Package 'LikertMakeR'

May 30, 2025

Type Package

Title Synthesise and Correlate Likert Scale and Rating-Scale Data Based on Summary Statistics

Version 1.1.0

Description Generate and correlate synthetic Likert and rating-scale data with predefined means, standard deviations, Cronbach's Alpha, Factor Loading table, and other summary statistics. Worked examples and documentation are available in the package vignettes, accessible via browseVignettes(``LikertMakeR'').

License MIT + file LICENSE

URL https://github.com/WinzarH/LikertMakeR, https://winzarh.github.io/LikertMakeR/

BugReports https://github.com/WinzarH/LikertMakeR/issues

Depends R (>= 4.2.0)

Imports dplyr, gtools, Matrix, Rcpp

Suggests kableExtra, knitr, psych, psychTools, rmarkdown, rosetta, testthat (>= 3.0.0)

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Config/Needs/website rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

RoxygenNote 7.3.2

NeedsCompilation yes

Author Hume Winzar [cre, aut] (ORCID: <https://orcid.org/0000-0001-7475-2641>)

Maintainer Hume Winzar <winzar@gmail.com>

Repository CRAN

Date/Publication 2025-05-30 08:20:02 UTC

Contents

`alpha` *Calculate Cronbach's Alpha from a correlation matrix or dataframe*

Description

`alpha()` calculate Cronbach's Alpha from a given correlation matrix or a given dataframe.

Usage

```
alpha(cormatrix = NULL, data = NULL)
```

Arguments

`cormatrix`	(real) a square symmetrical matrix with values ranging from -1 to +1 and '1' in the diagonal
`data`	(real) a dataframe or matrix

Value

a single value

Examples

```
## Sample data frame
df <- data.frame(
  V1  =  c(4, 2, 4, 3, 2, 2, 2, 1),
  V2  =  c(4, 1, 3, 4, 4, 3, 2, 3),
  V3  =  c(4, 1, 3, 5, 4, 1, 4, 2),
  V4  =  c(4, 3, 4, 5, 3, 3, 3, 3)
)

## example correlation matrix
corMat <- matrix(
```

```
  c(
    1.00, 0.35, 0.45, 0.70,
    0.35, 1.00, 0.60, 0.55,
    0.45, 0.60, 1.00, 0.65,
    0.70, 0.55, 0.65, 1.00
  ),
  nrow = 4, ncol = 4
)

## apply function examples

alpha(cormatrix = corMat)

alpha(, df)

alpha(corMat, df)
```

`correlateScales` *Create a dataframe of correlated scales from different dataframes of scale items*

Description

`correlateScales()` creates a dataframe of scale items representing correlated constructs, as one might find in a completed questionnaire.

Usage

```
correlateScales(dataframes, scalecors)
```

Arguments

`dataframes`	a list of 'k' dataframes to be rearranged and combined
`scalecors`	target correlation matrix - should be a symmetric k*k positive-semi-definite matrix, where 'k' is the number of dataframes

Details

Correlated rating-scale items generally are summed or averaged to create a measure of an "unobservable", or "latent", construct. `correlateScales()` takes several such dataframes of rating-scale items and rearranges their rows so that the scales are correlated according to a predefined correlation matrix. Univariate statistics for each dataframe of rating-scale items do not change, but their correlations with rating-scale items in other dataframes do.

Value

Returns a dataframe whose columns are taken from the starter dataframes and whose summated values are correlated according to a user-specified correlation matrix

Examples

```
## three attitudes and a behavioural intention
n <- 32
lower <- 1
upper <- 5

### attitude #1
cor_1 <- makeCorrAlpha(items = 4, alpha = 0.90)
means_1 <- c(2.5, 2.5, 3.0, 3.5)
sds_1 <- c(0.9, 1.0, 0.9, 1.0)

Att_1 <- makeItems(
  n = n, means = means_1, sds = sds_1,
  lowerbound = rep(lower, 4), upperbound = rep(upper, 4),
  cormatrix = cor_1
)



### attitude #2
cor_2 <- makeCorrAlpha(items = 5, alpha = 0.85)
means_2 <- c(2.5, 2.5, 3.0, 3.0, 3.5)
sds_2 <- c(1.0, 1.0, 0.9, 1.0, 1.5)

Att_2 <- makeItems(
  n = n, means = means_2, sds = sds_2,
  lowerbound = rep(lower, 5), upperbound = rep(upper, 5),
  cormatrix = cor_2
)



### attitude #3
cor_3 <- makeCorrAlpha(items = 6, alpha = 0.75)
means_3 <- c(2.5, 2.5, 3.0, 3.0, 3.5, 3.5)
sds_3 <- c(1.0, 1.5, 1.0, 1.5, 1.0, 1.5)

Att_3 <- makeItems(
  n = n, means = means_3, sds = sds_3,
  lowerbound = rep(lower, 6), upperbound = rep(upper, 6),
  cormatrix = cor_3
)



### behavioural intention
intent <- lfast(n, mean = 3.0, sd = 3, lowerbound = 0, upperbound = 10) |>
  data.frame()
names(intent) <- "int"


### target scale correlation matrix
```

```
scale_cors <- matrix(
  c(
    1.0, 0.6, 0.5, 0.3,
    0.6, 1.0, 0.4, 0.2,
    0.5, 0.4, 1.0, 0.1,
    0.3, 0.2, 0.1, 1.0
  ),
  nrow = 4
)

data_frames <- list("A1" = Att_1, "A2" = Att_2, "A3" = Att_3, "Int" = intent)


### apply the function
my_correlated_scales <- correlateScales(
  dataframes = data_frames,
  scalecors = scale_cors
)
head(my_correlated_scales)
```

`eigenvalues` *calculate eigenvalues of a correlation matrix with optional scree plot*

Description

`eigenvalues()` calculate eigenvalues of a correlation matrix and optionally produces a scree plot.

Usage

```
eigenvalues(cormatrix, scree = FALSE)
```

Arguments

`cormatrix`	(real, matrix) a correlation matrix
`scree`	(logical) default = `FALSE`. If `TRUE` (or 1), then `eigenvalues()` produces a scree plot to illustrate the eigenvalues

Value

a vector of eigenvalues

report on positive-definite status of cormatrix

Examples

```
## define parameters

correlationMatrix <- matrix(
  c(
    1.00, 0.25, 0.35, 0.40,
    0.25, 1.00, 0.70, 0.75,
    0.35, 0.70, 1.00, 0.80,
    0.40, 0.75, 0.80, 1.00
  ),
  nrow = 4, ncol = 4
)

## apply function

evals <- eigenvalues(cormatrix = correlationMatrix)
evals <- eigenvalues(correlationMatrix, 1)
```

---

lcor *Rearrange elements in each column of a data-frame to fit a predefined correlation matrix*

Description

`lcor()` rearranges values in each column of a data-frame so that columns are correlated to match a predefined correlation matrix.

Usage

```
lcor(data, target, passes = 10)
```

Arguments

`data`	data-frame that is to be rearranged
`target`	target correlation matrix - should be a symmetric k*k positive-semi-definite matrix
`passes`	Number of optimization passes (default = 10) Increasing this value *MAY* improve results if n-columns (target correlation matrix dimensions) are many.

Details

Values in a column do not change, so univariate statistics remain the same.

Value

Returns a dataframe whose column-wise correlations approximate a user-specified correlation matrix

Examples

```
## parameters
n <- 32
lowerbound <- 1
upperbound <- 5
items <- 5

mydat3 <- data.frame(
  x1 = lfast(n, 2.5, 0.75, lowerbound, upperbound, items),
  x2 = lfast(n, 3.0, 1.50, lowerbound, upperbound, items),
  x3 = lfast(n, 3.5, 1.00, lowerbound, upperbound, items)
)

cor(mydat3) |> round(3)

tgt3 <- matrix(
  c(
    1.00, 0.50, 0.75,
    0.50, 1.00, 0.25,
    0.75, 0.25, 1.00
  ),
  nrow = 3, ncol = 3
)

## apply function
new3 <- lcor(mydat3, tgt3)

## test output
cor(new3) |> round(3)
```

---

lexact *Deprecated. Use lfast() instead*

Description

`lexact` is DEPRECATED. Replaced by new version of `lfast`.

`lexact` remains as a legacy for earlier package users. It is now just a wrapper for `lfast`

Previously, `lexact` used a Differential Evolution (DE) algorithm to find an optimum solution with desired mean and standard deviation, but we found that the updated `lfast` function is much faster and just as accurate.

Also the package is much less bulky.

Usage

```
lexact(n, mean, sd, lowerbound, upperbound, items = 1)
```

Arguments

n	(positive, int) number of observations to generate
mean	(real) target mean
sd	(real) target standard deviation
lowerbound	(positive, int) lower bound
upperbound	(positive, int) upper bound
items	(positive, int) number of items in the rating scale.

Value

a vector of simulated data approximating user-specified conditions.

Examples

```
x <- lexact(
  n = 256,
  mean = 4.0,
  sd = 1.0,
  lowerbound = 1,
  upperbound = 7,
  items = 6
)

x <- lexact(256, 2, 1.8, 0, 10)
```

---

lfast *Synthesise rating-scale data with predefined mean and standard deviation*

Description

`lfast()` applies a simple Evolutionary Algorithm to find a vector that best fits the desired moments.

`lfast()` generates random discrete values from a scaled Beta distribution so the data replicate a rating scale - for example, a 1-5 Likert scale made from 5 items (questions) or 0-10 likelihood-of-purchase scale.

Usage

```
lfast(n, mean, sd, lowerbound, upperbound, items = 1, precision = 0)
```

Arguments

n	(positive, int) number of observations to generate
mean	(real) target mean, between upper and lower bounds
sd	(positive, real) target standard deviation
lowerbound	(int) lower bound (e.g. '1' for a 1-5 rating scale)
upperbound	(int) upper bound (e.g. '5' for a 1-5 rating scale)
items	(positive, int) number of items in the rating scale. Default = 1
precision	(positive, real) can relax the level of accuracy required. (e.g. '1' generally generates a vector with moments correct within '0.025', '2' generally within '0.05') Default = 0

Value

a vector approximating user-specified conditions.

Examples

```
## six-item 1-7 rating scale
x <- lfast(
  n = 256,
  mean = 4.0,
  sd = 1.25,
  lowerbound = 1,
  upperbound = 7,
  items = 6
)

## five-item -3 to +3 rating scale
x <- lfast(
  n = 64,
  mean = 0.025,
  sd = 1.25,
  lowerbound = -3,
  upperbound = 3,
  items = 5
)

## four-item 1-5 rating scale with medium variation
x <- lfast(
  n = 128,
  mean = 3.0,
  sd = 1.00,
  lowerbound = 1,
  upperbound = 5,
  items = 4,
  precision = 5
)

## eleven-point 'likelihood of purchase' scale
```

```
x <- lfast(256, 3, 3.0, 0, 10)
```

`makeCorrAlpha` *Correlation matrix from Cronbach's Alpha*

Description

`makeCorrAlpha()` generates a random correlation matrix of given dimensions and predefined *Cronbach's Alpha*.

Such a correlation matrix can be applied to the `makeItems()` function to generate synthetic data with the predefined alpha.

Usage

```
makeCorrAlpha(items, alpha, variance = 0.5, precision = 0)
```

Arguments

`items`	(positive, int) matrix dimensions: number of rows & columns to generate
`alpha`	(real) target Cronbach's Alpha (usually positive, must be between about -0.3 and +1)
`variance`	(positive, real) Default = 0.5. User-provided standard deviation of values sampled from a normally-distributed log transformation.
`precision`	(positive, real) Default = 0. User-defined value ranging from '0' to '3' to add some random variation around the target *Cronbach's Alpha*. '0' gives an exact alpha (to two decimal places)

Value

a correlation matrix

Note

Random values generated by `makeCorrAlpha()` are highly volatile. `makeCorrAlpha()` may not generate a feasible (positive-definite) correlation matrix, especially when

- variance is high relative to
  - desired Alpha, and
  - desired correlation dimensions

`makeCorrAlpha()` will inform the user if the resulting correlation matrix is positive definite, or not.

If the returned correlation matrix is not positive-definite, a feasible solution may still be possible. The user is encouraged to try again, possibly several times, to find one.

Examples

```
# define parameters
items <- 4
alpha <- 0.85
variance <- 0.5

# apply function
set.seed(42)
cor_matrix <- makeCorrAlpha(items = items, alpha = alpha, variance = variance)

# test function output
print(cor_matrix)
alpha(cor_matrix)
eigenvalues(cor_matrix, 1)

# higher alpha, more items
cor_matrix2 <- makeCorrAlpha(items = 8, alpha = 0.95)

# test output
cor_matrix2 |> round(2)
alpha(cor_matrix2) |> round(3)
eigenvalues(cor_matrix2, 1) |> round(3)


# large random variation around alpha
set.seed(42)
cor_matrix3 <- makeCorrAlpha(items = 6, alpha = 0.85, precision = 2)

# test output
cor_matrix3 |> round(2)
alpha(cor_matrix3) |> round(3)
eigenvalues(cor_matrix3, 1) |> round(3)
```

---

makeCorrLoadings *Correlation matrix from item factor loadings*

Description

`makeCorrLoadings()` generates a correlation matrix of inter-item correlations based on item factor loadings as might be seen in *Exploratory Factor Analysis* (**EFA**) or a *Structural Equation Model* (**SEM**).

Such a correlation matrix can be applied to the `makeItems()` function to generate synthetic data with those predefined factor structures.

Usage

```
makeCorrLoadings(
  loadings,
```

```
  factorCor = NULL,
  uniquenesses = NULL,
  nearPD = FALSE
)
```

Arguments

loadings	(numeric matrix) **k** (items) by **f** (factors) matrix of **standardised** factor loadings. Item names and Factor names are taken from the row_names (items) and the column_names (factors), if present. @note "Censored" loadings (for example, where loadings less than '0.30' are removed for clarity) tend to severely reduce the accuracy of the `makeCorrLoadings()` function. For a detailed demonstration, see the file **makeCorrLoadings_Validate.pdf** in the package website on GitHub.
factorCor	(numeric matrix) **f** x **f** factor correlation matrix
uniquenesses	(numeric vector) length **k** vector of uniquenesses. If *NULL*, the default, compute from the calculated communalities.
nearPD	(logical) If TRUE, project factorCor and the final correlation matrix onto nearest Positive Definite matrix, if needed.

Value

Correlation matrix of inter-item correlations

Note

The *nearPD* option applies the *Matrix::nearPD()* function to force a non-positive-definite matrix to be positive-definite. It should be used only when a matrix is "nearly" PD. In most cases, a non-PD matrix that appears in the makeCorrLoadings() function means that there is a problem with the input parameters.

Examples

```
# Example loadings
factorLoadings <- matrix(
  c(
    0.05, 0.20, 0.70,
    0.10, 0.05, 0.80,
    0.05, 0.15, 0.85,
    0.20, 0.85, 0.15,
    0.05, 0.85, 0.10,
    0.10, 0.90, 0.05,
    0.90, 0.15, 0.05,
    0.80, 0.10, 0.10
  ),
  nrow = 8, ncol = 3, byrow = TRUE
)

# row and column names
rownames(factorLoadings) <- c("Q1", "Q2", "Q3", "Q4", "Q5", "Q6", "Q7", "Q8")
```

```
colnames(factorLoadings) <- c("Factor1", "Factor2", "Factor3")

# Factor correlation matrix
factorCor <- matrix(
  c(
    1.0,  0.7, 0.6,
    0.7,  1.0, 0.4,
    0.6,  0.4, 1.0
  ),
  nrow = 3, byrow = TRUE
)

# Apply the function
itemCorrelations <- makeCorrLoadings(factorLoadings, factorCor)

round(itemCorrelations, 3)
```

`makeItems` *Synthesise rating-scale data with given first and second moments and a predefined correlation matrix*

Description

`makeItems()` generates a dataframe of random discrete values so the data replicate a rating scale, and are correlated close to a predefined correlation matrix.

`makeItems()` is wrapper function for:

- `lfast()`, generates a dataframe that best fits the desired moments, and
- `lcor()`, which rearranges values in each column of the dataframe so they closely match the desired correlation matrix.

Usage

```
makeItems(n, means, sds, lowerbound, upperbound, cormatrix)
```

Arguments

`n`	(positive, int) sample-size - number of observations
`means`	(real) target means: a vector of length k of mean values for each scale item
`sds`	(positive, real) target standard deviations: a vector of length k of standard deviation values for each scale item
`lowerbound`	(positive, int) a vector of length k (same as rows & columns of correlation matrix) of values for lower bound of each scale item (e.g. '1' for a 1-5 rating scale)
`upperbound`	(positive, int) a vector of length k (same as rows & columns of correlation matrix) of values for upper bound of each scale item (e.g. '5' for a 1-5 rating scale)
`cormatrix`	(real, matrix) the target correlation matrix: a square symmetric positive-semi-definite matrix of values ranging between -1 and +1, and '1' in the diagonal.

Value

a dataframe of rating-scale values

Examples

```
## define parameters

n <- 16
dfMeans <- c(2.5, 3.0, 3.0, 3.5)
dfSds <- c(1.0, 1.0, 1.5, 0.75)
lowerbound <- rep(1, 4)
upperbound <- rep(5, 4)

corMat <- matrix(
  c(
    1.00, 0.30, 0.40, 0.60,
    0.30, 1.00, 0.50, 0.70,
    0.40, 0.50, 1.00, 0.80,
    0.60, 0.70, 0.80, 1.00
  ),
  nrow = 4, ncol = 4
)

item_names <- c("Q1", "Q2", "Q3", "Q4")
rownames(corMat) <- item_names
colnames(corMat) <- item_names

## apply function

df <- makeItems(
  n = n, means = dfMeans, sds = dfSds,
  lowerbound = lowerbound, upperbound = upperbound, cormatrix = corMat
)

## test function

str(df)

# means
apply(df, 2, mean) |> round(3)

# standard deviations
apply(df, 2, sd) |> round(3)

# correlations
cor(df) |> round(3)
```

`makeItemsScale` *Generate scale items from a summated scale, with desired Cronbach's Alpha*

Description

`makeItemsScale()` generates a random dataframe of scale items based on a predefined summated scale (such as created by the `lfast()` function), and a desired *Cronbach's Alpha*.

scale, lowerbound, upperbound, items, alpha, variance

Usage

```
makeItemsScale(
  scale,
  lowerbound,
  upperbound,
  items,
  alpha = 0.8,
  variance = 0.5
)
```

Arguments

`scale`	(int) a vector or dataframe of the summated rating scale. Should range from ('lowerbound' * 'items') to ('upperbound' * 'items')
`lowerbound`	(int) lower bound of the scale item (example: '1' in a '1' to '5' rating)
`upperbound`	(int) upper bound of the scale item (example: '5' in a '1' to '5' rating)
`items`	(positive, int) k, or number of columns to generate
`alpha`	(posiitve, real) desired *Cronbach's Alpha* for the new dataframe of items. Default = '0.8'. See `@details` for further information on the `alpha` parameter
`variance`	(positive, real) the quantile from which to select items that give given summated scores. Must lie between '0' and '1'. Default = '0.5'. See `@details` for further information on the `variance` parameter

Details

alpha:

`makeItemsScale()` rearranges the item values within each row, attempting to give a dataframe of Likert-scale items that produce a predefined *Cronbach's Alpha*.

Default value for target alpha is '0.8'.

More extreme values for the 'variance' parameter may reduce the chances of achieving the desired Alpha. So you may need to experiment a little.

variance:

There may be many ways to find a combination of integers that sum to a specific value, and these combinations have different levels of variance:

- low-variance: '3 + 4 = 7'
- high-variance: '1 + 6 = 7'

The 'variance' parameter defines guidelines for the amount of variance among item values that your new dataframe should have.

For example, consider a summated value of '9' on which we apply the `makeItemsScale()` function to generate three items. With zero variance (variance parameter = '0'), then we see all items with the same value, the mean of '3'. With variance = '1', then we see all items with values that give the maximum variance among those items.

variance	v1	v2	v3	sum
0.0	3	3	3	9
0.2	3	3	3	9
0.4	2	3	4	9
0.6	1	4	4	9
0.8	2	2	5	9
1.0	1	3	5	9

Similarly, the same mean value applied to six items with `makeItemsScale()` gives the following combinations at different values of the 'variance' parameter.

variance	v1	v2	v3	v4	v5	v6	sum
0.0	3	3	3	3	3	3	18
0.2	1	3	3	3	4	4	18
0.4	1	2	3	4	4	4	18
0.6	1	1	4	4	4	4	18
0.8	1	1	3	4	4	5	18
1.0	1	1	1	5	5	5	18

And a mean value of '3.5' gives the following combinations.

variance	v1	v2	v3	v4	v5	v6	sum
0.0	3	3	3	4	4	4	21
0.2	3	3	3	3	4	5	21
0.4	2	2	4	4	4	5	21
0.6	1	3	4	4	4	5	21
0.8	1	2	4	4	5	5	21
1.0	1	1	4	5	5	5	21

The default value for 'variance' is '0.5' which gives a reasonable range of item values. But if you want 'responses' that are more consistent then choose a lower variance value.

Value

a dataframe with 'items' columns and 'length(scale)' rows

Examples

```
## define parameters
k <- 4
lower <- 1
upper <- 5

## scale properties
n <- 64
mean <- 3.0
sd <- 0.85

## create scale
set.seed(42)
meanScale <- lfast(
  n = n, mean = mean, sd = sd,
  lowerbound = lower, upperbound = upper,
  items = k
)
summatedScale <- meanScale * k

## create new items
newItems <- makeItemsScale(
  scale = summatedScale,
  lowerbound = lower, upperbound = upper,
  items = k
)

### test new items
# str(newItems)
# alpha(data = newItems) |> round(2)


## very low variance usually gives higher Cronbach's Alpha
mydat_20 <- makeItemsScale(
  scale = summatedScale,
  lowerbound = lower, upperbound = upper,
  items = k, alpha = 0.8, variance = 0.20
)

### test new data frame
# str(mydat_20)

# moments <- data.frame(
#   means = apply(mydat_20, MARGIN = 2, FUN = mean) |> round(3),
#   sds = apply(mydat_20, MARGIN = 2, FUN = sd) |> round(3)
# ) |> t()

# moments
```

```
# cor(mydat_20) |> round(2)
# alpha(data = mydat_20) |> round(2)


## default alpha (0.8) and higher variance (0.8)
mydat_80 <- makeItemsScale(
  scale = summatedScale,
  lowerbound = lower, upperbound = upper,
  items = k, variance = 0.80
)

### test new dataframe
# str(mydat_80)

# moments <- data.frame(
#   means = apply(mydat_80, MARGIN = 2, FUN = mean) |> round(3),
#   sds = apply(mydat_80, MARGIN = 2, FUN = sd) |> round(3)
# ) |> t()

# moments

# cor(mydat_80) |> round(2)
# alpha(data = mydat_80) |> round(2)
```

`makePaired` *Synthesise a dataset from paired-sample t-test summary statistics*

Description

`makePaired()` generates a dataset from paired-sample t-test summary statistics.

`makePaired()` generates correlated values so the data replicate rating scales taken, for example, in a before and after experimental design.

The function is effectively a wrapper function for `lfast()` and `lcor()` with the addition of a t-statistic from which the between-column correlation is inferred.

Paired t-tests apply to observations that are associated with each other. For example: the same people before and after a treatment; the same people rating two different objects; ratings by husband & wife; *etc.*

The t-test for paired data is given by:

- t = mean(D) / (sd(D) / sqrt(n))

where:

- D = differences in values,
- mean(D) = mean of the differences,
- sd(D) = standard deviation of the differences, where

- sd(D)^2 = sd(X_before)^2 + sd(X_after)^2 - 2 * cov(X_before, X_after)

A paired-sample t-test thus requires an estimate of the covariance between the two sets of observations. `makePaired()` rearranges these formulae so that the covariance is inferred from the t-statistic.

Usage

```
makePaired(
  n,
  means,
  sds,
  t_value,
  lowerbound,
  upperbound,
  items = 1,
  precision = 0
)
```

Arguments

`n`	(positive, integer) sample size
`means`	(real) 1:2 vector of target means for two before/after measures
`sds`	(real) 1:2 vector of target standard deviations
`t_value`	(real) desired paired t-statistic
`lowerbound`	(integer) lower bound (e.g. '1' for a 1-5 rating scale)
`upperbound`	(integer) upper bound (e.g. '5' for a 1-5 rating scale)
`items`	(positive, integer) number of items in the rating scale. Default = 1
`precision`	(positive, real) relaxes the level of accuracy required. Default = 0

Value

a dataframe approximating user-specified conditions.

Note

Larger sample sizes usually result in higher t-statistics, and correspondingly small p-values.

Small sample sizes with relatively large standard deviations and relatively high t-statistics can result in impossible correlation values.

Similarly, large sample sizes with low t-statistics can result in impossible correlations. That is, a correlation outside of the -1:+1 range.

If this happens, the function will fail with an *ERROR* message. The user should review the input parameters and insert more realistic values.

Examples

```
n <- 20
pair_m <- c(2.5, 3.0)
pair_s <- c(1.0, 1.5)
lower <- 1
upper <- 5
k <- 6
t <- -2.5

pairedDat <- makePaired(
  n = n, means = pair_m, sds = pair_s,
  t_value = t,
  lowerbound = lower, upperbound = upper, items = k
)

str(pairedDat)
cor(pairedDat) |> round(2)

t.test(pairedDat$X1, pairedDat$X2, paired = TRUE)
```

Index