Package ‘statisfactory’

March 7, 2022

Type Package

Title Statistical and Geometrical Tools

Version 1.0.1

Date 2022-03-02

Maintainer Adam B. Smith <adam.smith@mobot.org>

Description A collection of statistical and geometrical tools including the aligned rank transform (ART; Higgins et al. 1990 <doi:10.4148/2475-7772.1443>; Peterson 2002 <doi:10.22237/jmasm/1020255240>; Wobbrock et al. 2011 <doi:10.1145/1978942.1978963>), 2-D histograms and histograms with overlapping bins, a function for making all possible formulae within a set of constraints, amongst others.

Depends

Imports omnibus, rlist

Suggests

License GPL (>= 3)

LazyLoad yes

URL https://github.com/adamlilith/statisfactory

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation no

Author Adam B. Smith [aut, cre] (<https://orcid.org/0000-0002-6420-1659>)

Repository CRAN

Date/Publication 2022-03-07 20:30:02 UTC

R topics documented:

art 2
countConnected 4

euclid 5
geoMean 6
hist2d 7
histOverlap 8
invLogitAdj 9
logitAdj 10
makeFormulae 11
mmode 13
nagelR2 14
rankMulti 15
rmsd 15
sampleAcross 17
sampleStrat 18
se 20

Index **21**

`art` *Aligned rank transform of non-parametric data for further analysis using ANOVA*

Description

This function performs the aligned rank transforms on non-parametric data which is useful for further analysis using parametric techniques like ANOVA.

Usage

```
art(
  x,
  response = names(x)[1],
  factors = names(x)[2:ncol(x)],
  subject = NULL,
  fun = function(x) mean(x, na.rm = TRUE),
  verbose = FALSE
)
```

Arguments

`x`	Data frame.
`response`	Character. Names of column of `x` that has response variable (default is to use the first column).
`factors`	Character list. Names of columns of `x` used to define factors and levels (default is to use all columns except for the first).
`subject`	`NULL` or character. Name of column in `x` that has the subject variable. If `NULL` then this is ignored. If specified, residuals are calculated for each cell defined by factors, *not* by subject and factors, but aligning is done using both factors and subject.

`fun` Function. Function used to calculate cell centering statistic (the default is to use: `mean` with `na.rm=TRUE`). The function can be any that handles a list of one or more elements.

`verbose` Logical. If TRUE then display progress.

Details

The function successfully re-creates rankings given by **ARTool** (Wobbrock et al. 2011) of data in Higgins et al. (1990) for data with 2 and 3 factors. If `response` is ranks and the set of ranks in each cell is the same (e.g., each cell has ranks 1, 2, and 3, but not necessarily in that order), then all values will be equal across the different ART variables. This occurs because the center of each cell (e.g., the mean) is the same as the grand mean, so the aligned values are simply the residuals. An ANOVA on this data yields no variance across cells, so the F tests are invalid.

Value

Data frame.

References

Higgins, J.J., Blair, R.C., and Tashtoush, S. 1990. The aligned rank transform procedure. *Proceedings of the Conference on Applied Statistics in Agriculture.* Manhattan, Kansas: Kansas State University, pp. 185-195. doi: 10.4148/24757772.1443

Peterson, K. 2002. Six modifications of the aligned rank transform test for interaction. *Journal of Modern Applied Statistical Methods* 1:100-109. doi: 10.22237/jmasm/1020255240

Wobbrock, J.O., Findlater, L., Gergle, D., and Higgins, J.J. 2011. The aligned rank transform for nonparametric factorial analysis using only ANOVA procedures. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2011).* Vancouver, British Columbia (May 7-12, 2011). New York: ACM Press, pp. 143-146. doi: 10.1145/1978942.1978963.

Examples

```
x <- data.frame(
   subject=c('a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c'),
   factor1=c('up', 'up', 'up', 'up', 'up', 'up', 'down', 'down', 'down', 'down',
      'down', 'down'),
   factor2=c('high', 'med', 'low', 'high', 'med', 'low', 'high', 'med', 'low', 'high',
      'med', 'low'),
   response=c(1, 17, 1, 1, 0, 4, 5, 6, 3, 7, 100, 70)
)
art(x=x, response='response', factors=c('factor1', 'factor2'))
```

`countConnected` *Count number of contiguous "blocks" of cells*

Description

This function calculates the number of objects formed by one or more adjacent cells that touch on their edges (i.e., not just at a corner). One way to solve this (inefficiently) is using a "ink-spreading" algorithm that accumulates adjacent cells until all are accounted for, then counts this as a single component. This function uses an efficient solution based on the Euler characteristic.

Usage

```
countConnected(x, count = 1)
```

Arguments

`x`	Matrix
`count`	Value to count as a "presence" in the matrix. All other values will be assumed to be not part of a component.

Details

Inspired by an answer by Alon Amit to the question on Quora, "*What are some programming problems that look hard at a first glance but are actually easy?*".

Value

An integer (the number of connected, non-conterminous components).

Examples

```
v <- c(
1, 1, 0, 1,
1, 1, 0, 0,
1, 0, 0, 0,
0, 0, 0, 1,
0, 0, 1, 1,
1, 0, 0, 0,
0, 0, 0, 0)

x <- matrix(v, ncol=4, byrow=TRUE)
x

countConnected(x)

## Not run:
# will break because of connection at a vertex
v <- c(
```

```
1, 1, 0, 1,
1, 1, 0, 0,
1, 0, 0, 0,
0, 0, 0, 1,
0, 0, 1, 1,
1, 0, 0, 0,
0, 1, 0, 0)

x <- matrix(v, ncol=4, byrow=TRUE)
x

countConnected(x)

## End(Not run)
```

---

euclid *Euclidean distance*

Description

Euclidian distance in one or more dimensions.

Usage

```
euclid(a, b, na.rm = FALSE)
```

Arguments

a	Numeric vector.
b	Numeric vector of same length as a.
na.rm	Logical. If TRUE, calculation ignores NA's in a and/or b.

Value

Numeric.

Examples

```
euclid(0, 5)
euclid(c(0, 0), c(1, 1))
euclid(c(0, 0, 0), c(1, 1, 1))
```

geoMean — *Geometric mean*

Description

Geometric mean, with optional removal of NA's and propagation of zeros.

Usage

```
geoMean(x, prop0 = FALSE, na.rm = TRUE)
```

Arguments

x	Numeric list.
prop0	Logical, if FALSE (default) then if any value in x equals 0 then the output will be zero. If TRUE, then zero values will be removed before calculation of the geometric mean.
na.rm	Logical, if TRUE then remove NA values first.

Details

Adapted from Paul McMurdie on StackOverflow.

Value

Numeric.

Examples

```
x <- seq(0.01, 1, by=0.01)
mean(x)
geoMean(x)
x <- seq(0, 1, by=0.01)
mean(x)
geoMean(x)
geoMean(x, prop0=TRUE)
```

`hist2d` *Two-dimensional histogram*

Description

Two-dimensional histogram

Usage

```
hist2d(x, breaks1 = "Sturges", breaks2 = "Sturges", right = TRUE, ...)
```

Arguments

`x`	Data frame or matrix with at least two columns. Only first two columns are used to tally frequencies.
`breaks1`	One of the following describing how breaks for the first variable are calculated: • Numeric vector: Breakpoints for bins for the first variable. • Single integer: The number of bins into which to tally values of the first variable. • Function: To compute the vector of breakpoints. • Function: To compute the number of cells. Used as a suggestion only (see `hist`). • Character: The name of a function to compute the number of cells (see the *Details* section in `hist`). Used as a suggestion only (see `hist`).
`breaks2`	Same as `breaks1` but for the second variable.
`right`	Logical, if `TRUE` (default) then use left-open and right-closed intervals.
`...`	Arguments to pass to `hist`.

Value

Object of class `matrix` and `histogram2d`. Columns pertain to bins of `x1` and rows `x2`. Column names and row names are mid-points of bins.

See Also

`hist`

Examples

```
x1 <- rnorm(1000)
x2 <- 0.5 * x1 * rnorm(1000)
x <- data.frame(x1=x1, x2=x2)
hist2d(x)
```

`histOverlap` *Count number of values in overlapping bins*

Description

Count number of values in overlapping bins

Usage

```
histOverlap(x, breaks, right = TRUE, graph = TRUE, indices = FALSE)
```

Arguments

`x` Numeric values.

`breaks` One integer, three numeric values, or a matrix or data frame with at least two columns:

- Single integer: The number of overlapping bins into which to enumerate values of `x`. The range of `x` covered by the bins bins will extend from the least value minus 2.5 percent of the range to the largest value plus 2.5 percent of the range.
- Three numeric values: The first two values are the range of covered by the bins (least and greatest). The third value is the number of bins.
- Matrix or data frame with at least two columns. Each row corresponds to a different bin. The first column represents the minimum values of each bin and the second column the maximum value. Subsequent columns are ignored. Note that by using this option arbitrary bins can be used–they need not overlap or even be continuous in coverage.

`right` Logical, if `TRUE` (default), then use left-open and right-closed intervals.

`graph` Logical, if `TRUE` (default), then plot frequencies.

`indices` Logical, if `TRUE`, then the output will have an attribute which is a list item with one element per bin in the output, with the indices of `x` that fall in each bin. Default is `FALSE`.

Value

Matrix

See Also

`hist`

Examples

```
set.seed(123)
x <- rnorm(1000)
histOverlap(x, breaks=10, graph=TRUE)
histOverlap(x, breaks=c(0, 1, 10), graph=TRUE)
mat <- matrix(c(seq(0, 1, by=0.1), seq(0.3, 1.3, by=0.1)), ncol=2)
histOverlap(x, breaks=mat, graph=TRUE)
histOverlap(x, breaks=mat, indices=TRUE)
```

---

invLogitAdj — *Inverse logit is robust to cases that equal 0 or 1*

Description

This function is the inverse of `logitAdj`. That function calculates the logit of values but is robust to cases where the operand is 0 or 1. The adjusted inverse logit is equal to `(base^x + epsilon * base^x -epsilon) / (base^x + 1)`.

Usage

```
invLogitAdj(x, epsilon = 0.01, base = 10, auto = FALSE)
```

Arguments

`x`	Numeric vector.
`epsilon`	Value or character. If a numeric value (typically ~0.01 or smaller), then this is added/subtracted from `x` to ensure log of 0 or 1 is not taken. If equal to `'auto'` then the value of `epsilon` is taken from the attributes of `x`. If `x` has no such attribute, a warning is given and a value of 0.01 is used.
`base`	Base of logarithm. Use `base=exp(1)` for base `e`.
`auto`	If `TRUE` then if the attributes of `x` have slots named `epsilon` and `base` then use these instead of the user-supplied values of `epsilon` and `base`. If they do not appear as attributes of `x` but `auto` is `TRUE` then the function prints warnings and uses 0.01 and 10, respectively. If `FALSE` (default) then use the user-supplied values of `epsilon` and `base`.

Value

Numeric.

See Also

`logitAdj`

Examples

```
x <- seq(0, 1, by=0.1)
y <- logitAdj(x)
xx <- invLogitAdj(y, auto = TRUE)
```

---

logitAdj *A logit() function robust to values that equal 0 or 1*

Description

This function returns the logit value (`log(x / (1 -x))`) where a small value can be added to `x` to avoid problems of calculating the log when `x` equals 0 or 1.

Usage

```
logitAdj(x, epsilon = 0.01, base = 10)
```

Arguments

`x`	Numeric vector.
`epsilon`	Value to add/subtract from x to ensure log of 0 or 1 is not taken (usually a small number). If `NULL`, then the smallest value of any `x` > 0 and `1 -x` for all `x` < 1 is used.
`base`	Base of logarithm.

Value

Numeric equal to `log((x + epsilon)/(1 -x + epsilon),base=base)`.

See Also

`invLogitAdj`

Examples

```
set.seed(123)
x <- seq(0, 1, by=0.01)
logitAdj(x)
logitAdj(x, 0.001)
invLogitAdj(x, 0.001)
invLogitAdj(x, 0.001)
invLogitAdj(x, auto = TRUE)
```

`makeFormulae` *Make all possible formula*

Description

This functions creates a list of formulae that contain all possible linear, quadratic, and two-way interaction terms from individual terms in an object of class `formula`. The formulae respect marginality conditions (i.e., they will always include lower-order terms if higher-order terms are included in a formula). Note that if there are more than several terms (i.e., >=3) and interactions and/or quadratic terms are desired, then formula generation may take a long time.

Usage

```
makeFormulae(
  formula,
  intercept = TRUE,
  interceptOnly = TRUE,
  linearOnly = TRUE,
  quad = TRUE,
  ia = TRUE,
  verboten = FALSE,
  verbotenCombos = NULL,
  minTerms = NULL,
  maxTerms = NULL,
  returnFx = stats::as.formula,
  verbose = FALSE
)
```

Arguments

`formula`	A `formula` object with *just* linear terms.
`intercept`	Logical, if `TRUE` (default) then all models include an intercept. If `FALSE` then then formula will specify that regression occurs through the origin (e.g., `y ~ -1 + etc.`)
`interceptOnly`	Logical, if `TRUE` then an intercept-only model is included in final set.
`linearOnly`	Logical, if `TRUE` (default) then models with only linear terms are included in final set (plus other kinds of models if desired).
`quad`	Logical, if `TRUE` (default), then include quadratic terms.
`ia`	Logical, if `TRUE` (default), then include 2-way interaction terms.
`verboten`	Character list of terms that should not appear in the models. Ignored if `NULL` (default). Note that using this argument only makes sense if interaction or quadratic terms are specified (if you don't a particular term to appear anywhere in the model it will be faster to remove it from `formula`).
`verbotenCombos`	List of lists, used to specify specific combinations of terms that should not occur together. See section *Details* below. Ignored if `NULL` (default).

`minTerms` Either a positive integer representing the minimum number of terms required to be in a model, *or* `NULL` (default) in which case the smallest model can have just one term.

`maxTerms` Either a positive integer representing the maximum number of terms allowed to be in a model, *or* `NULL` (default) in which case there is no practical limit on the number of terms in a model.

`returnFx` Function used to generate the class of the output objects. Sensible functions in include `as.formula` (default) or `as.character`.

`verbose` Logical, if `TRUE` then display progress. Default is `FALSE`.

Details

The argument `verbotenCombos` can be used to specify variables or terms that should not occur in the same formula. The argument `verbotenCombos` is composed of a list of lists. Each sublist comprises names of two variables or terms stated as characters followed by two logical values (`TRUE`/`FALSE`). The second variable/term is removed from the model if the first is in the model. If the first logical value is `TRUE` then the second variable/term is removed if the first variable appears alone in the formula (e.g., not in an interaction with another variable). If the first logical value is `FALSE` then the second variable/term is removed if the first variable/term appears in any term (e.g., as an interaction with another term). Examples:

- `verbotenCombos=list(list('x1','x2',TRUE,TRUE))`: Removes `x2` if `x1` occurs in the model as a linear term.
- `verbotenCombos=list(list('x1','x2',FALSE,TRUE))`: Removes the linear term `x2` if `x1` occurrs in *any* term in the model.
- `verbotenCombos=list(list('x1','x2',TRUE,FALSE))`: Removes *any* term with `x2` if the linear term `x1` occurrs in the model.
- `verbotenCombos=list(list('x1','x2',FALSE,FALSE))`: Removes any term with `x2` if any term has `x1`.

Quadratic terms and interaction terms can also be used, so:

- `verbotenCombos=list(list('x1','x1:x2',TRUE,TRUE))`: Removes `x1:x2` if `x1` were in the model.
- `verbotenCombos=list(list('x1','I(x2^2)',TRUE,TRUE))`: Removes `I(x2^2)` if `x1` occurs in the model.

Note that inexact matching can remove terms incorrectly if inexact matches exist between names of terms or variables. For example, if using an inexact match, then `verbotenCombos(list('x1','x2',FALSE,FALSE))` will find any term that has an `x1` (e.g., `x11`) and if it exists, remove any term with an `x2` (e.g., `x25`). Note that reciprocally removing predictors makes little sense since, for example `list(list('x1','x2',FALSE,FALSE),lis` removes all formulae with `x2` if `x1` appears then tries to find any models with `x2` that have `x1` (of which there will be none after the first set is removed).

Value

A vector of formulae.

Examples

```
makeFormulae(y ~ x1 + x2 + x3, maxTerms=3)
makeFormulae(y ~ x1 + x2 + x3, ia=FALSE, maxTerms=3)
verboten <- c('x1:x2', 'I(x1^2)')
makeFormulae(y ~ x1 + x2 + x3, verboten=verboten, maxTerms=3)

makeFormulae(y ~ x1 + x2 + x3, maxTerms=3)
verbotenCombos <- list(list('x1', 'x2', TRUE, TRUE))
makeFormulae(y ~ x1 + x2 + x3, verbotenCombos=verbotenCombos, maxTerms=3)
```

---

mmode *Calculate the mode of numeric, character, or factor data*

Description

Calculate the mode of numeric, character, or factor data

Usage

```
mmode(x, na.rm = FALSE)
```

Arguments

x	Numeric, character, or factor vector.
na.rm	Logical. If TRUE then remove NAs first. Otherwise fail.

Value

Numeric, character, or factor value.

Examples

```
mmode(round(10 * rnorm(1000, 2)))
mmode(c('a', 'b', 'b', 'b', 'c', 'd', 'd'))
```

nagelR2 *Nagelkerge's / Craig & Uhler's R2*

Description

Nagelkerge's / Craig & Uhler's R2

Usage

```
nagelR2(likeNull, likeFull, n)
```

Arguments

likeNull	Likelihood (not log-likelihood) of the null model or an object of class logLik with log-likelihood of the null model (usually an intercept-only model).
likeFull	Likelihood (not log-likelihood) of the "full" model or an object of class logLik with log-likelihood of the "full" model (usually a model with covariates).
n	Sample size.

Value

Numeric.

Examples

```
# create data
x <- 1:100
y <- 2 + 1.7 * x + rnorm(100, 0, 30)

# models
nullModel <- lm(y ~ 1)
fullModel <- lm(y ~ x)

# plot
plot(x, y)
abline(nullModel, col='red')
abline(fullModel, col='blue')
legend('bottomright', legend=c('Null', 'Full'), lwd=1, col=c('red', 'blue'))

# R2
likeNull <- exp(as.numeric(logLik(nullModel)))
likeFull <- exp(as.numeric(logLik(fullModel)))
nagelR2(likeNull, likeFull, 100)
```

rankMulti — *A multivariate adaptation of the rank() function*

Description

This function ranks values in a data frame or matrix by more than one field, with ties in one field broken by subsequent fields.

Usage

```
rankMulti(x, cols = 1:ncol(x), ...)
```

Arguments

`x`	Data frame or matrix.
`cols`	Names or indices of columns by which to rank, with first one gaining preference over the second, second over the third, etc.
`...`	Arguments to pass to `rank`. Note that if the `ties.method` argument is used the options `'first'` or `'random'` will rank by the first column uniquely such that there are no ties for subsequent columns to break.

Value

Numeric vector of ranks.

Examples

```
x <- data.frame(x1=c('a', 'b', 'b', 'c', 'a', 'a'), x2=c(11, 2, 1, NA, 10, 11))
rankMulti(x)
rankMulti(x, c('x2', 'x1'))
```

rmsd — *Root-mean-square deviation (error)*

Description

Calculate the root-mean-square deviation (`sqrt(mean((x1 -x2)^2))`). If non-constant weights `w` are supplied, then the calculation is `sqrt(sum(w * (x1 -x2)^2) / sum(w))`. Alternatively, `w` can be a function, in which case the returned value is equal to `sqrt(mean(w((x1 -x2)^2)))`.

Usage

```
rmsd(x1, x2, w = NULL, na.rm = FALSE)
```

Arguments

x1	Numeric vector, matrix, or data frame.
x2	Numeric vector the same length as x1, or a matrix or data frame the same dimensions as x1.
w	Weights or a function defining weights. If x1 and x2 are vectors, this can be a numeric vector the same length as x1 or x2. If x1 and x2 are matrices or data frames then this can be either a matrix or data frame with the same dimensions as x1 and x2. Alternatively, this can be a function to define weights. The function will be applied to each value of (x1 -x2)^2. The default value of NULL assigns each pair of values in x1 and x2 equal weight.
na.rm	Logical, if TRUE then remove any elements in x1 *and* x2 where either x1 or x2 is NA. Default is FALSE, in which case any NA returns NA.

Value

Numeric.

Examples

```
set.seed(123)
# numeric vectors
x1 <- 1:20
x2 <- 1:20 + rnorm(20)
rmsd(x1, x2)
x1[1] <- NA
rmsd(x1, x2)
rmsd(x1, x2, na.rm=TRUE)

# matrices
x1 <- matrix(1:20, ncol=5)
x2 <- matrix(1:20 + rnorm(20), ncol=5)
rmsd(x1, x2)
x1[1, 1] <- NA
rmsd(x1, x2)
rmsd(x1, x2, na.rm=TRUE)

# weights as values
x1 <- matrix(1:20, ncol=5)
x2 <- matrix(1:20 + rnorm(20, 0, 2), ncol=5)
w <- matrix(1:20, ncol=5)
rmsd(x1, x2)
rmsd(x1, x2, w)

# weights as a function
x1 <- matrix(1:20, ncol=5)
x2 <- matrix(20:1, ncol=5)
w <- function(x) 1 - exp(-x)
rmsd(x1, x2)
rmsd(x1, x2, w)
```

sampleAcross	*Permute values across two vectors or columns in two data frames or matrices*

Description

This function permutes values across two or more vectors or columns between two or more data frames or matrices. If vectors, then all values are swapped randomly and the output is a list object with vectors of the same length. If data frames or matrices, then values in selected columns are swapped across the data frames or matrices and the output is a list object with data frames or matrices of the same dimension as the originals.

Usage

```
sampleAcross(..., by = NULL, replace = FALSE)
```

Arguments

`...`	One or more vectors, data frames, or matrices (all objects must be the same class).
`by`	Character list or list of integers. Names of columns or column numbers to permute (only used if `...` is data frames or matrices). If left as `NULL` (default) the all columns are permuted.
`replace`	Logical. If `TRUE` then sample with replacement. If `FALSE` (default) then sample without replacement.

Value

A list object with same number of elements as in `...` with the original dimensions. The order is the same as in `...` (e.g., so if the call is like `sampleAcross(a,b,c)` then the output will be a list with permuted versions of `a`, `b`, and `c` in that order).

See Also

`sample`

Examples

```
x1 <- 1:5
x2 <- 6:10
x3 <- 50:60
sampleAcross(x1, x2, x3)
sampleAcross(x1, x2, x3, replace=TRUE)

a <- data.frame(x=1:10, y=letters[1:10])
b <- data.frame(x=11:20, y=letters[11:20])
sampleAcross(a, b, by='y')
```

```
sampleAcross(a, b)
```

`sampleStrat` *Stratified randomization*

Description

This function scrambles values of a given column of a data frame in a stratified manner with respect to one or more other "covariate" columns. The covariate columns can be specified, as well as the width of the range of each covariate around each focal value from which to sample candidates for swapping.

Usage

```
sampleStrat(
  x,
  col,
  w = function(x) stats::sd(x, na.rm = TRUE)/(max(x, na.rm = TRUE) - min(x, na.rm =
    TRUE)),
  d = 0.1,
  by = "all",
  permuteBy = TRUE
)
```

Arguments

`x`	Data frame containing at least two columns, one with numeric values and at least one more with numeric or factor values.
`col`	Character or integer, name or number of column in `x` to swap values.
`w`	Function or numeric value >0, sets window size of *non-factor* covariates as a *proportion* of their range. If using a function it must work on a list of values. It can be helpful if this function accepts the argument `'na.rm=T'` to avoid problems with `NA`s in the column specified by `col`. The default is the standard deviation divided by the range. This reduces the correlation between erstwhile perfectly correlated variables to ~0.80 (on average). Ignored for covariates that are factors.
`d`	Numeric > 0, if no swappable value is found within `w * (max(col) -min(col))`, then `w` is expanded by `1 + d` iteratively until a value is found. Ignored for covariates that are factors.
`by`	Character vector or integers. Name(s) or columns numbers of covariates by which to stratify the target column. Can also specify `'all'` (default) to stratify by all columns with a numeric/integer/factor class except the target column.
`permuteBy`	Logical, if `TRUE` then in each step scramble the order of values in `by`. If `FALSE` then strata are considered for each covariate in teh order listed by `by`. This argument has no effect if `by` has just one value.

Details

The script starts by randomly selecting a value v_i from the target column. It then finds the value of covariate c_j, that is associated with v_i. Call the particular value of c_j associated with v_i c_j:i. If c_j is a continuous variable it then finds all values c_{v} that fall within c_j:i -w,c_j:i + w where w is a proportion of the range of c_j.
The function then randomly selects a value of v_k from those associated with this range of c_j and swaps v_i with this value. Depending on the random number generator, v_i can = v_k and in fact be the same value. If no values of c_j other than the one associated with v_i are found within this range, then the window is expanded iteratively by a factor of w * (1 + d) until at least one more values that have yet to be swapped have been found. The procedure then finds a window around v_k as described above (or randomly selects a new v_i if v_i was v_k) and continues. If there is an odd number of values then the last value is kept as is (not scrambled). If c_j is a categorical variable (a factor), then the script finds all values of of v in same factor level as v_i. Swaps of v occur within this level of c_j. However, if there are <2 of values in the level (including the value associated with v_i), then the script looks to the next factor level. The "next" is taken to be the factor level with the least difference between v_i and the average of values of v associated with the potential "next" factor level. The "window" for a factor level is thus the level plus one or more levels with the closest average values of v given that there is >1 value of v within this group that has yet to be swapped.
If there is more than one covariate, then these steps are repeated iteratively for each covariate (i.e., selecting values of v given the stratum identified in covariate c_j, then among these values those also in the stratum identified in covariate c_k, and so on). In this case the order in which the covariates are listed in by can affect the outcome. The order can be permuted each values of v_i if permuteBy is TRUE.

Value

A data frame with one column swapped in a stratified manner relative another column or set of columns.

See Also

sample

Examples

```
# Example #1: Scramble column 1 with respect to columns 2 and 3.
# Note in the output high values of "a" tend to be associated with
# high values of "b" and low values of "c". This tendency decreases as "w" increases.

x <- data.frame(a=1:20, b=1:20, c=20:1, d=c(rep('a', 10), rep('b', 10)))
x$d <- as.factor(x$d)
x

# scramble by all other columns
sampleStrat(x=x, col=1, w=0.2, by='all', d=0.1)

# scramble by column "d"
sampleStrat(x=x, col=1, w=0.2, by='d', d=0.1)
```

```
# Example #2: The target variable and covariate are equal
# (perfectly collinear). How wide must the window (set by
# argument "w'" be to reduce the average correlation
# between them to an arbitrary low level?

df <- data.frame(a=1:100, b=1:100)
cor(df) # perfect correlation

corFrame <- data.frame()
for (w in seq(0.1, 1, 0.1)) {
    for (countRep in 1:10) {
       df2 <- sampleStrat(x=df, col=1, w=w)
       corFrame <- rbind(corFrame, data.frame(w=w, cor=cor(df2)[1, 2]))
    }
}

boxplot(cor ~ w, data=corFrame, xlab='w', ylab='correlation coefficient')
```

`se` *Standard error*

Description

Calculate the standard error of the mean.

Usage

```
se(x, na.rm = FALSE)
```

Arguments

`x`	Numeric vector.
`na.rm`	Logical. If TRUE then remove `NA`s before calculation.

Value

Numeric.

See Also

`link[stats]{sd}`

Examples

```
se(1:100)
```

Index

art, 2
as.character, *12*
as.formula, *12*

countConnected, 4

euclid, 5

geoMean, 6

hist, *7, 8*
hist2d, 7
histOverlap, 8

invLogitAdj, 9, *10*

logitAdj, *9*, 10

makeFormulae, 11
mmode, 13

nagelR2, 14

rank, *15*
rankMulti, 15
rmsd, 15

sample, *17*, *19*
sampleAcross, 17
sampleStrat, 18
se, 20