Package 'superb'

October 4, 2021

Type Package

Title Summary Plots with Adjusted Error Bars

Version 0.9.7.6

Date 2021-10-04

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://dcousin3.github.io/superb/

Description Computes standard error and confidence interval of various descriptive statistics under various designs and sampling schemes. The main function, superbPlot(), can either return a plot or a dataframe with the statistic and its precision interval so that other plotting package 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.

License GPL-3

Encoding UTF-8

VignetteBuilder knitr

LazyData true

RoxygenNote 7.1.1

Depends R (>= 3.5.0)

Imports foreign, psych, plyr (>= 1.8.4), ggplot2 (>= 3.1.0), MASS, lsr (>= 0.5), methods, Rdpack (>= 0.7), stats, shiny, shinyBS, stringr, utils

Suggests emojifont, fMultivar, grid, gridExtra, knitr, lattice, lawstat, boot, car, reshape2, rmarkdown, sadists, scales, testthat, tibble

RdMacros Rdpack

NeedsCompilation no

Repository CRAN

Date/Publication 2021-10-04 15:40:03 UTC

R topics documented:

biasCorrectionTransform . . . 3
bootstrapPrecisionMeasures . . . 3
CousineauLaurencelleLambda . . . 5
dataFigure1 . . . 6
dataFigure2 . . . 7
dataFigure3 . . . 8
dataFigure4 . . . 9
geom_superberrorbar . . . 10
GRD . . . 13
HyunhFeldtEpsilon . . . 15
makeTransparent . . . 16
MauchlySphericityTest . . . 17
measuresWithMissingData . . . 18
poolSDTransform . . . 19
precisionMeasures . . . 19
precisionMeasureWithCustomDF . . . 21
runDebug . . . 22
showSignificance . . . 23
ShroutFleissICC1 . . . 25
slope . . . 26
subjectCenteringTransform . . . 27
summaryStatistics . . . 28
superbData . . . 29
superbPlot . . . 31
superbPlot.bar . . . 34
superbPlot.halfwidthline . . . 35
superbPlot.line . . . 37
superbPlot.point . . . 38
superbPlot.pointindividualline . . . 40
superbPlot.pointjitter . . . 41
superbPlot.pointjitterviolin . . . 43
superbPlot.raincloud . . . 44
superbShiny . . . 46
TMB1964r . . . 47
twoStepTransform . . . 50
WelchDegreeOfFreedom . . . 51
WinerCompoundSymmetryTest . . . 52

Index 53

`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)

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.

References

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 (rpw19; 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, https://doi.org/10.1016/S0378-3758(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.

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 <- superbPlot(dataFigure1, BSFactors = "grp",
    adjustments=list(purpose = "single"),
    variables = c("score"), plotStyle="bar" ) +
  xlab("Group") + ylab("Score") + labs(title="95% CI\n") +
  coord_cartesian( ylim = c(85,115) ) +
  geom_hline(yintercept = 100, colour = "black", size = 0.5, linetype=2)
plt1b <- superbPlot(dataFigure1, BSFactors = "grp",
    adjustments=list(purpose = "difference"),
    variables = c("score"), plotStyle="bar" ) +
  xlab("Group") + ylab("Score") + labs(title="Difference-adjusted 95% CI\n") +
  coord_cartesian( ylim = c(85,115) ) +
  geom_hline(yintercept = 100, colour = "black", size = 0.5, linetype=2)
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 <- superbPlot(dataFigure2, WSFactors = "Moment(2)",
    adjustments=list(purpose = "difference"),
    variables = c("pre","post"), plotStyle="bar" ) +
  xlab("Group") + ylab("Score") + labs(title="Difference-adjusted 95% CI\n") +
  coord_cartesian( ylim = c(85,115) ) +
  geom_hline(yintercept = 100, colour = "black", size = 0.5, linetype=2)
```

```
plt2b <- superbPlot(dataFigure2, WSFactors = "Moment(2)",
    adjustments=list(purpose = "difference", decorrelation = "CA"),
    variables = c("pre","post"), plotStyle="bar" ) +
 xlab("Group") + ylab("Score") + labs(title="Correlation and difference-adjusted\n95% CI") +
  coord_cartesian( ylim = c(85,115) ) +
  geom_hline(yintercept = 100, colour = "black", size = 0.5, linetype=2)
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 <- superbPlot(dataFigure3, BSFactors = "grp",
    adjustments=list(purpose = "difference", samplingDesign = "SRS"),
```

```
    variables = c("VD"), plotStyle="bar" ) +
  xlab("Group") + ylab("Score") + labs(title="Difference-adjusted 95% CI\n") +
  coord_cartesian( ylim = c(85,115) ) +
  geom_hline(yintercept = 100, colour = "black", size = 0.5, linetype=2)
plt3b <- superbPlot(dataFigure3, BSFactors = "grp",
    adjustments=list(purpose = "difference", samplingDesign = "CRS"),
    variables = c("VD"), plotStyle="bar", clusterColumn = "cluster" ) +
  xlab("Group") + ylab("Score") + labs(title="Cluster and difference-adjusted\n95% CI") +
  coord_cartesian( ylim = c(85,115) ) +
  geom_hline(yintercept = 100, colour = "black", size = 0.5, linetype=2)
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)
```

`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 = superbPlot(dataFigure4, BSFactors = "group",
    adjustments=list(purpose = "single", popSize = Inf),
    variables = c("score"), plotStyle="bar" ) +
  xlab("Group") + ylab("Score") + labs(title="Difference-adjusted 95% CI\n") +
  coord_cartesian( ylim = c(85,115) ) +
  geom_hline(yintercept = 100, colour = "black", size = 0.5, linetype=2)
plt4b = superbPlot(dataFigure4, BSFactors = "group",
    adjustments=list(purpose = "single", popSize = 50 ),
    variables = c("score"), plotStyle="bar" ) +
 xlab("Group") + ylab("Score") + labs(title="Population size and difference-\nadjusted 95% CI") +
  coord_cartesian( ylim = c(85,115) ) +
  geom_hline(yintercept = 100, colour = "black", size = 0.5, linetype=2)
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_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 three different ways. First, it is possible to decide whether the error bar tips are unidirectional, pointing to the "left" or to the "right" or if they go in "both" directions. Second, it is possible to "double" or "triple" the horizontal marks at the extremities of the error bar, with a "tipgap" of your liking. Third, a new characteristici is vcolour to set a different colour for the vertical part of the error bar. The colour can also be "NA" to have it invisible.

Usage

```
geom_superberrorbar(
  mapping = NULL,
  data = NULL,
  stat = "identity",
```

```
  position = "identity",
  direction = "both",
  tipformat = "single",
  tipgap = 0.1,
  ...,
  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
direction	(NEW) "left", "right" or "both" (Default is "both")
tipformat	(NEW) "single", "double" or "triple" to add additional marker lines to the tips (default is "single")
tipgap	(NEW) The spacing between the markers when "double" or "triple" is used (default 0.1)
...	all additional parameters are sent to the underlying geom_path
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_string(ymin="center-width", ymax="center+width", x = "grp" ) ) +
  geom_superberrorbar()

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

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

# This new geom is integrated inside superbPlot() 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(WSFactors = "Moment(5)",
            Effects = list("Moment" = extent(10) ),
            Population = list(mean = 100, stddev = 25, rho = 0.8) )

ornate = list(
        labs(title =paste("(left)      95% confidence intervals",
                        "\n(right)    99% confidence intervals",
                        "\n(center) 99.9% confidence intervals")),
        xlab("Moment"), ylab("Score"),
        coord_cartesian( ylim = c(85,110) )
)

plt1 <- superbPlot(test,
            WSFactors = "Moment(5)",
            variables = c("DV.1","DV.2","DV.3","DV.4","DV.5"),
            adjustments=list(purpose = "difference", decorrelation = "CA"),
            errorbarParams = list(direction = "left",
                                  width = 0.2, position = position_nudge(-0.05) ),
            gamma     = 0.95,
            plotStyle = "line" ) + ornate
plt2 <- superbPlot(test,
            WSFactors = "Moment(5)",
            variables = c("DV.1","DV.2","DV.3","DV.4","DV.5"),
            adjustments=list(purpose = "difference", decorrelation = "CA"),
           errorbarParams = list(direction = "right", tipgap = 0.5, tipformat = "double",
                                  width = 0.2, position = position_nudge(+0.05) ),
            gamma     = 0.99,
            plotStyle = "line" ) + ornate
plt3 <- superbPlot(test,
            WSFactors = "Moment(5)",
            variables = c("DV.1","DV.2","DV.3","DV.4","DV.5"),
            adjustments=list(purpose = "difference", decorrelation = "CA"),
            errorbarParams = list(direction = "both", tipformat = "single",
                                  width = 0.2, position = position_nudge(0) ),
```

```
            gamma     = 0.999,
            plotStyle = "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)
```

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)
)
```

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
`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)

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.

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 = 1000 )
hist(dta$score )

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

# 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",
        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=5),
    Effects = list("Difficulty" = slope(15) )  )
# show the mean performance as a function of difficulty:
superbPlot(dta, BSFactors = "Difficulty", variables="DV")

# An example in which the moments are correlated
dta <- GRD( BSFactors = "Difficulty(2)",WSFactors = "Moment (2)",
   SubjectsPerGroup = 1000,
    Effects = list("Difficulty" = slope(3), "Moment" = slope(1) ),
    Population=list(mean=50,stddev=20,rho=0.85)
)
# the mean plot on the raw data...
superbPlot(dta, BSFactors = "Difficulty", WSFactors = "Moment(2)",
    variables=c("DV.1","DV.2"), plotStyle="line",
    adjustments = list (purpose="difference") )
# ... and the mean plot on the decorrelated data;
# because of high correlation, the error bars are markedly different
superbPlot(dta, BSFactors = "Difficulty", WSFactors = "Moment(2)",
    variables=c("DV.1","DV.2"), plotStyle="line",
    adjustments = list (purpose="difference", decorrelation = "CM") )
```

`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.

`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; set the elements to transparent.

Examples

```
# make a basic plot
superbPlot(ToothGrowth, BSFactors = c("dose", "supp"),
  variables = "len")
# make a basic plot with transparent elements
superbPlot(ToothGrowth, BSFactors = c("dose", "supp"),
  variables = "len") + 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.

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.

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 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.

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 (if unspecified, uses `n-1` where `n` is the number of observations in x).

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 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.
```

`runDebug` *runDebug*

Description

runDebug is an internal function used by GRD and superbPlot 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.

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 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,
  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	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(
    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 <- superbPlot(dta,
        BSFactor = c("Group"),
        WSFactor = c("Moment(3)"),
        variables = c("DV.1","DV.2","DV.3"),
        adjustments = list(purpose="difference"),
        factorOrder = c("Moment","Group")
    )
plt3 <- superbPlot(dta,
        BSFactor = c("Group","Age"),
        WSFactor = c("Moment(3)"),
        variables = c("DV.1","DV.2","DV.3"),
        adjustments = list(purpose="difference"),
        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)
plt3 +
    showSignificance( c(0.75, 1.25), 90, -1, "++1++", panel = list(Age= 1)) +
    showSignificance( c(1.75, 2.25), 90, -1, "++2++", panel = list(Age= 2)) +
    showSignificance( c(0.75, 1.25), 90, -1, "++3++", panel = list(Age= 3)) +
    showSignificance( c(1.75, 3.25), 95, -1, "++4++", 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(size = 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(size = 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. R code result alters 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.

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. *superbPlot-compatible* summary statistics functions must have one 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
```

---

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.

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*, **in press**, 1–46. 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
```

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

Description

The function `suberbPlot()` 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.

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 = "bar",
  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`	The type of object to plot on the graph. See full list below. Defaults to "bar".
`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

- 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).

In version 0.9.5, 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()`.

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, 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*, **in press**, 1–46. 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
superbPlot(ToothGrowth, BSFactors = c("dose", "supp"),
  variables = "len")

# 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()

# This example is based on repeated measures
library(lsr)
library(gridExtra)
options(superb.feedback = 'none') # shut down 'warnings' and 'design' interpretation messages

# define shorter column names...
names(Orange) <- c("Tree","age","circ")
# turn the data into a wide format
Orange.wide <- longToWide(Orange, circ ~ age)

# Makes the plots two different way:
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="Basic confidence intervals")
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")
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
superbPlot(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len",
   plotStyle="bar"
)

# if you extract the data with superbData, you can
# run this layout directly
processedData <- superbData(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len"
)

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

---

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
superbPlot(ToothGrowth,
   BSFactor = c("dose","supp"), variables = "len",
   plotStyle="halfwidthline"
)

# if you extract the data with superbData, you can
# run this layout directly
processedData <- superbData(ToothGrowth,
   BSFactor = c("dose","supp"), variables = "len"
)

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
superbPlot(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len",
   plotStyle="line"
)

# if you extract the data with superbData, you can
# run this layout directly
processedData <- superbData(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len"
)

superbPlot.line(processedData$summaryStatistic,
   "dose",
   "supp",
   ".~.",
   processedData$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",
   plotStyle = "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,
  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
`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

Value

a ggplot object

Examples

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

# we take the Orange built-in data.frame which has a within-subject design
names(Orange) <- c("Tree","age","circ")
# turn the data into a wide format
Orange.wide <- longToWide(Orange, circ ~ age)
# the identifier to each tree must be in a column called id
Orange.wide$id = Orange.wide$Tree

# Makes the plots two different way:
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"),
  plotStyle= "pointindividualline"
)

# if you extract the data with superbData, you can
# run this layout directly
processedData <- superbData(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"),
)

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
superbPlot(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len",
   plotStyle="pointjitter"
)

# if you extract the data with superbData, you can
# run this layout directly
processedData <- superbData(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len"
)

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
`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
superbPlot(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len",
   plotStyle = "pointjitterviolin"
)

# if you extract the data with superbData, you can
# run this layout directly
processedData <- superbData(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len"
)

superbPlot.pointjitterviolin(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 Allen, Poggiali, Whitaker, Marshall, & Kievit (2018) 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
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()
superbPlot(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len",
   plotStyle="raincloud"
)

# if you extract the data with superbData, you can
# run this layout directly
processedData <- superbData(ToothGrowth,
   BSFactors = c("dose","supp"), variables = "len"
)

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 not 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 `superbPlot()`.

Usage

```
superbShiny()
```

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*, **in press**, 1–46. doi: 10.1177/25152459211035109.

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

Examples

```
# Launch the user interface:

if (interactive())
   superbShiny()
```

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

Description

The data comes from Bradley-Garcia and 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, others 3 (2021). "Getting the most from your curves: Exploring and reporting data using informative graphical techniques." *The Quantitative Methods for Psychology*, **17**(2), r1-r10. 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
superbPlot(TMB1964r,
    WSFactors = "T(7)",      # the within-subject factor (spanning 7 columns)
    BSFactors = "Condition", # the between-subject factor (4 levels)
    variables = c("T1","T2","T3","T4","T5","T6","T7"),
    adjustments = list(purpose="difference", decorrelation="CM"),
    plotStyle = "line"
)
```

```
# We add directives for the error bars (thick), for the points (larger) and for the lines (thick)
plt <- superbPlot(TMB1964r,
    WSFactors = "T(7)",
    BSFactors = "Condition",
    variables = c("T1","T2","T3","T4","T5","T6","T7"),
    adjustments = list(purpose="difference", decorrelation="CM"),
    plotStyle = "line",
    errorbarParams = list(width = 0.5, size=1.25, position = position_dodge(.5) ),
    pointParams = list(size=2.5, position = position_dodge(.5)),
    lineParams = list(size=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)
superbPlot(TMB1964r,
    WSFactors = "T(7)",
    BSFactors = c("Condition","Sex"),
    variables = c("T1","T2","T3","T4","T5","T6","T7"),
    adjustments = list(purpose="difference", decorrelation="CM"),
    plotStyle = "line",
    errorbarParams = list(size=0.15, position = position_dodge(.5) ),
    pointParams = list(size=2.5, position = position_dodge(.5)),
    lineParams = list(size=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.
# even then, one cell is near empty. Only CA would work...
mee3 <- TMB1964r[(TMB1964r$Language != "I prefer not to answer")&TMB1964r$Language !="Other",]


# advanced plots are available, such as pointjitter ...
superbPlot(mee3,
    WSFactors = "T(7)",
    BSFactors = c("Condition","Language"),
    variables = c("T1","T2","T3","T4","T5","T6","T7"),
    adjustments = list(purpose="difference", decorrelation="CM"),
    plotStyle = "pointjitter",
    jitterParams = list(alpha = 0.25) #near transparent jitter points
) +
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"))


# ... and pointjitterviolin : a plot that superimposes the distribution as a violin plot
#
superbPlot(mee3,
    WSFactors = "T(7)",
    BSFactors = c("Condition","Language"),
    variables = c("T1","T2","T3","T4","T5","T6","T7"),
    adjustments = list(purpose="difference", decorrelation="CM"),
    plotStyle = "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)

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.

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.

`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

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

Examples

```
# creates a small data frames 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

∗ **datasets**
dataFigure1, 6
dataFigure2, 7
dataFigure3, 8
dataFigure4, 9
TMB1964r, 47

biasCorrectionTransform, 3
bootstrapPI.gmean *(*bootstrapPrecisionMeasures*)*, 3
bootstrapPI.hmean *(*bootstrapPrecisionMeasures*)*, 3
bootstrapPI.mean *(*bootstrapPrecisionMeasures*)*, 3
bootstrapPI.median *(*bootstrapPrecisionMeasures*)*, 3
bootstrapPI.sd *(*bootstrapPrecisionMeasures*)*, 3
bootstrapPI.var *(*bootstrapPrecisionMeasures*)*, 3
bootstrapPrecisionMeasures, 3
bootstrapSE.gmean *(*bootstrapPrecisionMeasures*)*, 3
bootstrapSE.hmean *(*bootstrapPrecisionMeasures*)*, 3
bootstrapSE.mean *(*bootstrapPrecisionMeasures*)*, 3
bootstrapSE.median *(*bootstrapPrecisionMeasures*)*, 3
bootstrapSE.sd *(*bootstrapPrecisionMeasures*)*, 3
bootstrapSE.var *(*bootstrapPrecisionMeasures*)*, 3

CI.fisherkurtosis *(*precisionMeasures*)*, 19
CI.fisherskew *(*precisionMeasures*)*, 19
CI.gmean *(*precisionMeasures*)*, 19
CI.hmean *(*precisionMeasures*)*, 19
CI.IQR *(*precisionMeasures*)*, 19
CI.MAD *(*precisionMeasures*)*, 19
CI.mean *(*precisionMeasures*)*, 19
CI.meanNArm *(*measuresWithMissingData*)*, 18
CI.median *(*precisionMeasures*)*, 19
CI.pearsonskew *(*precisionMeasures*)*, 19
CI.sd *(*precisionMeasures*)*, 19
CI.var *(*precisionMeasures*)*, 19
CIwithDF.mean *(*precisionMeasureWithCustomDF*)*, 21
CousineauLaurencelleLambda, 5
custom *(*slope*)*, 26

dataFigure1, 6
dataFigure2, 7
dataFigure3, 8
dataFigure4, 9

extent *(*slope*)*, 26

fisherkurtosis *(*summaryStatistics*)*, 28
fisherskew *(*summaryStatistics*)*, 28

geom_superberrorbar, 10
gmean *(*summaryStatistics*)*, 28
GRD, 13

hmean *(*summaryStatistics*)*, 28
HyunhFeldtEpsilon, 15

MAD *(*summaryStatistics*)*, 28
makeTransparent, 16
MauchlySphericityTest, 17
meanNArm *(*measuresWithMissingData*)*, 18
meanNArm, *(*measuresWithMissingData*)*, 18
measuresWithMissingData, 18

pearsonskew *(*summaryStatistics*)*, 28
poolSDTransform, 19
precisionMeasures, 19

precisionMeasureWithCustomDF, 21

Rexpression (slope), 26
runDebug, 22

SE.fisherkurtosis (precisionMeasures), 19
SE.fisherskew (precisionMeasures), 19
SE.gmean (precisionMeasures), 19
SE.hmean (precisionMeasures), 19
SE.IQR (precisionMeasures), 19
SE.MAD (precisionMeasures), 19
SE.mean (precisionMeasures), 19
SE.meanNArm (measuresWithMissingData), 18
SE.median (precisionMeasures), 19
SE.pearsonskew (precisionMeasures), 19
SE.sd (precisionMeasures), 19
SE.var (precisionMeasures), 19
showHorizontalSignificance (showSignificance), 23
showSignificance, 23
showVerticalSignificance (showSignificance), 23
ShroutFleissICC1, 25
ShroutFleissICC11 (ShroutFleissICC1), 25
ShroutFleissICC1k (ShroutFleissICC1), 25
slope, 26
subjectCenteringTransform, 27
summaryStatistics, 28
superbData, 29
superbPlot, 31
superbPlot.bar, 34
superbPlot.halfwidthline, 35
superbPlot.line, 37
superbPlot.point, 38
superbPlot.pointindividualline, 40
superbPlot.pointjitter, 41
superbPlot.pointjitterviolin, 43
superbPlot.raincloud, 44
superbShiny, 46

TMB1964r, 47
twoStepTransform, 50

WelchDegreeOfFreedom, 51
WinerCompoundSymmetryTest, 52