Package 'superb'

May 9, 2026

Type Package

Title Summary Plots with Adjusted Error Bars

Version 1.0.1

Date 2025-11-30

Author Denis Cousineau [aut, cre],
Bradley Harding [ctb],
Marc-Andre Goulet [ctb],
Jesika Walker [art, pre]

Maintainer Denis Cousineau <denis.cousineau@uottawa.ca>

BugReports https://github.com/dcousin3/superb/issues/

URL https://github.com/dcousin3/superb/,
https://CRAN.R-project.org/package=superb,
https://dcousin3.github.io/superb/

Description Computes standard error and confidence interval of various descriptive statistics under various designs and sampling schemes. The main function, superb(), return a plot. It can also be used to obtain a dataframe with the statistics and their precision intervals so that other plotting environments (e.g., Excel) can be used. See Cousineau and colleagues (2021) <doi:10.1177/25152459211035109> or Cousineau (2017) <doi:10.5709/acp-0214-z> for a review as well as Cousineau (2005) <doi:10.20982/tqmp.01.1.p042>, Morey (2008) <doi:10.20982/tqmp.04.2.p061>, Baguley (2012) <doi:10.3758/s13428-011-0123-7>, Cousineau & Laurencelle (2016) <doi:10.1037/met0000055>, Cousineau & O'Brien (2014) <doi:10.3758/s13428-013-0441-z>, Calderini & Harding <doi:10.20982/tqmp.15.1.p001> for specific references. The documentation is available at <https://dcousin3.github.io/superb/> .

License GPL-3

Encoding UTF-8

VignetteBuilder knitr

LazyData true

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports Rdpack (>= 0.7), methods, utils, stats, MASS, lsr (>= 0.5), plyr (>= 1.8.4), ggplot2 (>= 3.5.0), stringr, foreign, reshape2, shiny, shinyBS, rrapply

Suggests psych, rlang, dplyr, gridExtra, emojifont, fMultivar, grid, knitr, lattice, boot, png, rmarkdown, rstatix, RColorBrewer, sadists, scales, testthat, tibble

RdMacros Rdpack

NeedsCompilation no

Repository CRAN

Date/Publication 2025-12-04 16:50:02 UTC

Contents

`biasCorrectionTransform` *bias-correction transform*

Description

`biasCorrectionTransform()` is a transformation that can be applied to a matrix of data. The resulting matrix's variance is corrected for bias (Morey 2008; Baguley 2012; Cousineau and O'Brien 2014);

Usage

```
biasCorrectionTransform(dta, variables)
```

Arguments

`dta`	a data.frame containing the data in wide format;
`variables`	a vector of column names on which the transformation will be applied. the remaining columns will be left unchanged

Value

a data.frame of the same form as dta with the variables transformed.

This function is useful when passed to the argument `preprocessfct` of `superb()` where it performs a modification of the data matrix.

References

Baguley T (2012). "Calculating and graphing within-subject confidence intervals for ANOVA." *Behavior Research Methods*, **44**, 158 – 175. doi:10.3758/s1342801101237.

Cousineau D, O'Brien F (2014). "Error bars in within-subject designs: a comment on Baguley (2012)." *Behavior Research Methods*, **46**(4), 1149–1151. doi:10.3758/s134280130441z.

Morey RD (2008). "Confidence Intervals from Normalized Data: A correction to Cousineau (2005)." *Tutorials in Quantitative Methods for Psychology*, **4**, 61 – 64. doi:10.20982/tqmp.04.2.p061.

`bootstrapPrecisionMeasures`

Bootstrapped measures of precision

Description

superb also comes with a few built-in measures of precisions that uses bootstrap. More can be added based on users needs. All `bootstrapSE.fct()` functions produces an interval width; all `bootstrapPI.fct()` produces the lower and upper limits of an interval. These estimates are based on 5,000 sub-samples by default. Change this default with`options("superb.bootstrapIter" = number )`. See Efron and Tibshirani (1994) for a comprehensive introduction. The bootstrap estimates are called PI which stands for Precision intervals. This is to denote that they estimate the sampling distribution, not the predictive distribution on which all confidence intervals are based (Rousselet et al. 2019; Poitevineau and Lecoutre 2010; Lecoutre 1999).

Usage

```
bootstrapSE.mean(x)

bootstrapPI.mean(x, gamma)

bootstrapSE.median(x)

bootstrapPI.median(x, gamma)

bootstrapSE.hmean(x)

bootstrapPI.hmean(x, gamma)

bootstrapSE.gmean(x)

bootstrapPI.gmean(x, gamma)

bootstrapSE.var(x)

bootstrapPI.var(x, gamma)
```

```
bootstrapSE.sd(x)

bootstrapPI.sd(x, gamma)
```

Arguments

x	a vector of numbers, the sample data (mandatory);
gamma	a confidence level for PI (default 0.95).

Value

a measure of precision (SE) or an interval of precision (PI).

References

Efron B, Tibshirani RJ (1994). *An introduction to the bootstrap*. CRC press.

Lecoutre B (1999). "Two useful distributions for Bayesian predictive procedures under normal models." *Journal of Statistical Planning and Inference*, **79**, 93 – 105. doi:10.1016/S03783758(98)00231-6.

Poitevineau J, Lecoutre B (2010). "Implementing Bayesian predictive procedures: The K-prime and K-square distributions." *Computational Statistics and Data Analysis*, **54**, 724 – 731. doi:10.1016/j.csda.2008.11.004.

Rousselet GA, Pernet CR, Wilcox RR (2019). "A practical introduction to the bootstrap: A versatile method to make inferences by using data-driven simulations." *psyArXiv*. doi:10.31234/osf.io/h8ft7.

Examples

```
# the confidence interval of the mean for default 95% and 90% confidence level
bootstrapPI.mean( c(1,2,3) )
bootstrapPI.mean( c(1,2,3), gamma = 0.90)

# Standard errors for standard deviation or variance
bootstrapSE.sd( c(1,2,3) )
bootstrapSE.var( c(1,2,3) )
```

`CousineauLaurencelleLambda` *Cousineau-Laurencelle's lambda correction for cluster-randomized sampling*

Description

The functions `CousineauLaurencelleLambda()` returns the correction factor for cluster-randomized sampling. This correction is then used in a variety of ways, for example, to get the effective number of participants (in a power study) or to correct a t-test. See (Cousineau and Laurencelle 2016).

Usage

```
CousineauLaurencelleLambda(paramvector)
```

Arguments

`paramvector`	A vector with, in that order, the intra-class correlation r, the number of clusters, then the number of participants in all the clusters.

Value

lambda the correction factor for cluster-randomized sampling.

References

Cousineau D, Laurencelle L (2016). "A Correction Factor for the Impact of Cluster Randomized Sampling and Its Applications." *Psychological Methods*, **21**, 121 – 135. doi:10.1037/met0000055.

Examples

```
# Example from Cousineau & Laurencelle, 2017, p. 124:
CousineauLaurencelleLambda( c(0.2, 5, 20, 20, 20, 20, 20) )
# 2.234188
```

`dataFigure1` *Data for Figure 1*

Description

The data, taken from (Cousineau 2017), is an example where the "stand-alone" 95\% confidence interval of the means returns a result in contradiction with the result of a statistical test. The paradoxical result is resolved by using adjusted confidence intervals, here the different-adjusted confidence interval.

Usage

```
data(dataFigure1)
```

Format

An object of class data.frame.

Source

doi:10.5709/acp0214z

References

Cousineau D (2017). "Varieties of confidence intervals." *Advances in Cognitive Psychology*, **13**, 140 – 155. doi:10.5709/acp0214z.

Examples

```
library(ggplot2)
library(gridExtra)
data(dataFigure1)

options(superb.feedback = 'none') # shut down 'warnings' and 'design' interpretation messages

## realize the plot with unadjusted (left) and ajusted (right) 95% confidence intervals
plt1a <- superb(
    score ~ grp,
    dataFigure1,
    adjustments=list(purpose = "single"),
    plotLayout="bar" ) +
  xlab("Group") + ylab("Score") + labs(title="95% CI\n") +
  coord_cartesian( ylim = c(85,120) ) +
  geom_hline(yintercept = 100, colour = "black", linewidth = 0.5, linetype=2) +
  showSignificance( c(0.5, 2.5), 115, -1, "significant???",
                    segmentParams = list( colour = "red" ) )
plt1b <- superb(
    score ~ grp,
    dataFigure1,
    adjustments=list(purpose = "difference"),
    plotLayout="bar" ) +
  xlab("Group") + ylab("Score") + labs(title="Difference-adjusted 95% CI\n") +
  coord_cartesian( ylim = c(85,120) ) +
  geom_hline(yintercept = 100, colour = "black", linewidth = 0.5, linetype=2)+
  showSignificance( c(0.5, 2.5), 115, -1, "Not significant!",
                    segmentParams = list( colour = "chartreuse3" ) )
plt1  <- grid.arrange(plt1a,plt1b,ncol=2)

## realise the correct t-test to see the discrepancy
t.test(dataFigure1$score[dataFigure1$grp==1],
       dataFigure1$score[dataFigure1$grp==2],
       var.equal=TRUE)
```

`dataFigure2` *Data for Figure 2*

Description

The data, taken from (Cousineau 2017)7, is an example where the "stand-alone" 95\% confidence interval of the means returns a result in contradiction with the result of a statistical test. The paradoxical result is resolved by using adjusted confidence intervals, here the correlation- and different-adjusted confidence interval.

Usage

```
data(dataFigure2)
```

Format

An object of class data.frame.

Source

doi:10.5709/acp0214z

References

Cousineau D (2017). "Varieties of confidence intervals." *Advances in Cognitive Psychology*, **13**, 140 – 155. doi:10.5709/acp0214z.

Examples

```
library(ggplot2)
library(gridExtra)
data(dataFigure2)

options(superb.feedback = 'none') # shut down 'warnings' and 'design' interpretation messages

## realize the plot with unadjusted (left) and ajusted (right) 95% confidence intervals
plt2a <- superb(
    cbind(pre, post) ~ .,
    dataFigure2,
    WSFactors = "Moment(2)",
    adjustments=list(purpose = "difference"),
    plotLayout="bar" ) +
  xlab("Group") + ylab("Score") + labs(title="Difference-adjusted 95% CI\n") +
  coord_cartesian( ylim = c(85,120) ) +
  geom_hline(yintercept = 100, colour = "black", linewidth = 0.5, linetype=2) +
  showSignificance( c(0.5, 2.5), 115, -1, "not significant???",
                    segmentParams = list( colour = "red" ) )
plt2b <- superb(
    cbind(pre, post) ~ .,
    dataFigure2,
    WSFactors = "Moment(2)",
    adjustments=list(purpose = "difference", decorrelation = "CA"),
    plotLayout="bar" ) +
 xlab("Group") + ylab("Score") + labs(title="Correlation and difference-adjusted\n95% CI") +
  coord_cartesian( ylim = c(85,120) ) +
```

```
  geom_hline(yintercept = 100, colour = "black", linewidth = 0.5, linetype=2)+
  showSignificance( c(0.5, 2.5), 115, -1, "highly significant!!!",
                    segmentParams = list( colour = "chartreuse3" ) )
plt2  <- grid.arrange(plt2a,plt2b,ncol=2)

## realise the correct t-test to see the discrepancy
t.test(dataFigure2$pre, dataFigure2$post, paired=TRUE)
```

`dataFigure3` *Data for Figure 3*

Description

The data, inspired from (Cousineau and Laurencelle 2016), is an example where the "stand-alone" 95\ a result in contradiction with the result of a statistical test. The paradoxical result is resolved by using adjusted confidence intervals, here the cluster- and different-adjusted confidence interval.

Usage

```
data(dataFigure3)
```

Format

An object of class data.frame.

Source

doi:10.5709/acp0214z

References

Cousineau D, Laurencelle L (2016). "A Correction Factor for the Impact of Cluster Randomized Sampling and Its Applications." *Psychological Methods*, **21**, 121 – 135. doi:10.1037/met0000055.

Examples

```
library(ggplot2)
library(gridExtra)
data(dataFigure3)

options(superb.feedback = 'none') # shut down 'warnings' and 'design' interpretation messages

## realize the plot with unadjusted (left) and ajusted (right) 95% confidence intervals
plt3a <- superb(
    VD ~ grp,
    dataFigure3,
    adjustments=list(purpose = "difference", samplingDesign = "SRS"),
    plotLayout="bar" ) +
```

```
  xlab("Group") + ylab("Score") + labs(title="Difference-adjusted 95% CI\n") +
  coord_cartesian( ylim = c(85,120) ) +
  geom_hline(yintercept = 100, colour = "black", linewidth = 0.5, linetype=2)+
  showSignificance( c(0.5, 2.5), 115, -1, "significant???",
                    segmentParams = list( colour = "red" ) )
plt3b <- superb(
    VD ~ grp,
    dataFigure3,
    adjustments=list(purpose = "difference", samplingDesign = "CRS"),
    plotLayout="bar", clusterColumn = "cluster" ) +
  xlab("Group") + ylab("Score") + labs(title="Cluster and difference-adjusted\n95% CI") +
  coord_cartesian( ylim = c(85,120) ) +
  geom_hline(yintercept = 100, colour = "black", linewidth = 0.5, linetype=2)+
  showSignificance( c(0.5, 2.5), 117, -1, "not significant!!!",
                    segmentParams = list( colour = "chartreuse3" ) )
plt3  <- grid.arrange(plt3a,plt3b,ncol=2)

## realise the correct t-test to see the discrepancy
res   <- t.test(dataFigure3$VD[dataFigure3$grp==1],
                dataFigure3$VD[dataFigure3$grp==2],
                var.equal=TRUE)
micc  <- mean(c(0.491334683772226, 0.20385744842838)) # mean ICC given by superbPlot
lam   <- CousineauLaurencelleLambda(c(micc, 5,5,5,5,5,5))
tcorr <- res$statistic / lam
pcorr <- 1-pt(tcorr,4)
# let's see the t value and its p value:
c(tcorr, pcorr)
```

`dataFigure4` *Data for Figure 4*

Description

The data, inspired from Cousineau (2017), shows an example where the "stand-alone" 95\ a result in contradiction with the result of a statistical test. The paradoxical result is resolved by using adjusted confidence intervals, here the population size-adjusted confidence interval.

Usage

```
data(dataFigure4)
```

Format

An object of class data.frame.

Source

doi:10.5709/acp0214z

References

Cousineau D (2017). "Varieties of confidence intervals." *Advances in Cognitive Psychology*, **13**, 140 – 155. doi:10.5709/acp0214z.

Examples

```
library(ggplot2)
library(gridExtra)
data(dataFigure4)

options(superb.feedback = 'none') # shut down 'warnings' and 'design' interpretation messages

## realize the plot with unadjusted (left) and ajusted (right) 95% confidence intervals
plt4a = superb(
    score ~ group,
    dataFigure4,
    adjustments=list(purpose = "single", popSize = Inf),
    plotLayout="bar" ) +
  xlab("Group") + ylab("Score") + labs(title="Difference-adjusted 95% CI\n") +
  coord_cartesian( ylim = c(85,120) ) +
  geom_hline(yintercept = 100, colour = "black", linewidth = 0.5, linetype=2)+
  showSignificance( c(0.5, 1.5), 115, -1, "not significant???",
                    segmentParams = list( colour = "red" ) )
plt4b = superb(
    score ~ group,
    dataFigure4,
    adjustments=list(purpose = "single", popSize = 50 ),
    plotLayout="bar" ) +
 xlab("Group") + ylab("Score") + labs(title="Population size and difference-\nadjusted 95% CI") +
  coord_cartesian( ylim = c(85,120) ) +
  geom_hline(yintercept = 100, colour = "black", linewidth = 0.5, linetype=2)+
  showSignificance( c(0.5, 1.5), 115, -1, "highly significant!!!",
                    segmentParams = list( colour = "chartreuse3" ) )
plt4 = grid.arrange(plt4a,plt4b,ncol=2)

## realise the correct t-test to see the discrepancy
res = t.test(dataFigure4$score, mu=100)
tcorr = res$statistic /sqrt(1-25/50)
pcorr = 1-pt(tcorr,24)
c(tcorr, pcorr)
```

`geom_flat_violin` *geom_flat_violin for expanded density displays*

Description

`geom_flat_violin()` is a geom for ggplots; it is based on the original script to create raincloud plots. It relies largely on this code previously written by David Robinson and the package ggplot2 by Hadley Wickham.

Code from Allen et al. (2019)

It is expanded in two different ways. First, it is possible to decide the direction of the violin using the `direction` argument (values are 0 = symmetrical; 1 = extending to the right; -1 = extending to the left); the last two cases are "half"-violin. The second argument is `push` which pushed the violin away from the median line (default = 0).

Usage

```
geom_flat_violin(
  mapping = NULL,
  data = NULL,
  stat = "ydensity",
  position = "dodge",
  trim = TRUE,
  scale = "area",
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	(as usual) see `geom_violin()`
`data`	(as usual) see `geom_violin()`
`stat`	(as usual) see `geom_violin()`
`position`	(as usual) see `geom_violin()`
`trim`	If `TRUE` (default), trim the tails of the violins to the range of the data. If `FALSE`, don't trim the tails.
`scale`	if "area" (default), all violins have the same area (before trimming the tails). If "count", areas are scaled proportionally to the number of observations. If "width", all violins have the same maximum width.
`show.legend`	(as usual) see `geom_violin()`
`inherit.aes`	(as usual) see `geom_violin()`
`...`	all additional parameters are sent to the underlying `geom_path()`. It includes • `direction` either -1,0, or +1; • `push` a positive number. • `na.rm` (as usual) see `geom_violin()` • `orientation` (as usual) see `geom_violin()`

Value

a layer containing violins in a ggplot object

References

Allen M, Poggiali D, Whitaker K, Marshall TR, van Langen J, Kievit RA (2019). "Raincloud plots: a multi-platform tool for robust data visualization." *Wellcome open research*, **4**.

Examples

```
library(superb) # to import the geom_flat_violin
library(ggplot2)

# let's have a fake data frame with three groups:
dta <- dta <- GRD( SubjectsPerGroup = 20,
    BSFactors = "Vacations(yes,no,maybe)",
    RenameDV = "tiredeness",
    Population = list(mean=75, stddev=15),
    Effects = list("Vacations" = custom(-20,+20,+10))
)

# The most basic plot = a regular error bar
superb( tiredeness ~ Vacations, dta)

# an example with default violins
superb( tiredeness ~ Vacations, dta,
    plotLayout = "pointjitterviolin" )

# the same with some ornementations:
superb( tiredeness ~ Vacations, dta,
    plotLayout = "pointjitterviolin",
    violinParams = list(direction = 1, push = 0.2, fill="green", alpha = 0.3)
) + theme_bw() + coord_flip() + ylab("Tiredeness")

# This new geom is integrated inside superb() so that you can use it
# directly. Let's see examples:

# show the violins only
ggplot(dta, aes(y = tiredeness, x = Vacations ) ) +
   geom_flat_violin()

# change the parameters of the violins
ggplot(dta, aes(y = tiredeness, x = Vacations ) ) +
   geom_flat_violin( fill = "green")

# all the arguments manipulated
ggplot(dta, aes(y = tiredeness, x = Vacations ) ) +
    geom_flat_violin( fill = "green", direction = 1, push =0.)

# using direction within aes
dta <- transform(dta, dir = ifelse(Vacations == "no", 1, -1))

ggplot(dta, aes(y = tiredeness, x = Vacations, direction = dir ) ) +
    geom_flat_violin( fill = "green", push =0.2)
```

---

geom_superberrorbar *geom_superberrorbar for expanded error bar displays*

Description

`geom_superberrorbar()` is a geom for ggplots; it is based on the original `geom_errorbar()` (and is totally compatible with it) but expands this geom in four different ways. First, it is possible to decide the error bar tips `direction` which can be unidirectional, pointing to the "left" or to the "right" or go in "both" directions. Second, it is possible set `tipformat` to "double" or "triple" the horizontal marks at the extremities of the error bar, with a `tipgap` of your liking. Third, an additional characteristic is `vcolour` to set a different colour for the vertical part of the error bar or the pair `vcolour` and `wcolour` for the top half and bottom half of the vertical error bar. The colour(s) can also be "NA" to have it invisible. Lastly, the error bar can be `pointing` "up" and "down" or go in "both" (the default)

Usage

```
geom_superberrorbar(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	(as usual) see geom_errorbar
`data`	(as usual) see geom_errorbar
`stat`	(as usual) see geom_errorbar
`position`	(as usual) see geom_errorbar
`...`	all additional parameters are sent to the underlying geom_path. Includes • `pointing` either "up", "down" or "both" up and down (default is "both"); • `direction` "left", "right" or "both" (Default is "both") • `tipformat` "single", "double" or "triple" to add additional marker lines to the tips (default is "single") • `tipgap` The spacing between the markers when "double" or "triple" is used (default 0.1) • `vcolour` for the vertical part of the error bar • `wcolour` if specified, for the second half of the vertical part of the error bar.
`na.rm`	(as usual) see `geom_errorbar()`
`orientation`	(as usual) see `geom_errorbar()`
`show.legend`	(as usual) see `geom_errorbar()`
`inherit.aes`	(as usual) see `geom_errorbar()`

Value

a layer containing error bars in a ggplot object

Examples

```
library(superb) # to import the geom_superberrorbar
library(ggplot2)

# let's have a fake data frame
dta <- data.frame(grp = c(1,2,3), center=c(1,2,3), width = c(1,1,1.5) )

# an example with none of the new features = a regular error bar
ggplot(dta, aes(ymin=center-width, ymax=center+width, x = grp ) ) +
  geom_superberrorbar()

# an example with left-pointing error bars
ggplot(dta, aes(ymin=center-width, ymax=center+width, x = grp ) ) +
  geom_superberrorbar(direction="left", width = 0.1)

# an example with doubled-tipped error bar and the default tipgap
ggplot(dta, aes(ymin=center-width, ymax=center+width, x = grp ) ) +
  geom_superberrorbar(tipformat = "double", width = 0.1)

# an example with left-pointing tripled-tip error bars with small gaps
ggplot(dta, aes(ymin=center-width, ymax=center+width, x = grp ) ) +
  geom_superberrorbar(tipformat = "triple", width= 0.1, tipgap = 0.04, direction = "left")

# an example with unidirectional error bars (here "up" bars)
ggplot(dta, aes(y= center, ymin=center-width, ymax=center+width, x = grp ) ) +
  geom_bar(stat="identity", fill = "yellow") +
  geom_superberrorbar(pointing = "up")

# a final example with two-coloured, left-pointing tripled-tip error bars with small gaps
ggplot(dta, aes(ymin=center-width, ymax=center+width, x = grp ) ) +
  geom_superberrorbar(tipformat = "triple", width= 0.1, tipgap = 0.04, direction = "left",
           colour = "black", vcolour = "orange")

# This new geom is integrated inside superb() so that you can vary the
# error bar shapes. Let's see examples:

# using GRD to generate random data with a moderate effect
options(superb.feedback = 'none') # shut down 'warnings' and 'design' interpretation messages
test <- GRD(SubjectsPerGroup  = 20,
   WSFactors = "Moment(5)",
            Effects = list("Moment" = extent(10) ),
            Population = list(mean = 100, stddev = 15, rho = 0.8) )

ornate = list(
        labs(title =paste("(left)            95% confidence intervals",
                        "\n(right)          99% confidence intervals",
                        "\n(center, up) 99.9% confidence intervals")),
        xlab("Moment"), ylab("Score"),
```

```
        coord_cartesian( ylim = c(85,125) )
)

plt1 <- superb(
            crange(DV.1, DV.5) ~ .,
            test,
            WSFactors = "Moment(5)",
            adjustments=list(purpose = "difference", decorrelation = "CA"),
            errorbarParams = list(direction = "left", color="purple",
                                  width = 0.2, position = position_nudge(-0.05) ),
            gamma     = 0.95,
            plotLayout = "line" ) + ornate
plt2 <- superb(
            crange(DV.1, DV.5) ~ .,
            test,
            WSFactors = "Moment(5)",
            adjustments=list(purpose = "difference", decorrelation = "CA"),
          errorbarParams = list(direction = "right", tipgap = 0.40, tipformat = "double",
                                  width = 0.2, position = position_nudge(+0.05) ),
            gamma     = 0.99,
            plotLayout = "line" ) + ornate
plt3 <- superb(
            crange(DV.1, DV.5) ~ .,
            test,
            WSFactors = "Moment(5)",
            adjustments=list(purpose = "difference", decorrelation = "CA"),
           errorbarParams = list(direction = "both", tipformat = "single", pointing="up",
                                  width = 0.2, position = position_nudge(0) ),
            gamma     = 0.999,
            plotLayout = "line" ) + ornate

# transform the ggplots into "grob" so that they can be manipulated
plt1 <- ggplotGrob(plt1)
plt2 <- ggplotGrob(plt2 + makeTransparent() )
plt3 <- ggplotGrob(plt3 + makeTransparent() )

# put the grobs onto an empty ggplot
ggplot() +
    annotation_custom(grob=plt1) +
    annotation_custom(grob=plt2) +
    annotation_custom(grob=plt3)


# all of them as aesthetics
set.seed(1)
library(dplyr)
dat <- data.frame(Trial = c(rep("Pre",9),rep("Post",9)),
                    Time = rep.int(seq(0,120,15),2),
                    var = c(rnorm(9,15,2),rnorm(9,22,2)),
                    var_sd = c(rnorm(18,3,1)))
dat <- mutate(dat, point = ifelse(Trial == "Pre","down","up"))
dat <- mutate(dat, direc = ifelse(Trial == "Pre","left","right"))
dat <- mutate(dat, tipfo = ifelse(Trial == "Pre","double","triple"))
```

```
dat <- mutate(dat, vcolo = ifelse(Trial == "Pre","red","blue"))

ggplot(data = dat,
       aes(x = Time, y = var, group = Trial)) +
   geom_line(aes(linetype = Trial)) +
   geom_point(aes(shape= Trial, fill = Trial), size=2) +
   geom_superberrorbar(aes(ymin=var-var_sd,
                           ymax=var+var_sd,
                           direction = direc,
                           pointing = point,
                           wcolour = vcolo,
                           vcolour = "green",
                           tipgap = 0.40,
                           tipformat = tipfo
       ),
       width = 4)
```

GRD *Generate random data*

Description

The function GRD() generates a data frame containing random data suitable for analyses. The data can be from within-subject or between-group designs. Within-subject designs are in wide format. The function was originally presented in Calderini and Harding (2019).

Usage

```
GRD(
  RenameDV = "DV",
  SubjectsPerGroup = 100,
  BSFactors = "",
  WSFactors = "",
  Effects = list(),
  Population = list(mean = 0, stddev = 1, rho = 0, scores =
    "rnorm(1, mean = GM, sd = STDDEV)"),
  Contaminant = list(mean = 0, stddev = 1, rho = 0, scores =
    "rnorm(1, mean = CGM, sd = CSTDDEV)", proportion = 0),
  Instrument = list(precision = 10^(-8), range = c(-Inf, +Inf))
)
```

Arguments

RenameDV	provide a name for the dependent variable (default DV)
SubjectsPerGroup	indicates the number of simulated scores per group (default 100 in each group)

BSFactors	a string indicating the between-subject factor(s) with, between parenthesis, the number of levels or the list of level names. Multiple factors are separated with a colon ":" or enumerated in a vector of strings.
WSFactors	a string indicating the within-subject factor(s) in the same format as the between-subject factors
Effects	a list detailing the effects to apply to the data. The effects can be given with a list of "factorname" = effect_specification or "factorname1*factorname2" = effect_specification pairs, in which effect_specification can either be slope(), extent(), custom() and Rexpression(). For slope and extent, provide a range, for custom, indicate the deviation from the grand mean for each cell, finally, for Rexpression, give between quote any R commands which returns the deviation from the grand mean, using the factors. See the last example below.
Population	a list providing the population characteristics (default is a normal distribution with a mean of 0 and standard deviation of 1)
Contaminant	a list providing the contaminant characteristics and the proportion of contaminant (default 0)
Instrument	a list providing some characteristics of the measurement instrument (at this time, its precision and range only).

Value

a data.frame() with the simulated scores.

Note

Note that the range effect specification has been renamed extent to avoid masking the base function base::range().

References

Calderini M, Harding B (2019). "GRD for R: An intuitive tool for generating random data in R." *The Quantitative Methods for Psychology*, **15**(1), 1–11. doi:10.20982/tqmp.15.1.p001.

Calderini M, Harding B (2019). "GRD for R: An intuitive tool for generating random data in R." *The Quantitative Methods for Psychology*, **15**(1), 1–11. doi:10.20982/tqmp.15.1.p001.

Examples

```
# Simplest example using all the default arguments:
dta <- GRD()
head(dta)
hist(dta$DV)

# Renaming the dependant variable and setting the group size:
dta <- GRD( RenameDV = "score", SubjectsPerGroup = 200 )
hist(dta$score )

# Examples for a between-subject design and for a within-subject design:
dta <- GRD( BSFactors = '3', SubjectsPerGroup = 20)
dta <- GRD( WSFactors = "Moment (2)", SubjectsPerGroup = 20)
```

```
# A complex, 3 x 2 x (2) mixed design with a variable amount of participants in the 6 groups:
dta <- GRD(BSFactors = "difficulty(3) : gender (2)",
        WSFactors="day(2)",
        SubjectsPerGroup=c(20,24,12,13,28,29)
      )

# Defining population characteristics :
dta <- GRD(
        RenameDV = "IQ",
SubjectsPerGroup = 20,
        Population=list(
                    mean=100,  # will set GM to 100
                    stddev=15  # will set STDDEV to 15
                  )
       )
hist(dta$IQ)

# This example adds an effect along the "Difficulty" factor with a slope of 15
dta <- GRD(BSFactors="Difficulty(5)", SubjectsPerGroup = 100,
    Population=list(mean=50,stddev=15),
    Effects = list("Difficulty" = slope(15) )  )
# show the mean performance as a function of difficulty:
superb(DV ~ Difficulty, dta )

# An example in which the moments are correlated
dta <- GRD( BSFactors = "Difficulty(2)",WSFactors = "Moment (2)",
    SubjectsPerGroup = 125,
    Effects = list("Difficulty" = slope(3), "Moment" = slope(1) ),
    Population=list(mean=50,stddev=20,rho=0.85)
)
# the mean plot on the raw data...
superb(cbind(DV.1,DV.2) ~ Difficulty, dta, WSFactors = "Moment(2)",
    plotLayout="line",
    adjustments = list (purpose="difference") )
# ... and the mean plot on the decorrelated data;
# because of high correlation, the error bars are markedly different
superb(cbind(DV.1,DV.2) ~ Difficulty, dta, WSFactors = "Moment(2)",
    plotLayout="line",
    adjustments = list (purpose="difference", decorrelation = "CM") )

# This example creates a dataset in a 3 x 2 design. It has various effects,
# one effect of difficulty, with an overall effect of 10 more (+3.33 per level),
# one effect of gender, whose slope is 10 points (+10 points for each additional gender),
# and finally one interacting effect, which is 0 for the last three cells of the design:
GRD(
    SubjectsPerGroup = 10,
    BSFactors  = c("difficulty(3)","gender(2)"),
    Population = list(mean=100,stddev=15),
    Effects    = list(
        "difficulty" = extent(10),
        "gender"=slope(10),
        "difficulty*gender"=custom(-300,+200,-100,0,0,0)
```

```
    )
)


# This last example creates a single group dataset,
# The instrument is assumed to return readings to
# plus or minus 0.1 only
GRD(
    SubjectsPerGroup = 10,
    Population = list(mean=100,stddev=15),
    Instrument    = list(
        precision = 0.1, range = c(+0,+200)
    )
)
```

`HyunhFeldtEpsilon` *Hyunh and Feldt's epsilon measure of sphericity*

Description

HyunhFeldtEpsilon() is a measure of sphericity created by Geisser and Greenhouse (1958). The original measure was biased and therefore, Huynh and Feldt (1976) produced a revised version (note that the 1976 paper contained typos that were uncorrected in SPSS; Lecoutre (1991))

Usage

```
HyunhFeldtEpsilon(dta, cols)
```

Arguments

`dta`	a data.frame
`cols`	a vector of column names indicating the relevant columns on which to compute epsilon. Any other columns are ignored.

Value

returns the Hyunh-Feldt estimate of sphericity epsilon

References

Geisser S, Greenhouse SW (1958). "An extension of Box's results on the use of the F distribution in multivariate analysis." *Annals of Mathematical Statistics*, **29**(3), 885–891.

Huynh H, Feldt LS (1976). "Estimation of the Box correction for degrees of freedom from sample data in randomized block and split-plot designs." *Journal of educational statistics*, **1**(1), 69–82.

Lecoutre B (1991). "A correction for the ε approximate test in repeated measures designs with two or more independent groups." *Journal of Educational Statistics*, **16**(4), 371–372.

`is.formula` *logical functions for formulas*

Description

The functions `is.formula()`, `is.one.sided()`, `has.nested.terms()`, `has.cbind.terms()`, `has.crange.terms()`, `in.formula()` and `sub.formulas()` performs checks or extract sub-formulas from a given formula.

Usage

```
is.formula(frm)

is.one.sided(frm)

has.nested.terms(frm)

has.cbind.terms(frm)

has.crange.terms(frm)

in.formula(frm, whatsym)

sub.formulas(frm, head)
```

Arguments

`frm`	a formula;
`whatsym`	a symbol to search in the formula;
`head`	the beginning of a sub-formula to extract

Details

These formulas are for internal use.

Value

`is.formula(frm)`, `has.nested.terms(frm)`, `has.crange.terms(frm)` and `has.cbind.terms(frm)` returns TRUE if frm is a formula, contains a `|` or a `crange` or a `cbind` respectively; `in.formula(frm, whatsym)` returns TRUE if the symbol `whatsym` is somewhere in 'frm'; `sub.formulas(frm, head)` returns a list of all the sub-formulas which contains `head`.

Examples

```
is.formula( Frequency ~ Intensity * Pitch )

has.nested.terms( Level ~ Factor | Level )
```

```
has.cbind.terms( Frequency ~ cbind(Low,Medium,High) * cbind(Soft, Hard) )

has.crange.terms( Frequency ~ crange(Low,High) * cbind(Soft, Hard) )

in.formula( Frequency ~ Intensity * Pitch, "Pitch" )

sub.formulas( Frequency ~ cbind(Low,Medium,High) * cbind(Soft, Hard), "cbind" )
```

`makeTransparent` *makes ggplots with transparent elements*

Description

`makeTransparent()` is an extension to ggplots which makes all the elements of the plot transparent except the data being displayed. This is useful to superimpose multiple plots, e.g. to generate plots with multiple error bars for example.

Usage

```
makeTransparent()
```

Value

does not return anything new; set the elements of the current plot to transparent.

Examples

```
# make a basic plot
superb(len ~ dose + supp, ToothGrowth )
# make a basic plot with transparent elements
superb(len ~ dose + supp, ToothGrowth) + makeTransparent()
```

`MauchlySphericityTest` *Mauchly's test of Sphericity*

Description

Performs a test of sphericity on a dataframe with multiple measures, one subject per line. It assesses the significance of the null hypothesis that the covariance matrix is spherical. This test is described in (Abdi 2010)

Usage

```
MauchlySphericityTest(dta, cols)
```

Arguments

dta	A data frame containing within-subject measures, one participant per line;
cols	A vector indicating the columns containing the measures.

Value

p the p-value of the null hypothesis that the data are spherical.

References

Abdi H (2010). "The greenhouse-geisser correction." *Encyclopedia of research design*, **1**(1), 544–548.

Examples

```
# creates a small data frames with 4 subject's scores for 5 measures:
dta <- data.frame(cbind(
        col1 <- c(3., 6., 2., 2., 5.),
        col2 <- c(4., 5., 4., 4., 3.),
        col3 <- c(2., 7., 7., 8., 6.),
        col4 <- c(6., 8., 4., 6., 5.)
    ))
# performs the test (here p = 0.5824)
MauchlySphericityTest(dta)
```

---

measuresWithMissingData

Measures with missing data

Description

The following three functions can be used with missing data. They return the mean, the standard error of the mean and the confidence interval of the mean.Note that we hesitated to provide these functions: you should deal with missing data prior to making your plot. Removing NAs from the mean in a univariate setting is equivalent to performing mean imputation. See Wikipedia's missing data for more. Also note that for repeated-measure design, only CA adjustment is available.

Usage

```
meanNArm(x)

SE.meanNArm(x)

CI.meanNArm(x, gamma)
```

Arguments

x	a vector of numbers, the sample data (mandatory);
gamma	a confidence level for CI (default 0.95).

Value

the means, a measure of precision (SE) or an interval of precision (CI) in the presence of missing data.

References

There are no references for Rd macro \insertAllCites on this help page.

Examples

```
# the confidence interval of the mean for default 95% and 90% confidence level
meanNArm( c(1,2,3, NA) )
SE.meanNArm( c(1,2,3, NA) )
CI.meanNArm( c(1,2,3, NA) )
CI.meanNArm( c(1,2,3, NA), gamma = 0.90)
```

---

poolSDTransform *pooled standard deviation transform*

Description

poolSDTransform() is a transformations that can be applied to a matrix of data. The resulting matrix has the column- standard deviations equal to the pool standard deviations of the individual columns, the solution adopted by (Loftus and Masson 1994).

Usage

```
poolSDTransform(dta, variables)
```

Arguments

dta	a data.frame containing the data in wide format;
variables	a vector of column names on which the transformation will be applied. the remaining columns will be left unchanged

Value

a data.frame of the same form as dta with the variables transformed.

This function is useful when passed to the argument preprocessfct of superb() where it performs a modification of the data matrix.

References

Loftus GR, Masson MEJ (1994). "Using confidence intervals in within-subject designs." *Psychonomic Bulletin & Review*, **1**, 476 – 490. doi:10.3758/BF03210951.

`precisionMeasures` *Precision measures*

Description

superb comes with a few built-in measures of precisions. All `SE.fct()` functions produces an interval width; all `CI.fct()` produces the lower and upper limits of an interval. See (Harding et al. 2014; Harding et al. 2015) for more. "superbPlot-compatible" precision measures must have these parameters:

Usage

```
SE.mean(x)

CI.mean(x, gamma)

SE.median(x)

CI.median(x, gamma)

SE.hmean(x)

CI.hmean(x, gamma)

SE.gmean(x)

CI.gmean(x, gamma)

SE.var(x)

CI.var(x, gamma)

SE.sd(x)

CI.sd(x, gamma)

SE.MAD(x)

CI.MAD(x, gamma)

SE.IQR(x)
```

```
CI.IQR(x, gamma)

SE.fisherskew(x)

CI.fisherskew(x, gamma)

SE.pearsonskew(x)

CI.pearsonskew(x, gamma)

SE.fisherkurtosis(x)

CI.fisherkurtosis(x, gamma)
```

Arguments

x	a vector of numbers, the sample data (mandatory);
gamma	a confidence level for CI (default 0.95).

Value

a measure of precision (SE) or an interval of precision (CI).

References

Harding B, Tremblay C, Cousineau D (2014). "Standard errors: A review and evaluation of standard error estimators using Monte Carlo simulations." *The Quantitative Methods for Psychology*, **10**(2), 107–123.

Harding B, Tremblay C, Cousineau D (2015). "The standard error of the Pearson skew." *The Quantitative Methods for Psychology*, **11**(1), 32–36.

Examples

```
# the confidence interval of the mean for default 95% and 90% confidence level
CI.mean( c(1,2,3) )
CI.mean( c(1,2,3), gamma = 0.90)

# Standard errors for standard deviation, for MAD and for fisher skew
SE.sd( c(1,2,3) )
SE.MAD( c(1,2,3) )
SE.fisherskew( c(1,2,3) )
```

precisionMeasureWithCustomDF *Confidence intervals with custom degree of freedom*

Description

The following function computes a confidence interval with custom degree of freedom. The default is to use $N-1$ but this number is not always appropriate. To get the exact critical value from which to construct the confidence interval, it is necessary to "pool" the degrees of freedom. This last expression means that the degree of freedom is the total number of data minus 1 for each condition except the last, and minus 1 for each participant except the last. In formula, if the number of repeated measures is p, the number of participants is n, and the total sample size is N (with $N = pxn$, then the pooled degree of freedom is $(p-1) \times (n-1)$ or equivalently $N - p - n + 1$. Another example where custom degree of freedom can be used is when there are heterogeneous variances, the confidence interval of the mean should mirror a Welsh test where the degrees of freedom are altered based on variances. The function `CIwithDF.mean()` accept any arbitrary defined degree of freedom (df). The df must be combined to the argument 'gamma' after the confidence level.

Usage

```
CIwithDF.mean(x, gamma = 0.95 )
```

Arguments

`x`	a vector of numbers, the sample data (mandatory);
`gamma`	a vector containing first a confidence level for CI (default 0.95) and a custom degree of freedom (when unspecified, it uses `n-1` where `n` is the number of observations in each of the condition).

Details

See the vignette "Unequal variances, Welch test, Tryon adjustment, and superb" for an example of use.

Value

the confidence interval (CI) where the `t` value is based on the custom-set degree of freedom.

References

There are no references for Rd macro `\insertAllCites` on this help page.

Examples

```
# this will issue a warning as no custom degree of freedom (df) is provided
CIwithDF.mean( c(1,2,3), gamma = 0.90)
# the confidence interval of the mean for 90% confidence level
CIwithDF.mean( c(1,2,3), gamma = c(0.90, 1.5) ) # uses 1.5 as df instead of 2.
```

```
# ====================================================
# A COMPLETE EXAMPLE:
# ====================================================

# Let's generate random measurements with GRD:
# (we generate a very small group of 10 to have a chance to see differences)
dta <- GRD( WSFactors = "Moment (3)", SubjectsPerGroup = 10)

# We need ggplotGrop
library(ggplot2)

# First, a regular plot
plt1 <- superb(
           cbind(DV.1,DV.2,DV.3) ~ .,
           dta,
           WSFactors      = "Moment(3)",
           plotLayout     = "line",
           adjustments    = list (purpose="difference",decorrelation="CM"),
           errorbar       = "CI",
           gamma          = 0.95,
           errorbarParams = list(color="orange", width= 0.1, direction = "both",
                                 position = position_nudge(-0.0) )
        )
# Second, a plot where the df are set to the default
plt2 <- superb(
           cbind(DV.1,DV.2,DV.3) ~ .,
           dta,
           WSFactors      = "Moment(3)",
           plotLayout     = "line",
           adjustments    = list (purpose="difference",decorrelation="CM"),
           errorbar       = "CIwithDF",    # NEW: change the CI computation
           gamma          = c(0.95, 10-1), # NEW: specify explicitely the unpooled df
           errorbarParams = list(color="red", width= 0.1, direction = "left",
                                   position = position_nudge(-0.05) )
        )
# Third, a plot where the pooled df are explicitely set
plt3 <- superb(
           cbind(DV.1,DV.2,DV.3) ~ .,
           dta,
           WSFactors      = "Moment(3)",
           plotLayout     = "line",
           adjustments    = list (purpose="difference",decorrelation="CM"),
           errorbar       = "CIwithDF",         # NEW: again, change the CI computation
           gamma          = c(0.95, 30-3-10+1), # NEW: this time, specify the pooled df
           errorbarParams = list(color="blue", width= 0.1, direction = "right",
                                 position = position_nudge(+0.05) )
        )
# Convert the plots into grapphical objects all with the same scale...
plt1b <- ggplotGrob(plt1 + ylim(-1.65,1.65) )
plt2b <- ggplotGrob(plt2 + ylim(-1.65,1.65) + makeTransparent() )
plt3b <- ggplotGrob(plt3 + ylim(-1.65,1.65) + makeTransparent() )
```

```
# ... and superimpose these grobs onto an empty ggplot
ggplot() +
    annotation_custom(grob=plt1b) +
    annotation_custom(grob=plt2b) +
    annotation_custom(grob=plt3b)

# As seen and as expected, the orange and red bars are identical;
# The blue bars, based on the (correct) pooled degree of freedom
# are just a little bit smaller because the pooled df are larger.
# However, the difference is visible only because the group size
# is ridiculously small (10 participants only).

# ====================================================
# AN EXAMPLE with heterogeneous variance
# ====================================================

# We create simulated scores with a large amount of heterogeneity
dta <- GRD(
     BSFactors = "Group(3)",
     SubjectsPerGroup = 10,
     Population = list(
         mean   = 100, # will set GM to 100
         stddev = 15,  # will set STDDEV to 15
         scores = "rnorm(1, mean = GM, sd = STDDEV*Group)"
     )
 )

# This computes the Welch's degree of freedom
wdf <- WelchDegreeOfFreedom(dta, "DV", "Group" )
wdf # should be between n-1 and N-n-p+1.

# A regular plot
plt1 <- superb(
           DV ~ Group,
           dta,
           plotLayout     = "line",
           adjustments    = list (purpose="difference"),
           errorbar       = "CI",
           gamma          = 0.95,
           errorbarParams = list(color="orange", width= 0.1, direction = "both",
                                 position = position_nudge(-0.0) )
        )
# Second, a plot where the df are set to the pooled df
plt2 <- superb(
           DV ~ Group,
           dta,
           plotLayout     = "line",
           adjustments    = list (purpose="difference"),
           errorbar       = "CIwithDF",         # NEW: change the CI computation
          gamma         = c(0.95, 30-10-3+1), # NEW: specify explicitely the unpooled df
           errorbarParams = list(color="red", width= 0.1, direction = "left",
                                 position = position_nudge(-0.05) )
        )
```

```
# Third, a plot where the pooled df are explicitely set
plt3 <- superb(
           DV ~ Group,
           dta,
           plotLayout     = "line",
           adjustments    = list (purpose="difference"),
           errorbar       = "CIwithDF",       # NEW: again, change the CI computation
           gamma          = c(0.95, wdf),     # NEW: this time, specify the pooled df
           errorbarParams = list(color="blue", width= 0.1, direction = "right",
                                 position = position_nudge(+0.05) )
       )
# Convert the plots into grapphical objects all with the same scale...
plt1b <- ggplotGrob(plt1 + ylim(25,175) )
plt2b <- ggplotGrob(plt2 + ylim(25,175) + makeTransparent() )
plt3b <- ggplotGrob(plt3 + ylim(25,175) + makeTransparent() )

# ... and superimpose these grobs onto an empty ggplot
ggplot() +
    annotation_custom(grob=plt1b) +
    annotation_custom(grob=plt2b) +
    annotation_custom(grob=plt3b)

# As seen, the Welch's corrected df results in error bars (blue) which are
# just a little bit longer than the pooled df bars (red). In all cases, the
# unadjusted (default) error bars (n-1) are longer (orange bars), resulting in a more
# conservative representation of the data.
```

---

runDebug *runDebug*

Description

runDebug is an internal function used by GRD and superb to help in debugging the functions. It assigns in the global environment the variables that are local to a function so that they become visible. Use `options("superb.feedback" = "all")` to turn all debug on.

The default values to the option "superb.feedback" is c("design","warnings","summary","experimental") where the four entries allow:

- design: (in superbPlot) showing information on how the within-subject variables are understood;
- warnings: (in superbPlot) returning 'FYI' messages about the data to help decide if the appropriate error bars were used
- summary: (in GRD) showing a recapitulation of the design;
- experimental: (in superbPlot) showing a message if experimental features are used. This last feedback information is currently not inhibited but will be in future versions.

Usage

```
runDebug(where, title, vars, vals)
```

Arguments

where	indicates where in the program runDebug was called
title	string text to be displayed when this function is triggered
vars	strings names of the variables to be placed in the global environment
vals	numeric values to be given to the variables.

Value

puts in the globalenvironment the variables named "vars"

---

showSignificance *Annotate significance of results on plots*

Description

showSignificance() is used to add an annotation to a ggplot in the form of a square bracket with a text. The bracket extends from x range (left, right) with a heigth of width. It is also possible to have the bracket and the text vertical when y is a range (bottom, top).

Usage

```
showSignificance(
  x,
  y,
  width,
  text = NULL,
  panel = list(),
  segmentParams = list(),
  textParams = list()
)
```

Arguments

x	(a vector of 2 when horizontal) indicates the limits of the annotation;
y	(a vector of 2 when vertical) the location of the annotation in the y direction
width	height of the annotation; for negative width, the legs extends towards the bottom;
text	(optional) string text to be display on the opposite side of width;
panel	(optional) a list to identify in which panel to put the annotation;
segmentParams	(optional) a list of directives that will be sent to the geom_segment() items;
textParams	(optional) a list of directives that will be sent to the geom_text() item.

Value

adds an annotation in a ggplot

Examples

```
# loading required libraries
library(superb)
library(ggplot2)
library(grid)

# making one random data set with three factors 2 x 3 x (3)
dta <- GRD(
    SubjectsPerGroup = 20,
    BSFactors = c("Group(2)","Age(3)"),
    WSFactors = c("Moment(3)"),
    Population = list(mean = 75, stddev = 5),
    Effects   = list("Group" = slope(10) )
)

# making a two-factor plot and a three-factor plots (having panels)
plt2 <- superb(
        cbind(DV.1,DV.2,DV.3) ~ Group,
        dta,
        WSFactor = c("Moment(3)"),
        adjustments = list(purpose="difference"),
        plotLayout  = "bar",
        factorOrder = c("Moment","Group")
    )
plt3 <- superb(
        cbind(DV.1,DV.2,DV.3) ~ Group + Age,
        dta,
        WSFactor = c("Moment(3)"),
        adjustments = list(purpose="difference"),
        plotLayout  = "bar",
        factorOrder = c("Moment","Group","Age")
    )

# lets decorate these plots a bit...
plt2 <- plt2 +  scale_fill_manual( name = "Group",
        labels = c("Easy", "Hard"),
        values = c("blue", "purple")) +
  scale_colour_manual( name = "Group",
        labels = c("Easy", "Hard"),
        values = c("blue", "purple")) +
  coord_cartesian( ylim = c(50,100), xlim = c(0.5, 3.9) )
plt3 <- plt3 +  scale_fill_manual( name = "Group",
        labels = c("Easy", "Hard"),
        values = c("blue", "purple")) +
  scale_colour_manual( name = "Group",
        labels = c("Easy", "Hard"),
        values = c("blue", "purple")) +
  coord_cartesian( ylim = c(50,105) )
```

```
# a very basic example
plt2 +  showSignificance( c(0.75, 1.25), 90, -1, "++1++")

# the annotation can be vertical when y is a vector with bottom and top location:
plt2 + showSignificance( 3.75, c(70,80), -0.1, "++1++")

# an example with panels; the "panel" argument is used to identify on
# which panel to put the annotation (or else they appear on all panels)
# and with arms of differing lengths, and one flat ending
plt3 +
    showSignificance( c(0.75, 1.25), 90, -2.5,      "++1++", panel = list(Age= 1)) +
    showSignificance( c(1.75, 2.25), 90, -2.5,      "++2++", panel = list(Age= 2)) +
    showSignificance( c(0.75, 1.25), 90, c(-10,-5), "++3++", panel = list(Age= 3)) +
    showSignificance( c(2.00, 3.25), 95, -10,       "++4++", panel = list(Age= 3)) +
    showSignificance( c(1.75, 2.25), 85, 0,                  panel = list(Age= 3))

# here, we send additional directives to the annotations
plt3 +
    showSignificance( c(0.75, 1.25), 90, -5,  "++1++", panel = list(Age= 1)) +
    showSignificance( c(1.75, 2.25), 95, -10, "++2++", panel = list(Age = 2),
        textParams    = list(size = 3,              # smaller font
                             family  = "mono",      # courrier font
                             colour= "chartreuse3"   # dark green color
        ),
        segmentParams = list(linewidth = 1.,             # thicker lines
                             arrow   = arrow(length = unit(0.2, "cm") ), # arrow heads
                             colour = "chartreuse3"  # dark green color as well
        )
    ) +
    showSignificance( c(1.75, 3.25), 95, -30, "++3++", panel = list(Age = 3),
        textParams    = list(size = 5,              # larger font
                             family  = "serif",      # times font
                             alpha = 0.3 ),          # transparent
        segmentParams = list(linewidth = 2.,
                             arrow   = arrow(length = unit(0.2, "cm") ),
                             alpha = 0.3,
                             lineend = "round"       # so that line end overlap nicely
        )
    )
```

ShroutFleissICC1 *Shrout and Fleiss intra-class correlation functions*

Description

The functions ShroutFleissICC1, ShroutFleissICC11 and ShroutFleissICC1k computes the intra-class correlation ICC for a given data frame containing repeated measures in columns cols when the measures are in distinct clusters, identified in column clustercol. See (Shrout and Fleiss 1979).

Usage

```
ShroutFleissICC1(dta, clustercol, cols)
```

Arguments

dta	A data frame containing within-subject measures, one participant per line;
clustercol	is the column index where cluster belonging are given;
cols	A vector indicating the columns containing the measures.

Value

ICC the intra-class measure of association.

References

Shrout PE, Fleiss JL (1979). "Intraclass correlations: uses in assessing rater reliability." *Psychological bulletin*, **86**(2), 420.

Shrout PE, Fleiss JL (1979). "Intraclass correlations: uses in assessing rater reliability." *Psychological bulletin*, **86**(2), 420.

Examples

```
# creates a small data frames with 4 subject's scores for 5 measures:
dta <- data.frame(cbind(
        clus <- c(1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3),
        col1 <- c(2, 4, 4, 6, 4, 5, 8, 8, 5, 8, 9, 9)
    ))

ShroutFleissICC1(dta, 1, 2)
# 0.434343434
ShroutFleissICC11(dta[, 1], dta[,2])
# 0.434343434

dta2 <- data.frame(cbind(
        clus <- c(1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3),
        col1 <- c(1, 3, 3, 5, 3, 4, 7, 7, 4, 7, 8, 8),
        col1 <- c(2, 4, 4, 6, 4, 5, 8, 8, 5, 8, 9, 9),
        col1 <- c(3, 5, 5, 7, 5, 6, 9, 9, 6, 9, 10, 10)
    ))

ShroutFleissICC1(dta2, 1, 2:4)
# 0.7543859649
ShroutFleissICC1k(dta2[, 1], dta2[,2:4])
# 0.7543859649
```

slope *Effect description*

Description

There is four ways that effects can be defined in GRD. ‘"factor" = slope(s)‘ will vary the means by an amount of s for each step of the factor; ‘"factor" = extent(s)‘ will vary the means uniformly so that there is a difference of s between the first and the last factor level; ‘"factor" = custom(a,b,c..)‘ will alter each means by an amount of a for the first, b for the second, etc. Finally ‘"factor" = Rexpression("R code")‘ will apply R code to all levels of the factors, altering the base mean.

Usage

```
slope(s)

extent(s)

custom(...)

Rexpression(str)
```

Arguments

s	the size of the effect
...	a sequence with the sizes of the effects
str	R code string

Value

These internal functions are not meant to be used in isolation in any meaningful way...

subjectCenteringTransform *subject-centering transform*

Description

`subjectCenteringTransform()` is a transformation that can be applied to a matrix of data. the resulting matrix have means that are centered on the grand mean, subject-wise (Cousineau 2005).

Usage

```
subjectCenteringTransform(dta, variables)
```

Arguments

`dta`	a data.frame containing the data in wide format;
`variables`	a vector of column names on which the transformation will be applied. the remaining columns will be left unchanged

Value

a data.frame of the same form as dta with the variables transformed.

This function is useful when passed to the argument `preprocessfct` of `superb()` where it performs a modification of the data matrix.

References

Cousineau D (2005). "Confidence intervals in within-subject designs: A simpler solution to Loftus and Masson's method." *Tutorials in Quantitative Methods for Psychology*, **1**, 42 – 45. doi:10.20982/tqmp.01.1.p042.

`summaryStatistics` *Additional summary statistics*

Description

`superb` adds a few summary statistics that can be used to characterize a dataset. All comes with `SE.fct()` and `CI.fct()`. See (Harding et al. 2014; Harding et al. 2015) for more. *superb-compatible* summary statistics functions must have a single vector parameter:

Usage

```
hmean(x)

gmean(x)

MAD(x)

fisherskew(x)

pearsonskew(x)

fisherkurtosis(x)
```

Arguments

`x`	a vector of numbers, the sample data (mandatory);

Value

a summary statistic describing the sample.

References

Harding B, Tremblay C, Cousineau D (2014). "Standard errors: A review and evaluation of standard error estimators using Monte Carlo simulations." *The Quantitative Methods for Psychology*, **10**(2), 107–123.

Harding B, Tremblay C, Cousineau D (2015). "The standard error of the Pearson skew." *The Quantitative Methods for Psychology*, **11**(1), 32–36.

Examples

```
# the confidence interval of the mean for default 95% and 90% confidence level
gmean( c(1,2,3) )  # the geometric mean; also available in psych::geometric.mean
hmean( c(1,2,3) )  # the harmonic mean;  also available in psych::harmonic.mean
MAD( c(1,2,3) )    # the median absolute deviation to the median (not the same as mad)
fisherskew( c(1,2,3) )     # the Fisher skew corrected for sample size
fisherkurtosis( c(1,2,3) ) # the Fisher kurtosis corrected for sample size
pearsonskew( c(1,2,3) )    # the Pearson skew
```

`superb` *superb*

Description

The function `superb()` plots standard error or confidence interval for various descriptive statistics under various designs, sampling schemes, population size and purposes, according to the `superb` framework. See (Cousineau et al. 2021) for more. The functions `superb()` is now the entry point to realize summary plots. Compared to the previously documented `superbPlot()`, `superb()` is based on formula and accept long and wide format.

Usage

```
superb(
  formula,
  data,
  WSFactors = NULL,
  WSDesign = NULL,
  factorOrder = NULL,
  statistic = "mean",
  errorbar = "CI",
  gamma = 0.95,
  adjustments = list(purpose = "single", popSize = Inf, decorrelation = "none",
    samplingDesign = "SRS"),
  showPlot = TRUE,
  plotStyle = NULL,
  plotLayout = "line",
  preprocessfct = NULL,
```

```
  postprocessfct = NULL,
  clusterColumn = NULL,
  ...
)
```

Arguments

formula	a formula describing the design of the data frame
data	Dataframe in wide or long format
WSFactors	The name of the within-subject factor(s)
WSDesign	the within-subject design if not a full factorial design (default "fullfactorial")
factorOrder	Order of factors as shown in the graph (in that order: x axis, groups, horizontal panels, vertical panels)
statistic	The summary statistic function to use as a string
errorbar	The function that computes the error bar. Should be "CI" or "SE" or any function name if you defined a custom function (see `vignette("Vignette4", package = "superb")`). Default to "CI"
gamma	The coverage factor; necessary when `errorbar == "CI"`. Default is 0.95.
adjustments	List of adjustments as described below. Default is `adjustments = list(purpose = "single", decorrelation = "none", samplingDesign = "SRS", popSize = Inf)`
showPlot	Defaults to TRUE. Set to FALSE if you want the output to be the summary statistics and intervals.
plotStyle	**[Deprecated]**
plotLayout	The type of object to plot on the graph. See full list below. Defaults to "line".
preprocessfct	is a transform (or vector of) to be performed first on data matrix of each group
postprocessfct	is a transform (or vector of)
clusterColumn	used in conjunction with samplingDesign = "CRS", indicates which column contains the cluster membership
...	In addition to the parameters above, superbPlot also accept a number of optional arguments that will be transmitted to the plotting function, such as `pointParams` (a list of ggplot2 parameters to input inside geoms; see `?geom_bar2`) and `errorbarParams` (a list of ggplot2 parameters for geom_errorbar; see `?geom_errorbar`)

Details

The possible adjustements are the following

- `purpose`: The purpose of the comparisons. Defaults to "single". Can be "single", "difference", or "tryon".
- `decorrelation`: Decorrelation method for repeated measure designs. Chooses among the methods "CM", "LM", "CA", "UA", "LDr" (with *r* a positive integer) or "none". Defaults to "none". "CA" is correlation-adjusted (Cousineau 2019); "CM" is Cousineau-Morey (Baguley 2012); "LM" is Loftus and Masson (Loftus and Masson 1994); "UA" is based on the unitary

Alpha method (derived from the Cronbach alpha; see (Laurencelle and Cousineau 2023)). "LDr" is local decorrelation (useful for long time series with autoregressive correlation structures; see (Cousineau et al. 2024)).

- `popsize`: Size of the population under study. Defaults to Inf
- `samplingDesign`: Sampling method to obtain the sample. implemented sampling is "SRS" (Simple Randomize Sampling) and "CRS" (Cluster-Randomized Sampling).

The formulas can be for long format data using | notation, e.g.,

- `superb( extra ~ group | ID, sleep )`

or for wide format, using cbind() or crange() notation, e.g.,

- `superb( cbind(DV.1.1, DV.2.1,DV.1.2, DV.2.2,DV.1.3, DV.2.3) ~ . , dta, WSFactors = c("a(2)","b(3)"))`
- `superb( crange(DV.1.1, DV.2.3) ~ . , dta, WSFactors = c("a(2)","b(3)"))`

The avaialble `plotLayout` are the following:

- These are basic plots:
  - "bar" Shows the summary statistics with bars and error bars;
  - "line" Shows the summary statistics with lines connecting the conditions over the first factor;
  - "point" Shows the summary statistics with isolated points
  - "lineband" illustrates the confidence intervals as a band;
- These plots add distributional information in addition
  - "pointjitter" Shows the summary statistics along with jittered points depicting the raw data;
  - "pointjitterviolin" Also adds violin plots to the previous layout
  - "pointindividualline" Connects the raw data with line along the first factor (which should be a repeated-measure factor)
  - "raincloud" Illustrates the distribution with a cloud (half_violin_plot) and jittered dots next to it. Looks better when coordinates are flipped `+coord_flip()`
  - "corset" illustrates within-subject designs with individual lines and clouds.
- Circular plots (aka radar plots) results from the following layouts:
  - "circularpoint" Shows the summary statistics with isolated points
  - "circularline" Shows the summary statistics with lines;
  - "circularlineband" Also adds error bands instead of error bars;
  - "circularpointjitter" Shows summary statistics and error bars but also jittered dots;
  - "circularpointlinejitter" Same as previous layout, but connect the points with lines. New layouts are added from times to time. Personalized layouts can also be created (see `vignette("Vignette5", package = "superb")`)

Value

a plot with the correct error bars or a table of those summary statistics. The plot is a ggplot2 object with can be modified with additional declarations.

References

Baguley T (2012). "Calculating and graphing within-subject confidence intervals for ANOVA." *Behavior Research Methods*, **44**, 158 – 175. doi:10.3758/s1342801101237.

Cousineau D (2019). "Correlation-adjusted standard errors and confidence intervals for within-subject designs: A simple multiplicative approach." *The Quantitative Methods for Psychology*, **15**, 226 – 241. doi:10.20982/tqmp.15.3.p226.

Cousineau D, Goulet M, Harding B (2021). "Summary plots with adjusted error bars: The superb framework with an implementation in R." *Advances in Methods and Practices in Psychological Science*, **4**, 1–18. doi:10.1177/25152459211035109.

Cousineau D, Proulx A, Potvin-Pilon A, Fiset D (2024). "Local decorrelation for error bars in time series." *The Quantitative Methods for Psychology*, **20**(2), 173-185. doi:10.20982/tqmp.20.2.p173.

Laurencelle L, Cousineau D (2023). "Analysis of proportions using arcsine transform with any experimental design." *Frontiers in Psychology*, **13**, 1045436. doi:10.3389/fpsyg.2022.1045436.

Loftus GR, Masson MEJ (1994). "Using confidence intervals in within-subject designs." *Psychonomic Bulletin & Review*, **1**, 476 – 490. doi:10.3758/BF03210951.

Examples

```
# Basic example using a built-in dataframe as data.
# By default, the mean is computed and the error bar are 95% confidence intervals
superb(len ~ dose + supp, ToothGrowth)

# Example changing the summary statistics to the median and
# the error bar to 80% confidence intervals
superb(len ~ dose + supp, ToothGrowth,
       statistic = "median", errorbar = "CI", gamma = .80)

# Example introducing adjustments for pairwise comparisons
# and assuming that the whole population is limited to 200 persons
superb(len ~ dose + supp, ToothGrowth,
  adjustments = list( purpose = "difference", popSize = 200) )

# This example adds ggplot directives to the plot produced
library(ggplot2)
superb(len ~ dose + supp, ToothGrowth) +
xlab("Dose") + ylab("Tooth Growth") +
theme_bw()



# The following examples are based on repeated measures
library(gridExtra)
options(superb.feedback = 'none') # shut down 'warnings' and 'design' interpretation messages

# A simple example: The sleep data
```

```
# The sleep data are paired data showing the additional time of sleep with
# the soporific drug #1 (("group = 1") and with the soporific drug #2 ("group = 2").
# There is 10 participants with two measurements.
# sleep is available in long format

# Makes the plots first without decorrelation:
superb( extra ~ group | ID, sleep )
# As seen the error bar are very long. Lets take into consideration correlation...
# ...  with decorrelation (technique Correlation-adjusted CA):
superb(extra ~ group | ID, sleep,
  # only difference:
  adjustments = list(purpose = "difference", decorrelation = "CA")
)
# The error bars shortened as the correlation is substantial (r = .795).




# Another example: The Orange data
# This example contains 5 trees whose diameter (in mm) has been measured at various age (in days):
data(Orange)

# Makes the plots first without decorrelation:
p1 <- superb( circumference ~ age | Tree, Orange,
  adjustments = list(purpose = "difference", decorrelation = "none")
) +
  xlab("Age level") + ylab("Trunk diameter (mm)") +
  coord_cartesian( ylim = c(0,250) ) + labs(title="''Standalone'' confidence intervals")
# ... and then with decorrelation (technique Correlation-adjusted CA):
p2 <- superb( circumference ~ age | Tree, Orange,
  adjustments = list(purpose = "difference", decorrelation = "CA")
) +
  xlab("Age level") + ylab("Trunk diameter (mm)") +
  coord_cartesian( ylim = c(0,250) ) + labs(title="Decorrelated confidence intervals")

# You can present both plots side-by-side
grid.arrange(p1, p2, ncol=2)
```

---

superbData *Obtain summary statistics with correct error bars.*

Description

The function suberbData() computes standard error or confidence interval for various descriptive statistics under various designs, sampling schemes, population size and purposes, according to the

suberb framework. See (Cousineau et al. 2021) for more. The functions `superb(..., showPlot = FALSE)` is now the entry point to obtain the summary statistics and error bar as numbers.

Usage

```
superbData(
  data,
  BSFactors = NULL,
  WSFactors = NULL,
  WSDesign = "fullfactorial",
  factorOrder = NULL,
  variables,
  statistic = "mean",
  errorbar = "CI",
  gamma = 0.95,
  adjustments = list(purpose = "single", popSize = Inf, decorrelation = "none",
    samplingDesign = "SRS"),
  preprocessfct = NULL,
  postprocessfct = NULL,
  clusterColumn = ""
)
```

Arguments

`data`	Dataframe in wide format
`BSFactors`	The name of the columns containing the between-subject factor(s)
`WSFactors`	The name of the within-subject factor(s)
`WSDesign`	the within-subject design if not a full factorial design (default "fullfactorial")
`factorOrder`	Order of factors as shown in the graph (x axis, groups, horizontal panels, vertical panels)
`variables`	The dependent variable(s)
`statistic`	The summary statistic function to use
`errorbar`	The function that computes the error bar. Should be "CI" or "SE" or any function name. Defaults to "CI"
`gamma`	The coverage factor; necessary when errorbar == "CI". Default is 0.95.
`adjustments`	List of adjustments as described below. Default is `adjustments = list(purpose = "single", popSize = Inf, decorrelation = "none", samplingDesign = "SRS")`
`preprocessfct`	is a transform (or vector of) to be performed first on data matrix of each group
`postprocessfct`	is a transform (or vector of)
`clusterColumn`	used in conjunction with samplingDesign = "CRS", indicates which column contains the cluster membership

Details

The possible adjustements are the following

- popsize: Size of the population under study. Defaults to Inf
- purpose: The purpose of the comparisons. Defaults to "single". Can be "single", "difference", or "tryon".
- decorrelation: Decorrelation method for repeated measure designs. Chooses among the methods "CM", "LM", "CA" or "none". Defaults to "none".
- samplingDesign: Sampling method to obtain the sample. implemented sampling is "SRS" (Simple Randomize Sampling) and "CRS" (Cluster-Randomized Sampling).

Value

a list with (1) the summary statistics in summaryStatistics (2) the raw data in long format in rawData (using numeric levels for repeated-measure variables).

References

Cousineau D, Goulet M, Harding B (2021). "Summary plots with adjusted error bars: The superb framework with an implementation in R." *Advances in Methods and Practices in Psychological Science*, **4**, 1–18. doi:10.1177/25152459211035109.

Examples

```
# Basic example using a built-in dataframe as data;
# by default, the mean is computed and the error bar are 95% confidence intervals
# (it also produces a $rawData dataframe, not shown here)
res <- superbData(ToothGrowth, BSFactors = c("dose", "supp"),
  variables = "len")
res$summaryStatistics

# Example introducing adjustments for pairwise comparisons
# and assuming that the whole population is limited to 200 persons
res <- superbData(ToothGrowth, BSFactors = c("dose", "supp"),
  variables = "len",
  statistic = "median", errorbar = "CI", gamma = .80,
  adjustments = list( purpose = "difference", popSize = 200) )
res$summaryStatistics

# Note that you can achieve the same with formulas
superb( len ~ dose + supp, ToothGrowth, showPlot=FALSE)
```

superbPlot *summary plot of any statistics with adjusted error bars.*

Description

The function superbPlot() plots standard error or confidence interval for various descriptive statistics under various designs, sampling schemes, population size and purposes, according to the suberb framework. See (Cousineau et al. 2021) for more. Note that this function has been superseded by superb().

Usage

```
superbPlot(
  data,
  BSFactors = NULL,
  WSFactors = NULL,
  WSDesign = "fullfactorial",
  factorOrder = NULL,
  variables,
  statistic = "mean",
  errorbar = "CI",
  gamma = 0.95,
  adjustments = list(purpose = "single", popSize = Inf, decorrelation = "none",
    samplingDesign = "SRS"),
  showPlot = TRUE,
  plotStyle = NULL,
  plotLayout = "line",
  preprocessfct = NULL,
  postprocessfct = NULL,
  clusterColumn = "",
  ...
)
```

Arguments

data	Dataframe in wide format
BSFactors	The name of the columns containing the between-subject factor(s)
WSFactors	The name of the within-subject factor(s)
WSDesign	the within-subject design if not a full factorial design (default "fullfactorial")
factorOrder	Order of factors as shown in the graph (in that order: x axis, groups, horizontal panels, vertical panels)
variables	The dependent variable(s) as strings
statistic	The summary statistic function to use as a string.
errorbar	The function that computes the error bar. Should be "CI" or "SE" or any function name if you defined a custom function. Default to "CI"

gamma	The coverage factor; necessary when errorbar == "CI". Default is 0.95.
adjustments	List of adjustments as described below. Default is adjustments = list(purpose = "single", popSize = Inf, decorrelation = "none", samplingDesign = "SRS")
showPlot	Defaults to TRUE. Set to FALSE if you want the output to be the summary statistics and intervals.
plotStyle	**[Deprecated]**
plotLayout	The layers to plot on the graph. See full list below. Defaults to "line".
preprocessfct	is a transform (or vector of) to be performed first on data matrix of each group
postprocessfct	is a transform (or vector of)
clusterColumn	used in conjunction with samplingDesign = "CRS", indicates which column contains the cluster membership
...	In addition to the parameters above, superbPlot also accept a number of optional arguments that will be transmitted to the plotting function, such as pointParams (a list of gglot2 parameters to input inside geoms; see ?geom_bar2) and errorbarParams (a list of ggplot2 parameters for geom_errorbar; see ?geom_errorbar)

Details

The possible adjustements are the following

- popsize: Size of the population under study. Defaults to Inf
- purpose: The purpose of the comparisons. Defaults to "single". Can be "single", "difference", or "tryon".
- decorrelation: Decorrelation method for repeated measure designs. Chooses among the methods "CM", "LM", "CA", "UA", "LDr" (with r an integer) or "none". Defaults to "none". "CA" is correlation-adjusted (Cousineau 2019); "UA" is based on the unitary Alpha method (derived from the Cronbach alpha; see (Laurencelle and Cousineau 2023)). "LDr" is local decorrelation (useful for long time series with autoregressive correlation structures; see (Cousineau et al. 2024)); .
- samplingDesign: Sampling method to obtain the sample. implemented sampling is "SRS" (Simple Randomize Sampling) and "CRS" (Cluster-Randomized Sampling).

As of version 0.97.15, the layouts for plots are the following:

- "bar" Shows the summary statistics with bars and error bars;
- "line" Shows the summary statistics with lines connecting the conditions over the first factor;
- "point" Shows the summary statistics with isolated points
- "pointjitter" Shows the summary statistics along with jittered points depicting the raw data;
- "pointjitterviolin" Also adds violin plots to the previous layout
- "pointindividualline" Connects the raw data with line along the first factor (which should be a repeated-measure factor)
- "raincloud" Illustrates the distribution with a cloud (half_violin_plot) and jittered dots next to it. Looks better when coordinates are flipped +coord_flip().
- "corset" Illustrates two repeated-measures with individual lines and clouds

- "boxplot" Illustrates the limits, the quartiles and the median using a box

but refer to `superb()` for a documentation that will be kept up do date.

EXPERIMENTAL: if both the statistic function and its related errorbar function are in a different namespace, you may use the notation "namespace::funcname" in the `statistic` argument. The same namespace will be used for the errorbar function.

Value

a plot with the correct error bars or a table of those summary statistics. The plot is a ggplot2 object with can be modified with additional declarations.

References

Cousineau D (2019). "Correlation-adjusted standard errors and confidence intervals for within-subject designs: A simple multiplicative approach." *The Quantitative Methods for Psychology*, **15**, 226 – 241. doi:10.20982/tqmp.15.3.p226.

Cousineau D, Goulet M, Harding B (2021). "Summary plots with adjusted error bars: The superb framework with an implementation in R." *Advances in Methods and Practices in Psychological Science*, **4**, 1–18. doi:10.1177/25152459211035109.

Cousineau D, Proulx A, Potvin-Pilon A, Fiset D (2024). "Local decorrelation for error bars in time series." *The Quantitative Methods for Psychology*, **20**(2), 173-185. doi:10.20982/tqmp.20.2.p173.

Laurencelle L, Cousineau D (2023). "Analysis of proportions using arcsine transform with any experimental design." *Frontiers in Psychology*, **13**, 1045436. doi:10.3389/fpsyg.2022.1045436.

Examples

```
# Basic example using a built-in dataframe as data.
# By default, the mean is computed and the error bar are 95% confidence intervals
superbPlot(ToothGrowth, BSFactors = c("dose", "supp"),
  variables = "len")

# Note that function superb() does the same with formula:
superb( len ~ dose + supp, ToothGrowth )

# Example changing the summary statistics to the median and
# the error bar to 80% confidence intervals
superbPlot(ToothGrowth, BSFactors = c("dose", "supp"),
  variables = "len", statistic = "median", errorbar = "CI", gamma = .80)

# Example introducing adjustments for pairwise comparisons
# and assuming that the whole population is limited to 200 persons
superbPlot(ToothGrowth, BSFactors = c("dose", "supp"),
  variables = "len",
  adjustments = list( purpose = "difference", popSize = 200) )

# This example adds ggplot directives to the plot produced
library(ggplot2)
```

```
superbPlot(ToothGrowth, BSFactors = c("dose", "supp"),
  variables = "len") +
xlab("Dose") + ylab("Tooth Growth") +
theme_bw()



# The following examples are based on repeated measures
library(gridExtra)
options(superb.feedback = 'none') # shut down 'warnings' and 'design' interpretation messages

# A simple example: The sleep data
# The sleep data are paired data showing the additional time of sleep with
# the soporific drugn #1 (("group = 1") and with the soporific drug #2 ("group = 2").
# There is 10 participants with two measurements.

# sleep is available in long format so we transform it to the in wide format:
sleep2 <- reshape(sleep, direction = "wide", idvar = "ID", timevar = "group")
sleep2

# Makes the plots first without decorrelation:
superbPlot(sleep2,
  WSFactors = "Times(2)",
  variables = c("extra.1", "extra.2")
)
# As seen the error bar are very long. Lets take into consideration correlation...
# ...  with decorrelation (technique Correlation-adjusted CA):
superbPlot(sleep2,
  WSFactors = "Times(2)",
  variables = c("extra.1", "extra.2"),
  # only difference:
  adjustments = list(purpose = "difference", decorrelation = "CA")
)
# The error bars shortened as the correlation is substantial (r = .795).



# Another example: The Orange data
data(Orange)
# Use the Orange example, but let's define shorter column names...
names(Orange) <- c("Tree","age","circ")
# ... and turn the data into a wide format using superbToWide:
Orange.wide <- superbToWide(Orange, id = "Tree", WSFactors = "age", variable = "circ")

# This example contains 5 trees whose diameter (in mm) has been measured at various age (in days):
Orange.wide

# Makes the plots first without decorrelation:
p1 <- superbPlot( Orange.wide, WSFactors = "age(7)",
 variables = c("circ.118","circ.484","circ.664","circ.1004","circ.1231","circ.1372","circ.1582"),
  adjustments = list(purpose = "difference", decorrelation = "none")
) +
```

```
  xlab("Age level") + ylab("Trunk diameter (mm)") +
  coord_cartesian( ylim = c(0,250) ) + labs(title="''Standalone'' confidence intervals")
# ... and then with decorrelation (technique Correlation-adjusted CA):
p2 <- superbPlot( Orange.wide, WSFactors = "age(7)",
 variables = c("circ.118","circ.484","circ.664","circ.1004","circ.1231","circ.1372","circ.1582"),
  adjustments = list(purpose = "difference", decorrelation = "CA")
) +
  xlab("Age level") + ylab("Trunk diameter (mm)") +
  coord_cartesian( ylim = c(0,250) ) + labs(title="Decorrelated confidence intervals")

# You can present both plots side-by-side
grid.arrange(p1, p2, ncol=2)
```

`superbPlot.bar` *superbPlot 'bar' layout*

Description

superbPlot comes with a few built-in templates for making the final plots. All produces ggplot objects that can be further customized. Additionally, it is possible to add custom-make templates (see vignette 6). The functions, to be "superbPlot-compatible", must have these parameters:

Usage

```
superbPlot.bar(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  barParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  xAsFactor = TRUE
)
```

Arguments

`summarydata`	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
`xfactor`	a string with the name of the column where the factor going on the horizontal axis is given;
`groupingfactor`	a string with the name of the column for which the data will be grouped on the plot;

addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors
barParams	(optional) list of graphic directives that are sent to the geom_bar layer
errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer
xAsFactor	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

Examples

```
# This will make a plot with bars
superb(
   len ~ dose + supp,
   ToothGrowth,
   plotLayout="bar"
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superb(
#   len ~ dose + supp,
#   ToothGrowth,
#   showPlot = FALSE
#)
#
#superbPlot.bar(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

---

superbPlot.boxplot *superbPlot 'boxplot' layout*

Description

superbPlot comes with a few built-in templates for making the final plots. All produces ggplot objects that can be further customized. Additionally, it is possible to create custom-make templates (see vignette 5). The functions, to be "superbPlot-compatible", must have these parameters:

Usage

```
superbPlot.boxplot(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  pointParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  boxplotParams = list(),
  xAsFactor = TRUE
)
```

Arguments

summarydata	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
xfactor	a string with the name of the column where the factor going on the horizontal axis is given;
groupingfactor	a string with the name of the column for which the data will be grouped on the plot;
addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors;
pointParams	(optional) list of graphic directives that are sent to the geom_bar layer;
errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer;
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer;
boxplotParams	(optional) list f graphic directives that are sent to the geo_boxplot layer;
xAsFactor	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete).

Value

a ggplot object

Examples

```
# This will make a plot with boxes for interquartile (box), median (line) and outliers (whiskers)
superb(
   len ~ dose + supp,
   ToothGrowth,
   plotLayout = "boxplot"
)
```

```
# This layout of course is more meaningful if the statistic displayed is the median
superb(
   len ~ dose + supp,
   ToothGrowth,
   statistic = "median",
   plotLayout = "boxplot"
)

# if you extracted the data with superbData, you can
# run this layout directly
processedData <- superb(
   len ~ dose + supp,
   ToothGrowth,
   statistic = "median",
   showPlot  = FALSE
)

superbPlot.boxplot(processedData$summaryStatistic,
   "dose", "supp", ".~.",
   processedData$rawData)

# This will make a plot with customized boxplot parameters and black dots
superb(
   len ~ dose + supp,
   ToothGrowth,
   statistic = "median",
   plotLayout = "boxplot",
   boxplotParams = list( outlier.shape=8, outlier.size=4 ),
   pointParams = list(color="black")
)

# You can customize the plot in various ways, e.g.
plt3 <- superb(
   len ~ dose + supp,
   ToothGrowth,
   statistic = "median",
   plotLayout = "boxplot",
   pointParams = list(color="black")
)

# ... by changing the colors of the fillings
library(ggplot2) # for scale_fill_manual, geom_jitter and geom_dotplot
plt3 + scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# ... by overlaying jittered dots of the raw data
plt3 + geom_jitter(data = processedData$rawData, mapping=aes(x=dose, y=DV),
   position= position_jitterdodge(jitter.width=0.5 , dodge.width=0.8 ) )

# ... by overlaying dots of the raw data, aligned along the center of the box
plt3 + geom_dotplot(data = processedData$rawData, mapping=aes(x=dose, y=DV), dotsize=0.5,
   binaxis='y', stackdir='center', position=position_dodge(0.7))
```

`superbPlot.circularline` *superbPlot 'circularline' layout*

Description

superb comes with a few circular layouts for making plots. It produces ggplot objects that can be further customized.

Usage

```
superbPlot.circularline(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  pointParams = list(),
  lineParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  radarParams = list(),
  xAsFactor = TRUE
)
```

Arguments

`summarydata`	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
`xfactor`	a string with the name of the column where the factor going on the horizontal axis is given;
`groupingfactor`	a string with the name of the column for which the data will be grouped on the plot;
`addfactors`	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
`rawdata`	always contains "DV" for each participants and each level of the factors
`pointParams`	(optional) list of graphic directives that are sent to the `geom_bar()` layer
`lineParams`	(optional) list of graphic directives that are sent to the `geom_line()` layer
`errorbarParams`	(optional) list of graphic directives that are sent to the `geom_superberrorbar()` layer
`facetParams`	(optional) list of graphic directives that are sent to the `facet_grid()` layer
`radarParams`	(optional) list of arguments to the radar coordinates (seel `coord_radial()`).
`xAsFactor`	(optional) Boolean to indicate if the factor on the horizontal should be continuous or discrete (default is discrete)

Details

A few things to note:

- You can at any time undo the polar coordinates by using + `coord_cartesian()`. It is sometimes easier when developping the plots.
- Also, if ever you want to modify the scale post-hoc (e.g., to change the labels of the group), you can, but your `scale_x_continuous` **must absolutely** contains the two arguments: scale_x_continuous(oob = scales::oob_keep, limits = c(0, 0.00001+ *NUMBER OF CONDITIONS*), # any other argument such as labels = c("",...))

It has these parameters:

Value

a ggplot object

Examples

```
# This will make a plot with lines
superb(
   len ~ dose + supp,
   ToothGrowth,
   plotLayout="circularline"
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superb(
#   len ~ dose + supp,
#   ToothGrowth,
#   showPlot = FALSE
#)
#
#superbPlot.circularline(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

`superbPlot.circularlineBand` *superbPlot 'circularlineBand' layout*

Description

superb comes with a few circular layouts for making plots. It produces ggplot objects that can be further customized.

It has these parameters:

Usage

```
superbPlot.circularlineBand(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  pointParams = list(),
  lineParams = list(),
  errorbandParams = list(),
  facetParams = list(),
  radarParams = list(),
  xAsFactor = TRUE
)
```

Arguments

summarydata	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
xfactor	a string with the name of the column where the factor going on the horizontal axis is given;
groupingfactor	a string with the name of the column for which the data will be grouped on the plot;
addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors
pointParams	(optional) list of graphic directives that are sent to the `geom_bar()` layer
lineParams	(optional) list of graphic directives that are sent to the `geom_line()` layer
errorbandParams	(optional) list of graphic directives that are sent to the `geom_ribbon()` layer
facetParams	(optional) list of graphic directives that are sent to the `facet_grid()` layer
radarParams	(optional) list of arguments to the radar coordinates (seel `coord_radial()`).
xAsFactor	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

Examples

```
# This will make a plot with points
superbPlot(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len",
   plotLayout = "circularlineBand"
)
```

```
# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superbData(ToothGrowth,
# BSFactors = c("dose","supp"), variables = "len"
#)
#
#superbPlot.circularlineBand(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

`superbPlot.circularpoint` *superbPlot 'circularpoint' layout*

Description

superb comes with a few circular layouts for making plots. It produces ggplot objects that can be further customized.

It has these parameters:

Usage

```
superbPlot.circularpoint(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  pointParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  radarParams = list(),
  xAsFactor = TRUE
)
```

Arguments

`summarydata`	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
`xfactor`	a string with the name of the column where the factor going on the horizontal axis is given;
`groupingfactor`	a string with the name of the column for which the data will be grouped on the plot;

addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors
pointParams	(optional) list of graphic directives that are sent to the geom_bar layer
errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer
radarParams	(optional) list of arguments to the radar coordinates (seel `coord_radial()`).
xAsFactor	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

Examples

```
# This will make a plot with bars
superb(
   len ~ dose + supp,
   ToothGrowth,
   plotLayout="circularpoint"
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superb(
#   len ~ dose + supp,
#   ToothGrowth,
#   showPlot = FALSE
#)
#
#superbPlot.circularpoint(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

---

superbPlot.circularpointjitter

superbPlot 'circularpointjitter' layout

Description

superb comes with a few circular layouts for making plots. It produces ggplot objects that can be further customized.

It has these parameters:

Usage

```
superbPlot.circularpointjitter(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  pointParams = list(),
  jitterParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  radarParams = list(),
  xAsFactor = TRUE
)
```

Arguments

summarydata	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
xfactor	a string with the name of the column where the factor going on the horizontal axis is given;
groupingfactor	a string with the name of the column for which the data will be grouped on the plot;
addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors
pointParams	(optional) list of graphic directives that are sent to the geom_bar layer
jitterParams	(optional) list of graphic directives that are sent to the geom_bar layer
errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer
radarParams	(optional) list of arguments to the radar coordinates (seel `coord_radial()`).
xAsFactor	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

Examples

```
# This will make a plot with points
superbPlot(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len",
   plotLayout = "circularpointjitter"
)
```

```
# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superbData(ToothGrowth,
# BSFactors = c("dose","supp"), variables = "len"
#)
#
#superbPlot.circularpointjitter(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

`superbPlot.circularpointlinejitter` *superbPlot 'circularpointlinejitter' layout*

Description

superb comes with a few circular layouts for making plots. It produces ggplot objects that can be further customized.

It has these parameters:

Usage

```
superbPlot.circularpointlinejitter(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  pointParams = list(),
  lineParams = list(),
  jitterParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  radarParams = list(),
  xAsFactor = TRUE
)
```

Arguments

`summarydata`	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
`xfactor`	a string with the name of the column where the factor going on the horizontal axis is given;

groupingfactor	a string with the name of the column for which the data will be grouped on the plot;
addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors
pointParams	(optional) list of graphic directives that are sent to the geom_bar layer
lineParams	(optional) list of graphic directives that are sent to the geom_bar layer
jitterParams	(optional) list of graphic directives that are sent to the geom_bar layer
errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer
radarParams	(optional) list of arguments to the radar coordinates (seel `coord_radial()`).
xAsFactor	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

Examples

```
# This will make a plot with points
superbPlot(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len",
   plotLayout = "circularpointlinejitter"
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superbData(ToothGrowth,
# BSFactors = c("dose","supp"), variables = "len"
#)
#
#superbPlot.circularpointlinejitter(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

`superbPlot.corset` *superbPlot 'corset' layout*

Description

superbPlot comes with a few built-in templates for making the final plots. The corset plot is specifically devised for 2-repeated-measure design: it merges the "pointindividualline" layout with a raincloud layout (Belisario 2021). All layout produces ggplot objects that can be further customized. Additionally, it is possible to create custom-make templates (see vignette 5). The functions, to be "superbPlot-compatible", must have these parameters:

Usage

```
superbPlot.corset(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  lineParams = list(),
  pointParams = list(),
  errorbarParams = list(),
  jitterParams = list(),
  violinParams = list(),
  facetParams = list(),
  xAsFactor = TRUE
)
```

Arguments

summarydata	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
xfactor	a string with the name of the column where the factor going on the horizontal axis is given;
groupingfactor	a string with the name of the column for which the data will be grouped on the plot;
addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors;
lineParams	(optional) list of graphic directives that are sent to the geom_line layer including the option colorize "bySlope", "byId" or "none";
pointParams	(optional) list of graphic directives that are sent to the geom_bar layer;
errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer;
jitterParams	(optional) list of graphic directives that are sent to the geom_jitter layer;
violinParams	(optional) list of graphic directives that are sent to the geom_boxplot layer;
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer;
xAsFactor	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete).

Value

a ggplot object

References

Belisario K (2021). *Corset Plots: Visualizing Heterogeneity in Change Outcomes Across Two Time-points*. doi:10.5281/zenodo.4905031, https://cran.r-project.org/package=ggcorset.

Examples

```
# We first generate randomly a 2-measurement dataset with 50 participants and a large effect
dta <- GRD(SubjectsPerGroup = 50, WSFactors = "moment(2)", Effects = list("moment"=slope(3)))

# This will make a basic corset plot
superb(
   cbind(DV.1, DV.2) ~ .,
   dta,
   WSFactors = "moment(2)",
   plotLayout = "corset"
)

# This will color the increasing and decreasing individuals
superb(
   cbind(DV.1, DV.2) ~ .,
   dta,
   WSFactors = "moment(2)",
   plotLayout = "corset",
   lineParams = list(colorize="bySlope")
)

# This layout has similarities with the "pointindividualline" layout
superb(
   cbind(DV.1, DV.2) ~ .,
   dta,
   WSFactors = "moment(2)",
   plotLayout = "pointindividualline"
)

# if you extract the data with superbData, you can
# run this layout directly
processedData <- superb(
   cbind(DV.1, DV.2) ~ .,
   dta,
   WSFactors = "moment(2)",
   showPlot  = FALSE
)

superbPlot.corset(processedData$summaryStatistic,
   "moment", NULL, ".~.",
   processedData$rawData,
   lineParams = list(colorize="bySlope") )
```

`superbPlot.halfwidthline` *superbPlot 'halfwidthline' layout*

Description

superbPlot comes with a few built-in templates for making the final plots. All produces ggplot objects that can be further customized. The half-width confidence interval line plot is EXPERIMENTAL. It divides the CI length by two, one thick section and one thin section. The functions, to be "superbPlot-compatible", must have these parameters:

Usage

```
superbPlot.halfwidthline(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  pointParams = list(),
  lineParams = list(),
  errorbarParams = list(),
  errorbarlightParams = list(),
  facetParams = list(),
  xAsFactor = TRUE
)
```

Arguments

`summarydata`	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
`xfactor`	a string with the name of the column where the factor going on the horizontal axis is given;
`groupingfactor`	a string with the name of the column for which the data will be grouped on the plot;
`addfactors`	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
`rawdata`	always contains "DV" for each participants and each level of the factors
`pointParams`	(optional) list of graphic directives that are sent to the geom_bar layer
`lineParams`	(optional) list of graphic directives that are sent to the geom_bar layer
`errorbarParams`	(optional) list of graphic directives that are sent to the geom_superberrorbar layer

errorbarlightParams
: (optional) graphic directives for the second half of the error bar;

facetParams
: (optional) list of graphic directives that are sent to the facet_grid layer

xAsFactor
: (optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

Examples

```
# This will make a plot with lines
superb(
   len ~ dose + supp,
   ToothGrowth,
   plotLayout="halfwidthline"
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superb(
#   len ~ dose + supp,
#   ToothGrowth,
#   showPlot = FALSE
#)
#
#superbPlot.halfwidthline(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

---

superbPlot.line *superbPlot 'line' layout*

Description

superbPlot comes with a few built-in templates for making the final plots. All produces ggplot objects that can be further customized. Additionally, it is possible to add custom-make templates (see vignette 6). The functions, to be "superbPlot-compatible", must have these parameters:

Usage

```
superbPlot.line(
  summarydata,
  xfactor,
  groupingfactor,
```

```
  addfactors,
  rawdata = NULL,
  pointParams = list(),
  lineParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  xAsFactor = TRUE
)
```

Arguments

summarydata	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
xfactor	a string with the name of the column where the factor going on the horizontal axis is given;
groupingfactor	a string with the name of the column for which the data will be grouped on the plot;
addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors
pointParams	(optional) list of graphic directives that are sent to the geom_bar layer
lineParams	(optional) list of graphic directives that are sent to the geom_bar layer
errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer
xAsFactor	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

Examples

```
# This will make a plot with lines
superb(
   len ~ dose + supp,
   ToothGrowth,
   plotLayout="line"
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superb(
#   len ~ dose + supp,
#   ToothGrowth,
#   showPlot = FALSE
#)
```

```
#
#superbPlot.line(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

`superbPlot.lineBand` *superbPlot 'lineBand' layout*

Description

The lineBand layout displays an error band instead of individual error bars. This layout is convenient when you have many points on your horizontal axis (so that the error bars are difficult to distinguish) and when the results are fairly smooth.

The functions has these parameters:

Usage

```
superbPlot.lineBand(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata,
  pointParams = list(),
  lineParams = list(),
  errorbandParams = list(),
  facetParams = list(),
  xAsFactor = TRUE
)
```

Arguments

`summarydata`	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
`xfactor`	a string with the name of the column where the factor going on the horizontal axis is given;
`groupingfactor`	a string with the name of the column for which the data will be grouped on the plot;
`addfactors`	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
`rawdata`	always contains "DV" for each participants and each level of the factors
`pointParams`	(optional) list of graphic directives that are sent to the geom_point layer
`lineParams`	(optional) list of graphic directives that are sent to the geom_jitter layer

errorbandParams
: (optional) list of graphic directives that are sent to the geom_ribbon layer

facetParams
: (optional) list of graphic directives that are sent to the facet_grid layer

xAsFactor
: (optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

References

There are no references for Rd macro \insertAllCites on this help page.

Examples

```
# this creates a fictious time series at 100 time points obtained in two conditions:
dta <- GRD( WSFactors = "timepoints (50) : condition(2)",
    SubjectsPerGroup = 20,
    RenameDV = "activation",
    Effects = list("timepoints" = extent(5), "condition" = extent(3) ),
    Population=list(mean=50,stddev=10,rho=0.75)
)

# This will make a plot with error band
superb(
   crange(activation.1.1, activation.50.2) ~ .,
   dta,
   WSFactors   = c("timepoints(50)", "condition(2)"),
   adjustments = list(
        purpose       = "single",
        decorrelation = "CM"        ## or none for no decorrelation
   ),
   plotLayout  = "lineBand",        # note the uppercase B
   pointParams = list(size= 1)      # making points smaller has better look
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superb(
#   crange(activation.1.1, activation.50.2) ~ .,
#   dta,
#   WSFactors   = c("timepoints(50)", "condition(2)"), variables = colnames(dta)[2:101],
#   adjustments = list(
#        purpose       = "single",
#        decorrelation = "CM"        ## or none for no decorrelation
#   )
#)
#
#superbPlot.lineBand(processedData$summaryStatistic,
#   "timepoints",
#   "condition",
```

```
#   ".~.",
#   processedData$rawData)
```

`superbPlot.point` *superbPlot 'point' layout*

Description

superbPlot comes with a few built-in templates for making the final plots. All produces ggplot objects that can be further customized. Additionally, it is possible to add custom-make templates (see vignette 6). The functions, to be "superbPlot-compatible", must have these parameters:

Usage

```
superbPlot.point(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  pointParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  xAsFactor = TRUE
)
```

Arguments

`summarydata`	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
`xfactor`	a string with the name of the column where the factor going on the horizontal axis is given;
`groupingfactor`	a string with the name of the column for which the data will be grouped on the plot;
`addfactors`	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
`rawdata`	always contains "DV" for each participants and each level of the factors
`pointParams`	(optional) list of graphic directives that are sent to the geom_bar layer
`errorbarParams`	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
`facetParams`	(optional) list of graphic directives that are sent to the facet_grid layer
`xAsFactor`	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

Examples

```
# This will make a plot with points
superbPlot(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len",
   plotLayout = "point"
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superbData(ToothGrowth,
# BSFactors = c("dose","supp"), variables = "len"
#)
#
#superbPlot.point(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

---

superbPlot.pointindividualline

superbPlot point and individual-line layout for within-subject design

---

Description

superbPlot comes with a few built-in templates for making the final plots. All produces ggplot objects that can be further customized. Additionally, it is possible to add custom-make templates (see vignette 6). The functions, to be "superbPlot-compatible", must have these parameters:

Usage

```
superbPlot.pointindividualline(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata,
  datapointParams = list(),
  pointParams = list(),
  lineParams = list(),
  errorbarParams = list(),
  facetParams = list()
)
```

Arguments

summarydata	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
xfactor	a string with the name of the column where the factor going on the horizontal axis is given;
groupingfactor	a string with the name of the column for which the data will be grouped on the plot;
addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors
datapointParams	(optional) list of graphic directives that are sent to the geom_point layer of the individual lines
pointParams	(optional) list of graphic directives that are sent to the geom_point layer
lineParams	(optional) list of graphic directives that are sent to the geom_line layer; the parameter `colorize` can be used to obtain distinct colors for decreasing segments of line (`colorize = "bySlope"`), to obtain distinct colors for each participants (`colorize = "byId"`), or to have them all gray (default with `colorize = "none"`).
errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer

Value

a ggplot object

Examples

```
# This will make a plot with points and individual lines for each subject's scores

# we take the Orange built-in data.frame but shorten the names...
names(Orange) <- c("Tree","age","circ")
# Makes the plot:
 superb( circ ~ age | Tree,
   Orange,
   adjustments = list(purpose = "difference", decorrelation = "none"),
   plotLayout= "pointindividualline"
 )

# if you extract the data, you can
# run this layout directly
#processedData <- superb( circ ~ age | Tree,
#  Orange,
#  adjustments = list(purpose = "difference", decorrelation = "none"),
#)
#
```

```
#superbPlot.pointindividualline(processedData$summaryStatistic,
#   "age",
#   NULL,
#   ".~.",
#   processedData$rawData)
```

`superbPlot.pointjitter`

superbPlot point-and-jitter dots layout

Description

superbPlot comes with a few built-in templates for making the final plots. All produces ggplot objects that can be further customized. Additionally, it is possible to add custom-make templates (see vignette 6). The functions, to be "superbPlot-compatible", must have these parameters:

Usage

```
superbPlot.pointjitter(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata,
  pointParams = list(),
  jitterParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  xAsFactor = TRUE
)
```

Arguments

`summarydata`	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
`xfactor`	a string with the name of the column where the factor going on the horizontal axis is given;
`groupingfactor`	a string with the name of the column for which the data will be grouped on the plot;
`addfactors`	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
`rawdata`	always contains "DV" for each participants and each level of the factors
`pointParams`	(optional) list of graphic directives that are sent to the geom_bar layer
`jitterParams`	(optional) list of graphic directives that are sent to the geom_bar layer

errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer
xAsFactor	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

Examples

```
# This will make a plot with jittered points, aka dot plots
superb(
   len ~ dose + supp,
   ToothGrowth,
   plotLayout="pointjitter"
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superb(
#   len ~ dose + supp,
#   ToothGrowth,
#   showPlot = FALSE
#)
#
#superbPlot.pointjitter(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

---

superbPlot.pointjitterviolin

superbPlot point, jitter and violin plot layout

Description

superbPlot comes with a few built-in templates for making the final plots. All produces ggplot objects that can be further customized. Additionally, it is possible to add custom-make templates (see vignette 6). The functions, to be "superbPlot-compatible", must have these parameters:

Usage

```
superbPlot.pointjitterviolin(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata,
  pointParams = list(),
  jitterParams = list(),
  violinParams = list(),
  errorbarParams = list(),
  facetParams = list()
)
```

Arguments

summarydata	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
xfactor	a string with the name of the column where the factor going on the horizontal axis is given;
groupingfactor	a string with the name of the column for which the data will be grouped on the plot;
addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors
pointParams	(optional) list of graphic directives that are sent to the geom_bar layer
jitterParams	(optional) list of graphic directives that are sent to the geom_bar layer
violinParams	(optional) list of graphic directives that are sent to the geom_bar layer this modified geom_violin has additional options "direction"/"antagonize" and "push".
errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer

Value

a ggplot object

Examples

```
# This will make a plot with jittered points and violins for the overall distribution
superb(
   len ~ dose + supp,
   ToothGrowth,
   plotLayout = "pointjitterviolin"
)
```

```
# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superb(
#   len ~ dose + supp,
#   ToothGrowth,
#   showPlot = FALSE
#)
#
#superbPlot.pointjitterviolin(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

`superbPlot.pointlinejitter`

superbPlot point-and-jitter lines layout

Description

superbPlot comes with a few built-in templates for making the final plots. All produces ggplot objects that can be further customized. Additionally, it is possible to add custom-make templates (see vignette 6). The functions, to be "superbPlot-compatible", must have these parameters:

Usage

```
superbPlot.pointlinejitter(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata,
  pointParams = list(),
  lineParams = list(),
  jitterParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  xAsFactor = TRUE
)
```

Arguments

`summarydata`	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
`xfactor`	a string with the name of the column where the factor going on the horizontal axis is given;

groupingfactor	a string with the name of the column for which the data will be grouped on the plot;
addfactors	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
rawdata	always contains "DV" for each participants and each level of the factors
pointParams	(optional) list of graphic directives that are sent to the geom_bar layer
lineParams	(optional) list of graphic directives that are sent to the geom_bar layer
jitterParams	(optional) list of graphic directives that are sent to the geom_bar layer
errorbarParams	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
facetParams	(optional) list of graphic directives that are sent to the facet_grid layer
xAsFactor	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

Examples

```
# This will make a plot with jittered points, aka dot plots
superb(
   len ~ dose + supp,
   ToothGrowth,
   plotLayout="pointlinejitter"
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superb(
#   len ~ dose + supp,
#   ToothGrowth,
#   showPlot = FALSE
#)
#
#superbPlot.pointlinejitter(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

`superbPlot.raincloud` *superbPlot 'raincloud' layout*

Description

The raincloud layout display jittered dots as well as a "cloud" (half of a violin) above them. See @allen2019. The functions has these parameters:

Usage

```
superbPlot.raincloud(
  summarydata,
  xfactor,
  groupingfactor,
  addfactors,
  rawdata = NULL,
  violinParams = list(),
  jitterParams = list(),
  pointParams = list(),
  errorbarParams = list(),
  facetParams = list(),
  xAsFactor = TRUE
)
```

Arguments

`summarydata`	a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
`xfactor`	a string with the name of the column where the factor going on the horizontal axis is given;
`groupingfactor`	a string with the name of the column for which the data will be grouped on the plot;
`addfactors`	a string with up to two additional factors to make the rows and columns panels, in the form "fact1 ~ fact2";
`rawdata`	always contains "DV" for each participants and each level of the factors
`violinParams`	(optional) list of graphic directives that are sent to the geom_violin layer; this modified geom_violin has additional options "direction" and "push".
`jitterParams`	(optional) list of graphic directives that are sent to the geom_jitter layer
`pointParams`	(optional) list of graphic directives that are sent to the geom_point layer
`errorbarParams`	(optional) list of graphic directives that are sent to the geom_superberrorbar layer
`facetParams`	(optional) list of graphic directives that are sent to the facet_grid layer
`xAsFactor`	(optional) Boolean to indicate if the factor on the horizontal should continuous or discrete (default is discrete)

Value

a ggplot object

References

There are no references for Rd macro `\insertAllCites` on this help page.

Examples

```
# This will make a plot with raincloud; they are better seen rotated: +coord_flip()
superb(
   len ~ dose + supp,
   ToothGrowth,
   plotLayout="raincloud"
)

# if you extract the data with superbData, you can
# run this layout directly
#processedData <- superb(ToothGrowth,
#   len ~ dose + supp,
#   showPlot = FALSE
#)
#
#superbPlot.raincloud(processedData$summaryStatistic,
#   "dose",
#   "supp",
#   ".~.",
#   processedData$rawData)
```

`superbShiny` *User Interface to get summary plot of any statistics with adjusted error bars.*

Description

The function `suberbShiny()` provides a simple user interface to plot standard error or confidence interval for various descriptive statistics under various designs, population size and purposes, according to the `suberb` framework. See (Cousineau et al. 2021) for more. Also see this video from (Walker 2021) for a demo using the shinyapps.io installation accessible at dcousin3.shinyapps.io/superbshiny Limitations: it is neither possible to use custom-made statistics with the graphical user interface, nor is it possible to request an adjustment for cluster- randomized sampling. These options are available with `superb()`.

Usage

```
superbShiny(graphicDirectives = NULL)
```

Arguments

graphicDirectives
: (optional) used to set graphic directives from the command line. This is useful for in-class demonstrations where the `ylim()` range should be set before reaching the last step of the interface.

Value

A plot that can be cut-and-paste.

References

Cousineau D, Goulet M, Harding B (2021). "Summary plots with adjusted error bars: The superb framework with an implementation in R." *Advances in Methods and Practices in Psychological Science*, **4**, 1–18. doi:10.1177/25152459211035109.

Walker JAL (2021). *Summary plots with adjusted error bars (superb) [Youtube video]*. https://www.youtube.com/watch?v=rw_6ll5nVus.

Examples

```
# Launch the user interface:

if (interactive())
   superbShiny()


# Example with two graphic directives given

if (interactive())
   superbShiny( "ylim(65,135)+theme_bw()" )
```

---

superbToWide *superbToWide: Reshape long data frame to wide, suitable for superbPlot*

Description

The function `suberbToWide()` is an extension to Navarro's WideToLong function with ample checks to make sure all is legit, so that the data is suitably organized for `suberb`. See Cousineau et al. (2021) for more. Other techniques are available to transform long to wide, but many asked for this function to ship within `superb`. Note that with the function `superb()` and a formula, there really is no need anymore to use `superbToWide()`.

Usage

```
superbToWide(
  data,
  id = NULL,
  BSFactors = NULL,
  WSFactors = NULL,
  variable = NULL
)
```

Arguments

data	Dataframe in long format
id	A column with unique identifiers per subject
BSFactors	The name(s) of the between-subject factor(s) as string(s)
WSFactors	The name(s) of the within-subject factor(s) as string(s)
variable	The dependent variable as string

Value

A wide-format data frame ready for `superbPlot()` or `superbData()`. All other variables will be erased.

References

Cousineau D, Goulet M, Harding B (2021). "Summary plots with adjusted error bars: The superb framework with an implementation in R." *Advances in Methods and Practices in Psychological Science*, **4**, 1–18. doi:10.1177/25152459211035109.

Examples

```
library(ggplot2)
library(gridExtra)

# Example using the built-in dataframe Orange.
data(Orange)
superbToWide(Orange, id = "Tree", WSFactors = c("age"), variable = "circumference")

# Optional: change column names to shorten "circumference" to "DV"
names(Orange) <- c("Tree","age","DV")
# turn the data into a wide format
Orange.wide <- superbToWide(Orange, id = "Tree", WSFactors = c("age"), variable = "DV")

# Makes the plots two different way:
p1=superbPlot( Orange.wide, WSFactors = "age(7)",
  variables = c("DV.118","DV.484","DV.664","DV.1004","DV.1231","DV.1372","DV.1582"),
  adjustments = list(purpose = "difference", decorrelation = "none")
) +
  xlab("Age level") + ylab("Trunk diameter (mm)") +
  coord_cartesian( ylim = c(0,250) ) + labs(title="Basic confidence intervals")
```

```
p2=superbPlot( Orange.wide, WSFactors = "age(7)",
  variables = c("DV.118","DV.484","DV.664","DV.1004","DV.1231","DV.1372","DV.1582"),
  adjustments = list(purpose = "difference", decorrelation = "CA")
) +
  xlab("Age level") + ylab("Trunk diameter (mm)") +
  coord_cartesian( ylim = c(0,250) ) + labs(title="Decorrelated confidence intervals")
grid.arrange(p1,p2,ncol=2)

# Note that with superb(), there is no need to reformat
# into a wide format anymore:
superb( DV ~ age | Tree, Orange )
```

TMB1964r *Data of Tulving, Mandler, & Baumal, 1964 (reproduction of 2021)*

Description

The data comes from Bradley-Garcia and 37 others (2021). It is a near exact replication of the original study from (Tulving et al. 1964).

The design is a (7) x 4 with: 7 levels of stimulus duration (within-subject) and 4 between-subject conditions. Additional variables included in the reproduction is the primary language of the participant in which he/she participated (mainly francophones and anglophones; and the gender (mainly male and female).

Usage

```
data(TMB1964r)
```

Format

An object of class data.frame.

References

Bradley-Garcia M, 37 others (2021). "The influence of exposure duration and context length on word recall: A replication of Tulving et al. (1964)." *The Quantitative Methods for Psychology*, **17**(2), r1-r9. doi:10.20982/tqmp.17.2.r001.

Tulving E, Mandler G, Baumal R (1964). "Interaction of two sources of information in tachistoscopic word recognition." *Canadian Journal of Psychology/Revue canadienne de psychologie*, **18**(1), 62.

Examples

```
library(ggplot2)

data(TMB1964r)

options(superb.feedback = 'none') # shut down 'warnings' and 'design' interpretation messages

# general plot ignoring covariates sex and languages with only defaults
# We illustrate correlation- and difference-adjusted 95% confidence intervals of the mean
superb(
    crange(T1, T7) ~ Condition,
    TMB1964r,
    WSFactors = "T(7)",      # the within-subject factor (spanning 7 columns)
    adjustments = list(purpose="difference", decorrelation="CM"),
    plotLayout = "line"
)

# We add directives for the error bars (thick), for the points (larger) and for the lines (thick)
plt <- superb(
    crange(T1, T7) ~ Condition,
    TMB1964r,
    WSFactors = "T(7)",
    adjustments = list(purpose="difference", decorrelation="CM"),
    plotLayout = "line",
    errorbarParams = list(width = 0.5, linewidth=1.25, position = position_dodge(.5) ),
    pointParams = list(size=2.5, position = position_dodge(.5)),
    lineParams = list(linewidth=1.25)
)
plt

# Additional directives to set manually the colors, shapes, thick marks and labels.
plt +
scale_colour_manual(
    labels = c("Context 0", "Context 2", "Context 4", "Context 8"),
    values = c("blue", "black", "purple", "red")) +
scale_shape_manual(
    labels = c("Context 0", "Context 2", "Context 4", "Context 8"),
    values = c("circle", "triangle", "square", "plus")) +
theme_bw(base_size = 16) +
labs(x = "Exposure duration (ms)", y = "Mean of correct responses",
    colour = "Context length\n", shape = "Context length\n" ) +
scale_x_discrete(labels=c("1" = "16.67", "2" = "33.33",
    "3"="50.00", "4" = "66.67", "5"="83.33", "6"="100.00", "7"="116.67"))



# Exploring three factors simultaneously: T, Condition and Sex (last two between-group)
superb(
    crange(T1, T7) ~ Condition + Sex,
    TMB1964r,
    WSFactors = "T(7)",      # the within-subject factor (spanning 7 columns)
    adjustments = list(purpose="difference", decorrelation="CM"),
```

```
    plotLayout = "line",
    errorbarParams = list(linewidth=0.15, position = position_dodge(.5) ),
    pointParams = list(size=2.5, position = position_dodge(.5)),
    lineParams = list(linewidth=0.25)
) +
scale_colour_manual(
    labels = c("Context 0", "Context 2", "Context 4", "Context 8"),
    values = c("blue", "black", "purple", "red")) +
scale_shape_manual(
    labels = c("Context 0", "Context 2", "Context 4", "Context 8"),
    values = c("circle", "triangle", "square", "plus")) +
theme_bw(base_size = 16) +
labs(x = "Exposure duration (ms)", y = "Mean of correct responses",
    colour = "Context length\n", shape = "Context length\n" ) +
scale_x_discrete(labels=c("1" = "16.67", "2" = "33.33",
    "3"="50.00", "4" = "66.67", "5"="83.33", "6"="100.00", "7"="116.67"))


#only keep 2 sex and 2 languages; the remaining cases are too sparse.
mee3 <- TMB1964r[(TMB1964r$Language != "I prefer not to answer")&TMB1964r$Language !="Other",]

### This last example is commented as CRAN servers are too slow
#
# advanced plots are available, such as pointjitter
# and pointjitterviolin : a plot that superimposes the distribution as a violin plot
#
# superb(
#   crange(T1, T7) ~ Condition + Language,
#   mee3,
#   WSFactors = "T(7)",
#   adjustments = list(purpose="difference", decorrelation="CM"),
#   plotLayout = "pointjitterviolin",
#   jitterParams = list(alpha = 0.4), #near transparent jitter points
#   violinParams = list(alpha = 0.2)
#) +
#scale_fill_manual( name = "Amount of context",
#    labels = c("Context 0", "Context 2", "Context 4", "Context 8"),
#    values = c("blue", "black", "purple", "red")) +
#scale_colour_manual( name = "Amount of context",
#    labels = c("Context 0", "Context 2", "Context 4", "Context 8"),
#    values = c("blue", "black", "purple", "red")) +
#scale_shape_manual( name = "Amount of context",
#    labels = c("Context 0", "Context 2", "Context 4", "Context 8"),
#    values = c("circle", "triangle", "square", "cross")) +
#theme_bw(base_size = 16) +
#labs(x = "Exposure duration (ms)", y = "Mean of correct responses" )+
#scale_x_discrete(labels=c("1" = "16.67", "2" = "33.33",
#    "3"="50.00", "4" = "66.67", "5"="83.33", "6"="100.00", "7"="116.67"))
#
```

`twoStepTransform` *two-step transform for subject centering and bias correction*

Description

`twoStepTransform()` is a transformation that can be applied to a matrix of data. The resulting matrix is both subject-centered and bias corrected, a technique called the CM technique (Baguley 2012; Cousineau 2005; Morey 2008; O'Brien and Cousineau 2014)

Usage

```
twoStepTransform(dta, variables)
```

Arguments

`dta`	a data.frame containing the data in wide format;
`variables`	a vector of column names on which the transformation will be applied. the remaining columns will be left unchanged

Value

a data.frame of the same form as dta with the variables transformed.

This function is useful when passed to the argument `preprocessfct` of `superb()` where it performs a modification of the data matrix.

References

Baguley T (2012). "Calculating and graphing within-subject confidence intervals for ANOVA." *Behavior Research Methods*, **44**, 158 – 175. doi:10.3758/s1342801101237.

Cousineau D (2005). "Confidence intervals in within-subject designs: A simpler solution to Loftus and Masson's method." *Tutorials in Quantitative Methods for Psychology*, **1**, 42 – 45. doi:10.20982/tqmp.01.1.p042.

Morey RD (2008). "Confidence Intervals from Normalized Data: A correction to Cousineau (2005)." *Tutorials in Quantitative Methods for Psychology*, **4**, 61 – 64. doi:10.20982/tqmp.04.2.p061.

O'Brien F, Cousineau D (2014). "Representing Error bars in within-subject designs in typical software packages." *The Quantitative Methods for Psychology*, **10**(1), 56-67. doi:10.20982/tqmp.10.1.p056.

`WelchDegreeOfFreedom` *Welch's rectified degree of freedom*

Description

When variance across groups are heterogeneous, the Student `t` distribution with `n - 1` df is not the exact distribution. However, (Welch 1947), using methods of moments, was able to find the best-fitting `t` distribution. This distribution has degree of freedom reduced based on the sample sizes and the variances of the group tests. The present function returns the rectified degree of freedom

Usage

```
WelchDegreeOfFreedom(dta, cols, groupingcols)
```

Arguments

`dta`	A data frame containing within-subject measures, one participant per line;
`cols`	A vector indicating the columns containing the measures.
`groupingcols`	A vector indicating the columns containing the groups.

Value

df the degrees of freedom rectified according to Welch (1947).

References

Welch BL (1947). "The generalization of student's' problem when several different population variances are involved." *Biometrika*, **34**(1/2), 28–35. doi:10.1093/biomet/34.12.28.

Examples

```
# creates a small data frame with 4 subject's scores for 5 measures:
dta <- data.frame(cbind(
        DV.1 = c(3., 6., 2., 2., 5.),
        DV.2 = c(4., 5., 4., 4., 3.),
        DV.3 = c(2., 7., 7., 8., 6.),
        DV.4 = c(6., 8., 4., 6., 5.),
        grp  = c(1., 1., 2., 2., 2.)
    ))
# performs the test (here rectified df = 1.898876)
WelchDegreeOfFreedom(dta, "DV.1","grp")
```

`WinerCompoundSymmetryTest` *Winer's test of compound symmetry*

Description

Run a test of compound symmetry. generates a data frame of random data suitable for analyses. It assesses the significance of the null hypothesis that the covariance matrix is compound symmetric. This test is given without demonstration in (Winer et al. 1991), p. 517.

Usage

```
WinerCompoundSymmetryTest(dta, cols)
```

Arguments

`dta`	A data frame containing within-subject measures, one participant per line;
`cols`	A vector indicating the columns containing the measures.

Value

p the p-value of the null hypothesis that the data are compound symmetric.

References

Winer BJ, Brown DR, Michels KM (1991). *Statistical principles in experimental design*. McGraw-Hill, New York.

Examples

```
# creates a small data frames with 4 subject's scores for 5 measures:
dta <- data.frame(cbind(
        col1 <- c(3., 6., 2., 2., 5.),
        col2 <- c(4., 5., 4., 4., 3.),
        col3 <- c(2., 7., 7., 8., 6.),
        col4 <- c(6., 8., 4., 6., 5.)
    ))
# performs the test (here p = 0.6733)
WinerCompoundSymmetryTest(dta)
```

Index