Suggests: dplyr, psych, emojifont, fMultivar, grid, gridExtra, knitr, lattice, lawstat, boot, png, reshape2, rmarkdown, sadists, scales, testthat, tibble

RdMacros: Rdpack

NeedsCompilation: no

Repository: CRAN

Date/Publication: 2023-01-22 20:50:02 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.lineBand .......................................................... 38
superbPlot.point ............................................................. 40
superbPlot.pointindividualline ................................................ 41
superbPlot.pointjitter .......................................................... 43
superbPlot.pointjitterviolin .................................................... 44
superbPlot.raincloud ........................................................... 46
superbShiny ................................................................. 48
TMB1964r ................................................................. 49
twoStepTransform ............................................................. 51
biasCorrectionTransform

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

References


bootstrapPrecisionMeasures

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


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) )`

---

**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


Examples

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


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 adjusted (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", linewidth = 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", linewidth = 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**

```r
data(dataFigure2)
```

**Format**

An object of class data.frame.

**Source**

doi: 10.5709/acp0214z

**References**


**Examples**

```r
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", linewidth = 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", linewidth = 0.5, linetype=2)
plt2b <- grid.arrange(plt2a,plt2b,ncol=2)

## realise the correct t-test to see the discrepancy

#% 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\% confidence intervals result in a 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**


**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 adjusted (right) 95% confidence intervals

pl3a <- 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", linewidth = 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", linewidth = 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)
# 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\% 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**

```r
data(dataFigure4)
```

**Format**

An object of class data.frame.

**Source**

doi: 10.5709/acp0214z

**References**

Examples

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

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

## realize the plot with unadjusted (left) and adjusted (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", linewidth = 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-adjusted 95% CI") +
                   coord_cartesian( ylim = c(85,115) ) +
                   geom_hline(yintercept = 100, colour = "black", linewidth = 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)
```

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 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, an additional characteristic 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. Lastly, the error bar can be pointing "up" and "down" or go in "both" (the default)

Usage

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

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(ymin=center-width, ymax=center+width, x = grp ) ) +
  geom_superberrorbar()

# an example with left-pointing error bars
nggplot(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 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",
         "(right) 99% confidence intervals",
         "(center, up) 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", color="purple",
                                          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 = 0.25, 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", pointing="up",
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**

```r
GRD(RenameDV = "DV",
SubjectsPerGroup = 100,
BSFactors = "",
WSFactors = "",
Effects = list(),
Population = list(mean = 0, stddev = 1, rho = 0, score = "rnorm(1, mean = GM, sd = STDDEV)",
Contaminant = list(mean = 0, stddev = 1, rho = 0, score = "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 contami-
ant (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 func-
tion base::range.

References


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(
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


---

**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**

```r
makeTransparent()
```

**Value**

does not return anything; set the elements to transparent.

**Examples**

```r
# 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


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)
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. Also note that for repeated-measure design, only CA adjustment is available.

Usage

- \texttt{meanNArm(x)}
- \texttt{SE.meanNArm(x)}
- \texttt{CI.meanNArm(x, gamma)}

Arguments

- \texttt{x} \hspace{1em} a vector of numbers, the sample data (mandatory);
- \texttt{gamma} \hspace{1em} 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 \texttt{\insertAllCites} on this help page.

Examples

```r
# 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 transformation 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 superbPlot() where it performs a modification of the data matrix.

**References**


---

***precisionMeasures***

**Precision measures**

**Description**

superb comes with a few built-in measures of precisions. All SE.fct() functions produce 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


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) )

Description

The following function accept an arbitrary defined degree of freedom (df). This df must be inserted in 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 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

```r
# 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.
```

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

```r
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 global environment 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**

```r
showSignificance(
  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 = NULL,  # (optional) string text to be display on the opposite side of width;
  panel = list(),  # (optional) a list to identify in which panel to put the annotation;
  segmentParams = list(),  # (optional) a list of directives that will be sent to the geom_segment items;
  textParams = list()  # (optional) a list of directives that will be sent to the geom_text item.
)
```

**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**

```r
# 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("Minute(3)"),
  variables = c("DV.1","DV.2","DV.3"),
  adjustments = list(purpose="difference"),
  factorOrder = c("Minute","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
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


Examples

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

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),
    col2 <- c(2, 4, 4, 6, 4, 5, 8, 8, 5, 8, 9, 9),
    col3 <- 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

<table>
<thead>
<tr>
<th>slope</th>
<th>Effect description</th>
</tr>
</thead>
</table>

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)
subjectCenteringTransform

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 superbPlot() where it performs a modification of the data matrix.

References

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


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
superbData

PEARSONSKEW(c(1,2,3)) # the Pearson skew

superbData Obtain summary statistics with correct error bars.

Description

The function `superbData()` computes 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.

Usage

```r
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 adjustments 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

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

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 superb framework. See (Cousineau et al. 2021) for more.

Usage
```r
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.
superbPlot

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 adjustments 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 Randomized 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

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:
pl=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")
pl2=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)") +

# This example is based on repeated measures

superbPlot.bar

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)
superbPlot.halfwidthline

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(),
    errorbarParams = list(),
    errorbarlightParams = 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

**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

```r
# 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)
```
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

```r
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)
superbPlot.lineBand

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)

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(),
  facetParams = list(),
  errorbandParams = 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;
- **facetParams** (optional): list of graphic directives that are sent to the facet_grid layer;
- **errorbandParams** (optional): list of graphic directives that are sent to the geom_ribbon layer;
- **xAsFactor** (optional): Boolean to indicate if the factor on the horizontal should be 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 fictitious time series at 100 time points obtained in two conditions:
dta <- GRD( WSFactors = "timepoints (50) : condition(2)",
SubjectsPerGroup = 100,
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
superbPlot(dta,
WSFactors = c("timepoints(50)", "condition(2)"),
variables = colnames(dta)[2:101], ## all the names of the dataframe except "id"
adjustments = list(  
purpose = "single",
decorrelation = "CM"  ## or none for no decorrelation
),
plotStyle="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 <- superbData(dta,
# WSFactors = c("timepoints(100)", "condition(2)", variables = colnames(dta)[2:201],
# 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**

```r
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;
superbPlot.pointindividualline

**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:

- **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**

```r
# 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

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 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**

```r
superbPlot.pointjitter(
  summarydata,  # a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
  xfactor,  # one of the factors
  groupingfactor,  # one of the factors
  addfactors,  # one or more additional factors
  rawdata,  # a data.frame with the raw data
  pointParams = list(),  # parameters for the points
  jitterParams = list(),  # parameters for the jitter
  errorbarParams = list(),  # parameters for the error bars
  facetParams = list(),  # parameters for the facets
  xAsFactor = TRUE  # whether to treat the x variable as a factor
)
```

**Arguments**

- `summarydata`: a data.frame with columns "center", "lowerwidth" and "upperwidth" for each level of the factors;
- `xfactor`: one of the factors
- `groupingfactor`: one of the factors
- `addfactors`: one or more additional factors
- `rawdata`: a data.frame with the raw data
- `pointParams`: parameters for the points
- `jitterParams`: parameters for the jitter
- `errorbarParams`: parameters for the error bars
- `facetParams`: parameters for the facets
- `xAsFactor`: whether to treat the x variable as a factor
superbPlot.pointjitterviolin

**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

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

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;
superbPlot.raincloud

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 superbShiny() 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 superb 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


Examples

# Launch the user interface:
if (interactive())
  superbShiny()
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


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, 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)
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(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"))
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.

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

References


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.

Value

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

References

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


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, 49

  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.lineBand, 38
superbPlot.point, 40
superbPlot.pointindividualline, 41
superbPlot.pointjitter, 43
superbPlot.pointjitterviolet, 44
superbPlot.raincloud, 46
superbShiny, 48
TMB1964r, 49
twoStepTransform, 51
WelchDegreeOfFreedom, 52
WinerCompoundSymmetryTest, 53