Package ‘faux’

September 13, 2021

Title  Simulation for Factorial Designs

Version  1.1.0

Date  2021-08-30


Depends  R (>= 3.2.4)

Imports  lme4, dplyr, ggplot2 (>= 3.3.0), jsonlite, truncnorm, rlang

License  MIT + file LICENSE

Suggests  testthat (>= 2.1.0), tidyr, knitr, rmarkdown, roxygen2, covr, cowplot, ggExtra, purrr, broom, broom.mixed, psych, lmerTest, kableExtra, glue, openssl, readr

VignetteBuilder  knitr

RoxygenNote  7.1.1

Encoding  UTF-8

LazyData  true

URL  https://github.com/debruine/faux

BugReports  https://github.com/debruine/faux/issues

NeedsCompilation  no

Author  Lisa DeBruine [aut, cre] (<https://orcid.org/0000-0002-7523-5539>), Anna Krystalli [ctb] (<https://orcid.org/0000-0002-2378-4915>), Andrew Heiss [ctb] (<https://orcid.org/0000-0002-3948-3914>)

Maintainer  Lisa DeBruine <debruine@gmail.com>

Repository  CRAN

Date/Publication  2021-09-13 21:40:02 UTC
R topics documented:

add_between ........................................... 3
add_contrast .......................................... 4
add_random ............................................ 5
add_ranef ............................................. 6
add_recode ........................................... 6
add_within ............................................ 7
average_r2tau_0 ..................................... 8
beta2norm ............................................. 8
binom2norm .......................................... 9
check_design ......................................... 10
check_mixed_design ................................. 11
codebook .............................................. 12
codebook_interactive ................................. 13
contr_code_anova .................................... 13
contr_code_difference ............................ 14
contr_code_helmert .................................. 15
contr_code_poly .................................... 16
contr_code_sum ..................................... 16
contr_code_treatment ................................ 17
cormat ................................................... 18
cormat_from_triangle ............................... 19
faceratings ........................................... 19
faux .................................................... 20
faux_options ......................................... 20
fix_name_labels ...................................... 21
fr4 .................................................... 22
gamma2norm ......................................... 22
getcols ............................................... 23
get_design ............................................. 24
get_design_long ..................................... 24
get_params .......................................... 25
interactive_design .................................. 26
is_pos_def .......................................... 27
json_design ......................................... 27
long2wide ............................................ 28
make_id ............................................... 29
messy .................................................. 29
nested_list .......................................... 30
norm2beta ........................................... 30
norm2binom ......................................... 31
norm2gamma ......................................... 32
norm2likert ......................................... 32
norm2pois .......................................... 33
norm2trunc ......................................... 34
norm2unif .......................................... 35
OR ...................................................... 36
**add_between**

Add between factors

**Description**

Add between factors

**Usage**

```
add_between(.data, .by = NULL, ..., .shuffle = FALSE, .prob = NULL)
```

**Arguments**

- `.data` the data frame
- `.by` the grouping column (groups by row if NULL)
- `...` the names and levels of the new factors
- `.shuffle` whether to assign cells randomly or in "order"
- `.prob` probability of each level, equal if NULL

**Value**

data frame
Examples

```r
add_random(subj = 4, item = 2) %>%
  add_between("subj", condition = c("cntl", "test")) %>%
  add_between("item", version = c("A", "B"))
```

---

**add_contrast**  
*Add a contrast to a data frame*

**Description**

Add a contrast to a data frame

**Usage**

```r
add_contrast(
  data, 
  col,
  contrast = c("anova", "sum", "treatment", "helmert", "poly", "difference"),
  levels = NULL,
  ...,
  add_cols = TRUE,
  colnames = NULL
)
```

**Arguments**

- `data`  
  the data frame
- `col`  
  the column to recode
- `contrast`  
  the contrast to recode to
- `levels`  
  the levels of the factor in order
- `...`  
  arguments to pass to the contrast function (base or omit)
- `add_cols`  
  whether to just add the contrast to the existing column or also to create new explicit columns in the dataset (default)
- `colnames`  
  optional list of column names for the added contrasts

**Value**

the data frame with the recoded column and added columns (if `add_cols == TRUE`)
Examples

df <- sim_design(between = list(time = 1:6), plot = FALSE) %>%
    add_contrast("time", "poly")

# test all polynomial contrasts
lm(y ~ time, df) %>% broom::tidy()

# test only the linear and quadratic contrasts
lm(y ~ `time^1` + `time^2`, df) %>% broom::tidy()

add_random

Add random factors to a data structure

Description

Add random factors to a data structure

Usage

add_random(.data = NULL, ..., .nested_in = NULL)

Arguments

.data
  the data frame

... 
  the new random factor column name and the number of values of the random
  factor (if crossed) or the n per group (if nested); can be a vector of n per group
  if nested

.nested_in
  the column(s) to nest in (if NULL, the factor is crossed with all columns)

Value

a data frame

Examples

# start a data frame
data1 <- add_random(school = 3)

# nest classes in schools (2 classes per school)
data2 <- add_random(data1, class = 2, .nested_in = "school")

# nest pupils in each class (different n per class)
data3 <- add_random(data2, pupil = c(20, 24, 23, 21, 25, 24), .nested_in = "class")

# cross each pupil with 10 questions
data4 <- add_random(data3, question = 10)

# compare nesting in 2 different factors
data <- add_random(A = 2, B = 2)
add_random(data, C = 2, .nested_in = "A")
add_random(data, C = 2, .nested_in = "B")
add_ranef  
Add random effects to a data frame

Description
Add random effects to a data frame

Usage
add_ranef(.data, .by = NULL, ..., .cors = 0, .empirical = FALSE)

Arguments
- .data: the data frame
- .by: the grouping column (groups by row if NULL)
- ...: the name and standard deviation of each random effect
- .cors: the correlations among multiple random effects, to be passed to `rnorm_multi` as `r`
- .empirical: logical. To be passed to `rnorm_multi` as `empirical`

Value
data frame with new random effects columns

Examples
add_random(rater = 2, stimulus = 2, time = 2) >>
add_ranef("rater", u0r = 1.5) >>
add_ranef("stimulus", u0s = 2.2, u1s = 0.75, .cors = 0.5) >>
add_ranef(c("rater", "stimulus"), u0sr = 1.2)

add_recode  
Recode a categorical column

Description
Recode a categorical column

Usage
add_recode(.data, .col, .newcol = paste0(col, ".c"), ...)
### add_within

**Add within factors**

**Description**

Add within factors

**Usage**

```r
add_within(.data, .by = NULL, ...)
```

**Arguments**

- `.data` the data frame
- `.by` the grouping column (groups by row if NULL)
- `...` the names and levels of the new factors

**Value**

data frame

**Examples**

```r
add_random(subj = 2, item = 2) %>%
  add_within("subj", time = c("pre", "post"))
```
average_r2tau_0: Average r to Random Intercept SD

**Description**

Average r to Random Intercept SD

**Usage**

`average_r2tau_0(average_r, sigma)`

**Arguments**

- `average_r`: The average inter-item correlation
- `sigma`: Total error variance

**Value**

The standard deviation of the random intercept

beta2norm: Convert beta to normal

**Description**

Convert beta to normal

**Usage**

`beta2norm(x, mu = 0, sd = 1, shape1 = NULL, shape2 = NULL, ...)`

**Arguments**

- `x`: the gamma distributed vector
- `mu`: the mean of the normal distribution to convert to
- `sd`: the SD of the normal distribution to convert to
- `shape1, shape2`: non-negative parameters of the beta distribution
- `...`: further arguments to pass to pbeta (e.g., ncp)

**Value**

a vector with a normal distribution
Examples

```r
x <- rbinom(10000, 2, 3)
y <- binom2norm(x)
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
ggExtra::ggMarginal(g, type = "histogram")
```

**binom2norm**  
*Convert binomial to normal*

**Description**

Convert a binomial distribution to a normal (gaussian) distribution with specified mu and sd.

**Usage**

`binom2norm(x, mu = 0, sd = 1, size = NULL, prob = NULL)`

**Arguments**

- `x` the binomially distributed vector
- `mu` the mean of the normal distribution to return
- `sd` the SD of the normal distribution to return
- `size` number of trials (set to max value of `x` if not specified)
- `prob` the probability of success on each trial (set to mean probability if not specified)

**Value**

a vector with a gaussian distribution

**Examples**

```r
x <- rbinom(10000, 20, 0.75)
y <- binom2norm(x, 0, 1, 20, 0.75)
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
ggExtra::ggMarginal(g, type = "histogram")
```
check_design

Validates the specified design

Description

Specify any number of within- and between-subject factors with any number of levels.

Usage

```r
check_design(
  within = list(),
  between = list(),
  n = 100,
  mu = 0,
  sd = 1,
  r = 0,
  dv = list(y = "value"),
  id = list(id = "id"),
  vardesc = list(),
  plot = faux_options("plot"),
  design = NULL,
  fix_names = FALSE,
  sep = faux_options("sep")
)
```

Arguments

- **within**: a list of the within-subject factors
- **between**: a list of the between-subject factors
- **n**: the number of samples required
- **mu**: a vector giving the means of the variables
- **sd**: the standard deviations of the variables
- **r**: the correlations among the variables (can be a single number, full correlation matrix as a matrix or vector, or a vector of the upper right triangle of the correlation matrix)
- **dv**: the name of the DV column list(y = "value")
- **id**: the name of the ID column list(id = "id")
- **vardesc**: a list of variable descriptions having the names of the within- and between-subject factors
- **plot**: whether to show a plot of the design
- **design**: a design list including within, between, n, mu, sd, r, dv, id
- **fix_names**: deprecated
- **sep**: separator for factor levels
Details

Specify n for each between-subject cell; mu and sd for each cell, and r for the within-subject cells for each between-subject cell.

This function returns a validated design list for use in sim_data to simulate a data table with this design, or to archive your design.

See vignette("sim_design", package = "faux") for details.

Value

list

Examples

```r
within <- list(time = c("day", "night"))
between <- list(pet = c("dog", "cat"))
mu <- list(dog = 10, cat = 5)
vardesc <- list(time = "Time of Day", pet = "Type of Pet")
check_design(within, between, mu = mu, vardesc = vardesc)
```

```r
between <- list(language = c("dutch", "thai"),
                 pet = c("dog", "cat"))
mu <- list(dutch_dog = 12, dutch_cat = 7, thai_dog = 8, thai_cat = 3)
check_design(within, between, mu = mu)
```

check_mixed_design

Get random intercepts for subjects and items

Description

Get error terms from an existing data table.

Usage

```r
check_mixed_design(data, dv = 1, sub_id = 2, item_id = 3, formula = NULL)
```

Arguments

- `data`: the existing tbl
- `dv`: the column name or index containing the DV
- `sub_id`: the column name or index for the subject IDs
- `item_id`: the column name or index for the item IDs
- `formula`: the formula to run in lmer (defaults to null model dv ~ 1 + (1|sub_id) + (1|item_id))

Value

- a list of parameters
Examples

des <- check_mixed_design(fr4, "rating", "rater_id", "face_id")
str(des[1:4])

---

codebook

Create PsychDS Codebook from Data

Description

See vignette("codebook", package = "faux") for details.

Usage

codebook(
  data,
  name = NULL,
  vardesc = list(),
  ...,
  schemaVersion = "Psych-DS 0.1.0",
  return = c("json", "list", "data"),
  interactive = FALSE
)

Arguments

data
  The data frame to generate a codebook for

name
  The name of this dataset (if NULL, will be the same as ‘data’, limited to 64
  characters)

vardesc
  Optional variable properties in the format of a named list of vectors (can be
  named or unnamed and in the same order as the data) from the options “de-
  scription”, “privacy”, “dataType”, “identifier”, “minValue”, “maxValue”, “lev-
  els”, “levelsOrdered”, “na”, “naValue”, “alternateName”, “privacy”, “unitCode”,
  “unitText”

...
  Further dataset properties (e.g., description, license, author, citation, funder, url,
  identifier, keywords, privacyPolicy)

schemaVersion
  defaults to "Psych-DS 0.1.0"

return
  Whether the output should be in JSON format (json), a list (list) or the reformatted
  data with the codebook as an attribute (data)

interactive
  Whether the function should prompt the user to describe columns and factor
  levels

Value

a list or json-formatted codebook, or reformatted data with the codebook as an attribute
Examples

vardesc = list(
    description = c("Length of the sepal",
                    "Width of the sepal",
                    "Length of the petal",
                    "Width of the petal",
                    "The flower species"),
    type = c("float", "float", "float", "float", "string")
)
codebook(iris, vardesc = vardesc)

description

Create a Psych-DS formatted codebook from data by answering questions interactively in the console.

Usage

codebook_interactive(data, cb = NULL)

Arguments

data The data frame to generate a codebook for

Value

codebook list

contr_code_anova

Anova code a factor

Description

Anova coding (also called deviation or simple coding) sets the grand mean as the intercept. Each contrast compares one level with the reference level (base).

Usage

contr_code_anova(fct, levels = NULL, base = 1)
contr_code_difference

Arguments

- `fct` - the factor to contrast code (or a vector)
- `levels` - the levels of the factor in order
- `base` - the index of the level to use as baseline

Value

- the factor with contrasts set

Examples

```r
df <- sim_design(between = list(pet = c("cat", "dog")),
                 mu = c(10, 20), plot = FALSE)
df$pet <- contr_code_anova(df$pet)
lm(y ~ pet, df) %>% broom::tidy()

df <- sim_design(between = list(pet = c("cat", "dog", "ferret")),
                 mu = c(2, 4, 9), empirical = TRUE, plot = FALSE)
df$pet <- contr_code_anova(df$pet, base = 1)
lm(y ~ pet, df) %>% broom::tidy()
df$pet <- contr_code_anova(df$pet, base = 2)
lm(y ~ pet, df) %>% broom::tidy()
df$pet <- contr_code_anova(df$pet, base = "ferret")
lm(y ~ pet, df) %>% broom::tidy()
```

Description

Difference coding sets the grand mean as the intercept. Each contrast compares one level with the previous level.

Usage

```r
contr_code_difference(fct, levels = NULL)
```

Arguments

- `fct` - the factor to contrast code (or a vector)
- `levels` - the levels of the factor in order

Value

- the factor with contrasts set
Examples

```r
df <- sim_design(between = list(pet = c("cat", "dog", "ferret")),
               mu = c(2, 4, 9), empirical = TRUE, plot = FALSE)

df$pet <- contr_code_helmert(df$pet)
lm(y ~ pet, df) %>% broom::tidy()
```

---

**contr_code_helmert**  
_Helmert code a factor_

Description

Helmert coding sets the grand mean as the intercept. Each contrast compares one level with the mean of previous levels.

Usage

```r
contr_code_helmert(fct, levels = NULL)
```

Arguments

- `fct` the factor to contrast code (or a vector)
- `levels` the levels of the factor in order

Value

the factor with contrasts set

Examples

```r
df <- sim_design(between = list(pet = c("cat", "dog")),
               mu = c(10, 20), plot = FALSE)

df$pet <- contr_code_helmert(df$pet)
lm(y ~ pet, df) %>% broom::tidy()

df <- sim_design(between = list(pet = c("cat", "dog", "ferret")),
               mu = c(2, 4, 9), empirical = TRUE, plot = FALSE)

df$pet <- contr_code_helmert(df$pet)
lm(y ~ pet, df) %>% broom::tidy()

# reorder the levels to change the comparisons
df$pet <- contr_code_helmert(df$pet, levels = c("dog", "cat", "ferret"))
lm(y ~ pet, df) %>% broom::tidy()

df$pet <- contr_code_helmert(df$pet, levels = c("ferret", "dog", "cat"))
lm(y ~ pet, df) %>% broom::tidy()
```
**Description**

Polynomial coding sets the grand mean as the intercept. Each contrast tests a trend (linear, quadratic, cubic, etc.). This is only suitable for ordered factors.

**Usage**

```r
contr_code_poly(fct, levels = NULL)
```

**Arguments**

- `fct`: the factor to contrast code (or a vector)
- `levels`: the levels of the factor in order

**Value**

the factor with contrasts set

**Examples**

```r
df <- sim_design(within = list(time = 1:6),
                 mu = 1:6 + (1:6-3.5)^2,
                 long = TRUE, plot = FALSE)

df$time <- contr_code_poly(df$time)
lm(y ~ time, df) %>% broom::tidy()
```

---

**Description**

Sum coding sets the grand mean as the intercept. Each contrast compares one level with the grand mean.

**Usage**

```r
contr_code_sum(fct, levels = NULL, omit = length(levels))
```
contr_code_treatment

Arguments

fct the factor to contrast code (or a vector)
levels the levels of the factor in order
omit the level to omit (defaults to the last level)

Value

the factor with contrasts set

Examples

df <- sim_design(between = list(pet = c("cat", "dog", "bird", "ferret")),
                 mu = c(2, 4, 9, 13), empirical = TRUE, plot = FALSE)

df$pet <- contr_code_sum(df$pet)
lm(y ~ pet, df) %>% broom::tidy()

df$pet <- contr_code_sum(df$pet, omit = "cat")
lm(y ~ pet, df) %>% broom::tidy()

df$pet <- contr_code_sum(df$pet, omit = 1)
lm(y ~ pet, df) %>% broom::tidy()
Examples

df <- sim_design(between = list(pet = c("cat", "dog")),
mu = c(10, 20), plot = FALSE)
df$pet <- contr_code_treatment(df$pet)
lm(y ~ pet, df) %>% broom::tidy()

df <- sim_design(between = list(pet = c("cat", "dog", "ferret")),
mu = c(2, 4, 9), empirical = TRUE, plot = FALSE)
df$pet <- contr_code_treatment(df$pet)
lm(y ~ pet, df) %>% broom::tidy()
df$pet <- contr_code_treatment(df$pet, base = 2)
lm(y ~ pet, df) %>% broom::tidy()
df$pet <- contr_code_treatment(df$pet, base = "ferret")
lm(y ~ pet, df) %>% broom::tidy()

cormat

Make a correlation matrix

description

Cormat makes a correlation matrix from a single number, vars\*vars matrix, vars\*vars vector, or a vars\*(vars-1)/2 vector.

Usage

cormat(cors = 0, vars = 3)

Arguments

cors the correlations among the variables (can be a single number, vars\*vars matrix, vars\*vars vector, or a vars\*(vars-1)/2 vector)
vars the number of variables in the matrix

Value

matrix

Examples

cormat(.5, 3)
cormat(c( 1, .2, .3, .4,
   .2, 1, .5, .6,
   .3, .5, 1, .7,
   .4, .6, .7, 1), 4)
cormat(c(.2, .3, .4, .5, .6, .7), 4)
Make Correlation Matrix from Triangle

cormat_from_triangle makes a correlation matrix from a vector of the upper right triangle

Usage

cormat_from_triangle(cors)

Arguments

cors the correlations among the variables as a vars*(vars-1)/2 vector

Value

matrix

Examples

cormat_from_triangle(c(.2, .3, .4,
                .5, .6,
                .7))

faceratings Attractiveness ratings of faces

A dataset containing attractiveness ratings (on a 1-7 scale from "much less attractiveness than average" to "much more attractive than average") for the neutral front faces from 2513 people (ages 17-90)

Usage

faceratings

Format

A data frame with 256326 rows and 9 variables:

rater_id rater’s ID
rater_sex rater’s sex (female, male, intersex, NA)
rater_age rater’s age (17-90 years)
rater_sexpref rater’s preferred sex for romantic relationships (either, men, neither, women, NA)
faux_options

face_id face’s ID
face_sex face’s sex (female, male)
face_age face’s age (in years)
face_eth face’s ethnic group
rating attractiveness rating on a scale from 1 (much less attractive than average) to 7 (much more attractive than average)

Source

https://figshare.com/articles/dataset/Face_Research_Lab_London_Set/5047666

faux

faux: Simulation Functions.

Description

The faux package provides functions for simulating datasets with specified structure.

faux_options

Set/get global faux options

Description

Global faux options are used, for example, to set the default separator for cell names.

Usage

faux_options(...)

Arguments

... One of four: (1) nothing, then returns all options as a list; (2) a name of an option element, then returns its value; (3) a name-value pair which sets the corresponding option to the new value (and returns nothing), (4) a list with option-value pairs which sets all the corresponding arguments.

Value

a list of options, values of an option, or nothing
Examples

faux_options() # see all options
faux_options("sep") # see value of faux.sep

## Not run:
# changes cell separator (e.g., A1.B2)
faux_options(sep = ".")

# changes cell separator back to default (e.g., A1_B2)
faux_options(sep = "_")

## End(Not run)

fix_name_labels Fix name labels

Description

Fixes if a factor list does not have named levels or has special characters in the names

Usage

fix_name_labels(x, pattern = NA, replacement = ".")

Arguments

x the vector or list to fix
pattern regex pattern to replace; defaults to non-word characters and the value of faux_options("sep")
(default = ")
replacement the character to replace; defaults to . (or _ if faux_options("sep") == ".")

Value

a named list with fixed names

Examples

source <- list("full.stop", " space ", "under_score", "plus+", "dash-", "tab\t", "line\nbreak")
fix_name_labels(source)
fr4  
*Attractiveness rating subset*

**Description**

The faceratings dataset cut down for demos to the first 4 raters of each sex and sexpref and the first 4 faces of each sex and ethnicity with non-NA ages

**Usage**

fr4

**Format**

A data frame with 768 rows and 9 variables:

- *rater_id*  rater’s ID
- *rater_sex*  rater’s sex (female, male)
- *rater_age*  rater’s age (17.4-54.3 years)
- *rater_sexpref*  rater’s preferred sex for romantic relationships (either, men, women)
- *face_id*  face’s ID
- *face_sex*  face’s sex (female, male)
- *face_age*  face’s age (19-47 years)
- *face_eth*  face’s ethnic group (black, east_asian, west_asian, white)
- *rating*  attractiveness rating on a scale from 1 (much less attractive than average) to 7 (much more attractive than average)

**Source**

https://figshare.com/articles/dataset/Face_Research_Lab_London_Set/5047666

---

**gamma2norm**  
*Convert gamma to normal*

**Description**

Convert gamma to normal

**Usage**

```
gamma2norm(x, mu = 0, sd = 1, shape = NULL, rate = 1, scale = 1/rate)
```
Arguments

x  the gamma distributed vector
mu the mean of the normal distribution to convert to
sd the SD of the normal distribution to convert to
shape gamma distribution parameter (must be positive)
rate an alternative way to specify the scale
scale gamma distribution parameter (must be positive)

Value

a vector with a normal distribution

Examples

```r
x <- rgamma(10000, 2)
y <- gamma2norm(x)
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
ggExtra::ggMarginal(g, type = "histogram")
```

getcols

Get data columns

Description

Get columns from a data table by specifying the index, column name as a string, or unquoted column name. Returns the column names or indices.

Usage

```r
getcols(data, ..., as_index = FALSE)
```

Arguments

- data the existing tbl
- ... Columns to get
- as_index return the column indices (defaults to name)

Value

vector of column names or indices

Examples

```r
getcols(mtcars, 1, cyl, "disp", 5:7)
```
get_design

Get design

Description

Get the design specification from a data table created in faux. This can be used to create more simulated data with the same design.

Usage

get_design(data)

Arguments

data The data table to check

Value

list with class design

Examples

data <- sim_design(2, 2, plot = FALSE)
design <- get_design(data)
data2 <- sim_design(design, plot = FALSE)

get_design_long

Get design from long data

Description

Makes a best guess at the design of a long-format data frame.

Usage

generate_design_long(
data,     # data frame
  dv = c(y = "score"),     # dependent variable
  id = c(id = "id"),       # identifier
  plot = faux_options("plot")  # plot options
)


Arguments

- **data**: the data frame (in long format)
- **dv**: the column name that identifies the DV
- **id**: the column name(s) that identify a unit of analysis
- **plot**: whether to show a plot of the design

Details

Finds all columns that contain a single value per unit of analysis (between factors), all columns that contain the same values per unit of analysis (within factors), and all columns that differ over units of analysis (dv, continuous factors)

Value

a design list

---

### Description

Generates a table of the correlations and means of numeric columns in a data frame. If data was generated by `sim_design` and has a "design" attribute, between, within, dv and id are retrieved from that, unless overridden (use between = 0 to

#### Usage

```r
get_params(
  data,
  between = NULL,
  within = NULL,
  dv = NULL,
  id = NULL,
  digits = 2
)
```

```r
check_sim_stats(
  data,
  between = NULL,
  within = NULL,
  dv = NULL,
  id = NULL,
  digits = 2
)
```
interactive_design

**Arguments**

- `data` the existing tbl
- `between` a vector of column names for between-subject factors
- `within` a vector of column names for within-subject factors (if data is long)
- `dv` the column name(s) of the dv, if NULL all numeric columns will be selected
- `id` the column name(s) of the subject ID, excluded from the table even if numeric
- `digits` how many digits to round to (default = 2)

**Value**

- a tbl of correlations, means and sds

**Examples**

```r
get_params(iris, "Species")
```

---

**interactive_design**  
*Set design interactively*

**Description**

Set design interactively

**Usage**

```r
interactive_design(output = c("faux"), plot = faux_options("plot"))
```

**Arguments**

- `output` what type of design to output (faux)
- `plot` whether to show a plot of the design

**Value**

- list

**Examples**

```r
if(interactive()) { des <- interactive_design() }
```
**is_pos_def**

*Check a Matrix is Positive Definite*

**Description**

is_pos_def makes a correlation matrix from a vector

**Usage**

```r
is_pos_def(cor_mat, tol = 1e-08)
```

**Arguments**

- `cor_mat`: a correlation matrix
- `tol`: the tolerance for comparing eigenvalues to 0

**Value**

logical value

**Examples**

```r
is_pos_def(matrix(c(1, .5, .5, 1), 2)) # returns TRUE
is_pos_def(matrix(c(1, .9, .9, 
                   .9, 1, -.2, 
                   .9, -.2, 1), 3)) # returns FALSE
```

**json_design**

*Convert design to JSON*

**Description**

Convert a design list to JSON notation for archiving (e.g. in scienceverse)

**Usage**

```r
json_design(design, filename = NULL, digits = 8, pretty = FALSE, ...)```

**Arguments**

- `design`: a design list including within, between, n, mu, sd, r, dv, id
- `filename`: option name of file to save the json to
- `digits`: number of digits to save
- `pretty`: whether to print condensed or readable
- `...`: other options to send to jsonlite::toJSON
long2wide

Value

a JSON string

Examples

des <- check_design(2, 2)
json_design(des)
json_design(des, pretty = TRUE)

df_long <- sim_design(2, 2, long = TRUE)
long2wide(df_long, "A", "B")

Description

Convert data from long to wide format

Usage

long2wide(data, within = c(), between = c(), dv = "y", id = "id")

Arguments

data the tbl in long format
within the names of the within column(s)
between the names of between column(s) (optional)
dv the name of the DV (value) column
id    the names of the column(s) for grouping observations

Value

a tbl in wide format

Examples

df_long <- sim_design(2, 2, long = TRUE)
long2wide(df_long, "A", "B")
**make_id**  \hspace{1cm}  *Make ID*

**Description**

Make IDs with fixed length and a prefix (e.g., S001, S002, ..., S100).

**Usage**

`make_id(n = 100, prefix = "S", digits = 0, suffix = ")`  

**Arguments**

- **n**  \hspace{1cm} the number of IDs to generate (or a vector of numbers)
- **prefix**  \hspace{1cm} the prefix to the number (default "S")
- **digits**  \hspace{1cm} the number of digits to use for the numeric part. Only used if this is larger than the largest number of digits in n.
- **suffix**  \hspace{1cm} the suffix to the number (default ")")

**Value**

a vector of IDs

**Examples**

```
make_id(20, "SUBJECT_{")
make_id(10:30, digits = 3)
```

---

**messy**  \hspace{1cm} *Simulate missing data*

**Description**

Insert NA or another replacement value for some proportion of specified columns to simulate missing data.

**Usage**

`messy(data, prop = 0, ..., replace = NA)`
nested_list

Output a nested list in RMarkdown list format

Description

Output a nested list in RMarkdown list format

Usage

nested_list(x, pre = "", quote = "")

Arguments

x The list
pre Text to prefix to each line (e.g., if you want all lines indented 4 spaces to start, use " ")
quote Text to quote values with (e.g., use "" to make sure values are not parsed as markdown)

Value

A character string

Arguments

data the tbl
prop the proportion of data to mess up
... the columns to mess up (as a vector of column names or numbers)
replace the replacement value (defaults to NA)

Value

the messed up table

Examples

messy(iris, 0.1, "Species", replace = "NO SPECIES")
messy(iris, 0.5, 1:4)
Examples

```r
x <- list(
  a = list(a1 = "Named", a2 = "List"),
  b = list("Unnamed", "List"),
  c = c(c1 = "Named", c2 = "Vector"),
  d = c("Unnamed", "Vector"),
  e = list(e1 = list("A", "B", "C"),
           e2 = list(a = "A", b = "B"),
           e3 = c("A", "B", "C"),
           e4 = 100),
  f = "single item vector",
  g = list()
)

nested_list(x)
```

Description

Convert normal to beta

Usage

```r
norm2beta(x, shape1, shape2, mu = mean(x), sd = stats::sd(x), ...)
```

Arguments

- **x**: the normally distributed vector
- **shape1**, **shape2**: non-negative parameters of the distribution to return
- **mu**: the mean of x (calculated from x if not given)
- **sd**: the SD of x (calculated from x if not given)
- **...**: further arguments to pass to qbeta (e.g., ncp)

Value

A vector with a beta distribution

Examples

```r
x <- rnorm(10000)
y <- norm2beta(x, 1, 3)
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
ggExtra::ggMarginal(g, type = "histogram")
```
norm2binom

Convert normal to binomial

Description

Convert normal to binomial

Usage

\[
\text{norm2binom}(x, \text{size} = 1, \text{prob} = 0.5, \text{mu} = \text{mean}(x), \text{sd} = \text{stats::sd}(x))
\]

Arguments

- \(x\): the normally distributed vector
- \(\text{size}\): number of trials (0 or more)
- \(\text{prob}\): the probability of success on each trial (0 to 1)
- \(\text{mu}\): the mean of \(x\) (calculated from \(x\) if not given)
- \(\text{sd}\): the SD of \(x\) (calculated from \(x\) if not given)

Value

a vector with a binomial distribution

Examples

\[
x <- \text{rnorm}(10000)
y <- \text{norm2binom}(x)
g <- \text{ggplot2::ggplot()} + \text{ggplot2::geom_point(ggplot2::aes(x, y))}
ggExtra::ggMarginal(g, \text{type} = \text{"histogram"})
\]

norm2gamma

Convert normal to gamma

Description

Convert normal to gamma

Usage

\[
\text{norm2gamma}(x, \text{shape}, \text{rate} = 1, \text{scale} = 1/\text{rate}, \text{mu} = \text{mean}(x), \text{sd} = \text{stats::sd}(x))
\]
**Arguments**

- `x`: the normally distributed vector
- `shape`: gamma distribution parameter (must be positive)
- `rate`: an alternative way to specify the scale
- `scale`: gamma distribution parameter (must be positive)
- `mu`: the mean of `x` (calculated from `x` if not given)
- `sd`: the SD of `x` (calculated from `x` if not given)

**Value**

- a vector with a gamma distribution

**Examples**

```r
x <- rnorm(10000)
y <- norm2gamma(x, shape = 2)
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
ggExtra::ggMarginal(g, type = "histogram")
```

---

**norm2likert**

*Convert normal to likert*

**Description**

Convert normal to likert

**Usage**

```r
norm2likert(x, prob, mu = mean(x), sd = stats::sd(x))
```

**Arguments**

- `x`: the normally distributed vector
- `prob`: a vector of probabilities or counts; if named, the output is a factor
- `mu`: the mean of `x` (calculated from `x` if not given)
- `sd`: the SD of `x` (calculated from `x` if not given)

**Value**

- a vector with the specified distribution
Examples

```r
x <- rnorm(10000)
y <- norm2pois(x, c(.1, .2, .35, .2, .1, .05))
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
  ggExtra::ggMarginal(g, type = "histogram")

y <- norm2pois(x, c(40, 30, 20, 10))
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
  ggExtra::ggMarginal(g, type = "histogram")

y <- norm2pois(x, c(lower = .5, upper = .5))
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
  ggExtra::ggMarginal(g, type = "histogram")
```

---

**norm2pois**

Convert normal to poisson

**Description**

Convert normal to poisson

**Usage**

```r
norm2pois(x, lambda, mu = mean(x), sd = stats::sd(x))
```

**Arguments**

- `x`  
  the normally distributed vector
- `lambda`  
  the mean of the distribution to return
- `mu`  
  the mean of `x` (calculated from `x` if not given)
- `sd`  
  the SD of `x` (calculated from `x` if not given)

**Value**

a vector with a poisson distribution

**Examples**

```r
x <- rnorm(10000)
y <- norm2pois(x, 2)
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
  ggExtra::ggMarginal(g, type = "histogram")
```
norm2trunc  

Convert normal to truncated normal

Description
Convert a normal (gaussian) distribution to a truncated normal distribution with specified minimum and maximum

Usage
norm2trunc(x, min = -Inf, max = Inf, mu = mean(x), sd = stats::sd(x))

Arguments
- x: the normally distributed vector
- min: the minimum of the truncated distribution to return
- max: the maximum of the truncated distribution to return
- mu: the mean of the distribution to return (calculated from x if not given)
- sd: the SD of the distribution to return (calculated from x if not given)

Value
a vector with a uniform distribution

Examples
x <- rnorm(10000)
y <- norm2trunc(x, 1, 7, 3.5, 2)
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
ggExtra::ggMarginal(g, type = "histogram")

---

norm2unif  

Convert normal to uniform

Description
Convert a normal (gaussian) distribution to a uniform distribution with specified minimum and maximum

Usage
norm2unif(x, min = 0, max = 1, mu = mean(x), sd = stats::sd(x))
Arguments

- **x**: the normally distributed vector
- **min**: the minimum of the uniform distribution to return
- **max**: the maximum of the uniform distribution to return
- **mu**: the mean of x (calculated from x if not given)
- **sd**: the SD of x (calculated from x if not given)

Value

A vector with a uniform distribution

Examples

```r
x <- rnorm(10000)
y <- norm2unif(x)
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
ggExtra::ggMarginal(g, type = "histogram")
```

OR

**Piped OR**

Description

LHS if not NULL, otherwise RHS

Usage

`l %||% r`

Arguments

- **l**: LHS.
- **r**: RHS.

Value

LHS if not NULL, otherwise RHS.

Examples

```r
x <- list(b = 2, c = 3)
x$a %||% x$b %||% x$c
x$a %||% "default_value"
```
Plot design

Description

Plots the specified within and between design. See vignette("plots", package = "faux") for examples and details.

Usage

plot_design(x, ..., geoms = NULL, palette = "Dark2", labeller = "label_value")

## S3 method for class 'design'
plot(x, ...)

## S3 method for class 'faux'
plot(x, ...)

Arguments

x
A list of design parameters created by check_design() or a data tbl (in long format)

...  
A list of factor names to determine visualisation (see vignette) in the order color, x, facet row(s), facet col(s)

geoms  
A list of ggplot2 geoms to display, defaults to "pointrangeSD" (mean ± 1SD) for designs and c("violin", "box") for data, options are: pointrangeSD, pointrangeSE, violin, box, jitter

palette  
A brewer palette, defaults to "Dark2" (see ggplot2::scale_colour_brewer)

labeller  
How to label the facets (see ggplot2::facet_grid). "label_value" is used by default.

Value

plot

Methods (by generic)

• plot: Plotting from a faux design list
• plot: Plotting from a faux data table

Examples

within <- list(time = c("day", "night"))
between <- list(pet = c("dog", "cat"))
des <- check_design(within, between, plot = FALSE)
plot_design(des)

data <- sim_design(within, between, plot = FALSE)
plot_design(data)

---

pos_def_limits  
*Limits on Missing Value for Positive Definite Matrix*

**Description**

pos_def_limits returns min and max possible values for a positive definite matrix with a specified missing value.

**Usage**

```r
pos_def_limits(..., steps = 0.01, tol = 1e-08)
```

**Arguments**

- `...` the correlations among the variables as a `vars*(vars-1)/2` vector
- `steps` the tolerance for min and max values
- `tol` the tolerance for comparing eigenvalues to 0

**Value**

dataframe with min and max values

**Examples**

```r
pos_def_limits(.8, .2, NA)
```

---

print.design  
*Print Design List*

**Description**

Print Design List

**Usage**

```r
## S3 method for class 'design'
print(x, ...)
```

**Arguments**

- `x` The design list
- `...` Additional parameters for print
Print Nested List

Description
Print Nested List

Usage

## S3 method for class 'nested_list'
print(x, ...)

Arguments

x       The nested_list string
...

Additional parameters for print

Print Codebook Object

Description
Print Codebook Object

Usage

## S3 method for class 'psychds_codebook'
print(x, ...)

Arguments

x       The psychds_codebook list
...

Additional parameters for print
readline_check

Check readline input

Description

Check readline input

Usage

readline_check(
  prompt,
  type = c("numeric", "integer", "length", "grep"),
  min = -Inf,
  max = Inf,
  warning = NULL,
  default = NULL,
  ...
)

Arguments

prompt the prompt for readline

type what type of check to perform, one of c("numeric", "integer", "length", "grep")

min the minimum value

max the maximum value

warning an optional custom warning message

default the default option to return if the entry is blank, NULL allows no default, the
default value will be displayed after the text as [default]

... other arguments to pass to grep

Value

the validated result of readline

Examples

if(interactive()){  
  readline_check("Type a number: ", "numeric")
  readline_check("Type two characters: ", "length", min = 2, max = 2)
  readline_check("Type at least 3 characters: ", "length", min = 3)
  readline_check("Type no more than 4 characters: ", "length", max = 44)
  readline_check("Type a letter and a number: ", "grep", pattern = "^[a-zA-Z]\d$")
}
rnorm_multi  Multiple correlated normal distributions

Description

Make normally distributed vectors with specified relationships. See vignette("rnorm_multi", package = "faux") for details.

Usage

rnorm_multi(
  n = 100,
  vars = NULL,
  mu = 0,
  sd = 1,
  r = 0,
  varnames = NULL,
  empirical = FALSE,
  as.matrix = FALSE,
  seed = NULL
)

Arguments

n                   the number of samples required  
vars                the number of variables to return 
mu                  a vector giving the means of the variables (numeric vector of length 1 or vars)  
sd                  the standard deviations of the variables (numeric vector of length 1 or vars)  
r                   the correlations among the variables (can be a single number, vars\*vars matrix, vars\*vars vector, or a vars\*(vars-1)/2 vector)  
varnames            optional names for the variables (string vector of length vars) defaults if r is a matrix with column names  
empirical           logical. If true, mu, sd and r specify the empirical not population mean, sd and covariance  
as.matrix           logical. If true, returns a matrix  
seed                DEPRECATED use set.seed() instead before running this function

Value

a tbl of vars vectors
Examples

# 4 10-item vectors each correlated r = .5
rnorm_multi(10, 4, r = 0.5)

# set r with the upper right triangle
b <- rnorm_multi(100, 3, c(0, .5, 1), 1,
                 r = c(0.2, -0.5, 0.5),
                 varnames=c("A", "B", "C"))
cor(b)

# set r with a correlation matrix and column names from mu names
m <- rnorm_multi(
    n = 100,
    mu = c(A = 0, B = 0.5, C = 1),
    r = c(1, 0.2, -0.5,
          0.2, 1, 0.5,
          -0.5, 0.5, 1))
cor(m)

---

rnorm_pre

Make a normal vector correlated to existing vectors

Description

rnorm_pre produces a random normally distributed vector with the specified correlation to one or more existing vectors.

Usage

rnorm_pre(x, mu = 0, sd = 1, r = 0, empirical = FALSE, threshold = 1e-12)

Arguments

- **x**: the existing vector or data table of all vectors
- **mu**: desired mean of returned vector
- **sd**: desired SD of returned vector
- **r**: desired correlation(s) between existing and returned vectors
- **empirical**: logical. If true, mu, sd and r specify the empirical not population mean, sd and covariance
- **threshold**: for checking correlation matrix

Value

vector
Example

```r
v1 <- rnorm(10)
v2 <- rnorm_pre(v1, 0, 1, 0.5)
cor(v1, v2)

x <- rnorm_multi(50, 2, .5)
x$y <- rnorm_pre(x, r = c(0.5, 0.25))
cor(x)
```

---

**sample_from_pop**  
*Sample Parameters from Population Parameters*

**Description**

Sample Parameters from Population Parameters

**Usage**

```r
sample_from_pop(n = 100, mu = 0, sd = 1, r = 0)
```

**Arguments**

- `n` sample size
- `mu` population mean
- `sd` population SD
- `r` population r

**Value**

list of sample parameters (mu, sd, r)

**Examples**

```r
sample_from_pop(10)
```
set_design  
*Set design*

**Description**

Add a design specification to a data table

**Usage**

```r
set_design(data, design)
```

**Arguments**

- `data` The data table
- `design` The design list

**Value**

A data frame with a design attribute

**Examples**

```r
design <- check_design()
data <- data.frame(id = 1:100, y = rnorm(100)) %>%
  set_design(design)
```

sim_design  
*Simulate data from design*

**Description**

Generates a data table with a specified within and between design. See `vignette("sim_design", package = "faux")` for examples and details.

**Usage**

```r
sim_design(
  within = list(),
  between = list(),
  n = 100,
  mu = 0,
  sd = 1,
  r = 0,
  empirical = FALSE,
  long = FALSE,
  dv = list(y = "value"),
)```
```r
id = list(id = "id"),
vardesc = list(),
plot = faux_options("plot"),
interactive = FALSE,
design = NULL,
rep = 1,
nested = TRUE,
seed = NULL,
sep = faux_options("sep")
)
```

**Arguments**

- `within`: a list of the within-subject factors
- `between`: a list of the between-subject factors
- `n`: the number of samples required
- `mu`: the means of the variables
- `sd`: the standard deviations of the variables
- `r`: the correlations among the variables (can be a single number, full correlation matrix as a matrix or vector, or a vector of the upper right triangle of the correlation matrix)
- `empirical`: logical. If true, mu, sd and r specify the empirical not population mean, sd and covariance
- `long`: Whether the returned tbl is in wide (default = FALSE) or long (TRUE) format
- `dv`: the name of the dv for long plots (defaults to y)
- `id`: the name of the id column (defaults to id)
- `vardesc`: a list of variable descriptions having the names of the within- and between-subject factors
- `plot`: whether to show a plot of the design
- `interactive`: whether to run the function interactively
- `design`: a design list including within, between, n, mu, sd, r, dv, id, and vardesc
- `rep`: the number of data frames to return (default 1); if greater than 1, the returned data frame is nested by rep (if nested = TRUE)
- `nested`: Whether to nest data frames by rep if rep > 1
- `seed`: DEPRECATED use set.seed() instead before running this function
- `sep`: separator for factor levels

**Value**

A tbl
Simulate an existing dataframe

Description

Produces a data table with the same distributions and correlations as an existing data table. Only returns numeric columns and simulates all numeric variables from a continuous normal distribution (for now).

Usage

```r
sim_df(
  data,
  n = 100,
  within = c(),
  between = c(),
  id = "id",
  dv = "value",
  empirical = FALSE,
  long = FALSE,
  seed = NULL,
  missing = FALSE
)
```

Arguments

data  the existing tbl
n     the number of samples to return per group
within a list of the within-subject factor columns (if long format)
between a list of the between-subject factor columns
id    the names of the column(s) for grouping observations
dv    the name of the DV (value) column
empirical Should the returned data have these exact parameters? (versus be sampled from a population with these parameters)
long  whether to return the data table in long format
seed  DEPRECATED use set.seed() instead before running this function
missing simulate missing data?

Details

See vignette("sim_df", package = "faux") for details.

Value

a tbl
**Examples**

```r
iris100 <- sim_df(iris, 100)
iris_species <- sim_df(iris, 100, between = "Species")
```

---

**sim_joint_dist**

*Simulate category joint distribution*

---

**Description**

This function is mainly used internally, such as for simulating missing data patterns, but is available in case anyone finds it useful.

**Usage**

```r
sim_joint_dist(data, ..., n = 100, empirical = FALSE)
```

**Arguments**

- `data`: the existing tbl
- `...`: columns to calculate the joint distribution from, if none are chosen, all columns with 10 or fewer unique values will be chosen
- `n`: the number of total observations to return
- `empirical`: Should the returned data have the exact same distribution of conditions? (versus be sampled from a population with this distribution)

**Value**

data table

**Examples**

```r
sim_joint_dist(ggplot2::diamonds, cut, color, n = 10)
```

---

**sim_mixed_cc**

*Generate a cross-classified sample*

---

**Description**

Makes a basic cross-classified design with random intercepts for subjects and items. See vignette("sim_mixed", package = "faux") for examples and details.
Usage

```r
sim_mixed_cc(
  sub_n = 100,
  item_n = 20,
  grand_i = 0,
  sub_sd = 1,
  item_sd = 1,
  error_sd = 1,
  empirical = FALSE,
  seed = NULL
)
```

Arguments

- `sub_n`: the number of subjects
- `item_n`: the number of items
- `grand_i`: the grand intercept (overall mean)
- `sub_sd`: the SD of subject random intercepts (or a sub_n-length named vector of random intercepts for each subject)
- `item_sd`: the SD of item random intercepts (or an item_n-length named vector of random intercepts for each item)
- `error_sd`: the SD of the error term
- `empirical`: Should the returned data have these exact parameters? (versus be sampled from a population with these parameters)
- `seed`: DEPRECATED use set.seed() instead before running this function

Value

a tbl

Examples

```r
sim_mixed_cc(10, 10)
```

---

**sim_mixed_df**

*Generate a mixed design from existing data*

**Description**

`sim_mixed_df()` produces a data table with the same distributions of by-subject and by-item random intercepts as an existing data table.
Usage

sim_mixed_df(
    data,
    sub_n = NULL,
    item_n = NULL,
    dv = "y",
    sub_id = "sub_id",
    item_id = "item_id"
)

Arguments

- **data**: the existing tbl
- **sub_n**: the number of subjects to simulate (if NULL, returns data for the same subjects)
- **item_n**: the number of items to simulate (if NULL, returns data for the same items)
- **dv**: the column name or index containing the DV
- **sub_id**: the column name or index for the subject IDs
- **item_id**: the column name or index for the item IDs

Value

- a tbl

Examples

sim_mixed_df(faceratings, 10, 10, "rating", "rater_id", "face_id")

---

**std_alpha2average_r**  
**Standardized Alpha to Average R**

Description

Standardized Alpha to Average R

Usage

std_alpha2average_r(std_alpha, n)

Arguments

- **std_alpha**: The standarized alpha
- **n**: The number of items

Value

The average inter-item correlation
Examples

```
std_alpha2average_r(.8, 10)
```

---

**trunc2norm**  
*Convert truncated normal to normal*

### Description

Convert a truncated normal distribution to a normal (gaussian) distribution

### Usage

```
trunc2norm(x, min = NULL, max = NULL, mu = mean(x), sd = stats::sd(x))
```

### Arguments

- **x**: the truncated normally distributed vector
- **min**: the minimum of the truncated distribution (calculated from x if not given)
- **max**: the maximum of the truncated distribution (calculated from x if not given)
- **mu**: the mean of the distribution to return (calculated from x if not given)
- **sd**: the SD of the distribution to return (calculated from x if not given)

### Value

a vector with a uniform distribution

### Examples

```
x <- truncnorm::rtruncnorm(10000, 1, 7, 3.5, 2)
y <- trunc2norm(x, 1, 7)
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
ggExtra::ggMarginal(g, type = "histogram")
```
### unif2norm

**Convert uniform to normal**

**Description**
Convert a uniform distribution to a normal (gaussian) distribution with specified mu and sd

**Usage**

```
unif2norm(x, mu = 0, sd = 1, min = NULL, max = NULL)
```

**Arguments**
- `x`: the uniformly distributed vector
- `mu`: the mean of the normal distribution to return
- `sd`: the SD of the normal distribution to return
- `min`: the minimum possible value of x (calculated from x if not given)
- `max`: the maximum possible value of x (calculated from x if not given)

**Value**
a vector with a gaussian distribution

**Examples**

```r
x <- runif(10000)
y <- unif2norm(x)
g <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(x, y))
ggExtra::ggMarginal(g, type = "histogram")
```

### unique_pairs

**Make unique pairs of level names for correlations**

**Description**
Make unique pairs of level names for correlations

**Usage**

```
unique_pairs(v)
```

**Arguments**
- `v`: a vector of level names or a number of levels
Value

a vector of all unique pairs

Examples

unique_pairs(c("O", "C", "E", "A", "N"))
unique_pairs(3)

wide2long

Convert data from wide to long format

Description

Convert data from wide to long format

Usage

wide2long(
  data,
  within_factors = c(),
  within_cols = c(),
  dv = "y",
  id = "id",
  sep = faux_options("sep")
)

Arguments

data the tbl in wide format
within_factors the names of the within factors
within_cols the names (or indices) of the within-subject (value) columns
dv the name of the dv column (defaults to "y")
id the name of the ID column(s) if they don’t exist, a new column will be made
  (defaults to ("id")
sep separator for within-columns (to be used in strsplit, so can be regex), defaults to
  "."

Value

a tbl in long format

Examples

wide2long(iris, c("Feature", "Measure"), 1:4, sep = "\."
Index

* datasets
  faceratings, 19
  fr4, 22
* package
  faux, 20
  add_between, 3
  add_contrast, 4
  add_random, 5
  add_ranef, 6
  add_recode, 6
  add_within, 7
  average_r2tau_0, 8
  beta2norm, 8
  binom2norm, 9
  check_design, 10
  check_mixed_design, 11
  check_sim_stats (get_params), 25
  codebook, 12
  codebook_interactive, 13
  contr_code_anova, 13
  contr_code_difference, 14
  contr_code_helmert, 15
  contr_code_poly, 16
  contr_code_sum, 16
  contr_code_treatment, 17
  cormat, 18
  cormat_from_triangle, 19
  faceratings, 19
  faux, 20
  faux_options, 20
  fix_name_labels, 21
  fr4, 22
  gamma2norm, 22
  get_design, 24
  get_design_long, 24
  get_params, 25
  getcols, 23
  interactive_design, 26
  is_pos_def, 27
  json_design, 27
  long2wide, 28
  make_id, 29
  messy, 29
  nested_list, 30
  norm2beta, 31
  norm2binom, 32
  norm2gamma, 32
  norm2likert, 33
  norm2pois, 34
  norm2trunc, 35
  norm2unif, 35
  OR, 36
  plot.design (plot_design), 37
  plot.faux (plot_design), 37
  plot_design, 37
  pos_def_limits, 38
  print.design, 38
  print.nested_list, 39
  print.psychds_codebook, 39
  readline_check, 40
  rnorm_multi, 6, 41
  rnorm_pre, 42
  sample_from_pop, 43
  set_design, 44
  sim_design, 44
  sim_df, 46
  sim_joint_dist, 47
  sim_mixed_cc, 47

53
simMixedDf, 48
stdAlpha2AverageR, 49
trunc2norm, 50
unif2norm, 51
unique_pairs, 51
wide2long, 52