Package 'pairscale'

May 19, 2026

Type Package

Title Pairwise Rescaling of Numeric Matrices

Version 1.0

Description Normalization of numerical matrices by minimizing the mean/median/mode difference between all column pairs.

URL https://github.com/ftwkoopmans/pairscale/

License AGPL (>= 3)

Encoding UTF-8

SystemRequirements C++20

Depends R (>= 4.1.0)

Imports Rcpp (>= 1.0.5)

Suggests testthat (>= 3.0.0)

LinkingTo Rcpp, RcppArmadillo (>= 14.7)

Config/testthat/edition 3

RoxygenNote 7.3.3

Language en-US

NeedsCompilation yes

Author Frank Koopmans [aut, cre] (ORCID: <https://orcid.org/0000-0002-4973-5732>)

Maintainer Frank Koopmans <ftwkoopmans@gmail.com>

Repository CRAN

Date/Publication 2026-05-19 07:10:02 UTC

Contents

`pairdiff_madmean` *Distance matrix between columns of a matrix using their MAD-trimmed mean differences*

Description

Pairwise differences between all columns in a matrix

Usage

```
pairdiff_madmean(
  x,
  cols = NULL,
  min_value_count = 3L,
  threshold_std = 3,
  na_mode = "check"
)
```

Arguments

`x`	a numeric matrix that needs to be normalized. This variable is changed by reference! i.e. after this function, the original variable is updated
`cols`	optionally, provide an integer vector with column indices (in `x`) that should be used (these should be 1-based indices as per usual in R). Or set to `NULL` or `integer()` to use all columns
`min_value_count`	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)
`threshold_std`	ratio of MAD a value has to be away from the median to be considered an outlier (and thus removed/ignored). Note that the MAD thresholds are inclusive, i.e. values at +/- threshold_std*MAD from median are included

na_mode	string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a N x N numeric matrix (where N is number of column in input x) representing the MAD-trimmed mean difference between each column

---

`pairdiff_mean` *Distance matrix between columns of a matrix using their mean differences*

Description

Pairwise differences between all columns in a matrix

Usage

```
pairdiff_mean(x, cols = NULL, min_value_count = 3L, na_mode = "check")
```

Arguments

x	a numeric matrix that needs to be normalized. This variable is changed by reference! i.e. after this function, the original variable is updated
cols	optionally, provide an integer vector with column indices (in x) that should be used (these should be 1-based indices as per usual in R). Or set to NULL or integer() to use all columns
min_value_count	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)
na_mode	string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a N x N numeric matrix (where N is number of column in input x) representing the mean difference between each column

`pairdiff_median` *Distance matrix between columns of a matrix using their trimmed median differences*

Description

Pairwise differences between all columns in a matrix

Usage

```
pairdiff_median(x, cols = NULL, min_value_count = 3L, na_mode = "check")
```

Arguments

`x`	a numeric matrix that needs to be normalized. This variable is changed by reference! i.e. after this function, the original variable is updated
`cols`	optionally, provide an integer vector with column indices (in `x`) that should be used (these should be 1-based indices as per usual in R). Or set to `NULL` or `integer()` to use all columns
`min_value_count`	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)
`na_mode`	string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a N x N numeric matrix (where N is number of column in input `x`) representing the median difference between each column

`pairdiff_mode` *Distance matrix between columns of a matrix using their mode differences*

Description

Pairwise differences between all columns in a matrix

Usage

```
pairdiff_mode(
  x,
  cols = NULL,
  min_value_count = 3L,
  n_bins = 512L,
  adjust = 1,
  kernel_width_in_sd = 3,
  bandwidth_method = "nrd",
  mode_frac_maxdens = 1,
  na_mode = "check"
)
```

Arguments

`x`	a numeric matrix that needs to be normalized. This variable is changed by reference! i.e. after this function, the original variable is updated
`cols`	optionally, provide an integer vector with column indices (in `x`) that should be used (these should be 1-based indices as per usual in R). Or set to `NULL` or `integer()` to use all columns
`min_value_count`	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)
`n_bins`	grid size for density computation (the resolution / number of data points to use for binning column diffs)
`adjust`	bandwidth adjust factor for density computation
`kernel_width_in_sd`	maximum distance in standard deviations at which we'll include data points for the Gaussian kernal. Typically 3 or 4
`bandwidth_method`	method in which this function computes bandwidth and optionally trims the data prior to binning. "nrd" is the robust, safe default. "nrd_fast" is faster and yields similar results for most distributions. Use "nrd_fastest" only when all pairwise distances are known to be near gaussian (i.e. no strong outliers and sd() is a reliable metric). "nrd_subset" is an experimental option that may be removed, it is fast but heavily favors symmetric distributions and is thus biased ! Valid options: • `"nrd_fastest"` ; use all data points, for bandwidth don't use IQR. Sensitive to outliers ! • `"nrd_fast"` ; trim to quantiles 0.05 and 0.95, then use sd over these data points to compute bandwidth. Not as robust as the original nrd method, which also considers IQR, but much faster. • `"nrd"` ; adapted version of nrd; instead of IQR and sd we use IQR and sd over data trimmed at quantiles 0.05 and 0.95. Since we already sorted the data, computing sd on trimmed subset is 'free'

mode_frac_maxdens
: set to 1 to return the x-coordinate where the density is highest (mode). Setting this to a value < 1 will make this function compute not the mode, but the mean (x) value of the density where the density is some fraction higher than the maximum density. Typical value; 1. Optionally, set to 0.9 or 0.8 for possibly more robust center-finding, depending in your data distribution. Must not be smaller than 0.1 but recommended to never be lower than 0.7

na_mode
: string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a N x N numeric matrix (where N is number of column in input x) representing the mode difference between each column

---

pairdiff_trimmedmean *Distance matrix between columns of a matrix using their trimmed mean differences*

Description

Pairwise differences between all columns in a matrix

Usage

```
pairdiff_trimmedmean(
  x,
  cols = NULL,
  min_value_count = 3L,
  trim = 0.2,
  na_mode = "check"
)
```

Arguments

x
: a numeric matrix that needs to be normalized. This variable is changed by reference! i.e. after this function, the original variable is updated

cols
: optionally, provide an integer vector with column indices (in x) that should be used (these should be 1-based indices as per usual in R). Or set to NULL or integer() to use all columns

min_value_count
: the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)

`trim` amount of trim to apply to both the lower- and upper-parts of a vector before computing the mean. 0 indicates no trim, 0.5 indicates 100% trim (i.e. 50% of data on both sides) so that value is out of bounds. Typically set to 0.1-0.3

`na_mode` string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a N x N numeric matrix (where N is number of column in input `x`) representing the trimmed mean difference between each column

---

`pairscale_madmean` *Normalize matrix columns using their MAD-trimmed mean differences*

Description

Pairwise normalization of columns in a matrix, using the MAD-trimmed mean to define pairwise distances between columns

Usage

```
pairscale_madmean(
  x,
  clusters = NULL,
  min_value_count = 3L,
  threshold_std = 3,
  niter_irls = 50L,
  na_mode = "check"
)
```

Arguments

`x` a numeric matrix that needs to be normalized. This variable is changed by reference! i.e. after this function, the original variable is updated

`clusters` an integer vector, of same length as `ncol(x)`, that describes cluster identifiers. These values must describe a continuous set of integers between 1 and the total number of unique clusters. For example, a 6 column matrix with 3 columns from some experimental condition followed by 3 columns from another condition would be `c(1,1,1, 2,2,2)`. To disable mode-between rescaling, i.e. treat the entire matrix as one cluster, provide a vector with ones.

`min_value_count` the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)

threshold_std	ratio of MAD a value has to be away from the median to be considered an outlier (and thus removed/ignored). Note that the MAD thresholds are inclusive, i.e. values at +/- threshold_std*MAD from median are included
niter_irls	number of iterations to increase robustness (set to zero to disable)
na_mode	string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a numeric vector that represents the normalization factors that were applied to each column in x. Note that x is updated by reference.

---

pairscale_mean *Normalize matrix columns using their mean differences*

Description

Pairwise normalization of columns in a matrix, using the mean to define pairwise distances between columns

Usage

```
pairscale_mean(
  x,
  clusters = NULL,
  min_value_count = 3L,
  niter_irls = 50L,
  na_mode = "check"
)
```

Arguments

x	a numeric matrix that needs to be normalized. This variable is changed by reference! i.e. after this function, the original variable is updated
clusters	an integer vector, of same length as `ncol(x)`, that describes cluster identifiers. These values must describe a continuous set of integers between 1 and the total number of unique clusters. For example, a 6 column matrix with 3 columns from some experimental condition followed by 3 columns from another condition would be `c(1,1,1, 2,2,2)`. To disable mode-between rescaling, i.e. treat the entire matrix as one cluster, provide a vector with ones.
min_value_count	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)

niter_irls	number of iterations to increase robustness (set to zero to disable)
na_mode	string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a numeric vector that represents the normalization factors that were applied to each column in x. Note that x is updated by reference.

---

pairscale_median *Normalize matrix columns using their median differences*

Description

Pairwise normalization of columns in a matrix, using the median to define pairwise distances between columns

Usage

```
pairscale_median(
  x,
  clusters = NULL,
  min_value_count = 3L,
  niter_irls = 50L,
  na_mode = "check"
)
```

Arguments

x	a numeric matrix that needs to be normalized. This variable is changed by reference! i.e. after this function, the original variable is updated
clusters	an integer vector, of same length as `ncol(x)`, that describes cluster identifiers. These values must describe a continuous set of integers between 1 and the total number of unique clusters. For example, a 6 column matrix with 3 columns from some experimental condition followed by 3 columns from another condition would be `c(1,1,1, 2,2,2)`. To disable mode-between rescaling, i.e. treat the entire matrix as one cluster, provide a vector with ones.
min_value_count	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)
niter_irls	number of iterations to increase robustness (set to zero to disable)

`na_mode` string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a numeric vector that represents the normalization factors that were applied to each column in x. Note that x is updated by reference.

`pairscale_mode` *Normalize matrix columns using their mode differences*

Description

Pairwise normalization of columns in a matrix, using the mode to define pairwise distances between columns

Usage

```
pairscale_mode(
  x,
  clusters = NULL,
  min_value_count = 3L,
  n_bins = 512L,
  adjust = 1,
  kernel_width_in_sd = 3,
  bandwidth_method = "nrd",
  mode_frac_maxdens = 1,
  niter_irls = 50L,
  na_mode = "check"
)
```

Arguments

`x` a numeric matrix that needs to be normalized. This variable is changed by reference! i.e. after this function, the original variable is updated

`clusters` an integer vector, of same length as `ncol(x)`, that describes cluster identifiers. These values must describe a continuous set of integers between 1 and the total number of unique clusters. For example, a 6 column matrix with 3 columns from some experimental condition followed by 3 columns from another condition would be `c(1,1,1, 2,2,2)`. To disable mode-between rescaling, i.e. treat the entire matrix as one cluster, provide a vector with ones.

`min_value_count` the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)

`n_bins` grid size for density computation (the resolution / number of data points to use for binning column diffs)

`adjust` bandwidth adjust factor for density computation

`kernel_width_in_sd`
maximum distance in standard deviations at which we'll include data points for the Gaussian kernal. Typically 3 or 4

`bandwidth_method`
method in which this function computes bandwidth and optionally trims the data prior to binning. "nrd" is the robust, safe default. "nrd_fast" is faster and yields similar results for most distributions. Use "nrd_fastest" only when all pairwise distances are known to be near gaussian (i.e. no strong outliers and sd() is a reliable metric). "nrd_subset" is an experimental option that may be removed, it is fast but heavily favors symmetric distributions and is thus biased !

Valid options:

- `"nrd_fastest"` ; use all data points, for bandwidth don't use IQR. Sensitive to outliers !
- `"nrd_fast"` ; trim to quantiles 0.05 and 0.95, then use sd over these data points to compute bandwidth. Not as robust as the original nrd method, which also considers IQR, but much faster.
- `"nrd"` ; adapted version of nrd; instead of IQR and sd we use IQR and sd over data trimmed at quantiles 0.05 and 0.95. Since we already sorted the data, computing sd on trimmed subset is 'free'

`mode_frac_maxdens`
set to 1 to return the x-coordinate where the density is highest (mode). Setting this to a value < 1 will make this function compute not the mode, but the mean (x) value of the density where the density is some fraction higher than the maximum density. Typical value; 1. Optionally, set to 0.9 or 0.8 for possibly more robust center-finding, depending in your data distribution. Must not be smaller than 0.1 but recommended to never be lower than 0.7

`niter_irls` number of iterations to increase robustness (set to zero to disable)

`na_mode` string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a numeric vector that represents the normalization factors that were applied to each column in x. Note that x is updated by reference.

`pairscale_trimmedmean` *Normalize matrix columns using their trimmed-mean differences*

Description

Pairwise normalization of columns in a matrix, using the trimmed-mean to define pairwise distances between columns

Usage

```
pairscale_trimmedmean(
  x,
  clusters = NULL,
  min_value_count = 3L,
  trim = 0.2,
  niter_irls = 50L,
  na_mode = "check"
)
```

Arguments

`x`	a numeric matrix that needs to be normalized. This variable is changed by reference! i.e. after this function, the original variable is updated
`clusters`	an integer vector, of same length as `ncol(x)`, that describes cluster identifiers. These values must describe a continuous set of integers between 1 and the total number of unique clusters. For example, a 6 column matrix with 3 columns from some experimental condition followed by 3 columns from another condition would be `c(1,1,1, 2,2,2)`. To disable mode-between rescaling, i.e. treat the entire matrix as one cluster, provide a vector with ones.
`min_value_count`	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)
`trim`	amount of trim to apply to both the lower- and upper-parts of a vector before computing the mean. 0 indicates no trim, 0.5 indicates 100% trim (i.e. 50% of data on both sides) so that value is out of bounds. Typically set to 0.1-0.3
`niter_irls`	number of iterations to increase robustness (set to zero to disable)
`na_mode`	string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a numeric vector that represents the normalization factors that were applied to each column in x. Note that x is updated by reference.

`solve_graph_laplacian` *graph Laplacian approach to finding normalization factors*

Description

find normalization factors for a given distance matrix computed with e.g. `pairdiff_median()`. For increased robustness, this function offers iterative reweighted improvement of the initial estimate.

Usage

```
solve_graph_laplacian(M, niter_irls = 1L)
```

Arguments

`M`	skew-symmetric input matrix, generated with e.g. `pairdiff_median()`
`niter_irls`	refine the initial estimate using N additional iterative reweighted least squares loops for robust graph laplacian

Value

a numeric vector of length `ncol(M)` that contains scaling factors for `M`

Examples

```
# toy example
x = cbind(
  c(1,2,3,4),
  c(2,3,4,9),
  c(1,2,4,5),
  c(1,0,1,0)
)
# compute pairwide median difference between all columns
M = pairscale::pairdiff_median(x)
# solve matrix M to find scaling factors, without and with reweighting
s1 = pairscale::solve_graph_laplacian(M, niter_irls = 0)
s2 = pairscale::solve_graph_laplacian(M, niter_irls = 10)
# rescaled matrices; only the robust variant correctly aligns columns 1 and 2
t(t(x) - s1)
t(t(x) - s2)
```

`vector_madmean` *Compute MAD-trimmed mean value of a vector, with optional filtering for N datapoints*

Description

Compute measure of central tendency using efficient C++ code

Usage

```
vector_madmean(x, min_value_count = 1L, threshold_std = 3, na_mode = "check")
```

Arguments

`x`	numeric input vector, may contain non-finite values (removed if `na_mode` is left to default)
`min_value_count`	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)
`threshold_std`	ratio of MAD a value has to be away from the median to be considered an outlier (and thus removed/ignored). Note that the MAD thresholds are inclusive, i.e. values at +/- threshold_std*MAD from median are included
`na_mode`	string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a single numeric value representing the MAD-trimmed mean

`vector_mean` *Compute mean value of a vector, with optional filtering for N datapoints*

Description

Compute measure of central tendency using efficient C++ code

Usage

```
vector_mean(x, min_value_count = 1L, na_mode = "check")
```

Arguments

x	numeric input vector, may contain non-finite values (removed if na_mode is left to default)
min_value_count	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)
na_mode	string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a single numeric value representing the mean

---

vector_median *Compute median value of a vector, with optional filtering for N datapoints*

Description

Compute measure of central tendency using efficient C++ code

Usage

```
vector_median(x, min_value_count = 1L, na_mode = "check")
```

Arguments

x	numeric input vector, may contain non-finite values (removed if na_mode is left to default)
min_value_count	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)
na_mode	string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a single numeric value representing the median

vector_mode *Compute mode of a vector, with optional filtering for N datapoints*

Description

Compute measure of central tendency using efficient C++ code

Usage

```
vector_mode(
  x,
  min_value_count = 3L,
  n_bins = 512L,
  adjust = 1,
  kernel_width_in_sd = 3,
  bandwidth_method = "nrd",
  mode_frac_maxdens = 1,
  na_mode = "check"
)
```

Arguments

x	numeric input vector, may contain non-finite values (removed if na_mode is left to default)
min_value_count	the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)
n_bins	grid size for density computation (the resolution / number of data points to use for binning column diffs)
adjust	bandwidth adjust factor for density computation
kernel_width_in_sd	maximum distance in standard deviations at which we'll include data points for the Gaussian kernal. Typically 3 or 4
bandwidth_method	method in which this function computes bandwidth and optionally trims the data prior to binning. "nrd" is the robust, safe default. "nrd_fast" is faster and yields similar results for most distributions. Use "nrd_fastest" only when all pairwise distances are known to be near gaussian (i.e. no strong outliers and sd() is a reliable metric). "nrd_subset" is an experimental option that may be removed, it is fast but heavily favors symmetric distributions and is thus biased ! Valid options: • "nrd_fastest" ; use all data points, for bandwidth don't use IQR. Sensitive to outliers ! • "nrd_fast" ; trim to quantiles 0.05 and 0.95, then use sd over these data points to compute bandwidth. Not as robust as the original nrd method, which also considers IQR, but much faster.

- "nrd" ; adapted version of nrd; instead of IQR and sd we use IQR and sd over data trimmed at quantiles 0.05 and 0.95. Since we already sorted the data, computing sd on trimmed subset is 'free'

`mode_frac_maxdens`
: set to 1 to return the x-coordinate where the density is highest (mode). Setting this to a value < 1 will make this function compute not the mode, but the mean (x) value of the density where the density is some fraction higher than the maximum density. Typical value; 1. Optionally, set to 0.9 or 0.8 for possibly more robust center-finding, depending in your data distribution. Must not be smaller than 0.1 but recommended to never be lower than 0.7

`na_mode`
: string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a single numeric value representing the mode

`vector_trimmedmean` *Compute trimmed mean value of a vector, with optional filtering for N datapoints*

Description

Compute measure of central tendency using efficient C++ code

Usage

```
vector_trimmedmean(x, min_value_count = 1L, trim = 0.2, na_mode = "check")
```

Arguments

`x`
: numeric input vector, may contain non-finite values (removed if `na_mode` is left to default)

`min_value_count`
: the minimum number of values overlapping between a pair of columns (if there are fewer overlapping values, respective rescaling is not computed)

`trim`
: amount of trim to apply to both the lower- and upper-parts of a vector before computing the mean. 0 indicates no trim, 0.5 indicates 100% trim (i.e. 50% of data on both sides) so that value is out of bounds. Typically set to 0.1-0.3

na_mode	string value that indicated how should we should deal with NA values (default). "check" = test if NA values are present and if so, remove these. "present" = you already know NA values are present so we can skip the check for NA values for efficiency. "unchecked" = you guarantee that there are no NA values in input, thus we'll use the fastest code paths that skip over any NA checks downstream; ONLY select this if you are sure there are no NA/NaN/Inf/-Inf values !

Value

a single numeric value representing the trimmed mean

Index