Package 'dummy'

January 8, 2013

Type Package

Title Tools for being a dummy

Version 0.1.0

Date 2012-11-26

Author Tyler Rinker

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

Description This is where I test things and don't worry about it all going bad.

Depends R (>= 2.15)

License GPL-2

Collate
'bracketX.R' 'common.R' 'bag.o.words.R' 'all_words.R''automated_readability_index.R' 'blank2NA.R' 'capitalizer.R
package.R'

R topics documented:

`abbreviations` *Small Abrreviations 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 abbraviations

`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 (.a, .c or .d) and generates an adjacency matrix for use with igraph

Usage

```
adjacency_matrix(matrix.obj)

adjmat(matrix.obj)
```

Arguments

`matrix.obj`	A matrix object, preferably, of the class "termco_d" or "termco_c" generated from `terco.a`, `termco.d` or `termco.c`.

Value

Generates an adjacency matrix

See Also

`dist`

Examples

```
## Not run:
wordLIST <- c(" montague", " capulet", " court", " marry")
(raj.termco <- with(raj.act.1, termco.a(dialogue, person,
    wordLIST, ignore.case = T)))
(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)
layout1 <- layout.auto(g)
plot(g, layout=layout1)

## 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 begining 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 containt he provided word chunk.

Note

Can not 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:
all_words(raj$dialogue, begins.with="re")
all_words(raj$dialogue, "q")
all_words(raj$dialogue, contains="conc")
all_words(raj$dialogue)

## End(Not run)
```

---

automated_readability_index
Readabilitiy 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 word list 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 `endf`.
`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.

Note

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:
with(rajSPLIT, automated_readability_index(dialogue, list(person, act)))
with(rajSPLIT, automated_readability_index(dialogue, list(sex, fam.aff)))

with(rajSPLIT, coleman_liau(dialogue, list(person, act)))
with(rajSPLIT, coleman_liau(dialogue, list(sex, fam.aff)))

with(rajSPLIT, SMOG(dialogue, list(person, act)))
with(rajSPLIT, SMOG(dialogue, list(sex, fam.aff)))

with(rajSPLIT, flesch_kincaid(dialogue, list(person, act)))
with(rajSPLIT, flesch_kincaid(dialogue, list(sex, fam.aff)))
```

```
(x <- with(rajSPLIT, fry(dialogue, list(sex, fam.aff))))
with(rajSPLIT, fry(dialogue, list(sex, fam.aff), labels = "click"))

with(rajSPLIT, linsear_write(dialogue, list(person, act)))
with(rajSPLIT, linsear_write(dialogue, list(sex, fam.aff)))

## 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 qda recognized endmarks (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. `"."`, `"!"`, `"?"`, `"*"`, `"-"`).

Warning

fdf

Examples

```
## Not run:
bag.o.words(DATA$state)
by(DATA$state, DATA$person, bag.o.words)
lapply(DATA$state,  bag.o.words)
bag.o.words("I'm going home!", apostrophe.remove = FALSE)

DATA
```

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

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

## End(Not run)
```

blank2NA *Replace Blanks in Data Frame*

Description

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

Usage

```
blank2NA(dataframe, missing = NA)
```

Arguments

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

Value

Returns a dataframe with blank spaces replaced.

See Also

unblanker

Examples

```
## Not run:
dat <- data.frame(matrix(sample(c(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.

Usage

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

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

Arguments

`text.var` The text variable

`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.

`missing` Value to assign to empty cells.

`names` logical. If TRUE the sentences are given as the names of the counts.

`with` logical. If TRUE returns the brackets and the bracketted text.

Value

`bracketX` - returns a vector of text with brackets removed.

`bracketXtract` - returns a list of vectors of bracketed text.

Author(s)

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

References

http://stackoverflow.com/questions/8621066/remove-text-inside-brackets-parens-and-or-braces

Examples

```
## Not run:
examp2 <- examp2 <- 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.[interupting]",
    "Yep it’s awesome {reading}.", "Agreed. {is so much fun}")), .Names =
    c("person", "text"), row.names = c(NA, -4L), class = "data.frame")

examp1
bracketX(examp2$text, ’square’)
bracketX(examp2$text, ’curly’)
bracketX(examp2$text)

examp2
bracketXtract(examp2$text, 'square')
bracketXtract(examp2$text, 'curly')
bracketXtract(examp2$text)
bracketXtract(examp2$text, with = TRUE)

paste2(bracketXtract(examp2$text, 'curly'), " ")

## 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 *Capitalizes 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.
`no.apostrophe`	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

Pre process 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
`combin.code.list`	A list of named character vertors 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 excludsion 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")
nots <- list(notAABB=qcv(AA, BB), notAACC=qcv(AA, CC), notBBCC=qcv(BB, CC))
z <- cm_code.blank(z, nots, "time", overlap=0)
z <- cm_code.blank(z, list(atleastAABBCC=qcv(AA, BB, CC)), "time", overlap=1)
z <- cm_code.blank(z, list(AACC=qcv(AA, CC)), "time", overlap=FALSE)  #combined
cm_code.blank(z, list(AACCnoAA=qcv(AACC, AA)), "time", overlap=1)     #remove the AA part

#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)
cm_code.blank(dat, list(P=qcv(A, B), Q=qcv(B, C), R=qcv(A, B, C)),
    "variable", overlap=TRUE)

## End(Not run)
```

cm_code.combine *Combine Codes*

Description

Combine all occurences 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 vertors 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")
combines <- list(AB=qcv(AA, BB), ABC=qcv(AA, BB, CC))
cm_code.combine(x, list(AB=qcv(AA, BB)))
cm_code.combine(x, list(ALL=qcv(AA, BB, CC)))
cm_code.combine(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)
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 occurences of n codes excluding the nth code. e.g. 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 vise 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 vertors 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)
cm_code.exclude(dat, list(P=qcv(A, B), Q=qcv(B, C), R=qcv(A, B, C)), "variable")

## End(Not run)
```

cm_code.overlap *Find Co-occurrence Between Codes*

Description

Combine co-occurances 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 vertors of at least two code column names to aggregate co-occurences.

`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")
combines <- list(AB=qcv(AA, BB), ABC=qcv(AA, BB, CC))
cm_code.overlap(x, list(AB=qcv(AA, BB)))
cm_code.overlap(x, list(ALL=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)
cm_code.overlap(dat, list(P=qcv(A, B), Q=qcv(B, C), R=qcv(A, B, C)), "variable")

## End(Not run)
```

cm_code.transform *Transform Codes*

Description

Transform co-occurences 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 vertors of at least two code column names to aggregate co-occurences.
`combine.code.list`	A list of named character vertors of at least two code column names to combine
`exclude.code.list`	A list of named character vertors 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)
z <- cm_range2long(foo, foo2, v.name="time")
overlaps <- list(AB=qcv(AA, BB), ABC=qcv(AA, BB, CC))
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)))
cm_code.transform(z, overlaps, rm.var="time")
cm_code.transform(z, overlaps,
   exclude.code.list=list(AABB_no_CC = qcv(AA, BB, CC)), 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)
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")

## 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 vertors 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.

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='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 of a Code Matrix*

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.
`codes`	Optional vector of codes.
`transform`	logical. If TRUE the words are located across the top of dataframe.

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_df2long`

Examples

```
## Not run:
codes <- qcv(dc, sf, wes, pol, rejk, lk, azx, mmm)
X <- cm_df.temp(DATA, "state", codes)
cm_df.fill(X, 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))
cm_df.fill(X, list(sf=c(4, 6:9, 11), dc=c(1:3, 5), azx=1:30, mmm=5))

## 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 dtaframe name.
`transpose`	logical. If TRUE transposes the dataframe so that the text is across the top.
`strip`	logical. If TRUE all punctuation is removed.

Value

Generates a dataframe, and optional csv file, of individual words while maintaing demgraphic 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.fill

Examples

```
## Not run:
codes <- qcv(dc, sf, wes, pol, rejk, lk, azx, mmm)
cm_df.temp(DATA, "state", codes)
cm_df.temp(DATA, "state", codes, transpose = TRUE)
head(cm_df.temp(raj.act.1, "dialogue", codes))
cm_df.temp(raj.act.1, "dialogue", codes, transpose = TRUE)[, 1:9]

## End(Not run)
```

cm_df.transcript *Transcript With Word Number*

Description

Out put 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	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).

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 out put from this file. 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

See Also as `cm_df2long` See Also as `cm_df.temp`

Examples

```
## Not run:
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` 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 `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 TURE 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 varaibles.

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)
cm_df2long(x1,  code.vars = codes)
x1[, 7:14] <- lapply(7:14,  function(i) sample(0:1, nrow(x1), TRUE))
cm_df2long(x1,  code.vars = codes)

## End(Not run)
```

cm_distance *Distance Matrix Between Codes*

Description

Generate distance measures to assertain a mean distance emasure 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 ebtween x and y given that x must procede 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 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=T))
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='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)))

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

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

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'),
    BB = qcv(terms='1:2, 3:10, 19'),
    CC = qcv(terms='1:3, 5:6')
)

foo2  <- list(
    AA = qcv(terms='4'),
    BB = qcv(terms='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")
cm_long2dummy(z, "time")

## 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)
```

Arguments

`codes`	List of codes.
`file`	A connection, or a character string naming the file to print to (.txt 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_time.temp`

Examples

```
## Not run:
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`.
`star.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 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`

Examples

```
## Not run:
foo <- list(
    AA = qcv(terms='1'),
    BB = qcv(terms='1:2, 3:10, 19'),
    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")
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 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”)
x <- list(
    transcript_time_span = qcv(terms=’00:00 - 1:12:00’),
    A = qcv(terms=’2.40:3.00, 5.01, 6.62:7.00, 9.00’),
    B = qcv(terms=’2.40, 3.01:3.02, 5.01, 6.62:7.00, 9.00, 1.12.00:1.19.01’),
    C = qcv(terms=’2.40:3.00, 5.01, 6.62:7.00, 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`.

`star.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 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`

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 (internal use within colsplit2df).

Value

Returns a dataframe of split columns.

See Also

colsplit2df, paste2

Examples

```
## Not run:
(foo <- paste2(CO2[, 1:3]))
colSplit(foo)
(bar <- paste2(mtcars[, 1:3], sep="|"))
colSplit(bar, col.sep = "|")

## End(Not run)
```

colsplit2df *Wrapper for colSplit that Returns a Dataframe*

Description

Wrapper for `colSplit` that returns a dataframe.

Usage

```
colsplit2df(dataframe, splitcol = 1, new.names = NULL,
  sep = ".", 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.
`orig.keep`	logical. If TRUE the original pasted column will be retained as well.

Value

Returns a dataframe with the pasted column cplit into new columns.

See Also

`colSplit`, `paste2`

Examples

```
## Not run:
CO2$`Plant&Type&Treatment` <- paste2(CO2[, 1:3])
CO2 <- CO2[, -c(1:3)]
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

## End(Not run)
```

common *Find Common Words Between Groups*

Description

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

Usage

```
common(x, ...)

## Default S3 method:
common(..., overlap = "all",
  equal.or = "equal")

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

Arguments

`word.list`	A list of names chacter vectors.
`overlap`	Minimum/exact amount of overlap.
`equal.or`	A character vector of c("`equal`", "`greater`", "`more`", "`less`").
`...`	In liu 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`.

NULL

NULL

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)
```

convert *Convert Seconds to h:m:s*

Description

Converts a vector of seconds to h:m:s

Usage

```
convert(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.

Examples

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

## End(Not run)
```

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

delete *Easy File Handling*

Description

`delete` - Deletes files and directories.

`folder` - Create a folder/directory.

Usage

```
delete(file = NULL)

folder(folder.name = NULL)
```

Arguments

`file`	The name of the file in the working directory or the path to the file to be deleted. If NULL provides a menu of files from the working directory.
`folder.name`	The name of the folder to be created. Default NULL creates a file in the working directory with the creation date and time stamp.

Value

`delete` permanently removes a file/directory.

`folder` creates a folder/directory.

See Also

unlink, file.remove, dir.create

Examples

```
## Not run:
(x <- folder("DELETE.ME"))
which(dir() == "DELETE.ME")
delete("DELETE.ME")
which(dir() == "DELETE.ME")

## End(Not run)
```

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

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,
  digits = 3)
```

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
`upper`	logical. If True returns the upper triangle of the matrix
`p`	The power of the Minkowski distance
`digits`	integer indicating the number of decimal places (round) or significant digits (signif) to be used. Negative values are allowed

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 dsitribution 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 cumalative 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)
distTab(mtcars)
distTab(mtcars, 4)

wdst <- with(mraja1spl, word_stats(dialogue, list(sex, fam.aff, died)))
distTab(wdst$gts)

## 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`

`env.syl` *Syllable Lookup Environment*

Description

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

Format

A environment with

Details

For internal use.

References

UCI Machine Learning Repository website

`increase.amplification.words` *Amplifying Words*

Description

A dataset containing a vector of words that amplifly 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/`

`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`

`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 Intellgience.

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

`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`

`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 Intellgience.

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

`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*

Description

Prints an adjacency.matrix.

Usage

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

Arguments

`x`	The adjacency.matrix object
`...`	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

qdap *qdap: Quantitative Discourse Analysis Package*

Description

This package automates many of the tasks associated with quantitative discourse analysis oftranscripts containing discourse including frequency counts of sentence types, words, sentence, turns of talk, syllable counts and other assorted analyysis 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 provides the user with a more efficient and targeted analysis.

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

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 makem up about one-half of all printed material in English. The first 300 makem 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 makem up about one-half of all printed material in English. The first 300 makem 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 makem up about one-half of all printed material in English. The first 300 makem up about 65% of all printed material in English."

References

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

Index