopwords="raptors")
Filter(x, 4, stopwords="raptors", ignore.case = FALSE)

DATA[, "state"] <- Filter(DATA[, "state"], 4)
DATA <- qdap::DATA

## Filter 'all_words'
head(all_words(raj$dialogue))
Filter(head(all_words(raj$dialogue)), min = 3)

## End(Not run)

---

<table>
<thead>
<tr>
<th>formality</th>
<th>Formality Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Transcript apply formality score by grouping variable(s) and optionally plot the breakdown of the model.

**Usage**

formality(text.var, grouping.var = NULL, order.by.formality = TRUE,
digits = 2, ...)

**Arguments**

text.var The text variable (or an object from pos, pos_by or formality. Passing the later three object will greatly reduce run time.

grouping.var The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of I or more grouping variables.

order.by.formality logical. If TRUE orders the results by formality score.

digits The number of digits displayed.

... Other arguments passed to pos_by.
Details

Heylighen & Dewaele (2002)’s formality score is calculated as:

\[ F = 50\left(\frac{n_f - n_c}{N} + 1\right) \]

Where:

\[ f = \{\text{noun, adjective, preposition, article}\} \]

\[ c = \{\text{pronoun, verb, adverb, interjection}\} \]

\[ N = \sum (f + c + \text{conjunctions}) \]

Value

A list containing at the following components:

- **text**: The text variable
- **POSTagged**: Raw part of speech for every word of the text variable
- **POSprop**: Part of speech proportion for every word of the text variable
- **POSfreq**: Part of speech count for every word of the text variable
- **pos.by.freq**: The part of speech count for every word of the text variable by grouping variable(s)
- **pos.by.prop**: The part of speech proportion for every word of the text variable by grouping variable(s)
- **form.freq.by**: The nine broad part of speech categories count for every word of the text variable by grouping variable(s)
- **form.prop.by**: The nine broad part of speech categories proportion for every word of the text variable by grouping variable(s)
- **formality**: Formality scores by grouping variable(s)
- **pos.reshaped**: An expanded formality scores output (grouping, word.count, pos & form.class) by word

Warning

Heylighen & Dewaele (2002) state, "At present, a sample would probably need to contain a few hundred words for the measure to be minimally reliable. For single sentences, the F-value should only be computed for purposes of illustration" (p. 24).

References

Examples

```r
## Not run:
with(DATA, formality(state, person))
(x1 <- with(DATA, formality(state, list(sex, adult))))
plot(x1)
plot(x1, short.names = FALSE)
scores(x1)
counts(x1)
proportions(x1)
preprocessed(x1)

plot(scores(x1))
plot(counts(x1))
plot(proportions(x1), high="darkgreen")
plot(preprocessed(x1))

data(rajPOS) # A data set consisting of a pos list object
x2 <- with(raj, formality(rajPOS, act))
plot(x2)
cumulative(x2)
x3 <- with(raj, formality(rajPOS, person))
plot(x3, bar.colors="Dark2")
plot(x3, bar.colors=c("Dark2", "Set1"))
x4 <- with(raj, formality(rajPOS, list(person, act)))
plot(x4, bar.colors=c("Dark2", "Set1"))

rajDEM <- key_merge(raj, raj.demographics) # merge demographics with transcript.
x5 <- with(rajDEM, formality(rajPOS, sex))
plot(x5, bar.colors="RdBu")
x6 <- with(rajDEM, formality(rajPOS, list(fam.aff, sex)))
plot(x6, bar.colors="RdBu")
x7 <- with(rajDEM, formality(rajPOS, list(died, fam.aff)))
plot(x7, bar.colors="RdBu", point.cex=2, point.pch = 3)
x8 <- with(rajDEM, formality(rajPOS, list(died, sex)))
plot(x8, bar.colors="RdBu", point.cex=2, point.pch = ",")

names(x8)
colsplit2df(x8$formality)

# pass an object from pos or pos_by
ltruncdf(with(raj, formality(x8, list(act, person))), 6, 4)

#==========
## ANIMATION ##
#==========
## EXAMPLE 1
form_ani <- formality(DATA.SPLIT$state, DATA.SPLIT$person)
forma <- Animate(form_ani, contextual="white", formal="blue",
current.color = "yellow", current.speaker.color="grey70")
bgb <- vertex_apply(forma, label.color="grey80", size=20, color="grey40")
```
bgb <- edge_apply(bgb, label.color="yellow")

print(bgb, bg="black", net.legend.color = "white", pause=1)

## EXAMPLE 2
form_ani2 <- formality(raj.act.1POS, mraja$person)
forma2 <- Animate(form_ani2, contextual="white", formal="blue",
                  current.color = "yellow", current.speaker.color="grey70")

bgb2 <- vertex_apply(forma2, label.color="grey80", size=17, color="grey40")
bgb2 <- edge_apply(bgb2, label.color="yellow")
print(bgb2, bg="black", pause=.75, net.legend.color = "white")

## EXAMPLE 3 (bar plot)
Animate(form_ani2, as.network=FALSE)

============================
## Complex Animation ##
============================
library(animation)
library(grid)
library(gridBase)
library(qdap)
library(reports)
library(igraph)
library(plotrix)

form_ani2 <- formality(raj.act.1POS, mraja$person)

## Set up the network version
form_net <- Animate(form_ani2, contextual="white", formal="blue",
                     current.color = "yellow", current.speaker.color="grey70")
bgb <- vertex_apply(form_net, label.color="grey80", size=17, color="grey40")
bgb <- edge_apply(bgb, label.color="yellow")

## Set up the bar version
form_bar <- Animate(form_ani2, as.network=FALSE)

## Generate a folder
loc <- reports::folder(animation_formality)

## Set up the plotting function
oopt <- animation::ani.options(interval = 0.1)

FUN <- function(follow=FALSE, theseq = seq_along(bgb)) {
    Title <- "Animated Formality: Romeo and Juliet Act 1"
    Legend <- c(.2, -1, 1.5, -.95)
    Legend.cex <- 1

    lapply(theseq, function(i) {
        }
if (follow) {
    png(file=sprintf("%s/images/Rplot%s.png", loc, i),
        width=650, height=725)
}
## Set up the layout
layout(matrix(c(rep(1, 9), rep(2, 4)), 13, 1, byrow = TRUE))

## Plot 1
par(mar=c(2, 0, 2, 0), bg="black")
#par(mar=c(2, 0, 2, 0))
set.seed(22)
plot.igraph(bgb[[i]], edge.curved=TRUE)
graphics::mtext(title, side=3, col="white")
color.legend(Legend[1], Legend[2], Legend[3], Legend[4],
    c("Contextual", "Formal"), attributes(bgb)["legend"],
    cex = Legend.cex, col="white")

## Plot2
plot.new()
vps <- baseViewports()
uns <- unit(c(-1.3,.5,-.75,.25), "cm")
p <- form_bar[[i]] +
    theme(plot.margin = uns,
          text=element_text(color="white"),
          legend.text=element_text(color="white"),
          legend.background = element_rect(fill = "black"),
          plot.background = element_rect(fill = "black",
                                          color="black"))
print(p, vp = vpStack(vps$figure, vps$plot))
animation::ani.pause()

if (follow) {
    dev.off()
}
}

FUN()

## Detect OS
type <- if (.Platform$OS.type == "windows") shell else system

saveHTML(FUN, 1:20), autoplay = FALSE, loop = TRUE, verbose = FALSE,
     ani.height = 1000, ani.width=650,
     outdir = loc, single.opts =
     "'controls': ['first', 'play', 'loop', 'speed', 'delayMin': 0")

FUN(TRUE)

====================
## Static Network ##
freq_terms

Find Frequent Terms

Description

Find the most frequently occurring terms in a text vector.

Usage

freq_terms(text.var, top = 20, at.least = 1, stopwords = NULL,
          extend = TRUE, ...)

#=====================================================
(formdat <- with(sentSplit(DATA, 4), formality(state, person)))
m <- Network(formdat)
m
print(m, bg="grey97", vertex.color="grey75")

print(m, title="Formality Discourse Map", title.color="white", bg="black",
      legend.text.color="white", vertex.label.color = "grey70",
      edge.label.color="yellow")

## or use themes:
dev.off()
m + qtheme()
m + theme_nightheat
dev.off()
m + theme_nightheat(title="Formality Discourse Map",
                     vertex.label.color = "grey50")

#=====================================================
## Formality Over Time Example ##
#=====================================================
formpres <- lapply(with(pres_debates2012, split(dialogue, time)), function(x) {
  formality(x)
})
formplots <- lapply(seq_along(formpres), function(i) {
  m <- plot(cumulative(formpres[[i]]))
  if (i != 2) m <- m + ylab("")
  if (i != 3) m <- m + xlab(NULL)
  m + ggtitle(paste("Debate", i))
})

library(grid)
library(gridExtra)
do.call(grid.arrange, formplots)

## End(Not run)
freq_terms

Arguments

- **text.var**: The text variable.
- **top**: Top number of terms to show.
- **at.least**: An integer indicating at least how many letters a word must be to be included in the output.
- **stopwords**: A character vector of words to remove from the text. qdap has a number of data sets that can be used as stop words including: Top200Words, Top100Words, Top25Words. For the tm package's traditional English stop words use tm::stopwords("english").
- **extend**: logical. If TRUE the top argument is extended to any word that has the same frequency as the top word.
- **...**: Other arguments passed to `all_words`.

Value

Returns a dataframe with the top occurring words.

See Also

- `word_list`, `all_words`

Examples

```r
## Not run:
freq_terms(DATA$state, 5)
freq_terms(DATA$state)
freq_terms(DATA$state, extend = FALSE)
freq_terms(DATA$state, at.least = 4)
(out <- freq_terms(pres_debates2012$dialogue, stopwords = Top200Words))
plot(out)

## All words by sentence (row)
library(qdapTools)
x <- raj$dialogue
list_df2df(setNames(lapply(x, freq_terms, top=Inf), seq_along(x)), "row")
list_df2df(setNames(lapply(x, freq_terms, top=10, stopwords = Dolch),
     seq_along(x)), "Title")

## All words by person
FUN <- function(x, n=Inf) freq_terms(paste(x, collapse=" "), top=n)
list_df2df(lapply(split(x, raj$person), FUN), "person")

## Plot it
out <- lapply(split(x, raj$person), FUN, n=10)
pdf("Freq Terms by Person.pdf", width=13)
lapply(seq_along(out), function(i) {
    ## dev.new()
    plot(out[[i]], plot=FALSE) + ggtie(title(names(out)[i]))
})
```
## gantt

**Gantt Durations**

**Description**

`gantt` - Generates start and end times of supplied text selections (i.e., text selections are determined by any number of grouping variables).

`plot_gantt_base` - For internal use.

**Usage**

```r
gantt(text.var, grouping.var, units = "words", sums = FALSE,
     col.sep = "\_")

plot_gantt_base(x, sums = NULL, fill.colors = NULL, box.color = "white",
                title = NULL)
```

**Arguments**

- `text.var` The text variable
- `grouping.var` The grouping variables. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `units` The unit of measurement to analyze. One of the strings "character", "syllable", "word", or "sentence".
- `sums` logical. If `TRUE` reports and (optionally (or plots) the total units used by grouping variable(s).
- `col.sep` The character string to use to separate pasted variables in the merged grouping variable header/name.
- `x` an object of the class "gantt".
- `fill.colors` The colors of the Gantt plot bars. Either a single color or a length equal to the number of grouping variable(s). If `NULL`, `rainbow` is used.
- `box.color` A color to wrap the boxes with.
- `title` An optional title.

**Value**

Returns a data frame of start and end times by grouping variable(s) or optionally returns a list of two: (1) A data frame of the total units used by grouping variable(s) and (2) a data frame of start and end times by grouping variable(s).
Note
For non-repeated measures data use `gantt`. For more flexible plotting needs use `gantt_wrap` over the generic plotting method.

Author(s)
DigEmAll (stackoverflow.com) and Tyler Rinker <tyler.rinker@gmail.com>.

References

See Also
`gantt_rep`, `gantt_wrap`, `gantt_plot`

Examples
```r
## Not run:
(a <- gantt(DATA$state, DATA$person))
plot(a)
plot(a, base = TRUE)

(b <- gantt(DATA$state, DATA$person, sums = TRUE))
plot(b)
plot(b, base = FALSE)

(d <- gantt(DATA$state, list(DATA$sex, DATA$adult)))
plot(d)

x <- gantt(mraja1$dialogue, mraja1$person)
plot(x, base = TRUE)
plot(x, , base = TRUE, box.color = "black")

z <- gantt(mraja1$dialogue, mraja1$sex)
plot(z)

e <- with(mraja1, gantt(dialogue, list(fam.aff, sex, died),
units = "characters", sums = TRUE))
plot(e)

f <- gantt(mraja1$dialogue, mraja1$person, units = "syllables",
sums = TRUE)
plot(f, box.color = "red")
plot(f, base = FALSE)

dat <- gantt(mraja1$dialogue, list(mraja1$fam.aff, mraja1$sex),
units = "sentences", col.sep = ",")

## Animate It
```


```r
## gantt_plot
ani_gannt <- with(DATA_SPLIT, gantt(state, person))
Animate(ani_gannt)
Animate(plot(ani_gannt))

library(animation)
loc <- reports::folder(animation_gantt)

## Set up the plotting function
oont <- animation::ani.options(interval = 0.1)

FUN <- function() {
  out <- Animate(ani_gannt)
  lapply(out, function(x) {
    print(x)
    animation::ani.pause()
  })
}

type <- if(.Platform$OS.type == "windows") shell else system
saveGIF(FUN(), interval = 0.1, outdir = loc, cmd.fun = type)

## End(Not run)
```

---

### gantt_plot

**Gantt Plot**

### Description

A convenience function that wraps `gantt`, `gantt_rep` and `gantt_wrap` into a single plotting function.

### Usage

```r
gantt_plot(text.var, grouping.var = NULL, rm.var = NULL, fill.var = NULL,
          xlab = "duration (in words)", units = "words", col.sep = "__", ...)
```

### Arguments

- **text.var**: The text variable.
- **grouping.var**: The grouping variables. Also takes a single grouping variable or a list of 1 or more grouping variables.
- **rm.var**: An optional single vector or list of 1 or 2 of repeated measures to facet by
- **fill.var**: An optional variable to fill the code strips by.
- **xlab**: The name of the x-axis label.
- **units**: The unit of measurement.
- **col.sep**: The column separator.
- **...**: Other arguments passed to `gantt_wrap`. 


**Value**

Returns a Gantt style visualization. Invisibly returns the ggplot2 list object.

**Note**

For non-repeated measures data/plotting use `gantt`; for repeated measures data output use `gantt_rep`; and for a flexible gantt plot that works with code matrix functions (cm) use `gantt_wrap`.

**References**


**See Also**

`gantt`, `gantt_rep`, `gantt_wrap`

**Examples**

```r
## Not run:
with(rajSPLIT, gantt_plot(text.var = dialogue, 
    grouping.var = person, size=4))

with(rajSPLIT, gantt_plot(text.var = dialogue, 
    grouping.var = list(fam.aff, sex), rm.var = act, 
    title = "Romeo and Juliet's dialogue"))

with(rajSPLIT, gantt_plot(dialogue, list(fam.aff, sex), act, 
    transform=T))

rajSPLIT2 <- rajSPLIT 
rajSPLIT2$newb <- as.factor(sample(LETTERS[1:2], nrow(rajSPLIT2), 
    replace=TRUE))

z <- with(rajSPLIT2, gantt_plot(dialogue, list(fam.aff, sex), 
    list(act, newb), size = 4))

library(ggplot2); library(scales); library(RColorBrewer); library(grid) 
z + theme(panel.spacing = unit(1, "lines") + scale_colour_brewer() 
z + scale_colour_brewer(palette="Dark2")

## Fill Variable Example
dat <- rajSPLIT[rajSPLIT$act == 1, ] 
dat$end_mark <- factor(end_mark(dat$dialogue))

with(dat, gantt_plot(text.var = dialogue, grouping.var = list(person, sex), 
    fill.var=end_mark))

## Repeated Measures with Fill Example
rajSPLIT$end_mark <- end_mark(rajSPLIT$dialogue)
```
Generate Unit Spans for Repeated Measures

```r
with(rajsplit, gantt_plot(text.var = dialogue,
    grouping.var = list(fam.aff), rm.var = list(act),
    fill.var=end_mark, title = "Romeo and Juliet's dialogue"))

## Repeated Measures Sentence Type Example
with(rajsplit, gantt_plot(text.var = dialogue,
    grouping.var = list(fam.aff, sex), rm.var = list(end_mark, act),
    title = "Romeo and Juliet's dialogue"))

## Reset rajsplit
rajsplit <- qdap::rajsplit

## Animate It
###
ani_gantt <- with(mraja1, gantt_plot(dialogue, person))

library(animation)
loc <- reports::folder(animation_gantt)

## Set up the plotting function
oopt <- animation::ani.options(interval = 0.1)

FUN <- function() {
  out <- Animate(ani_gantt)
  lapply(out, function(x) {
    print(x)
    animation::ani.pause()
  })
}

}

type <- if(.Platform$OS.type == "windows") shell else system
saveVideo(FUN(), video.name = "animation.avi", interval = 0.1, outdir = loc)
saveLatex(FUN(), autoplay = TRUE, loop = FALSE, latex.filename = "tester.tex",
    caption = "animated dialogue", outdir = loc, ani.type = "pdf",
    ani.dev = "pdf", ani.width = 5, ani.height = 5.5, interval = 0.1)

saveHTML(FUN(), autoplay = FALSE, loop = TRUE, verbose = FALSE,
    ani.width=600, ani.height=280,
    outdir = file.path(loc, "new"), single.opts =
    "'controls': ['first', 'play', 'loop', 'speed'], 'delayMin': 0")

## End(Not run)
```
Description

Produces start and end times for occurrences for each repeated measure condition.

Usage

gantt_rep(rm.var, text.var, grouping.var = NULL, units = "words",
  col.sep = "_", name.sep = "_")

Arguments

rm.var           An optional single vector or list of 1 or 2 of repeated measures to facet by.
text.var         The text variable.
grouping.var     The grouping variables. Also takes a single grouping variable or a list of 1 or
                 more grouping variables.
units             The unit of measurement to analyze. One of the strings "character", "syllable",
                 "word", or "sentence".
col.sep           The character string to use to separate pasted variables in the pasted columns.
name.sep          The character string to use to separate column names of the pasted columns.

Value

Returns a data frame of start and end times by repeated measure and grouping variable(s)

Note

For non-repeated measures data use gantt. For more flexible plotting needs use gantt_wrap over
the generic plotting method.

References

Clark, W. & Gantt, H. (1922) The Gantt chart, a working tool of management. New York, Ronald
Press.

See Also

gantt, gantt_wrap, gantt_plot

Examples

## Not run:
dat <- with(rajSPLIT, gantt_rep(act, dialogue, list(fam.aff, sex),
  units = "words", col.sep = "_"))
head(dat, 20)
plot(dat)

gantt_wrap(dat, "fam.aff sexe", facet.vars = "act",
  title = "Repeated Measures Gantt Plot",
  minor.line.freq = 25, major.line.freq = 100)
## Two facets variables

```r
dat2 <- with(DATA2, gantt_rep(list(day, class), state, person,
  units = "words", col.sep = ","))
head(dat2, 20)
plot(dat2)
```

## End(Not run)

---

### Description

A `ggplot2` wrapper that produces a Gantt plot.

### Usage

```r
gantt_wrap(dataframe, plot.var, facet.vars = NULL, fill.var = NULL,
  title = NULL, ylab = plot.var, xlab = "duration.default",
  rev.factor = TRUE, transform = FALSE, ncol = NULL,
  minor.line.freq = NULL, major.line.freq = NULL, sig.dig.line.freq = 1,
  hms.scale = NULL, scale = NULL, space = NULL, size = 3,
  rm.horiz.lines = FALSE, x.ticks = TRUE, y.ticks = TRUE,
  legend.position = NULL, bar.color = NULL, border.color = NULL,
  border.size = 2, border.width = 0.1, constrain = TRUE, plot = TRUE)
```

### Arguments

- **dataframe**: A data frame with plotting variable(s) and a column of start and end times.
- **plot.var**: A factor plotting variable (y axis).
- **facet.vars**: An optional single vector or list of 1 or 2 to facet by.
- **fill.var**: An optional variable to fill the code strips by.
- **title**: An optional title for the plot.
- **ylab**: An optional y label.
- **xlab**: An optional x label.
- **rev.factor**: logical. If TRUE reverse the current plotting order so the first element in the plotting variable’s levels is plotted on top.
- **transform**: logical. If TRUE the repeated facets will be transformed from stacked to side by side.
- **ncol**: if an integer value is passed to this `gantt_wrap` uses `facet_wrap` rather than `facet_grid`.
- **minor.line.freq**: A numeric value for frequency of minor grid lines.
- **major.line.freq**: A numeric value for frequency of major grid lines.
sig. dig. line.freq
- An internal rounding factor for minor and major line freq. Generally, default value of 1 suffices for larger range of x scale may need to be set to -2.

hms.scale
- logical. If TRUE converts scale to h:mm:ss format. Default NULL attempts to detect if object is a cm_time2long object.

scale
- Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y")

space
- If "fixed", the default, all panels have the same size. If "free_y" their height will be proportional to the length of the y scale; if "free_x" their width will be proportional to the length of the x scale; or if "free" both height and width will vary. This setting has no effect unless the appropriate scales also vary.

size
- The width of the plot bars.

rm.horiz.lines
- logical. If TRUE the horizontal lines will be removed.

x.ticks
- logical. If TRUE the x ticks will be displayed.

y.ticks
- logical. If TRUE the y ticks will be displayed.

legend.position
- The position of legends. ("left", "right", "bottom", "top", or two-element numeric vector).

bar.color
- Optional color to constrain all bars.

border.color
- The color to plot border around Gantt bars (default is NULL).

border.size
- An integer value for the size to plot borders around Gantt bars. Controls length (width also controlled if not specified).

border.width
- Controls border width around Gantt bars. Use a numeric value in addition to border size if plot borders appear disproportional.

constrain
- logical. If TRUE the Gantt bars touch the edge of the graph.

plot
- logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knitr, sweave, etc. to add additional plot layers.

Value
- Returns a Gantt style visualization. Invisibly returns the ggplot2 list object.

Note
- For non-repeated measures data/plotting use gantt; for repeated measures data output use gantt_rep; and for a convenient wrapper that takes text and generates plots use gantt_plot.

Author(s)
- Andrie de Vries and Tyler Rinker <tyler.rinker@gmail.com>.

References
See Also

gantt, gantt_plot, gantt_rep, facet_grid, facet_wrap

Examples

## Not run:

dat <- gantt(mrjaja$dialogue, list(mrjaja$fam.aff, mrjaja$sex),
  units = "sentences", col.sep = "_")
htruncdf(dat)
gantt_wrap(dat, "fam.aff_sex", title = "Gantt Plot")
dat$codes <- sample(LETTERS[1:3], nrow(dat), TRUE)
gantt_wrap(dat, "fam.aff_sex", fill.var = "codes",
  legend.position = "bottom")

dat2 <- with(rajSPLIT, gantt_rep(act, dialogue,
  list(fam.aff, sex), units = "words", col.sep = "_"))
htruncdf(dat2)
x <- gantt_wrap(dat2, "fam.aff_sex", facet.vars = "act",
  title = "Repeated Measures Gantt Plot")

library(ggplot2); library(scales); library(RColorBrewer)
x + scale_color_manual(values=rep("black",
  length(levels(dat2$fam.aff.sex)))))

## End(Not run)

---

class_gradient_cloud Gradient Word Cloud

Description

Produces a gradient word cloud colored by a binary grouping variable.

Usage

gradient_cloud(text.var, bigroup.var, rev.binary = FALSE, X = "red",
  Y = "blue", stem = FALSE, stopwords = NULL, caps = TRUE,
  caps.list = NULL, I.list = TRUE, random.order = FALSE, rot.per = 0,
  min.freq = 1, max.word.size = NULL, min.word.size = 0.5, breaks = 10,
  cloud.font = NULL, title = NULL, title.font = NULL,
  title.color = "black", title.padj = 0.25, title.location = 3,
  title.cex = NULL, legend.cex = 0.8, legend.location = c(0.025, 0.025,
  0.25, 0.04), char2space = "~~")

Arguments

text.var The text variable.
bigroup.var A binary grouping variable.
gradient_cloud

rev.binary logical. If TRUE the ordering of the binary levels of bigroup.var is reversed.
X The first gradient color for variable X.
Y The second gradient color for variable Y.
stem logical. If TRUE the text.var will be stemmed.
stopwords Words to exclude from the cloud. Words will be removed after determining proportional word usage.
caps logical. If TRUE selected words will be capitalized.
caps.list A vector of words to capitalize (caps must be TRUE).
I.list logical. If TRUE capitalizes I words and contractions.
random.order Plot words in random order. If FALSE, they will be plotted in decreasing frequency.
rot.per Proportion words with 90 degree rotation.
min.freq An integer value indicating the minimum frequency a word must appear to be included.
max.word.size A size argument to control the minimum size of the words.
min.word.size A size argument to control the maximum size of the words.
breaks An integer describing the number of breaks (odd numbers will be rounded up).
cloud.font The font family of the cloud text.
title A character string used as the plot title.
title.font The font family of the cloud title.
title.color A character vector of length one corresponding to the color of the title.
title.padj Adjustment for the title. For strings parallel to the axes, padj = 0 means right or top alignment, and padj = 1 means left or bottom alignment.
title.location On which side of the plot (1=bottom, 2=left, 3=top, 4=right).
title.cex Character expansion factor for the title. NULL and NA are equivalent to 1.0.
legend.cex Character expansion factor for the legend. NULL and NA are equivalent to 1.0.
legend.location A vector of length 4 denoting the lower left (x and y left) and upper right (x and y right) coordinates of the rectangle of colors in user coordinates.
char2space A vector of characters to be turned into spaces.

Details

Breaking is done using quantile. This will ensure a certain percentage of words will be colored at each bin.

Value

Plots a gradient word cloud and invisibly returns the dataframe used to make the cloud.

See Also

trans_cloud, wordcloud, color.legend
hamlet

Examples

```r
## Not run:
DATA$state <- space_fill(DATA$state, c("is fun", "too fun", "you liar"))

gradient_cloud(DATA$state, DATA$sex, title="fun")
gradient_cloud(DATA$state, DATA$sex, title="fun", rev.binary = TRUE)
gradient_cloud(DATA$state, DATA$sex, title="fun", max.word.size = 5,
  min.word.size = .025)

with(mrajaL, gradient_cloud(dialogue, died, stopwords = Top25Words,
  rot.per = .5, title="Heatcloud", title.color="orange",
  title.cex=1.75))
x <- with(subset(mrajaL, fam.aff %in% qcv(cap, mont)),
  gradient_cloud(dialogue, fam.aff))
head(x)

## 2012 U.S. Presidential Debates
invisible(lapply(split(pres.debates2012, pres.debates2012$time), function(x) {
  x <- x[x$person %in% qcv(ROMNEY, OBAMA), ]
  dev.new()
  gradient_cloud(x$dialogue, x$person,
    title = paste("Debate", char2end(x$time[1]))),
  stopwords = BuckleySaltonSWL,
  X = "blue", Y = "red",
  max.word.size = 2.2,
  min.word.size = 0.55
})
```

## End(Not run)

---

**hamlet**  
*Hamlet (Complete & Split by Sentence)*

**Description**

A dataset containing the complete dialogue of Hamlet with turns of talk split into sentences.

**Usage**

```r
data(hamlet)
```

**Format**

A data frame with 2007 rows and 7 variables
Details

- act. The act (akin to repeated measures)
- tot. The turn of talk
- scene. The scene (nested within an act)
- location. Location of the scene
- person. Character in the play
- died. Logical coded death variable if yes the character dies in the play
- dialogue. The spoken dialogue

References

http://www.gutenberg.org

Description

**htruncdf** - Convenience function to view the head of a truncated dataframe.
**truncdf** - Convenience function to view a truncated dataframe.
**ltruncdf** - Convenience function to view the head of a list of truncated dataframes.
**qview** - Convenience function to view a summary and head of a dataframe.
**lview** - Convenience function to view the list (list view) of qdap objects that have print methods that print a single dataframe.

Usage

- `htruncdf(dataframe, n = 10, width = 10, ...)`
- `truncdf(dataframe, end = 10, begin = 1)`
- `ltruncdf(dat.list, n = 6, width = 10, ...)`
- `qview(dataframe, ...)`
- `lview(x, print = TRUE)`
Arguments

dataframe  A data.frame object.
n  Number of rows to display.
width  The width of the columns to be displayed.
end  The last character to be displayed (width).
begin  The first character to be displayed (width).
datNlist  A list of data.frame objects.
x  A class qdap object that is a list which prints as a dataframe.
print  logical. If TRUE prints to the console.
...  Other arguments passed to htruncdf (qview; ltruncdf) or head (htruncdf).

Value

htrundf - returns n number of rows of a truncated dataframe.
trundf - returns a truncated dataframe.
ltruncdf - returns a list of n number of rows of a truncated dataframes.
qview - returns a dataframe head with summary statistics.
lview - prints a list of the qdap object and invisibly returns the unclassed object.

See Also

head

Examples

## Not run:
truncdf(raj[1:10, ])
truncdf(raj[1:10, ], 40)
htruncdf(raj)
htruncdf(raj, 28)
htruncdf(raj, ,28)
ltruncdf(rajPOS, width = 4)
qview(raj)
qview(CO2)
lview(question_type(DATA.SPLIT$state, DATA.SPLIT$person))
lview(rajPOS)
lview(lm(mpg~hp, data = mtcars))

## End(Not run)
**imperative**

*Intuitively Remark Sentences as Imperative*

**Description**

Automatic imperative remarking.

**Usage**

```r
imperative(dataframe, person.var, text.var, lock.incomplete = FALSE,
additional.names = NULL, parallel = FALSE, warning = FALSE)
```

**Arguments**

- `dataframe` A data.frame object.
- `person.var` The person variable.
- `text.var` The text variable.
- `lock.incomplete` logical. If TRUE locks incomplete sentences (sentences ending with "!") from being marked as imperative.
- `additional.names` Additional names that may be used in a command (people in the context that do not speak).
- `parallel` logical. If TRUE attempts to run the function on multiple cores. Note that this may not mean a speed boost if you have one core or if the data set is smaller as the cluster takes time to create. With the mraja1sp1 data set, with an 8 core machine, imperative had 1/3 the running time.
- `warning` logical. If TRUE provides comma warnings (sentences that contain numerous commas that may be handled incorrectly by the algorithm).

**Value**

Returns a dataframe with a text variable indicating imperative sentences. Imperative sentences are marked with * followed by the original end mark.

**Warning**

The algorithm used by imperative is sensitive to English language dialects and types. Commas can indicate a choppy sentence and may indicate a false positive. Sentences marked with ‘AAVE’ may be the use of African American Vernacular English and not an imperative sentence.
Examples

```r
## Not run:
dat <- data.frame(name=c("sue", rep(c("greg", "tyler", "phil", "sue"), 2)), statement=c("go get it!", "I hate to read.", "Stop running!", "I like it!", "You are terrible!", "Don't!", "Greg, go to the red, brick office.", "Tyler go to the gym.", "Alex don't run."), stringsAsFactors = FALSE)

imperative(dat, "name", "statement", c("Alex"))
imperative(dat, "name", "statement", lock.incomplete = TRUE, c("Alex"))
imperative(dat, "name", "statement", c("Alex"), warning=TRUE)
imperative(dat, "name", "statement", c("Alex"), warning=TRUE, parallel = TRUE)

## End(Not run)
```

---

**incomplete_replace**  
*Denote Incomplete End Marks With "|"*

**Description**
Replaces incomplete sentence end marks (..., .?, ..?, en & em dash etc.) with "|".

**Usage**

```
incomplete_replace(text.var, scan.mode = FALSE)

incomp(text.var, scan.mode = FALSE)
```

**Arguments**
- `text.var` The text variable.
- `scan.mode` logical. If TRUE only scans and reports incomplete sentences.

**Value**
Returns a text variable (character string) with incomplete sentence marks (..., .?, ..?, en & em dash etc.) replaced with "|". If scan mode is TRUE returns a data frame with incomplete sentence location.

**Examples**

```r
## Not run:
x <- c("the...", "I.?", "you.", "threw..", "we?")
incomplete_replace(x)
incomp(x)
incomp(x, scan.mode = TRUE)

## End(Not run)
```
**inspect_text**

**Inspect Text Vectors**

**Description**

`inspect_text` - Inspect a text vector with adjustable string wrapping; created a pretty printed named list.

`inspect_text.default` - Default method for `inspect_text` used to convert to a vector to a pretty printed named list.

`inspect_text.Corporus` - Corpus method for `inspect_text` used to convert to a Corpus.

**Usage**

```
inspect_text(text.var, grouping.var = NULL, ...)
```

## Default S3 method:

```
inspect_text(text.var, grouping.var = NULL, ...)
```

## S3 method for class 'Corpus'

```
inspect_text(text.var, ...)
```

**Arguments**

- `text.var`  
  The text variable or a `wfm` object.

- `grouping.var`  
  The grouping variables. Default `NULL` generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.

- `...`  
  Ignored.

**Value**

Returns a named list (prints pretty).

**Examples**

```
# Not run:
with(raj, inspect_text(dialogue))
with(raj, inspect_text(dialogue, person))
with(raj, inspect_text(dialogue, list(paste("Act", act), person)))

# With a tm Corpus object
library(tm)
data(crude)
inspect_text(crude)

# End(Not run)
```
is.global

Test If Environment is Global

Description

A logical test to determine if the current environment is the global environment.

Usage

is.global(n = 1)

Arguments

n The number of generations to go back. If used as a function argument n should be set to 2.

Value

A logical response.

Author(s)

Simon O’Hanlon and Tyler Rinker <tyler.rinker@gmail.com>

References


See Also

globalenv, parent.frame

Examples

is.global()
lapply(1:3, function(i) is.global())
FUN <- function() is.global(); FUN()

FUN2 <- function(x = is.global(2)) x
FUN2()
FUN3 <- function() FUN2(); FUN3()
**key_merge**

**Merge Demographic Information with Person/Text Transcript**

**Description**

Wrapper function *(merge)* for merging demographic information with a person/text transcript.

**Usage**

```r
key_merge(transcript.df, key.df, common.column = NULL,
          defualt.arrange = TRUE)
```

**Arguments**

- `transcript.df`: The text/person transcript dataframe
- `key.df`: The demographic dataframe.
- `common.column`: The column(s) shared by `transcript.df` and `key.df`. If NULL function defaults to use any columns with the same name.
- `default.arrange`: logical. If TRUE will arrange the columns with text to the far right.

**Value**

Outputs a merged transcript dataframe with demographic information.

**See Also**

*merge*

**Examples**

```r
## Not run:
#First view transcript dataframe and demographics dataframe.
ltruncdf(list(raj, raj.demographics), 10, 50)
merged.raj <- key_merge(raj, raj.demographics)
htruncdf(merged.raj, 10, 40)

## End(Not run)
```
kullback_leibler  Kullback Leibler Statistic

Description

A proximity measure between two probability distributions applied to speech.

Usage

kullback_leibler(x, y = NULL)

Arguments

x
A numeric vector, matrix or data frame.
y
A second numeric vector if x is also a vector. Default is NULL.

Details

Uses Kullback & Leibler’s (1951) formula:

\[ D_{KL}(P||Q) = \sum_i \ln \left( \frac{P_i}{Q_i} \right) P_i \]

Value

Returns a matrix of the Kullback Leibler measure between each vector of probabilities.

Note

The kullback_leibler function generally receives the output of either wfm or wfdf functions.

References


Examples

```r
## Not run:
p.df <- wfdf(DATA$state, DATA$person)
p.mat <- wfm(text.var = DATA$state, grouping.var = DATA$person)
kullback_leibler(p.mat)
(x <- kullback_leibler(p.df))
print(x, digits = 5)
kullback_leibler(p.df$greg, p.df$sam)

## p.df2 <- wfdf(raj$dialogue, raj$person)
## x <- kullback_leibler(p.df2)

## End(Not run)
```
left_just | Text Justification

Description

left_just - Left justifies a text/character column.
right_just - A means of undoing a left justification.

Usage

left_just(dataframe, column = NULL, keep.class = FALSE)
right_just(dataframe)

Arguments

dataframe  A data.frame object with the text column.
column     The column to be justified. If NULL all columns are justified.
keep.class logical. If TRUE will attempt to keep the original classes of the dataframe if the justification is not altered (i.e., numeric will not be honored but factor may be).

Value

Returns a dataframe with selected text column left/right justified.

Note

left_just inserts spaces to achieve the justification. This could interfere with analysis and therefore the output from left_just should only be used for visualization purposes, not analysis.

Examples

```r
## Not run:
left_just(DATA)
left_just(DATA, "state")
left_just(CO2[1:15,])
right_just(left_just(CO2[1:15,]))
## End(Not run)
```
**Description**

Transcript apply lexical classification score (content to functional word proportion) by grouping variable(s) and optionally plot the breakdown of the model.

**Usage**

```r
lexical_classification(text.var, grouping.var = NULL,
order.by.lexical_classification = TRUE,
function.words = qdapDictionaries::function.words, bracket = "all", ...)
```

**Arguments**

- `text.var` The text variable.
- `grouping.var` The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `order.by.lexical_classification` logical. If TRUE orders the results by # lexical_classification score.
- `function.words` A vector of function words. Default is `function.words`.
- `bracket` The bracket type to remove. Use NULL to not remove bracketed substrings. See bracket argument in `bracketX` for bracket types.
- `...` Other arguments passed to `bracketX`.

**Details**

Content words (i.e., nouns, verbs, adjectives, and adverbs) tend to be the words speakers stresses in language use. Whereas, functional words are the "glue" that holds the content together. Speakers devote much less time and stress to these words (i.e., pronouns, articles, conjunctions, quantifiers, and prepositions).

**Value**

A list containing at the following components:

- `content` A data.frame of all content words used and corresponding frequencies
- `functional` A data.frame of all content words used and corresponding frequencies
- `raw` Sentence level descriptive statistics on content vs. functional word use (ave.content.rate is also nown as lexical density
- `lexical_classification` Summarized (grouping variable level) descriptive statistics for content vs. functional word use
References


Examples

```r
## Not run:
lexical_classification("I did not like the dog.")
lexical_classification(DATA.SPLIT$state, DATA.SPLIT$person)

(out <- with(pres.debates2012, lexical_classification(dialogue, list(person, time))))
plot(out)

scores(out)

out2 <- preprocessed(out)
htaucdf(out2)
plot(out2)

plot(out[["content"]])
dev.new()
plot(out[["functional"]])

## cloud of functional vs. content
## Highlight Content Words
set.seed(10)
par(mar = c(0,0,0,0))
list(
  content = out[["content"]],
  functional = out[["Functional"]]
) %>%
list_df2df("type") %>%
dplyr::mutate(colors = ifelse(type == "functional", "gray80", "blue")) %>%
with(. , wordcloud::wordcloud(
  word, 
  freq, 
  min.freq = 8, 
  random.order=FALSE, 
  ordered.colors = TRUE, 
  colors = colors
))
legend(
```
.05, .12, bty = "n",
legend = c("functional", "content"),
fill = c("gray80", "blue"),

cex = .7
)

## Highlight Functional Words
set.seed(10)
par(mar = c(0,0,0,0))
list(
  content = out[["content"]],
  functional = out[["functional"]]
)

list_df2df("type")

dplyr::mutate(colors = ifelse(type == "functional", "red", "gray80"))

with(.[], wordcloud::wordcloud(
  word,
  freq,
  min.freq = 8,
  random.order=FALSE,
  ordered.colors = TRUE,
  colors = colors
))


legend(
  .05, .12, bty = "n",
  legend = c("functional", "content"),
  fill = c("red", "gray80"),
  cex = .7
)

# ================
## ANIMATION ##
# ================

## EXAMPLE 1
lex_ani <- lexical_classification(DATA_SPLIT$state, DATA_SPLIT$person)
lexa <- Animate(lex_ani, content="white", functional="blue",
  current.color = "yellow", current.speaker.color="grey70")

bgb <- vertex_apply(lexa, label.color="grey80", size=20, color="grey40")
bgb <- edge_apply(bgb, label.color="yellow")

print(bgb, bg="black", net.legend.color ="white", pause=1)

## EXAMPLE 2
lex_ani2 <- lexical_classification(mraja1spl$dialogue, mraja1spl$person)
lexa2 <- Animate(lex_ani2, content="white", functional="blue",
  current.color = "yellow", current.speaker.color="grey70")

bgb2 <- vertex_apply(lexa2, label.color="grey80", size=17, color="grey40")
bgb2 <- edge_apply(bgb2, label.color="yellow")

print(bgb2, bg="black", pause=.75, net.legend.color = "white")
## EXAMPLE 3 (bar plot)
Animate(lex_anii2, type="bar")

## EXAMPLE 4 (text plot)
Animate(lex_anii2, type="text")

# Complex Animations #
# EXAMPLE 1: Network + Text + Bar

library(animation)
library(grid)
library(gridBase)
library(qdap)
library(reports)
library(igraph)
library(plotrix)

lex_anii2 <- lexical_classification(mraja1s1$dialogue, mraja1s1$person)

## Set up the network version
lex_net <- Animate(lex_anii2, contextual="white", lexal="blue",
current.color = "yellow", current.speaker.color="grey70")
bgb <- vertex_apply(lex_net, label.color="grey80", size=17, color="grey40")
bgb <- edge_apply(bgb, label.color="yellow")

## Set up the bar version
lex_bar <- Animate(lex_anii2, type="bar")

## Set up the text
lex_text <- Animate(lex_anii2, type="text", size = 3, width=125, color="white")

## Generate a folder
loc <- reports::folder(animation_lexical_classification)
setwd(loc)

## Set up the plotting function
oopt <- animation::ani.options(interval = 0.1)

lex_text_bar <- Map(function(x, y){
  uns <- unit(c(-1.6,.5,-.2,.25), "cm")
  x <- x +
  theme(plot.margin = uns,
    text = element_text(color="white"),
    legend.text = element_text(color="white"),
    legend.background = element_rect(fill = "black"),
    panel.border = element_rect(color = "black"),
    panel.background = element_rect(fill = "black"),
})
plot.background = element_rect(fill = "black",
   color = "black")

uns2 <- unit(c(-.5,.5,-.45,.25), "cm")

y <- y +
  theme(plot.margin = uns2,
        text = element_text(color = "white"),
        legend.text = element_text(color = "white"),
        legend.background = element_rect(fill = "black"),
        plot.background = element_rect(fill = "black",
            color = "black")

gA <- ggplotGrob(x)
gB <- ggplotGrob(y)
maxWidth <- grid::unit.pmax(gA$widths[2:5], gB$widths[2:5])
gA$widths[2:5] <- as.list(maxWidth)
gB$widths[2:5] <- as.list(maxWidth)
out <- arrangeGrob(gA, gB, ncol = 1, heights = grid::unit(c(.3, .7), "native"))
## grid.draw(out)
invisible(out)

}, lex_text, lex_bar)

FUN <- function(follow = FALSE, theseq = seq_along(bgb)) {

  Title <- "Animated Content Rate: Romeo and Juliet Act 1"
  Legend <- c(.2, -.1, 1.5, -.95)
  Legend.cex <- 1

  lapply(theseq, function(i) {
    if (follow) {
      png(file = sprintf("%s/images/Rplot%s.png", loc, i),
           width = 750, height = 875)
    }
    ## Set up the layout
    layout(matrix(c(rep(1, 7), rep(2, 6)), 13, 1, byrow = TRUE))

    ## Plot 1
    par(mar = c(2, 0, 2, 0), bg = "black")
    #par(mar = c(2, 0, 2, 0))
    set.seed(22)
    plot.igraph(bgb[[i]], edge.curved = TRUE)
    mtext(Title, side = 3, col = "white")
    color.legend(Legend[1], Legend[2], Legend[3], Legend[4],
                 c("Functional", "Content"), attributes(bgb)[["legend"]],
                 cex = Legend.cex, col = "white")

    ## Plot 2
    plot.new()
    vps <- baseViewports()
  })
print(lex_text_bar[[i]], vp = vpStack(vps$figure,vps$plot))

animation::ani.pause()

if (follow) {
  dev.off()
}
}
FUN()

## Detect OS

fun <- if(.Platform$OS.type == "windows") shell else system

saveHTML(FUN(), autoplay = FALSE, loop = TRUE, verbose = FALSE,
          ani.height = 1000, ani.width=750,
          outdir = loc, single.opts =
          "'controls': ['first', 'previous', 'play', 'next', 'last', 'loop', 'speed'], 'delayMin': 0")

FUN(TRUE)

## EXAMPLE 2: Line + Text + Bar

## Generate a folder

loc2 <- reports::folder(animation_lexical_classification2)
setwd(loc2)

lex ani2 <- lexical_classification(mrja1spl$dialogue, mrja1spl$person)

## Set up the bar version

lex_bar <- Animate(lex ani2, type="bar")
cumline <- cumulative(lex_bar)
lex_line <- plot(cumline)
ylims <- range(cumline[[1]][-c(1:100)]) + c(-1, 1)

## Set up the text

lex_text <- Animate(lex ani2, type="text", size = 4, width = 80)

lex_line_text_bar <- Map(function(x, y, z){
  mar <- theme(plot.margin = unit(c(0, .5, 0, .25), "cm"))

  gA <- ggplotGrob(x + mar +
     theme(panel.background = element_rect(fill = NA, colour = NA),
           panel.border = element_rect(fill = NA, colour = NA),
           plot.background = element_rect(fill = NA, colour = NA)))
  gB <- ggplotGrob(y + mar)
  gC <- ggplotGrob(z + mar + ylab("Average Content Rate") +
                   coord_cartesian(ylim = ylims) +
                   ggtitle("Average Content Rate: Romeo & Juliet Act 1"))

  gA
})
maxWidth <- grid::unit.pmax(gA$widths[2:5], gB$widths[2:5], gC$widths[2:5])
gA$widths[2:5] <- as.list(maxWidth)
gB$widths[2:5] <- as.list(maxWidth)
gC$widths[2:5] <- as.list(maxWidth)
out <- arrangeGrob(gC, gA, gB, ncol=1, heights = grid::unit(c(.38, .25, .37), "native"))
## grid.draw(out)
invisible(out)
}, lex_text, lex_bar, lex_line)

FUN2 <- function(follow=FALSE, theseq = seq_along(lex_line_text_bar)) {

  lapply(theseq, function(i) {
    if (follow) {
      png(file=sprintf("%s/images/Rplot%s.png", loc2, i),
           width=750, height=875)
    }
    print(lex_line_text_bar[i])
    animation::ani.pause()

    if (follow) {
      dev.off()
    }
  })
}

FUN2()

## Detect OS
type <- if (.Platform$OS.type == "windows") shell else system

library(animation)
saveHTML(FUN2(), autoplay = FALSE, loop = TRUE, verbose = FALSE,
         ani.height = 1000, ani.width=750,
         outdir = loc2, single.opts =
             "'controls': ['first', 'previous', 'play', 'next', 'last', 'loop', 'speed'], 'delayMin': 0")

FUN2(TRUE)

##-------------------
## Static Network ##
##-------------------

(lexdat <- with(sentSplit(DATA, 4), lexical_classification(state, person)))
m <- Network(lexdat)
m
print(m, bg="grey97", vertex.color="grey75")

print(m, title="Lexical Content Discourse Map", title.color="white",


Read/Write Multiple csv Files at a Time

**mcsv_r** - Read and assign multiple csv files at the same time.

**mcsv_w** - Write multiple csv files into a file at the same time.

**Usage**

```r
mcsv_r(files, a.names = NULL, l.name = NULL, list = TRUE, pos = 1,
       envir = as.environment(pos))

mcsv_w(..., dir = NULL, open = FALSE, sep = "", dataframes = NULL,
       pos = 1, envir = as.environment(pos))
```
mcsv_r

Arguments
files csv file(s) to read.
a.names object names to assign the csv file(s) to. If NULL assigns the name(s) of the csv files in the directory, without the file extension, to the objects in the global environment.
l.name A single character string of a name to assign to the list if dataframes created by the csv files being read in. Default (NULL) uses L1.
list logical. If TRUE then a list of dataframes is created in the global environment in addition to the individual dataframes.
pos where to do the removal. By default, uses the current environment.
envir the environment to use.
... data.frame object(s) to write to a file or a list of data.frame objects. If the objects in a list are unnamed V + digit will be assigned. Lists of dataframes (e.g., the output from termco or polarity) can be passed as well.
dir optional directory names. If NULL a directory will be created in the working directory with the data and time stamp as the folder name.
open logical. If TRUE opens the directory upon completion.
sep A character string to separate the terms.
dataframes An optional character vector of dataframes in lieu of ... argument.

details
mcsv is short for "multiple csv" and the suffix c(_r, _w) stands for "read" (r) or "write" (w).

Value
mcsv_r - reads in multiple csv files at once.
mcsv_w - creates a directory with multiple csv files. Silently returns the path of the directory.

Note
mcsv_r is useful for reading in multiple csv files from cm_df.temp for interaction with cm_range2long.

See Also
cm_range2long, cm_df.temp, condense, assign

Examples
## Not run:
## mcsv_r EXAMPLE:
mtcarsb <- mtcars[1:5, ]; CO2b <- CO2[1:5, ]
(a <- mcsv_w(mtcarsb, CO2b, dir="foo"))
rm("mtcarsb", "CO2b") # gone from .GlobalEnv
(nms <- dir(a))
mcsv_r(file.path(a, nms))
mtcarsb; CO2b
rm("mtcarsb", "CO2b")  # gone from .GlobalEnv
mcsv_r(file.path(a, nms), paste0("foo.dat", 1:2))
foo.dat1; foo.dat2
rm("foo.dat1", "foo.dat2")  # gone from .GlobalEnv
library(reports); delete("Foo")

## mcsv_w EXAMPLES:
(a <- mcsv_w(mtcars, CO2, dir="foo"))
delete("foo")

## Write lists of dataframes as well
poldat <- with(DATA.SPLIT, polarity(state, person))
term <- c("the ", "she ", " who")
termdat <- with (raj.act.1, termco (dialogue, person, term))
mcsv_w(poldat, termdat, mtcars, CO2, dir="foo2")
delete("foo2")

## End(Not run)

---

**Romeo and Juliet: Act 1 Dialogue Merged with Demographics**

**Description**

A dataset containing act 1 of Romeo and Juliet with demographic information.

**Usage**

```r
data(mraja1)
```

**Format**

A data frame with 235 rows and 5 variables

**Details**

- person. Character in the play
- sex. Gender
- fam.aff. Family affiliation of character
- died. Dummy coded death variable (0-no; 1-yes); if yes the character dies in the play
- dialogue. The spoken dialogue

**References**

A dataset containing act 1 of Romeo and Juliet with demographic information and turns of talk split into sentences.

Usage

data(mraja1spl)

Format

A data frame with 508 rows and 7 variables

Details

- person. Character in the play
- tot.
- sex. Gender
- fam.aff. Family affiliation of character
- died. Dummy coded death variable (0-no; 1-yes); if yes the character dies in the play
- dialogue. The spoken dialogue
- stem.text.

References

http://shakespeare.mit.edu/romeo_juliet/full.html

Description

multigsub - A wrapper for gsub that takes a vector of search terms and a vector or single value of replacements.

sub_holder - This function holds the place for particular character values, allowing the user to manipulate the vector and then revert the place holders back to the original values.
Usage

multigsub(pattern, replacement, text.var, leadspace = FALSE,
          trailspace = FALSE, fixed = TRUE, trim = TRUE, order.pattern = fixed,
          ...
)

mgsub(pattern, replacement, text.var, leadspace = FALSE, trailspace = FALSE,
       fixed = TRUE, trim = TRUE, order.pattern = fixed, ...)

sub_holder(pattern, text.var, alpha.type = TRUE, ...)

Arguments

pattern Character string to be matched in the given character vector.
replacement Character string equal in length to pattern or of length one which are a replace-
            ment for matched pattern.
text.var The text variable.
leadspace logical. If TRUE inserts a leading space in the replacements.
trailspace logical. If TRUE inserts a trailing space in the replacements.
fixed logical. If TRUE, pattern is a string to be matched as is. Overrides all conflicting
        arguments.
trim logical. If TRUE leading and trailing white spaces are removed and multiple
        white spaces are reduced to a single white space.
order.pattern logical. If TRUE and fixed = TRUE, the pattern string is sorted by number of
        characters to prevent substrings replacing meta strings (e.g., pattern = c("the", "then")
        resorts to search for "then" first).
... Additional arguments passed to gsub.
alpha.type logical. If TRUE alpha (lower case letters) are used for the key. If FALSE numbers
        are used as the key.

Value

multigsub - Returns a vector with the pattern replaced.
sub_holder - Returns a list with the following:
output keyed place holder character vector
unhold A function used to revert back to the original values

Note

The unhold function for sub_holder will only work on keys that have not been disturbed by sub-
sequent alterations. The key follows the pattern of ‘qdapholder’ followed by lower case letter
keys followed by ‘qdap’.

See Also

gsub
Examples

```r
## Not run:
## ---------------------------------
## `mgsb` Function
## ---------------------------------

multigsub(c("it's", "I'm"), c("it is", "I am"), DATA$state)
mgsb(c("it's", "I'm"), c("it is", "I am"), DATA$state)
mgsb("[[:punct:]]", "PUNC", DATA$state, fixed = FALSE)
```

```r
## ---------------------------------
## `sub_holder` Function
## ---------------------------------

## `alpha.type` as TRUE
(fake_dat <- paste(emoticon[1:11,2], DATA$state))
(m <- sub_holder(emoticon[,2], fake_dat))
m$unhold(strip(m$output))
# With Stemming
m$unhold(stemmer(strip(m$output)), capitalize = FALSE)

## `alpha.type` as FALSE (numeric keys)
vowels <- LETTERS[c(1, 5, 9, 15, 21)]
(m2 <- sub_holder(vowels, toupper(DATA$state), alpha.type = FALSE))
m2$unhold(gsub("[0-9]", "", m2$output))
mtabulate(strsplit(m2$unhold(gsub("[0-9]", "", m2$output)), ""))

## End(Not run)
```

---

## multiscale

### Nested Standardization

**Description**

Standardize within a subgroup and then within a group.

**Usage**

```r
multiscale(numeric.var, grouping.var, original_order = TRUE, digits = 2)
```

**Arguments**

- **numeric.var** A numeric variable.
- **grouping.var** The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- **original_order** logical. IF TRUE orders by the original order. If FALSE orders by group.
- **digits** Integer; number of decimal places to round.
Value

Returns a list of two:

SCALED_OBSERVATIONS
A dataframe of scaled observations at level one and two of the nesting with possible outliers.

DESCRIPTIVES_BY_GROUP
A data frame of descriptives by group.

See Also

scale

Examples

```r
## Not run:
dat <- with(mraja1sp1, word_stats(dialogue, list(person, sex, fam.aff)))
htunrclr(colspit2t(fig(dat$ts, ,4)
out1 <- with(colspit2df(dat$ts), multiscale(word.count, person))
htunrclr(out1, 10)
out2 <- with(colspit2df(dat$ts), multiscale(word.count,
    list(fam.aff, sex)))
htunrclr(out2, 10)
out3 <- with(colspit2df(dat$ts), multiscale(word.count,
    list(fam.aff, sex), original.order = FALSE))
htunrclr(out3, 10)

## End(Not run)
```

---

### NAer

**Replace Missing Values (NA)**

**Description**

Replace missing values (NA) in a vector or dataframe.

**Usage**

```r
NAer(x, replace = 0)
```

**Arguments**

- `x` A vector or dataframe with missing values (NA).
- `replace` The value to replace missing values (NA) with.

**Value**

Returns a vector or dataframe with missing values replaced.
**name2sex**

## Names to Gender

A wrapper for the `gender` function used to predict gender based on first name.

### Usage

```r
name2sex(names.list, USE.NAMES = FALSE, ...)
```

### Arguments

- **names.list**: Character vector containing first names.
- **USE.NAMES**: logical. If TRUE `names.list` is used to name the gender vector.
- **...**: Other arguments passed to `gender`.

### Value

Returns a vector of predicted gender (M/F) based on first name.

### See Also

`gender`

### Examples

```r
## Not run:
name2sex(qcv(mary, jenn, linda, JAME, GABRIEL, OLIVA,
       tyler, jamie, JAMES, tyrone, cheryl, drew))
```

## End(Not run)
Network

Generic Network Method

Description
Create a network plot for select qdap outputs.

Usage
Network(x, ...)

Arguments
- x: A select qdap object.
- ...: Arguments passed to Network method of other classes.

Value
Returns a network plot.

Network.formality

Network Formality

Description
Network.formality - Network a formality object.

Usage

## S3 method for class 'formality'
Network(x, contextual = "yellow", formal = "red",
edge.constant, title = NULL, digits = 3, plus.300.color = "grey40",
der.under.300.color = "grey88", missing.color = "purple", ...)

Arguments
- x: A formality object.
- contextual: The color to use for 0% formality (purely contextual).
- formal: The color to use for 100% formality (purely formal).
- edge.constant: A constant to multiple edge width by.
- title: The title to apply to the Networked image(s).
- digits: The number of digits to use in the current turn of talk formality.
- plus.300.color: The bar color to use for grouping variables exceeding 299 words per Heylighen & Dewaele’s (2002) minimum word recommendations.
under.300.color
The bar color to use for grouping variables less than 300 words per Heylighen & Dewaele’s (2002) minimum word recommendations.

missing.color
The color to use in a network plot for edges corresponding to missing text data. Use na.omit before hand to remove the missing values all together.

Other arguments passed to discourse_map.

Details

formality Method for Network

Network.lexical_classification

Description

Network.lexical_classification - Network a lexical_classification object.

Usage

## S3 method for class 'lexical_classification'
Network(x, functional = "yellow",
  content = "red", edge.constant, title = NULL, digits = 2, ...)

Arguments

x
A lexical_classification object.

functional
The color to use for 0% lexical_classification (purely functional).

content
The color to use for 100% lexical_classification (purely content).

digits
The number of digits to use in the current turn of talk lexical_classification.

Other arguments passed to discourse_map.

Details

lexical_classification Method for Network
Network.polarity  

*Network Polarity*

### Description

Network.polarity - Network a `polarity` object.

### Usage

```r
## S3 method for class 'polarity'
Network(x, negative = "blue", positive = "red",
        neutral = "yellow", edge.constant, title = NULL, digits = 3, ...)
```

### Arguments

- **x**: A `polarity` object.
- **negative**: The color to use for negative polarity.
- **positive**: The color to use for positive polarity.
- **neutral**: The color to use for neutral polarity.
- **edge.constant**: A constant to multiple edge width by.
- **title**: The title to apply to the Networked image(s).
- **digits**: The number of digits to use in the current turn of talk polarity.
- **...**: Other arguments passed to `discourse_map`.

### Details

- **polarity** Method for Network

---

new_project  

*Project Template*

### Description

Generate a project template to increase efficiency.

### Usage

```r
new_project(project = "new", path = getwd(), open = is.global(2),
             github = FALSE, ...)
```
new_project

Arguments

project A character vector of the project name.
path The path to where the project should be created. Default is the current working directory.
open logical. If TRUE the project will be opened in RStudio. The default is to test if new_project is being used in the global environment, if it is then the project directory will be opened.
github logical. If TRUE the repo will be sent to public GitHub account.
... Other arguments passed to new_report.

Details

The project template includes these main directories and scripts:

- **CODEBOOK** - A directory to store coding conventions or demographics data:
  - KEY.csv - A blank template for demographic information

- **CORRESPONDENCE** - A directory to store correspondence and agreements with the client:
  - CONTACT_INFO.txt - A text file to put research team members’ contact information

- **DATA** - A directory to store data:
  - CLEANED_TRANSCRIPTS - A directory to store the cleaned transcripts (If the transcripts are already cleaned you may choose to not utilize the RAW_TRANSCRIPTS directory)
  - CM_DATA - A directory to export/import scripts for cm_xxx family of functions
  - DATA_FOR_REVIEW - A directory to put data that may need to be altered or needs to be inspected more closely
  - RAW_DATA - A directory to store non-transcript data related to the project:
    * ANALYTIC_MEMOS - A directory to put audio files (or shortcuts)
    * AUDIO - A directory to put audio files (or shortcuts)
    * FIELD_NOTES - A directory to put audio files (or shortcuts)
    * PAPER_ARTIFACTS - A directory to put paper artifacts
    * PHOTOGRAPHS - A directory to put photographs
    * VIDEO - A directory to put video files (or shortcuts)
  - TRANSCRIPTS - A directory to put transcription data:
    * CLEANED_TRANSCRIPTS - A directory to store the cleaned transcripts (If the transcripts are already cleaned you may choose to not utilize the RAW_TRANSCRIPTS directory)
    * RAW_TRANSCRIPTS - A directory to store the raw transcripts

- **DOCUMENTATION** - A directory to store documents related to the project

- **PLOTS** - A directory to store plots

- **REPORTS** - A directory with report and presentation related tools.

- **SCRIPTS** - A directory to store scripts; already contains the following:
  - 01_clean_data.R - initial cleaning of raw transcripts
The template comes with a .Rproj file. This makes operating in RStudio very easy. The file can be kept on the desktop or a git application such as github, bitbucket or dropbox, depending on what the client/research team is comfortable utilizing.

Value

Creates a project template.

---

### ngrams

**Generate ngrams**

**Description**

Transcript apply ngrams.

**Usage**

```r
ngrams(text.var, grouping.var = NULL, n = 2, ...)
```

**Arguments**

- `text.var` The text variable
- `grouping.var` The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `n` The max number of grams calculate
- `...` Further arguments passed to strip function.
ngrams

Value

Returns a list of:

- **raw**: A list of pasted single vectors of the ngrams per row.
- **group**: A list of pasted vectors of ngrams grouped by grouping.var.
- **unlist1**: A list of a single vector of pasted ngrams per grouping.var in the order used.
- **unlist2**: A list of a single vector of pasted ngrams per grouping.var in alphabetical order.
- **group_n**: A list of a single vector of pasted ngrams per grouping.var in the order used.
- **all**: A single vector of pasted ngrams sorted alphabetically.
- **all_n**: A list of lists a single vectors of ngrams sorted alphabetically (not pasted).

Examples

```r
## Not run:
ngrams(DATA$state, DATA$person, 2)
ngrams(DATA$state, DATA$person, 3)
ngrams(DATA$state, , 3)
with(mraji, ngrams(dialogue, list(sex, fam.aff), 3))

## Alternative ngram analysis:

n_gram <- function(x, n = 2, sep = " "){
  m <- qdap::bag_o_words(x)
  if (length(m) < n) return(character(0))
  starts <- 1:(length(m) - (n - 1))
  ends <- n:length(m)
  Map(function(x, y){
    paste(m[x:y], collapse=sep)
  }, starts, ends)
}

dat <- sentSplit(DATA, "state")
dat[,"grams"] <- sapply(dat[,"state"], function(x) {
  unbag(n_gram(x, sep = "--"))
})

m <- with(dat, as.tdm(grams, person))
rownames(m) <- gsub("--", " ", rownames(m))
as.matrix(m)
rowSums(as.matrix(m))

dat2 <- sentSplit(raj, "dialogue")
dat2[,"grams"] <- sapply(dat2[,"dialogue"], function(x) {
  unbag(n_gram(x, sep = "--"))
})
```
object_pronoun_type

Count Object Pronouns Per Grouping Variable

Description
Count the number of object pronouns per grouping variables.

Usage
object_pronoun_type(text.var, grouping.var = NULL,
  object.pronoun.list = NULL, ...)

Arguments
text.var The text variable
grouping.var The grouping variables. Default NULL generates one word list for all text. Also
takes a single grouping variable or a list of 1 or more grouping variables.
object.pronoun.list A named list of object pronouns. See Details for more.
... Other arguments passed to termco

Details
The following object pronoun categories are the default searched terms:

- me = c(" me ", " my ", " mine ")
- us = c(" us ", " our ", " ours ")
- you = c(" you'd ", " you'll ", " you're ", " you've ", " you ", " your ")
- him = c(" him ", " his ")
- her = c(" her ", " hers ")
- them = c(" them ")
- their = c(" their ", " theirs ")
- it = c(" it'd ", " it'll ", " it's ", " it ")
outlier_detect

Value
Returns a list, of class "object_pronoun_type", of data frames regarding object pronoun word counts:

- **preprocessed**: List of uncollapsed dataframes (raw, prop, rnp) of the class "termco" that contain all searchable object pronouns.
- **raw**: raw word counts by grouping variable
- **prop**: proportional word counts by grouping variable; proportional to each individual's object pronoun use
- **rnp**: a character combination data frame of raw and proportional object pronoun use

See Also

subject_pronoun_type, pronoun_type

Examples

```r
## Not run:
dat <- pres_debates2012
dat <- dat[dat$person %in% c('ROMNEY', 'OBAMA'), ]
(out <- object_pronoun_type(dat$dialogue, dat$person))
plot(out)
plot(out, 2)
plot(out, 3)
plot(out, 3, ncol=2)

scores(out)
counts(out)
proportions(out)
preprocessed(out)

plot(scores(out))
plot(counts(out))
plot(proportions(out))

## End(Not run)
```

---

**outlier_detect**  
Detect Outliers in Text

Description
Locate possible outliers for text variables given numeric word function.

Usage

```r
outlier_detect(text.var, grouping.var = NULL, FUN = word_count, scale.by = "grouping")
```
outlier_labeler

Locate Outliers in Numeric String

Description

Locate and label possible outliers in a string.

Usage

outlier_labeler(x, standardize = TRUE, ...)

Arguments

x A numeric vector.
standardize logical. If TRUE scales the vector first.
... Other arguments passed to scale.

Value

Returns a matrix (one column) of possible outliers coded as "3sd", "2sd" and "1.5sd", corresponding to \( \geq 3 \), \( \geq 2 \), or \( \geq 1.5 \) standard deviations.
paste2

Description

paste2 - Paste unspecified columns or a list of vectors together.

colpaste2df - Wrapper for paste2 that returns a dataframe with columns pasted together.

Usage

collpaste2df(mat, combined.columns, sep = ".", name.sep = "&", keep.orig = TRUE, ...)

Arguments

- multi.columns: The multiple columns or a list of vectors to paste together.
- sep: The character to be used in paste2 to paste the columns.
- handle.na: logical. If TRUE returns NA if any column/vector contains a missing value.
- trim: logical. If TRUE leading/trailing white space is removed.
- mat: A matrix or dataframe.
- combined.columns: A list of named vectors of the colnames/indexes of the numeric columns to be pasted. If a vector is unnamed a name will be assigned.
- name.sep: The character to be used to paste the column names.
- keep.orig: logical. If TRUE the original columns (i.e., combined.columns) will be retained as well.
- ...: Other arguments passed to paste2.

Value

paste2 - Returns a vector with row-wise elements pasted together.

collpaste2df - Returns a dataframe with pasted columns.

See Also

scale

Examples

## Not run:
outlier_labeler(mtcars$hp)[20:32]
by(mtcars$mpg, mtcars$cyl, outlier_labeler)
tapply(mtcars$mpg, mtcars$cyl, outlier_labeler)

## End(Not run)
Note

`paste` differs from `paste2` because `paste` does not allow an unspecified number of columns to be pasted. This behavior can be convenient for inside of functions when the number of columns being pasted is unknown.

See Also

`paste`, `colsplite2df`

Examples

```r
## Not run:
## paste2 examples
v <- rep(list(state.abb[1:8], month.abb[1:8]), 5)
nsample(5:10, 1)
paste(v[1:n]) # odd looking return
paste2(v[1:n])
paste2(v[1:n], sep="|")
paste2(mtcars[1:10,], sep="|")
paste2(mtcars[1:10,], sep="|") # odd looking return
paste2(C02[1:10,], sep="|-")

## colpaste2df examples
A <- list(
a = c(1, 2, 3),
b = qcv(mpg, hp),
c = c("disp", "am")
)
B <- list(
c(1, 2, 3),
new.col = qcv(mpg, hp),
c("disp", "am")
)
E <- list(
c(1, 2, 3, 4, 5),
qcv(mpg, hp),
c("disp", "am")
)

colpaste2df(head(mtcars), A)
colpaste2df(head(mtcars), B)
colpaste2df(head(mtcars), E)
colpaste2df(head(mtcars), qcv(am, disp, drat), sep = ",", name.sep = "|")
colpaste2df(head(C02), list(c(1, 2, 3, 4, 5), qcv("conc", "uptake")))

## End(Not run)
```
phrase_net

**Phrase Nets**

### Description

Create Many Eyes style phrase nets.

### Usage

```r
phrase_net(text.var, freq = 4, r = 0.35, edge.constant = 6,
vertex.constant = 3, ...)  
```

### Arguments

- `text.var` The text variable.
- `freq` The minimum word frequency occurrence.
- `r` The minimum correlation value
- `edge.constant` A constant to multiple the edges by.
- `vertex.constant` A constant to multiple the vertex label sizes by.
- `...` Other arguments passed to `filter`.

### Value

Returns an igraph object.

### Note

While Many Eyes phrase nets inspired this function the two outputs are not identical. The `phrase_net` function operates off of correlations between words in sentences.

### References

- [http://trinker.github.io/many-eye](http://trinker.github.io/many-eye)

### Examples

```r
## Not run:
x <- "Questions must be at least 2 days old to be eligible for a bounty. There can only be 1 active bounty per question at any given time. Users must have at least 75 reputation to offer a bounty, and may only have a maximum of 3 active bounties at any given time. The bounty period lasts 7 days. Bounties must have a minimum duration of at least 1 day. After the bounty ends, there is a grace period of 24 hours to manually award the bounty. If you do not award your bounty within 7 days (plus the grace period), the highest voted answer created after the bounty started with at least 2 upvotes will be";  
```
awarded half the bounty amount. If there's no answer meeting that criteria, the bounty is not awarded to anyone. If the bounty was started by the question owner, and the question owner accepts an answer during the bounty period, and the bounty expires without an explicit award - we assume the bounty owner liked the answer they accepted and award it the full bounty amount at the time of bounty expiration. In any case, you will always give up the amount of reputation specified in the bounty, so if you start a bounty, be sure to follow up and award your bounty to the best answer! As an additional bonus, bounty awards are immune to the daily reputation cap and community wiki mode."

```r
phrase_net(sent_detect(x), r=.5)
library(igraph)
plot(phrase_net(sent_detect(x), r=.5), edge.curved = FALSE)
```

```r
## Declaration of Independence Example
y <- readLines("http://www.constitution.org/usdeclar.txt")
y <- paste(y[grepl("When, in the", y):length(y)], collapse=" ")
phrase_net(sent_detect(y), r=.7)

## Multiple grouping variables
z <- lapply(split(raj.act.1$dialogue, raj.act.1$person), paste, collapse = " ")
par(mfrow=c(2, 5), mai = c(.05, .15, .15, .15))
lapply(seq_along(z), function(i) {
  x <- try(phrase_net(sent_detect(z[i]), r=.6))
  if (!inherits(x, "try-error")) {
    print(x)
    box()
    mtext(names(z)[i])
  }
})

lapply(seq_along(z), function(i) {
  x <- try(phrase_net(sent_detect(z[i]), r=.6))
  if (!inherits(x, "try-error")) {
    dev.new()
    print(x)
    mtext(names(z)[i], adj=-1, cex=1.7, col="red")
  }
})

## End(Not run)
```

---

**plot.animated_character**

*Plots an animated_character Object*
plot.animated_discourse_map

Description
Plots an animated_discourse_map object.

Usage
## S3 method for class 'animated_discourse_map'
plot(x, ...)

Arguments
x The animated_discourse_map object.
... Other arguments passed to print.animated_discourse_map.

plot.animated_discourse_map

Plots an animated_discourse_map Object

---------------------

plot.animated_formality

Description
Plots a animated_formality object.

Usage
## S3 method for class 'animated_formality'
plot(x, ...)

Arguments
x The animated_formality object.
... Other arguments passed to print.animated_formality.

plot.animated_formality

Plots a animated_formality Object

---------------------
Arguments

- **x**: The animated_formality object.
- **...**: Other arguments passed to `print.animated_formality`.

---

**plot.animated_lexical_classification**

Plots an `animated_lexical_classification` Object

Description

Plots an `animated_lexical_classification` object.

Usage

```r
## S3 method for class 'animated_lexical_classification'
plot(x, ...)
```

Arguments

- **x**: The `animated_lexical_classification` object.
- **...**: Other arguments passed to `print.animated_lexical_classification`.

---

**plot.animated_polarity**

Plots an `animated_polarity` Object

Description

Plots an `animated_polarity` object.

Usage

```r
## S3 method for class 'animated_polarity'
plot(x, ...)
```

Arguments

- **x**: The `animated_polarity` object.
- **...**: Other arguments passed to `print.animated_polarity`.
plot.automated_readability_index

Plots a automated_readability_index Object

Description

Plots a automated_readability_index object.

Usage

```r
### S3 method for class 'automated_readability_index'
plot(x, ...)
```

Arguments

- **x**: The readability_score object.
- **...**: ignored

plot.character_table

Plots a character_table Object

Description

Plots a character_table object.

Usage

```r
### S3 method for class 'character_table'
plot(x, label = FALSE, lab.digits = 1,
     percent = NULL, zero.replace = NULL, ...)
```

Arguments

- **x**: The character_table object
- **label**: logical. If TRUE the cells of the heat map plot will be labeled with count and proportional values.
- **lab.digits**: Integer values specifying the number of digits to be printed if label is TRUE.
- **percent**: logical. If TRUE output given as percent. If FALSE the output is proportion. If NULL uses the value from question_type. Only used if label is TRUE.
- **zero.replace**: Value to replace 0 values with. If NULL uses the value from question_type. Only used if label is TRUE.
- **...**: Other arguments passed to qheat
plot.cmspans  
*Plots a cmspans object*

**Description**

Plots a cmspans object.

**Usage**

```r
## S3 method for class 'cmspans'
plot(x, plot.var = NULL, facet.vars = NULL,
     title = "Gantt Plot", ...)
```

**Arguments**

- `x` The sums_cmspans object
- `plot.var` A factor plotting variable (y axis).
- `facet.vars` An optional single vector or list of 1 or 2 to facet by.
- `title` An optional title.
- `...` Other arguments passed to gantt_wrap.

---

plot.cm_distance  
*Plots a cm_distance object*

**Description**

Plots a cm_distance object.

**Usage**

```r
## S3 method for class 'cm_distance'
plot(x, digits = 3, constant = 1,
     label.dist = FALSE, layout = igraph::layout.fruchterman.reingold,
     label.cex = 1, label.cex.scale.by.n = FALSE, alpha = NULL,
     label.color = "black", use.vertex.shape = FALSE, arrow.size = 0.6, ...)
```

**Arguments**

- `x` A cm_distance object.
- `digits` The number of digits to use if distance labels are included on the edges.
- `constant` A constant to weight the edges by.
- `label.dist` logical. If TRUE distance measures are placed on the edges.
- `layout` A layout; see `layout`. 
label.cex A constant to use for the label size.
label.cex.scale.by.n
  logical. If TRUE the label size is scaled by the number of uses of the code.
alpha The cut off value for pvalue inclusion of edges.
label.color Color of the vertex labels.
use.vertex.shape
  logical. If TRUE the vertex label if plotted on a circle.
arrow.size The size of the arrows. Currently this is a constant, so it is the same for every edge.
... Further arguments passed to the chosen layout.

Value

Returns the igraph object.

Note

This plotting method is not particularly well developed. It is suggested that the user further develop the graph via direct use of the igraph package.
# plot.combo_syllable_sum

*Plots a combo_syllable_sum Object*

## Description

Plots a combo_syllable_sum object.

## Usage

```r
## S3 method for class 'combo_syllable_sum'
plot(x, ...)
```

## Arguments

- `x`: The combo_syllable_sum object.
- `...`: ignored

# plot.cumulative_animated_formality

*Plots a cumulative_animated_formality Object*

## Description

Plots a cumulative_animated_formality object.

## Usage

```r
## S3 method for class 'cumulative_animated_formality'
plot(x, ...)
```

## Arguments

- `x`: The cumulative_animated_formality object.
- `...`: ignored
plot.cumulative_animated_lexical_classification

Plots a cumulative_animated_lexical_classification Object

Description

Plots a cumulative_animated_lexical_classification object.

Usage

## S3 method for class 'cumulative_animated_lexical_classification'
plot(x, ...)

Arguments

x The cumulative_animated_lexical_classification object.
...
... ignored

plot.cumulative_animated_polarity

Plots a cumulative_animated_polarity Object

Description

Plots a cumulative_animated_polarity object.

Usage

## S3 method for class 'cumulative_animated_polarity'
plot(x, ...)

Arguments

x The cumulative_animated_polarity object.
...
... ignored
**plot.cumulative_combo_syllable_sum**

*Plots a cumulative_combo_syllable_sum Object*

---

**Description**

Plots a cumulative_combo_syllable_sum object.

**Usage**

```r
## S3 method for class 'cumulative_combo_syllable_sum'
plot(x, ...)
```

**Arguments**

- `x` The cumulative_combo_syllable_sum object.
- `...` ignored

---

**plot.cumulative_end_mark**

*Plots a cumulative_end_mark Object*

---

**Description**

Plots a cumulative_end_mark object.

**Usage**

```r
## S3 method for class 'cumulative_end_mark'
plot(x, ...)
```

**Arguments**

- `x` The cumulative_end_mark object.
- `...` ignored
**plot.cumulative_formality**

*Plots a cumulative_formality Object*

---

**Description**

Plots a cumulative_formality object.

**Usage**

```r
## S3 method for class 'cumulative_formality'
plot(x, ...)
```

**Arguments**

- `x` The cumulative_formality object.
- `...` ignored

---

**plot.cumulative_lexical_classification**

*Plots a cumulative_lexical_classification Object*

---

**Description**

Plots a cumulative_lexical_classification object.

**Usage**

```r
## S3 method for class 'cumulative_lexical_classification'
plot(x, ...)
```

**Arguments**

- `x` The cumulative_lexical_classification object.
- `...` ignored
plot.cumulative_polarity

Plots a cumulative_polarity Object

Description

Plots a cumulative_polarity object.

Usage

## S3 method for class 'cumulative_polarity'
plot(x, ...)

Arguments

x The cumulative_polarity object.
...

plot.cumulative_syllable_freq

Plots a cumulative_syllable_freq Object

Description

Plots a cumulative_syllable_freq object.

Usage

## S3 method for class 'cumulative_syllable_freq'
plot(x, ...)

Arguments

x The cumulative_syllable_freq object.
...

ignored
**plot.discourse_map**

Plots a discourse_map Object

---

**Description**

Plots a discourse_map object.

**Usage**

```r
## S3 method for class 'discourse_map'
plot(x, ...)  
```

**Arguments**

- `x` The discourse_map object.
- `...` Other arguments passed to `print.discourse_map`.

---

**plot.diversity**

Plots a diversity object

---

**Description**

Plots a diversity object.

**Usage**

```r
## S3 method for class 'diversity'
plot(x, ...)  
```

**Arguments**

- `x` The diversity object
- `...` Other arguments passed to `qheat`
plot.end_mark

Plots an end_mark Object

Description

Plots an end_mark object.

Usage

## S3 method for class 'end_mark'
plot(x, ...)

Arguments

x    The end_mark object.
...
    ignored

plot.end_mark_by

Plots a end_mark_by Object

Description

Plots a end_mark_by object.

Usage

## S3 method for class 'end_mark_by'
plot(x, values = FALSE, ...)

Arguments

x    The end_mark_by object.
values logical. If TRUE the cell values will be included on the heatmap.
...
    Other arguments passed to qheat.
plot.end_mark_by_count

Plots an end_mark_by_count Object

Description

Plots a end_mark_by_count object.

Usage

```r
## S3 method for class 'end_mark_by_count'
plot(x, values = TRUE, ...)
```

Arguments

- `x` The end_mark_by_count object.
- `values` logical. If TRUE the cell values will be included on the heatmap.
- `...` Arguments passed to `qheat`.

plot.end_mark_by_preprocessed

Plots an end_mark_by_preprocessed Object

Description

Plots a end_mark_by_preprocessed object.

Usage

```r
## S3 method for class 'end_mark_by_preprocessed'
plot(x, ncol = 1, ...)
```

Arguments

- `x` The end_mark_by_preprocessed object.
- `ncol` The number of columns to use for `facet_wrap`.
- `...` ignored
plot.end_mark_by_proportion

Plots a end_mark_by_proportion Object

Description

Plots a end_mark_by_proportion object.

Usage

```r
## S3 method for class 'end_mark_by_proportion'
plot(x, values = TRUE, ...)
```

Arguments

- `x` The end_mark_by_proportion object.
- `values` logical. If TRUE the cell values will be included on the heatmap.
- `...` Arguments passed to `qheat`.

plot.end_mark_by_score

Plots a end_mark_by_score Object

Description

Plots a end_mark_by_score object.

Usage

```r
## S3 method for class 'end_mark_by_score'
plot(x, values = TRUE, ...)
```

Arguments

- `x` The end_mark_by_score object.
- `values` logical. If TRUE the cell values will be included on the heatmap.
- `...` Arguments passed to `qheat`. 
plot.flesch_kincaid  
Plots a flesch_kincaid Object

Description

Plots a flesch_kincaid object.

Usage

```r
## S3 method for class 'flesch_kincaid'
plot(x, ...)
```

Arguments

- `x` The readability_score object.
- `...` ignored

plot.formality  
Plots a formality Object

Description

Plots a formality object including the parts of speech used to calculate contextual/formal speech.

Usage

```r
## S3 method for class 'formality'
plot(x, point.pch = 20, point.cex = 0.5,
     point.colors = c("gray65", "red"), bar.colors = NULL,
     short.names = TRUE, min.wrdcnt = NULL, order.by.formality = TRUE,
     plot = TRUE, ...)
```

Arguments

- `x` The formality object.
- `point.pch` The plotting symbol.
- `point.cex` The plotting symbol size.
- `point.colors` A vector of colors (length of two) to plot word count and formality score.
- `bar.colors` A palette of colors to supply to the bars in the visualization. If two palettes are provided to the two bar plots respectively.
- `short.names` logical. If TRUE shortens the length of legend and label names for more compact plot width.
- `min.wrdcnt` A minimum word count threshold that must be achieved to be considered in the results. Default includes all subgroups.
order.by.formality

  logical. If TRUE the group formality plot will be ordered by average formality score, otherwise alphabetical order is assumed.

plot

  logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knitr, sweave, etc. to add additional plot layers.

... ignored

Value

  Invisibly returns the ggplot2 objects that form the larger plot.

plot.formality_scores  Plots a formality_scores Object

Description

  Plots a formality_scores object.

Usage

  ## S3 method for class 'formality_scores'
  plot(x, ...)

Arguments

  x

  The formality_scores object.

  ...

  ignored

plot.freq_terms  Plots a freq_terms Object

Description

  Plots a freq_terms object.

Usage

  ## S3 method for class 'freq_terms'
  plot(x, plot = TRUE, ...)

Arguments

  x

  The freq_terms object.

  plot

  logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knitr, sweave, etc. to add additional plot layers.

  ...

  ignored.
plot.gantt  

Plots a gantt object

Description

Plots a gantt object.

Usage

```r
## S3 method for class 'gantt'
plot(x, base = FALSE, title = NULL, ...)
```

Arguments

- `x`: The sums_gantt object
- `base`: logical. If TRUE prints in base graphics system. If FALSE prints in ggplot graphics system.
- `title`: An optional title.
- `...`: Other arguments passed to gantt_wrap or plot_gantt_base

plot.kullback_leibler  

Plots a kullback_leibler object

Description

Plots a kullback_leibler object.

Usage

```r
## S3 method for class 'kullback_leibler'
plot(x, digits = 3, ...)
```

Arguments

- `x`: The kullback_leibler object
- `digits`: Number of decimal places to print.
- `...`: Other arguments passed to qheat
plot.lexical

Plots a lexical Object

Description

Plots a lexical object.

Usage

```r
## S3 method for class 'lexical'
plot(x, min.freq = 1, rot.per = 0, random.order = FALSE,
     title = TRUE, title.color = "blue", ...)
```

Arguments

- `x`: The lexical object.
- `min.freq`: Words with frequency below `min.freq` will not be plotted.
- `rot.per`: Proportion words with 90 degree rotation.
- `random.order`: Logical. If `TRUE` plot words in random order. If `FALSE`, they will be plotted in decreasing frequency.
- `title`: The title of the plot. Use `NULL` to eliminate.
- `title.color`: The color of the title.
- `...`: Other arguments passed to `wordcloud`.

plot.lexical_classification

Plots a lexical_classification Object

Description

Plots a lexical_classification object as a heat map Gantt plot with lexical_classification over time (measured in words) and lexical_classification scores per sentence. In the dotplot plot the black dots are the average lexical_classification per grouping variable.

Usage

```r
## S3 method for class 'lexical_classification'
plot(x, bar.size = 5, low = "blue",
     mid = "grey99", high = "red", ave.lexical_classification.shape = "*",
     alpha = 1/4, shape = 19, point.size = 2.5, jitter = 0.1,
     nrow = NULL, na.rm = TRUE, order.by.lexical_classification = TRUE,
     plot = TRUE, error.bars = TRUE, error.bar.height = 0.5,
     error.bar.size = 0.5, error.bar.color = "black", error.bar.alpha = 0.6,
     ...)
```
Arguments

- **x**: The `lexical_classification` object.
- **bar.size**: The size of the bars used in the Gantt plot.
- **low**: The color to be used for lower values.
- **mid**: The color to be used for mid-range values (default is a less striking color).
- **high**: The color to be used for higher values.
- **ave.lexical_classification.shape**: The shape of the average `lexical_classification` score used in the dot plot.
- **alpha**: Transparency level of points (ranges between 0 and 1).
- **shape**: The shape of the points used in the dot plot.
- **point.size**: The size of the points used in the dot plot.
- **jitter**: Amount of vertical jitter to add to the points.
- **nrow**: The number of rows in the dotplot legend (used when the number of grouping variables makes the legend too wide). If NULL no legend if plotted.
- **na.rm**: logical. Should missing values be removed?
- **order.by.lexical_classification**: logical. If TRUE the group `lexical_classification` plot will be ordered by average `lexical_classification` score, otherwise alphabetical order is assumed.
- **plot**: logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knitr, sweave, etc. to add additional plot layers.
- **error.bars**: logical. If TRUE error bars are added to the `lexical_classification` dot plot using the standard error of the mean `lexical_classification` score.
- **error.bar.height**: The height of the error bar ends.
- **error.bar.size**: The size/thickness of the error bars.
- **error.bar.color**: The color of the error bars. If NULL each bar will be colored by grouping variable.
- **error.bar.alpha**: The alpha level of the error bars.
- **...**: ignored

Value

Invisibly returns the ggplot2 objects that form the larger plot.
**plot.lexical_classification_preprocessed**

Plots a `lexical_classification_preprocessed` Object

**Description**

Plots a lexical_classification_preprocessed object.

**Usage**

```r
## S3 method for class 'lexical_classification_preprocessed'
plot(x, jitter = 0.1,
     text.size = 3.5, alpha = 0.3, ncol = 3, ...)
```

**Arguments**

- **x**
  - The lexical_classification_preprocessed object.
- **jitter**
  - The amount to jitter the points by in the boxplots.
- **text.size**
  - The text size to use for plotting the mean in the boxplots.
- **alpha**
  - The alpha level to use for points.
- **ncol**
  - The number of columns to use for `facet_wrap`.
- **...**
  - ignored

**plot.lexical_classification_score**

Plots a `lexical_classification_score` Object

**Description**

Plots a lexical_classification_score object.

**Usage**

```r
## S3 method for class 'lexical_classification_score'
plot(x, error.bar.height = 0.35,
     error.bar.size = 0.5, error.bar.alpha = 0.3, ...)
```

**Arguments**

- **x**
  - The lexical_classification_score object.
- **error.bar.height**
  - The height of the error bar ends.
- **error.bar.size**
  - The size/thickness of the error bars.
- **error.bar.alpha**
  - The alpha level of the error bars.
- **...**
  - ignored
plot.linsear_write  Plots a linsear_write Object

Description

Plots a linsear_write object.

Usage

## S3 method for class 'linsear_write'
plot(x, alpha = 0.4, ...)

Arguments

- **x**: The readability_score object.
- **alpha**: The alpha level for the points and smooth fill in the scatterplot (length one or two; if two 1-points, 2-smooth fill).
- **...**: ignored

plot.linsear_write_count  Plots a linsear_write_count Object

Description

Plots a linsear_write_count object.

Usage

## S3 method for class 'linsear_write_count'
plot(x, ...)

Arguments

- **x**: The linsear_write_count object.
- **...**: ignored
plot.linsear_write_scores

Plots a linsear_write_scores Object

Description
Plots a linsear_write_scores object.

Usage
## S3 method for class 'linsear_write_scores'
plot(x, alpha = c(0.4, 0.08), ...)

Arguments
- x: The readability_score object.
- alpha: The alpha level for the points and smooth fill in the scatterplot (length one or two; if two 1-points, 2-smooth fill).
- ...: Other arguments passed to geom_smooth.

plot.Network

Plots a Network Object

Description
Plots a Network object.

Usage
## S3 method for class 'Network'
plot(x, ...)

Arguments
- x: The Network object.
- ...: Other arguments passed to print.Network.
plot.object_pronoun_type

Plots an object_pronoun_type Object

Description

Plots an object_pronoun_type object.

Usage

```r
## S3 method for class 'object_pronoun_type'
plot(x, type = 1, ...)
```

Arguments

- `x`: The object_pronoun_type object.
- `type`: An integer of 1, 2, 3) corresponding to 1 - heat map; 2 - lexical dispersion plot; 3 - faceted bar graph.
- `...`: Other arguments passed to `qheat`, `dispersion_plot`, or `facet_wrap`.

plot.polarity

Plots a polarity Object

Description

Plots a polarity object as a heat map Gantt plot with polarity over time (measured in words) and polarity scores per sentence. In the dotplot plot the black dots are the average polarity per grouping variable.

Usage

```r
## S3 method for class 'polarity'
plot(x, bar.size = 5, low = "blue", mid = "grey99", high = "red", ave.polarity.shape = "+", alpha = 1/4, shape = 19,
     point.size = 2.5, jitter = 0.1, nrow = NULL, na.rm = TRUE,
     order.by.polarity = TRUE, plot = TRUE, error.bars = TRUE,
     error.bar.height = 0.5, error.bar.size = 0.5, error.bar.color = "black",
     ...)
```
Arguments

- **x**  
The polarity object.
- **bar.size**  
The size of the bars used in the Gantt plot.
- **low**  
The color to be used for lower values.
- **mid**  
The color to be used for mid-range values (default is a less striking color).
- **high**  
The color to be used for higher values.
- **ave.polarity.shape**  
The shape of the average polarity score used in the dot plot.
- **alpha**  
Transparency level of points (ranges between 0 and 1).
- **shape**  
The shape of the points used in the dot plot.
- **point.size**  
The size of the points used in the dot plot.
- **jitter**  
Amount of vertical jitter to add to the points.
- **nrow**  
The number of rows in the dotplot legend (used when the number of grouping variables makes the legend too wide). If NULL no legend if plotted.
- **na.rm**  
logical. Should missing values be removed?
- **order.by.polarity**  
logical. If TRUE the group polarity plot will be ordered by average polarity score, otherwise alphabetical order is assumed.
- **plot**  
logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knitr, sweave, etc. to add additional plot layers.
- **error.bars**  
logical. If TRUE error bars are added to the polarity dot plot using the standard error of the mean polarity score.
- **error.bar.height**  
The height of the error bar ends.
- **error.bar.size**  
The size/thickness of the error bars.
- **error.bar.color**  
The color of the error bars. If NULL each bar will be colored by grouping variable.
- **...**  
ignored

Value

Invisibly returns the ggplot2 objects that form the larger plot.
plot.polarity_count  Plots a polarity_count Object

Description

Plots a polarity_count object as a heat map Gantt plot with polarity over time (measured in words) and polarity scores per sentence. In the dotplot plot the black dots are the average polarity per grouping variable.

Usage

```r
## S3 method for class 'polarity_count'
plot(x, bar.size = 5, low = "blue",
     mid = "grey99", high = "red", ave.polarity.shape = "+", alpha = 1/4,
     shape = 19, point.size = 2.5, jitter = 0.1, nrow = NULL,
     na.rm = TRUE, order.by.polarity = TRUE, plot = TRUE,
     error.bars = TRUE, error.bar.height = 0.5, error.bar.size = 0.5,
     error.bar.color = "black", ...)
```

Arguments

- `x` The polarity_count object.
- `bar.size` The size of the bars used in the Gantt plot.
- `low` The color to be used for lower values.
- `mid` The color to be used for mid-range values (default is a less striking color).
- `high` The color to be used for higher values.
- `ave.polarity.shape` The shape of the average polarity score used in the dot plot.
- `alpha` Transparency level of points (ranges between 0 and 1).
- `shape` The shape of the points used in the dot plot.
- `point.size` The size of the points used in the dot plot.
- `jitter` Amount of vertical jitter to add to the points.
- `nrow` The number of rows in the dotplot legend (used when the number of grouping variables makes the legend too wide). If NULL no legend if plotted.
- `na.rm` logical. Should missing values be removed?
- `order.by.polarity` logical. If TRUE the group polarity plot will be ordered by average polarity score, otherwise alphabetical order is assumed.
- `plot` logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knitr, sweave, etc. to add additional plot layers.
- `error.bars` logical. If TRUE error bars are added to the polarity dot plot using the standard error of the mean polarity score.
error.bar.height
The height of the error bar ends.

error.bar.size
The size/thickness of the error bars.

error.bar.color
The color of the error bars. If NULL each bar will be colored by grouping variable.

... ignored

Value

Invisibly returns the ggplot2 objects that form the larger plot.

plot.polarity_score
Plots a polarity_score Object

Description

Plots a polarity_score object.

Usage

## S3 method for class 'polarity_score'
plot(x, error.bar.height = 0.35,
     error.bar.size = 0.5, error.bar.alpha = 0.3, ...)

Arguments

x The polarity_score object.

error.bar.height The height of the error bar ends.
	error.bar.size The size/thickness of the error bars.
	error.bar.alpha The alpha level of the error bars.

... ignored
plot.pos

Plots a pos Object

Description
Plots a pos object.

Usage

```r
## S3 method for class 'pos'
plot(x, ...)
```

Arguments

- `x` The pos object
- `...` ignored

plot.pos_by

Plots a pos_by Object

Description
Plots a pos_by object.

Usage

```r
## S3 method for class 'pos_by'
plot(x, label = FALSE, lab.digits = 1, percent = NULL,
     zero.replace = NULL, ...)
```

Arguments

- `x` The pos_by object
- `label` logical. If TRUE the cells of the heat map plot will be labeled with count and proportional values.
- `lab.digits` Integer values specifying the number of digits to be printed if label is TRUE.
- `percent` logical. If TRUE output given as percent. If FALSE the output is proportion. If NULL uses the value from question_type. Only used if label is TRUE.
- `zero.replace` Value to replace 0 values with. If NULL uses the value from question_type. Only used if label is TRUE.
- `...` Other arguments passed to qheat.
plot.pos_preprocessed  
Plots a pos_preprocessed Object

Description

Plots a pos_preprocessed object.

Usage

```r
## S3 method for class 'pos_preprocessed'
plot(x, ...)
```

Arguments

- `x`  
The pos_preprocessed object.
- `...`  
Ignored

plot.pronoun_type  
Plots an pronoun_type Object

Description

Plots an pronoun_type object.

Usage

```r
## S3 method for class 'pronoun_type'
plot(x, type = 1, ...)
```

Arguments

- `x`  
The pronoun_type object.
- `type`  
An integer of 1, 2, 3) corresponding to 1 - heat map; 2 - lexical dispersion plot; 3 - facetted bar graph.
- `...`  
Other arguments passed to `qheat`, `dispersion_plot`, or `facet_wrap`. 
plot.question_type  
Plots a question_type Object

Description
Plots a question_type object.

Usage

## S3 method for class 'question_type'
plot(x, label = FALSE, lab.digits = 1,
     percent = NULL, zero.replace = NULL, ...)

Arguments

x  The question_type object.
label  logical. If TRUE the cells of the heat map plot will be labeled with count and
       proportional values.
lab.digits  Integer values specifying the number of digits to be printed if label is TRUE.
percent  logical. If TRUE output given as percent. If FALSE the output is proportion. If
          NULL uses the value from question_type. Only used if label is TRUE.
zero.replace  Value to replace 0 values with. If NULL uses the value from question_type.
              Only used if label is TRUE.
...  Other arguments passed to qheat.

plot.question_type_preprocessed
Plots a question_type_preprocessed Object

Description
Plots a question_type_preprocessed object.

Usage

## S3 method for class 'question_type_preprocessed'
plot(x, ...)

Arguments

x  The question_type_preprocessed object.
...  Arguments passed to gantt_plot.
plot.readability_count

*Plots a readability_count Object*

---

**Description**

Plots a readability_count object.

**Usage**

```r
## S3 method for class 'readability_count'
plot(x, alpha = 0.3, ...)
```

**Arguments**

- `x` The readability_count object.
- `alpha` The alpha level to use for points.
- `...` ignored

---

plot.readability_score

*Plots a readability_score Object*

---

**Description**

Plots a readability_score object.

**Usage**

```r
## S3 method for class 'readability_score'
plot(x, alpha = 0.3, auto.label, grid, div.col, 
     ...)```

**Arguments**

- `x` The readability_score object.
- `alpha` The alpha level to be used for the points.
- `auto.label` logical. For plotting fry only, if TRUE labels automatically added. If FALSE the user clicks interactively.
- `grid` logical. For plotting fry only, if TRUE a micro grid is displayed similar to Fry’s original depiction, though this makes visualizing more difficult.
- `div.col` For plotting fry only, the color of the grade level division lines.
- `...` ignored
**plot.rmgantt**

*Plots a rmgantt object*

**Description**

Plots a rmgantt object.

**Usage**

```r
## S3 method for class 'rmgantt'
plot(x, title, transform = FALSE, ...)
```

**Arguments**

- `x` The sums_rmgantt object
- `title` An optional title.
- `transform` logical. If TRUE and there are two repeated measures the faceting is reversed.
- `...` Other arguments passed to gantt_wrap

---

**plot.sent_split**

*Plots a sent_split Object*

**Description**

Plots a sent_split object.

**Usage**

```r
## S3 method for class 'sent_split'
plot(x, text.var = NULL, rm.var = NULL, ...)
```

**Arguments**

- `x` The sent_split object.
- `text.var` The text variable (character string).
- `rm.var` An optional repeated measures character vector of 1 or 2 to facet by. If NULL the rm.var from sentSplit is used. To avoid this behavior use FALSE.
- `...` Other arguments passed to tot_plot.
plot.SMOG

*Plots a SMOG Object*

**Description**

Plots a SMOG object.

**Usage**

```r
## S3 method for class 'SMOG'
plot(x, ...)
```

**Arguments**

- `x`: The readability_score object.
- `...`: ignored

---

plot.subject_pronoun_type

*Plots an subject_pronoun_type Object*

**Description**

Plots an subject_pronoun_type object.

**Usage**

```r
## S3 method for class 'subject_pronoun_type'
plot(x, type = 1, ...)
```

**Arguments**

- `x`: The subject_pronoun_type object.
- `type`: An integer of 1, 2, 3) corresponding to 1 - heat map; 2 - lexical dispersion plot; 3 - facetted bar graph.
- `...`: Other arguments passed to `qheat`, `dispersion_plot`, or `facet_wrap`.
plot.sums_gantt

Plots a sums_gantt object

Description

Plots a sums_gantt object.

Usage

```r
## S3 method for class 'sums_gantt'
plot(x, base = TRUE, title = NULL, ...)
```

Arguments

- `x`: The sums_gantt object
- `base`: logical. If TRUE prints in base graphics system. If FALSE prints in ggplot graphics system.
- `title`: An optional title.
- `...`: Other arguments passed to gantt_wrap or plot_gantt_base

plot.sum_cmspans

Plot Summary Stats for a Summary of a cmspans Object

Description

Plots a heat map of summary statistics for sum_cmspans objects (the object produced by calling summary on a cmspans object).

Usage

```r
## S3 method for class 'sum_cmspans'
plot(x, digits = 3, sep = ".", name.sep = "&",
     values = TRUE, high = "red", transpose = TRUE, plot = TRUE,
     facet.vars = "time", rev.codes = !transpose, rev.stats = !transpose,
     ...)
```

Arguments

- `x`: The sum_cmspans object (the object produced by calling summary on a cmspans object)
- `digits`: The number of digits displayed if values is TRUE.
- `sep`: The character that was used in paste2 to paste the columns.
- `name.sep`: The character that was used to paste the column names.
values logical. If TRUE the cell values will be included on the heatmap.
high The color to be used for higher values.
transpose logical. If TRUE the dataframe is rotated 90 degrees.
plot logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knit, sweave, etc. to add additional plot layers.
facet.vars A character vector of names to facet by. Default is "time".
rev.codes logical If TRUE the plotting order of the code groups is reversed.
rev.stats logical If TRUE the plotting order of the code descriptive statistics is reversed.
... Other arguments passed to qheat.

See Also

summary.cmspans

plot.syallable_freq  Plots a syllable_freq Object

Description

Plots a syllable_freq object.

Usage

## S3 method for class 'syallable_freq'
plot(x, ...)

Arguments

x The syllable_freq object.
... ignored

plot.table_count  Plots a table_count Object

Description

Plots a table_count object.

Usage

## S3 method for class 'table_count'
plot(x, values = TRUE, high = "red", ...)

plot.table_proportion  

**Arguments**

- `x`: The table_count object.
- `values`: logical. If TRUE the cell values will be included on the heatmap.
- `high`: The color to be used for higher values.
- `...`: Other arguments passed to `qheat`.

---

**Description**

Plots a table_proportion object.

**Usage**

```r
## S3 method for class 'table_proportion'
plot(x, values = TRUE, high = "red", ...)
```

**Arguments**

- `x`: The table_proportion object.
- `values`: logical. If TRUE the cell values will be included on the heatmap.
- `high`: The color to be used for higher values.
- `...`: Other arguments passed to `qheat`.

---

plot.table_score  

**Description**

Plots a table_score object.

**Usage**

```r
## S3 method for class 'table_score'
plot(x, values = TRUE, high = "red", ...)
```

**Arguments**

- `x`: The table_score object.
- `values`: logical. If TRUE the cell values will be included on the heatmap.
- `high`: The color to be used for higher values.
- `...`: Other arguments passed to `qheat`.
plot.termco  

Plots a termco object

Description

Plots a termco object.

Usage

```r
## S3 method for class 'termco'
plot(x, label = FALSE, lab.digits = 1, percent = NULL,
     zero.replace = NULL, ...)
```

Arguments

- `x`  
The termco object.
- `label`  
  logical. If TRUE the cells of the heat map plot will be labeled with count and proportional values.
- `lab.digits`  
  Integer values specifying the number of digits to be printed if `label` is TRUE.
- `percent`  
  logical. If TRUE output given as percent. If FALSE the output is proportion. If NULL uses the value from `termco`. Only used if `label` is TRUE.
- `zero.replace`  
  Value to replace 0 values with. If NULL uses the value from `termco`. Only used if `label` is TRUE.
- `...`  
  Other arguments passed to qheat.

plot.type_token_ratio  

Plots a type_token_ratio Object

Description

Plots a type_token_ratio object.

Usage

```r
## S3 method for class 'type_token_ratio'
plot(x, ...)  
```

Arguments

- `x`  
  The type_token_ratio object.
- `...`  
  ignored.
plot.weighted_wfm

Plots a weighted_wfm object

Description

Plots a weighted_wfm object.

Usage

## S3 method for class 'weighted_wfm'
plot(x, non.zero = FALSE, digits = 0,
     by.column = NULL, high = ifelse(non.zero, "black", "blue"),
     grid = ifelse(non.zero, "black", "white"), plot = TRUE, ...)

Arguments

x  The weighted_wfm object

non.zero  logical. If TRUE all values converted to dummy coded based on x_ij > 0.

digits  The number of digits displayed if values is TRUE.

by.column  logical. If TRUE applies scaling to the column. If FALSE applies scaling by row (use NULL to turn off scaling).

high  The color to be used for higher values.

grid  The color of the grid (Use NULL to remove the grid).

plot  logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knit, sweave, etc. to add additional plot layers.

...  Other arguments passed to qheat.

plot.wfdf

Plots a wfdf object

Description

Plots a wfdf object.

Usage

## S3 method for class 'wfdf'
plot(x, ...)

Arguments

x  The wfdf object

...  Other arguments passed to plot.wfm.
plot.wfm

Plots a wfm object

Description

Plots a wfm object.

Usage

## S3 method for class 'wfm'
plot(x, non.zero = FALSE, digits = 0, by.column = NULL,
     high = ifelse(non.zero, "black", "blue"), grid = ifelse(non.zero, "black",
               "white"), plot = TRUE, ...)

Arguments

- **x**
  - The wfm object
- **non.zero**
  - logical. If TRUE all values converted to dummy coded based on x_ij > 0.
- **digits**
  - The number of digits displayed if values is TRUE.
- **by.column**
  - logical. If TRUE applies scaling to the column. If FALSE applies scaling by row
    (use NULL to turn off scaling).
- **high**
  - The color to be used for higher values.
- **grid**
  - The color of the grid (Use NULL to remove the grid).
- **plot**
  - logical. If TRUE the plot will automatically plot. The user may wish to set to
    FALSE for use in knit, sweave, etc. to add additional plot layers.
- ...
  - Other arguments passed to qheat.

plot.word_cor

Plots a word_cor object

Description

Plots a word_cor object.

Usage

## S3 method for class 'word_cor'
plot(x, label = TRUE, lab.digits = 3, high = "red",
     low = "white", grid = NULL, ncol = NULL, ...)
**Arguments**

- **x**: The word_length object.
- **label**: logical. If TRUE the cells of the heat map plot will be labeled with count and proportional values.
- **lab.digits**: Integer values specifying the number of digits to be printed if label is TRUE.
- **high**: The color to be used for higher values.
- **low**: The color to be used for lower values.
- **grid**: The color of the grid (Use NULL to remove the grid).
- **ncol**: The number of columns to arrange the facets in (specifying an integer results in the use of `facet_wrap`, specifying NULL utilizes a single column with `facet_grid`. The second approach limits columns but allows the y scale’s space to be free.
- **...**: Other arguments passed to qheat if matrix and other arguments passed to `geom_point` if a list.

---

**Description**

Plots a word_length object.

**Usage**

```r
## S3 method for class 'word_length'
plot(x, label = FALSE, lab.digits = 1,
     percent = NULL, zero.replace = NULL, ...)
```

**Arguments**

- **x**: The word_length object.
- **label**: logical. If TRUE the cells of the heat map plot will be labeled with count and proportional values.
- **lab.digits**: Integer values specifying the number of digits to be printed if label is TRUE.
- **percent**: logical. If TRUE output given as percent. If FALSE the output is proportion. If NULL uses the value from `word_length`. Only used if label is TRUE.
- **zero.replace**: Value to replace 0 values with. If NULL uses the value from `word_length`. Only used if label is TRUE.
- **...**: Other arguments passed to qheat.
plot.word_position  
Plots a word_position object

Description
Plots a word_position object.

Usage

```r
## S3 method for class 'word_position'
plot(x, qheat = TRUE, scale = TRUE, ...)
```

Arguments
- `x`: The word_position object.
- `qheat`: logical. If TRUE `qheat` is used to plot. If FALSE `heatmap` is used.
- `scale`: logical. If TRUE scales heatmap by row. If FALSE no scaling occurs.
- `...`: Other arguments passed to `qheat` or `heatmap`.

plot.word_proximity  
Plots a word_proximity object

Description
Plots a word_proximity object.

Usage

```r
## S3 method for class 'word_proximity'
plot(x, label = TRUE, lab.digits = NULL,
     high = "red", low = "white", grid = NULL, ...)
```

Arguments
- `x`: The word_proximity object
- `label`: logical. If TRUE the cells of the heat map plot will be labeled with count and proportional values.
- `lab.digits`: Integer values specifying the number of digits to be printed if `label` is TRUE.
- `high`: The color to be used for higher values.
- `low`: The color to be used for lower values.
- `grid`: The color of the grid (Use NULL to remove the grid).
- `...`: Other arguments passed to `qheat`. 
plot.word_stats  

Plots a word_stats object

Description

Plots a word_stats object.

Usage

## S3 method for class 'word_stats'
plot(x, label = FALSE, lab.digits = NULL, ...)

Arguments

x  The word_stats object

label  logical. If TRUE the cells of the heat map plot will be labeled with count and proportional values.

lab.digits  Integer values specifying the number of digits to be printed if label is TRUE.

...  Other arguments passed to qheat.

plot.word_stats_counts

Plots a word_stats_counts Object

Description

Plots a word_stats_counts object.

Usage

## S3 method for class 'word_stats_counts'
plot(x, alpha = 0.3, ...)

Arguments

x  The word_stats_counts object.

alpha  The alpha level to use for points.

...  ignored
polarity - Approximate the sentiment (polarity) of text by grouping variable(s).

Usage

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

Arguments

text.var The text variable.
grouping.var The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
polarity.frame A dataframe or hash key of positive/negative words and weights.
constrain logical. If TRUE polarity values are constrained to be between -1 and 1 using the following transformation:

\[
\left[\left(1 - \frac{1}{\exp(\delta)}\right) \cdot 2\right] - 1
\]

negators A character vector of terms reversing the intent of a positive or negative word.
amplifiers A character vector of terms that increase the intensity of a positive or negative word.
deamplifiers A character vector of terms that decrease the intensity of a positive or negative word.
question.weight The weighting of questions (values from 0 to 1). Default 0 corresponds with the belief that questions (pure questions) are not polarized. A weight may be applied based on the evidence that the questions function with polarity.
amplifier.weight The weight to apply to amplifiers/deamplifiers (values from 0 to 1). This value will multiply the polarized terms by 1 + this value.
n.before The number of words to consider as valence shifters before the polarized word.
n.after The number of words to consider as valence shifters after the polarized word.
rm.incomplete logical. If TRUE text rows ending with qdap’s incomplete sentence end mark (|) will be removed from the analysis.
digits Integer; number of decimal places to round when printing.
...
Other arguments supplied to strip.
Details

The equation used by the algorithm to assign value to polarity of each sentence first utilizes the sentiment dictionary (Hu and Liu, 2004) to tag polarized words. A context cluster \((x^T_i)\) of words is pulled from around this polarized word (default 4 words before and two words after) to be considered as valence shifters. The words in this context cluster are tagged as neutral \((x^0_i)\), negator \((x^N_i)\), amplifier \((x^a_i)\), or de-amplifier \((x^d_i)\). Neutral words hold no value in the equation but do affect word count \((n)\). Each polarized word is then weighted \(w\) based on the weights from the polarity.frame argument and then further weighted by the number and position of the valence shifters directly surrounding the positive or negative word. The researcher may provide a weight \(c\) to be utilized with amplifiers/de-amplifiers (default is .8; deamplifier weight is constrained to -1 lower bound). Last, these context cluster \((x^T_i)\) are summed and divided by the square root of the word count \((\sqrt{n})\) yielding an unbounded polarity score \((\delta)\). Note that context clusters containing a comma before the polarized word will only consider words found after the comma.

\[
\delta = \frac{x^T_i}{\sqrt{n}}
\]

Where:

\[
x^T_i = \sum \left( (1 + c(x^A_i - x^D_i)) \cdot w(-1)\sum x^N_i \right)
\]

\[
x^A_i = \sum (w_{neg} \cdot x^a_i)
\]

\[
x^D_i = \max(x^D'_i, -1)
\]

\[
x^D'_i = \sum (-w_{neg} \cdot x^a_i + x^d_i)
\]

\[
w_{neg} = \left( \sum x^N_i \right) \mod 2
\]

Value

Returns a list of:

- all A dataframe of scores per row with:
  - group.var - the grouping variable
  - wc - word count
  - polarity - sentence polarity score
  - pos.words - words considered positive
  - neg.words - words considered negative
  - text.var - the text variable

- group A dataframe with the average polarity score by grouping variable:
• group.var - the grouping variable
• total.sentences - Total sentences spoken.
• total.words - Total words used.
• ave.polarity - The sum of all polarity scores for that group divided by number of sentences spoken.
• sd.polarity - The standard deviation of that group’s sentence level polarity scores.
• stan.mean.polarity - A standardized polarity score calculated by taking the average polarity score for a group divided by the standard deviation.

digits integer value of number of digits to display; mostly internal use

Note

The polarity score is dependent upon the polarity dictionary used. This function defaults to the word polarity dictionary used by Hu, M., & Liu, B. (2004), however, this may not be appropriate for the context of children in a classroom. The user may (is encouraged) to provide/augment the dictionary (see the sentiment_frame function). For instance the word "sick" in a high school setting may mean that something is good, whereas "sick" used by a typical adult indicates something is not right or negative connotation (deixis).

Also note that polarity assumes you’ve run sentSplit.

References


http://www.slideshare.net/jeffreybreen/r-by-example-mining-twitter-for
http://hedonometer.org/papers.html Links to papers on hedonometrics

See Also

https://github.com/trestletech/Sermon-Sentiment-Analysis

Examples

## Not run:
with(DATA, polarity(state, list(sex, adult)))
(poldat <- with(sentSplit(DATA, 4), polarity(state, person)))
counts(poldat)
scores(poldat)
plot(poldat)

poldat2 <- with(mraja$spl, polarity(dialogue,
  list(sex, fam.aff, died)))
colsplit2df(scores(poldat2))
plot(poldat2)
plot(scores(poldat2))
cumulative(poldat2)

poldat3 <- with(rajSPLIT, polarity(dialogue, person))
poldat3["group"][, "OL"] <- outlier_labeler(scores(poldat3)[, "ave.polarity"])
poldat3["all"][, "OL"] <- outlier_labeler(counts(poldat3)[, "polarity"])
htruncdf(scores(poldat3), 10)
htruncdf(counts(poldat3), 15, 8)
plot(poldat3)
plot(poldat3, nrow=4)
qheat(scores(poldat3)[, -7], high="red", order.b="ave.polarity")

### Create researcher defined sentiment.frame
POLKEY <- sentiment_frame(positive.words, negative.words)
POLKEY
c("abrasive", "abrupt", "happy") %>% POLKEY

# Augmenting the sentiment.frame
mycorpus <- c("Wow that's a raw move.", "His jokes are so corny")
counts(polarity(mycorpus))

POLKEY <- sentiment_frame(c(positive.words, "raw"), c(negative.words, "corny"))
counts(polarity(mycorpus以防POLKEY))

### ANIMATION

(deb2 <- with(subset(pres_debates2012, time="time 2"),
  polarity(dialogue, person)))

bg_black <- Animate(deb2, neutral="white", current.speaker.color="grey70")
print(bg_black, pause=.75)

bgb <- vertex_apply(bg_black, label.color="grey80", size=20, color="grey40")
bgb <- edge_apply(bgb, label.color="yellow")
print(bgb, bg="black", pause=.75)

### Save it
library(animation)
library(igraph)
library(plotrix)

loc <- folder(animation_polarity)

### Set up the plotting function
oopt <- animation::ani.options(interval = 0.1)

FUN <- function() {
  Title <- "Animated Polarity: 2012 Presidential Debate 2"
  Legend <- c(-1.1, -1.25, -.2, -1.2)
  Legend.cex <- 1
  lapply(seq_along(bgb), function(i) {
    par(mar=c(2, 0, 1, 0), bg="black")
    set.seed(10)
    plot.igraph(bgb[[i]], edge.curved=TRUE)
    mtext(Title, side=3, col="white")
  })
}
color.legend(Legend[1], Legend[2], Legend[3], Legend[4],
c("Negative", "Neutral", "Positive"), attributes(bgb)[["legend"]],
cex = Legend.cex, col="white")
animation::ani.pause()
}

FUN()

## Detect OS

if (Platform$OS.type == "windows") shell
else system

saveHTML(FUN(), autoplay = FALSE, loop = TRUE, verbose = FALSE,
ani.height = 500, ani.width=500,
outdir = file.path(loc, "new"), single.opts =
"controls": ['first', 'play', 'loop', 'speed'], 'delayMin': 0)

## Animated corresponding text plot

Animate(deb2, type="text")

#---------------------#
## Complex Animation ##
#---------------------#

library(animation)
library(grid)
library(gridExtra)
library(qdap)
library(reports)
library(igraph)
library(plotrix)

deb2dat <- subset(pres.debates2012, time="time 2")
deb2dat[, "person"] <- factor(deb2dat[, "person"])
(deb2 <- with(deb2dat, polarity(dialogue, person)))

## Set up the network version

bg_black <- Animate(deb2, neutral="white", current.speaker.color="grey70")
bgb <- vertex_apply(bg_black, label.color="grey80", size=30, label.size=22,
                   color="grey40")
bgb <- edge_apply(bgb, label.color="yellow")

## Set up the bar version

deb2_bar <- Animate(deb2, as.network=FALSE)
## Generate a folder

```r
loc2 <- folder(animation_polarity2)
```

## Set up the plotting function

```r
oopt <- animation::ani.options(interval = 0.1)
```

```r
FUN2 <- function(follow=FALSE, theseq = seq_along(bgb)) {
  Title <- "Animated Polarity: 2012 Presidential Debate 2"
  Legend <- c(.2, -1.075, 1.5, -1.005)
  Legend.cex <- 1

  lapply(theseq, function(i) {
    if (follow) {
      png(file=sprintf("%s/images/Rplot%s.png", loc2, i),
           width=650, height=725)
    }
    ## Set up the layout
    layout(matrix(c(rep(1, 9), rep(2, 4)), 1, 1, byrow = TRUE))

    ## Plot 1
    par(mar=c(2, 0, 2, 0), bg="black")
    set.seed(20)
    plot.igraph(bgb[[i]], edge.curved=TRUE)
    mtext(Title, side=3, col="white")
    color.legend(Legend[1], Legend[2], Legend[3], Legend[4],
                 c("Negative", "Neutral", "Positive"), attributes(bgb)["legend"],
                 cex = Legend.cex, col="white")

    ## Plot 2
    plot.new()
    vps <- baseViewports()

    uns <- unit(c(-1.3, .5, -.75, .25), "cm")
    p <- deb2_bar[[i]] +
    theme(plot.margin = uns,
          text = element_text(color="white"),
          plot.background = element_rect(fill = "black",
                                           color="black"))
    print(p, vp = vpStack(vps$figure, vps$plot))
    animation::ani.pause()

    if (follow) {
      dev.off()
    }
  })
}
```

```r
FUN2()
```
## Detect OS

type <- if(.Platform$OS.type == "windows") shell else system

saveHTML(FUN2(), autoplay = FALSE, loop = TRUE, verbose = FALSE,
    ani.height = 1000, ani.width = 650,
    outdir = loc2, single.opts =
    "'controls': ['first', 'play', 'loop', 'speed'], 'delayMin': 0")

FUN2(TRUE)

#====================================
library(animation)
library(grid)
library(gridBase)
library(qdap)
library(reports)
library(igraph)
library(plotrix)
library(gplots)

deb2dat <- subset(pres.debates2012, time="time 2")
deb2dat[, "person"] <- factor(deb2dat[, "person"])
(deb2 <- with(deb2dat, polarity(dialogue, person)))

## Set up the network version
bg_black <- Animate(deb2, neutral="white", current.speaker.color="grey70")
bgb <- vertex_apply(bg_black, label.color="grey80", size=30, label.size=22,
    color="grey40")
bgb <- edge_apply(bgb, label.color="yellow")

## Set up the bar version
deb2_bar <- Animate(deb2, as.network=FALSE)

## Set up the line version
deb2_line <- plot(cumulative(deb2_bar))

## Generate a folder
loc2b <- folder(animation_polarity2)

## Set up the plotting function
oopt <- animation::ani.options(interval = 0.1)

FUN2 <- function(follow=FALSE, theseq = seq_along(bgb)) {

    Title <- "Animated Polarity: 2012 Presidential Debate 2"
    Legend <- c(.2, -1.075, 1.5, -1.005)
    Legend.cex <- 1

    lapply(theseq, function(i) {
        if (follow) {
            png(file=sprintf("%s/images/Rplot%s.png", loc2b, i),
                width=650, height=725)
        }
    })
## Set up the layout
layout(matrix(c(rep(1, 9), rep(2, 4)), 13, 1, byrow = TRUE))

## Plot 1
par(mar=c(2, 0, 2, 0), bg="black")
set.seed(20)
plot.igraph(bgb[[i]], edge.curved=TRUE)
mtext(Title, side=3, col="white")
color.legend(Legend[1], Legend[2], Legend[3], Legend[4],
c("Negative", "Neutral", "Positive"), attributes(bgb)["legend"],
cex = Legend.cex, col="white")

## Plot2
plot.new()
vps <- baseViewports()
uns <- unit(c(-1.3,.5,-.75,.25), "cm")
p <- deb2_bar[[i]] +
  theme(plot.margin = uns,
    text=element_text(color="white"),
    plot.background = element_rect(fill="black",
      color="black"))
print(p, wp = vpStack(vps$figure, vps$plot))
animation::ani.pause()
if (follow) {
  dev.off()
}
}

FUN2()

## Detect OS
type <- if (.Platform$OS.type == "windows") shell else system

saveHTML(FUN2(), autoplay = FALSE, loop = TRUE, verbose = FALSE,
  ani.height = 1000, ani.width=650,
  outdir = loc2b, single.opts =
  "'controls': ['first', 'play', 'loop', 'speed', 'delayMin': 0"])

FUNCTION(TRUE)

## Increased complexity

## Helper function to cbind ggpplots
cbinder <- function(x, y){
  uns_x <- unit(c(-1.3,.15,-.75,.25), "cm")
  uns_y <- unit(c(-1.3,.5,-.75,.15), "cm")
```r
x <- x + theme(plot.margin = uns_x,
              text = element_text(color = "white"),
              plot.background = element_rect(fill = "black",
                                            color = "black")
)

y <- y + theme(plot.margin = uns_y,
              text = element_text(color = "white"),
              plot.background = element_rect(fill = "black",
                                            color = "black")
)

plots <- list(x, y)
grobs <- list()
heights <- list()

for (i in 1:length(plots)){
  grobs[[i]] <- ggplotGrob(plots[[i]])
  heights[[i]] <- grobs[[i]]$heights[2:5]
}

maxheight <- do.call(grid::unit.pmax, heights)

for (i in 1:length(grobs)){
  grobs[[i]]$heights[2:5] <- as.list(maxheight)
}

do.call("arrangeGrob", c(grobs, ncol = 2))
}

deb2_combo <- Map(cbinder, deb2_bar, deb2_line)

## Generate a folder
loc3 <- folder(animation_polarity3)

FUN3 <- function(follow=FALSE, theseq = seq_along(bgb)) {

  Title <- "Animated Polarity: 2012 Presidential Debate 2"
  Legend <- c(.2, -1.075, 1.5, -1.005)
  Legend.cex <- 1

  lapply(theseq, function(i) {
    if (follow) {
      png(file=sprintf("%s/images/Rplot%i.png", loc3, i),
           width=650, height=725)
    }
    ## Set up the layout
    layout(matrix(c(rep(1, 9), rep(2, 4)), 13, 1, byrow = TRUE))
    ## Plot 1
    par(mar=c(2, 0, 2, 0), bg="black")
  })
```

polarity

```r
#par(mar=c(2, 0, 2, 0))
set.seed(20)
plot.igraph(bgb[[i]], edge.curved=TRUE)
mtext(Ttitle, side=3, col="white")
color.legend(Legend[1], Legend[2], Legend[3], Legend[4],
c("Negative", "Neutral", "Positive"), attributes(bgb)[["legend"]],
cex = Legend.cex, col="white")

## Plot2
plot.new()
vps <- baseViewports()
p <- deb2_combo[[i]]
print(p, vp = vpStack(vps$figure, vps$plot))
animation::ani.pause()

if (follow) {
  dev.off()
}
)

FUN3()

type <- if (.Platform$OS.type == "windows") shell else system

saveHTML(FUN3(), autoplay = FALSE, loop = TRUE, verbose = FALSE,
  ani.height = 1000, ani.width=650,
  outdir = loc3, single.opts =
  "'controls': ['first', 'play', 'loop', 'speed', 'delayMin': 0")

FUN3(TRUE)

## Constraining between -1 & 1 ##

## The old behavior of polarity constrained the output to be between -1 and 1
## this can be replicated via the 'constrain = TRUE' argument:

polarity("really hate anger")
polarity("really hate anger", constrain=TRUE)

# Static Network #

(poldat <- with(sentSplit(DATA, 4), polarity(state, person)))
m <- Network(poldat)
m

print(m, bg="grey97", vertex.color="grey75")

print(m, title="Polarity Discourse Map", title.color="white", bg="black",
  legend.text.color="white", vertex.label.color = "grey70",
  edge.label.color="yellow")
```
```r
## or use themes:
dev.off()
m + qtheme()
m + theme_nightheat
dev.off()
m+ theme_nightheat(title="Polarity Discourse Map")

### CUMULATIVE POLARITY EXAMPLE ###
### CUMULATIVE POLARITY EXAMPLE ###
# Hedonometrics#
# Hedonometrics#
poldt4 <- with(rajSPLIT, polarity(dialogue, act, constrain = TRUE))
polcount <- na.omit(counts(poldt4)$polarity)
len <- length(polcount)
cummean <- function(x){cumsum(x)/seq_along(x)}
cumpolarity <- data.frame(cum_mean = cummean(polcount), Time=1:len)

## Calculate background rectangles
ends <- cumsum(rle(counts(poldt4)$act)$lengths)
starts <- c(1, head(ends + 1, -1))
rects <- data.frame(xstart = starts, xend = ends + 1,
Act = c("I", "II", "III", "IV", "V")

library(ggplot2)
ggplot() + theme_bw() +
geom_rect(data = rects, aes(xmin = xstart, xmax = xend,
    ymin = -Inf, ymax = Inf, fill = Act), alpha = 0.17) +
geom_smooth(data = cumpolarity, aes(y=cum_mean, x = Time)) +
geom_hline(y=mean(polcount), color="grey30", size=1, alpha=.3, linetype=2) +
annotate("text", x = mean(ends[1:2]), y = mean(polcount), color="grey30",
    label = "Average Polarity", vjust = .3, size=3) +
geom_line(data = cumpolarity, aes(y=cum_mean, x = Time), size=1) +
ylab("Cumulative Average Polarity") + xlab("Duration") +
scale_x_continuous(expand = c(0,0)) +
geom_text(data=rects, aes(x=(xstart + xend)/2, y=-.04,
    label=paste("Act", Act)), size=3) +
guides(fill=FALSE) +
scale_fill_brewer(palette="Set1")

## End(Not run)
```

### Description

**pos**  
*Parts of Speech Tagging*

**pos** - Apply part of speech tagger to transcript(s).
pos_by - Apply part of speech tagger to transcript(s) by zero or more grouping variable(s).
pos_tags - Useful for interpreting the parts of speech tags created by pos and pos_by.

Usage

pos(text.var, parallel = FALSE, cores = detectCores() / 2,
progress.bar = TRUE, na.omit = FALSE, digits = 1, percent = TRUE,
zero.replace = 0, gc.rate = 10)

pos_by(text.var, grouping.var = NULL, digits = 1, percent = TRUE,
zero.replace = 0, ...)

pos_tags(type = "pretty")

Arguments

text.var The text variable.
parallel logical. If TRUE attempts to run the function on multiple cores. Note that this
may not mean a speed boost if you have one core or if the data set is smaller as
the cluster takes time to create.
cores The number of cores to use if parallel = TRUE. Default is half the number of
available cores.
progress.bar logical. If TRUE attempts to provide a OS appropriate progress bar. If parallel
is TRUE this argument is ignored. Note that setting this argument to TRUE may
slow down the function.
na.omit logical. If TRUE missing values (NA) will be omitted.
digits Integer; number of decimal places to round when printing.
percent logical. If TRUE output given as percent. If FALSE the output is proportion.
zero.replace Value to replace 0 values with.
gc.rate An integer value. This is a necessary argument because of a problem with the
garbage collection in the openNLP function that pos wraps. Consider adjusting
this argument upward if the error java.lang.OutOfMemoryError occurs.

Arguments supplied to pos.

type An optional character string giving the output of the pos tags. This must be
one of the strings "pretty" (a left justified version of the output optimized
for viewing but not good for export), "matrix" (a matrix version of the output),
"dataframe"\"\df\" (a dataframe version of the output), "all" (a list of all three
of the previous output types).

Value

pos - returns a list of 4:

text The original text
POSStagged: The original words replaced with parts of speech in context.
POSprop: Dataframe of the proportion of parts of speech by row.
POSfreq: Dataframe of the frequency of parts of speech by row.
POSrnp: Dataframe of the frequency and proportions of parts of speech by row.
percent: The value of percent used for plotting purposes.
zero.replace: The value of zero.replace used for plotting purposes.

pos_by - returns a list of 6:
text: The original text
POSStagged: The original words replaced with parts of speech in context.
POSprop: Dataframe of the proportion of parts of speech by row.
POSfreq: Dataframe of the frequency of parts of speech by row.
POSrnp: Dataframe of the frequency and proportions of parts of speech by row.
pos.by.prop: Dataframe of the proportion of parts of speech by grouping variable.
pos.by.freq: Dataframe of the frequency of parts of speech by grouping variable.
pos.by.rnp: Dataframe of the frequency and proportions of parts of speech by grouping variable.
percent: The value of percent used for plotting purposes.
zero.replace: The value of zero.replace used for plotting purposes.

Note
Note that contractions are treated as two words; for example the word count on "what's" is 2 for "what + is". This is not consistent with the word_count treatment of contractions but makes sense in a part of speech framework where a phrase such as "She’s cool" is treated as a pronoun, verb and adjective respectively for "She + is + cool".

References
http://opennlp.apache.org

See Also
Maxent_POS_Tag_Annotator, colcomb2class

Examples
```r
## Not run:
posdat <- pos(DATA$state)
ltruncdf(posdat, 7, 4)
## str(posdat)
names(posdat)
posdat$text         #original text
## Methods
```
preprocessed(posdat)  # words replaced with parts of speech
counts(posdat)  # frequency of parts of speech by row
proportions(posdat)  # proportion of parts of speech by row

## Methods Plotting
plot(preprocessed(posdat))
plot(counts(posdat))
plot(proportions(posdat))
plot(posdat)

out1 <- pos(DATA$state, parallel = TRUE)  # not always useful
ltruncdf(out1, 7, 4)

# use pos_tags to interpret part of speech tags used by pos & pos_by
pos_tags()[[1:10, ]]
pos_tags("matrix")[[1:10, ]]
pos_tags("dataframe")[[1:10, ]]
pos_tags("df")[[1:10, ]]
ltruncdf(pos_tags("all"), 3)

posbydat <- with(DATA, pos_by(state, sex))
names(posbydat)

## Methods
scores(posbydat)
preprocessed(posbydat)
counts(posbydat)
proportions(posbydat)

## Methods Plotting
plot(preprocessed(posbydat))
plot(counts(posbydat))
plot(proportions(posbydat))
plot(posbydat)

ltruncdf(posbydat, 7, 4)
truncdf(posbydat$pos.by.prop, 4)

POSby <- with(DATA, pos_by(state, list(adult, sex)))
plot(POSby, values = TRUE, digits = 2)
# for more quickly - reuse the output from before
out2 <- with(DATA, pos_by(posbydat, list(adult, sex)))

## Definite/Indefinite Noun
## 2 approached compared...
## The later is more efficient but less accurate

#-----------------------------------#
## Part off speech tagging ##
#-----------------------------------#
pos_after <- function(text.var, words, pos){
  posses <- strsplit(as.character(text.var[["POStagged"]][["POStagged"]]), "\s+")
namespos <- lapply(posses, function(x) {
  y <- unlist(strsplit(x, "/"))
  setNames(y[c(TRUE, FALSE)], y[c(FALSE, TRUE)])
})

lapply(namespos, function(x, thewords = words, thepos = pos)(
  locs <- which(x %in% thewords)
  locs <- locs[!is.na(locs)]
  if (identical(unclass(locs), integer(0))) return(NA_character_)

  nounlocs <- which(names(x) %in% thepos)

  unname(x[unique(sapply(locs, function(x){
    min(nounlocs[nounlocs - x > 0])
  })))
})

out2 <- setNames(lapply(list(a=c("a", "an"), the="the"), function(x) {
  o <- pos_after(rajPOS, x, c("NN", "NNS", "NNP", "NNPS"))
  m <- stats::setNames(data.frame(sort(table(unlist(o)))),
    stringsAsFactors = FALSE), c("word", "freq"))
  m[m$freq > 3, ]), c("a", "the")
})

dat2 <- setNames(Reduce(function(x, y) {
  merge(x, y, by = "word", all = TRUE), out2), c("Word", "A", "THE"))

dat2 <- reshape2::melt(dat2, id="Word", variable.name="Article", value.name="freq")

dat2 <- dat2[order(dat2$freq, dat2$Word), ]

ord2 <- aggregate(freq ~ Word, dat2, sum)

dat2$Word <- factor(dat2$Word, levels=ord2[order(ord2[[2]]), 1])

rownames(dat2) <- NULL

ggplot(dat2, aes(x=freq, y=Word)) +
  geom_point()+ facet_grid(~Article) +
  ggtitle("Part Of Speech Parsing Approach")

dev.new()

## --------------------------#
## Regular Expressions ##
## --------------------------#

library(qdapRegex);library(ggplot2);library(reshape2)

out <- setNames(lapply(c("@after_a", "@after_the"), function(x) {
  o <- rm_default(stringi::stri_trans_tolower(raj$d_1 dialogue),
    pattern = x, extract=TRUE)
potential_NA

Search for Potential Missing Values

Description

Search for potential missing values (i.e., sentences that are merely a punctuation mark) and optionally replace with missing value (NA). Useful in the initial cleaning process.

Usage

```
potential_NA(text.var, n = 3)
```

Arguments

- **text.var**: The text variable.
- **n**: Number of characters to consider for missing (default is 3).

Value

Returns a dataframe of potential missing values row numbers and text.

Examples

```
## Not run:
DATA$state[c(3, 7)] <- " ."
potential_NA(DATA$state, 20)
potential_NA(DATA$state)
```
# USE TO SELECTIVELY REPLACE CELLS WITH MISSING VALUES
DATA$state[potential_na(DATA$state, 20)$row[-c(3)]] <- NA
DATA
DATA <- qdap::DATA

## End(Not run)

---

### preprocessed

**Generic Preprocessed Method**

**Description**

Access the preprocessed dataframes/lists from select qdap outputs.

**Usage**

`preprocessed(x, ...)`

**Arguments**

- `x`:
  - A qdap object (list) with a dataframe/list of preprocessed data (e.g., `pos_by`).
- `...`:
  - Arguments passed to preprocessed method of other classes.

**Value**

Returns a data.frame or list of preprocessed data.

**See Also**

`scores, counts, proportions, visual`

---

### preprocessed.check_spelling_interactive

**Check Spelling**

**Description**

View check_spelling_interactive preprocessed.

**Usage**

```
## S3 method for class 'check_spelling_interactive'
preprocessed(x, ...)
```
Arguments

x  The `check_spelling_interactive` object.

... ignored

Details

`check_spelling_interactive` Method for `preprocessed`

---

`preprocessed.end_mark_by`

**Question Counts**

---

**Description**

View `end_mark_by` `preprocessed`.

**Usage**

```r
## S3 method for class 'end_mark_by'
preprocessed(x, ...)
```

**Arguments**

x  The `end_mark_by` object.

... ignored

**Details**

`end_mark_by` Method for `preprocessed`

---

`preprocessed.formality`

**Formality**

---

**Description**

View `formality` `preprocessed`.

**Usage**

```r
## S3 method for class 'formality'
preprocessed(x, ...)```
Arguments

x  The `formality` object.

... ignored

Details

formality Method for preprocessed.

---

`preprocessed.lexical_classification`

*Lexical Classification*

Description

`preprocessed.lexical_classification` - View preprocessed from `lexical_classification`.

Usage

```r
## S3 method for class 'lexical_classification'
preprocessed(x, ...)
```

Arguments

x  The `lexical_classification` object.

... ignored

Details

`lexical_classification` Method for preprocessed.

---

`preprocessed.object_pronoun_type`

*Question Counts*

Description

View `object_pronoun_type` preprocessed.

Usage

```r
## S3 method for class 'object_pronoun_type'
preprocessed(x, ...)
```
**preprocessed.pos**

**Arguments**

- **x** The `object_pronoun_type` object.
- **...** ignored

**Details**

- `object_pronoun_type` Method for preprocessed

---

<table>
<thead>
<tr>
<th>preprocessed.pos</th>
<th>Parts of Speech</th>
</tr>
</thead>
</table>

**Description**

View `pos` preprocessed.

**Usage**

```r
## S3 method for class 'pos'
preprocessed(x, ...)
```

**Arguments**

- **x** The `pos` object.
- **...** ignored

**Details**

- `pos` Method for preprocessed

---

<table>
<thead>
<tr>
<th>preprocessed.pos_by</th>
<th>Parts of Speech</th>
</tr>
</thead>
</table>

**Description**

View `pos_by` preprocessed.

**Usage**

```r
## S3 method for class 'pos_by'
preprocessed(x, ...)
```

**Arguments**

- **x** The `pos_by` object.
- **...** ignored
Details

pos_by Method for preprocessed

---

preprocessed.pronoun_type

*Question Counts*

---

**Description**

View `pronoun_type` preprocessed.

**Usage**

```r
## S3 method for class 'pronoun_type'
preprocessed(x, ...)
```

**Arguments**

- `x` The `pronoun_type` object.
- `...` ignored

Details

`pronoun_type` Method for preprocessed

---

preprocessed.question_type

*Question Counts*

---

**Description**

View `question_type` preprocessed.

**Usage**

```r
## S3 method for class 'question_type'
preprocessed(x, ...)
```

**Arguments**

- `x` The `question_type` object.
- `...` ignored

Details

`question_type` Method for preprocessed
**preprocessed.subject_pronoun_type**

*Question Counts*

---

**Description**

View `subject_pronoun_type` preprocessed.

**Usage**

```r
## S3 method for class 'subject_pronoun_type'
preprocessed(x, ...)
```

**Arguments**

- `x` : The `subject_pronoun_type` object.
- `...` : ignored

**Details**

`subject_pronoun_type` Method for `preprocessed`.

---

**preprocessed.word_position**

*Word Position*

---

**Description**

View `word_position` preprocessed.

**Usage**

```r
## S3 method for class 'word_position'
preprocessed(x, ...)
```

**Arguments**

- `x` : The `word_position` object.
- `...` : ignored

**Details**

`word_position` Method for `preprocessed`.
Description

A dataset containing a cleaned version of all three presidential debates for the 2012 election.

Usage

data(pres_debates2012)

Format

A data frame with 2912 rows and 4 variables

Details

- person. The speaker
- tot. Turn of talk
- dialogue. The words spoken
- time. Variable indicating which of the three debates the dialogue is from

Description

A dataset containing the raw version of the first presidential debate.

Usage

data(pres_debate_raw2012)

Format

A data frame with 94 rows and 2 variables

Details

- person. The speaker
- dialogue. The words spoken
**print.adjacency_matrix**

*Prints an adjacency_matrix Object*

**Description**

Prints an adjacency_matrix object.

**Usage**

```r
## S3 method for class 'adjacency_matrix'
print(x, ...)
```

**Arguments**

- `x` The adjacency_matrix object.
- `...` ignored

---

**print.all_words**  
*Prints an all_words Object*

**Description**

Prints an all_words object.

**Usage**

```r
## S3 method for class 'all_words'
print(x, ...)
```

**Arguments**

- `x` The all_words object.
- `...` ignored
print.animated_character

*Prints an animated_character Object*

**Description**

Prints an animated_character object.

**Usage**

```r
## S3 method for class 'animated_character'
print(x, pause = 0, ...)
```

**Arguments**

- `x` The animated_character object.
- `pause` The length of time to pause between plots.
- `...` ignored.

print.animated_discourse_map

*Prints an animated_discourse_map Object*

**Description**

Prints an animated_discourse_map object.

**Usage**

```r
## S3 method for class 'animated_discourse_map'
print(x, title = NULL,
      seed = sample(1:10000, 1), layout = layout.auto, pause = 0, ...)
```

**Arguments**

- `x` The animated_discourse_map object.
- `title` The title of the plot.
- `seed` The seed to use in plotting the graph.
- `layout` *igraph* layout to use.
- `pause` The length of time to pause between plots.
- `...` Other Arguments passed to `plot.igraph`. 
print.animated_formality

Prints a animated_formality Object

Description

Prints a animated_formality object.

Usage

```r
## S3 method for class 'animated_formality'
print(x, title = NULL, seed = sample(1:10000, 1), layout = layout.auto, pause = 0, legend = c(-0.5, -1.5, 0.5, -1.45), legend.cex = 1, bg = NULL, net.legend.color = "black", ...)
```

Arguments

- `x` The animated_formality object.
- `title` The title of the plot.
- `seed` The seed to use in plotting the graph.
- `layout` igraph layout to use.
- `pause` The length of time to pause between plots.
- `legend` The coordinates of the legend. See `color.legend` for more information.
- `legend.cex` character expansion factor. NULL and NA are equivalent to 1.0. See `mtext` for more information.
- `bg` The color to be used for the background of the device region. See `par` for more information.
- `net.legend.color` The text legend color for the network plot.
- `...` Other Arguments passed to `plot.igraph`.

print.animated_lexical_classification

Prints an animated_lexical_classification Object

Description

Prints an animated_lexical_classification object.
print.animated_polarity

Usage

## S3 method for class 'animated_lexical_classification'
print(x, title = NULL,
      seed = sample(1:10000, 1), layout = layout.auto, pause = 0,
      legend = c(-0.5, -1.5, 0.5, -1.45), legend.cex = 1, bg = NULL,
      net.legend.color = "black", ...)

Arguments

x The animated_lexical_classification object.
title The title of the plot.
seed The seed to use in plotting the graph.
layout igraph layout to use.
pause The length of time to pause between plots.
legend The coordinates of the legend. See color.legend for more information.
legend.cex character expansion factor. NULL and NA are equivalent to 1.0. See mtext for more information.
bg The color to be used for the background of the device region. See par for more information.
net.legend.color The text legend color for the network plot.
... Other Arguments passed to plot.igraph.

print.animated_polarity

Prints an animated_polarity Object

Description

Prints an animated_polarity object.

Usage

## S3 method for class 'animated_polarity'
print(x, title = NULL, seed = sample(1:10000,
      1), layout = layout.auto, pause = 0, legend = c(-0.5, -1.5, 0.5, -1.45),
      legend.cex = 1, bg = NULL, net.legend.color = "black", ...)
Arguments

x  The automated_readability_index object.
title  The title of the plot.
seed  The seed to use in plotting the graph.
layout  igraph layout to use.
pause  The length of time to pause between plots.
legend  The coordinates of the legend. See color.legend for more information.
legend.cex  character expansion factor. NULL and NA are equivalent to 1.0. See mtext for more information.
bg  The color to be used for the background of the device region. See par for more information.
net.legend.color  The text legend color for the network plot.
...  Other Arguments passed to plot.igraph.

print.automated_readability_index

Prints an automated_readability_index Object

Description

Prints an automated_readability_index object.

Usage

## S3 method for class 'automated_readability_index'
print(x, digits = 3, ...)

Arguments

x  The automated_readability_index object.
digits  The number of digits displayed if values is TRUE.
...  ignored
print.boolean_qdap  \hspace{1cm} \textit{Prints a boolean_qdap object}

\textbf{Description}

Prints a boolean_qdap object

\textbf{Usage}

\begin{verbatim}
## S3 method for class 'boolean_qdap'
print(x, ...)
\end{verbatim}

\textbf{Arguments}

- \texttt{x} \hspace{1cm} The boolean_qdap object
- \texttt{...} \hspace{1cm} ignored

---

print.character_table  \hspace{1cm} \textit{Prints a character_table object}

\textbf{Description}

Prints a character_table object.

\textbf{Usage}

\begin{verbatim}
## S3 method for class 'character_table'
print(x, digits = 2, percent = NULL,
       zero.replace = NULL, ...)
\end{verbatim}

\textbf{Arguments}

- \texttt{x} \hspace{1cm} The character_table object
- \texttt{digits} \hspace{1cm} Integer values specifying the number of digits to be printed.
- \texttt{percent} \hspace{1cm} logical. If TRUE output given as percent. If FALSE the output is proportion. If NULL uses the value from termco. Only used if label is TRUE.
- \texttt{zero.replace} \hspace{1cm} Value to replace 0 values with. If NULL uses the value from termco. Only used if label is TRUE.
- \texttt{...} \hspace{1cm} ignored
print.check_spelling  
*Prints a check_spelling Object*

**Description**

Prints a check_spelling object.

**Usage**

```r
## S3 method for class 'check_spelling'
print(x, ...)
```

**Arguments**

- `x`  
  The check_spelling object.
- `...`  
  ignored

---

print.check_spelling_interactive  
*Prints a check_spelling_interactive Object*

**Description**

Prints a check_spelling_interactive object.

**Usage**

```r
## S3 method for class 'check_spelling_interactive'
print(x, ...)
```

**Arguments**

- `x`  
  The check_spelling_interactive object.
- `...`  
  ignored
print.check_text

Prints a check_text Object

Description

Prints a check_text object.

Usage

```r
## S3 method for class 'check_text'
print(x, include.text = TRUE, file = NULL, ...)
```

Arguments

- **x**: The check_text object.
- **include.text**: logical. If TRUE the offending text is printed as well.
- **file**: A connection, or a character string naming the file to print to. If NULL prints to the console.
- **...**: ignored

print.cm_distance

Prints a cm_distance Object

Description

Prints a cm_distance object.

Usage

```r
## S3 method for class 'cm_distance'
print(x, mean.digits = 0, sd.digits = 2,
     sd.mean.digits = 3, pval.digits = 3, new.order = NULL,
     na.replace = "-", diag.replace = na.replace, print = TRUE, ...)
```

Arguments

- **x**: The cm_distance object.
- **mean.digits**: The number of digits to print for the mean code distances.
- **sd.digits**: The number of digits to print for the standard deviations of the code distances.
- **sd.mean.digits**: The number of digits to print for the standardized mean distances.
- **pval.digits**: The number of digits to print for the p-values.
- **new.order**: An integer vector reordering the columns and rows of the output. Omission of a column number will result in omission from the output.
print.coleman_liau

**Description**

Prints an coleman_liau object.

**Usage**

```r
## S3 method for class 'coleman_liau'
print(x, digits = 3, ...)
```

**Arguments**

- `x` The coleman_liau object.
- `digits` The number of digits displayed if `values` is `TRUE`.
- `...` ignored

print.colsplit2df

**Description**

Prints a colsplit2df object.

**Usage**

```r
## S3 method for class 'colsplit2df'
print(x, ...) 
```

**Arguments**

- `x` The colsplit2df object
- `...` ignored
print.combo_syllable_sum

*Prints an combo_syllable_sum object*

**Description**

Prints an combo_syllable_sum object

**Usage**

```r
## S3 method for class 'combo_syllable_sum'
print(x, ...)
```

**Arguments**

- `x` The combo_syllable_sum object
- `...` ignored

print.cumulative_animated_formality

*Prints a cumulative_animated_formality Object*

**Description**

Prints a cumulative_animated_formality object.

**Usage**

```r
## S3 method for class 'cumulative_animated_formality'
print(x, ...)
```

**Arguments**

- `x` The cumulative_animated_formality object.
- `...` ignored
print.cumulative_animated_lexical_classification

*Prints a cumulative_animated_lexical_classification Object*

**Description**

Prints a cumulative_animated_lexical_classification object.

**Usage**

```r
## S3 method for class 'cumulative_animated_lexical_classification'
print(x, ...)
```

**Arguments**

- `x` The cumulative_animated_lexical_classification object.
- `...` ignored

---

print.cumulative_animated_polarity

*Prints a cumulative_animated_polarity Object*

**Description**

Prints a cumulative_animated_polarity object.

**Usage**

```r
## S3 method for class 'cumulative_animated_polarity'
print(x, ...)
```

**Arguments**

- `x` The cumulative_animated_polarity object.
- `...` ignored
print.cumulative_combo_syllable_sum

Prints a cumulative_combo_syllable_sum Object

Description

Prints a cumulative_combo_syllable_sum object.

Usage

```r
## S3 method for class 'cumulative_combo_syllable_sum'
print(x, ...)
```

Arguments

- `x`: The cumulative_combo_syllable_sum object.
- `...`: ignored

print.cumulative_end_mark

Prints a cumulative_end_mark Object

Description

Prints a cumulative_end_mark object.

Usage

```r
## S3 method for class 'cumulative_end_mark'
print(x, ...)
```

Arguments

- `x`: The cumulative_end_mark object.
- `...`: ignored
print.cumulative_formality

Prints a cumulative_formality Object

Description

Prints a cumulative_formality object.

Usage

```r
## S3 method for class 'cumulative_formality'
print(x, ...)
```

Arguments

- `x` The cumulative_formality object.
- `...` ignored

print.cumulative_lexical_classification

Prints a cumulative_lexical_classification Object

Description

Prints a cumulative_lexical_classification object.

Usage

```r
## S3 method for class 'cumulative_lexical_classification'
print(x, ...)
```

Arguments

- `x` The cumulative_lexical_classification object.
- `...` ignored
print.cumulative_polarity

*Prints a cumulative_polarity Object*

**Description**

Prints a cumulative_polarity object.

**Usage**

```r
## S3 method for class 'cumulative_polarity'
print(x, ...)
```

**Arguments**

- `x` The cumulative_polarity object.
- `...` ignored

---

print.cumulative_syllable_freq

*Prints a cumulative_syllable_freq Object*

**Description**

Prints a cumulative_syllable_freq object.

**Usage**

```r
## S3 method for class 'cumulative_syllable_freq'
print(x, ...)
```

**Arguments**

- `x` The cumulative_syllable_freq object.
- `...` ignored
print.discourse_map  
*Prints a discourse_map Object*

**Description**

Prints a discourse_map object.

**Usage**

```r
## S3 method for class 'discourse_map'
print(x, edge.curved = TRUE, title = NULL, ...)
```

**Arguments**

- `x`: The discourse_map object.
- `edge.curved`: logical. If TRUE edges are plotted with curves.
- `title`: The title of the plot.
- `...`: Other Arguments passed to `plot.igraph`.

---

print.Dissimilarity  
*Prints a Dissimilarity object*

**Description**

Prints a Dissimilarity object.

**Usage**

```r
## S3 method for class 'Dissimilarity'
print(x, digits = 3, ...)
```

**Arguments**

- `x`: The Dissimilarity object
- `digits`: Number of decimal places to print.
- `...`: ignored
**print.diversity** *Prints a diversity object*

**Description**

Prints a diversity object.

**Usage**

```r
## S3 method for class 'diversity'
print(x, digits = 3, ...)
```

**Arguments**

- `x` The diversity object
- `digits` Number of decimal places to print.
- `...` ignored

---

**print.end_mark** *Prints an end_mark object*

**Description**

Prints an end_mark object.

**Usage**

```r
## S3 method for class 'end_mark'
print(x, ...)  
```

**Arguments**

- `x` The end_mark object
- `...` ignored
**print.end_mark_by**

Prints an end_mark_by object

---

**Description**

Prints an end_mark_by object

**Usage**

```r
## S3 method for class 'end_mark_by'
print(x, ...)
```

**Arguments**

- `x` The end_mark_by object
- `...` ignored

---

**print.end_mark_by_preprocessed**

Prints a end_mark_by_preprocessed object

---

**Description**

Prints a end_mark_by_preprocessed object

**Usage**

```r
## S3 method for class 'end_mark_by_preprocessed'
print(x, ...)
```

**Arguments**

- `x` The end_mark_by_preprocessed object
- `...` ignored
print.flesch_kincaid  

**Prints an flesch_kincaid Object**

**Description**

Prints an flesch_kincaid object.

**Usage**

```r
## S3 method for class 'flesch_kincaid'
print(x, digits = 3, ...)
```

**Arguments**

- `x`  
The flesch_kincaid object.
- `digits`  
The number of digits displayed if values is TRUE.
- `...`  
  ignored

print.formality  

**Prints a formality Object**

**Description**

Prints a formality object.

**Usage**

```r
## S3 method for class 'formality'
print(x, digits, ...)
```

**Arguments**

- `x`  
The formality object.
- `digits`  
The number of digits to print.
- `...`  
  ignored
print.formality_scores

*Prints a formality_scores object*

**Description**

Prints a formality_scores object

**Usage**

```r
## S3 method for class 'formality_scores'
print(x, ...)  
```

**Arguments**

- `x` The formality_scores object
- `...` ignored

---

print.fry

*Prints an fry Object*

**Description**

Prints an fry object.

**Usage**

```r
## S3 method for class 'fry'
print(x, digits = 3, auto.label, grid, div.col, plot, ...)  
```

**Arguments**

- `x` The fry object.
- `digits` The number of digits displayed if values is TRUE.
- `auto.label` logical. If TRUE labels automatically added. If FALSE the user clicks interactively.
- `grid` logical. If TRUE a micro grid is displayed similar to Fry’s original depiction, though this makes visualizing more difficult.
- `div.col` The color of the grade level division lines.
- `plot` logical. If TRUE a graph is plotted corresponding to Fry’s graphic representation.
- `...` ignored
print.inspect_text

Prints an inspect_text Object

Description

Prints an inspect_text object.

Usage

```r
## S3 method for class 'inspect_text'
print(x, file = "", ...)  # prints to the standard output
```

Arguments

- `x`: The inspect_text object.
- `file`: A connection, or a character string naming the file to print to. If "" (the default), prints to the standard output connection, the console unless redirected by `sink`.
- `...`: Other arguments passed to `strwrap`.

print.kullback_leibler

Prints a kullback_leibler Object.

Description

Prints a kullback_leibler object.

Usage

```r
## S3 method for class 'kullback_leibler'
print(x, digits = 3, ...)  # prints to the standard output
```

Arguments

- `x`: The kullback_leibler object
- `digits`: Number of decimal places to print.
- `...`: Ignored
print.lexical_classification

Prints an lexical_classification Object

Description

Prints an lexical_classification object.

Usage

```r
## S3 method for class 'lexical_classification'
print(x, ...)
```

Arguments

- `x` The lexical_classification object.
- `...` Other arguments passed to `print.lexical_classification_by`.

print.lexical_classification_by

Prints a lexical_classification Object

Description

Prints a lexical_classification_by object.

Usage

```r
## S3 method for class 'lexical_classification_by'
print(x, ave.digits = 1, se.digits = 2, trunc = 25, ...)
```

Arguments

- `x` The lexical_classification_by object.
- `ave.digits` The number of average lexical distribution proportion digits to print.
- `se.digits` The number of standard error of the lexical distribution proportion digits to print.
- `trunc` The width to truncate content/function word lists.
- `...` ignored
print.lexical_classification_preprocessed

*Prints a lexical_classification_preprocessed Object*

---

**Description**

Prints a lexical_classification_preprocessed object.

**Usage**

```r
## S3 method for class 'lexical_classification_preprocessed'
print(x, ...)
```

**Arguments**

- `x` The lexical_classification_preprocessed object.
- `...` ignored

---

print.lexical_classification_score

*Prints a lexical_classification_score Object*

---

**Description**

Prints a lexical_classification_score object.

**Usage**

```r
## S3 method for class 'lexical_classification_score'
print(x, digits = 3, ...)
```

**Arguments**

- `x` The lexical_classification_score object.
- `digits` The number of digits displayed if values is TRUE.
- `...` ignored
print.linsear_write

*Prints an linsear_write Object*

**Description**

Prints an linsear_write object.

**Usage**

```r
## S3 method for class 'linsear_write'
print(x, digits = 3, ...)
```

**Arguments**

- `x` The linsear_write object.
- `digits` The number of digits displayed if values is TRUE.
- `...` ignored

---

print.linsear_write_count

*Prints a linsear_write_count Object*

**Description**

Prints a linsear_write_count object.

**Usage**

```r
## S3 method for class 'linsear_write_count'
print(x, digits = 3, ...)
```

**Arguments**

- `x` The linsear_write_count object.
- `digits` The number of digits displayed.
- `...` ignored
print.linsear_write_scores

*Prints a linsear_write_scores Object*

**Description**

Prints a linsear_write_scores object.

**Usage**

```r
## S3 method for class 'linsear_write_scores'
print(x, digits = 3, ...)
```

**Arguments**

- `x`: The linsear_write_scores object.
- `digits`: The number of digits displayed.
- `...`: ignored

---

print.Network

*Prints a Network Object*

**Description**

Prints a Network object.

**Usage**

```r
## S3 method for class 'Network'
print(x, title = NA, title.color = "black",
       seed = sample(1:10000, 1), layout = igraph::layout.auto,
       legend = c(-0.5, -1.5, 0.5, -1.45), legend.cex = 1, bg = NULL,
       legend.text.color = "black", legend.gradient = NULL,
       vertex.color = "grey80", vertex.size = 9, vertex.frame.color = NA,
       vertex.label.color = "grey40", vertex.label.cex = 1.1,
       edge.label.color = "black", edge.label.cex = 0.9, ...)
```

**Arguments**

- `x`: The Network object.
- `title`: The title of the plot. NULL eliminates title. NA uses title attribute of the Network object.
- `title.color`: The color of the title.
- `seed`: The seed to use in plotting the graph.
print.ngrams

Description
Prints an ngrams object

Usage
```r
## S3 method for class 'ngrams'
print(x, ...)
```

Arguments
- `x` The ngrams object
- `...` ignored
print.object_pronoun_type

Prints a object_pronoun_type object

Description

Prints a object_pronoun_type object

Usage

```r
## S3 method for class 'object_pronoun_type'
print(x, ...)
```

Arguments

- `x`: The object_pronoun_type object
- `...`: ignored

print.phrase_net

Prints a phrase_net Object

Description

Prints a phrase_net object.

Usage

```r
## S3 method for class 'phrase_net'
print(x, edge.curved = TRUE, ...)
```

Arguments

- `x`: The phrase_net object.
- `edge.curved`: logical. If TRUE edges are plotted with curves.
- `...`: Other Arguments passed to `plot.igraph`.
**print.polarity**  
*Prints an polarity Object*

**Description**

Prints an polarity object.

**Usage**

```r
## S3 method for class 'polarity'
print(x, digits = 3, ...)
```

**Arguments**

- `x`: The polarity object.
- `digits`: The number of digits displayed if `values` is `TRUE`.
- `...`: ignored

---

**print.polarity_count**  
*Prints a polarity_count Object*

**Description**

Prints a polarity_count object.

**Usage**

```r
## S3 method for class 'polarity_count'
print(x, digits = 3, ...)
```

**Arguments**

- `x`: The polarity_count object.
- `digits`: The number of digits displayed.
- `...`: ignored
print.polarity_score  *Prints a polarity_score Object*

**Description**

Prints a polarity_score object.

**Usage**

```r
## S3 method for class 'polarity_score'
print(x, digits = 3, ...)
```

**Arguments**

- `x` The polarity_score object.
- `digits` The number of digits displayed if values is TRUE.
- `...` ignored

print.polysyllable_sum  *Prints an polysyllable_sum object*

**Description**

Prints an polysyllable_sum object

**Usage**

```r
## S3 method for class 'polysyllable_sum'
print(x, ...)}
```

**Arguments**

- `x` The polysyllable_sum object
- `...` ignored
print.pos

*Prints a pos Object.*

**Description**

Prints a pos object.

**Usage**

```r
## S3 method for class 'pos'
print(x, digits = 1, percent = NULL, zero.replace = NULL, 
      ...)  
```

**Arguments**

- `x` The pos object
- `digits` Integer values specifying the number of digits to be printed.
- `percent` logical. If TRUE output given as percent. If FALSE the output is proportion. If NULL uses the value from `termco`. Only used if `label` is TRUE.
- `zero.replace` Value to replace 0 values with. If NULL uses the value from `termco`. Only used if `label` is TRUE.
- `...` ignored

print.pos_by

*Prints a pos_by Object.*

**Description**

Prints a pos_by object.

**Usage**

```r
## S3 method for class 'pos_by'
print(x, digits = 1, percent = NULL, zero.replace = NULL, 
      ...)  
```

**Arguments**

- `x` The pos_by object
- `digits` Integer values specifying the number of digits to be printed.
- `percent` logical. If TRUE output given as percent. If FALSE the output is proportion. If NULL uses the value from `termco`. Only used if `label` is TRUE.
- `zero.replace` Value to replace 0 values with. If NULL uses the value from `termco`. Only used if `label` is TRUE.
- `...` ignored
Description

Prints a pos_preprocessed object

Usage

```r
## S3 method for class 'pos_preprocessed'
print(x, ...)
```

Arguments

- `x`: The pos_preprocessed object
- `...`: ignored

Description

Prints a pronoun_type object

Usage

```r
## S3 method for class 'pronoun_type'
print(x, ...)
```

Arguments

- `x`: The pronoun_type object
- `...`: ignored
print.qdapProj

Prints a qdapProj Object

Description
Prints a qdapProj object.

Usage

```r
## S3 method for class 'qdapProj'
print(x, ...)
```

Arguments

- `x` The qdapProj object.
- `...` ignored

print.qdap_context

Prints a qdap_context object

Description
Prints a qdap_context object

Usage

```r
## S3 method for class 'qdap_context'
print(x, file = NULL, pretty = TRUE, width = 70,
      sep.block = TRUE, double_space = TRUE, ...)
```

Arguments

- `x` The qdap_context object
- `file` The name of the file (can print csv, xlsx, txt, doc and other text based files). If NULL file prints to the console.
- `pretty` logical. If TRUE generates a prettier text version of the output (cannot be used with csv/xlsx file types). If FALSE a semi-structured dataframe is generated.
- `width` A positive integer giving the target column for wrapping lines in the output.
- `sep.block` logical. If TRUE the blocked events are separated with text lines.
- `double_space` logical. If TRUE and pretty = TRUE double spacing between speech chunks (speakers) is used.
- `...` ignored
print.question_type Preprocessed

Description
Prints a question_type object

Usage
```r
## S3 method for class 'question_type'
print(x, ...)
```

Arguments
- `x` The question_type object
- `...` ignored

print.question_type_preprocessed

Description
Prints a question_type_preprocessed object

Usage
```r
## S3 method for class 'question_type_preprocessed'
print(x, ...)
```

Arguments
- `x` The question_type_preprocessed object
- `...` ignored
**print.readability_count**

*Prints a readability_count Object*

**Description**

Prints a readability_count object.

**Usage**

```r
## S3 method for class 'readability_count'
print(x, digits = 3, ...)
```

**Arguments**

- `x` The readability_count object.
- `digits` The number of digits displayed.
- `...` ignored

**print.readability_score**

*Prints a readability_score Object*

**Description**

Prints a readability_score object.

**Usage**

```r
## S3 method for class 'readability_score'
print(x, digits = 3, ...)
```

**Arguments**

- `x` The readability_score object.
- `digits` The number of digits displayed if values is TRUE.
- `...` ignored
print.sent_split: Prints a sent_split object

Description
Prints a sent_split object

Usage

```r
## S3 method for class 'sent_split'
print(x, ...)
```

Arguments
- `x`: The sent_split object
- `...`: ignored

print.SMOG: Prints an SMOG Object

Description
Prints an SMOG object.

Usage

```r
## S3 method for class 'SMOG'
print(x, digits = 3, ...)
```

Arguments
- `x`: The SMOG object.
- `digits`: The number of digits displayed if `values` is TRUE.
- `...`: ignored
print.subject_pronoun_type

Prints a subject_pronoun_type object

Description
Prints a subject_pronoun_type object

Usage
```r
## S3 method for class 'subject_pronoun_type'
print(x, ...)
```

Arguments
- `x` The subject_pronoun_type object
- `...` ignored

print.sub_holder

Prints a sub_holder object

Description
Prints a sub_holder object

Usage
```r
## S3 method for class 'sub_holder'
print(x, ...)
```

Arguments
- `x` The sub_holder object
- `...` ignored
print.sums_gantt  

Prints a sums_gantt object

Description

Prints a sums_gantt object.

Usage

```r
## S3 method for class 'sums_gantt'
print(x, ...)
```

Arguments

- `x`  
The sums_gantt object
- `...`  
    ignored

print.sum_cmspans  

Prints a sum_cmspans object

Description

Prints a sum_cmspans object.

Usage

```r
## S3 method for class 'sum_cmspans'
print(x, digits = NULL, ...)
```

Arguments

- `x`  
The sum_cmspans object
- `digits`  
    Integer; number of decimal places to round in the display of the output.
- `...`  
    ignored
**print.syllable_sum**

Prints a syllable_sum object

**Description**

Prints a syllable_sum object

**Usage**

```r
## S3 method for class 'syllable_sum'
print(x, ...)
```

**Arguments**

- `x`  The syllable_sum object
- `...` ignored

---

**print.table_count**

Prints a table_count object

**Description**

Prints a table_count object

**Usage**

```r
## S3 method for class 'table_count'
print(x, ...)
```

**Arguments**

- `x`  The table_count object
- `...` ignored
Prints a table_proportion object

Usage

## S3 method for class 'table_proportion'
print(x, ...)

Arguments

x The table_proportion object
...
ignored

Prints a table_score object

Usage

## S3 method for class 'table_score'
print(x, ...)

Arguments

x The table_score object
...
ignored
print.termco

Prints a termco object.

Description

Prints a termco object.

Usage

```r
## S3 method for class 'termco'
print(x, digits = NULL, percent = NULL,
      zero.replace = NULL, ...)
```

Arguments

- **x**: The termco object
- **digits**: Integer values specifying the number of digits to be printed.
- **percent**: logical. If TRUE output given as percent. If FALSE the output is proportion. If NULL uses the value from `termco`. Only used if label is TRUE.
- **zero.replace**: Value to replace 0 values with. If NULL uses the value from `termco`. Only used if label is TRUE.
- **...**: ignored

print.trunc

Prints a trunc object

Description

Prints a trunc object

Usage

```r
## S3 method for class 'trunc'
print(x, ...)
```

Arguments

- **x**: The trunc object
- **...**: ignored
Description

Prints a type_token_ratio object.

Usage

```r
## S3 method for class 'type_token_ratio'
print(x, digits = 3, ...)
```

Arguments

- `x` : The type_token_ratio object.
- `digits` : The number of type-token ratio digits to print.
- `...` : ignored

Description

Prints a wfm object.

Usage

```r
## S3 method for class 'wfm'
print(x, digits = 3, width = 10000, ...)
```

Arguments

- `x` : The wfm object.
- `digits` : The number of digits displayed if values is TRUE.
- `width` : The width to temporarily set for printing (default = 10000). See `options` for more.
- `...` : ignored
print.wfm_summary

Prints a wfm_summary Object

Description

Prints a wfm_summary object.

Usage

## S3 method for class 'wfm_summary'
print(x, ...)

Arguments

x The wfm_summary object.

... ignored

print.which_misspelled

Prints a which_misspelled Object

Description

Prints a which_misspelled object.

Usage

## S3 method for class 'which_misspelled'
print(x, ...)

Arguments

x The which_misspelled object.

... ignored


print.word_associate  *Prints a word_associate object*

**Description**

Prints a word_associate object.

**Usage**

```r
## S3 method for class 'word_associate'
print(x, ...)
```

**Arguments**

- `x` : The word_associate object
- `...` : ignored

---

print.word_cor  *Prints a word_cor object*

**Description**

Prints a word_cor object

**Usage**

```r
## S3 method for class 'word_cor'
print(x, digits = 3, ...)
```

**Arguments**

- `x` : The word_cor object
- `digits` : The number of digits to print
- `...` : ignored
**print.word_length**

*Prints a word_length object*

---

**Description**

Prints a word_length object.

**Usage**

```r
## S3 method for class 'word_length'
print(x, ...)
```

**Arguments**

- `x` The word_length object
- `...` ignored

---

**print.word_list**

*Prints a word_list Object*

---

**Description**

Prints a word_list object.

**Usage**

```r
## S3 method for class 'word_list'
print(x, ...)
```

**Arguments**

- `x` The word_list object
- `...` ignored
print.word_position  Prints a word_position object.

Description
Prints a word_position object.

Usage

```r
top
```

Arguments

- **x**: The word_position object
- **...**: Values passed to `plot.word_position`

print.word_proximity  Prints a word_proximity object

Description
Prints a word_proximity object

Usage

```r
top
```

Arguments

- **x**: The word_proximity object
- **digits**: The number of digits to print
- **...**: ignored
print.word_stats

Prints a word_stats object

Description
Prints a word_stats object.

Usage

## S3 method for class 'word_stats'
print(x, digits = NULL, ...)

Arguments

x The word_stats object
digits Integer; number of decimal places to round in the display of the output.
... ignored

print.word_stats_counts

Prints a word_stats_counts object

Description
Prints a word_stats_counts object

Usage

## S3 method for class 'word_stats_counts'
print(x, ...)

Arguments

x The word_stats_counts object
... ignored
Count Object/Subject Pronouns Per Grouping Variable

**Description**

Count the number of subject/object pronouns per grouping variables.

**Usage**

```r
pronoun_type(text.var, grouping.var = NULL, pronoun.list = NULL, ...)
```

**Arguments**

- `text.var`: The text variable
- `grouping.var`: The grouping variables. Default `NULL` generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `pronoun.list`: A named list of subject/object pronouns. See **Details** for more.
- `...`: Other arguments passed to `termco`

**Details**

The following subject/object pronoun categories are the default searched terms:

- `I = c(" i'd ", " i'll ", " i'm ", " i've ", " i ")`
- `we = c(" we'd ", " we'll ", " we're ", " we've ", " we ")`
- `you = c(" you'd ", " you'll ", " you're ", " you've ", " you ", " your ")`
- `he = c(" he'd ", " he'll ", " he's ", " he ")`
- `she = c(" she'd ", " she'll ", " she's ", " she ")`
- `they = c(" they'd ", " they'll ", " they're ", " they've ", " they ")`
- `it = c(" it'd ", " it'll ", " it's ", " it ")`
- `me = c(" me ", " my ", " mine ")`
- `us = c(" us ", " our ", " ours ")`
- `him = c(" him ", " his ")`
- `her = c(" her ", " hers ")`
- `them = c(" them ")`
- `their = c(" their ", " theirs ")`
pronoun_type

Value

Returns a list of class "pronoun_type", of data frames regarding subject/object pronoun word counts:

- `preprocessed`: List of uncollapsed dataframes (raw, prop, rnp) of the class "termco" that contain all searchable subject/object pronouns.
- `raw`: raw word counts by grouping variable
- `prop`: proportional word counts by grouping variable; proportional to each individual’s subject/object pronoun use
- `rnp`: a character combination data frame of raw and proportional subject/object pronoun use

References


See Also

`object_pronoun_type, subject_pronoun_type`

Examples

```r
## Not run:
dat <- pres_debates2012
dat <- dat[dat$person %in% c(ROMNEY, OBAMA), ]
(out <- pronoun_type(dat$dialogue, dat$person))
plot(out)
plot(out, 2)
plot(out, 3)
plot(out, 3, ncol=2)

scores(out)
counts(out)
proportions(out)
preprocessed(out)

plot(scores(out))
plot(counts(out))
```
prop(proportions(out))

(out2 <- pronoun_type(hamlet$dialogue, hamlet$person))
plot(out2, 3, ncol=7)

## End(Not run)

---

**prop**

*Convert Raw Numeric Matrix or Data Frame to Proportions*

**Description**

Convert a raw matrix or dataframe to proportions/percent. Divides each element of a column by the column sum.

**Usage**

```r
prop(mat, digits = 2, percent = FALSE, by.column = TRUE, round = FALSE)
```

**Arguments**

- `mat` A numeric matrix or dataframe.
- `digits` Integer; number of decimal places to round.
- `percent` logical. If TRUE output given as percent. If FALSE the output is proportion.
- `by.column` logical. If TRUE applies to the column. If FALSE applies by row.
- `round` logical. If TRUE rounds the returned values (controlled by digits).

**Value**

Returns a matrix with proportionally scaled values.

**Examples**

```r
## Not run:
y <- wdf(DATA$state, DATA$person, stopwords = c("your", "yours"), margins = TRUE)
prop(wfm(y), 4)[1:10, ]  # as a proportion
prop(wfm(y), 4, TRUE)[1:10, ]  # as a percentage
heatmap(prop(wfm(y), 4))
wdstraj <- word_stats(rajSPLIT$dialogue, rajSPLIT$person)
prop(wdstraj$gts[, -1], 5)[1:15, 1:6]
## End(Not run)
```
proportions

Generic Proportions Method

Description
Access the proportions dataframes from select qdap outputs.

Usage
proportions(x, ...)

Arguments
x A qdap object (list) with a proportions dataframe (e.g., termco).
... Arguments passed to proportions method of other classes.

Value
Returns a data.frame of proportions.

See Also
scores, counts, preprocessed, visual

proportions.character_table

Term Counts

Description
View character_table proportions.

Usage
## S3 method for class 'character_table'
proportions(x, ...)

Arguments
x The character_table object.
... ignored

Details
character_table Method for proportions
proportions.end_mark_by

*Question Counts*

### Description

View `end_mark_by` proportions.

### Usage

```r
## S3 method for class 'end_mark_by'
proportions(x, ...)
```

### Arguments

- `x` The `end_mark_by` object.
- `...` ignored

### Details

`end_mark_by` Method for proportions

---

proportions.formality

*Formality*

### Description

View `formality` proportions.

### Usage

```r
## S3 method for class 'formality'
proportions(x, ...)
```

### Arguments

- `x` The `formality` object.
- `...` ignored

### Details

`formality` Method for proportions
proportions.object

Description

View object proportions.

Usage

```r
## S3 method for class 'object'
proportions(x, ...)
```

Arguments

- `x` The object.
- `...` ignored

Details

object Method for proportions

---

proportions.pos

Description

View pos proportions.

Usage

```r
## S3 method for class 'pos'
proportions(x, ...)
```

Arguments

- `x` The pos object.
- `...` ignored

Details

pos Method for proportions
proportions.pos_by

**Description**

View `pos_by` proportions.

**Usage**

```r
## S3 method for class 'pos_by'
proportions(x, ...)
```

**Arguments**

- `x` The `pos_by` object.
- `...` ignored

**Details**

`pos_by` Method for proportions

---

proportions.pronoun_type

**Question Counts**

**Description**

View `pronoun_type` proportions.

**Usage**

```r
## S3 method for class 'pronoun_type'
proportions(x, ...)
```

**Arguments**

- `x` The `pronoun_type` object.
- `...` ignored

**Details**

`pronoun_type` Method for proportions
proportions.question_type

Description

View question_type proportions.

Usage

```r
## S3 method for class 'question_type'
proportions(x, ...)
```

Arguments

- `x` The question_type object.
- `...` ignored

Details

question_type Method for proportions

---

proportions.subject_pronoun_type

Description

View subject_pronoun_type proportions.

Usage

```r
## S3 method for class 'subject_pronoun_type'
proportions(x, ...)
```

Arguments

- `x` The subject_pronoun_type object.
- `...` ignored

Details

subject_pronoun_type Method for proportions
proportions.termco  

**Term Counts**

**Description**
View `termco` proportions.

**Usage**
```r
## S3 method for class 'termco'
proportions(x, ...)
```

**Arguments**
- `x`  The `termco` object.
- `...` ignored

**Details**
`termco` Method for proportions

---

proportions.word_length  

**Word Length Counts**

**Description**
View `word_length` proportions.

**Usage**
```r
## S3 method for class 'word_length'
proportions(x, ...)
```

**Arguments**
- `x`  The `word_length` object.
- `...` ignored

**Details**
`word_length` Method for proportions
proportions.word_position

Word Position

Description

View word_position proportions.

Usage

```r
## S3 method for class 'word_position'
proportions(x, ...)
```

Arguments

- `x` The word_position object.
- `...` ignored

Details

word_position Method for proportions

qcombine

Combine Columns

Description

Quickly combine columns (summed) and rename.

Usage

```r
qcombine(mat, combined.columns, elim.old = TRUE)
```

Arguments

- `mat` A matrix or dataframe with numeric combine columns.
- `combined.columns` A list of named vectors of the colnames/indexes of the numeric columns to be combined (summed). If a vector is unnamed a name will be assigned.
- `elim.old` logical. If TRUE eliminates the columns that are combined together by the named match.list. TRUE outputs the table proportionally (see prop).

Value

Returns a dataframe with combines columns.
See Also

transform

Examples

```r
## Not run:
A <- list(
a = c(1, 2, 3),
b = qcv(mpg, hp),
c = c("disp", "am")
)
B <- list(
c(1, 2, 3),
d = qcv(mpg, hp),
c("disp", "am")
)
qcombine(head(mtcars), A)
qcombine(head(mtcars), B)
qcombine(head(mtcars), B, elim.old = FALSE)

## End(Not run)
```

---

### qcv

**Quick Character Vector**

**Description**

Create a character vector without the use of quotation marks.

**Usage**

```r
qcv(..., terms = NULL, space.wrap = FALSE, trailing = FALSE, leading = FALSE, split = " ", rm.blank = TRUE)
```

**Arguments**

- **terms**: An optional argument to present the terms as one long character string. This is useful if the split (separator) is not a comma (e.g., spaces are the term separators).
- **space.wrap**: logical. If TRUE wraps the vector of terms with a leading/trailing space.
- **trailing**: logical. If TRUE wraps the vector of terms with a trailing space.
- **leading**: logical. If TRUE wraps the vector of terms with a leading space.
- **split**: Character vector of length one to use for splitting (i.e., the separator used in the vector). For use with the argument terms.
- **rm.blank**: logical. If TRUE removes all blank spaces from the vector.
- **...**: Character objects. Either ... or terms argument must be utilized.
qdap

Value

Returns a character vector.

See Also

c

Examples

```r
## Not run:
qcv(I, like, dogs)
qcv(terms = "I, like, dogs") #default separator is " "
qcv(terms = "I, like, dogs", split = ",")
qcv(terms = "I like dogs")
qcv(I, like, dogs, space.wrap = TRUE)
qcv(I, like, dogs, trailing = TRUE)
qcv(I, like, dogs, leading = TRUE)
exclude(Top25Words, qcv(the, of, and))
qcv(terms = "mpg cyl disp hp drat wt qsec vs am gear carb")

## End(Not run)
```

qdap

qdap: Quantitative Discourse Analysis Package

Description

This package automates many of the tasks associated with quantitative discourse analysis of transcripts containing discourse. The package provides parsing tools for preparing transcript data, coding tools and analysis tools for richer understanding of the data. Many functions allow the user to aggregate data by any number of grouping variables, providing analysis and seamless integration with other R packages which enable higher level analysis and visualization of text. This empowers the researcher with more flexible, efficient and targeted methods and tools.

qdap_df

Create qdap Specific Data Structure

Description

Creating this qdap specific data structure enables short hand with subsequent qdap function calls that utilize the text.var argument. Combined with the %% operator, the user need not specify a data set or the text.var argument (as many qdap functions contain a text.var argument).

Change text.var column of a qdap_df object.
Usage

qdap_df(dataframe, text.var)

Text(object)

Text(object) <- value

Arguments

dataframe  A data.frame with a text variable. Generally, sentSplit should be run first (sentSplit actually produces a data.frame that is of the class "qdap_df").
text.var    The name of the text.var column.
object      A data.frame of the class "qdap_df".
value       A character string of the updated text.var column.

Value

Returns a data.frame of the class "qdap_df".

References

Inspired by dplyr's tbl_df structure.

See Also

%%, sentSplit

Examples

## Not run:
dat <- qdap_df(DATA, state)
dat %% trans_cloud(grouping.var=person)
dat %% trans_cloud(grouping.var=person, text.var=stemmer(DATA$state))
dat %% termco(grouping.var=person, match.list=list("fun", "computer"))
class(dat)

## Change text column in `qdap_df` (Example 1)
dat2 <- sentSplit(DATA, "state", stem.col = TRUE)
class(dat2)
dat2 %% trans_cloud()
Text(dat2)
## change the `text.var` column
Text(dat2) <- "stem.text"
dat2 %% trans_cloud()

## Change text column in `qdap_df` (Example 2)
(dat2$fake_dat <- paste(emoticon[1:11,2], dat2$state))
Text(dat2) <- "fake_dat"
(m <- dat2 %% sub_holder(emoticon[,2]))
m$unhold(stri

## Various examples with qdap functions

dat <- sentSplit(DATA, "state")
dat %% trans_cloud(grouping.var=person)
dat %% termco(person, match.list=list("fun", "computer"))
dat %% trans_venn(person)
dat %% polarity(person)
dat %% formality(person)
dat %% automated_readability_index(person)
dat %% Dissimilarity(person)
dat %% gradient_cloud(sex)
dat %% dispersion_plot(c("fun", "computer"))
dat %% discourse_map(list(sex, adult))
dat %% gantt_plot(person)
dat %% word_list(adult)
dat %% end_mark_by(person)
dat %% end_mark()
dat %% word_stats(person)
dat %% wfm(person)
dat %% word_cor(person, "i")
dat %% sentCombine(person)
dat %% question_type(person)
dat %% word_network_plot()
dat %% character_count()
dat %% char_table(person)
dat %% phrase_net(2, .1)
dat %% boolean_search("it||!")
dat %% trans_context(person, which(end_mark(DATA.SPLIT[, "state"])) == "]?")
dat %% mgsub(c("it's", "I'm"), c("it is", "I am"))

## combine with magrittr/dplyr chaining

dat %% wfm(person) %>%% plot()
dat %% polarity(person) %>%% scores()
dat %% polarity(person) %>%% counts()
dat %% polarity(person) %>%% scores()
dat %% polarity(person) %>%% scores() %>%% plot()
dat %% polarity(person) %>%% scores %>%% plot

## End(Not run)

### qheat

**Quick Heatmap**

#### Description

A quick heatmap function for visualizing typical qdap dataframe/matrix outputs.

- **diversity** Method for qheat
- **termco** Method for qheat
- **word_stats** Method for qheat
character_table Method for qheat

question_type Method for qheat

pos_by Method for qheat

Usage

```r
defalut_3_method:
  qheat(mat, low = "white", high = "darkblue",
        values = FALSE, digits = 1, text.size = 3, text.color = "grey40",
        xaxis.col = "black", yaxis.col = "black", order.by = NULL,
        grid = "white", by.column = TRUE, auto.size = FALSE, mat2 = NULL,
        plot = TRUE, facet.vars = NULL, facet.flip = FALSE, diag.na = FALSE,
        diag.values = "", ...)

# S3 method for class 'diversity'
qheat(mat, low = "white", high = "darkblue",
        values = FALSE, digits = 1, text.size = 3, text.color = "grey40",
        xaxis.col = "black", yaxis.col = "black", order.by = NULL,
        grid = "white", by.column = TRUE, auto.size = FALSE, mat2 = NULL,
        plot = TRUE, facet.vars = NULL, facet.flip = FALSE, diag.na = FALSE,
        diag.values = "", ...)

# S3 method for class 'termco'
qheat(mat, low = "white", high = "darkblue",
        values = FALSE, digits = 1, text.size = 3, text.color = "grey40",
        xaxis.col = "black", yaxis.col = "black", order.by = NULL,
        grid = "white", by.column = TRUE, auto.size = FALSE, mat2 = NULL,
        plot = TRUE, facet.vars = NULL, facet.flip = FALSE, diag.na = FALSE,
        diag.values = "", ...)

# S3 method for class 'word_stats'
qheat(mat, low = "white", high = "darkblue",
        values = FALSE, digits = 1, text.size = 3, text.color = "grey40",
        xaxis.col = "black", yaxis.col = "black", order.by = NULL,
        grid = "white", by.column = TRUE, auto.size = FALSE, mat2 = NULL,
        plot = TRUE, facet.vars = NULL, facet.flip = FALSE, diag.na = FALSE,
        diag.values = "", ...)

# S3 method for class 'character_table'
qheat(mat, low = "white", high = "darkblue",
        values = FALSE, digits = 1, text.size = 3, text.color = "grey40",
        xaxis.col = "black", yaxis.col = "black", order.by = NULL,
        grid = "white", by.column = TRUE, auto.size = FALSE, mat2 = NULL,
        plot = TRUE, facet.vars = NULL, facet.flip = FALSE, diag.na = FALSE,
        diag.values = "", ...)```
Arguments

mat A matrix or dataframe produced by many qdap functions in which the first column is the grouping variable and the rest of the matrix is numeric. Also accepts objects directly from `word_stats` and `question_type`.

low The color to be used for lower values.

high The color to be used for higher values.

values logical. If TRUE the cell values will be included on the heatmap.

digits The number of digits displayed if values is TRUE.

text.size A integer size to plot the text if values is TRUE.

text.color A character vector to plot the text if values is TRUE.

xaxis.col A single character vector color choice for the high values.

yaxis.col A single character vector color choice for the low values.

order.by An optional character vector of a variable name to order the columns by. To reverse use a negative (–) before the column name.

grid The color of the grid (Use NULL to remove the grid).

by.column logical. If TRUE applies scaling to the column. If FALSE applies scaling by row (use NULL to turn off scaling).

auto.size logical. If TRUE the visual will be resized to create square cells.

mat2 A second matrix equal in dimensions to mat that will be used for cell labels if values is TRUE.

plot logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knitr, sweave, etc. to add additional plot layers.
facet.vars  A character vector of 1 or 2 column names to facet by.
facet.flip   logical If TRUE the direction of the faceting is reversed.
diag.na     logical. If TRUE and mat is a symmetrical matrix the diagonals are set to NA. This is useful with correlation matrices because the diagonal of ones do not affect the scaling of the heatmap.
diag.values The string to be used for the diagonal labels (values) if diag.na is set to TRUE. Default is to not print a value.

... Not currently used.

Details

qheat is useful for finding patterns and anomalies in large qdap generated dataframes and matrices.

Note

qheat is a fast way of working with data formats produced by qdap. The function isn’t designed to be extended beyond exploratory qdap usage.

Examples

```r
## Not run:
dat <- sentSplit(DATA, "state")
ws.ob <- with(dat, word_stats(state, list(sex, adult), tot=tot))
qheat(ws.ob)
qheat(ws.ob) + coord_flip()
qheat(ws.ob, order.by = "sptot",
     xaxis.col = c("red", "black", "green", "blue"))
qheat(ws.ob, order.by = "sptot")
qheat(ws.ob, order.by = "-sptot")
qheat(ws.ob, values = TRUE)
qheat(ws.ob, values = TRUE, text.color = "red")
qheat(ws.ob, "yellow", "red", grid = FALSE)
qheat(mtcars, facet.vars = "cyl")
qheat(mtcars, facet.vars = c("gear", "cyl"))
qheat(t(mtcars), by.column=FALSE)
qheat(cor(mtcars), diag.na=TRUE, diag.value="", by.column=NULL, values = TRUE)

dat1 <- data.frame(G=LETTERS[1:5], matrix(rnorm(20), ncol = 4))
dat2 <- data.frame(matrix(LETTERS[1:25], ncol=5))
qheat(dat1, values=TRUE)
qheat(dat1, values=TRUE, mat2=dat2)

## End(Not run)
```
qprep

Quick Preparation of Text

Description

Wrapper for \texttt{bracketX}, \texttt{replace_number}, \texttt{replace_symbol}, \texttt{replace_abbreviation} and \texttt{scrubber} to quickly prepare text for analysis. Care should be taken with this function to ensure data is properly formatted and complete.

Usage

\begin{verbatim}
qprep(text.var, rm.dash = TRUE, bracket = "all", missing = NULL,
names = FALSE, abbreviation = qdapDictionaries::abbreviations,
replace = NULL, ignore.case = TRUE, num.paste = TRUE, ...)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{text.var} \hspace{1cm} The text variable.
  \item \texttt{rm.dash} \hspace{1cm} logical. If \texttt{TRUE} dashes will be removed.
  \item \texttt{bracket} \hspace{1cm} The type of bracket (and encased text) to remove. This is one of the strings \texttt{"curly"}, \texttt{"square"}, \texttt{"round"}, \texttt{"angle"} and \texttt{"all"}. These strings correspond to: \{, [, (, < or all four types. Also takes the argument \texttt{null} which turns off this parsing technique.
  \item \texttt{missing} \hspace{1cm} Value to assign to empty cells.
  \item \texttt{names} \hspace{1cm} logical. If \texttt{TRUE} the sentences are given as the names of the counts.
  \item \texttt{abbreviation} \hspace{1cm} A two column key of abbreviations (column 1) and long form replacements (column 2) or a vector of abbreviations. Default is to use \texttt{qdap}'s abbreviations data set. Also takes the argument \texttt{NULL} which turns off this parsing technique.
  \item \texttt{replace} \hspace{1cm} A vector of long form replacements if a data frame is not supplied to the abbreviation argument.
  \item \texttt{ignore.case} \hspace{1cm} logical. If \texttt{TRUE} replaces without regard to capitalization.
  \item \texttt{num.paste} \hspace{1cm} logical. If \texttt{TRUE} a the elements of larger numbers are separated with spaces. If \texttt{FALSE} the elements will be joined without spaces. Also takes the argument \texttt{NULL} which turns off this parsing technique.
  \end{itemize}

Note

Care should be taken with this function to ensure data is properly formatted and complete.

See Also

\texttt{bracketX}, \texttt{replace_abbreviation}, \texttt{replace_number}, \texttt{replace_symbol}
Examples

```r
## Not run:
x <- "I like 60 (laughter) #d-bot and $6 @ the store w/o 8p.m."
qprep(x)

## End(Not run)
```

qtheme

Add themes to a Network object.

Description

qtheme - This function builds generic themes to add a theme to a Network object rather than individual print arguments.

- `theme_nightheat`: A night heat theme.
- `theme_badkitchen`: A 70s kitchen theme.
- `theme_cafe`: A cafe theme.
- `theme_grayscale`: A grayscale theme.
- `theme_norah`: A Norah theme.
- `theme_hipster`: A hipster theme.
- `theme_duskheat`: A duskheat theme.

Usage

```r
qtheme(x = "generic", title, title.color, layout, legend, legend.cex,
        legend.text.color, legend.gradient, bg, vertex.color, vertex.size,
        vertex.frame.color, vertex.label.color, vertex.label.cex, edge.label.color,
        edge.label.cex)

theme_nightheat(x = pars["x"], title = pars["title"],
                title.color = pars["title.color"], layout = pars["layout"],
                legend = pars["legend"], legend.cex = pars["legend.cex"],
                legend.gradient = pars["legend.gradient"], bg = pars["bg"],
                legend.text.color = pars["legend.text.color"],
                vertex.color = pars["vertex.color"],
                vertex.size = pars["vertex.size"],
                vertex.frame.color = pars["vertex.frame.color"],
                vertex.label.color = pars["vertex.label.color"],
                vertex.label.cex = pars["vertex.label.cex"],
                edge.label.color = pars["edge.label.color"],
                edge.label.cex = pars["edge.label.cex"], ...)

theme_badkitchen(x = pars["x"], title = pars["title"],
                 title.color = pars["title.color"], layout = pars["layout"],
                 ...)
```

## Not run:
x <- "I like 60 (laughter) #d-bot and $6 @ the store w/o 8p.m."
qprep(x)
legend = pars[['legend']], legend.cex = pars[['legend.cex']],
legend.gradient = pars[['legend.gradient']], bg = pars[['bg']],
legend.text.color = pars[['legend.text.color']],
vertex.color = pars[['vertex.color']],
vertex.size = pars[['vertex.size']],
vertex.frame.color = pars[['vertex.frame.color']],
vertex.label.color = pars[['vertex.label.color']],
vertex.label.cex = pars[['vertex.label.cex']],
edge.label.color = pars[['edge.label.color']],
edge.label.cex = pars[['edge.label.cex']], ...

theme_cafe(x = pars[['x']], title = pars[['title']],
title.color = pars[['title.color']], layout = pars[['layout']],
legend = pars[['legend']], legend.cex = pars[['legend.cex']],
legend.gradient = pars[['legend.gradient']], bg = pars[['bg']],
legend.text.color = pars[['legend.text.color']],
vertex.color = pars[['vertex.color']],
vertex.size = pars[['vertex.size']],
vertex.frame.color = pars[['vertex.frame.color']],
vertex.label.color = pars[['vertex.label.color']],
vertex.label.cex = pars[['vertex.label.cex']],
edge.label.color = pars[['edge.label.color']],
edge.label.cex = pars[['edge.label.cex']], ...

theme_grayscale(x = pars[['x']], title = pars[['title']],
title.color = pars[['title.color']], layout = pars[['layout']],
legend = pars[['legend']], legend.cex = pars[['legend.cex']],
legend.gradient = pars[['legend.gradient']], bg = pars[['bg']],
legend.text.color = pars[['legend.text.color']],
vertex.color = pars[['vertex.color']],
vertex.size = pars[['vertex.size']],
vertex.frame.color = pars[['vertex.frame.color']],
vertex.label.color = pars[['vertex.label.color']],
vertex.label.cex = pars[['vertex.label.cex']],
edge.label.color = pars[['edge.label.color']],
edge.label.cex = pars[['edge.label.cex']], ...

theme_greyscale(x = pars[['x']], title = pars[['title']],
title.color = pars[['title.color']], layout = pars[['layout']],
legend = pars[['legend']], legend.cex = pars[['legend.cex']],
legend.gradient = pars[['legend.gradient']], bg = pars[['bg']],
legend.text.color = pars[['legend.text.color']],
vertex.color = pars[['vertex.color']],
vertex.size = pars[['vertex.size']],
vertex.frame.color = pars[['vertex.frame.color']],
vertex.label.color = pars[['vertex.label.color']],
vertex.label.cex = pars[['vertex.label.cex']],
edge.label.color = pars[['edge.label.color']],
Arguments

x The name of the qtheme.
title The title of the plot. NULL eliminates title. NA uses title attribute of the Network object.
title.color The color of the title.
layout igraph layout to use.
qtheme

*legend*  The coordinates of the legend. See `color.legend` for more information.

*legend.cex*  character expansion factor. NULL and NA are equivalent to 1.0. See `mtext` for more information.

*legend.text.color*  The text legend text color.

*legend.gradient*  A vector of ordered colors to use for the gradient fills in the network edges.

*bg*  The color to be used for the background of the device region. See `par` for more information.

*vertex.color*  The font family to be used for vertex labels.

*vertex.size*  The size of the vertex.

*vertex.frame.color*  The color of the vertex border.

*vertex.label.color*  The color of the labels.

*vertex.label.cex*  The font size for vertex labels.

*edge.label.color*  The color for the edge labels. Use NA to remove.

*edge.label.cex*  The font size of the edge labels.

...  Additional arguments supplied to `qtheme`.

**Examples**

```r
## Not run:
(poldat <- with(sentSplit(DATA, 4), polarity(state, person)))

m <- Network(poldat)

m + theme_nightheat
m + theme_cafe
m + theme_grayscale
m + theme_norah
m + theme_hipster
m + theme_badkitchen
m + theme_duskheat

## make your own themes
theme_irish <- qtheme(x = "irish", bg = "grey25",
vertex.label.color = "grey50", legend.text.color = "white",
legend.gradient = c("darkgreen", "white", "darkorange"),
edge.label.color="white", vertex.size=20)

m + theme_irish

## End(Not run)
```
# question_type

<table>
<thead>
<tr>
<th>question_type</th>
<th>Count of Question Type</th>
</tr>
</thead>
</table>

## Description

Transcript apply question counts.

## Usage

```r
question_type(text.var, grouping.var = NULL, neg.cont = FALSE, percent = TRUE, zero.replace = 0, digits = 2, contraction = qdapDictionaries::contractions, bracket = "all", amplifiers = qdapDictionaries::amplification.words, ...)```

## Arguments

- **text.var**: The text variable
- **grouping.var**: The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- **neg.cont**: logical. If TRUE provides separate counts for the negative contraction forms of the interrogative words.
- **percent**: logical. If TRUE output given as percent. If FALSE the output is proportion.
- **zero.replace**: Value to replace 0 values with.
- **digits**: Integer; number of decimal places to round when printing.
- **contraction**: A two column key of contractions (column 1) and expanded form replacements (column 2) or a vector of contractions. Default is to use qdapDictionaries's `contractions` data set.
- **bracket**: The type of bracket (and encased text) to remove. This is one or more of the strings "curly", "square", "round", "angle" and "all". These strings correspond to: {, [,. (, < or all four types.
- **amplifiers**: A character vector of terms that increase the intensity of a positive or negative word. Default is to use qdapDictionaries's `amplification.words` data set.
- **...**: Other arguments passed to `bracketX`.

## Details

The algorithm searches for the following interrogative words (and optionally, their negative contraction form as well):

1) whose 2) whom 3) who 4) where 5) what 6) which 7) why 8) when 9) were* 10) was* 11) does* 12) did* 13) do* 14) is 15) are* 16) will* 17) how 18) should 19) could 20) would* 21) shall 22) may 23) might* 24) must* 25) can* 26) has 27) have* 28) had* 29) ok 30) right 31) correct 32) implied do/does/did

The interrogative word that is found first (with the exception of "ok", "right"/"alright", and "correct") in the question determines the sentence type. "ok", "right"/"alright", and "correct" sentence
types are determined if the sentence is a question with no other interrogative words found and "ok", "right"/"alright", or "correct" is the last word of the sentence. Those interrogative sentences beginning with the word "you", "wanna", or "want" are categorized as implying do/does/did question type, though the use of do/does/did is not explicit. Those sentence beginning with "you" followed by a select interrogative word (and or their negative counter parts) above (marked with *) or 1-2 amplifier(s) followed by the select interrogative word are categorized by the select word rather than an implied do/does/did question type. A sentence that is marked "ok" over rides an implied do/does/did label. Those with undetermined sentence type are labeled unknown.

Value

Returns a list of:

- **raw** A dataframe of the questions used in the transcript and their type.
- **count** A dataframe of total questions (tot.quest) and counts of question types (initial interrogative word) by grouping variable(s).
- **rnp** Dataframe of the frequency and proportions of question types by grouping variable.
- **inds** The indices of the original text variable that contain questions.
- **missing** The row numbers of the missing data (excluded from analysis).
- **percent** The value of percent used for plotting purposes.
- **zero.replace** The value of zero.replace used for plotting purposes.

See Also

- `colcomb2class.bracketX`

Examples

```r
# Not run:
# Inspect the algorithm classification
x <- c("Kate's got no appetite doesn't she?",
    "Wanna tell Daddy what you did today?",
    "You helped getting out a book?", "umm hum?",
    "Do you know what it is?", "What do you want?",
    "Who's there?", "Whose?", "Why do you want it?",
    "Want some?", "Where did it go?", "Was it fun?")

left.just(preprocessed(question_type(x))[, c(2, 6)])

# Transcript/dialogue examples
(x <- question_type(DATA.SPLIT$state, DATA.SPLIT$person))

# methods
scores(x)
plot(scores(x))
counts(x)
plot(counts(x))
proportions(x)
```
raj

Romeo and Juliet (Unchanged & Complete)

Description

A dataset containing the original transcript from Romeo and Juliet as it was scraped from: http://shakespeare.mit.edu/romeo_juliet/full.html.

Usage

data(raj)

Format

A data frame with 840 rows and 3 variables

Details

• person. Character in the play
• dialogue. The spoken dialogue
• act. The act (akin to repeated measures)

References

http://shakespeare.mit.edu/romeo_juliet/full.html
**Romeo and Juliet: Act 1**

### Description
A dataset containing Romeo and Juliet: Act 1.

### Usage
```r
data(raj.act.1)
```

### Format
A data frame with 235 rows and 2 variables

### Details
- **person.** Character in the play
- **dialogue.** The spoken dialogue

### References

---

**Romeo and Juliet: Act 1 Parts of Speech by Person**

### Description
A dataset containing a list from `pos_by` using the `mraja1spl` data set (see `pos_by` for more information).

### Usage
```r
data(raj.act.1POS)
```

### Format
A list with 10 elements [http://shakespeare.mit.edu/romeo_juliet/full.html](http://shakespeare.mit.edu/romeo_juliet/full.html)
Details

- **text**  The original text
- **POSStaged** The original words replaced with parts of speech in context.
- **POSprop** Dataframe of the proportion of parts of speech by row.
- **POSfreq** Dataframe of the frequency of parts of speech by row.
- **POSrmp** Dataframe of the frequency and proportions of parts of speech by row
- **percent** The value of percent used for plotting purposes.
- **zero.replace** The value of zero.replace used for plotting purposes.
- **pos.by.freq** Dataframe of the frequency of parts of speech by grouping variable.
- **pos.by.prop** Dataframe of the proportion of parts of speech by grouping variable.
- **pos.by.rnp** Dataframe of the frequency and proportions of parts of speech by grouping variable.

---

**raj.act.2**  
**Romeo and Juliet: Act 2**

Description

A dataset containing Romeo and Juliet: Act 2.

Usage

```
data(raj.act.2)
```

Format

A data frame with 205 rows and 2 variables

Details

- person. Character in the play
- dialogue. The spoken dialogue

References

Description
A dataset containing Romeo and Juliet: Act 3.

Usage
data(raj.act.3)

Format
A data frame with 197 rows and 2 variables

Details
• person. Character in the play
• dialogue. The spoken dialogue

References
http://shakespeare.mit.edu/romeo_juliet/full.html

Description
A dataset containing Romeo and Juliet: Act 4.

Usage
data(raj.act.4)

Format
A data frame with 115 rows and 2 variables

Details
• person. Character in the play
• dialogue. The spoken dialogue

References
http://shakespeare.mit.edu/romeo_juliet/full.html
raj.act.5  Romeo and Juliet: Act 5

Description
A dataset containing Romeo and Juliet: Act 5.

Usage
data(raj.act.5)

Format
A data frame with 88 rows and 2 variables

Details
- person. Character in the play
- dialogue. The spoken dialogue

References
http://shakespeare.mit.edu/romeo_juliet/full.html

raj.demo...  Romeo and Juliet Demographics

Description
A dataset containing Romeo and Juliet demographic information for the characters.

Usage
data(raj.demo...)

Format
A data frame with 34 rows and 4 variables

Details
- person. Character in the play
- sex. Gender
- fam.aff. Family affiliation of character
- died. Dummy coded death variable (0-no; 1-yes); if yes the character dies in the play
rajPOS

Romeo and Juliet Split in Parts of Speech

Description
A dataset containing a list from pos using the raj data set (see pos for more information).

Usage
data(rajPOS)

Format
A list with 4 elements

Details
text The original text
POStagged The original words replaced with parts of speech in context.
POSprop Dataframe of the proportion of parts of speech by row.
POSfreq Dataframe of the frequency of parts of speech by row.

References
http://shakespeare.mit.edu/romeo_juliet/full.html

rajsPLIT
Romeo and Juliet (Complete & Split)

Description
A dataset containing the complete dialogue of Romeo and Juliet with turns of talk split into sentences.

Usage
data(rajsPLIT)

Format
A data frame with 2151 rows and 8 variables
Details

- person. Character in the play
- sex. Gender
- fam.aff. Family affiliation of character
- died. Dummy coded death variable (0-no; 1-yes); if yes the character dies in the play
- dialogue. The spoken dialogue
- act. The act (akin to repeated measures)
- stem.text. Text that has been stemmed

References

http://shakespeare.mit.edu/romeo_juliet/full.html

random_sent

Generate Random Dialogue Data

Description

random_sent - Generates a random sample of sentences (sentences are sampled at the word level and there for are likely nonsensical).

random_data - Generate random dialogue, people, and demographic variables

Usage

random_sent(n = 10, len = 14, range = len - 1,
dictionary = qdapDictionaries::Top200Words, endmark.fun = function()
sample(c(".", ":", ":", ":"), 1, prob = c(0.85, 0.05, 0.05, 0.05)))

random_data(n = 10, ..., n.people = 10, ages = 7:10,
people.names = unique(tolower(qdapDictionaries::NAMES[[1]])))

Arguments

n Number of sentences to create.
len Average length of sentences (in words).
range Range around len that number of words may vary. This may be a recycled single integer vector or an integer vector of length 2.
dictionary A dictionary of words to sample from.
dendmark.fun A function to create random end marks.
n.people An integer of the number of people to include in the sample (number of people is sampled from; if n is smaller not all people may be included).
ages The possible ages to choose from (numeric).
people.names A vector of names to choose from at least as large as n.people.
... Other arguments passed to random_sent
**Value**

random_sent - Returns a random vector of sentence strings.

random_data - Returns a data.frame of people, dialogue, and demographic variables of the class sent_split.

**Examples**

```r
## Not run:
random_sent()
random_sent(200, 10)

dict <- sort(unique(bag_o_words(pres_debates2012["dialogue"])))
random_sent(dictionary=dict)

random_data()
random_data(ages = seq(10, 20, by = .5))
random_data(50) ## word_stats(person)
random_data(100) ## word_stats(list(race, sex))
random_data(dictionary = dict)

## End(Not run)
```

---

**rank_freq_mplot**

**Rank Frequency Plot**

**Description**

rank_freq_mplot - Plot a faceted word rank versus frequencies by grouping variable(s).

rank_freq_plot - Plot word rank versus frequencies.

**Usage**

```r
rank_freq_mplot(text.var, grouping.var = NULL, ncol = 4, jitter = 0.2,
                 log.freq = TRUE, log.rank = TRUE, hap.col = "red", dis.col = "blue",
                 alpha = 1, shape = 1, title = "Rank-Frequency Plot", digits = 2,
                 plot = TRUE)

rank_freq_plot(words, frequencies, plot = TRUE, title.ext = NULL,
                jitter.amount = 0.1, log.scale = TRUE, hap.col = "red",
                dis.col = "blue")
```

**Arguments**

- **text.var** The text variable.
- **grouping.var** The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- **ncol** integer value indicating the number of columns in the facet wrap.
jitter    Amount of horizontal jitter to add to the points.
log.freq  logical. If TRUE plots the frequencies in the natural log scale.
log.rank  logical. If TRUE plots the ranks in the natural log scale.
hap.col   Color of the hapax legomenon points.
dis.col   Color of the dis legomenon points.
alpha     Transparency level of points (ranges between 0 and 1).
shape     An integer specifying the symbol used to plot the points.
title     Optional plot title.
digits    Integer; number of decimal places to round.
plot      logical. If TRUE provides a rank frequency plot.
words     A vector of words.
frequencies A vector of frequencies corresponding to the words argument.
title.ext  The title extension that extends: "Rank-Frequency Plot ...
jitter.amount Amount of horizontal jitter to add to the points.
log.scale  logical. If TRUE plots the rank and frequency as a log scale.

Value
Returns a rank-frequency plot and a list of three dataframes:

   WORD_COUNTS    The word frequencies supplied to rank_freq_plot or created by rank_freq_mplot.
   RANK_AND_FREQUENCY_STATS A dataframe of rank and frequencies for the words used in the text.
   LEGOMENA_STATS A dataframe displaying the percent hapax legomena and percent dis legomena of the text.

Note
rank_freq_mplot utilizes the ggplot2 package, whereas, rank_freq_plot employs base graphics. rank_freq_mplot is more general & flexible; in most cases rank_freq_mplot should be preferred.

References

Examples
## Not run:
#rank_freq_mplot EXAMPLES:
x1 <- rank_freq_mplot(DATA$state, DATA$person, ncol = 2, jitter = 0)
ltruncdf(x1, 10)
x2 <- rank_freq_mplot(mraja$sp$dialogue, mraja$sp$person, ncol = 5,
                     hap.col = "purple")
ltruncdf(x2, 10)
invisible(rank_freq_mplot(mraja$sp$dialogue, mraja$sp$person, ncol = 5,
log.freq = FALSE, log.rank = FALSE, jitter = .6))
invisible(rank_freq_mplot(raj$dialogue, jitter = .5, alpha = 1/15))
invisible(rank_freq_mplot(raj$dialogue, jitter = .5, shape = 19, alpha = 1/15))

#rank_freq_plot EXAMPLES:
mod <- with(mrajal$pl, word_list(dialogue, person, cut.n = 10,
   cap.list=unique(mrajal$pl$person)))
x3 <- rank_freq_plot(mod$fwl$Romeo$WORD, mod$fwl$Romeo$FREQ, title.ext = 'Romeo')
ltruncdf(x3, 10)
ltruncdf(rank_freq_plot(mod$fwl$Romeo$WORD, mod$fwl$Romeo$FREQ, plot = FALSE), 10)
invisible(rank_freq_plot(mod$fwl$Romeo$WORD, mod$fwl$Romeo$FREQ, title.ext = 'Romeo',
   jitter.ammount = 0.15, hap.col = "darkgreen", dis.col = "purple")
invisible(rank_freq_plot(mod$fwl$Romeo$WORD, mod$fwl$Romeo$FREQ, title.ext = 'Romeo',
   jitter.ammount = 0.5, log.scale=FALSE))
invisible(lapply(seq_along(mod$fwl)), function(i){
   dev.new()
   rank_freq_plot(mod$fwl[[i]]$WORD, mod$fwl[[i]]$FREQ,
      title.ext = names(mod$fwl)[i], jitter.ammount = 0.5, log.scale=FALSE)
})

## End(Not run)

---

**raw.time.span**

Minimal Raw Time Span Data Set

**Description**

A dataset containing a list of named vectors of time spans.

**Usage**

data(raw.time.span)

**Format**

A list with 3 elements

---

**read.transcript**

Read Transcripts Into R

**Description**

Read .docx, .csv or .xlsx files into R.
Usage

read.transcript(file, col.names = NULL, text.var = NULL,
merge.broke.tot = TRUE, header = FALSE, dash = "", ellipsis = "...",
quote2bracket = FALSE, rm.empty.rows = TRUE, na.strings = c("999", "NA",
"",""), sep = NULL, skip = 0, nontext2factor = TRUE, text,
comment.char = "", ...)  

Arguments

file The name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an absolute path, the file name is relative to the current working directory, getwd().
col.names A character vector specifying the column names of the transcript columns.
text.var A character string specifying the name of the text variable will ensure that variable is classed as character. If NULL read.transcript attempts to guess the text.variable (dialogue).
merge.broke.tot logical. If TRUE and if the file being read in is .docx with broken space between a single turn of talk read.transcript will attempt to merge these into a single turn of talk.
header logical. If TRUE the file contains the names of the variables as its first line.
dash A character string to replace the en and em dashes special characters (default is to remove).
ellipsis A character string to replace the ellipsis special characters (default is text ...).
quote2bracket logical. If TRUE replaces curly quotes with curly braces (default is FALSE). If FALSE curly quotes are removed.
rm.empty.rows logical. If TRUE read.transcript attempts to remove empty rows.
na.strings A vector of character strings which are to be interpreted as NA values.
sep The field separator character. Values on each line of the file are separated by this character. The default of NULL instructs read.transcript to use a separator suitable for the file type being read in.
skip Integer; the number of lines of the data file to skip before beginning to read data.
nontext2factor logical. If TRUE attempts to convert any non-text to a factor.
text Character string: if file is not supplied and this is, then data are read from the value of text. Notice that a literal string can be used to include (small) data sets within R code.
comment.char A character vector of length one containing a single character or an empty string. Use "" to turn off the interpretation of comments altogether.
... Further arguments to be passed to read.table.

Value

Returns a dataframe of dialogue and people.
read.transcript

Warning

read.transcript may contain errors if the file being read in is .docx. The researcher should carefully investigate each transcript for errors before further parsing the data.

Note

If a transcript is a .docx file read transcript expects two columns (generally person and dialogue) with some sort of separator (default is colon separator). .doc files must be converted to .docx before reading in.

Author(s)

Bryan Goodrich and Tyler Rinker <tyler.rinker@gmail.com>.

References

https://github.com/trinker/qdap/wiki/Reading-.docx-%5BMS-Word%5D-Transcripts-into-R

See Also

dir_map

Examples

## Not run:
#Note: to view the document below use the path:
system.file("extdata/transcripts/", package = "qdap")
(doc1 <- system.file("extdata/transcripts/trans1.docx", package = "qdap"))
(doc2 <- system.file("extdata/transcripts/trans2.docx", package = "qdap"))
(doc3 <- system.file("extdata/transcripts/trans3.docx", package = "qdap"))
(doc4 <- system.file("extdata/transcripts/trans4.xlsx", package = "qdap"))

dat1 <- read.transcript(doc1)
truncdf(dat1, 40)
dat2 <- read.transcript(doc1, col.names = c("person", "dialogue"))
truncdf(dat2, 40)
dat2b <- rm_row(dat2, "person", "[C]" #remove bracket row
truncdf(dat2b, 40)

## read.transcript(doc2) #throws an error (need skip)
dat3 <- read.transcript(doc2, skip = 1); truncdf(dat3, 40)

## read.transcript(doc3, skip = 1) #incorrect read; wrong sep
dat4 <- read.transcript(doc3, sep = "-", skip = 1); truncdf(dat4, 40)

dat5 <- read.transcript(doc4); truncdf(dat5, 40) #an .xlsx file
trans <- "sam: Computer is fun. Not too fun.
greg: No it's not, it's dumb.
teacher: What should we do?
sam: You liar, it stinks!"
replacer

Replace Cells in a Matrix or Data Frame

Description

Replace elements of a dataframe, matrix or vector with least restrictive class.

Usage

replacer(dat, replace = 0, with = "-")

Arguments

dat              Data; either a dataframe, matrix or vector.
replace          Element to replace.
with             Replacement element.

Value

Returns a dataframe, matrix or vector with the element replaced.
Examples

```r
## Not run:
replace(mtcars[1:10, ], 0, "REP")
replace(mtcars[1:10, , 4], NA)
replace(c("a", "b"), "a", "foo")
# replace missing values (NA)
dat <- data.frame(matrix(sample(c(1:3, NA), 25, TRUE), ncol=5))
replace(dat, NA, "FOO")

## End(Not run)
```

---

**replace_abbreviation**  
*Replace Abbreviations*

Description

This function replaces abbreviations with long form.

Usage

```r
replace_abbreviation(text.var, abbreviation = qdapDictionaries::abbreviations,
replace = NULL, ignore.case = TRUE)
```

Arguments

- **text.var**: The text variable.
- **abbreviation**: A two column key of abbreviations (column 1) and long form replacements (column 2) or a vector of abbreviations. Default is to use qdapDictionaries’s `abbreviations` data set.
- **replace**: A vector of long form replacements if a data frame is not supplied to the abbreviation argument.
- **ignore.case**: logical. If TRUE replaces without regard to capitalization.

Value

Returns a vector with abbreviations replaced.

See Also

`bracketX`, `qprep`, `replace_contraction`, `replace_number`, `replace_symbol`
replace_contraction

Examples

```r
## Not run:
x <- c("Mr. Jones is here at 7:30 p.m.",
    "Check it out at www.github.com/trinker/qdap",
    "i.e. He's a sr. dr.; the best in 2012 A.D.",
    "the robot at t.s. is 10ft. 3in.")

replace_abbreviation(x)

# create abbreviation and replacement vectors
abv <- c("in.", "ft.", "t.s.")
repl <- c("inch", "feet", "talkstats")

replace_abbreviation(x, abv, repl)

(KEY <- rbind(abbreviations, data.frame(abv = abv, rep = repl)))
replace_abbreviation(x, KEY)

## End(Not run)
```

Description

This function replaces contractions with long form.

Usage

```r
replace_contraction(text.var, contraction = qdapDictionaries::contractions,
    replace = NULL, ignore.case = TRUE, sent.cap = TRUE)
```

Arguments

- **text.var**: The text variable.
- **contraction**: A two column key of contractions (column 1) and expanded form replacements (column 2) or a vector of contractions. Default is to use qdapDictionaries’s `contractions` data set.
- **replace**: A vector of expanded form replacements if a data frame is not supplied to the contraction argument.
- **ignore.case**: logical. If TRUE replaces without regard to capitalization.
- **sent.cap**: logical. If TRUE capitalizes the beginning of every sentence.

Value

Returns a vector with contractions replaced.
replace_number

See Also
bracketX, qprep, replace_abbreviation, replace_number, replace_symbol

Examples

```r
## Not run:
x <- c("Mr. Jones isn't going.",
    "Check it out what's going on.",
    "He's here but didn't go.",
    "the robot at t.s. wasn't nice",
    "he'd like it if i'd go away")

replace_contraction(x)
## End(Not run)
```

replace_number  
Replace Numbers With Text Representation

Description
Replaces numeric represented numbers with words (e.g., 1001 becomes one thousand one).

Usage
replace_number(text.var, num.paste = TRUE, remove = FALSE)

Arguments

- `text.var` The text variable.
- `num.paste` logical. If TRUE the elements of larger numbers are separated with spaces. If FALSE the elements will be joined without spaces.
- `remove` logical. If TRUE numbers are removed from the text.

Value
Returns a vector with abbreviations replaced.

Note
The user may want to use replace_ordinal first to remove ordinal number notation. For example replace_number would turn "21st" into "twenty onest", whereas replace_ordinal would generate "twenty first".

References
See Also

bracketX, qprep, replace_abbreviation, replace_contraction, replace_symbol, replace_ordinal

Examples

```r
## Not run:
x <- c("I like 346,457 ice cream cones.", "They are 99 percent good")
y <- c("I like 346457 ice cream cones.", "They are 99 percent good")
replace_number(x)
replace_number(y)
replace_number(x, FALSE)
replace_number(x, remove=TRUE)

## End(Not run)
```

---

`replace_ordinal` *Replace Mixed Ordinal Numbers With Text Representation*

Description

Replaces mixed text/numeric represented ordinal numbers with words (e.g., "1st" becomes "first").

Usage

```
replace_ordinal(text.var, num.paste = TRUE, remove = FALSE)
```

Arguments

- `text.var` The text variable.
- `num.paste` logical. If TRUE a the elements of larger numbers are separated with spaces. If FALSE the elements will be joined without spaces.
- `remove` logical. If TRUE ordinal numbers are removed from the text.

Note

Currently only implemented for ordinal values 1 through 100

See Also

bracketX, qprep, replace_abbreviation, replace_contraction, replace_symbol, replace_number
Examples

```
## Not run:
x <- c(
   "I like the 1st one not the 22nd one.",
   "For the 100th time stop!
)
replace_ordinal(x)
replace_ordinal(x, FALSE)
replace_ordinal(x, remove = TRUE)
I like the 1st 1 not the 22nd 1." %>% replace_ordinal %>% replace_number
```

## End(Not run)

---

**replace_symbol**  
*Replace Symbols With Word Equivalents*

**Description**

This function replaces symbols with word equivalents (e.g., @ becomes "at").

**Usage**

```
replace_symbol(text.var, dollar = TRUE, percent = TRUE, pound = TRUE, 
at = TRUE, and = TRUE, with = TRUE)
```

**Arguments**

- `text.var`: The text variable.
- `dollar`: logical. If TRUE replaces dollar sign ($) with "dollar".
- `percent`: logical. If TRUE replaces percent sign (%) with "percent".
- `pound`: logical. If TRUE replaces pound sign (#) with "number".
- `at`: logical. If TRUE replaces at sign (@) with "at".
- `and`: logical. If TRUE replaces and sign (&) with "and".
- `with`: logical. If TRUE replaces with sign (w/) with "with".

**Value**

Returns a character vector with symbols replaced.

**See Also**

`bracketX`, `qprep`, `replace_abbreviation`, `replace_contraction`, `replace_number`
Examples

```r
## Not run:
x <- c("I am @ Jon's & Jim's w/ Marry",
       "I owe $41 for food",
       "two is 10% of a ")
replace_symbol(x)

## End(Not run)
```

---

### rm_row

**Remove Rows That Contain Markers**

**Description**

- `rm_row` - Remove rows from a data set that contain a given marker/term.
- `rm_empty_row` - Removes the empty rows of a data set that are common in reading in data (default method in `read.transcript`).

**Usage**

```r
rm_row(dataframe, search.column, terms, contains = FALSE, ignore.case = FALSE, keep.rownames = FALSE, ...)
rm_empty_row(dataframe)
```

**Arguments**

- `dataframe` - A dataframe object.
- `search.column` - Column name to search for markers/terms.
- `terms` - Terms/markers of the rows that are to be removed from the dataframe. The term/marker must appear at the beginning of the string and is case sensitive.
- `contains` - logical. If TRUE `rm_row` searches for the terms anywhere within the string. If FALSE `rm_row` searches only the beginning of the string.
- `ignore.case` - logical. If TRUE case is ignored during matching, if FALSE the pattern matching is case sensitive.
- `keep.rownames` - logical. If TRUE the original, non-sequential, rownames will be used.
- `...` - Other arguments passed to `grepl`.

**Value**

- `rm_row` - returns a dataframe with the termed/markered rows removed.
- `rm_empty_row` - returns a dataframe with empty rows removed.
Examples

```r
## Not run:
# rm_row EXAMPLE:
rm_row(DATA, "person", c("sam", "greg"))
rm_row(DATA, 1, c("sam", "greg"))
rm_row(DATA, "state", c("Comp"))
rm_row(DATA, "state", c("I "))
rm_row(DATA, "state", c("you"), contains = TRUE, ignore.case=TRUE)

#rm_empty_row EXAMPLE:
(dat <- rbind.data.frame(DATA[, c(1, 4)], matrix(rep(" ", 4),
   ncol =2, dimnames=list(12:13, colnames(DATA)[c(1, 4)]))
rm_empty_row(dat)
## End(Not run)
```

---

**rm_stopwords**

**Remove Stop Words**

**Description**

Removal of stop words in a variety of contexts.

%sw% - Binary operator version of `rm_stopwords` that defaults to separate = FALSE.

**Usage**

```r
rm_stopwords(text.var, stopwords = qdapDictionaries::Top25Words,
   unlist = FALSE, separate = TRUE, strip = FALSE, unique = FALSE,
   char.keep = NULL, names = FALSE, ignore.case = TRUE,
   apostrophe.remove = FALSE, ...)

rm_stop(text.var, stopwords = qdapDictionaries::Top25Words, unlist = FALSE,
   separate = TRUE, strip = FALSE, unique = FALSE, char.keep = NULL,
   names = FALSE, ignore.case = TRUE, apostrophe.remove = FALSE, ...)

text.var %sw% stopwords
```

**Arguments**

- `text.var` A character string of text or a vector of character strings.
- `stopwords` A character vector of words to remove from the text. qdap has a number of data sets that can be used as stop words including: Top200Words, Top100Words, Top25Words. For the tm package's traditional English stop words use tm::stopwords("english").
- `unlist` logical. If TRUE unlists into one vector. General use intended for when separate is FALSE.
- `separate` logical. If TRUE separates sentences into words. If FALSE retains sentences.
strip logical. If TRUE strips the text of all punctuation except apostrophes.
unique logical. If TRUE keeps only unique words (if unlist is TRUE) or sentences (if unlist is FALSE). General use intended for when unlist is TRUE.
char.keep If strip is TRUE this argument provides a means of retaining supplied character(s).
names logical. If TRUE will name the elements of the vector or list with the original text.var.
ignore.case logical. If TRUE stopwords will be removed regardless of case. Additionally, case will be stripped from the text. If FALSE stop word removal is contingent upon case. Additionally, case is not stripped.
apostrophe.remove logical. If TRUE removes apostrophe’s from the output.
... further arguments passed to strip function.

Value

Returns a vector of sentences, vector of words, or (default) a list of vectors of words with stop words removed. Output depends on supplied arguments.

See Also

strip, bag.o_words, stopwords

Examples

```r
# Not run:
rm_stopwords(DATA$state)
rm_stopwords(DATA$state, tm::stopwords("english"))
rm_stopwords(DATA$state, Top200Words)
rm_stopwords(DATA$state, Top200Words, strip = TRUE)
rm_stopwords(DATA$state, Top200Words, separate = FALSE)
rm_stopwords(DATA$state, Top200Words, separate = FALSE, ignore.case = FALSE)
rm_stopwords(DATA$state, Top200Words, unlist = TRUE)
rm_stopwords(DATA$state, Top200Words, unlist = TRUE, strip=TRUE)
rm_stop(DATA$state, Top200Words, unlist = TRUE, unique = TRUE)

c("I like it alot", "I like it too") %sw% qdapDictionaries::Top25Words

# End(Not run)
```

---

```r
sample.time.span

Minimal Time Span Data Set

Description

A fictitious dataset containing time spans for codes A and B.
```
Usage
  data(sample.time.span)

Format
  A data frame with 9 rows and 6 variables

Details
  • code. The qualitative code.
  • start. The integer start time.
  • end. The integer end time.
  • Start. The chron start time.
  • End. The chron end time.
  • variable. An arbitrary single time repeated measures variable (ignore).

scores  Generic Scores Method

Description
  Access the scores dataframes from select qdap outputs.

Usage
  scores(x, ...)

Arguments
  x          A qdap object (list) with a dataframe of scores (e.g., fry, formality).
  ...        Arguments passed to scores method of other classes.

Value
  Returns a data.frame of scores.

See Also
  counts
  proportions
  preprocessed
Readability Measures

scores.automated_readability_index

Description

scores.automated_readability_index - View scores from automated_readability_index.

Usage

```r
## S3 method for class 'automated_readability_index'
scores(x, ...)
```

Arguments

- `x` The automated_readability_index object.
- `...` ignored

Details

automated_readability_index Method for scores

scores.character_table

Term Counts

Description

View character_table scores.

Usage

```r
## S3 method for class 'character_table'
scores(x, ...)
```

Arguments

- `x` The character_table object.
- `...` ignored

Details

character_table Method for scores
**scores.coleman_liau**  
*Readability Measures*

**Description**

`scores.coleman_liau` - View scores from `coleman_liau`.

**Usage**

```r
## S3 method for class 'coleman_liau'
scores(x, ...)
```

**Arguments**

- `x`  
The `coleman_liau` object.
- `...`  
Ignored

**Details**

`coleman_liau` Method for scores

---

**scores.end_mark_by**  
*Question Counts*

**Description**

View `end_mark_by` scores.

**Usage**

```r
## S3 method for class 'end_mark_by'
scores(x, ...)
```

**Arguments**

- `x`  
The `end_mark_by` object.
- `...`  
Ignored

**Details**

`end_mark_by` Method for scores
scores.flesch_kincaid  Readability Measures

Description

scores.flesch_kincaid - View scores from flesch_kincaid.

Usage

## S3 method for class 'flesch_kincaid'
scores(x, ...)

Arguments

x The flesch_kincaid object.
...

Details

flesch_kincaid Method for scores

scores.formality  Formality

Description

View formality scores.

Usage

## S3 method for class 'formality'
scores(x, ...)

Arguments

x The formality object.
...

Details

formality Method for scores
scores.fry

Readability Measures

Description

scores.fry - View scores from fry.

Usage

```r
## S3 method for class 'fry'
scores(x, ...)
```

Arguments

- `x`: The fry object.
- `...`: ignored

Details

fry Method for scores

scores.lexical_classification

Lexical Classification

Description

scores.lexical_classification - View scores from lexical_classification.

Usage

```r
## S3 method for class 'lexical_classification'
scores(x, ...)
```

Arguments

- `x`: The lexical_classification object.
- `...`: ignored

Details

lexical_classification Method for scores
### scores.linsear_write

**Readability Measures**

**Description**

`scores.linsear_write` - View scores from `linsear_write`.

**Usage**

```r
## S3 method for class 'linsear_write'
scores(x, ...)
```

**Arguments**

- `x` The `linsear_write` object.
- `...` ignored

**Details**

`linsear_write` Method for scores

### scores.object_pronoun_type

**Question Counts**

**Description**

View `object_pronoun_type` scores.

**Usage**

```r
## S3 method for class 'object_pronoun_type'
scores(x, ...)
```

**Arguments**

- `x` The `object_pronoun_type` object.
- `...` ignored

**Details**

`object_pronoun_type` Method for scores
### scores.polarity

**Description**

scores.polarity - View scores from `polarity`.

**Usage**

```r
## S3 method for class 'polarity'
scores(x, ...)
```

**Arguments**

- `x` - The polarity object.
- `...` - ignored

**Details**

`polarity` Method for scores

---

### scores.pos_by

**Description**

View pos_by scores.

**Usage**

```r
## S3 method for class 'pos_by'
scores(x, ...)
```

**Arguments**

- `x` - The pos_by object.
- `...` - ignored

**Details**

`pos_by` Method for scores
### `scores.pronoun_type`  
#### Question Counts

**Description**  
View pronoun type scores.

**Usage**  
```r
## S3 method for class 'pronoun_type'
scores(x, ...)
```

**Arguments**  
- `x`: The pronoun type object.  
- `...`: ignored

**Details**  
pronoun type Method for scores

### `scores.question_type`  
#### Question Counts

**Description**  
View question type scores.

**Usage**  
```r
## S3 method for class 'question_type'
scores(x, ...)
```

**Arguments**  
- `x`: The question type object.  
- `...`: ignored

**Details**  
question_type Method for scores
**scores.SMOG**

*Readability Measures*

**Description**

scores.SMOG - View scores from SMOG.

**Usage**

```r
## S3 method for class 'SMOG'
scores(x, ...)
```

**Arguments**

- `x` The SMOG object.
- `...` ignored

**Details**

SMOG Method for scores

---

**scores.subject_pronoun_type**

*Question Counts*

**Description**

View subject_pronoun_type scores.

**Usage**

```r
## S3 method for class 'subject_pronoun_type'
scores(x, ...)
```

**Arguments**

- `x` The subject_pronoun_type object.
- `...` ignored

**Details**

subject_pronoun_type Method for scores
**scores.termco**  
*Term Counts*

**Description**  
View termco scores.

**Usage**  
```r  
## S3 method for class 'termco'
scores(x, ...)
```

**Arguments**  
- `x` The termco object.
- `...` ignored

**Details**  
termco Method for scores

**scores.word_length**  
*Word Length Counts*

**Description**  
View word_length scores.

**Usage**  
```r  
## S3 method for class 'word_length'
scores(x, ...)
```

**Arguments**  
- `x` The word_length object.
- `...` ignored

**Details**  
word_length Method for scores
scores.word_position  Word Position

Description

View word_position scores.

Usage

```r
## S3 method for class 'word_position'
scores(x, ...)
```

Arguments

- `x` The `word_position` object.
- `...` ignored

Details

word_position Method for scores

scores.word_stats  Word Stats

Description

View question_type scores.

Usage

```r
## S3 method for class 'word_stats'
scores(x, ...)
```

Arguments

- `x` The `question_type` object.
- `...` ignored

Details

question_type Method for scores
Description

Use to clean text variables when importing a new data set. Removes extra white spaces other textual anomalies that may cause errors.

Usage

scrubber(text.var, num2word = FALSE, rm.quote = TRUE, fix.comma = TRUE, fix.space = TRUE, ...)

Arguments

text.var The text variable.
num2word logical If TRUE replaces a numbers with text representations.
rm.quote logical If TRUE removes any ".
fix.comma logical If TRUE removes any spaces before a comma.
fix.space logical. If TRUE extra spaces before endmarks are removed.
... Other arguments passed to replace_number.

Value

Returns a parsed character vector.

See Also

strip

Examples

## Not run:
x <- c("I like 456 dogs\t, don't you?", 'The end'")
scrubber(x)
scrubber(x, TRUE)

## End(Not run)
Search

Search Columns of a Data Frame

Description

Search - Find terms located in columns of a data frame.

boolean_search - Conducts a Boolean search for terms/strings within a character vector.

%bs% - Binary operator version of boolean_search.

Usage

Search(dataframe, term, column.name = NULL, max.distance = 0.02, ...)

boolean_search(text.var, terms, ignore.case = TRUE, values = FALSE, exclude = NULL, apostrophe.remove = FALSE, char.keep = NULL, digit.remove = FALSE)

text.var %bs% terms

Arguments

dataframe A dataframe object to search.
term A character string to search for.
column.name Optional column of the data frame to search (character name or integer index).
max.distance Maximum distance allowed for a match. Expressed either as integer, or as a fraction of the pattern length times the maximal transformation cost (will be replaced by the smallest integer not less than the corresponding fraction).
text.var The text variable.
terms A character string(s) to search for. The terms are arranged in a single string with AND (use AND or & & to connect terms together) and OR (use OR or | | to allow for searches of either set of terms. Spaces may be used to control what is searched for. For example using " I " on c("I'm", "I want", "in") will result in FALSE TRUE FALSE whereas "I" will match all three (if case is ignored).
ignore.case logical. If TRUE case is ignored.
values logical. ShouLd the values be returned or the index of the values.
exclude Terms to exclude from the search. If one of these terms is found in the sentence it cannot be returned.
apostrophe.remove logical. If TRUE removes apostrophes from the text before examining.
char.keep A character vector of symbol character (i.e., punctuation) that strip should keep. The default is to strip everything except apostrophes. termco attempts to auto detect characters to keep based on the elements in match.list.
digit.remove logical. If TRUE strips digits from the text before counting. termco attempts to auto detect if digits should be retained based on the elements in match.list.

... Other arguments passed to agrep.
Details

The terms string is first split by the OR separators into a list. Next the list of vectors is split on the AND separator to produce a list of vectors of search terms. Each sentence is matched against the terms. For a sentence to be counted it must fit all of the terms in an AND Boolean or one of the conditions in an OR Boolean.

Value

Search - Returns the rows of the data frame that match the search term.

boolean_search - Returns the values (or indices) of a vector of strings that match given terms.

See Also

trans_context
termco

Examples

```r
## Not run:
## Dataframe search:
(SampDF <- data.frame("islands"=names(islands)[1:32],mtcars, row.names=NULL))

Search(SampDF, "Cuba", "islands")
Search(SampDF, "New", "islands")
Search(SampDF, "Ho")
Search(SampDF, "Ho", max.distance = 0)
Search(SampDF, "Axel Heiberg")
Search(SampDF, 19) #too much tolerance in max.distance
Search(SampDF, 19, max.distance = 0)
Search(SampDF, 19, "qsec", max.distance = 0)

##Boolean search:
boolean_search(DATA$state, " I ORliar&&stinks")
boolean_search(DATA$state, " I &&.", values=TRUE)
boolean_search(DATA$state, " I OR.", values=TRUE)
boolean_search(DATA$state, " I &&.")

## Exclusion:
boolean_search(DATA$state, " I ||.", values=TRUE)
boolean_search(DATA$state, " I ||.", exclude = c("way", "truth"), values=TRUE)

## From stackoverflow: http://stackoverflow.com/q/19640562/1000343
dat <- data.frame(x = c("Doggy", "Hello", "Hi Dog", "Zebra"), y = 1:4)
z <- data.frame(z =c("Hello", "Dog"))
dat[boolean_search(dat$x, paste(z$z, collapse = "OR")), ]

## Binary operator version
dat[dat$x %bs% paste(z$z, collapse = "OR"), ]

## Passing to `trans_context`
```
sentiment_frame

inds <- boolean_search(DATA.SPLIT$state, " I &, || I &!", ignore.case = FALSE)
with(DATA.SPLIT, trans_context(state, person, inds=inds))

(inds2 <- boolean_search(raj$dialogue, spaste(paste(negation.words,
collapse = " || ")))
trans_context(raj$dialogue, raj$person, inds2)

## End(Not run)

---

sentiment_frame  

### Power Score (Sentiment Analysis)

**Description**

sentiment_frame - Generate a sentiment lookup hash table for use with the `xxx.frame` argument of various sentiment functions.

**Usage**

```r
sentiment_frame(positives, negatives, pos.weights = 1, neg.weights = -1)
```

**Arguments**

- **positives**: A character vector of positive words.
- **negatives**: A character vector of negative words.
- **pos.weights**: A vector of weights to weight each positive word by. Length must be equal to length of positives or length 1 (if 1 weight will be recycled).
- **neg.weights**: A vector of weights to weight each negative word by. Length must be equal to length of negatives or length 1 (if 1 weight will be recycled).

---

sentSplit  

### Sentence Splitting

**Description**

sentSplit - Splits turns of talk into individual sentences (provided proper punctuation is used). This procedure is usually done as part of the data read in and cleaning process.

sentCombine - Combines sentences by the same grouping variable together.

TOT - Convert the tot column from `sentSplit` to turn of talk index (no sub sentence). Generally, for internal use.

sent_detect - Detect and split sentences on endmark boundaries.

sent_detect_nlp - Detect and split sentences on endmark boundaries using openNLP & NLP utilities which matches the old version of the openNLP package's now removed sentDetect function.
Usage

```r
sentSplit(dataframe, text.var, rm.var = NULL, endmarks = c("?", ".", "!", "|"), incomplete.sub = TRUE, rm.bracket = TRUE, stem.col = FALSE, text.place = "right", verbose = is.global(2), ...)
```

```r
sentCombine(text.var, grouping.var = NULL, as.list = FALSE)
```

```r
TOT(tot)
```

```r
sent_detect(text.var, endmarks = c("?", ".", "!", "|"), incomplete.sub = TRUE, rm.bracket = TRUE, ...)
```

```r
sent_detect_nlp(text.var, ...)
```

Arguments

dataframe A dataframe that contains the person and text variable.
text.var The text variable.
rm.var An optional character vector of 1 or 2 naming the variables that are repeated measures (This will restart the "tot" column).
endmarks A character vector of endmarks to split turns of talk into sentences.
incomplete.sub logical. If TRUE detects incomplete sentences and replaces with "|".
rm.bracket logical. If TRUE removes brackets from the text.
stem.col logical. If TRUE stems the text as a new column.
text.place A character string giving placement location of the text column. This must be one of the strings "original", "right" or "left".
verbose logical. If TRUE select diagnostics from check_text are reported.
grouping.var The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
as.list logical. If TRUE returns the output as a list. If FALSE the output is returned as a dataframe.
tot A tot column from a sentSplit output.
... Additional options passed to stem2df.

Value

sentSplit - returns a dataframe with turn of talk broken apart into sentences. Optionally a stemmed version of the text variable may be returned as well.
sentCombine - returns a list of vectors with the continuous sentences by grouping.var pasted together. returned as well.
TOT - returns a numeric vector of the turns of talk without sentence sub indexing (e.g. 3.2 become 3).
sent_detect - returns a character vector of sentences split on endmark.
sent_detect - returns a character vector of sentences split on endmark.
Warning

`sentSplit` requires the dialogue (text) column to be cleaned in a particular way. The data should contain qdap punctuation marks (c("?", ".", "!", ":")) at the end of each sentence. Additionally, extraneous punctuation such as abbreviations should be removed (see `replace_abbreviation`). Trailing sentences such as I thought I... will be treated as incomplete and marked with "|" to denote an incomplete/trailing sentence.

Suggestion

It is recommended that the user runs `check_text` on the output of `sentSplit`'s text column.

Author(s)

Dason Kurkiewicz and Tyler Rinker <tyler.rinker@gmail.com>.

See Also

`bracketX, incomplete_replace, stemRdf, TOT`

Examples

```r
## Not run:
## `sentSplit` EXAMPLE:
(out <- sentSplit(DATA, "state"))
out %>% check_text()  ## check output text
sentSplit(DATA, "state", stem.col = TRUE)
sentSplit(DATA, "state", text.place = "left")
sentSplit(DATA, "state", text.place = "original")
sentSplit(raj, "dialogue")[1:20, ]

## plotting
plot(out)
plot(out, grouping.var = "person")

out2 <- sentSplit(DATA2, "state", rm.var = c("class", "day"))
plot(out2)
plot(out2, grouping.var = "person")
plot(out2, grouping.var = "person", rm.var = "day")
plot(out2, grouping.var = "person", rm.var = c("day", "class"))

## `sentCombine` EXAMPLE:
(dat <- sentSplit(DATA, "state")
sentCombine(dat$state, dat$person)
truncdf(sentCombine(dat$state, dat$sex), 50)

## `TOT` EXAMPLE:
(dat <- sentSplit(DATA, "state")
TOT(dat$tot)

## `sent_detect`
(sent_detect(DATA$state)
```
## space_fill

### Description

Replace spaces in words groups that should be grouped together.

### Usage

```r
space_fill(text.var, terms, sep = "~~", rm.extra = TRUE,
            ignore.case = TRUE, fixed = FALSE, ...)
```

### Arguments

- `text.var`: The text variable.
- `terms`: A character vector of grouped word terms to insert a new separating/space character.
- `sep`: A character string to separate the terms.
- `rm.extra`: logical. Should trailing, leading and > 1 continuous white spaces be removed?
- `ignore.case`: logical. If FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.
- `fixed`: logical. If TRUE, pattern is a string to be matched as is. Overrides all conflicting arguments.
- `...`: Other arguments passed to `gsub`.

### Details

`snow_fill` is useful for keeping grouped words together. Many functions in `qdap` take a `char keeper` or `char2space` argument. This can be used to prepare multi word phrases (e.g., proper nouns) as a single unit.

### Value

Returns a character vector with extra, trailing and/or leading spaces removed.

### Note

`link[qdap][strip]` by default does not remove the double tilde "~~" character.
Examples

```r
## Not run:
x <- c("I want to hear the Dr. Martin Luther King Jr. speech.",
       "I also want to go to the white House to see President Obama speak.")

keeps <- c("Dr. Martin Luther King Jr.", "The White House", "President Obama")
space_fill(x, keeps)
strip(space_fill(x, keeps))

## End(Not run)
```

---

**spaste**  
Add Leading/Trailing Spaces

---

**Description**

Adds trailing and/or leading spaces to a vector of terms.

**Usage**

```r
spaste(terms, trailing = TRUE, leading = TRUE)
```

**Arguments**

- `terms`: A character vector of terms to insert trailing and/or leading spaces.
- `trailing`: logical. If `TRUE` inserts a trailing space in the terms.
- `leading`: logical. If `TRUE` inserts a leading space in the terms.

**Value**

Returns a character vector with trailing and/or leading spaces.

**Examples**

```r
## Not run:
spaste(Top25Words)
spaste(Top25Words, FALSE)
spaste(Top25Words, trailing = TRUE, leading = FALSE) # or
spaste(Top25Words, , FALSE)

## End(Not run)
```
speakerSplit 

Break and Stretch if Multiple Persons per Cell

Description
Look for cells with multiple people and create separate rows for each person.

Usage
speakerSplit(dataframe, person.var = 1, sep = c("and", ",", ";", "\&"),
  track.reps = FALSE)

Arguments
dataframe A dataframe that contains the person variable.
person.var The person variable to be stretched.
sep The separator(s) to search for and break on. Default is: c("and", ",", ";")
track.reps logical. If TRUE leaves the row names of person variable cells that were repeated
  and stretched.

Value
Returns an expanded dataframe with person variable stretched and accompanying rows repeated.

Examples
## Not run:
DATA$person <- as.character(DATA$person)
DATA$person[c(1, 4, 6)] <- c("greg, sally, & sam",
  "greg, sally", "sam and sally")

speakerSplit(DATA)
speakerSplit(DATA, track.reps=TRUE)

DATA$person[c(1, 4, 6)] <- c("greg_sallysam",
  "greg.sally", "sam; sally")

speakerSplit(DATA, sep = c(".", ",", ";"))

DATA <- qdap::DATA # reset DATA

## End(Not run)
Stem Text

**Description**

stemmer - Stems a vector of text strings (A wrapper for the **tm** package’s **stemDocument**.

stem_words - Wrapper for stemmer that stems a vector of words.

stem2df - Wrapper for stemmer that stems a vector of text strings and returns a dataframe with the vector added.

**Usage**

```r
stemmer(text.var, rm.bracket = TRUE, capitalize = TRUE, warn = TRUE,
char.keep = "~~", ...)

stem_words(...)

stem2df(dataframe, text.var, stem.name = NULL, ...)
```

**Arguments**

- **text.var** The text variable. In stemmer this is a vector text string. For stem2df this is a character vector of length one naming the text column.
- **rm.bracket** logical. If TRUE brackets are removed from the text.
- **capitalize** logical. If TRUE selected terms are capitalized.
- **warn** logical. If TRUE warns about rows not ending with standard qdap punctuation endmarks.
- **char.keep** A character vector of symbols that should be kept within sentences.
- **...** Various:
  - stemmer - Other arguments passed to capitalizer
  - stem_words - Words or terms.
  - stem2df - Other arguments passed to stemmer
- **dataframe** A dataframe object.
- **stem.name** A character vector of length one for the stemmed column. If NULL defaults to "stem.text".

**Value**

stemmer - returns a character vector with stemmed text.

stem_words - returns a vector of individually stemmed words.

stem2df - returns a dataframe with a character vector with stemmed text.

**See Also**

capitalizer
Examples

```r
## Not run:
#stemmer EXAMPLE:  
stemmer(DATA$state)  
out1 <- stemmer(raj$dialogue)  
htruncdf(out1, 20, 60)

#stem_words EXAMPLE:  
stem_words(doggies, jumping, swims)

#stem2df EXAMPLE:  
out2 <- stem2df(DATA, "state", "new")  
thtruncdf(out2, 30)
```

## End(Not run)

### Description

Strip text of unwanted characters.

- `strip.character` - factor method for `strip`.
- `strip.factor` - factor method for `strip`.
- `strip.default` - factor method for `strip`.
- `strip.list` - factor method for `strip`.

### Usage

```r
strip(x, char.keep = "~~", digit.remove = TRUE, apostrophe.remove = TRUE,  
      lower.case = TRUE)
```

```r
## S3 method for class 'character'  
strip(x, char.keep = "~~", digit.remove = TRUE,  
      apostrophe.remove = TRUE, lower.case = TRUE)
```

```r
## S3 method for class 'factor'  
strip(x, char.keep = "~~", digit.remove = TRUE,  
      apostrophe.remove = TRUE, lower.case = TRUE)
```

```r
## Default S3 method:  
strip(x, char.keep = "~~", digit.remove = TRUE,  
      apostrophe.remove = TRUE, lower.case = TRUE)
```

```r
## S3 method for class 'list'  
strip(x, char.keep = "~~", digit.remove = TRUE,  
      apostrophe.remove = TRUE, lower.case = TRUE)
```
**Arguments**

- **x**  
  The text variable.

- **char.keep**  
  A character vector of symbols (i.e., punctuation) that `strip` should keep. The default is to strip every symbol except apostrophes and a double tilde "~~". The double tilde "~~" is included for a convenient means of keeping word groups together in functions that split text apart based on spaces. To remove double tildes "~~" set char.keep to NULL.

- **digit.remove**  
  logical. If TRUE strips digits from the text.

- **apostrophe.remove**  
  logical. If TRUE removes apostrophes from the output.

- **lower.case**  
  logical. If TRUE forces all alpha characters to lower case.

**Value**

Returns a vector of text that has been stripped of unwanted characters.

**See Also**

`rm_stopwords`

**Examples**

```r
## Not run:
data$state #no strip applied
strip(data$state)
strip(data$state, apostrophe.remove=FALSE)
strip(data$state, char.keep = c("?", "."))

## End(Not run)
```

---

**strWrap**  
Wrap Character Strings to Format Paragraphs

**Description**

A wrapper for `as.character` that writes to the Mac/Windows clipboard.

**Usage**

```r
strWrap(text = "clipboard", width = 70, copy2clip = interactive())
```

**Arguments**

- **text**  
  character vector, or an object which can be converted to a character vector by `as.character`.

- **width**  
  A positive integer giving the target column for wrapping lines in the output.

- **copy2clip**  
  logical. If TRUE attempts to copy the output to the clipboard.
Value

Prints a wrapped text vector to the console and copies the wrapped text to the clipboard on a Mac or Windows machine.

See Also

strwrap

Examples

```r
## Not run:
x <- paste2(DATA$state, sep = " ")
strWrap(x)
strWrap(x, 10)
# should be copied to the clipboard on a Mac or Windows machine.
## End(Not run)
```

subject_pronoun_type  Count Subject Pronouns Per Grouping Variable

Description

Count the number of subject pronouns per grouping variables.

Usage

```r
subject_pronoun_type(text.var, grouping.var = NULL, 
subject.pronoun.list = NULL, ...)
```

Arguments

- **text.var** The text variable
- **grouping.var** The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- **subject.pronoun.list** A named list of subject pronouns. See Details for more.
- **...** Other arguments passed to termco

Details

The following subject pronoun categories are the default searched terms:

- I - `c(" i'd ", " i'll ", " i'm ", " i've ", " i ")`
- we - `c(" we'd ", " we'll ", " we're ", " we've ", " we ")`
- you - `c(" you'd ", " you'll ", " you're ", " you've ", " you ", " your ")`
subject_pronoun_type

- he - c(" he’d ", " he’ll ", " he’s ", " he ")
- she - c(" she’d ", " she’ll ", " she’s ", " she ")
- it - c(" it’d ", " it’ll ", " it’s ", " it ")
- they - c(" they’d ", " they’ll ", " they’re ", " they’ve ", " they ")

Value

Returns a list, of class "subject_pronoun_type", of data frames regarding subject pronoun word counts:

preprocessed  List of uncollapsed dataframes (raw, prop, rnp) of the class "termco" that contain all searchable subject pronouns.

raw  raw word counts by grouping variable

prop  proportional word counts by grouping variable; proportional to each individual’s subject pronoun use

rnp  a character combination data frame of raw and proportional subject pronoun use

See Also

object_pronoun_type, pronoun_type

Examples

```r
## Not run:
data <- pres_debates2012
data <- data[dat$person %in% qcv(ROMNEY, OBAMA), ]
(out <- subject_pronoun_type(dat$dialogue, dat$person))
plot(out)
plot(out, 2)
plot(out, 3)
plot(out, 3, ncol=2)

scores(out)
counts(out)
proportions(out)
preprocessed(out)

plot(scores(out))
plot(counts(out))
plot(proportions(out))

## End(Not run)
```
Summarize a cmspans object

Usage

```r
## S3 method for class 'cmspans'
summary(object, grouping.var = NULL, rm.var = NULL,
         total.span = TRUE, aggregate = FALSE, percent = TRUE, digits = 2, ...)```

Arguments

- `object` The cmspans object
- `grouping.var` The grouping variables. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `rm.var` An optional single vector or list of 1 or 2 of repeated measures to aggregate by.
- `total.span` logical or an option list of vectors (length 1 or 2) of the total duration of the event. If FALSE the "total" column is divided by the sum of the total duration for all codes in that rm.var to arrive at "total_percent". If TRUE and object is from `cm_time2long` the difference for the time span from the `transcript_time_span` of the list used in `cm_time2long` are utilized to divide the "total" column. The user may also provide a list of vectors with each vector representing a single total time duration or provide the start and end time of the event. The user may give input in numeric seconds or in character "hh:mm:ss" form.
- `aggregate` logical. If TRUE the output will be aggregated (i.e., the output will collapse the rm.var).
- `percent` logical. If TRUE output given as percent. If FALSE the output is proportion.
- `digits` Integer; number of decimal places to round when printing.
- `...` Other argument passed to `qheat` in plot (ignored in summary).

See Also

- `plot.sum_cmspans`

Examples

```r
## Not run:
## Example 1
foo <- list(
  person_greg = qcv(terms='7:11, 20:24, 30:33, 49:56'),
  person_researcher = qcv(terms='42:48'),
  person_sally = qcv(terms='25:29, 37:41'),
  person_sam = qcv(terms='1:6, 16:19, 34:36'),
)```
person_teacher = qcv(terms='12:15'),
adult_0 = qcv(terms='1:11, 16:41, 49:56'),
adult_1 = qcv(terms='12:15, 42:48'),
AA = qcv(terms='1"'),
BB = qcv(terms='1:2, 3:10, 19"'),
CC = qcv(terms='1:9, 100:150")
)

foo2 <- list(
  person_greg = qcv(terms='7:11, 20:24, 30:33, 49:56'),
  person_researcher = qcv(terms='42:48'),
  person_sally = qcv(terms='25:29, 37:41'),
  person_sam = qcv(terms='1:6, 16:19, 34:36'),
  person_teacher = qcv(terms='12:15'),
  adult_0 = qcv(terms='1:11, 16:41, 49:56'),
  adult_1 = qcv(terms='12:15, 42:48'),
  AA = qcv(terms='40"'),
  BB = qcv(terms='50:90"'),
  CC = qcv(terms='60:90, 100:120, 150"'),
  DD = qcv(terms='")
)

v <- cm_2long(foo, foo2, v.name = "time")
plot(v)
summary(v)
plot(summary(v))

## Example 2
x <- list(
  transcript_time_span = qcv(00:00 - 1:12:00),
  A = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00"),
  B = qcv(terms = "2.40, 3.01:3.02, 5.01, 6.02:7.00, 9.00, 1.12:00:1.19:01"),
  C = qcv(terms = "2.40:3.00, 5.01, 6.02:7.00, 9.00, 17.01")
)

z <- cm_2long(x)

summary(z)
summary(z, total.span = FALSE)
summary(z, total.span = c(0, 3333))
summary(z, total.span = c("00:01:00", "03:02:00"))
plot(summary(z))

## suppress printing measurement units
suppressMessages(print(summary(z)))

## remove print method
as.data.frame(summary(z))

## End(Not run)
**summary.wdf**

*Summarize a wdf object*

**Description**

Summarize a wdf object with familiar tm package look.

**Usage**

```r
## S3 method for class 'wdf'
summary(object, ...)
```

**Arguments**

- `object` : The wdf object
- `...` : Ignored.

**Details**

- **Non-/sparse entries** is the ratio of non-zeros to zero counts. **Sparsity** is that ratio represented as a percent. **Hapax legomenon** is the number(percent) of terms that appear only once in the dialogue. **Dis legomenon** is the number(percent) of terms that appear exactly two times once.

**Examples**

```r
## Not run:
x <- with(DATA, wdf(state, list(sex, adult)))
summary(x)

## End(Not run)
```

---

**summary.wfm**

*Summarize a wfm object*

**Description**

Summarize a wfm object with familiar tm package look.

**Usage**

```r
## S3 method for class 'wfm'
summary(object, ...)
```

**Arguments**

- `object` : The wfm object
- `...` : Ignored.
Details

Non-/sparse entries is the ratio of non-zeros to zero counts. Sparsity is that ratio represented as a percent. Hapax legomenon is the number(%) of terms that appear only once in the dialogue. Dis legomenon is the number(%) of terms that appear exactly two times once.

Examples

```r
## Not run:
x <- with(DATA, wfm(state, list(sex, adult)))
summary(x)

## End(Not run)
```

### syllable_sum

#### Description

- **syllable_sum** - Count the number of syllables per row of text.
- **syllable_count** - Count the number of syllables in a single text string.
- **polysyllable_sum** - Count the number of polysyllables per row of text.
- **combo_syllable_sum** - Count the number of both syllables and polysyllables per row of text.

#### Usage

```r
syllable_sum(text.var, parallel = FALSE, ...)
syllable_count(text, remove.bracketed = TRUE, algorithm.report = FALSE,
               env = qdap::env.syl)
polysyllable_sum(text.var, parallel = FALSE)
combo_syllable_sum(text.var, parallel = FALSE)
```

#### Arguments

- `text.var` The text variable
- `parallel` logical. If TRUE attempts to run the function on multiple cores. Note that this may not mean a speed boost if you have one core or if the data set is smaller as the cluster takes time to create.
- `text` A single character vector of text.
- `remove.bracketed` logical. If TRUE brackets are removed from the analysis.
- `algorithm.report` logical. If TRUE generates a report of words not found in the dictionary (i.e., syllables were calculated with an algorithm).
env

A lookup environment to lookup the number of syllables in found words.

... Other arguments passed to syllable_count.

Details

The worker function of all the syllable functions is `syllable_count`, though it is not intended for direct use on a transcript. This function relies on a combined dictionary lookup (based on the Nettalk Corpus (Sejnowski & Rosenberg, 1987)) and backup algorithm method.

Value

`syllable_sum` - returns a vector of syllable counts per row.

`syllable_count` - returns a dataframe of syllable counts and algorithm/dictionary uses and, optionally, a report of words not found in the dictionary.

`polysyllable_sum` - returns a vector of polysyllable counts per row.

`combo_syllable_sum` - returns a dataframe of syllable and polysyllable counts per row.

References


Examples

```r
## Not run:
syllable_count("Robots like Dason lie.")
syllable_count("Robots like Dason lie.", algorithm.report = TRUE)

syllable_sum(DATA$state)
x1 <- syllable_sum(rajSPLIT$dialogue)
plot(x1)
cumulative(x1)

polysyllable_sum(DATA$state)
x2 <- polysyllable_sum(rajSPLIT$dialogue)
plot(x2)
cumulative(x2)

combo_syllable_sum(DATA$state)
x3 <- combo_syllable_sum(rajSPLIT$dialogue)
plot(x3)
cumulative(x3)

## End(Not run)
```
Description

synonyms - Search for synonyms that match term(s).
synonyms_frame - Generate a synonym lookup hash key for use with the synonym.frame argument in the synonym function.

Usage

synonyms(terms, return.list = TRUE, multiwords = TRUE, report.null = TRUE, synonym.frame = qdapDictionaries::key.syn)
syn(terms, return.list = TRUE, multiwords = TRUE, report.null = TRUE, synonym.frame = qdapDictionaries::key.syn)
synonyms_frame(synonym.list, prior.frame)
syn_frame(synonym.list, prior.frame)

Arguments

terms The terms to find synonyms for.
return.list logical. If TRUE returns the output for multiple synonyms as a list by search term rather than a vector.
multiwords logical. IF TRUE retains vector elements that contain phrases (defined as having one or more spaces) rather than a single word.
report.null logical. If TRUE reports the words that no match was found at the head of the output.
synonym.frame A dataframe or hash key of positive/negative words and weights.
synonym.list A named list of lists (or vectors) of synonyms.
prior.frame A prior synonyms data.frame in the format produced by synonyms_frame.

Value

Returns a list of vectors or vector of possible words that match term(s).

References

The synonyms dictionary (see key.syn) was generated by web scraping the Reverso Online Dictionary. The word list fed to Reverso is the unique words from the combination of DICTIONARY and labMT.
Examples

```r
## Not run:
synonyms(c("the", "cat", "job", "environment", "read", "teach"))
head(syn(c("the", "cat", "job", "environment", "read", "teach"),
       return.list = FALSE, 30)
syn(c("the", "cat", "job", "environment", "read", "teach"), multiwords = FALSE)

## User defined synonym lookup
syn_dat <- list(
  like = list(c("want", "desire"), c("love", "care")),
  show = list(c("reveal"), c("movie", "opera")),
  R = c("old friend", "statistics language")
)
synonyms_frame(syn_dat)
syn(c("R", "show"), synonym.frame = syn_frame(syn_dat))
syns.hash <- syn_frame(syn_dat, prior.frame = qdapDictionaries::key.syn)
syn(c("R", "show", "like", "robot"), synonym.frame = syns.hash)

## End(Not run)
```

---

t.DocumentTermMatrix  Transposes a DocumentTermMatrix object

Description

Transposes a DocumentTermMatrix object

Usage

```r
## S3 method for class 'DocumentTermMatrix'
t(x, ...)
```

Arguments

- `x` The DocumentTermMatrix object
- `...` ignored
**t.TermDocumentMatrix**  
Transposes a TermDocumentMatrix object

**Description**

Transposes a TermDocumentMatrix object

**Usage**

```r
## S3 method for class 'TermDocumentMatrix'
t(x, ...)
```

**Arguments**

- `x`: The TermDocumentMatrix object
- `...`: ignored

---

**termco**  
Search For and Count Terms

**Description**

termco - Search a transcript by any number of grouping variables for categories (themes) of grouped root terms. While there are other termco functions in the termco family (e.g., termco_d) termco is a more powerful and flexible wrapper intended for general use.

termco_d - Search a transcript by any number of grouping variables for root terms.

term_match - Search a transcript for words that exactly match term(s).

termco2mat - Convert a termco dataframe to a matrix for use with visualization functions (e.g., heatmap.R).

**Usage**

```r
termco(text.var, grouping.var = NULL, match.list, short.term = TRUE,  
ignore.case = TRUE, elim.old = TRUE, percent = TRUE, digits = 2,  
apostrophe.remove = FALSE, char.keep = NULL, digit.remove = NULL,  
zero.replace = 0, ...)
```

```r
termco_d(text.var, grouping.var = NULL, match.string, short.term = FALSE,  
ignore.case = TRUE, zero.replace = 0, percent = TRUE, digits = 2,  
apostrophe.remove = FALSE, char.keep = NULL, digit.remove = TRUE, ...)
```

```r
term_match(text.var, terms, return.list = TRUE, apostrophe.remove = FALSE)
```

```r
termco2mat(dataframe, drop.wc = TRUE, short.term = TRUE,  
rm.zerocol = FALSE, no.quote = TRUE, transform = TRUE,  
trim.terms = TRUE)
```
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text.var</td>
<td>The text variable.</td>
</tr>
<tr>
<td>grouping.var</td>
<td>The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.</td>
</tr>
<tr>
<td>match.list</td>
<td>A list of named character vectors.</td>
</tr>
<tr>
<td>short.term</td>
<td>logical. If TRUE column names are trimmed versions of the match list, otherwise the terms are wrapped with ‘term(phrase)’.</td>
</tr>
<tr>
<td>ignore.case</td>
<td>logical. If TRUE case is ignored.</td>
</tr>
<tr>
<td>elim.old</td>
<td>logical. If TRUE eliminates the columns that are combined together by the named match.list.</td>
</tr>
<tr>
<td>percent</td>
<td>logical. If TRUE output given as percent. If FALSE the output is proportion.</td>
</tr>
<tr>
<td>digits</td>
<td>Integer; number of decimal places to round when printing.</td>
</tr>
<tr>
<td>apostrophe.remove</td>
<td>logical. If TRUE removes apostrophes from the text before examining.</td>
</tr>
<tr>
<td>char.keep</td>
<td>A character vector of symbol character (i.e., punctuation) that strip should keep. The default is to strip everything except apostrophes. termco attempts to auto detect characters to keep based on the elements in match.list.</td>
</tr>
<tr>
<td>digit.remove</td>
<td>logical. If TRUE strips digits from the text before counting. termco attempts to auto detect if digits should be retained based on the elements in match.list.</td>
</tr>
<tr>
<td>zero.replace</td>
<td>Value to replace 0 values with.</td>
</tr>
<tr>
<td>match.string</td>
<td>A vector of terms to search for. When using inside of term_match the term(s) must be words or partial words but do not have to be when using termco_d (i.e., they can be phrases, symbols etc.).</td>
</tr>
<tr>
<td>terms</td>
<td>The terms to search for in the text.var. Similar to match.list but these terms must be words or partial words rather than multiple words and symbols.</td>
</tr>
<tr>
<td>return.list</td>
<td>logical. If TRUE returns the output for multiple terms as a list by term rather than a vector.</td>
</tr>
<tr>
<td>dataframe</td>
<td>A termco (or termco_d) dataframe or object.</td>
</tr>
<tr>
<td>drop.wc</td>
<td>logical. If TRUE the word count column will be dropped.</td>
</tr>
<tr>
<td>rm.zerocol</td>
<td>logical. If TRUE any column containing all zeros will be removed from the matrix.</td>
</tr>
<tr>
<td>no.quote</td>
<td>logical. If TRUE the matrix will be printed without quotes if it’s character.</td>
</tr>
<tr>
<td>transform</td>
<td>logical. If TRUE the matrix will be transformed.</td>
</tr>
<tr>
<td>trim.terms</td>
<td>logical. If TRUE trims the column header/names to ensure there is not a problem with spacing when using in other R functions.</td>
</tr>
<tr>
<td>...</td>
<td>Other argument supplied to strip.</td>
</tr>
</tbody>
</table>

### Value

termco & termco_d - both return a list, of class "termco", of data frames and information regarding word counts:
raw word counts by grouping variable
prop proportional word counts by grouping variable; proportional to each individual’s word use
rnp a character combination data frame of raw and proportional
zero_replace value to replace zeros with; mostly internal use
percent The value of percent used for plotting purposes.
digits integer value of number of digits to display; mostly internal use

term_match - returns a list or vector of possible words that match term(s).
termco2mat - returns a matrix of term counts.

Warning
Percentages are calculated as a ratio of counts of match.list elements to word counts. Word counts do not contain symbols or digits. Using symbols, digits or small segments of full words (e.g., “to”) could total more than 100%.

Note
The match.list/match.string is (optionally) case and character sensitive. Spacing is an important way to grab specific words and requires careful thought. Using "read" will find the words "bread", "read" "reading", and "ready". If you want to search for just the word "read" you’d supply a vector of c(" read", " reads", " reading", " reader"). To search for non character arguments (i.e., numbers and symbols) additional arguments from strip must be passed.

See Also
termco_c, colcomb2class

Examples

```r
## Not run:
#termco examples:

term <- c("the ", "she", " wh")
(out <- with(raj.act.1, termco(dialogue, person, term)))

plot(out)
scores(out)
plot(scores(out))
counts(out)
plot(counts(out))
proportions(out)
plot(proportions(out))

# General form for match.list as themes
#
# ml <- list(
#   cat1 = c(),
```
# cat2 = c()
# catn = c()
#
ml <- list(
  cat1 = c(" the ", " a ", " an "),
  cat2 = c(" I" ),
  "good",
  the = c("the", " the", " the", "the")
)

(dat <- with(raj.act.1, termco(dialogue, person, ml)))
scores(dat) # useful for presenting in tables
counts(dat) # prop and raw counts are useful for performing calculations
proportions(dat)
datb <- with(raj.act.1, termco(dialogue, person, ml,
  short.term = FALSE, elim.old=FALSE))
ltuncdf(datb, 20, 6)

(dat2 <- data.frame(dialogue=c("@bryan is bryan good @br",
  "indeed", "@ brian"), person=qcv(A, B, A)))
ml2 <- list(wrds=c("bryan", "indeed"), "@", bryan=c("bryan", "@ br", "@br"))
with(dat2, termco(dialogue, person, match.list=ml2))
with(dat2, termco(dialogue, person, match.list=ml2, percent = FALSE))

DATA$state[1] <- "12 4 rgfr r0ffrg0"
termco(DATA$state, DATA$person, '0', digit.remove=FALSE)
DATA <- qdap::DATA

# Using with term_match and exclude
exclude(term_match(DATA$state, qcv(th), FALSE), "truth")
termco(DATA$state, DATA$person, exclude(term_match(DATA$state, qcv(th),
  FALSE), "truth"))
MTCH.LST <- exclude(term_match(DATA$state, qcv(th, i)), qcv(truth, stinks))
termco(DATA$state, DATA$person, MTCH.LST)
syns <- synonyms("doubt")
syns[1]
termco(DATA$state, DATA$person, unlist(syns[1]))
synonyms("doubt", FALSE)
termco(DATA$state, DATA$person, list(doubt = synonyms("doubt", FALSE)))
termco(DATA$state, DATA$person, syns)

# termco_d examples:
termco_d(DATA$state, DATA$person, c(" the", " i"))
termco_d(DATA$state, DATA$person, c(" the", " i"), ignore.case=FALSE)
termco_d(DATA$state, DATA$person, c(" the ", " i"))

# termco2mat example:
MTCH.LST <- exclude(term_match(DATA$state, qcv(a, i)), qcv(is, it, am, shall))
termco_obj <- termco(DATA$state, DATA$person, MTCH.LST)
termco2mat(termco_obj)
plot(termco_obj)
plot(termco_obj, label = TRUE)
plot(termco_obj, label = TRUE, text.color = "red")
plot(termco_obj, label = TRUE, text.color = "red", lab.digits = 3)

## REVERSE TERMCO (return raw words found per variable)
df <- data.frame(x=1:6,
                  y = c("the fluffy little bat", "the man was round like a ball",
                         "the fluffy little bat", "the man was round like a ball",
                         "he ate the chair", "cough, cough"),
                  stringsAsFactors = FALSE)

l <- list("bat", "man", "ball", "heavy")
z <- counts(termco(df$y, qdapTools::id(df), l))[[-2]]
counts2list(z[, -1], z[, 1])

## politness
politness <- c("please", "excuse me", "thank you", "you welcome",
                "you're welcome", "i'm sorry", "forgive me", "pardon me")

with(pres_debates2012, termco(dialogue, person, politness))
with(hamlet, termco(dialogue, person, politness))

## Term Use Percentage per N Words
dat <- with(raj, chunker(dialogue, person, n.words = 100, rm.unequal = TRUE))
dat2 <- list2df(dat, "Dialogue", "Person")
dat2["Duration"] <- unlist(lapply(dat, id, pad = FALSE))
dat2 <- qdap_df(dat2, "Dialogue")

Top5 <- sapply(split(raj$dialogue, raj$person), wc, FALSE) %>%
       sort(decreasing = TRUE) %>%
       list2df("wordcount", "person") %>%
       \`
         \[1:5, 2\]

propdat <- dat2 %<% termco(list(Person, Duration), as.list(Top25Words[1:5]), percent = FALSE) %>%
           proportions %>%
           colsplit2df %>%
           reshape2::melt(id = c("Person", "Duration", "word.count"), variable = "Word") %>%
           dplyr::filter(Person %in% Top5)

head(propdat)

ggplot(propdat, aes(y = value, x = Duration, group = Person, color = Person)) +
   geom_line(size = 1.25) +
   facet_grid(Word ~ , scales = "free_y") +
   ylab("Percent of Word Use") +
   xlab("Per 100 Words") +
   scale_y_continuous(labels = percent)
termco_c

Combine Columns from a termco Object

Description

Combines the columns of a termco object. Generally intended for internal use but documented for completeness.
Usage

termco_c(termco.object, combined.columns, new.name, short.term = TRUE, zero.replace = NULL, elim.old = TRUE, percent = NULL, digits = 2)

Arguments

termco.object An object generated by either termco, termco_d or termco_c.
combined.columns The names/indexes of the columns to be combined.
new.name A character vector of length one to name the new combined column.
short.term logical. If TRUE column names are trimmed versions of the match list, otherwise the terms are wrapped with 'term(phrase)'
zero.replace Value to replace zeros with.
elim.old logical. If TRUE eliminates the columns that are combined together by the named match.list.
percent logical. If TRUE output given as percent. If FALSE the output is proportion.
digits Integer; number of decimal places to round when printing.

Value

Returns a return a list, of class "termco", of data frames and information regarding word counts:

raw raw word counts by grouping variable
prop proportional word counts by grouping variable; proportional to each individual’s word use
rnp a character combination data frame of raw and proportional
zero_replace value to replace zeros with; mostly internal use
percent The value of percent used for plotting purposes.
digits integer value od number of digits to display; mostly internal use

See Also

termco

Title

Add Title to Select qdap Plots

Description

Add title to select qdap objects that store a plot.
Usage

Title(object)

Title(object) <- value

Arguments

object A select qdap object that stores a plot.
value The value to assign to title.

tot_plot  Visualize Word Length by Turn of Talk

Description

Uses a bar graph to visualize patterns in sentence length and grouping variables by turn of talk.

Usage

tot_plot(dataframe, text.var, grouping.var = NULL, facet.vars = NULL, 
tot = TRUE, transform = FALSE, ncol = NULL, ylab = NULL, 
  xlab = NULL, bar.space = 0, scale = NULL, space = NULL, plot = TRUE)

Arguments

dataframe A dataframe that contains the text variable and optionally the grouping.var and tot variables.
text.var The text variable (character string).
grouping.var The grouping variables to color by. Default NULL colors everything in "black". Also takes a single grouping variable or a list of 1 or more grouping variables.
facet.vars An optional single vector or list of 1 or 2 to facet by.
tot The turn of talk variable (character string). May be TRUE (assumes "tot" is the variable name), FALSE (use row numbers), or a character string of the turn of talk column.
transform logical. If TRUE the repeated facets will be transformed from stacked to side by side.
ncol number of columns. gantt_wrap uses facet_wrap rather than facet_grid.
ylab Optional y label.
xlab Optional x label.
bar.space The amount space between bars (ranging between 1 and 0).
scale Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x","free_y")
space

If "fixed", the default, all panels have the same size. If "free_y" their height will be proportional to the length of the y scale; if "free_x" their width will be proportional to the length of the x scale; or if "free" both height and width will vary. This setting has no effect unless the appropriate scales also vary.

plot

logical. If TRUE the plot will automatically plot. The user may wish to set to FALSE for use in knitr, sweave, etc. to add additional plot layers.

Value

Invisibly returns the ggplot2 object.

Examples

## Not run:
dataframe <- sentSplit(DATA, "state")
tot_plot(dataframe, "state")
tot_plot(DATA, "state", tot=FALSE)
tot_plot(dataframe, "state", bar.space=.03)
tot_plot(dataframe, "state", "sex")
tot_plot(dataframe, "state", "person", tot = "sex")
tot_plot(mrajal, "dialogue", "fam.aff", tot=FALSE)
tot_plot(mrajal, "dialogue", "died", tot=FALSE)
tot_plot(mrajal, "dialogue", c("sex", "fam.aff"), tot=FALSE) +
  scale_fill_hue(l=40)
tot_plot(mrajal, "dialogue", c("sex", "fam.aff"), tot=FALSE)+
  scale_fill_brewer(palette="Spectral")
tot_plot(mrajal, "dialogue", c("sex", "fam.aff"), tot=FALSE)+
  scale_fill_brewer(palette="Set1")

## repeated measures
rajsPLIT2 <- do.call(rbind, lapply(split(rajSPLIT, rajSPLIT$act), head, 25))
tot_plot(rajsPLIT2, "dialogue", "fam.aff", facet.var = "act")

## add mean and +/- 2 sd
tot_plot(mrajal, "dialogue", grouping.var = c("sex", "fam.aff"), tot=FALSE)+
  scale_fill_brewer(palette="Set1") +
  geom_hline(aes(yintercept=mean(word.count))) +
  geom_hline(aes(yintercept=mean(word.count) + (2 *sd(word.count)))) +
  geom_hline(aes(yintercept=mean(word.count) + (3 *sd(word.count)))) +
  geom_text(parse=TRUE, hjust=0, vjust=0, family="serif", size = 4, aes(x = 2,
    y = mean(word.count) + 2, label = "bar(x)")) +
  geom_text(hjust=0, vjust=0, family="serif", size = 4, aes(x = 1,
    y = mean(word.count) + (2 *sd(word.count)) + 2, label = "+2 sd") +
  geom_text(hjust=0, vjust=0, family="serif", size = 4, aes(x = 1,
    y = mean(word.count) + (3 *sd(word.count)) + 2, label = "+3 sd")

## End(Not run)
trans_cloud

Word Clouds by Grouping Variable

Description

Produces word clouds with optional theme coloring by grouping variable.

Usage

```r
trans_cloud(text.var = NULL, grouping.var = NULL, word.list = NULL, stem = FALSE, target.words = NULL, expand.target = TRUE, target.exclude = NULL, stopwords = NULL, min.freq = 1, caps = TRUE, caps.list = NULL, random.order = FALSE, rot.per = 0, cloud.colors = NULL, title = TRUE, cloud.font = NULL, title.font = NULL, title.color = "black", title.padj = -4.5, title.location = 3, title.cex = NULL, title.names = NULL, proportional = FALSE, max.word.size = NULL, min.word.size = 0.5, legend = NULL, legend.cex = 0.8, legend.location = c(-0.03, 1.03), char.keep = "~~", char2space = "~")
```

Arguments

- `text.var`: The text variable.
- `grouping.var`: The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `word.list`: A frequency word list passed from `word_list`.
- `stem`: logical. If TRUE the `text.var` will be stemmed.
- `target.words`: A named list of vectors of words whose length corresponds to `cloud.colors` (+1 length in cloud colors for non-matched terms).
- `expand.target`: logical. If TRUE `agrep` will be used to expand the `target.words`.
- `target.exclude`: A vector of words to exclude from the `target.words`.
- `stopwords`: Words to exclude from the cloud.
- `min.freq`: An integer value indicating the minimum frequency a word must appear to be included.
- `caps`: logical. If TRUE selected words will be capitalized.
- `caps.list`: A vector of words to capitalize (caps must be TRUE).
- `random.order`: Plot words in random order. If false, they will be plotted in decreasing frequency.
- `rot.per`: Proportion words with 90 degree rotation.
- `cloud.colors`: A vector of colors equal to the length of target words +1.
- `title`: logical. If TRUE adds a title corresponding to the `grouping.var`.
- `cloud.font`: The font family of the cloud text.
- `title.font`: The font family of the cloud title.
trans_cloud

```r
## Not run:
terms <- list(
  I=c("i", "i'm"),
  mal=qcv(stinks, dumb, distrust),
  articles=qcv(the, a, an),
  pronoun=qcv(we, you)
)

with(DATA, trans_cloud(state, person, target.words=terms,
  cloud.colors=qcv(red, green, blue, black, gray65),
  expand.target=FALSE, proportional=TRUE, legend=c(names(terms), "other")))

with(DATA, trans_cloud(state, person, target.words=terms,
  stopwords=exclude(with(DATA, unique(bag_o_words(state)))))
```

---

### Value

Returns a series of word cloud plots with target words (themes) colored.

### See Also

wordcloud, gradient_cloud

### Examples

```r
## Not run:
terms <- list(
  I=c("i", "i'm"),
  mal=qcv(stinks, dumb, distrust),
  articles=qcv(the, a, an),
  pronoun=qcv(we, you)
)

with(DATA, trans_cloud(state, person, target.words=terms,
  cloud.colors=qcv(red, green, blue, black, gray65),
  expand.target=FALSE, proportional=TRUE, legend=c(names(terms), "other")))

with(DATA, trans_cloud(state, person, target.words=terms,
  stopwords=exclude(with(DATA, unique(bag_o_words(state)))))
```
```r
unique(unlist(terms)),
cloud.colors=qcv(red, green, blue, black, gray65),
expand.target=FALSE, proportional=TRUE, legend=names(terms))

#color the negated phrases opposite:
DATA <- qdap::DATA
DATA[1, 4] <- "This is not good!"
DATA[8, 4] <- "I don't distrust you."

DATA$state <- space_fill(DATA$state, paste0(negation.words, " "),
  rm.extra = FALSE)

txt <- gsub("--", " ", breaker(DATA$state))
rev.neg <- sapply(negation.words, paste, negative.words)
rev.pos <- sapply(negation.words, paste, positive.words)

tw <- list(
  positive = c(positive.words, rev.neg[rev.neg %in% txt]),
  negative = c(negative.words, rev.pos[rev.pos %in% txt])
)

with(DATA, trans_cloud(state, person,
  target.words=tw,
  cloud.colors=qcv(darkgreen, red, gray65),
  expand.target=FALSE, proportional=TRUE, legend=names(tw)))

DATA <- qdap::DATA  ## Reset DATA

## End(Not run)
```

---

**trans_context**  
*Print Context Around Indices*

**Description**

Print (or save to an external file) n text elements before and after indices.

**Usage**

```r
trans_context(text.var, grouping.var, inds, n.before = 3, tot = TRUE,
  n.after = n.before, ord.inds = TRUE)
```

**Arguments**

- `text.var` The text variable.
- `grouping.var` The grouping variables. Also takes a single grouping variable or a list of 1 or more grouping variables.
trans_context

inds A list of integer indices to print context for.
n.before The number of rows before the indexed occurrence.
tot logical. If TRUE condenses sub-units (e.g., sentences) into turns of talk for that grouping.var.
n.after The number of rows after the indexed occurrence.
ord.inds logical. If TRUE inds is ordered least to greatest.

Value

Returns a dataframe of the class "qdap_context" that can be printed (i.e., saved) in flexible outputs. The dataframe can be printed as a dataframe style or pretty text output. The resulting file contains n rows before and after each index of a vector of indices.

See Also

boolean_search, question_type, end_mark

Examples

## Not run:
(x <- with(DATA, trans_context(state, person, inds=c(1, 4, 7, 11))))
print(x, pretty=FALSE)
print(x, double_space = FALSE)
print(x, file="foo.xlsx")
print(x, file="foo.csv")
print(x, file="foo.txt")
print(x, file="foo.txt", pretty = FALSE)
print(x, file="foo.doc")

## With 'end_mark'
inds1 <- which(end_mark(DATA.SPLIT[, "state"] == "?"))
with(DATA.SPLIT, trans_context(state, person, ind=inds1))
with(DATA.SPLIT, trans_context(state, person, n.before = 0, ind=inds1))

## With 'boolean_search'
inds2 <- boolean_search(DATA.SPLIT$state, " I &\&."
with(DATA.SPLIT, trans_context(state, person, ind=inds2))

inds3 <- boolean_search(DATA$state, " I ||.
with(DATA.SPLIT, trans_context(state, person, ind=inds3))
with(DATA.SPLIT, trans_context(state, list(person, sex), ind=inds3))
with(DATA.SPLIT, trans_context(state, list(sex, adult), ind=inds3))

inds4 <- boolean_search(raj$dialogue, spaste(paste(negation.words, collapse = " || ")))
trans_context(raj$dialogue, raj$person, ind=inds4)

## With 'question_type'
(x <- question_type(DATA.SPLIT$state, DATA.SPLIT$person))

## All questions
with(DATA.SPLIT, trans_context(state, person, ind=x$inds))
## Specific question types

```r
y <- x[["raw"]]
inds5 <- y[y[, "q.type"] %in% qcv(what, how), "n.row"]
with(DATA.SPLIT, trans_context(state, person, inds=inds5))
with(DATA.SPLIT, trans_context(state, person, inds=inds5, tot=F))
```

## trans_venn

### Venn Diagram by Grouping Variable

**Description**

Produce a Venn diagram by grouping variable.

**Usage**

```r
trans_venn(text.var, grouping.var, stopwords = NULL, rm.duplicates = TRUE,
title = TRUE, title.font = NULL, title.color = "black",
title.cex = NULL, title.name = NULL, legend = TRUE, legend.cex = 0.8,
legend.location = "bottomleft", legend.text.col = "black",
legend.horiz = FALSE, ...)
```

**Arguments**

- `text.var`: The text variable.
- `grouping.var`: The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `stopwords`: Words to exclude from the analysis.
- `rm.duplicates`: logical. If TRUE removes the duplicated words from the analysis (only single usage is considered).
- `title`: logical. IF TRUE adds a title corresponding to the grouping.var.
- `title.font`: The font family of the cloud title.
- `title.color`: A character vector of length one corresponding to the color of the title.
- `title.cex`: Character expansion factor for the title. NULL and NA are equivalent to 1.0
- `title.name`: A title for the plot.
- `legend`: logical. If TRUE uses the names from the target.words list corresponding to cloud.colors.
- `legend.cex`: Character expansion factor for the legend. NULL and NA are equivalent to 1.0.
- `legend.location`: The x and y co-ordinates to be used to position the legend. The location may also be specified by setting x to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location.
trim

Description

Remove leading/trailing white space.

Usage

_trim(x)_

Arguments

x  
The text variable.

Value

Returns a vector with the leading/trailing white spaces removed.
Examples

```r
## Not run:
(x <- c("talkstats.com ", " really? ", " yeah"))
Trim(x)

## End(Not run)
```

---

**type_token_ratio**

**Type-Token Ratio**

**Description**

Calculate type-token ratio by grouping variable.

**Usage**

`type_token_ratio(text.var, grouping.var = NULL, n.words = 1000, ...)`

**Arguments**

- `text.var` The text variable
- `grouping.var` The grouping variables. Default `NULL` generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `n.words` An integer specifying the number of words in each chunk.
- `...` ignored.

**Value**

Returns a list of class `type_text_ratio`. This object contains a type-token ratio for the overall text and a data frame type-token ratios per grouping variable.

**References**


**Examples**

```r
with(raj, type_token_ratio(dialogue, person))
plot(with(raj, type_token_ratio(dialogue, person)))
```
### unique_by

**Find Unique Words by Grouping Variable**

**Description**

Find unique words used by grouping variable.

**Usage**

`unique_by(text.var, grouping.var)`

**Arguments**

- `text.var`: The text variable
- `grouping.var`: The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.

**Value**

Returns a list of unique words by grouping variable.

**Examples**

```r
## Not run:
dat <- pres_debates2012[pres_debates2012$time == "time 3", ]
with(dat, unique_by(dialogue, person))
with(pres_debates2012, unique_by(dialogue, list(time, person)))
with(DATA, unique_by(state, person))
## End(Not run)
```

### vertex_apply

**Apply Parameter to List of Igraph Vertices/Edges**

**Description**

- `vertex_apply`: Uniformly apply `igraph` vertex plotting parameters to a list of `igraph` objects.
- `edge_apply`: Uniformly apply `igraph` edge plotting parameters to a list of `igraph` objects.

**Usage**

- `vertex_apply(x, ..., hold.ends = NULL)`
- `edge_apply(x, ..., hold.ends = c("label.color"))`
visual

Arguments

x A list of igraph objects.
hold.ends A vector of parameters passed to ... that should not be altered for the first and last (ends) objects in the list.
... Arguments passed igraph's V and E. See Drawing Graphs for more.

Value

Returns a list of igraph objects.

Examples

## Not run:
x <- with(DATA.SPLIT, polarity(state, person))
bg_black <- Animate(x, neutral="white")
print(bg_black)

bgb <- vertex_apply(bg_black, label="grey80", size=20, color="grey40")
bgb <- edge_apply(bgb, label.color="yellow")
print(bgb, bg="black", pause=.75)

## End(Not run)

visual

Generic visual Method

Description

Access the visual-graph-plot object from select qdap outputs.

Usage

visual(x, ...)

Arguments

x A qdap object (list) with a visual-graph-plot object (e.g., discourse_map).
... Arguments passed to visual method of other classes.

Value

Returns a plot object.

See Also

scores, counts, preprocessed, proportions
visual.discourse_map  Discourse Map

Description

visual.discourse_map - View visual from discourse_map.

Usage

```
## S3 method for class 'discourse_map'
visual(x, ...)
```

Arguments

- `x`: The discourse_map object.
- `...`: ignored

Details

discourse_map Method for visual

---

weight  Weight a qdap Object

Description

Weight a word_proximity object.

Usage

```
weight(x, type = "scale", ...)
```

Arguments

- `x`: A qdap object with a weight method.
- `type`: A weighting type of: c("scale_log", "scale", "rev_scale", "rev_scale_log", "log", "sqrt", "scale_sqrt", "rev_sqrt", "rev_scale_sqrt"). The weight type section name (i.e. A_B_C where A, B, and C are sections) determines what action will occur: log will use log, sqrt will use sqrt, scale will standardize the values, rev will multiply by -1 to give the inverse sign. This enables a comparison similar to correlations rather than distance.
- `...`: ignored.

Value

Returns a weighted list of matrices.
Note

A constant of .000000000001 is added to each element when log is used to deal with the problem of log(0).

---

**wfm**  
*Word Frequency Matrix*

**Description**

- **wfm** - Generate a word frequency matrix by grouping variable(s).
- **wfm.wfdf** - wdf method for wfm.
- **wfm.character** - character method for wfm.
- **wfm.factor** - factor method for wfm.
- **wfd** - Generate a word frequency data frame by grouping variable.
- **wfm.expanded** - Expand a word frequency matrix to have multiple rows for each word.
- **wfm.combined** - Combines words (rows) of a word frequency matrix (wdf) together.
- **weight** - Weight a word frequency matrix for analysis where such weighting is sensible.
- **weight.wdf** - Weight a word frequency matrix for analysis where such weighting is sensible.
- **as.wfm** - Attempts to coerce a matrix to a wfm.
- **as.wfm.matrix** - matrix method for as.wfm used to convert matrices to a wfm.
- **as.wfm.default** - Default method for as.wfm used to convert matrices to a wfm.
- **as.wfm.TermDocumentMatrix** - TermDocumentMatrix method for as.wfm used to a TermDocumentMatrix to a wfm.
- **as.wfm.DocumentTermMatrix** - DocumentTermMatrix method for as.wfm used to a DocumentTermMatrix to a wfm.
- **as.wfm.data.frame** - data.frame method for as.wfm used to convert matrices to a wfm.
- **as.wfm.wfdf** - wdf method for as.wfm used to convert matrices to a wfm.
- **as.wfm.Corpus** - Corpus method for as.wfm used to convert matrices to a wfm.
- **wfm.Corpus** - Corpus method for wfm.

**Usage**

```r
wfm(text.var = NULL, grouping.var = NULL, output = "raw",
     stopwords = NULL, char2space = "~\~", ...)## S3 method for class 'wdf'
wfm(text.var = NULL, grouping.var = NULL, output = "raw",
     stopwords = NULL, char2space = "~\~", ...)## S3 method for class 'character'
wfm(text.var = NULL, grouping.var = NULL,
```


output = "raw", stopwords = NULL, char2space = "~~", ...)

## S3 method for class 'factor'
wf(df(text.var = NULL, grouping.var = NULL, output = "raw",
stopwords = NULL, char2space = "~~", ...)

wf(df(text.var, grouping.var = NULL, stopwords = NULL, margins = FALSE,
output = "raw", digits = 2, char2space = "~~", ...)

wf_expanded(text.var, grouping.var = NULL, ...)

wf_combine(wf.obj, word.lists, matrix = TRUE)

## S3 method for class 'wf'
weight(x, type = "prop", ...)

## S3 method for class 'wf'
weight(x, type = "prop", ...)

as.wf(x, ...)

## S3 method for class 'matrix'
as.wf(x, ...)

## Default S3 method:
as.wf(x, ...)

## S3 method for class 'TermDocumentMatrix'
as.wf(x, ...)

## S3 method for class 'DocumentTermMatrix'
as.wf(x, ...)

## S3 method for class 'data.frame'
as.wf(x, ...)

## S3 method for class 'wfdf'
as.wf(x, ...)

## S3 method for class 'Corpus'
as.wf(x, col = "docs", row = "text", ...)

## S3 method for class 'Corpus'
w(df(text.var, ...)

Arguments

text.var The text variable.
The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.

Output type (either "proportion" or "percent").

A vector of stop words to remove.

A vector of characters to be turned into spaces. If char.keep is NULL, char2space will activate this argument.

Logical. If TRUE provides grouping.var and word variable totals.

An integer indicating the number of decimal places (round) or significant digits (signif) to be used. Negative values are allowed.

A wfm or wfdf object.

A list of character vectors of words to pass to wfm_combine

Logical. If TRUE returns the output as a wfm rather than a wfdf object.

An object with words for row names and integer values.

The type of weighting to use: c("prop", "max", "scaled"). All weight by column. "prop" uses a proportion weighting and all columns sum to 1. "max" weights in proportion to the max value; all values are integers and column sums may not be equal. "scaled" uses scale to scale with center = FALSE; output is not integer and column sums may not be equal.

The column name (generally not used).

The row name (generally not used).

Other arguments supplied to Corpus or TermDocumentMatrix. If as.wfm this is other arguments passed to as.wfm methods (currently ignored).

wfm - returns a word frequency of the class matrix.

wfdf - returns a word frequency of the class data.frame with a words column and optional margin sums.

wfm_expanded - returns a matrix similar to a word frequency matrix (wfm) but the rows are expanded to represent the maximum usages of the word and cells are dummy coded to indicate that number of uses.

wfm_combine - returns a word frequency matrix (wfm) or dataframe (wfdf) with counts for the combined word.lists merged and remaining terms (else).

weight - Returns a weighted matrix for use with other R packages. The output is not of the class "wfm".

as.wfm - Returns a matrix of the class "wfm".

Words can be kept as one by inserting a double tilde ("~~"), or other character strings passed to char2space, as a single word/entry. This is useful for keeping proper names as a single unit.
Examples

## Not run:
## word frequency matrix (wfm) example:
with(DATA, wfm(state, list(sex, adult)))[1:15, ]
with(DATA, wfm(state, person))[1:15, ]
Filter(with(DATA, wfm(state, list(sex, adult))), 5)
with(DATA, wfm(state, list(sex, adult)))

## Filter particular words based on max/min values in wfm
v <- with(DATA, wfm(state, list(sex, adult)))
Filter(v, 5)
Filter(v, 5, count.apostrophe = FALSE)
Filter(v, 5, 7)
Filter(v, 4, 4)
Filter(v, 3, 4)
Filter(v, 3, 4, stopwords = Top25Words)

## insert double tilde ("~~") to keep phrases (i.e., first last name)
alts <- c(" fun", " I ")
state2 <- space_fill(DATA$state, alts, rm.extra = FALSE)
with(DATA, wfm(state2, list(sex, adult)))[1:18, ]

## word frequency dataframe (wfdf) example:
with(DATA, wfdf(state, list(sex, adult)))[1:15, ]
with(DATA, wfdf(state, person))[1:15, ]

## wfm_expanded example:
z <- wfm(DATA$state, DATA$person)
wfm_expanded(z)[30:45, ] #two "you"s

## wfm_combine examples:
=====
## raw no margins (will work)
x <- wfm(DATA$state, DATA$person)

## raw with margin (will work)
y <- wfdf(DATA$state, DATA$person, margins = TRUE)

## Proportion matrix
z2 <- wfm(DATA$state, DATA$person, output="proportion")

WL1 <- c(y[, 1])
WL2 <- list(c("read", "the", "a"), c("you", "your", "you're"))
WL3 <- list(bob = c("read", "the", "a"), yous = c("you", "your", "you're"))
WL4 <- list(bob = c("read", "the", "a"), yous = c("a", "you", "your", "you're"))
WL5 <- list(yous = c("you", "your", "you're"))
WL6 <- list(c("you", "your", "you're")) #no name so will be called words 1
WL7 <- c("you", "your", "your")
wfm_combine(z2, WL2) #Won't work not a raw frequency matrix
wfm_combine(x, WL2) #Works (raw and no margins)
wfm_combine(y, WL2) #Works (raw with margins)
wfm_combine(y, c("you", "your", "your\'re"))
wfm_combine(y, WL1)
wfm_combine(y, WL3)
## wfm_combine(y, WL4) #Error
wfm_combine(y, WL5)
wfm_combine(y, WL6)
wfm_combine(y, WL7)

worlis <- c("you", "it", "it\'s", "no", "not", "we")
y <- wdf(DATA$state, list(DATA$sex, DATA$adult), margins = TRUE)
z <- wfm_combine(y, worlis)

chisq.test(z)
chisq.test(wfm(y))

## Dendrogram
presdeb <- with(pres_debates2012, wfm(dialogue, list(person, time)))
library(sjPlot)
sjc.dend(t(presdeb), 2:4)

## Words correlated within turns of talk
## EXAMPLE 1
library(reports)
x <- factor(with(rajsplit, paste(act, pad(TOT(tot)), sep = "|")))
dat <- wfm(rajsPLIT$dialogue, x)

cor(t(dat)[, c("romeo", "juliet")])
cor(t(dat)[, c("romeo", "banished")])
cor(t(dat)[, c("romeo", "juliet", "hate", "love")])
qheat(cor(t(dat)[, c("romeo", "juliet", "hate", "love")]),
      diag.na = TRUE, values = TRUE, digits = 3, by.column = NULL)

dat2 <- wfm(DATA$state, id(DATA))
qheat(cor(t(dat2)), low = "yellow", high = "red",
      grid = "grey90", diag.na = TRUE, by.column = NULL)

## EXAMPLE 2
x2 <- factor(with(pres_debates2012, paste(time, pad(TOT(tot)), sep = "|")))
dat2 <- wfm(pres_debates2012$dialogue, x2)
wrds <- word_list(pres_debates2012$dialogue,
      stopwords = c("it\'s", "that\'s", Top200Words))
wrds2 <- tolower(sort(wrds$rfswl[[1]][, 1]))
qheat(word_cor(t(dat2)), word = wrds2, r = NULL,
      diag.na = TRUE, values = TRUE, digits = 3, by.column = NULL,
      high="red", low="yellow", grid=NULL)

## EXAMPLE 3
library(gridExtra); library(ggplot2); library(grid)
dat3 <- lapply(qcv(OBAMA, ROMNEY), function(x) {
    with(pres_debates2012, wfm(dialogue[person == x], x2[person == x]))
})
# Presidential debates by person
dat5 <- pres_debates2012
dat5 <- dat5[dat5$person %in% qcv(ROMNEY, OBAMA), ]

disp <- with(dat5, dispersion_plot(dialogue, wrds2, grouping.var = person,
total.color = NULL, rm.vars=time))

cors <- lapply(dat3, function(m) {
  word_cor(t(m), word = wrds2, r = NULL)
})

plots <- lapply(cors, function(x) {
  qheat(x, diag.na = TRUE, values = TRUE, digits = 3, plot = FALSE,
        by.column = NULL, high="red", low="yellow", grid=NULL)
})

plots <- lapply(1:2, function(i) {
  plots[[i]] + ggtitle(qcv(OBAMA, ROMNEY)[i]) +
  theme(axis.title.x = element_blank(),
        plot.margin = unit(rep(0, 4), "lines")
})

grid.arrange(disp, arrangeGrob(plots[[1]], plots[[2]], ncol=1), ncol=2)

## With 'word_cor'
worlis <- list(
  pronouns = c("you", "it", "it's", "we", "i'm", "i"),
  negative = qcv(no, dumb, distrust, not, stinks),
  literacy = qcv(computer, talking, telling)
)
y <- wddf(DATA$state, qdapTools:::id(DATA, prefix = TRUE))
z <- wfm_combine(y, worlis)

word_cor(t(z), word = names(worlis), r = NULL)

## Plotting method
plot(y, TRUE)
plot(z)

## Correspondence Analysis
library(ca)

dat <- pres_debates2012
dat <- dat[dat$person %in% qcv(ROMNEY, OBAMA), ]

speech <- stemmer(dat$dialogue)
mytable1 <- with(dat, wfm(speech, list(person, time), stopwords = Top25Words))

fit <- ca(mytable1)
summary(fit)
plot(fit)
plot3d.ca(fit, labels=1)
word_associate <- with(dat, wfm(speech, list(person, time), stopwords = Top200Words))

fit2 <- ca(mytable2)
sample(fit2)
plot(fit2)
plot3d.ca(fit2, labels=1)

## Weight a wfm
WFM <- with(DATA, wfm(state, list(sex, adult)))
plot(weight(WFM, "scaled"), TRUE)
weight(WFM, "prop")
weight(WFM, "max")
weight(WFM, "scaled")

## End(Not run)

---

word_associate

Find Associated Words

Description
Find words associated with a given word(s) or a phrase(s). Results can be output as a network graph and/or wordcloud.

Usage

word_associate(text.var, grouping.var = NULL, match.string, 
text.unit = "sentence", extra.terms = NULL, target.exclude = NULL, 
stopwords = NULL, network.plot = FALSE, wordcloud = FALSE, 
cloud.colors = c("black", "gray55"), title.color = "blue", 
nw.label.cex = 0.8, title.padj = -4.5, nw.label.colors = NULL, 
nw.layout = NULL, nw.edge.color = "gray90", 
nw.label.proportional = TRUE, nw.title.padj = NULL, 
nw.title.location = NULL, title.font = NULL, title.cex = NULL, 
nw.edge.curved = TRUE, cloud.legend = NULL, cloud.legend.cex = 0.8, 
cloud.legend.location = c(-0.03, 1.03), nw.legend = NULL, 
nw.legend.cex = 0.8, nw.legend.location = c(-1.54, 1.41), 
legend.override = FALSE, char2space = "~~", ...)

Arguments

text.var The text variable.

grouping.var The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.

match.string A list of vectors or vector of terms to associate in the text.
text.unit  The text unit (either "sentence" or "tot"). This argument determines what unit to find the match string words within. For example if "sentence" is chosen the function pulls all text for sentences the match string terms are found in.

extra.terms  Other terms to color beyond the match string.

target.exclude  A vector of words to exclude from the match.string.

stopwords  Words to exclude from the analysis.

network.plot  logical. If TRUE plots a network plot of the words.

wordcloud  logical. If TRUE plots a wordcloud plot of the words.

cloud.colors  A vector of colors equal to the length of match.string +1.

title.color  A character vector of length one corresponding to the color of the title.

nw.label.cex  The magnification to be used for network plot labels relative to the current setting of cex. Default is .8.

title.padj  Adjustment for the title. For strings parallel to the axes, padj = 0 means right or top alignment, and padj = 1 means left or bottom alignment.

nw.label.colors  A vector of colors equal to the length of match.string +1.

nw.layout  layout types supported by igraph. See layout.

nw.edge.color  A character vector of length one corresponding to the color of the plot edges.

nw.label.proportional  logical. If TRUE scales the network plots across grouping.var to allow plot to plot comparisons.

nw.title.padj  Adjustment for the network plot title. For strings parallel to the axes, padj = 0 means right or top alignment, and padj = 1 means left or bottom alignment.

nw.title.location  On which side of the network plot (1=bottom, 2=left, 3=top, 4=right).

title.font  The font family of the cloud title.

title.cex  Character expansion factor for the title. NULL and NA are equivalent to 1.0.

nw.edge.curved  logical. If TRUE edges will be curved rather than straight paths.

cloud.legend  A character vector of names corresponding to the number of vectors in match.string. Both nw.legend and cloud.legend can be set separately; or one may be set and by default the other will assume those legend labels. If the user does not desire this behavior use the legend.override argument.

cloud.legend.cex  Character expansion factor for the wordcloud legend. NULL and NA are equivalent to 1.0.

cloud.legend.location  The x and y co-ordinates to be used to position the wordcloud legend. The location may also be specified by setting x to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location.
nw.legend   A character vector of names corresponding to the number of vectors in match.string. Both nw.legend and cloud.legend can be set separately; or one may be set and by default the other will assume those legend labels. If the user does not desire this behavior use the legend.override argument.

nw.legend.cex Character expansion factor for the network plot legend. NULL and NA are equivalent to 1.0.

nw.legend.location The x and y co-ordinates to be used to position the network plot legend. The location may also be specified by setting x to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location.

legend.override By default if legend labels are supplied to either cloud.legend or nw.legend may be set and if the other remains NULL it will assume the supplied vector to the previous legend argument. If this behavior is not desired legend.override should be set to TRUE.

car2space Currently a road to nowhere. Eventually this will allow the retention of characters as is allowed in trans_cloud already.

... Other arguments supplied to trans_cloud.

Value

Returns a list:

word frequency matrices  Word frequency matrices for each grouping variable.
dialogue  A list of dataframes for each word list (each vector supplied to match.string) and a final dataframe of all combined text units that contain any match string.
match.terms  A list of vectors of word lists (each vector supplied to match.string).

Optionally, returns a word cloud and/or a network plot of the text unit containing the match.string terms.

See Also

trans_cloud, word_network_plot, wordcloud, graph.adjacency

Examples

## Not run:
ms <- c("I ", "you")
et <- c("it", "tell", "true")
out1 <- word_associate(DATA2$state, DATA2$person, match.string = ms,
wordcloud = TRUE, proportional = TRUE,
network.plot = TRUE, nw.label.proportional = TRUE, extra.terms = et,
cloud.legend = c("A", "B", "C"),
title.color = "blue", cloud.colors = c("red", "purple", "gray70"))
# Note: You don’t have to name the vectors in the lists but I do for clarity.

```r
ms <- list(
  list1 = c(" I ", " you", " not"),
  list2 = c(" wh")
)

et <- list(
  B = c(" the", " do", " tru"),
  C = c(" it", " already", " we")
)

out2 <- word_associate(DATA2$state, DATA2$person, match.string = ms,
  wordcloud = TRUE, proportional = TRUE,
  network.plot = TRUE, nw.label.proportional = TRUE, extra.terms = et,
  cloud.legend = c("A", "B", "C", "D"),
  title.color = "blue", cloud.colors = c("red", "blue", "purple", "gray70"))

out3 <- word_associate(DATA2$state, list(DATA2$day, DATA2$person), match.string = ms)

m <- list(
  A1 = c(" you", " in"), # list 1
  A2 = c(" wh")       # list 2
)

n <- list(
  B = c(" the", " on"),
  C = c(" it", " no")
)

out4 <- word_associate(DATA2$state, list(DATA2$day, DATA2$person),
  match.string = m)
out5 <- word_associate(raja.1$dialogue, list(raja.1$person),
  match.string = m)
out6 <- with(mrjasp1, word_associate(dialogue, list(fam.aff, sex),
  match.string = m))
names(out6)
lapply(out6$dialogue, htruncdf, n = 20, w = 20)

DATA2$state2 <- space_fill(DATA2$state, c("is fun", "too fun"))

ms <- list(
  list1 = c(" I ", " you", " is fun", " too fun"),
  list2 = c(" wh")
)

et <- list(
  B = c(" the", " on"),
  C = c(" it", " no")
)
```
word_cor

Find Correlated Words

Description

Find associated words within grouping variable(s).

Usage

word_cor(text.var, grouping.var = qdapTools::id(text.var), word, r = 0.7,
values = TRUE, method = "pearson", ...)

Arguments

text.var The text variable (or frequency matrix).

grouping.var The grouping variables. Default uses each row as a group. Also takes a single grouping variable or a list of 1 or more grouping variables. Unlike other qdap functions, this cannot be NULL.

word The word(s) vector to find associated words for.

r The correlation level find associated words for. If positive this is the minimum value, if negative this is the maximum value.

values logical. If TRUE returns the named correlates (names are the words). If FALSE only the associated words are returned.

method A character string indicating which correlation coefficient is to be computed ("pearson", "kendall", or "spearman").

... Other arguments passed to wfm.

Value

Returns a vector of associated words or correlation matrix if \( r = \) NULL.

Note

Note that if a word has no variability in it’s usage across grouping variable(s) the \( sd \) will result in 0, thus \( \text{cor} \) will will likely return a warning as in this example: \( \text{cor}(\text{rep}(3, 10), \text{rnorm}(10)) \).
References

The plotting method for the list output was inspired by Ben Marwick; see http://stackoverflow.com/a/19925445/1000343 for more.

See Also

word_proximity, findAssocs, word_associate, wfm.cor

Examples

```r
## Not run:
x <- factor(with(rajSPLIT, paste(act, pad(TOT(tot)), sep = "|")))  
word_cor(rajSPLIT$dialogue, x, "romeo", .45)  
word_cor(rajSPLIT$dialogue, x, "love", .5)

## Negative correlation
word_cor(rajSPLIT$dialogue, x, "you", -.1)
with(rajSPLIT, word_cor(dialogue, list(person, act), "hate"))

words <- c("hate", "i", "love", "ghost")
with(rajSPLIT, word_cor(dialogue, x, words, r = .5))
with(rajSPLIT, word_cor(dialogue, x, words, r = .4))

## Set `r = NULL` to get matrix between words
with(rajSPLIT, word_cor(dialogue, x, words, r = NULL))

## Plotting
library(tm)
data("crude")
oil_cor1 <- apply_as_df(crude, word_cor, word = "oil", r = .7)
plot(oil_cor1)

oil_cor2 <- apply_as_df(crude, word_cor, word = qcv(texas, oil, money), r = .7)
plot(oil_cor2)
plot(oil_cor2, ncol=2)

oil_cor3 <- apply_as_df(crude, word_cor, word = qcv(texas, oil, money), r=NULL)
plot(oil_cor3)

## Run on multiple times/person/nested
## Split and apply to data sets
## Suggested use of stemming
DATA3 <- split(DATA2, DATA2$person)

## Find correlations between words per turn of talk by person
## Throws multiple warning because small data set
library(qdapTools)
lapply(DATA3, function(x) {
   word_cor(x[, "state"], qdapTools::id(x), qcv(computer, i, no, good), r = NULL)
})

## Find words correlated per turn of talk by person
```
## Throws multiple warning because small data set
```
lapply(DATA3, function(x) {
  word.cor(x[, "state"], qdapTools::id(x), qcv(computer, i, no, good))
})
```

## A real example
```
dat <- pres.debates2012
.dat$TOT <- factor(with(dat, paste(time, pad(TOT(tot)), sep = "|")))
.dat <- dat[dat$person %in% qcv(Obama, Romney), ]
.dat$person <- factor(dat$person)
.dat.split <- with(dat, split(dat, list(person, time)))
wrds <- qcv(america, debt, dollar, people, tax, health)
lapply(.dat.split, function(x) {
  word.cor(x[, "dialogue"], x[, "TOT"], wrds, r=NULL)
})
```

## Supply a matrix (make sure to use `t` on a `wfm` matrix)
```
worlis <- list(
  pronouns = c("you", "it", "it's", "we", "i'm", "i"),
  negative = qcv(no, dumb, distrust, not, stinks),
  literacy = qcv(computer, talking, telling)
)
y <- wdf(.dat$state, qdapTools::id(.dat), prefix = TRUE)
z <- wfm_combine(y, worlis)
out <- word.cor(t(z), word = c(names(worlis), "else.words"), r = NULL)
out
```

## Additional plotting/viewing
```
require(tm)
data("crude")
out1 <- word.cor(t(as.wfm(crude)), word = "oil", r=.7)
vect2df(out1[[1]], "word", "cor")
plot(out1)
qheat(vect2df(out1[[1]], "word", "cor"), values=TRUE, high="red",
  digits=2, order.by="cor", plot=FALSE) + coord_flip()
```
```
out2 <- word.cor(t(as.wfm(crude)), word = c("oil", "country"), r=.7)
plot(out2)
```

## End (Not run)
**word_count**

**Description**

word_count - Transcript apply word counts.

character_count - Transcript apply character counts.

character_table - Computes a table of character counts by grouping variable(s).

**Usage**

```r
word_count(text.var, byrow = TRUE, missing = NA, digit.remove = TRUE,
            names = FALSE)
```

```r
wc(text.var, byrow = TRUE, missing = NA, digit.remove = TRUE,
   names = FALSE)
```

```r
character_count(text.var, byrow = TRUE, missing = NA,
                apostrophe.remove = TRUE, digit.remove = TRUE, count.space = FALSE)
```

```r
character_table(text.var, grouping.var = NULL, percent = TRUE,
                prop.by.row = TRUE, zero.replace = 0, digits = 2, ...)
```

```r
char_table(text.var, grouping.var = NULL, percent = TRUE,
           prop.by.row = TRUE, zero.replace = 0, digits = 2, ...)
```

**Arguments**

- `text.var` The text variable
- `byrow` logical. If TRUE counts by row, if FALSE counts all words.
- `missing` Value to insert for missing values (empty cells).
- `digit.remove` logical. If TRUE removes digits before counting words.
- `names` logical. If TRUE the sentences are given as the names of the counts.
- `apostrophe.remove` logical. If TRUE apostrophes will be counted in the character count.
- `count.space` logical. If TRUE spaces are counted as characters.
- `grouping.var` The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `percent` logical. If TRUE output given as percent. If FALSE the output is proportion.
- `prop.by.row` logical. If TRUE applies proportional to the row. If FALSE applies by column.
- `zero.replace` Value to replace 0 values with.
- `digits` Integer; number of decimal places to round when printing.
- `...` Other arguments passed to `prop`.
Value

word_count - returns a word count by row or total.
character_count - returns a character count by row or total.
character_table - returns a list: dataframe of character counts by grouping variable.

raw Dataframe of the frequency of characters by grouping variable.
prop Dataframe of the proportion of characters by grouping variable.
rnp Dataframe of the frequency and proportions of characters by grouping variable.
percent The value of percent used for plotting purposes.
zero.replace The value of zero.replace used for plotting purposes.

Note

wc is a convenient short hand for word_count.

See Also

syllable_count, prop, colcomb2class

Examples

## Not run:
## WORD COUNT
word_count(DATA$state)
wc(DATA$state)
word_count(DATA$state, names = TRUE)
word_count(DATA$state, byrow=FALSE, names = TRUE)
sum(word_count(DATA$state))

sapply(split(raj$dialogue, raj$person), wc, FALSE) %>%
  sort(decreasing=TRUE) %>%
  list2df("wordcount", "person") %>%
  `\[`(, 2:1)

## PLOT WORD COUNTS
raj2 <- raj
raj2$scaled <- unlist(tapply(wc(raj$dialogue), raj2$act, scale))
raj2$scaled2 <- unlist(tapply(wc(raj$dialogue), raj2$act, scale, scale = FALSE))
raj2$ID <- factor(unlist(tapply(raj2$act, raj2$act, seq_along)))

ggplot(raj2, aes(x = ID, y = scaled, fill =person)) +
  geom_bar(stat="identity") +
  facet_grid(act~.) +
  ylab("Scaled") + xlab("Turn of Talk") +
  guides(fill = guide_legend(nrow = 5, byrow = TRUE)) +
  theme(legend.position="bottom") +
  ggtitle("Scaled and Centered")
ggplot(raj, aes(x = ID, y = scaled, fill = person)) +
  geom_bar(stat="identity") +
  facet_grid(act~.) +
  ylab("Scaled") + xlab("Turn of Talk") +
  guides(fill = guide_legend(nrow = 5, byrow = TRUE)) +
  theme(legend.position="bottom") +
  ggtitle("Mean Difference")

raj$wc <- wc(raj$dialogue)
raj$cum.wc <- unlist(with(raj, tapply(wc, act, cumsum)))
raj$turn <- unlist(with(raj, tapply(act, act, seq_along)))

ggplot(raj, aes(y=cum.wc, x=turn)) +
  geom_step(direction = "hv") +
  facet_wrap(~act)

## CHARACTER COUNTS

character_count(DATA$state)
character_count(DATA$state, byrow=FALSE)
sum(character_count(DATA$state))

## CHARACTER TABLE

x <- character_table(DATA$state, DATA$person)
plot(x)
plot(x, label = TRUE)
plot(x, label = TRUE, text.color = "red")
plot(x, label = TRUE, lab.digits = 1, zero.replace = "PP7")

scores(x)
counts(x)
proportions(x)

plot(scores(x))
plot(counts(x))
plot(proportions(x))

## combine columns

colcomb2class(x, list(vowels = c("a", "e", "i", "o", "u")))

## char_table(DATA$state, DATA$person)
## char_table(DATA$state, DATA$person, percent = TRUE)
## character_table(DATA$state, list(DATA$sex, DATA$adult))

library(ggplot2); library(reshape2)
dat <- character_table(DATA$state, list(DATA$sex, DATA$adult))
dat2 <- colsplit2df(melt(counts(dat)), keep.orig = TRUE)
head(dat2, 15)

ggplot(data = dat2, aes(y = variable, x = value, colour=sex)) +
  facet_grid(adult~.) +
  geom_line(size=1, aes(group =variable), colour = "black") +
  geom_point()
```r
ggplot(data = dat2, aes(x = variable, y = value)) +
  geom_bar(aes(fill = variable), stat = "identity") +
  facet_grid(sex ~ adult, margins = TRUE) +
  theme(legend.position="none")

## End(Not run)
```

---

**word_diff_list**  
_Differences In Word Use Between Groups_

**Description**

Look at the differences in word uses between grouping variable(s). Look at all possible "a" vs. "b" combinations or "a" vs. all others.

**Usage**

```r
word_diff_list(text.var, grouping.var, vs.all = FALSE, vs.all.cut = 1,
               stopwords = NULL, alphabetical = FALSE, digits = 2)
```

**Arguments**

- `text.var`: The text variable.
- `grouping.var`: The grouping variables. Default `NULL` generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- `vs.all`: logical. If `TRUE` looks at each grouping variable against all others ("a" vs. all comparison). If `FALSE` looks at each "a" vs. "b", comparison (e.g., for groups "a", "b", and "c"; "a" vs. "b", "a" vs. "c" and "b" vs. "c" will be considered).
- `vs.all.cut`: Controls the number of other groups that may share a word (default is 1).
- `stopwords`: A vector of stop words to remove.
- `alphabetical`: logical. If `TRUE` orders the word lists alphabetized by word. If `FALSE` order first by frequency and then by word.
- `digits`: the number of digits to be displayed in the proportion column (default is 3).

**Value**

An list of word data frames comparing grouping variables word use against one another. Each dataframe contains three columns:

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>The words unique to that group</td>
</tr>
<tr>
<td>freq</td>
<td>The number of times that group used that word</td>
</tr>
<tr>
<td>prop</td>
<td>The proportion of that group’s overall word use dedicated to that particular word</td>
</tr>
</tbody>
</table>
Examples

```r
## Not run:
out1 <- with(DATA, word_diff_list(text.var = state,
     grouping.var = list(sex, adult)))
lapply(unlist(out1, recursive = FALSE), head, n=3)

out2 <- with(DATA, word_diff_list(state, person))
lapply(unlist(out2, recursive = FALSE), head, n=3)

out3 <- with(DATA, word_diff_list(state, grouping.var = list(sex, adult),
     vs.all=TRUE, vs.all.cut=2))

out4 <- with(mraja1, word_diff_list(text.var = dialogue,
     grouping.var = list(mraja1$sex, mraja1$fam.aff)))

out5 <- word_diff_list(mraja1$dialogue, mraja1$person)

out6 <- word_diff_list(mraja1$dialogue, mraja1$fam.aff, stopwords = Top25Words)

out7 <- word_diff_list(mraja1$dialogue, mraja1$fam.aff, vs.all=TRUE, vs.all.cut=2)
lapply(out7, head, n=3)

## End(Not run)
```

---

word_length  

### Description

Transcript apply word length counts.

### Usage

```r
word_length(text.var, grouping.var = NULL, percent = TRUE,
    zero.replace = 0, digits = 2, ...)
```

### Arguments

- **text.var**  
The text variable.

- **grouping.var**  
The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.

- **percent**  
Logical. If TRUE output given as percent. If FALSE the output is proportion.

- **zero.replace**  
Value to replace 0 values with.

- **digits**  
Integer; number of decimal places to round when printing.

- **...**  
Other arguments passed to `bag_o_words`. 

---
Value

Returns a list of:

- `count`: Dataframe of word length counts by grouping variable(s).
- `prop`: Dataframe of the proportions of word length counts by grouping variable.
- `rnp`: Dataframe of the frequency and proportions of word length counts by grouping variable.
- `percent`: The value of percent used for plotting purposes.
- `zero.replace`: The value of zero.replace used for plotting purposes.

Examples

```r
## Not run:
(x <- with(DATA, word_length(state, person)))
plot(x)
scores(x)
proportions(x)
counts(x)
plot(scores(x))
plot(proportions(x))
plot(counts(x))

(x2 <- word_length(DATA["state"], apostrophe.remove=TRUE))

## Example Visualizations with Presidential Debate Data
library(tidyverse)
(x_long2 <- proportions(x2) %>%
gather("Letter_Length", "Proportion", -c(1:2)))
ggplot(x_long2, aes(x = Letter_Length, y = Proportion, color=person, group=person)) +
geom_line(size=.8)

(x3 <- with(pres_debates2012, word_length(dialogue, person)))
(x_long2 <- proportions(x3) %>%
gather("Letter_Length", "Proportion", -c(1:2)))
ggplot(x_long2, aes(x = Letter_Length, weight = Proportion, fill=person, group=person)) +
geom_bar()
ggplot(x_long2, aes(x = Letter_Length, weight = Proportion, fill=person)) +
geom_bar() +
facet_wrap(~person, ncol=1)
ggplot(x_long2, aes(x = Letter_Length, weight = Proportion, fill=person)) +
geom_bar() +
coord_flip() +
facet_wrap(~person, ncol=1)
ggplot(x_long2, aes(x = person, weight = Proportion)) +
geom_bar(fill="grey40") +
coord_flip() +
```

```
word_list

```r
facet_grid(Letter_Length ~ .)

## End(Not run)
```

### word_list  

**Raw Word Lists/Frequency Counts**

#### Description

Transcript Apply Raw Word Lists and Frequency Counts by grouping variable(s).

#### Usage

```r
word_list(text.var, grouping.var = NULL, stopwords = NULL, alphabetical = FALSE, cut.n = 20, cap = TRUE, cap.list = NULL, cap.I = TRUE, rm.bracket = TRUE, char.keep = NULL, apostrophe.remove = FALSE, ...)
```

#### Arguments

- `text.var`  
The text variable.

- `grouping.var`  
The grouping variables. Default `NULL` generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.

- `stopwords`  
A vector of stop words to remove.

- `alphabetical`  
If `true` the output of frequency lists is ordered alphabetically. If `false` the list is ordered by frequency rank.

- `cut.n`  
Cut off point for reduced frequency stop word list (rfswl).

- `cap`  
Logical. If `true` capitalizes words from the cap.list.

- `cap.list`  
Vector of words to capitalize.

- `cap.I`  
Logical. If `true` capitalizes words containing the personal pronoun I.

- `rm.bracket`  
Logical. If `true` all brackets and bracketed text are removed from analysis.

- `char.keep`  
A character vector of symbols (i.e., punctuation) that word_list should keep. The default is to remove every symbol except apostrophes.

- `apostrophe.remove`  
Logical. If `true` removes apostrophes from the output.

- `...`  
Other arguments passed to `strip`.

#### Value

An object of class "word_list" is a list of lists of vectors or dataframes containing the following components:

- `cwl`  
  complete word list; raw words

- `swl`  
  stop word list; same as rwl with stop words removed

- `fwl`  
  frequency word list; a data frame of words and corresponding frequency counts

- `fswl`  
  frequency stopword word list; same as fwl but with stop words removed

- `rfswl`  
  reduced frequency stopword word list; same as fswl but truncated to n rows
Examples

```r
## Not run:
word_list(raj.act.1$dialogue)

out1 <- with(raj, word_list(text.var = dialogue, 
    grouping.var = list(person, act)))
names(out1)
lapply(out1$cwl, "[", 1:5)

with(DATA, word_list(state, person))
with(DATA, word_list(state, person, stopwords = Top25Words))
with(DATA, word_list(state, person, cap = FALSE, cap.list=c("do", "we")))

## End(Not run)
```

Description

A network plot of words. Shows the interconnected and supporting use of words between textual units containing key terms.

Usage

```r
word_network_plot(text.var, grouping.var = 1:length(text.var),
    target.words = NULL, stopwords = qdapDictionaries::Top100Words,
    label.cex = 0.8, label.size = 0.5, edge.curved = TRUE,
    vertex.shape = "circle", edge.color = "gray70", label.colors = "black",
    layout = NULL, title.name = NULL, title.padj = -4.5,
    title.location = 3, title.font = NULL, title.cex = 0.8,
    log.labels = FALSE, title.color = "black", legend = NULL,
    legend.cex = 0.8, legend.location = c(-1.54, 1.41), plot = TRUE,
    char.space = "~~", ...
```

Arguments

- **text.var**: The text variable.
- **grouping.var**: The grouping variables. Default uses the sequence along the length of text variable (this may be the connection of sentences or turn of talk as the textual unit). Also takes a single grouping variable or a list of 1 or more grouping variables.
- **target.words**: A named list of vectors of words whose length corresponds to `label.colors` (+1 length in cloud colors for non-matched terms).
- **stopwords**: Words to exclude from the analysis (default is `Top100Words`).
- **label.cex**: The magnification to be used for network plot labels relative to the current setting of `cex`. Default is .8.
label.size
An optional sizing constant to add to labels if log.labels is TRUE.

edge.curved
logical. If TRUE edges will be curved rather than straight paths.

vertex.shape
The shape of the vertices (see igraph.vertex.shapes for more).

edge.color
A character vector of length one corresponding to the color of the plot edges.

label.colors
A character vector of length one corresponding to the color of the labels.

layout
Layout types supported by igraph. See layout.

title.name
The title of the plot.

title.padj
Adjustment for the network plot title. For strings parallel to the axes, padj = 0 means right or top alignment, and padj = 1 means left or bottom alignment.

title.location
On which side of the network plot (1=bottom, 2=left, 3=top, 4=right).

title.font
The font family of the cloud title.

title.cex
Character expansion factor for the title. NULL and NA are equivalent to 1.0.

log.labels
logical. If TRUE uses a proportional log label for more readable labels. The formula is: \( \log(\text{sums})/\max(\log(\text{sums})) \). label.size adds more control over the label sizes.

title.color
A character vector of length one corresponding to the color of the title.

legend
A character vector of names corresponding to the number of vectors in match.string.

legend.cex
Character expansion factor for the network plot legend. NULL and NA are equivalent to 1.0.

legend.location
The x and y co-ordinates to be used to position the network plot legend. The location may also be specified by setting x to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location.

plot
logical. If TRUE plots a network plot of the words.

char2space
A vector of characters to be turned into spaces. If char.keep is NULL, char2space will activate this argument.

... Other arguments passed to strip.

Note

Words can be kept as one by inserting a double tilde ("~~"), or other character strings passed to char2space, as a single word/entry. This is useful for keeping proper names as a single unit.

See Also

word_network_plot, graph.adjacency
Examples

```r
## Not run:
word_network_plot(text.var=DATA$state)
word_network_plot(text.var=DATA$state, stopwords=NULL)
word_network_plot(text.var=DATA$state, DATA$person)
word_network_plot(text.var=DATA$state, DATA$person, stopwords=NULL)
word_network_plot(text.var=DATA$state, grouping.var=list(DATA$sex, DATA$adult))
word_network_plot(text.var=DATA$state, grouping.var=DATA$person, 
                 title.name = "TITLE", log.labels=TRUE)
word_network_plot(text.var=raj.act.1$dialogue, grouping.var=raj.act.1$person, 
                 stopwords = Top200Words)

#insert double tilde ("--") to keep dual words (e.g., first last name)
alts <- c(" fun", "I ")
state2 <- mgsub(alts, gsub("\s", "--", alts), DATA$state)
word_network_plot(text.var=state2, grouping.var=DATA$person)

## Invisibly returns the igraph model
x <- word_network_plot(text.var=DATA$state, DATA$person)
str(x)
library(igraph)
pplot(x, vertex.size=0, vertex.color="white", edge.curved = TRUE)

x2 <- word_network_plot(text.var=DATA$state, grouping.var=DATA$person, 
                        title.name = "TITLE", log.labels = TRUE, label.size = 1.2)
l <- layout.drl(x2, options=list(simmer.attraction=0))
pplot(x2, vertex.size=0, layout = l)

## End(Not run)
```

word_position

<table>
<thead>
<tr>
<th>Word Position</th>
</tr>
</thead>
</table>

Description

Find counts of the positioning of words within a sentence.

Usage

```r
word_position(text.var, match.terms, digits = 2, percent = TRUE, 
              zero.replace = 0, ...)
```

Arguments

text.var The text variable.

match.terms A character vector of quoted terms to find the positions of.

digits Integer; number of decimal places to round when printing.
percent logical. If TRUE output given as percent. If FALSE the output is proportion.
zero.replace Value to replace 0 values with.

Value

Returns a list, of class "word_position", of data frames and information regarding word positions:

raw raw word position counts in long format (may be more useful for plotting)
count integer word position counts
prop proportional word position counts; proportional to each total word uses
rnp a character combination data frame of count and proportional
zero_replace value to replace zeros with; mostly internal use
percent The value of percent used for plotting purposes.
digits integer value of number of digits to display; mostly internal use

Note

Default printing is a heatmap plot.

Examples

## Not run:
position <- with(DATA, word_position(sent_detect(state), Top25Words))
position
tview(position)
plot(position)
scores(position)
preprocessed(position)
counts(position)
proportions(position)
plot(proportions(position))

stopwords <- unique(c(contractions[[1]], Top200Words))
topwords <- freq_terms(pres_debates2012["dialogue"], top = 40,
at.least = 4, stopwords = stopwords)[[1]]
word_position(pres_debates2012["dialogue"], topwords)
plot(word_position(pres_debates2012["dialogue"], topwords), FALSE)
plot(word_position(pres_debates2012["dialogue"], topwords), TRUE, scale=FALSE)

wordlist <- c("tax", "health", "rich", "america", "truth", "money", "cost",
"governor", "president", "we", "job", "i", "you", "because",
"our", "years")

word_position(pres_debates2012["dialogue"], wordlist)

## BY VARIABLES
library(gridExtra)
pres_deb_by_time <- with(pres_debates2012, split(dialogue, time))
out1 <- lapply(pres_deb_by_time, word_position, wordlist)
do.call("grid.arrange", c(lapply(out1, plot), ncol=1))

pres_deb_by_person <- with(pres_debates2012, split(dialogue, person))
out2 <- lapply(pres_deb_by_person, word_position, wordlist)
plots <- lapply(names(out2), function(x) plot(out2[[x]], scale=FALSE) +
ggtitle(x))
do.call("grid.arrange", c(plots, ncol=2))

## As a histogram
## theme taken from: http://jonlefcheck.net/2013/03/11/black-theme-for-ggplot2-2/
theme_black <- function(base_size=12, base_family="") {
  theme_grey(base_size=base_size, base_family=base_family) %+replace%
  theme(    
    # Specify axis options
    axis.line=element_blank(),
    axis.text.x=element_text(size=base_size*0.8, color="grey55",
                              lineheight=0.9, vjust=1),
    axis.text.y=element_text(size=base_size*0.8, color="grey55",
                              lineheight=0.9, hjust=1),
    axis.ticks=element_line(color="grey55", size = 0.2),
    axis.title.x=element_text(size=base_size, color="grey55", angle=90,
                              vjust=0.5),
    axis.ticks.length=unit(0.3, "lines"),
    axis.ticks.margin=unit(0.5, "lines"),
    # Specify legend options
    legend.background=element_rect(color=NA, fill="black"),
    legend.key=element_rect(color="grey55", fill="black"),
    legend.key.size=unit(1.2, "lines"),
    legend.key.height=NULL,
    legend.key.width=NaN,
    legend.text=element_text(size=base_size*0.8, color="grey55"),
    legend.title=element_text(size=base_size*0.8, face="bold", hjust=0,
                              color="grey55"),
    legend.position="right",
    legend.text.align=NULL,
    legend.title.align=NULL,
    legend.direction="vertical",
    legend.box=NULL,
    # Specify panel options
    panel.background=element_rect(fill="black", color = NA),
    panel.border=element_rect(fill=NA, color="grey55"),
    panel.grid.major=element_blank(),
    panel.grid.minor=element_blank(),
    panel.spacing=unit(0.25, "lines"),
    # Specify facetting options
    strip.background=element_rect(fill="grey30", color="grey10"),
    strip.text.x=element_text(size=base_size*0.8, color="grey55"),
    strip.text.y=element_text(size=base_size*0.8, color="grey55",
                              angle=-90),
    # Specify plot options
    plot.background=element_rect(color="black", fill="black"),
  )
}
word_proximity

Description

word_proximity - Generate proximity measures to ascertain a mean distance measure between word uses.

weight - weight Method for word_proximity.

Usage

word_proximity(text.var, terms, grouping.var = NULL, parallel = TRUE, cores = parallel::detectCores()/2)

## S3 method for class 'word_proximity'
weight(x, type = "scale", ...)
Arguments

text.var The text variable.
terms A vector of quoted terms.
grouping.var The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
parallel logical. If TRUE attempts to run the function on multiple cores. Note that this may not mean a speed boost if you have one core or if the data set is smaller as the cluster takes time to create.
cores The number of cores to use if parallel = TRUE. Default is half the number of available cores.
x An object to be weighted.
type A weighting type of: c("scale_log", "scale", "rev_scale", "rev_scale_log", "log", "sqrt", "scale_sqrt", "rev_sqrt", "rev_scale_sqrt"). The weight type section name (i.e. A_B_C where A, B, and C are sections) determines what action will occur. log will use log, sqrt will use sqrt, scale will standardize the values. rev will multiply by -1 to give the inverse sign. This enables a comparison similar to correlations rather than distance.

Details

Note that row names are the first word and column names are the second comparison word. The values for Word A compared to Word B will not be the same as Word B compared to Word A. This is because, unlike a true distance measure, word_proximity's matrix is asymmetrical. word_proximity computes the distance by taking each sentence position for Word A and comparing it to the nearest sentence location for Word B.

Value

Returns a list of matrices of proximity measures in the unit of average sentences between words (defaults to scaled).

Note

The match.terms is character sensitive. Spacing is an important way to grab specific words and requires careful thought. Using "read" will find the words "bread", "read" "reading", and "ready". If you want to search for just the word "read" you'd supply a vector of c(" read", " reads", " reading", " reader").

See Also

word_proximity
### Examples

```r
## Not run:
wrds <- word_list(pres_debates2012$dialogue,
  stopwords = c("it's", "that's", Top200Words))
wrds2 <- tolower(sort(wrds$rfswl[[1]][, 1]))

(x <- with(pres_debates2012, word_proximity(dialogue, wrds2)))
plot(x)
plot(weight(x))
plot(weight(x, "rev_scale_log"))

(x2 <- with(pres_debates2012, word_proximity(dialogue, wrds2, person)))

## The spaces around 'terms' are important
(x3 <- with(DATA, word_proximity(state, spaste(qcv(the, i)))))
(x4 <- with(DATA, word_proximity(state, qcv(the, i))))

## End(Not run)
```

---

**word_stats**  
*Descriptive Word Statistics*

**Description**

Transcript apply descriptive word statistics.

**Usage**

```r
word_stats(text.var, grouping.var = NULL, tot = NULL, parallel = FALSE,
  rm.incomplete = FALSE, digit.remove = FALSE, apostrophe.remove = FALSE,
  digits = 3, ...)
```

**Arguments**

- **text.var**  
The text variable or a "word_stats" object (i.e., the output of a word_stats function).
- **grouping.var**  
The grouping variables. Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
- **tot**  
Optional turns of talk variable that yields turn of talk measures.
- **parallel**  
logical. If TRUE attempts to run the function on multiple cores. Note that this may not mean a speed boost if you have one core or if the data set is smaller as the cluster takes time to create (parallel is slower until approximately 10,000 rows). To reduce run time pass a "word_stats" object to the word_stats function.
- **rm.incomplete**  
logical. If TRUE incomplete statements are removed from calculations in the output.
- **digit.remove**  
logical. If TRUE removes digits from calculating the output.
apostrophe.remove
logical. If TRUE removes apostrophes from calculating the output.
digits
Integer; number of decimal places to round when printing.
... Any other arguments passed to end_inc.

Details
Note that a sentence is classified with only one endmark. An imperative sentence is classified only as imperative (not as a state, quest, or exclm as well). If a sentence is both imperative and incomplete the sentence will be counted as incomplete rather than imperative. labeled as both imperative.

Value
Returns a list of three descriptive word statistics:

- ts A data frame of descriptive word statistics by row
- gts A data frame of word/sentence statistics per grouping variable:
  - n.tot - number of turns of talk
  - n.sent - number of sentences
  - n.words - number of words
  - n.char - number of characters
  - n.syl - number of syllables
  - n.poly - number of polysyllables
  - spot - syllables per turn of talk
  - wptot - words per turn of talk
  - wps - words per sentence
  - cps - characters per sentence
  - sps - syllables per sentence
  - psps - poly-syllables per sentence
  - cpw - characters per word
  - spw - syllables per word
  - n.state - number of statements
  - n.quest - number of questions
  - n.exclm - number of exclamations
  - n.incom - number of incomplete statements
  - p.state - proportion of statements
  - p.quest - proportion of questions
  - p.exclm - proportion of exclamations
  - p.incom - proportion of incomplete statements
  - n.hapax - number of hapax legomenon
  - n.dis - number of dis legomenon
  - grow.rate - proportion of hapax legomenon to words
  - prop.dis - proportion of dis legomenon to words

- mpun An account of sentences with an improper/missing end mark
word_stats

word.elem  | A data frame with word element columns from gts
sent.elem  | A data frame with sentence element columns from gts
omit      | Counter of omitted sentences for internal use (only included if some rows contained missing values)
percent   | The value of percent used for plotting purposes.
zero.replace | The value of zero.replace used for plotting purposes.
digits    | integer value of number of digits to display; mostly internal use

Warning

It is assumed the user has run sentSplit on their data, otherwise some counts may not be accurate.

See Also

end_inc

Examples

```
## Not run:
word.stats(mrajaspl$dialogue, mrajaspl$person)

(desc_wrds <- with(mrajaspl, word_stats(dialogue, person, tot = tot)))

## Recycle for speed boost
with(mrajaspl, word_stats(desc_wrds, person, tot = tot))

scores(desc_wrds)
counts(desc_wrds)
htruncdn(counts(desc_wrds), 15, 6)
plot(scores(desc_wrds))
plot(counts(desc_wrds))

names(desc_wrds)
htruncdn(desc_wrds$ts, 15, 5)
htruncdn(desc_wrds$gts, 15, 6)
desc_wrds$mpun
desc_wrds$word.elem
desc_wrds$sent.elem
plot(desc_wrds)
plot(desc_wrds, label=TRUE, lab.digits = 1)

## Correlation Visualization
qheat(cor(scores(desc_wrds)[, -1]), diag.na = TRUE, by.column =NULL,
     low = "yellow", high = "red", grid = FALSE)

## Parallel (possible speed boost)
with(mrajaspl, word_stats(dialogue, list.sex, died, fam.aff))
with(mrajaspl, word_stats(dialogue, list.sex, died, fam.aff),
     parallel = TRUE))
```
## Recycle for speed boost
word_stats(desc_wrd, mrajaispl$sex)

## End (Not run)

### qdap Chaining

**Description**

```r
## - Chain `qdap dfs` to `qdap` functions with a `text var` argument. Saves typing of an explicit `text var` argument and supplying a `data frame`.  
## - The `magrittr` "then" chain operator imported by `dplyr`. Imported for convenience. See https://github.com/smbache/magrittr for details.
```

**Usage**

```r
qdap_df.object `%%` qdap.fun

lhs `%%` rhs
```

**Arguments**

- `qdap_df.object`: A `data frame` of the class "`qdap_df".  
- `qdap.fun`: A `qdap` function with a `text var` argument.  
- `lhs`: The value to be piped.  
- `rhs`: A function or expression.

**References**

- Inspired by `magrittr`'s `%%` functionality.

**See Also**

- `%%`, `qdap_df`

**Examples**

```r
## Not run:
dat <- qdap_df(DATA, state)  
dat `%%` trans_cloud(grouping.var=person)  
dat `%%` trans_cloud(grouping.var=person, text.var=stemmer(DATA$state))  
dat `%%` termco(grouping.var=person, match.list=list("fun", "computer"))

## Various examples with qdap functions (sentSplit gives class "qdap_df")  
dat <- sentSplit(DATA, "state")  
dat `%%` trans_cloud(grouping.var=person)  
dat `%%` termco(person, match.list=list("fun", "computer"))
```
dat %>% trans_venn(person)
dat %>% polarity(person)
dat %>% formality(person)
dat %>% automated_readability_index(person)
dat %>% Dissimilarity(person)
dat %>% gradient_cloud(sex)
dat %>% dispersion_plot(c("fun", "computer")))
dat %>% discourse_map(list(sex, adult))
dat %>% gantt_plot(person)
dat %>% word_list(adult)
dat %>% end_mark_by(person)
dat %>% end_mark()
dat %>% word_stats(person)
dat %>% wfm(person)
dat %>% word_cor(person, "i")
dat %>% sentCombine(person)
dat %>% question_type(person)
dat %>% word_network_plot()
dat %>% character_count()
dat %>% char_table(person)
dat %>% phrase_net(RL NQI)
dat %>% boolean_search("it!!")
dat %>% trans_context(person, which(end_mark(DATA.SPLIT[, "state"])) == "?")
dat %>% mgsub(c("it's", "I'm"), c("it is", "I am"))

## combine with magrittr/dplyr chaining
dat %>% wfm(person) %>>% plot()
dat %>% polarity(person) %>>% scores()
dat %>% polarity(person) %>>% counts()
dat %>% polarity(person) %>>% scores()
dat %>% polarity(person) %>>% scores %>>% plot()
dat %>% polarity(person) %>>% scores %>>% plot

## Change text column in `qdap_df` (Example 1)
dat2 <- sentSplit(DATA, "state", stem.col = TRUE)
class(dat2)
dat2 %>% trans_cloud()
Text(dat2)
## change the `text.var` column
Text(dat2) <- "stem.text"
dat2 %>% trans_cloud()

## Change text column in `qdap_df` (Example 2)
(dat2$fake_dat <- paste(emoticon[1:11,2], dat2$state))
Text(dat2) <- "fake_dat"
(m <- dat2 %>% sub_holder(emoticon[,2]))
m%$unhold(strip(m%$output))

## End(Not run)
Index

*Topic Automated
automated_readability_index, 32

*Topic Boolean-matrix
adjacency_matrix, 12

*Topic Coleman
automated_readability_index, 32

*Topic Flesch-Kincaid
automated_readability_index, 32

*Topic Fry
automated_readability_index, 32

*Topic Gantt
gantt, 131
gantt_plot, 133
gantt_rep, 135
gantt_wrap, 137

*Topic Index
automated_readability_index, 32

*Topic Kullback-Leibler
kullback_leibler, 149

*Topic Liau
automated_readability_index, 32

*Topic Linsear
automated_readability_index, 32

*Topic Readability
automated_readability_index, 32

*Topic SMOG
automated_readability_index, 32

*Topic Text
qdap_df, 299

*Topic Write
automated_readability_index, 32

*Topic Zipf
rank_freq_mplot, 319

*Topic abbreviation
replace_abbreviation, 325

*Topic adjacency-matrix
adjacency_matrix, 12

*Topic association
cm_distance, 71

word_cor, 394

*Topic bag-of-words
bag_o_words, 36

*Topic bracket
bracketX, 39

*Topic bracket-remove
bracketX, 39

*Topic chaining
%%%, 414

*Topic chain
%%%, 414

*Topic character-count
word_count, 396

*Topic character
clean, 53
qcv, 298

*Topic check
check_text, 49

*Topic chunks
chunker, 51

clean
scrubber, 344

*Topic co-occurrence
cm_code.blank, 55
cm_code.combine, 57
cm_code.exclude, 59
cm_code.overlap, 60
cm_combine.dummy, 64

*Topic coded
cm_long2dummy, 75

*Topic codes
cm_distance, 71

*Topic coding

cm_df2long, 69
cm_range2long, 78
cm_time2long, 80

*Topic coding

cm_df.fill, 65
cm_df.temp, 67
cm_range.temp, 77
cm_time.temp, 79
*Topic column-split
colSplit, 83
colSplit2df, 84
*Topic combine,
combine, 297
*Topic comma
comma_spacer, 86
*Topic content
lexical_classification, 151
*Topic contraction
replace_contraction, 326
*Topic correlation,
word_cor, 394
*Topic curly-braces
bracketx, 39
*Topic datasets
DATA, 100
DATA.SPLIT, 101
DATA2, 101
env.syl, 119
hamlet, 141
mrajal, 160
mrajalspl, 161
pres_debate_raw2012, 242
pres_debates2012, 242
raj, 312
raj.act.1, 313
raj.act.1POS, 313
raj.act.2, 314
raj.act.3, 315
raj.act.4, 315
raj.act.5, 316
raj-demographics, 316
rajPOS, 317
rajSPLIT, 317
raw.time.span, 321
sample.time.span, 332
*Topic data
gdap_df, 299
*Topic demographic
key_merge, 148
*Topic descriptive
word_stats, 411
*Topic dispersion
dispersion_plot, 107
*Topic dissimilarity
Dissimilarity, 110
*Topic distance
cm_distance, 71
*Topic distribution,
dist_tab, 112
*Topic diversity
diversity, 114
*Topic dummy
cm_long2dummy, 75
*Topic end-mark
der_mark, 117
*Topic endmark
add_incomplete, 11
*Topic escaped
clean, 53
*Topic explicit,
formality, 124
*Topic formality,
formality, 124
*Topic frequency
dist_tab, 112
*Topic frequent_terms
freq_terms, 129
*Topic functional,
lexical_classification, 151
*Topic gender
name2sex, 165
*Topic group
chunker, 51
*Topic heatcloud
gradient_cloud, 139
*Topic heatmap
qheat, 301
*Topic incomplete-sentence
incomplete_replace, 145
*Topic incomplete
end_inc, 116
*Topic is.global
new_project, 168
*Topic justification
left_just, 150
*Topic justify,
left_just, 150
*Topic letters
word_length, 401
*Topic lexical_classification,
lexical_classification, 151
*Topic long
<table>
<thead>
<tr>
<th>Topic</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>cm_21long</td>
<td>53</td>
</tr>
<tr>
<td>merge</td>
<td>key_merge, 148</td>
</tr>
<tr>
<td>missing-value</td>
<td>NAer, 164</td>
</tr>
<tr>
<td>name</td>
<td>name2sex, 165</td>
</tr>
<tr>
<td>network</td>
<td>word_network_plot, 404</td>
</tr>
<tr>
<td>ngram</td>
<td>ngrams, 170</td>
</tr>
<tr>
<td>number-to-word</td>
<td>replace_number, 327</td>
</tr>
<tr>
<td>ordinal-to-word</td>
<td>replace_ordinal, 328</td>
</tr>
<tr>
<td>parenthesis</td>
<td>bracketX, 39</td>
</tr>
<tr>
<td>parse</td>
<td>scrubber, 344</td>
</tr>
<tr>
<td>parts-of-speech</td>
<td>formality, 124</td>
</tr>
<tr>
<td></td>
<td>lexical_classification, 151</td>
</tr>
<tr>
<td>parts-of-speech</td>
<td>pos, 230</td>
</tr>
<tr>
<td>paste</td>
<td>paste2, 175</td>
</tr>
<tr>
<td>percent</td>
<td>prop, 290</td>
</tr>
<tr>
<td>percentage</td>
<td>prop, 290</td>
</tr>
<tr>
<td>phrase_net</td>
<td>phrase_net, 177</td>
</tr>
<tr>
<td>pipe</td>
<td>%&amp;&amp;, 414</td>
</tr>
<tr>
<td>plural</td>
<td>add_s, 12</td>
</tr>
<tr>
<td>polarity</td>
<td>polarity, 220</td>
</tr>
<tr>
<td>polysyllable</td>
<td>syllable_sum, 361</td>
</tr>
<tr>
<td>position</td>
<td>word_position, 406</td>
</tr>
<tr>
<td>pos</td>
<td>formality, 124</td>
</tr>
<tr>
<td>print</td>
<td>inspect_text, 146</td>
</tr>
<tr>
<td>project</td>
<td>new_project, 168</td>
</tr>
<tr>
<td>pronouns</td>
<td>object_pronoun_type, 172</td>
</tr>
<tr>
<td></td>
<td>pronoun_type, 288</td>
</tr>
<tr>
<td></td>
<td>subject_pronoun_type, 356</td>
</tr>
<tr>
<td>proportion</td>
<td>prop, 290</td>
</tr>
<tr>
<td>question</td>
<td>question_type, 310</td>
</tr>
<tr>
<td>question-count</td>
<td>question_type, 310</td>
</tr>
<tr>
<td>random</td>
<td>random_sent, 318</td>
</tr>
<tr>
<td>rank-frequency</td>
<td>rank_freq_mplot, 319</td>
</tr>
<tr>
<td>readability</td>
<td>automated_readability_index, 32</td>
</tr>
<tr>
<td>replace</td>
<td>replacer, 324</td>
</tr>
<tr>
<td>sample</td>
<td>random_sent, 318</td>
</tr>
<tr>
<td>scale</td>
<td>outlier_labeler, 174</td>
</tr>
<tr>
<td>scale</td>
<td>multiscale, 163</td>
</tr>
<tr>
<td>sentence</td>
<td>sentSplit, 347</td>
</tr>
<tr>
<td></td>
<td>tot_plot, 372</td>
</tr>
<tr>
<td>sentence</td>
<td>random_sent, 318</td>
</tr>
<tr>
<td>sentiment</td>
<td>polarity, 220</td>
</tr>
<tr>
<td>space</td>
<td>comma_spacer, 86</td>
</tr>
<tr>
<td>span</td>
<td>cm_df2long, 69</td>
</tr>
<tr>
<td></td>
<td>cm_range2long, 78</td>
</tr>
<tr>
<td>spelling</td>
<td>check_text, 49</td>
</tr>
<tr>
<td>split</td>
<td>sentSplit, 347</td>
</tr>
<tr>
<td></td>
<td>tot_plot, 372</td>
</tr>
<tr>
<td>standardize</td>
<td>outlier_labeler, 174</td>
</tr>
<tr>
<td>statistic</td>
<td>word_stats, 411</td>
</tr>
<tr>
<td>stem</td>
<td>stemmer, 353</td>
</tr>
<tr>
<td>stopwords</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

rm_stopwords, 331
+Topic string-wrap
  strWrap, 355
+Topic structure
  qdap_df, 299
+Topic syllabication,
  syllable_sum, 361
+Topic syllable,
  syllable_sum, 361
+Topic symbol-replace
  replace_symbol, 329
+Topic text
  check_text, 49
  chunker, 51
+Topic time-span
  cm_time2long, 80
+Topic time
  cm_df2long, 69
  cm_range2long, 78
+Topic title
  Title, 371
+Topic transcript
  cm_df.transcript, 68
  read.transcript, 321
+Topic transform
  cm_code.transform, 62
  qcombine, 297
+Topic turn-of-talk
  sentSplit, 347
  tot_plot, 372
+Topic unique
  unique_by, 381
+Topic venn
  trans_venn, 378
+Topic vignette
  build_qdap_vignette, 41
+Topic word-count
  word.count, 396
+Topic word-frequency-matrix
  wfm, 384
+Topic word-list
  word_diff_list, 400
  word_list, 403
+Topic word-search
  termco, 365
+Topic wordcloud
  trans_cloud, 374
+Topic word
  word_position, 406
+Topic workflow,
  new_project, 168
+.Network, 10
%>% (%&%), 414
%bs% (Search), 345
%ex% (exclude), 120
%sw% (rm_stopwords), 331
%>%, 414
%&%, 299, 300, 414
abbreviations, 325
add_incomplete, 11
add_s, 12
adjacency_matrix, 12
adjmat (adjacency_matrix), 12
agrep, 374
all_words, 13, 14, 130
amplification.words, 310
Animate, 15
Animate.character, 15
Animate.discourse_map, 17
Animate.formality, 18
Animate.gantt, 19
Animate.gantt_plot, 20
Animate.lexical_classification, 20
Animate.polarity, 22
annotate, 16, 19, 21, 23
apply_as_df (as.tdm), 23
apply_as_tm (as.tdm), 23
as.character, 355
as.Corpus (as.tdm), 23
as.data.frame.Corpus (as.tdm), 23
as.DocumentTermMatrix (as.tdm), 23
as.dtm (as.tdm), 23
as.tdm, 23
as.TermDocumentMatrix (as.tdm), 23
as.wfm, 27
as.wfm (wfm), 384
assign, 159
automated_readability_index, 32, 89, 334
bag_o_words, 36, 332, 401
beg2char, 37
blank2NA, 38
boolean_search, 345, 377
boolean_search (Search), 345
bracketx, 39, 151, 305, 310, 311, 325,
  327–329, 349
INDEX

dist, 13, 111
dist_tab, 112
diversity, 114
DocumentTermMatrix, 23, 26, 27, 120
duplicates, 115

E, 267, 382
duplicate, (vertex_apply), 381
end_inc, 33, 116, 412, 413
end_mark, 117, 377
end_mark_by, 90, 237, 292, 335
end_mark_by (end_mark), 117
env.syl, 119
exclude, 120, 120

facet_grid, 137, 139, 217, 372
facet_wrap, 137, 139, 191, 198, 201, 206, 210, 217, 372
factor, 49
Filter, 27, 177
Filter (Filter.all_words), 121
Filter.all_words, 121
findAssocs, 395
flesch_kincaid, 91, 336
flesch_kincaid
(automated_readability_index), 32
formality, 18, 91, 124, 124, 166, 238, 292, 333, 336
freq_terms, 129
fry, 33, 88, 92, 208, 333, 337
fry (automated_readability_index), 32
function.words, 21, 151

gantt, 19, 131, 132–134, 136, 138, 139
gantt_plot, 20, 132, 133, 136, 138, 139, 207
gantt_rep, 132–134, 135, 138, 139
gantt_wrap, 19, 20, 132–134, 136, 137, 137, 372
gender, 165
genX (bracketX), 39
genXtract (bracketX), 39
gem_point, 217
gem_smooth, 200
globalenv, 147
gradient_cloud, 139, 375
graph.adjacency, 392, 405
grepl, 330
gsub, 161, 162, 350

hamlet, 141
head, 143
heatmap, 218
heatmap.2, 365
htruncdf, 142, 143

igraph.vertex.shapes, 405
imperative, 144, 144
incomp (incomplete_replace), 145
incomplete_replace, 145, 349
inspect_text, 146
is.global, 147

key.syn, 363
key_merge, 148
kullback.leibler, 149

labMT, 363
layout, 182, 391, 405
lcolsplit2df, 85
lcolsplit2df (colsplit2df), 84
left_just, 150, 150
lexical_classification, 20, 21, 151, 167, 238, 337
linsear_write, 92, 338
linsear_write
(automated_readability_index), 32
log, 383, 410
ltruncdf, 143
ltruncdf (htruncdf), 142
lview (htruncdf), 142

Maxent_POS_Tag_Annotator, 232
mcsv_r, 158, 159
mcsv_w, 88
mcsv_w (mcsv_r), 158
merge, 148
mgsub (multigsub), 161
mrajal, 160
mrajalspl, 161, 313
mtext, 245–247, 267, 309
multigsub, 161
multiscale, 163

na.omit, 18, 21, 167
NAer, 164
name2sex, 165
Network, 166
Network.formality, 166
Network.lexical_classification, 167
Network.polarity, 168
new_project, 168
new_report, 169
ngrams, 170

object.pronoun.type, 93, 172, 238, 289, 293, 338, 357
options, 282
outlier.detect, 173
outlier_labeler, 174

package.qdap (qdap), 299
par, 245–247, 267, 309
parent.frame, 147
paste, 176
paste2, 83–85, 175, 175, 176
phrase.net, 177, 177
plot.animated.character, 178
plot.animated.discourse.map, 179
plot.animated.formality, 179
plot.animated.lexical.classification, 180
plot.animated.polarity, 180
plot.automated.readability.index, 181
plot.character.table, 181
plot.cm.distance, 182
plot.cmspans, 182
plot.coleman.liau, 183
plot.combo.syllable.sum, 184
plot.cumulative.animated.formality, 184
plot.cumulative.animated.lexical.classification, 185
plot.cumulative.animated.polarity, 185
plot.cumulative.combo.syllable.sum, 186
plot.cumulative.end.mark, 186
plot.cumulative.formality, 187
plot.cumulative.lexical.classification, 187
plot.cumulative.polarity, 188
plot.cumulative.syllable.freq, 188
plot.discourse.map, 189
plot.diversity, 189
plot.end.mark, 190
plot.end.mark.by, 190
plot.end.mark.by.count, 191
plot.end.mark.by.preprocessed, 191
plot.end.mark.by.proportion, 192
plot.end.mark.by.score, 192
plot.flesch.kincaid, 193
plot.formality, 193
plot.formality.scores, 194
plot.freq.terms, 194
plot.gantt, 195
plot.igraph, 244–247, 257, 267, 268
plot.kullback.leibler, 195
plot.lexical, 196
plot.lexical.classification, 196
plot.lexical.classification.preprocessed, 198
plot.lexical.classification.score, 198
plot.linsear.write, 199
plot.linsear.write.count, 199
plot.linsear.write.scores, 200
plot.Network, 200
plot.object.pronoun.type, 201
plot.polarity, 201
plot.polarity.count, 203
plot.polarity.score, 204
plot.pos, 205
plot.pos.by, 205
plot.pos.preprocessed, 206
plot.pronoun.type, 206
plot.question.type, 207
plot.question.type.preprocessed, 207
plot.readability.count, 208
plot.readability.score, 208
plot.rmgantt, 209
plot.sent.split, 209
plot.SMOG, 210
plot.subject.pronoun.type, 210
plot.sum.cmspans, 211, 358
plot.sums.gantt, 211
plot.syllable.freq, 212
plot.table.count, 212
plot.table.proportion, 213
plot.table.score, 213
plot.termco, 214
plot.type.token.ratio, 214
plot.weighted.wfm, 215
plot.wddf, 215
plot.wfm, 215, 216
plot.word.cor, 216
plot.word_length, 217
plotNword_position, 218
plotNword_proximity, 218
plotNword_stats, 219
plotNword_stats_counts, 219
plotNword_stats(gantt), 131
polarity, 22, 93, 159, 168, 220, 222, 339
polysyllable_sum(syllable_sum), 361
pos, 94, 124, 230, 231, 239, 293, 317
pos_by, 94, 124, 236, 239, 294, 313, 339
pos_by(pos), 230
pos_tags(pos), 230
potential_NA, 235
preprocessed, 15, 88, 236, 291, 333, 382
preprocessedNcheck_spelling_interactive, 236
preprocessedNend_mark_by, 237
preprocessedNend_mark_by, 237
preprocessedNlexical_classification, 238
preprocessedNlexical_classification, 238
preprocessedNobject_pronoun_type, 238
preprocessedNpos, 239
preprocessedNpos_by, 239
preprocessedNpronoun_type, 240
preprocessedNquestion_type, 240
preprocessedNsubject_pronoun_type, 241
preprocessedNword_position, 241
pres_debate_raw2012, 242
pres_debates2012, 242
printNadjacency_matrix, 243
printNall_words, 243
printNautomated_character, 244
printNautomated_readability_index, 247
printNboolean_qdap, 248
printNcharacter_table, 248
printNcheck_spelling, 249
printNcheck_spelling_interactive, 249
printNcheck_text, 250
printNcm_distance, 73, 250
printNcoleman_liau, 251
printNcolsplitRdf, 251
printNcumulative_animated_lexical_classification, 253
printNcumulative_animated_polarity, 253
printNcumulative_combo_syllable_sum, 254
printNcumulative_end_mark, 254
printNcumulative_formality, 255
printNcumulative_lexical_classification, 255
printNcumulative_polarity, 256
printNcumulative_syllable_freq, 256
printNdissimilarity, 257
printNdiversity, 258
printNend_mark, 258
printNend_mark_by, 259
printNend_mark_by_preprocessed, 259
printNformality, 257
printNflesch_kincaid, 260
printNlexical_classification, 260
printNlexical_classification_scores, 261
printNlinsear_write, 265
printNlinsear_write_count, 265
printNlinsear_write_scores, 266
printNNetwork, 266
printNngrams, 267
printNobject_pronoun_type, 268
printNphrase_net, 268
printNphrase_net, 268
printNpolysyllable_sum, 270
printNpolarity, 269
printNpolarity_count, 269
printNpolarity_score, 270
printNpolarity_sum, 270
printNpos, 271
printNpos_by, 271
printNpos_preprocessed, 272
printNpronoun_type, 272
printNqdapproj, 273
printNqdapproj, 273
printNqdapproj, 273
printNqdapproj, 273
printNqdapproj, 273
printNqdapproj, 273
printNqdapproj, 273
printNqdapproj, 273
printNquestion_type, 274
printNquestion_type_preprocessed, 274
printNreadability_count, 275
printNreadability_score, 275
printNsent_split, 276
printNSMOG, 276
printNsubject_pronoun_type, 277
printNsum_cmspans, 278
printNsums_gantt, 278
printNsyllable_sum, 279
printNtable_count, 279
printNtable_proportion, 280
printNtable_score, 280
printNtermco, 281
printNtrunc, 281
printNtype_token_ratio, 282
printNwfm, 282
printNwfm_summary, 283
printNwhich_misspelled, 283
printNword_associate, 284
printNword_cor, 284
printNword_length, 285
printNword_list, 285
printNword_position, 286
printNword_proximity, 286
printNword_stats, 287
printNword_stats_counts, 287
pronoun_type, 95, 173, 240, 288, 294, 340, 357
prop, 82, 290, 297, 397, 398
proportions, 15, 88, 236, 291, 333, 382
proportions.character_table, 291
proportions.end_mark_by, 292
proportions.formality, 292
proportions.object_pronoun_type, 293
proportions.pos, 293
proportions.pos_by, 294
proportions.pronoun_type, 294
proportions.question_type, 295
proportions.subject_pronoun_type, 295
proportions.termsco, 296
proportions.word_length, 296
proportions.word_position, 297
qcombine, 297
qcv, 298
qdap, 299
qdap-package (qdap), 299
qdap_df, 299, 414
qheat, 181, 190–192, 201, 206, 210, 213, 218, 301, 304, 379
qprep, 305, 325, 327–329
qtheme, 306
quantile, 140
question_type, 95, 181, 205, 207, 240, 295, 303, 310, 340, 343, 377
qview, 143
qview (htruncdf), 142
raj, 312, 317
raj.act.1, 313
raj.act.1POS, 313
raj.act.2, 314
raj.act.3, 315
raj.act.4, 315
raj.act.5, 316
raj.demographics, 316
rajPOS, 317
rajsPLIT, 317
random_data (random_sent), 318
random_sent, 318
rank_freq_mplot, 319, 320
rank_freq_plot, 320
rank_freq_plot (rank_freq_mplot), 319
raw.time.span, 321
read.table, 322
read.transcript, 102, 103, 321, 322, 323, 330
regex, 40
replace_abbreviation, 305, 325, 327–329, 349
replace_contraction, 325, 326, 328, 329
replace_number, 305, 325, 327, 327, 328, 329, 344
replace_ordinal, 327, 328, 328
replace_symbol, 305, 325, 327, 328, 329
replacer, 324
right_just (left_just), 150
rm_empty_row (rm_row), 330
rm_row, 39, 330
rm_stop (rm_stopwords), 331
rm_stopwords, 331, 333, 355
sample.time.span, 332
scale, 164, 174, 175, 386
scores, 15, 33, 88, 236, 291, 333, 382
INDEX

scores.automated_readability_index, 334
scores.character_table, 334
scores.coleman_liau, 335
scores.end_mark_by, 335
scores.flesch_kincaid, 336
scores.formality, 336
scores.fry, 337
scores.lexical_classification, 337
scores.linear_write, 338
scores.object_pronoun_type, 338
scores.polarity, 339
scores.pos_by, 339
scores.pronoun_type, 340
scores.question_type, 340
scores.SMOG, 341
scores.subject_pronoun_type, 341
scores.termc, 342
scores.word_length, 342
scores.word_position, 343
scores.word_stats, 343
scrubber, 40, 305, 344
sd, 394
Search, 345
sent_detect (sentSplit), 347
sent_detect_nlp (sentSplit), 347
sentCombine (sentSplit), 347
sentiment_frame, 347
sentSplit, 24, 26, 101, 222, 300, 347, 348, 349
sink, 262
SMOG, 96, 341
SMOG (automated_readability_index), 32
space_fill, 350, 350
spaste, 351
speakerSplit, 352
sqrt, 383, 410
stem2df, 348, 349, 353
stem2df (stemmer), 353
stem_words (stemmer), 353
stemDocument, 353
stemmer, 353, 353
stopwords, 332
stringdist, 43–45, 47–49
strWrap, 355
strwrap, 262, 356

sub_holder (multigsub), 161
subject_pronoun_type, 96, 173, 241, 289, 295, 341, 356
summary.cmspans, 212, 358
summary.wdf, 360
summary.wfm, 360
syllable_count, 362, 398
syllable_count (syllable_sum), 361
syllable_sum, 361
syn (synonyms), 363
syn_frame (synonyms), 363
synonyms, 363
synonyms_frame (synonyms), 363
t.DocumentTermMatrix, 364
t.TermDocumentMatrix, 365
tbl_df, 300
term_match, 14, 108
term_match (termco), 365
termco2mat (termco), 365
termco_c, 12, 367, 370, 371
termco_d, 12, 365, 366, 371
termco_d (termco), 365
TermDocumentMatrix, 23, 24, 26, 27, 120, 386
Text (qdap_df), 299
Text<- (qdap_df), 299
theme, 16

theme_badkitchen (qtheme), 306
theme_cafe (qtheme), 306
theme_duskheat (qtheme), 306
theme_grayscale (qtheme), 306
theme_greyscale (qtheme), 306
theme_hipster (qtheme), 306
theme_nightheat (qtheme), 306
theme_norah (qtheme), 306
Title, 371
Title<- (Title), 371
TOT, 349
TOT (sentSplit), 347
tot_plot, 372
trans_cloud, 140, 374, 392
trans_context, 346, 376
trans_venn, 378
transform, 298
Trim, 379
truncdf (htruncdf), 142
type_token_ratio, 380

unbag (bag_o_words), 36
unique_by, 381

V, 267, 382
venneuler, 379
vertex_apply, 381
visual, 88, 236, 291, 382
visual.discourse_map, 383

wc (word_count), 396
weight, 383
weight.wfdf (wfm), 384
weight.wfm (wfm), 384
weight.word_proximity (word_proximity),
   409
wfdf, 386
wfdf (wfm), 384
wfm, 23–27, 111, 120, 122, 123, 146, 384, 384,
   386, 394, 395
wfm_combine (wfm), 384
wfm_expanded (wfm), 384
which_misspelled (check_spelling), 43
word_associate, 390, 395
word_cor, 394
word_count, 232, 396
word_diff_list, 400
word_length, 97, 217, 296, 342, 401
word_list, 42, 130, 374, 403
word_network_plot, 392, 404, 405
word_position, 98, 241, 297, 343, 406
word_proximity, 395, 409, 410
word_split (bag_o_words), 36
word_stats, 98, 303, 411, 411
wordcloud, 140, 196, 375, 392