Package ‘qdap’

June 19, 2013

Type Package

Title Bridging the gap between qualitative data and quantitative analysis

Version 0.2.3

Date 2013-05-18

Author Tyler Rinker

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

Depends R (>= 3.0.0), gdata, ggplot2 (>= 0.9.3.1), grid, reports,scales

Imports gridExtra, chron, grDevices, RColorBrewer, igraph, tm,wordcloud, venneuler, openNLPmodels.en, Snowball, gplots,gridExtra, openNLP, plotrix, XML, RCurl, reshape2, parallel,tools

Suggests lsa, plyr, koRpus

LazyData TRUE

Description This package 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.

Acknowledgments I would like to thank the many folks at talkstats.com and stackoverflow.com for their help in answering many R questions related to qdap. I'd like to particularly thank Dason Kurkiewicz for his constant mentoring/assistance in learning the R language, GitHub and package development.

License GPL-2

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

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

Collate
’adjacency_matrix.R’ ’all_words.R’’automated_readability_index.R’ ’bag.o.words.R’ ’blank2NA.R’’bracketX.R’ ’cap
package.R’ ’qheat.R’’qprep.R’ ’question_type.R’ ’rank_freq_plot.R’ ’raw_pro_comb.R’’read.docx.R’ ’read.transcript.

R topics documented:

`abbreviations` *Small Abbreviations Data Set*

Description

A dataset containing abbreviations and their qdap friendly form.

Format

A data frame with 14 rows and 2 variables

Details

- abv. Common transcript abbreviations
- rep. qdap representation of those abbreviations

action.verbs *Action Word List*

Description

A dataset containing a vector of action words. This is a subset of the Moby project: Moby Part-of-Speech.

Format

A vector with 1569 elements

Details

From Grady Ward's Moby project: "This second edition is a particularly thorough revision of the original Moby Part-of-Speech. Beyond the fifteen thousand new entries, many thousand more entries have been scrutinized for correctness and modernity. This is unquestionably the largest P-O-S list in the world. Note that the many included phrases means that parsing algorithms can now tokenize in units larger than a single word, increasing both speed and accuracy."

References

`http://icon.shef.ac.uk/Moby/mpos.html`

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

```
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

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

---

adverb *Adverb Word List*

Description

A dataset containing a vector of adverbs words. This is a subset of the Moby project: Moby Part-of-Speech.

Format

A vector with 13398 elements

Details

From Grady Ward's Moby project: "This second edition is a particularly thorough revision of the original Moby Part-of-Speech. Beyond the fifteen thousand new entries, many thousand more entries have been scrutinized for correctness and modernity. This is unquestionably the largest P-O-S list in the world. Note that the many included phrases means that parsing algorithms can now tokenize in units larger than a single word, increasing both speed and accuracy."

References

http://icon.shef.ac.uk/Moby/mpos.html

`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

```
all_words(text.var, begins.with = NULL, contains = NULL,
  alphabetical = TRUE)
```

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 frequency.

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

```
## 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$dialogue, contains="conc")
x3 <- all_words(raj$dialogue)
head(x3, 10)

## End(Not run)
```

automated_readability_index *Readability Measures*

Description

automated_readability_index - Apply Automated Readability Index to transcript(s) by zero or more grouping variable(s).

coleman_liau - Apply Coleman Liau Index to transcript(s) by zero or more grouping variable(s).

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, labels = "automatic",
  rm.incomplete = FALSE, ...)

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.
...	Other arguments passed to end_inc.
output	A character vector character string indicating output type. One of "valid" (default and congruent with McLaughlin's intent) or "all".
labels	A character vector character string indicating output type. One of "automatic" (default; adds labels automatically) or "click" (interactive).

Value

Returns a dataframe with selected readability statistic by grouping variable(s). The `frey` function returns a graphic representation of the readability as well as a list of two dataframe: 1) `SENTENCES_USED` and 2) `SENTENCE_AVERAGES`.

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.

References

Coleman, M., & Liau, T. L. (1975). A computer readability formula designed for machine scoring. Journal of Applied Psychology, Vol. 60, pp. 283-284.

Flesch R. (1948). A new readability yardstick. Journal of Applied Psychology. Vol. 32(3), pp. 221-233. doi: 10.1037/h0057532.

Gunning, T. G. (2003). Building Literacy in the Content Areas. Boston: Allyn & Bacon.

McLaughlin, G. H. (1969). SMOG Grading: A New Readability Formula. Journal of Reading, Vol. 12(8), pp. 639-646.

Senter, R. J., & Smith, E. A.. (1967) Automated readability index. Technical Report AMRLTR-66-220, University of Cincinnati, Cincinnati, Ohio.

Examples

```
## Not run:
AR1 <- with(rajSPLIT, automated_readability_index(dialogue, list(person, act)))
htruncdf(AR1,, 15)
AR2 <- with(rajSPLIT, automated_readability_index(dialogue, list(sex, fam.aff)))
htruncdf(AR2,, 15)

CL1 <- with(rajSPLIT, coleman_liau(dialogue, list(person, act)))
head(CL1)
CL2 <- with(rajSPLIT, coleman_liau(dialogue, list(sex, fam.aff)))
head(CL2)

SM1 <- with(rajSPLIT, SMOG(dialogue, list(person, act)))
head(SM1)
SM2 <- with(rajSPLIT, SMOG(dialogue, list(sex, fam.aff)))
head(SM2)

FL1 <- with(rajSPLIT, flesch_kincaid(dialogue, list(person, act)))
head(FL1)
FL2 <-  with(rajSPLIT, flesch_kincaid(dialogue, list(sex, fam.aff)))
head(FL2)

FR <- with(rajSPLIT, fry(dialogue, list(sex, fam.aff)))
htruncdf(FR$SENTENCES_USED)
head(FR$SENTENCE_AVERAGES)

LW1 <- with(rajSPLIT, linsear_write(dialogue, list(person, act)))
head(LW1)
LW2 <- with(rajSPLIT, linsear_write(dialogue, list(sex, fam.aff)))
head(LW2)
```

```
## End(Not run)
```

`bag.o.words` *Bag of Words*

Description

`bag.o.words` - Reduces a text column to a bag of words.

`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, ...)

breaker(text.var)

word.split(text.var)
```

Arguments

`text.var`	The text variable.
`apostrophe.remove`	logical. If `TRUE` removes apostrophe's from the output.
`...`	further arguments passed to strip function.

Value

Returns a vector of striped words.

`breaker` - returns a vector of striped words and qdap recognized endmarks (i.e., `"."`, `"!"`, `"?"`, `"*"`, `"-"`).

Examples

```
## Not run:
bag.o.words("I'm going home!")
bag.o.words("I'm going home!", apostrophe.remove = TRUE)

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)

word.split(c(NA, DATA$state))

## End(Not run)
```

beg2char *Grab Begin/End of Sting to Character*

Description

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

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

Usage

```
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 (stackoverflow.com) 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.

See Also

`rm_row`

Examples

```
## Not run:
set.seed(15)
dat <- data.frame(matrix(sample(c(month.abb[1:4], ""), 50, TRUE),
    10, byrow = TRUE), stringsAsFactors = FALSE)

dat
blank2NA(dat)

## End(Not run)
```

`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.

`genXtract` - Apply general chunk extraction to character vectors.

Usage

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

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

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

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.
`scrub`	logical. If TRUE `scrubber` will clean the text.
`with`	logical. If TRUE returns the brackets and the bracketed text.
`merge`	logical. If TRUE the results of each bracket type will be merged by sentence. FALSE returns a named list of lists of vectors of bracketed text per bracket type.
`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 checks 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 @arunning@b with /little cat#.")
genXtract(x, c("/", "@a"), c("#", "@b"))

x <- c("Where is the L1big dogL2?",
    "I think he's 98running99 with L1little catL2.")
genXtract(x, c("L1", 98), c("L2", 99))

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

## End(Not run)
```

BuckleySaltonSWL *Buckley & Salton Stopword List*

Description

A stopword list containing a character vector of stopwords.

Format

A character vector with 546 elements

Details

From Onix Text Retrieval Toolkit API Reference: "This stopword list was built by Gerard Salton and Chris Buckley for the experimental SMART information retrieval system at Cornell University. This stopword list is generally considered to be on the larger side and so when it is used, some implementations edit it so that it is better suited for a given domain and audience while others use this stopword list as it stands."

Note

Reduced from the original 571 words to 546.

References

http://www.lextek.com/manuals/onix/stopwords2.html

capitalizer *Capitalize Select Words*

Description

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

Usage

```
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

```
## 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)
```

`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

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

## End(Not run)
```

cm_code.blank *Blank Code Transformation*

Description

Transform codes with any binary operator combination.

Usage

```
cm_code.blank(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

```
## 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.transform(x, overlap.code.list=list(AB=qcv(AA, BB)))
cm_code.transform(x, combine.code.list = list(ALL=qcv(AA, BB, CC)))
cm_code.transform(x, overlap.code.list=list(AB=qcv(AA, BB)),
    combine.code.list = list(ALL=qcv(AA, BB, CC)))
overlaps <- list(AB=qcv(AA, BB), ABC=qcv(AA, BB, CC))
cm_code.transform(z, overlaps, rm.var="time")
out1 <- cm_code.transform(z, overlaps,
   exclude.code.list=list(AABB_no_CC = qcv(AA, BB, CC)), rm.var="time")
head(out1, 10)
#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")
)

dat <- cm_time2long(x, y)
head(dat, 10)
out2 <- cm_code.transform(dat, list(P=qcv(A, B), Q=qcv(B, C), R=qcv(A, B, C)),
    list(S=qcv(A, B), T=qcv(B, C), U=qcv(A, B, C)),
    list(ABnoC = qcv(A, B, C)), rm.var="variable")
head(out2, 10)

## End(Not run)
```

`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.

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")
)

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

```
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`

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.exclude(x, list(ABnoC=qcv(AA, BB, CC)))
cm_code.exclude(z, list(ABnoC=qcv(AA, BB, CC)), rm.var="time")
excludes <- list(AnoB=qcv(AA, BB), ABnoC=qcv(AA, BB, CC))
cm_code.exclude(z, excludes, 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")
)

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)
```

`cm_code.overlap` *Find Co-occurrence Between Codes*

Description

Combine co-occurrences of codes into a new code.

Usage

```
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

```
## 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))
cm_code.overlap(z, combines, "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")
)

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

```
## 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")
lapply(D2, head)
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
D1 <- cm_combine.dummy(D1, combine.code = list(CAB=qcv(AB, CC)), overlap=FALSE)
D1

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_combine.dummy` *Find Co-occurrence Between Codes*

Description

Combine code columns where they co-occur.

Usage

```
cm_combine.dummy(cm.l2d.obj, combine.code,
  rm.var = "time", overlap = TRUE)
```

Arguments

`cm.l2d.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(CAB=qcv(AB, 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

Miles, M. B. & Huberman, A. M. (1994). An expanded sourcebook: Qualitative data analysis. 2nd ed. Thousand Oaks, CA: SAGE Publications.

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
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)
```

`cm_df.temp` *Break Transcript Dialogue into Blank Code Matrix*

Description

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

Usage

```
cm_df.temp(dataframe, text.var, codes = NULL, csv = TRUE,
  file.name = 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.
`csv`	logical. If `TRUE` creates a csv in the working directory.
`file.name`	The name of the csv file. If `NULL` defaults to the dataframe name.
`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; 2-no it does not) representation of data and can be used for visualizations and statistical analysis.

References

Miles, M. B. & Huberman, A. M. (1994). An expanded sourcebook: Qualitative data analysis. 2nd ed. Thousand Oaks, CA: SAGE Publications.

See Also

`cm_range2long`, `cm_df.transcript`, `cm_df.fill`

Examples

```
## 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

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

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).
`...`	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)

DWin, Gavin Simpson and Tyler Rinker <tyler.rinker@gmail.com>.

See Also

cm_df2long, cm_df.temp

Examples

```
## 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(mraja1spl, cm_df.transcript(dialogue, list(person)))
##  with(mraja1spl, cm_df.transcript(dialogue, list(sex, fam.aff, died)))
##  with(mraja1spl, cm_df.transcript(dialogue, list(person), file="foo.doc"))
##  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 ed. Thousand Oaks, CA: SAGE Publications.

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)

## End(Not run)
```

cm_distance — *Distance Matrix Between Codes*

Description

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

Usage

```
cm_distance(dataframe, time.var = NULL, parallel = FALSE,
  code.var = "code", causal = FALSE, start.var = "start",
  end.var = "end", mean.digits = 2, sd.digits = 2,
  stan.digits = 2)
```

Arguments

`dataframe`	A data frame from the cm_x2long family (`cm_range2long`; `cm_df2long`; `cm_time2long`).
`time.var`	An optional variable to split the dataframe by (if you have data that is by various times this must be supplied).
`parallel`	logical. If `TRUE` runs the `cm_distance` on multiple cores. This is effective with larger data sets but may actually be slower with smaller data sets.
`code.var`	The name of the code variable column. Defaults to "codes" as out putted by x2long family.
`causal`	logical. If `TRUE` measures the distance between x and y given that x must proceed y.
`start.var`	The name of the start variable column. Defaults to "start" as out putted by x2long family.
`end.var`	The name of the end variable column. Defaults to "end" as out putted by x2long family.
`mean.digits`	The number of digits to be displayed in the mean matrix.
`sd.digits`	The number of digits to be displayed in the sd (standard deviation) matrix.
`stan.digits`	The number of digits to use in the standardized mean difference matrix.

Value

An object of the class cm.dist. This is a list of n lists with the following components per each list (time.var):

`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
`combined`	A matrix of combined mean, sd and n of distances between codes
`standardized`	A matrix of standardized values of distances between codes. The closer a value is to zero the closer two codes relate.

Examples

```
## 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_range2long(foo, foo2, v.name = "time"))
(out <- cm_distance(dat, time.var = "time", causal=TRUE))
names(out)
names(out$foo2)
out$foo2
#========================================
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_time2long(x))
gantt_wrap(dat, "code", border.color = "black", border.size = 5,
    sig.dig.line.freq = -2)
(a <- cm_distance(dat))
names(a)
names(a$dat)
a$dat

## End(Not run)
```

`cm_dummy2long` *Convert cm_combine.dummy Back to Long*

Description

`cm_combine.dummy` back to long.

Usage

```
cm_dummy2long(cm.comb.obj, rm.var = "time")
```

Arguments

`cm.comb.obj`	An object from cm_combine.dummy
`rm.var`	Name of the repeated measures column. Default is `"time"`.

Value

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

See Also

cm_long2dummy, cm_combine.dummy

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))
(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(D2, combine.code = combines)
head(A, 10)
B <- cm_combine.dummy(D1, combine.code = combines)
head(B, 10)

cm_dummy2long(A)
cm_dummy2long(B, "time")

## 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.
`rm.var`	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

```
## 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)
```

`cm_range.temp` *Range Code Sheet*

Description

Generates a range coding sheet for coding words.

Usage

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

Arguments

`codes`	Character vector of codes.
`file`	A connection, or a character string naming the file to print to (.txt or .doc is recommended).
`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.

References

Miles, M. B. & Huberman, A. M. (1994). An expanded sourcebook: Qualitative data analysis. 2nd ed. Thousand Oaks, CA: SAGE Publications.

See Also

`cm_time.temp`

Examples

```
## 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")
## delete("foo.txt")

## End(Not run)
```

`cm_range2long` *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)
```

Arguments

`...`	list object(s) in the form generated by `cm_time.temp`.
`v.name`	sn 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.

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 ed. Thousand Oaks, CA: SAGE Publications.

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'),
    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="")
)

(dat <- cm_range2long(foo, foo2, v.name = "time"))
gantt_wrap(dat, "code", "time")

## End(Not run)
```

cm_time.temp *Time Span Code Sheet*

Description

Generates a time span coding sheet and coding format sheet.

Usage

```
cm_time.temp(codes, start = ":00", end = NULL,
  file = NULL)
```

Arguments

codes	List of codes.
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).

References

Miles, M. B. & Huberman, A. M. (1994). An expanded sourcebook: Qualitative data analysis. 2nd ed. Thousand Oaks, CA: SAGE Publications.

See Also

cm_range.temp,

Examples

```
## Not run:
## cm_time.temp(qcv(AA, BB, CC), ":30", "7:40", file = "foo.txt")
## 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)
```

cm_time2long *Transform Codes to Start-End Times*

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

```
cm_time2long(..., v.name = "variable", list.var = TRUE,
  start.end = FALSE, debug = TRUE)
```

Arguments

...	List object(s) in the form generated by cm_time.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 passed to cm_t2l.
start.end	logical. If TRUE outputs stop and end times for each cm_time.temp list object.
debug	logical. If TRUE debugging mode is on. cm_time2long will return possible errors in time span inputs.

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 ed. Thousand Oaks, CA: SAGE Publications.

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))
gantt_wrap(dat, "code", border.color = "black", border.size = 5)

## End(Not run)
```

colSplit — *Separate a Column Pasted by paste2*

Description

Separates a paste2 column into separate columns.

Usage

```
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

```
## 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.

lcolsplit2df - Wrapper for colsplit2df designed for qdap lists that returns a list dataframes.

Usage

```
colsplit2df(dataframe, splitcol = 1, new.names = NULL,
  sep = ".", keep.orig = FALSE)

lcolsplit2df(qdap.list, keep.orig = FALSE)
```

Arguments

`dataframe`	A dataframe with a column that has been pasted together.
`splitcol`	The name of the column that has been pasted together.
`new.names`	A character vector of new names to assign to the columns. Default attempts to extract the original names before the paste.
`sep`	The character that used in `paste2` to paste the columns.
`keep.orig`	logical. If `TRUE` the original pasted column will be retained as well.
`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`, `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

(x <- question_type(DATA$state, list(DATA$sex, DATA$adult)))
ltruncdf(x)
z <- lcolsplit2df(x)
ltruncdf(z)

## End(Not run)
```

common — *Find Common Words Between Groups*

Description

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

Usage

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

Arguments

`word.list`	A list of names 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`.

Examples

```
## Not run:
a <- c("a", "cat", "dog", "the", "the")
b <- c("corn", "a", "chicken", "the")
d <- c("house", "feed", "a", "the", "chicken")
common(a, b, d, overlap=2)
common(a, b, d, overlap=3)

r <- list(a, b, d)
common(r)
common(r, overlap=2)

common(word_list(DATA$state, DATA$person)$cwl, overlap = 2)

## End(Not run)
```

common.list — *list Method for common*

Description

list Method for common

Usage

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

Arguments

`word.list`	A list of names 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.

`contractions` *Contraction Conversions*

Description

A dataset containing common contractions and their expanded form.

Format

A data frame with 65 rows and 2 variables

Details

- contraction. The contraction word.
- expanded. The expanded form of the contraction.

`DATA` *Fictitious Classroom Dialogue*

Description

A fictitious dataset useful for small demonstrations.

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

`DATA2` *Fictitious Repeated Measures Classroom Dialogue*

Description

A repeated measures version of the `DATA` dataset.

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

`DICTIONARY` *Nettalk Corpus Syllable Data Set*

Description

A dataset containing syllable counts.

Format

A data frame with 20137 rows and 2 variables

Details

- word. The word
- syllables. Number of syllables

Note

This data set is based on the Nettalk Corpus but has some researcher word deletions and additions based on the needs of the `syllable.sum` algorithm.

References

Sejnowski, T.J., and Rosenberg, C.R. (1987). "Parallel networks that learn to pronounce English text" in Complex Systems, 1, 145-168. Retrieved from: `http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Nettalk+Corpus)`

UCI Machine Learning Repository website

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

```
dir_map(loc, obj.prefix = "dat",
  col.names = c("person", "dialogue"), file = NULL,
  copy2clip = TRUE)
```

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.
`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.

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)
```

`dissimilarity` *Dissimilarity Statistics*

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

Value

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

See Also

`dist`

Examples

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

## End(Not run)
```

distTab *SPSS Style Frequency Tables*

Description

Generates a distribution table for vectors, matrices and dataframes.

Usage

```
distTab(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

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

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

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

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

## End(Not run)
```

`diversity` *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:

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

Shannon, C. E. (1948). A mathematical theory of communication. Bell System

Simpson index:

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

Simpson, E. H. (1949). Measurement of diversity. Nature 163, p. 688

Collision entropy:

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

Renyi, A. (1961). On measures of information and entropy. Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics and Probability, 1960. pp. 547-5661.

Berger Parker index:

$$D_{BP} = \frac{N_{max}}{N}$$

Berger, W. H., & Parker, F. L.(1970). Diversity of planktonic Foramenifera in deep sea sediments. Science 168, pp. 1345-1347.

Brillouin index:

$$H_B = \frac{ln(N!) - \sum ln(n_1)!}{N}$$

Magurran, A. E. (2004). Measuring biological diversity. Blackwell.

Value

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

Examples

```
## 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

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

## End(Not run)
```

`emoticon` *Emoticons Data Set*

Description

A dataset containing common emoticons (adapted from Popular Emoticon List).

Format

A data frame with 81 rows and 2 variables

Details

- meaning. The meaning of the emoticon
- emoticon. The graphic representation of the emoticon

References

`http://www.lingo2word.com/lists/emoticon_listH.html`

`end_inc` *Test for Incomplete Sentences*

Description

Test for incomplete sentences and optionally remove them.

Usage

```
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.
`warning.report`	logical. If `TRUE` prints a warning of regarding removal of incomplete sentences.
`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

```
## 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 *Sentence End marks*

Description

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

Usage

```
end_mark(text.var)
```

Arguments

text.var	The text variable.

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

```
## Not run:
end_mark(DATA$state)
end_mark(mraja1spl$dialogue)
ques <- mraja1spl[end_mark(mraja1spl$dialogue) == "?", ] #grab questions
htruncdf(ques)
non.ques <- mraja1spl[end_mark(mraja1spl$dialogue) != "?", ] #non questions
htruncdf(non.ques, 20)
```

```
ques.per <- mraja1spl[end_mark(mraja1spl$dialogue) %in% c(".", "?"), ] #grab ? and .
htruncdf(ques.per, 20)

## End(Not run)
```

`env.syl` *Syllable Lookup Environment*

Description

A dataset containing a syllable lookup environment (see `link[qdap]{DICTIONARY}`).

Format

A environment with the DICTIONARY data set.

Details

For internal use.

References

UCI Machine Learning Repository website

`env.syn` *Syllable Lookup Environment*

Description

A dataset containing a synonym lookup environment (see `link[qdap]{SYNONYM}`).

Format

A environment with

References

Scraped from: Reverso Online Dictionary. The word list fed to Reverso is the unique words from the combination of `DICTIONARY` and `labMT`.

exclude *Exclude Elements From a Vector*

Description

Quickly exclude words from a word list

Usage

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

Arguments

`word.list`	A list of words/terms 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

```
## 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$state, qcv(th), FALSE)
exclude(terms, "truth")
#all together
termco(DATA$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)

## End(Not run)
```

formality *Formality Score*

Description

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

Usage

```
formality(text.var, grouping.var = NULL,
  sort.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 formality score for all text. Also takes a single grouping variable or a list of 1 or more grouping variables.
`sort.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:

$$F = 50(\frac{n_f - n_c}{N} + 1)$$

Where:

$$f = \{noun, \ adjective, \ preposition, \ article\}$$

$$c = \{pronoun, \ verb, \ adverb, \ interjection\}$$

$$N = \sum (f \ + \ c \ + \ conjunctions)$$

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".

References

Heylighen, F., & Dewaele, J.M. (2002). Variation in the contextuality of language: An empirical measure. Context in Context, Special issue of Foundations of Science, 7 (3), 293-340.

Examples

```
## Not run:
with(DATA, formality(state, person))
(x1 <- with(DATA, formality(state, list(sex, adult))))
plot(x1)
plot(x1, short.names = TRUE)
data(rajPOS) #A data set consisting of a pos list object
x2 <- with(raj, formality(rajPOS, act))
plot(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)

## End(Not run)
```

---

gantt *Generate Unit Spans*

Description

Generates start and end times of supplied text selections (i.e., text selections are determined by any number of grouping variables).

Usage

```
gantt(text.var, grouping.var, plot = TRUE,
  units = "words", sums = FALSE, plot.colors = NULL,
  box.color = NULL, 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.
`plot`	logical. If `TRUE` plots the start-end times as a Gantt plot.
`units`	The unit of measurement to analyze. One of the strings `"character"`, `"syllable"`, `"word"`, or `"sentence"`.
`sums`	logical. If `TRUE` reports and optionally plots the total units used by grouping variable(s).
`plot.colors`	The colors of the Gantt plot bars. Either a single color or a length equal to the number of grouping variable(s).
`box.color`	A single color of the box around the Gantt plot bars.
`col.sep`	The character string to use to separate pasted variables in the merged grouping variable header/name.

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). Optionally plots a Gantt plot of the returned data.

Note

For repeated measures data output use `gantt_rep`; for a convenient wrapper that takes text and generates plots use `gantt_plot`; and for a flexible gantt plot that words with code matrix functions (cm) use `gantt_wrap`.

Author(s)

DigEmAll (`stackoverflow.com`) and Tyler Rinker <tyler.rinker@gmail.com>.

References

Clark, W. & Gantt, H. (1922) The Gantt chart, a working tool of management. New York, Ronald Press.

See Also

`gantt_rep`, `gantt_wrap`, `gantt_plot`

Examples

```
## Not run:
gantt(DATA$state, DATA$person)
gantt(DATA$state, DATA$person, sums = TRUE)
gantt(DATA$state, list(DATA$sex, DATA$adult))
x <- gantt(mraja1$dialogue, mraja1$person) #hard to see without box color
y <- gantt(mraja1$dialogue, mraja1$person, box.col = "black")
z <- gantt(mraja1$dialogue, mraja1$sex)
m <- gantt(mraja1$dialogue, list(mraja1$fam.aff, mraja1$sex),
    plot.colors = NULL)
n <- gantt(mraja1$dialogue, list(mraja1$fam.aff, mraja1$sex),
    plot.colors = "black")
o <- gantt(mraja1$dialogue, list(mraja1$fam.aff, mraja1$sex),
    plot = FALSE)
p <- gantt(mraja1$dialogue, mraja1$person, units = "characters",
    box.color = "black")
d <- gantt(mraja1$dialogue, list(mraja1$fam.aff, mraja1$sex),
    units = "characters")
e <- with(mraja1, gantt(dialogue, list(fam.aff, sex, died),
   units = "characters", sums = TRUE))
f <- gantt(mraja1$dialogue, mraja1$person, units = "syllables",
   box.color = "black", sums = TRUE)
g <- gantt(mraja1$dialogue, list(mraja1$fam.aff, mraja1$sex),
    units = "syllables")

dat <- gantt(mraja1$dialogue, list(mraja1$fam.aff, mraja1$sex),
    units = "sentences", plot.colors = 'black', sums = TRUE,
    col.sep = "_")$gantt.df
gantt_wrap(dat, fam.aff_sex, title = "Gantt Plot")

## 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

```
gantt_plot(text.var, grouping.var, 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 words with code matrix functions (cm) use `gantt_wrap`.

References

Clark, W. & Gantt, H. (1922) The Gantt chart, a working tool of management. New York, Ronald Press.

See Also

`gantt`, `gantt_rep`, `gantt_wrap`

Examples

```
## 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.margin = unit(1, "lines")) + scale_colour_grey()
z + scale_colour_brewer(palette="Dark2")

## End(Not run)
```

gantt_rep *Generate Unit Spans for Repeated Measures*

Description

Produces start and end times for occurrences for each repeated measure condition.

Usage

```
gantt_rep(rm.var, text.var, grouping.var,
  units = "words", col.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 merged grouping variable header/name.

Value

Returns a data frame of start and end times by repeated measure and grouping variable(s)

Note

For non repeated measures data/plotting use `gantt`; for a convenient wrapper that takes text and generates plots use `gantt_plot`; and for a flexible gantt plot that words with code matrix functions (cm) use `gantt_wrap`.

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)
gantt_wrap(dat, fam.aff_sex, facet.vars = "act",
    title = "Repeated Measures Gantt Plot",
    minor.line.freq = 25, major.line.freq = 100)

## End(Not run)
```

gantt_wrap *Gantt Plot*

Description

A ggplot2 wrapper that produces a Gantt plot.

Usage

```
gantt_wrap(dataframe, plot.var, facet.vars = NULL,
  fill.var = NULL, title = NULL,
  ylab = as.character(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)
```

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.
`ncol`	if an integer value is passed to this `gantt_wrap` uses `facet_wrap` rather than `facet_grid`.
`transform`	logical. If TRUE the repeated facets will be transformed from stacked to side by side.
`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:m:s 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.

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

Clark, W. & Gantt, H. (1922) The Gantt chart, a working tool of management. New York, Ronald Press.

See Also

`gantt`, `gantt_plot`, `gantt_rep`, `facet_grid`, `facet_wrap`

Examples

```
## Not run:
dat <- gantt(mraja1$dialogue, list(mraja1$fam.aff, mraja1$sex),
    units = "sentences", plot.colors = 'black', sums = TRUE,
    col.sep = "_")$gantt.df
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)
```

---

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.
`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`

Examples

```
## 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(mraja1, gradient_cloud(dialogue, died, stopwords = Top25Words,
    rot.per = .5, title="Heatcloud", title.color="orange", title.cex=1.75))
x <- with(subset(mraja1, fam.aff %in% qcv(cap, mont)),
    gradient_cloud(dialogue, fam.aff))
head(x)

## End(Not run)
```

hash *Hash/Dictionary Lookup*

Description

Creates a new environment for quick hash style dictionary lookup.

Usage

```
hash(x, mode.out = "numeric")
```

Arguments

x	A two column dataframe.
mode.out	The type of output (column 2) expected (e.g., "character", "numeric", etc.)

Value

Creates a "hash table" or a two column data frame in its own environment.

Author(s)

Bryan Goodrich and Tyler Rinker <tyler.rinker@gmail.com>.

References

http://www.talkstats.com/showthread.php/22754-Create-a-fast-dictionary

See Also

lookup, environment

Examples

```
## Not run:
(DF <- aggregate(mpg~as.character(carb), mtcars, mean))
new.hash <- hash(DF)  #numeric outcome
sapply(as.character(mtcars$carb), function(x) {
    if(exists(x, envir = new.hash)) {
        get(x, envir = new.hash)
    } else {
        NA
    }
})

new.hash <- hash(DF, "character") #character outcome
sapply(as.character(mtcars$carb), function(x) {
    if(exists(x, envir = new.hash)) {
        get(x, envir = new.hash)
    } else {
        NA
    }
})
```

```
## End(Not run)
```

`hms2sec` *Convert h:m:s to Seconds*

Description

Converts a vector of h:m:s to seconds.

Usage

```
hms2sec(x)
```

Arguments

`x`	A vector of times in h:m:s.

Value

Returns a vector of times in seconds. Generally, this function is for internal use.

See Also

`times`, `sec2hms`

Examples

```
## Not run:
hms2sec(c("02:00:03", "04:03:01"))
hms2sec(sec2hms(c(222, 1234, 55)))

## End(Not run)
```

`htruncdf` *Dataframe Viewing*

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.

Usage

```
htruncdf(dataframe, n = 10, width = 10, ...)

truncdf(dataframe, end = 10, begin = 1)

ltruncdf(dat.list, n = 6, width = 10, ...)

qview(dataframe, ...)
```

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).
`...`	Other arguments passed to `htruncdf` (`qview`; `ltruncdf`) or `head` (`htruncdf`).
`dat.list`	A list of data.frame objects.

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.

See Also

`head`

Examples

```
## Not run:
htruncdf(raj)
htruncdf(raj, 20)
htruncdf(raj, ,20)
truncdf(raj[1:10, ])
truncdf(raj[1:10, ], 40)
qview(raj)
qview(CO2)

## End(Not run)
```

`imperative` *Intuitively Remark Sentences as Imperative*

Description

Automatic imperative remarking.

Usage

```
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 `mraja1spl` 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.

Examples

```
## 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 sting) with incomplete sentence marks (.., ..., .?, ..?, en \& em dash etc. replaced with "|". If scan mode is TRUE returns a data frame with incomplete sentence location.

Examples

```
## Not run:
x <- c("the...",  "I.?", "you.", "threw..", "we?")
incomplete.replace(x)
incomp(x)
incomp(x, scan.mode = TRUE)

## End(Not run)
```

`increase.amplification.words` *Amplifying Words*

Description

A dataset containing a vector of words that amplify word meaning.

Format

A vector with 32 elements

Details

Valence shifters are words that alter or intensify the meaning of the polarized words and include negators and amplifiers. Negators are, generally, adverbs that negate sentence meaning; for example the word like in the sentence, "I do like pie.", is given the opposite meaning in the sentence, "I do not like pie.", now containing the negator not. Amplifiers are, generally, adverbs or adjectives that intensify sentence meaning. Using our previous example, the sentiment of the negator altered sentence, "I seriously do not like pie.", is heightened with addition of the amplifier seriously.

`interjections` *Interjections*

Description

A dataset containing a character vector of common interjections.

Format

A character vector with 139 elements

References

`http://www.vidarholen.net/contents/interjections/`

key_merge *Merge Demographic Information with Person/Text Transcript*

Description

Wrapper function (merge) for merging demographic information with a person/text transcript.

Usage

```
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.
defualt.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

```
## 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

Kullback, S., & Leibler, R.A. (1951). On Information and sufficiency. Annals of Mathematical Statistics 22 (1): 79-86. doi:10.1214/aoms/1177729694

Examples

```
## 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)
```

`labMT` *Language Assessment by Mechanical Turk (labMT) Sentiment Words*

Description

A dataset containing words, average happiness score (polarity), standard deviations, and rankings.

Format

A data frame with 10222 rows and 8 variables

Details

- word. The word.
- happiness_rank. Happiness ranking of words based on average happiness scores.
- happiness_average. Average happiness score.
- happiness_standard_deviation. Standard deviations of the happiness scores.
- twitter_rank. Twitter ranking of the word.
- google_rank. Google ranking of the word.
- nyt_rank. New York Times ranking of the word.
- lyrics_rank. lyrics ranking of the word.

References

Dodds, P.S., Harris, K.D., Kloumann, I.M., Bliss, C.A., & Danforth, C.M. (2011) Temporal patterns of happiness and information in a global social network: Hedonometrics and twitter. PLoS ONE 6(12): e26752. doi:10.1371/journal.pone.0026752

`http://www.plosone.org/article/fetchSingleRepresentation.action?uri=info:doi/10.1371/journal.pone.0026752.s001`

`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

```
## Not run:
left.just(DATA)
left.just(DATA, "state")
left.just(CO2[1:15,])
right.just(left.just(CO2[1:15,]))

## End(Not run)
```

`lookup` *Hash Table/Dictionary Lookup*

Description

Environment based hash table useful for large vector lookups.

Usage

```
lookup(terms, key.match, key.reassign = NULL,
  missing = NA)
```

Arguments

`terms`	A vector of terms to undergo a lookup.
`key.match`	Either a two column data frame (if data frame supplied no key reassign needed) of a match key and reassignment column or a single vector match key.
`key.reassign`	A single reassignment vector supplied if key.match is not a two column data frame.
`missing`	Value to assign to terms not matching the key.match.

Value

Outputs A new vector with reassigned values.

See Also

`new.env`

Examples

```
## Not run:
lookup(mtcars$carb, sort(unique(mtcars$carb)),
    c('one', 'two', 'three', 'four', 'six', 'eight'))
lookup(mtcars$carb, sort(unique(mtcars$carb)),
    seq(10, 60, by=10))

lookup(1:5, data.frame(1:4, 11:14))
lookup(LETTERS[1:5], data.frame(LETTERS[1:5], 100:104))

key <- data.frame(x=1:2, y=c("A", "B"))
big.vec <- sample(1:2, 3000000, T)
out <- lookup(big.vec, key)
```

```
out[1:20]

## End(Not run)
```

`mcsv_r` *Read/Write Multiple csv Files at a Time*

Description

`mcsv_w` - Read and assign multiple csv files at the same time.

`mcsv_w` - Write multiple csv files into a file at the same time.

Usage

```
mcsv_r(files, a.names = NULL, l.name = NULL, list = TRUE,
  pos = 1, envir = as.environment(pos))

mcsv_w(..., dir = NULL, open = FALSE)
```

Arguments

`files`	csv file(s) to read.
`a.names`	object names to assign the csv file(s) to. If NULL assigns the csv to the name(s) of the csv file(s) in the global environment.
`l.name`	A character vector of names to assign to the csv files (dataframes) being read in. Default (NULL) uses the names of the files in the directory without the file extension.
`list`	A character vector of length one to name the list being read in. Default is `"L1"`.
`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.
`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.

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, 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(paste(a, nms, sep="/"))
mtcarsb; CO2b
rm("mtcarsb", "CO2b")  # gone from .GlobalEnv
mcsv_r(paste(a, nms, sep="/"), paste0("foo.dat", 1:2))
foo.dat1; foo.dat2
rm("foo.dat1", "foo.dat2")  # gone from .GlobalEnv
delete("foo")

#mcsv_w EXAMPLE:
(a <- mcsv_w(mtcars, CO2, dir="foo"))
delete("foo")

## End(Not run)
```

---

mraja1 *Romeo and Juliet: Act 1 Dialogue Merged with Demographics*

Description

A dataset containing act 1 of Romeo and Juliet with demographic information.

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

http://shakespeare.mit.edu/romeo_juliet/full.html

mraja1spl *Romeo and Juliet: Act 1 Dialogue Merged with Demographics and Split*

Description

A dataset containing act 1 of Romeo and Juliet with demographic information and turns of talk split into sentences.

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

multigsub *Multiple gsub*

Description

A wrapper for `gsub` that takes a vector of search terms and a vector or single value of replacements.

Usage

```
multigsub(pattern, replacement = NULL, text.var,
  leadspace = FALSE, trailspace = FALSE, fixed = TRUE,
  trim = TRUE, ...)

mgsub(pattern, replacement = NULL, text.var,
  leadspace = FALSE, trailspace = FALSE, fixed = TRUE,
  trim = 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 replacement for matched pattern.
`text.var`	The text variable.
`leadspace`	logical. If `TRUE` inserts a leading space in the replacements.
`trailspace`	logical. If `TRUE`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.
`...`	Additional arguments passed to `gsub`.

Value

Returns a vector with the pattern replaced.

Note

The replacements occur sequentially rather than all at once. This means a previous (first in pattern string) sub could alter a later sub.

See Also

`gsub`

Examples

```
## Not run:
multigsub(c("it's", "I'm"), c("it is", "I am"), DATA$state)
mgsub(c("it's", "I'm"), c("it is", "I am"), DATA$state)
mgsub("[[:punct:]]", "PUNC", DATA$state, fixed = FALSE)

## End(Not run)
```

`multiscale` *Nested Standardization*

Description

Standardize within a subgroup and then within a group.

Usage

```
  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 output 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

```
## Not run:
dat <- with(mraja1spl, word_stats(dialogue, list(person, sex, fam.aff)))
htruncdf(colsplit2df(dat$ts), ,4)
out1 <- with(colsplit2df(dat$ts), multiscale(word.count, person))
ltruncdf(out1, 10)
out2 <- with(colsplit2df(dat$ts), multiscale(word.count,
    list(fam.aff, sex)))
ltruncdf(out2, 10)
out3 <- with(colsplit2df(dat$ts), multiscale(word.count,
    list(fam.aff, sex), original_order = FALSE))
ltruncdf(out3, 10)

## End(Not run)
```

`NAer` *Replace Missing Values (NA)*

Description

Replace missing values (`NA`) in a vector or dataframe.

Usage

```
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.

Examples

```
## Not run:
set.seed(10)
(x <- sample(c(rep(NA, 4), 1:10), 20, rep=T))
NAer(x)

set.seed(10)
(y <- data.frame(matrix(x, 5, 4))                        )
NAer(y)
NAer(y, "MISSING")

## End(Not run)
```

`name2sex` *Names to Gender Prediction*

Description

Predict gender from U.S. names (based on 1990 U.S. census data).

Usage

```
name2sex(names.list, pred.sex = TRUE,
  fuzzy.match = pred.sex, USE.NAMES = FALSE)
```

Arguments

`names.list`	Character vector containing first names.
`pred.sex`	logical. If `TRUE` overlapping M/F names will be predicted based on highest cummulative frequency. If `FALSE` the overlapping names will be denoted with a `"B"`.
`fuzzy.match`	ligical. If `TRUE` uses Levenshtein edit distance from `agrep` to predict gender from the closest name match starting with the same letter. This is computationally intensive and should not be used on larger vectors. Defaults to `pred.sex`.
`USE.NAMES`	logical. If `TRUE` names.list is used to name the gender vector.

Value

Returns a vector of predicted gender (M/F) based on first name.

Author(s)

Dason Kurkiewicz and Tyler Rinker <tyler.rinker@gmail.com>.

References

http://www.census.gov/genealogy/www/data/1990surnames/names_files.html

http://stackoverflow.com/a/818231/1000343

http://www.talkstats.com/showthread.php/31660

See Also

agrep

Examples

```
## Not run:
name2sex(qcv(mary, jenn, linda, JAME, GABRIEL, OLIVA,
    tyler, jamie, JAMES, tyrone, cheryl, drew))

name2sex(qcv(mary, jenn, linda, JAME, GABRIEL, OLIVA,
    tyler, jamie, JAMES, tyrone, cheryl, drew), FALSE)

name2sex(qcv(mary, jenn, linda, JAME, GABRIEL, OLIVA,
    tyler, jamie, JAMES, tyrone, cheryl, drew), FALSE, TRUE)

name2sex(qcv(mary, jenn, linda, JAME, GABRIEL, OLIVA,
    tyler, jamie, JAMES, tyrone, cheryl, drew), TRUE, FALSE)

## End(Not run)
```

NAMES *First Names and Gender (U.S.)*

Description

A dataset containing 1990 U.S. census data on first names.

Format

A data frame with 5494 rows and 7 variables

Details

- name. A first name.
- per.freq. Frequency in percent of the name by gender.
- cum.freq. Cumulative frequency in percent of the name by gender.
- rank. Rank of the name by gender.
- gender. Gender of the combined male/female list (M/F).
- gender2. Gender of the combined male/female list with "B" in place of overlapping (M/F) names.
- pred.sex. Predicted gender of the names with B's in `gender2` repalced with the gender that had a higher `per.freq`.

References

http://www.census.gov/genealogy/www/data/1990surnames/names_files.html

NAMES_LIST *First Names and Predictive Gender (U.S.) List*

Description

A list version of the `link[qdap]{NAMES_SEX}`) dataset broken down by first letter.

Format

A list with 26 elements

Details

Alphabetical list of dataframes with the following variables:

- name. A first name.
- gender2. Gender of the combined male/female list with "B" in place of overlapping (M/F) names.
- pred.sex. Predicted gender of the names with B's in `gender2` repalced with the gender that had a higher `per.freq`.

References

http://www.census.gov/genealogy/www/data/1990surnames/names_files.html

NAMES_SEX *First Names and Predictive Gender (U.S.)*

Description

A truncated version of the `link[qdap]{NAMES}`) dataset used for predicting.

Format

A data frame with 5163 rows and 3 variables

Details

- name. A first name.
- gender2. Gender of the combined male/female list with "B" in place of overlapping (M/F) names.
- pred.sex. Predicted gender of the names with B's in `gender2` repalced with the gender that had a higher `per.freq`.

References

http://www.census.gov/genealogy/www/data/1990surnames/names_files.html

`negation.words` *Negating Words*

Description

A dataset containing a vector of words that negate word meaning.

Format

A vector with 16 elements

Details

Valence shifters are words that alter or intensify the meaning of the polarized words and include negators and amplifiers. Negators are, generally, adverbs that negate sentence meaning; for example the word like in the sentence, "I do like pie.", is given the opposite meaning in the sentence, "I do not like pie.", now containing the negator not. Amplifiers are, generally, adverbs or adjectives that intensify sentence meaning. Using our previous example, the sentiment of the negator altered sentence, "I seriously do not like pie.", is heightened with addition of the amplifier seriously.

`negative.words` *Negative Words*

Description

A dataset containing a vector of negative words.

Format

A vector with 4783 elements

Details

A sentence containing more negative words would be deemed a negative sentence, whereas a sentence containing more positive words would be considered positive.

References

Hu, M., & Liu, B. (2004). Mining opinion features in customer reviews. National Conference on Artificial Intelligence.

`http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html`

`new_project` *Project Template*

Description

Generate a project template to increase efficiency.

Usage

```
new_project(project = "new", path = getwd(), ...)
```

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.
`...`	Other arguments passed to `new_report`.

Details

The project template includes these main directories and scripts:

- ANALYSIS - A directory containing the following analysis scripts:
  - 01_clean_data.R * initial cleaning of raw transcripts
  - 02_analysis_I.R * initial analysis
  - 03_plots.R * plotting script
- 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
- 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 txt file to put research team members' contact information
- DATA - A directory to store cleaned data (generally .RData format)
- DATA_FOR_REVIEW - A directory to put data that may need to be altered or needs to be inspected more closely
- DOCUMENTS - A directory to store documents related to the project
- PLOTS - A directory to store plots
- PROJECT_WORKFLOW_GUIDE.pdf * A pdf explaining the structure of the project template
- RAW_DATA - A directory to store non-transcript data related to the project:
  - 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)

- RAW_TRANSCRIPTS - A directory to store the raw transcripts
- REPORTS - A directory with report and presentation related tools. Please see the REPORT_WORKFLOW_GUIDE.pdf for more details
- TABLES - A directory to export tables to
- WORD_LISTS - A directory to store word lists that can be sourced and supplied to functions
- .Rprofile - Performs certain tasks such as loading libraries, data and sourcing functions upon startup in RStudio
- extra_functions.R - A script to store user made functions related to the project
  - email * A function to view, and optionally copy to the clipboard, emails for the client/lead researcher, analyst and/or other project members (information taking from ~/CORRESPONDENCE/CONTACT_INFO.txt file)
  - todo * A function to view, and optionally copy to the clipboard, non-completed tasks from the `TO_DO.txt` file
- LOG - A text file documenting project changes/needs etc.
- xxx.Rproj - A project file used by RStudio; clicking this will open the project in RStudio.
- TO_DO - A text file documenting project tasks

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

```
ngrams(text.var, grouping.var = NULL, n = 2)
```

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.
`n`	The max number of grams calculate.

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 list of vectors of ngrams per grouping.var & n (not pasted).
`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

```
## Not run:
ngrams(DATA$state, DATA$person, 2)
ngrams(DATA$state, DATA$person, 3)
ngrams(DATA$state, , 3)
with(mraja1, ngrams(dialogue, list(sex, fam.aff), 3))

## End(Not run)
```

`OnixTxtRetToolkitSWL1` *Onix Text Retrieval Toolkit Stopword List 1*

Description

A stopword list containing a character vector of stopwords.

Format

A character vector with 404 elements

Details

From Onix Text Retrieval Toolkit API Reference: "This stopword list is probably the most widely used stopword list. It covers a wide number of stopwords without getting too aggressive and including too many words which a user might search upon."

Note

Reduced from the original 429 words to 404.

References

`http://www.lextek.com/manuals/onix/stopwords1.html`

`outlier.detect` *Detect Outliers in Text*

Description

Locate possible outliers for text variables given numeric word function.

Usage

```
outlier.detect(text.var, grouping.var = NULL,
  FUN = word.count, scale.by = "grouping")
```

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.
`FUN`	A word function with a numeric vector output (e.g., `syllable.sum`, `character.count` or `word.count`).
`scale.by`	A character string indicating which dimensions to scale by. One of `"all"`, `"grouping"`, or `"both"`. Default NULL scales by all.

Value

Returns a dataframe with possible outliers.

Examples

```
## Not run:
with(DATA, outlier.detect(state))
with(DATA, outlier.detect(state, FUN = character.count))
with(DATA, outlier.detect(state, person, FUN = character.count))
with(DATA, outlier.detect(state, list(sex, adult), FUN = character.count))
with(DATA, outlier.detect(state, FUN = syllable.sum))
htruncdf(with(raj, outlier.detect(dialogue, person)), 15, 45)

## End(Not run)
```

`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 >= to 3, 2, or 1.5 standard deviations.

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)
```

---

paste2 *Paste an Unspecified Number Of Text Columns*

Description

Paste unspecified columns or a list of vectors together.

Usage

```
paste2(multi.columns, sep = ".", handle.na = TRUE,
  trim = TRUE)
```

Arguments

`multi.columns`	The multiple columns or a list of vectors to paste together.
`sep`	A character string to separate the terms.
`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.

Value

Returns a vector with row-wise elements pasted together.

Note

`paste` differs from `paste2` because `paste` does not allowed 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

Examples

```
## Not run:
v <- rep(list(state.abb[1:8],  month.abb[1:8]) , 5)
n <- sample(5:10, 1)
paste(v[1:n]) #odd looking return
paste2(v[1:n])
paste2(v[1:n], sep="|")
paste2(mtcars[1:10,], sep="|")
paste(mtcars[1:10,], sep="|") #odd looking return
paste2(CO2[1:10,], sep="|-|")

## End(Not run)
```

---

plot.character.table *Plots a character.table Object*

Description

Plots a character.table object.

Usage

```
  ## 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.diversity` *Plots a diversity object*

Description

Plots a diversity object.

Usage

```
## S3 method for class 'diversity'
plot(x, ...)
```

Arguments

`x`	The diversity object
`...`	Other arguments passed to qheat

`plot.formality` *Plots a formality Object*

Description

Plots a formality object including the parts of speech used to calculate contextual/formal speech.

Usage

```
## S3 method for class 'formality'
plot(x, point.pch = 20,
   point.cex = 0.5, point.colors = c("gray65", "red"),
   bar.colors = NULL, short.names = FALSE,
   min.wrdcnt = NULL, ...)
```

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.
`...`	ignored

Value

Invisibly returns the `ggplot2` objects that form the larger plot.

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 Gantt plot the black dots are the average polarity per grouping variable.

Usage

```
## S3 method for class 'polarity'
plot(x, bar.size = 5, low = "red",
  mid = "grey99", high = "blue",
  ave.polarity.shape = "+", alpha = 1/4, shape = 19,
  point.size = 2.5, jitter = 0.1, nrow = NULL,
  na.rm = TRUE, ...)
```

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?
`...`	ignored

Value

Invisibly returns the `ggplot2` objects that form the larger plot.

plot.pos.by *Plots a pos.by Object*

Description

Plots a pos.by object.

Usage

```
## 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.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.termco` *Plots a termco object*

Description

Plots a termco object.

Usage

```
## 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.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.

`polarity` *Polarity Score (Sentiment Analysis)*

Description

Approximate the sentiment (polarity) of text by grouping variable(s).

Usage

```
polarity(text.var, grouping.var = NULL,
  positive.list = positive.words,
  negative.list = negative.words,
  negation.list = negation.words,
  amplification.list = increase.amplification.words,
  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.
`positive.list`	A character vector of terms indicating positive reaction.
`negative.list`	A character vector of terms indicating negative reaction.
`negation.list`	A character vector of terms reversing the intent of a positive or negative word.
`amplification.list`	A character vector of terms that increases the intensity of a positive or negative 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 `end_inc`.

Details

The equation used by the algorithm to assign value to polarity to each sentence fist utilizes the sentiment dictionary (Hu and Liu, 2004) to tag each word as either positive (x_i^+), negative (x_i^-), neutral (x_i^0), negator($x_i\neg$), or amplifier ($x_i^\uparrow$). Neutral words hold no value in the equation but do affect word count (n). Each positive (x_i^+) and negative (x_i^-) word is then weighted by the amplifiers ($x_i^\uparrow$) directly proceeding the positive or negative word. Next, I consider amplification value, adding the assigned value $1/n - 1$ to increase the polarity relative to sentence length while ensuring that the polarity scores will remain between the values -1 and 1. This weighted value for each polarized word is then multiplied by -1 to the power of the number of negated ($x_i\neg$) words directly proceeding the positive or negative word. Last, these values are then summed and divided by the word count (n) yielding a polarity score (δ) between -1 and 1.

$$\delta = \frac{\sum(x_i^0, \quad x_i^\uparrow + x_i^+ \cdot (-1)^{\sum(x_i\neg)}, \quad x_i^\uparrow + x_i^- \cdot (-1)^{\sum(x_i\neg)})}{n}$$

Where:

$$x_i^\uparrow = \frac{1}{n-1}$$

Value

Returns a list of:

`all`	A dataframe of scores per row with: • group.var - the grouping variable • text.var - the text variable • wc - word count • polarity - sentence polarity score • raw - raw polarity score (considering only positive and negative words) • negation.adj.raw - raw adjusted for negation words • amplification.adj.raw - raw adjusted for amplification words • pos.words - words considered positive • neg.words - words considered negative
`group`	A dataframe with the average polarity score by grouping variable.
`digits`	integer value od 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 word 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. 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.

Also note that `polarity` assumes you've run `sentSplit`.

References

Hu, M., & Liu, B. (2004). Mining opinion features in customer reviews. National Conference on Artificial Intelligence.

`http://www.slideshare.net/jeffreybreen/r-by-example-mining-twitter-for`

See Also

`https://github.com/trestletech/Sermon-Sentiment-Analysis`

Examples

```
## Not run:
(poldat <- with(DATA, polarity(state, person)))
with(DATA, polarity(state, list(sex, adult)))
names(poldat)
truncdf(poldat$all, 8)
poldat$group
poldat2 <- with(mraja1spl, polarity(dialogue,
    list(sex, fam.aff, died)))
colsplit2df(poldat2$group)
plot(poldat)

poldat3 <- with(rajSPLIT, polarity(dialogue, person))
poldat3[["group"]][, "OL"] <- outlier.labeler(poldat3[["group"]][,
    "ave.polarity"])
poldat3[["all"]][, "OL"] <- outlier.labeler(poldat3[["all"]][,
    "polarity"])
head(poldat3[["group"]], 10)
htruncdf(poldat3[["all"]], 15, 8)
plot(poldat3)
plot(poldat3, nrow=4)

## End(Not run)
```

`pos` *Parts of Speech Tagging*

Description

`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, na.omit = FALSE,
  digits = 1, progress.bar = TRUE, 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.
`na.omit`	logical. If TRUE missing values (`NA`) will be omitted.
`digits`	Integer; number of decimal places to round when printing.
`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.
`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.
`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.
`...`	Other argument 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
`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.
`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
`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.
`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.

References

http:/opennlp.apache.org

See Also

`tagPOS`

Examples

```
## Not run:
posdat <- pos(DATA$state)
ltruncdf(posdat, 7, 4)
## str(posdat)
names(posdat)
posdat$text          #original text
posdat$POStagged     #words replaced with parts of speech
posdat$POSprop[, 1:8] #proportion of parts of speech by row
posdat$POSfreq       #frequency of parts of speech by row
```

```
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)
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)
#or more quickly - reuse the output from before
out2 <- with(DATA, pos.by(posbydat, list(adult, sex)))

## End(Not run)
```

`positive.words` *Positive Words*

Description

A dataset containing a vector of positive words.

Format

A vector with 2006 elements

Details

A sentence containing more negative words would be deemed a negative sentence, whereas a sentence containing more positive words would be considered positive.

References

Hu, M., & Liu, B. (2004). Mining opinion features in customer reviews. National Conference on Artificial Intelligence.

http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html

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$state)
# USE TO SELCTIVELY REPLACE CELLS WITH MISSING VALUES
DATA$state[potential_NA(DATA$state, 20)$row[-c(3)]] <- NA
DATA
DATA <- qdap::DATA

## End(Not run)
```

preposition *Preposition Words*

Description

A dataset containing a vector of common prepositions.

Format

A vector with 162 elements

`print.adjacency_matrix` *Prints an adjacency_matrix Object*

Description

Prints an adjacency_matrix object.

Usage

```
## S3 method for class 'adjacency_matrix'
print(x, ...)
```

Arguments

`x`	The adjacency_matrix object.
`...`	ignored

`print.character.table` *Prints a character.table object*

Description

Prints a character.table object.

Usage

```
## S3 method for class 'character.table'
print(x, digits = 2,
   percent = NULL, zero.replace = NULL, ...)
```

Arguments

`x`	The character.table 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.cm_distance` *Prints a cm_distance Object*

Description

Prints a cm_distance object.

Usage

```
## S3 method for class 'cm_distance'
print(x, ...)
```

Arguments

`x`	The cm_distance object.
`...`	ignored

`print.colsplit2df` *Prints a colsplit2df Object.*

Description

Prints a colsplit2df object.

Usage

```
## S3 method for class 'colsplit2df'
print(x, ...)
```

Arguments

`x`	The colsplit2df object
`...`	ignored

`print.dissimilarity` *Prints a dissimilarity object*

Description

Prints a dissimilarity object.

Usage

```
## 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

```
## S3 method for class 'diversity'
print(x, digits = 3, ...)
```

Arguments

`x`	The diversity object
`digits`	Number of decimal places to print.
`...`	ignored

`print.formality` *Prints a formality Object*

Description

Prints a formality object.

Usage

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

Arguments

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

print.kullback.leibler *Prints a kullback.leibler Object.*

Description

Prints a kullback.leibler object.

Usage

```
## S3 method for class 'kullback.leibler'
print(x, digits = 3, ...)
```

Arguments

x	The kullback.leibler object
digits	Number of decimal places to print.
...	ignored

print.ngram *Prints an ngram object*

Description

Prints an ngram object

Usage

```
## S3 method for class 'ngram'
print(x, ...)
```

Arguments

x	The ngram object
...	ignored

`print.polarity` *Prints a polarity Object*

Description

Prints a polarity object.

Usage

```
 ## S3 method for class 'polarity'
print(x, digits = NULL, ...)
```

Arguments

`x`	The polarity object.
`digits`	Number of decimal places to print.
`...`	ignored

`print.pos` *Prints a pos Object.*

Description

Prints a pos object.

Usage

```
 ## 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

```
  ## 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

print.qdapProj *Prints a qdapProj Object*

Description

Prints a qdapProj object.

Usage

```
  ## S3 method for class 'qdapProj'
 print(x, ...)
```

Arguments

x	The qdapProj object.
...	ignored

print.question_type — *Prints a question_type object*

Description

Prints a question_type object

Usage

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

Arguments

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

print.termco — *Prints a termco object.*

Description

Prints a termco object.

Usage

```
## 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.v.outer` *Prints a v.outer Object.*

Description

Prints a v.outer object.

Usage

```
## S3 method for class 'v.outer'
print(x, digits = 3, ...)
```

Arguments

`x`	The v.outer object
`digits`	Number of decimal places to print.
`...`	ignored

`print.word_associate` *Prints a word_associate object*

Description

Prints a word_associate object.

Usage

```
## S3 method for class 'word_associate'
print(x, ...)
```

Arguments

`x`	The word_associate object
`...`	ignored

`print.word_list` *Prints a word_list Object*

Description

Prints a word_list object.

Usage

```
## S3 method for class 'word_list'
print(x, ...)
```

Arguments

`x`	The word_list object
`...`	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

prop *Convert Raw Numeric Matrix or Data Frame to Proportions*

Description

Convert a raw matrix or dataframe to proportions/percents. Divides each element of a column by the column sum.

Usage

```
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

```
## Not run:
y <- wfdf(DATA$state, DATA$person, stopwords = c("your", "yours"),
    margins = TRUE)
prop(wfm(wfdf = y), 4)[1:10, ]        #as a proportion
prop(wfm(wfdf = y), 4, TRUE)[1:10, ]  #as a percentage
heatmap(prop(wfm(wfdf = y), 4))
wdstraj <- word_stats(rajSPLIT$dialogue, rajSPLIT$person)
prop(wdstraj$gts[, -1], 5)[1:15, 1:6]

## End(Not run)
```

`qcombine` *Combine Columns*

Description

Quickly combine columns (summed) and rename.

Usage

```
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

```
## 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

```
  qcv(..., terms = NULL, space.wrap = FALSE,
    trailing = FALSE, leading = FALSE, split = " ",
    rm.blank = TRUE)
```

Arguments

`...`	Character objects. Either ... or `terms` argument must be utilized.
`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.

Value

Returns a character vector.

See Also

`c`

Examples

```
## 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.

---

qheat *Quick Heatmap*

Description

A quick heatmap function for visualizing typical qdap dataframe/matrix outputs.

Usage

```
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)
```

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.

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

```
## Not run:
dat <- sentSplit(DATA, "state")
ws.ob <- with(dat, word_stats(state, list(sex, adult), tot=tot))
qheat(ws.ob)
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)

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 bracketX, replace_number, replace_symbol, replace_abbreviation and scrubber to quickly prepare text for analysis. Care should be taken with this function to ensure data is properly formatted and complete.

Usage

```
qprep(text.var, rm.dash = TRUE, bracket = "all",
  missing = NULL, names = FALSE,
  abbreviation = qdap::abbreviations, replace = NULL,
  ignore.case = TRUE, num.paste = "separate", ...)
```

Arguments

`text.var`	The text variable.
`rm.dash`	logical. If TRUE dashes will be removed.
`bracket`	The type of bracket (and encased text) to remove. This is one of the strings `"curly"`, `"square"`, `"round"`, `"angle"` and `"all"`. These strings correspond to: {, [, (, < or all four types. Also takes the argument `NULL` which turns off this parsing technique.
`missing`	Value to assign to empty cells.
`names`	logical. If TRUE the sentences are given as the names of the counts.
`abbreviation`	A two column key of abbreviations (column 1) and long form replacements (column 2) or a vector of abbreviations. Default is to use qdap's abbreviations data set. Also takes the argument `NULL` which turns off this parsing technique.
`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.
`num.paste`	A character string c(`"separate"`, `"combine"`); `"separate"` will treat each word section as separate, `"combine"` will lump the sections together as one word. Also takes the argument `NULL` which turns off this parsing technique.
`...`	Other arguments passed to `link[qdap]{replace_symbol}`.

Note

Care should be taken with this function to ensure data is properly formatted and complete.

See Also

`bracketX`, `replace_abbreviation`, `replace_number`, `replace_symbol`

Examples

```
## Not run:
x <- "I like 60 (laughter) #d-bot and $6 @ the store w/o 8p.m."
qprep(x)

## End(Not run)
```

`question_type` *Count of Question Type*

Description

Transcript apply question counts.

Usage

```
question_type(text.var, grouping.var = NULL,
  neg.cont = FALSE, percent = TRUE, zero.replace = 0,
  digits = 2)
```

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.
`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.

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

The interrogative word that is found first (with the exception of "ok", "right" and "correct") in the question determines the sentence type. "ok", "right" and "correct" sentence types are determined if the sentence is a question with no other interrogative words found and "ok", "right" or "correct" is the last word of the sentence. Those interrogative sentences beginning with the word "you" are categorized as implying do or does question type, though the use of do/does is not explicit. 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.
`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.

Examples

```
## Not run:
(x <- question_type(DATA$state, DATA$person))
truncdf(x$raw, 15)
x$count
plot(x)
plot(x, label = TRUE)
plot(x, label = TRUE, text.color = "red")
question_type(DATA$state, DATA$person, percent = FALSE)
DATA[8, 4] <- "Won't I distrust you?"
question_type(DATA$state, DATA$person)
DATA <- qdap::DATA
```

```
with(DATA, question_type(state, list(sex, adult)))

out1 <- with(mraja1spl, question_type(dialogue, person))
## out1
out2 <- with(mraja1spl, question_type(dialogue, list(sex, fam.aff)))
## out2
out3 <- with(mraja1spl, question_type(dialogue, list(sex, fam.aff),
   percent = FALSE))
plot(out3, label = TRUE, lab.digits = 3)

## End(Not run)
```

`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.

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

`raj.act.1` *Romeo and Juliet: Act 1*

Description

A dataset containing Romeo and Juliet: Act 1.

Format

A data frame with 235 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.2` *Romeo and Juliet: Act 2*

Description

A dataset containing Romeo and Juliet: Act 2.

Format

A data frame with 205 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.3` *Romeo and Juliet: Act 3*

Description

A dataset containing Romeo and Juliet: 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`

raj.act.4 *Romeo and Juliet: Act 4*

Description

A dataset containing Romeo and Juliet: 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.

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.demographics` *Romeo and Juliet Demographics*

Description

A dataset containing Romeo and Juliet demographic information for the characters.

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

References

`http://shakespeare.mit.edu/romeo_juliet/full.html`

`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).

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.

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

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

```
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.ammount = 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 output 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.ammount`	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.

References

Zipf, G. K. (1949). Human behavior and the principle of least effort. Cambridge, Massachusetts: Addison-Wesley. p. 1.

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(mraja1spl$dialogue, mraja1spl$person, ncol = 5,
    hap.col = "purple")
ltruncdf(x2, 10)
invisible(rank_freq_mplot(mraja1spl$dialogue, mraja1spl$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(mraja1spl , word_list(dialogue, person, cut.n = 10,
    cap.list=unique(mraja1spl$person)))
x3 <- rank_freq_plot(mod$fwl$Romeo$WORD, mod$fwl$Romeo$FREQ, title.ext = 'Romeo')
ltruncdf(x2, 10)
ltruncdf(rank_freq_plot(mod$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)
```

---

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, ...)
```

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.
`...`	Further arguments to be passed to `read.table`.

Value

Returns a dataframe of dialogue and people.

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:
gsub("trans1.docx", "", system.file("extdata/transcripts/trans1.docx", 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!"

read.transcript(text=trans)

## End(Not run)
```

`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

```
## Not run:
replacer(mtcars[1:10, ], 0, "REP")
replacer(mtcars[1:10, ], 4, NA)
replacer(c("a", "b"), "a", "foo")
#replace missing values (NA)
dat <- data.frame(matrix(sample(c(1:3, NA), 25, TRUE), ncol=5))
replacer(dat, NA, "FOO")

## End(Not run)
```

replace_abbreviation *Replace Abbreviations*

Description

This function replaces abbreviations with long form.

Usage

```
replace_abbreviation(text.var,
  abbreviation = qdap::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 qdap'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`

Examples

```
## 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)
```

replace_contraction *Replace Contractions*

Description

This function replaces contractions with long form.

Usage

```
replace_contraction(text.var,
  contraction = qdap::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 qdap'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.

See Also

bracketX, qprep, replace_abbreviation, replace_number, replace_symbol

Examples

```
## 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)

#create abbreviation and replacement vectors
abv <- c("isn't", "I'd")
repl <- c("is not", "I would")

replace_abbreviation(x, abv, repl)

(KEY <- rbind(abbreviations, data.frame(abv = abv, rep = repl)))
replace_abbreviation(x, KEY)

## 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 = "separate")
```

Arguments

`text.var`	The text variable.
`num.paste`	A character vector of either `"separate"` or `"combine"`. Of `"separate"` is specified the elements of larger numbers are separated with spaces. If `"combine"` is selected the elements will be joined without spaces.

Value

Returns a vector with abbreviations replaced.

References

Fox, J. (2005). Programmer's niche: How do you spell that number? R News. Vol. 5(1), pp. 51-55.

See Also

`bracketX`, `qprep`, `replace_abbreviation`, `replace_contraction`, `replace_symbol`

Examples

```
## 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, "combine")

## 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

```
## 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

```
rm_row(dataframe, search.column, terms)

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.

Value

rm_row - returns a dataframe with the termed/markered rows removed.

rm_empty_row - returns a dataframe with empty rows removed.

Examples

```
## 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_empty_row EXAMPLE:
x <- matrix(rep(" ", 4), ncol =2)
dat <- DATA[, c(1, 4)]
colnames(x) <- colnames(dat)
(dat <- data.frame(rbind(dat, x)))
rm_empty_row(dat)

## End(Not run)
```

scrubber *Clean Imported Text*

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.
`fix.comma`	logical If TRUE removes any spaces before a comma.
`fix.space`	logical. If TRUE extra spaces before endmarks are removed.
`rm.quote`	logical If TRUE removes any \".
`...`	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?\"")
scrubber(x)
scrubber(x, TRUE)

## End(Not run)
```

`Search` *Search Columns of a Data Frame*

Description

Find terms located in columns of a data frame.

Usage

```
Search(dataframe, term, column.name = NULL,
  max.distance = 0.02, ...)
```

Arguments

`dataframe`	A dataframe object to search.
`term`	A character vector term 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).
`...`	Other arguments passed to `agrep`.

Value

Returns the rows of the data frame that match the search term.

Examples

```
## Not run:
(SampDF <- data.frame("islands"=names(islands)[1:32],mtcars))

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)

## End(Not run)
```

sec2hms *Convert Seconds to h:m:s*

Description

Converts a vector of seconds to h:m:s.

Usage

```
sec2hms(x)
```

Arguments

x	A vector of times in seconds.

Value

Returns a vector of times in h:m:s format. Generally, this function is for internal use.

See Also

times, hms2sec

Examples

```
## Not run:
sec2hms(c(256, 3456, 56565))

## End(Not run)
```

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.

Usage

```
sentSplit(dataframe, text.var,
  endmarks = c("?", ".", "!", "|"),
  incomplete.sub = TRUE, rm.bracket = TRUE,
  stem.col = FALSE, text.place = "right", ...)

sentCombine(text.var, grouping.var = NULL,
  as.list = FALSE)

TOT(tot)
```

Arguments

`dataframe`	A dataframe that contains the person and text variable.
`text.var`	The text variable.
`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"`.
`...`	Additional options passed to `stem2df`.
`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.
`tot`	A tot column from a `sentSplit` output.
`as.list`	logical. If TRUE returns the output as a list. If false the output is returned as a dataframe.

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).

Author(s)

Dason Kurkiewicz and Tyler Rinker <tyler.rinker@gmail.com>.

See Also

`bracketX`, `incomplete.replace`, `stem2df` , `TOT`

Examples

```
## Not run:
#sentSplit EXAMPLE:
sentSplit(DATA, "state")
sentSplit(DATA, "state", stem.col = TRUE)
sentSplit(DATA, "state", text.place = "left")
sentSplit(DATA, "state", text.place = "original")
sentSplit(raj, "dialogue")[1:20, ]

#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)

## End(Not run)
```

space_fill *Replace Spaces*

Description

Replace spaces in words groups that should be grouped together.

Usage

```
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

`link[qdap]{space_fill}` is useful for keeping grouped words together. Many functions in qdap take a `char.keep` 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 trailing and/or leading spaces.

Note

`link[qdap]{strip}` by default does not remove the double tilde `"~~"` character.

Examples

```
## 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

```
spaste(terms, trailing = TRUE, leading = TRUE)
```

Arguments

`terms`	A character vector of terms to insert trailing and/or leading spaces.
`leading`	logical. If TRUE inserts a leading space in the terms.
`trailing`	logical. If TRUE inserts a trailing space in the terms.

Value

Returns a character vector with trailing and/or leading spaces.

Examples

```
## 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(".", "_", ";")
`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_sally_sam",
    "greg.sally", "sam; sally")

speakerSplit(DATA, sep = c(".", "_", ";"))

DATA <- qdap::DATA  #reset DATA

## End(Not run)
```

`stemmer` *Stem Text*

Description

`stemmer` - Stems a vector of text strings.

`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

```
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

```
## 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")
truncdf(out2, 30)

## End(Not run)
```

`stopwords` *Remove Stopwords*

Description

Transcript apply the removal of stopwords.

Usage

```
stopwords(textString, stopwords = Top25Words,
  unlist = FALSE, separate = TRUE, strip = FALSE,
  unique = FALSE, char.keep = NULL, names = FALSE,
  ignore.case = TRUE, apostrophe.remove = FALSE, ...)
```

Arguments

`textString`	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 stopwords 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 textString.
`ignore.case`	logical. If TRUE stop words will be removed regardless of case. Additionally, case will be stripped from the text. If FALSE stopwords 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

```
## Not run:
stopwords(DATA$state)
stopwords(DATA$state, tm::stopwords("english"))
stopwords(DATA$state, Top200Words)
stopwords(DATA$state, Top200Words, strip = TRUE)
stopwords(DATA$state, Top200Words, separate = FALSE)
stopwords(DATA$state, Top200Words, separate = FALSE, ignore.case = FALSE)
stopwords(DATA$state, Top200Words, unlist = TRUE)
stopwords(DATA$state, Top200Words, unlist = TRUE, strip=TRUE)
stopwords(DATA$state, Top200Words, unlist = TRUE, unique = TRUE)

## End(Not run)
```

`strip` *Strip Text*

Description

Strip text of unwanted characters.

Usage

```
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

stopwords

Examples

```
## 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

```
strWrap(text = "clipboard", width = 70, copy2clip = TRUE)
```

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

```
## 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)
```

syllable.sum *Syllabication*

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

```
syllable.sum(text.var, parallel = FALSE)

syllable.count(text, remove.bracketed = TRUE,
  algorithm.report = FALSE)

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).

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

Sejnowski, T.J., and Rosenberg, C.R. (1987). "Parallel networks that learn to pronounce English text" in Complex Systems, 1, 145-168.

Examples

```
## Not run:
syllable.count("Robots like Dason lie.")
syllable.count("Robots like Dason lie.", algorithm.report = TRUE)
syllable.sum(DATA$state)
polysyllable.sum(DATA$state)
combo_syllable.sum(DATA$state)

## End(Not run)
```

SYNONYM *Synonyms Data Set*

Description

A dataset containing words and possible synonym matches.

Format

A data frame with 11050 rows and 2 variables

Details

- word. The look up word.
- match.string. A single string of possible matches.

Note

Intended for internal use with the `synonyms` function.

References

Scraped from: Reverso Online Dictionary. The word list fed to Reverso is the unique words from the combination of `DICTIONARY` and `labMT`.

synonyms — *Search For Synonyms*

Description

Search for synonyms that match term(s).

Usage

```
synonyms(terms, return.list = TRUE, multiwords = TRUE,
  report.null = TRUE)

syn(terms, return.list = TRUE, multiwords = TRUE,
  report.null = TRUE)
```

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.

Value

Returns a list of vectors or vector of possible words that match term(s).

References

The synonyms dictionary (see `SYNONYM`) 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

```
## 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)

## End(Not run)
```

tdm — *Convert/Generate Term Document Matrix*

Description

Create term document matrices from raw text or `wfm` for use with other text analysis packages.

Usage

```
tdm(text.var, grouping.var = NULL, ...)
```

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.
`...`	Other arguments passed to `wfm`.

Details

Similar to the `tm` package's `TermDocumentMatrix` though not identical.

Examples

```
## Not run:
x <- wfm(DATA$state, DATA$person)
tdm(x)
library(lsa)
lsa(tdm(x), dims=dimcalc_share())
lsa(tdm(DATA$state, DATA$person), dims=dimcalc_share())

## End(Not run)
```

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.2`).

Usage

```
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, ...)

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, ...)

term.match(text.var, terms, return.list = TRUE,
  apostrophe.remove = FALSE)

termco2mat(dataframe, drop.wc = TRUE, short.term = TRUE,
  rm.zerocol = FALSE, no.quote = TRUE, transform = TRUE,
  trim.terms = TRUE)
```

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.list`	A list of named character vectors.
`short.term`	logical. If TRUE column names are trimmed versions of the match list, otherwise the terms are wrapped with 'term(phrase)'
`ignore.case`	logical. If TRUE case is ignored.
`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.
`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`.
`zero.replace`	Value to replace 0 values with.
`...`	Other argument supplied to strip.
`match.string`	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.).
`terms`	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.

`return.list`	logical. If TRUE returns the output for multiple terms as a list by term rather than a vector.
`dataframe`	A termco (or termco.d) dataframe or object.
`drop.wc`	logical. If TRUE the word count column will be dropped.
`rm.zerocol`	logical. If TRUE any column containing all zeros will be removed from the matrix.
`no.quote`	logical. If TRUE the matrix will be printed without quotes if it's character.
`transform`	logical. If TRUE the matrix will be transformed.
`trim.terms`	logical. If TRUE trims the column header/names to ensure there is not a problem with spacing when using in other R functions.

Value

`termco` & `termco.d` - both return a list, of class "termco.d", 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 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`

Examples

```
## Not run:
#termco examples:

term <- c("the ", "she", " wh")
with(raj.act.1,  termco(dialogue, person, term))
# 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)))
names(dat)
dat$rnp  #useful for presenting in tables
dat$raw  #prop and raw are useful for performing calculations
dat$prop
datb <- with(raj.act.1, termco(dialogue, person, ml,
    short.term = FALSE, elim.old=FALSE))
ltruncdf(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)

## End(Not run)
```

`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`

`text2color` *Map Words to Colors*

Description

A dictionary lookup that maps words to colors.

Usage

```
text2color(words, recode.words, colors)
```

Arguments

`words`	A vector of words.
`recode.words`	A vector of unique words or a list of unique word vectors that will be matched against corresponding colors.
`colors`	A vector of colors of equal in length to recode.words + 1(the +1 is for unmatched words).

Value

Returns a vector of mapped colors equal in length to the words vector.

See Also

`lookup`

Examples

```
## Not run:
set.seed(10)
x <- data.frame(X1 = sample(Top25Words[1:10], 20, TRUE))

#blue was recycled
text2color(x$X1, qcv(the, and, is), qcv(red, green, blue))
text2color(x$X1, qcv(the, and, is), qcv(red, green, blue, white))
x$X2 <- text2color(x$X1, list(qcv(the, and, is), "that"),
    qcv(red, green, white))
x

## End(Not run)
```

Top100Words *Fry's 100 Most Commonly Used English Words*

Description

A stopword list containing a character vector of stopwords.

Format

A character vector with 100 elements

Details

Fry's Word List: The first 25 make up about one-third of all printed material in English. The first 100 make up about one-half of all printed material in English. The first 300 make up about 65% of all printed material in English."

References

Fry, E. B. (1997). Fry 1000 instant words. Lincolnwood, IL: Contemporary Books.

Top200Words *Fry's 200 Most Commonly Used English Words*

Description

A stopword list containing a character vector of stopwords.

Format

A character vector with 200 elements

Details

Fry's Word List: The first 25 make up about one-third of all printed material in English. The first 100 make up about one-half of all printed material in English. The first 300 make up about 65% of all printed material in English."

References

Fry, E. B. (1997). Fry 1000 instant words. Lincolnwood, IL: Contemporary Books.

`Top25Words` *Fry's 25 Most Commonly Used English Words*

Description

A stopword list containing a character vector of stopwords.

Format

A character vector with 25 elements

Details

Fry's Word List: The first 25 make up about one-third of all printed material in English. The first 100 make up about one-half of all printed material in English. The first 300 make up about 65% of all printed material in English."

References

Fry, E. B. (1997). Fry 1000 instant words. Lincolnwood, IL: Contemporary Books.

`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)
```

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 (character string).
`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.

`ncol`	if an integer value is passed to this `gantt_wrap` uses `facet_wrap` rather than `facet_grid`.
`transform`	logical. If TRUE the repeated facets will be transformed from stacked to side by side.
`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.

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", space=.03)
tot_plot(dataframe, "state", "sex")
tot_plot(mraja1, "dialogue", "fam.aff", tot=FALSE)
tot_plot(mraja1, "dialogue", "died", tot=FALSE)
tot_plot(mraja1, "dialogue", c("sex", "fam.aff"), tot=FALSE) +
    scale_fill_hue(l=40)
tot_plot(mraja1, "dialogue", c("sex", "fam.aff"), tot=FALSE)+
    scale_fill_brewer(palette="Spectral")
tot_plot(mraja1, "dialogue", c("sex", "fam.aff"), tot=FALSE)+
    scale_fill_brewer(palette="Set1")

## End(Not run)
```

`trans.cloud` *Word Clouds by Grouping Variable*

Description

Produces word clouds with optional theme coloring by grouping variable.

Usage

```
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 output 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.
`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.
`title.names`	Optional vector of title names equal in length to the grouping.var that will override the default use of the grouping.var names.
`proportional`	logical. If TRUE scales the word clouds across grouping.var to allow cloud to cloud comparisons.
`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.
`legend`	A character vector of names corresponding to the number of vectors in target.words.
`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.
`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. If `char.keep` is NULL, `char2space` will activate this argument.

Value

Returns a series of word cloud plots with target words (themes) colored.

See Also

`wordcloud`, `gradient_cloud`

Examples

```
## 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))),
        unique(unlist(terms))),
    cloud.colors=qcv(red, green, blue, black, gray65),
    expand.target=FALSE, proportional=TRUE, legend=names(terms)))

with(mraja1, trans.cloud(dialogue, person,
    target.words=list(positive=positive.words, negative=negative.words,
         negator=negation.words, amplifier=increase.amplification.words),
    cloud.colors=qcv(green, red, black, orange, 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)))

## End(Not run)

#' DATA <- qdap::DATA  ## Reset DATA
```

`trans.venn` *Venn Diagram by Grouping Variable*

Description

Produce a Venn diagram by grouping variable.

Usage

```
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 output 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.
`legend.text.col`	The color used for the legend text.
`legend.horiz`	logical; if TRUE, set the legend horizontally rather than vertically.
`...`	Other arguments passed to plot.

Value

Returns a Venn plot by grouping variable(s).

Warning

The algorithm used to overlap the Venn circles becomes increasingly overburdened and less accurate with increased grouping variables. An alternative is to use a network plot with dissimilarity measures labeling the edges between nodes (grouping variables).

See Also

`venneuler`

Examples

```
## Not run:
with(DATA , trans.venn(state, person, legend.location = "topright"))
#the plot below will take a considerable amount of time to plot
with(raj.act.1 , trans.venn(dialogue, person, legend.location = "topleft"))

## End(Not run)
```

`Trim` *Remove Leading/Trailing White Space*

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

```
## Not run:
(x <- c("  talkstats.com ", "   really? ", " yeah"))
Trim(x)

## End(Not run)
```

---

url_dl *Download Instructional Documents*

Description

This function enables downloading documents for future instructional training.

Usage

```
url_dl(..., url = 61803503)
```

Arguments

`...`	Document names to download.
`url`	The download url or dropbox key.

Value

Places a copy of the downloaded document in the users working directory.

Note

Not intended for general use.

Examples

```
## Not run:
# download transcript of the debate to working directory
url_dl(pres.deb1.docx, pres.deb2.docx, pres.deb3.docx)

# load multiple files with read transcript and assign to working directory
dat1 <- read.transcript("pres.deb1.docx", c("person", "dialogue"))
dat2 <- read.transcript("pres.deb2.docx", c("person", "dialogue"))
dat3 <- read.transcript("pres.deb3.docx", c("person", "dialogue"))

docs <- qcv(pres.deb1.docx, pres.deb2.docx, pres.deb3.docx)
dir() %in% docs
delete(docs)    #remove the documents
dir() %in% docs

## End(Not run)
```

v.outer *Vectorized Version of outer*

Description

Vectorized outer.

Usage

```
v.outer(x, FUN, ...)
```

Arguments

x	A matrix, dataframe or equal length list of vectors.
FUN	A vectorized function.
...	Other arguments passed to the function supplied to FUN.

Value

Returns a matrix with the vectorized outer function.

Author(s)

Vincent Zoonekynd and Tyler Rinker <tyler.rinker@gmail.com>.

See Also

outer, cor

Examples

```
## Not run:
pooled.sd <- function(x, y) {
    n1 <- length(x)
    n2 <- length(y)
    s1 <- sd(x)
    s2 <- sd(y)
    sqrt(((n1-1)*s1 + (n2-1)*s2)/((n1-1) + (n2-1)))
}

euc.dist <- function(x,y) sqrt(sum((x - y) ^ 2))
sum2 <- function(x, y) sum(x, y)

v.outer(mtcars, cor)
v.outer(mtcars, pooled.sd)
v.outer(mtcars[, 1:7], euc.dist)
v.outer(mtcars[, 1:7], sum2)

#mtcars as a list
mtcars2 <- lapply(mtcars[, 1:7], "[")
v.outer(mtcars2, cor)
v.outer(mtcars2, cor,  method = "spearman")
v.outer(mtcars2, pooled.sd)
```

```
print(v.outer(mtcars[, 1:7], pooled.sd), digits = 1)
print(v.outer(mtcars[, 1:7], pooled.sd), digits = NULL)
v.outer(mtcars2, euc.dist)
v.outer(mtcars2, sum2)

wc3 <- function(x, y) sum(sapply(list(x, y), wc, byrow = FALSE))
L1 <- word_list(DATA$state, DATA$person)$cwl
(x <- v.outer(L1, wc3))
diag(x) <- (sapply(L1, length))
x

## End(Not run)
```

---

wfm *Word Frequency Matrix*

Description

`wfm` - Generate a word frequency matrix by grouping variable(s).

`wfdf` - Generate a word frequency data frame by grouping variable.

`wfm.expanded` - Expand a word frequency matrix to have multiple rows for each word.

`wf.combine` - Combines words (rows) of a word frequency dataframe (`wfdf`) together.

Usage

```
wfm(text.var = NULL, grouping.var = NULL, wfdf = NULL,
  output = "raw", stopwords = NULL, char2space = "~~",
  ...)

wfdf(text.var, grouping.var = NULL, stopwords = NULL,
  margins = FALSE, output = "raw", digits = 2,
  char2space = "~~", ...)

wfm.expanded(text.var, grouping.var = NULL, ...)

wf.combine(wf.obj, word.lists, matrix = 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.
`wfdf`	A word frequency data frame given instead of raw text.var and optional grouping.var. Basically converts a word frequency dataframe (wfdf) to a word frequency matrix (`wfm`). Default is NULL.
`output`	Output type (either `"proportion"` or `"percent"`).
`stopwords`	A vector of stop words to remove.
`char2space`	A vector of characters to be turned into spaces. If `char.keep` is NULL, `char2space` will activate this argument.

...	Other arguments supplied to strip.
digits	An integer indicating the number of decimal places (round) or significant digits (signif) to be used. Negative values are allowed.
margins	logical. If TRUE provides grouping.var and word variable totals.
word.lists	A list of character vectors of words to pass to wf.combine
matrix	logical. If TRUE returns the output as a wfm rather than a wfdf object.
wf.obj	A wfm or wfdf object.

Value

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.

wf.combine - returns a word frequency matrix (wfm) or dataframe (wfdf) with counts for the combined word.lists merged and remaining terms (else).

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.

Examples

```
## Not run:
#word frequency matrix (wfm) example:
with(DATA, wfm(state, list(sex, adult)))[1:15, ]
with(DATA, wfm(state, person))[1:15, ]

#insert double tilde ("~~") to keep phrases(i.e., first last name)
alts <- c(" fun", "I ")
state2 <- mgsub(alts, gsub("\\s", "~~", alts), DATA$state)
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, ]

#insert double tilde ("~~") to keep dual words (i.e., first last name)
alts <- c(" fun", "I ")
state2 <- mgsub(alts, gsub("\\s", "~~", alts), DATA$state)
with(DATA, wfdf(state2, list(sex, adult)))[1:18, ]

#wfm.expanded example:
z <- wfm(DATA$state, DATA$person)
wfm.expanded(z)[30:45, ] #two "you"s

#wf.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)

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", "your're"))
WL5 <- list(yous = c("you", "your", "your're"))
WL6 <- list(c("you", "your", "your're"))  #no name so will be called words 1
WL7 <- c("you", "your", "your're")

wf.combine(z, WL2) #Won't work not a raw frequency matrix
wf.combine(x, WL2) #Works (raw and no margins)
wf.combine(y, WL2) #Works (raw with margins)
wf.combine(y, c("you", "your", "your're"))
wf.combine(y, WL1)
wf.combine(y, WL3)
## wf.combine(y, WL4) #Error
wf.combine(y, WL5)
wf.combine(y, WL6)
wf.combine(y, WL7)

worlis <- c("you", "it", "it's", "no", "not", "we")
y <- wfdf(DATA$state, list(DATA$sex, DATA$adult), margins = TRUE)
z <- wf.combine(y, worlis, matrix = TRUE)

chisq.test(z)
chisq.test(wfm(wfdf = y))

## End(Not run)
```

`word.count` *Word Counts*

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

```
word.count(text.var, byrow = TRUE, missing = NA,
  digit.remove = TRUE, names = FALSE)

wc(text.var, byrow = TRUE, missing = NA,
  digit.remove = TRUE, names = FALSE)

character.count(text.var, byrow = TRUE, missing = NA,
  apostrophe.remove = TRUE, digit.remove = TRUE,
  count.space = FALSE)
```

```
character.table(text.var, grouping.var, percent = TRUE,
  prop.by.row = TRUE, zero.replace = 0, digits = 2, ...)

char.table(text.var, grouping.var, percent = TRUE,
  prop.by.row = TRUE, zero.replace = 0, digits = 2, ...)
```

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.
`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.
`prop.by.row`	logical. If TRUE applies proportional to the row. If FALSE applies by column.
`...`	Other arguments passed to `prop`.
`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.

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`

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))

# 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")
x$raw[, 1:20]
x$prop[, 1:8]
x$rnp[, 1:8]

## 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$adult))
dat2 <- colsplit2df(melt(dat$raw), keep.orig = TRUE)
head(dat2, 15)
dat3 <- dat2[rep(seq_len(dim(dat2)[1]), dat2[, 5]), -5]


ggplot(data = dat2, aes(y = variable, x = value, colour=sex)) +
    facet_grid(adult~.) +
    geom_line(size=1, aes(group =variable), colour = "black") +
    geom_point()

ggplot(data = dat3, aes(x = variable, fill = variable)) +
    geom_bar() +
    facet_grid(sex ~ adult, margins = TRUE) +
    theme(legend.position="none")

## End(Not run)
```

`word.network.plot` *Word Network Plot*

Description

A network plot of words. Shows the interconnected and supporting use of words between textual units containing key terms.

Usage

```
word.network.plot(text.var, grouping.var = NULL,
  target.words = NULL, stopwords = 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,
  char2space = "~~", ...)
```

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.
`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.
`log.labels`	logical. If TRUE uses a proportional log label for more readable labels. The formula is: `log(SUMS)/max(log(SUMS)))`. `label.size` adds more control over the label sizes.
`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.
`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

```
## Not run:
word.network.plot(text.var=DATA$state, grouping.var=DATA$person)
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.i., 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)

## 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 output 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.
`char2space`	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", "tru")
```

```
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
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$person),
    match.string = m)
out5 <- word_associate(raj.act.1$dialogue, list(raj.act.1$person),
    match.string = m)
out6 <- with(mraja1spl, word_associate(dialogue, list(fam.aff, sex),
     match.string = m))
names(out6)
lapply(out6$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")
)

out7 <- word_associate(DATA2$state2, DATA2$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"))

DATA2 <- qdap::DATA2

## 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

```
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. 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`	If vs.all.cut = TRUE this argument 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:

`word`	The words unique to that group
`freq`	The number of times that group used that word
`prop`	The proportion of that group's overall word use dedicated to that particular word

Examples

```
## 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_list   *Raw Word Lists/Frequency Counts*

Description

Transcript Apply Raw Word Lists and Frequency Counts by grouping variable(s).

Usage

```
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 stopwords removed
`rfswl`	reduced frequency stopword word list; same as fswl but truncated to n rows

Examples

```
## 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)
```

`word_stats` *Descriptive Word Statistics*

Description

Transcript apply descriptive word statistics.

Usage

```
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 output 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.

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
- sptot - 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.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 od number of digits to display; mostly internal use

Examples

```
## Not run:
word_stats(mraja1spl$dialogue, mraja1spl$person)
(desc_wrds <- with(mraja1spl, word_stats(dialogue, person, tot = tot)))
with(mraja1spl, word_stats(desc_wrds, person, tot = tot)) #speed boost
names(desc_wrds)
htruncdf(desc_wrds$ts, 15, 5)
htruncdf(desc_wrds$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)
with(mraja1spl, word_stats(dialogue, list(sex, died, fam.aff)))

## End(Not run)
```

Index