Package 'datawizard'

January 4, 2022

Type Package

Title Easy Data Wrangling

Version 0.2.2

Maintainer Indrajeet Patil <patilindrajeet.science@gmail.com>

Description A lightweight package to easily manipulate, clean, transform, and prepare your data for analysis. It also forms the data wrangling backend for the packages in the 'easystats' ecosystem.

License GPL-3

URL https://easystats.github.io/datawizard/

BugReports https://github.com/easystats/datawizard/issues

Depends R (>= 3.4)

Imports insight (>= 0.14.4), stats, utils

Suggests bayestestR, boot, dplyr, effectsize, gamm4, ggplot2, knitr, lfe, lme4, MASS, modelbased, parameters, performance, poorman, psych, rmarkdown, rstanarm, see, spelling, testthat (>= 3.0.0), tidyr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.1.2

NeedsCompilation no

Author Dominique Makowski [aut] (<https://orcid.org/0000-0001-5375-9967>, @Dom_Makowski),
Daniel Lüdecke [aut] (<https://orcid.org/0000-0002-8895-3206>, @strengejacke),
Indrajeet Patil [aut, cre] (<https://orcid.org/0000-0003-1995-6531>, @patilindrajeets),
Mattan S. Ben-Shachar [aut] (<https://orcid.org/0000-0002-4287-4801>),
Brenton M. Wiernik [aut] (<https://orcid.org/0000-0001-9560-6336>, @bmwiernik)

Repository CRAN

Date/Publication 2022-01-04 12:10:09 UTC

R topics documented:

`adjust` *Adjust data for the effect of other variable(s)*

Description

This function can be used to adjust the data for the effect of other variables present in the dataset. It is based on an underlying fitting of regressions models, allowing for quite some flexibility, such as including factors as random effects in mixed models (multilevel partialization), continuous variables as smooth terms in general additive models (non-linear partialization) and/or fitting these models under a Bayesian framework. The values returned by this function are the residuals of the regression models. Note that a regular correlation between two "adjusted" variables is equivalent to the partial correlation between them.

Usage

```
adjust(
  data,
  effect = NULL,
  select = NULL,
  exclude = NULL,
  multilevel = FALSE,
  additive = FALSE,
  bayesian = FALSE,
  keep_intercept = FALSE
)

data_adjust(
  data,
  effect = NULL,
  select = NULL,
  exclude = NULL,
  multilevel = FALSE,
  additive = FALSE,
  bayesian = FALSE,
  keep_intercept = FALSE
)
```

Arguments

`data`	A dataframe.
`effect`	Character vector of column names to be adjusted for (regressed out). If `NULL` (the default), all variables will be selected.
`select`	Character vector of column names. If `NULL` (the default), all variables will be selected.
`exclude`	Character vector of column names to be excluded from selection.
`multilevel`	If `TRUE`, the factors are included as random factors. Else, if `FALSE` (default), they are included as fixed effects in the simple regression model.
`additive`	If `TRUE`, continuous variables as included as smooth terms in additive models. The goal is to regress-out potential non-linear effects.
`bayesian`	If `TRUE`, the models are fitted under the Bayesian framework using `rstanarm`.
`keep_intercept`	If `FALSE` (default), the intercept of the model is re-added. This avoids the centering around 0 that happens by default when regressing out another variable (see the examples below for a visual representation of this).

Value

A data frame comparable to `data`, with adjusted variables.

Examples

```
adjusted_all <- adjust(attitude)
head(adjusted_all)
adjusted_one <- adjust(attitude, effect = "complaints", select = "rating")
head(adjusted_one)

adjust(attitude, effect = "complaints", select = "rating", bayesian = TRUE)
adjust(attitude, effect = "complaints", select = "rating", additive = TRUE)
attitude$complaints_LMH <- cut(attitude$complaints, 3)
adjust(attitude, effect = "complaints_LMH", select = "rating", multilevel = TRUE)


if (require("MASS") && require("bayestestR")) {
  # Generate data
  data <- simulate_correlation(n = 100, r = 0.7)
  data$V2 <- (5 * data$V2) + 20 # Add intercept

  # Adjust
  adjusted <- adjust(data, effect = "V1", select = "V2")
  adjusted_icpt <- adjust(data, effect = "V1", select = "V2", keep_intercept = TRUE)

  # Visualize
  plot(data$V1, data$V2,
    pch = 19, col = "blue",
    ylim = c(min(adjusted$V2), max(data$V2)),
    main = "Original (blue), adjusted (green), and adjusted - intercept kept (red) data"
  )
  abline(lm(V2 ~ V1, data = data), col = "blue")
  points(adjusted$V1, adjusted$V2, pch = 19, col = "green")
  abline(lm(V2 ~ V1, data = adjusted), col = "green")
  points(adjusted_icpt$V1, adjusted_icpt$V2, pch = 19, col = "red")
  abline(lm(V2 ~ V1, data = adjusted_icpt), col = "red")
}
```

---

center *Centering (Grand-Mean Centering)*

Description

Performs a grand-mean centering of data.

Usage

```
center(x, ...)

centre(x, ...)

## S3 method for class 'numeric'
center(
```

```
  x,
  robust = FALSE,
  weights = NULL,
  verbose = TRUE,
  reference = NULL,
  center = NULL,
  ...
)

## S3 method for class 'data.frame'
center(
  x,
  robust = FALSE,
  weights = NULL,
  verbose = TRUE,
  reference = NULL,
  select = NULL,
  exclude = NULL,
  remove_na = c("none", "selected", "all"),
  force = FALSE,
  append = FALSE,
  center = NULL,
  ...
)
```

Arguments

x	A data frame, a (numeric or character) vector or a factor.
...	Currently not used.
robust	Logical, if TRUE, centering is done by subtracting the median from the variables. If FALSE, variables are centered by subtracting the mean.
weights	Can be NULL (for no weighting), or: • For data frames: a numeric vector of weights, or a character of the name of a column in the data.frame that contains the weights. • For numeric vectors: a numeric vector of weights.
verbose	Toggle warnings and messages.
reference	A data frame or variable from which the centrality and deviation will be computed instead of from the input variable. Useful for standardizing a subset or new data according to another data frame.
center	Numeric value, which can be used as alternative to reference to define a reference centrality. If center is of length 1, it will be recycled to match the length of selected variables for centering. Else, center must be of same length as the number of selected variables. Values in center will be matched to selected variables in the provided order, unless a named vector is given. In this case, names are matched against the names of the selected variables.
select	Character vector of column names. If NULL (the default), all variables will be selected.

exclude	Character vector of column names to be excluded from selection.
remove_na	How should missing values (NA) be treated: if "none" (default): each column's standardization is done separately, ignoring NAs. Else, rows with NA in the columns selected with select / exclude ("selected") or in all columns ("all") are dropped before standardization, and the resulting data frame does not include these cases.
force	Logical, if TRUE, forces centering of factors as well. Factors are converted to numerical values, with the lowest level being the value 1 (unless the factor has numeric levels, which are converted to the corresponding numeric value).
append	Logical or string. If TRUE, centered variables get new column names (with the suffix "_c") and are appended (column bind) to x, thus returning both the original and the centered variables. If FALSE, original variables in x will be overwritten by their centered versions. If a character value, centered variables are appended with new column names (using the defined suffix) to the original data frame.

Value

The centered variables.

Note

Difference between centering and standardizing: Standardized variables are computed by subtracting the mean of the variable and then dividing it by the standard deviation, while centering variables involves only the subtraction.

See Also

If centering within-clusters (instead of grand-mean centering) is required, see demean(). For standardizing, see standardize().

Examples

```
data(iris)

# entire dataframe or a vector
head(iris$Sepal.Width)
head(center(iris$Sepal.Width))
head(center(iris))
head(center(iris, force = TRUE))

# only the selected columns from a dataframe
center(anscombe, select = c("x1", "x3"))
center(anscombe, exclude = c("x1", "x3"))

# centering with reference center and scale
d <- data.frame(
  a = c(-2, -1, 0, 1, 2),
  b = c(3, 4, 5, 6, 7)
)
```

```
# default centering at mean
center(d)

# centering, using 0 as mean
center(d, center = 0)

# centering, using -5 as mean
center(d, center = -5)
```

`convert_data_to_numeric` *Convert data to numeric*

Description

Convert data to numeric by converting characters to factors and factors to either numeric levels or dummy variables.

Usage

```
convert_data_to_numeric(x, dummy_factors = TRUE, ...)

data_to_numeric(x, dummy_factors = TRUE, ...)
```

Arguments

`x`	A data frame or a vector.
`dummy_factors`	Transform factors to dummy factors (all factor levels as different columns filled with a binary 0-1 value).
`...`	Arguments passed to or from other methods.

Value

A data frame of numeric variables.

Examples

```
head(convert_data_to_numeric(iris))
```

data_addprefix *Convenient dataframe manipulation functionalities*

Description

Safe and intuitive functions to manipulate dataframes.

Usage

```
data_addprefix(data, pattern, ...)

data_addsuffix(data, pattern, ...)

data_findcols(data, pattern = NULL, starts_with = NULL, ends_with = NULL, ...)

data_remove(data, pattern, ...)

data_rename(data, pattern = NULL, replacement = NULL, safe = TRUE, ...)

data_rename_rows(data, rows = NULL)

data_reorder(data, cols, safe = TRUE, ...)
```

Arguments

data	A data frame, or an object that can be coerced to a data frame.
pattern, replacement, starts_with, ends_with	Character strings.
...	Other arguments passed to or from other functions.
safe	Do not throw error if for instance the variable to be renamed/removed doesn't exist.
cols, rows	Vector of column or row names.

Value

A modified data frame.

Examples

```
# Add prefix / suffix to all columns
head(data_addprefix(iris, "NEW_"))
head(data_addsuffix(iris, "_OLD"))
# Find columns names by pattern
data_findcols(iris, starts_with = "Sepal")
data_findcols(iris, ends_with = "Width")
data_findcols(iris, pattern = "\\.")
data_findcols(iris, c("Petal.Width", "Sepal.Length"))
```

```
# Remove columns
head(data_remove(iris, "Sepal.Length"))
# Rename columns
head(data_rename(iris, "Sepal.Length", "length"))
# data_rename(iris, "FakeCol", "length", safe=FALSE)  # This fails
head(data_rename(iris, "FakeCol", "length")) # This doesn't
head(data_rename(iris, c("Sepal.Length", "Sepal.Width"), c("length", "width")))

# Reset names
head(data_rename(iris, NULL))

# Change all
head(data_rename(iris, paste0("Var", 1:5)))
# Reorder columns
head(data_reorder(iris, c("Species", "Sepal.Length")))
head(data_reorder(iris, c("Species", "dupa"))) # Safe for non-existing cols
```

`data_extract` *Extract a single column or element from an object*

Description

`extract()` is similar to `$`. It extracts a single column or element from an object (e.g., a data frame, list,)

Usage

```
data_extract(data, select, name = NULL, ...)

extract(data, select, name = NULL, ...)
```

Arguments

`data`	The object to subset. Methods are currently available for data frames and data frame extensions (e.g., tibbles).
`select`	A variable specified as: • a literal variable name (e.g., `column_name`) • a character vector with the variable name (e.g., `"column_name"`) • a positive integer, giving the position counting from the left • a negative integer, giving the position counting from the right. The default returns the last column. If the special value `0` or `"row.names"` is given, the row names of the object (if any) are extracted.
`name`	An optional argument that specifies the column to be used as names for for the vector after extraction. Specified in the same way as `select`.
`...`	For use by future methods.

Value

A vector containing the extracted element.

Examples

```
extract(mtcars, cyl, name = gear)
extract(mtcars, "cyl", name = gear)
extract(mtcars, -1, name = gear)
extract(mtcars, cyl, name = 0)
extract(mtcars, cyl, name = "row.names")
```

---

data_match *Find row indices of a data frame matching a specific condition*

Description

Find row indices of a data frame that match a specific condition.

Usage

```
data_match(x, to, ...)
```

Arguments

x	A data frame.
to	A data frame matching the specified conditions.
...	Other arguments passed to or from other functions.

Value

A dataframe containing rows that match the specified configuration.

Examples

```
matching_rows <- data_match(mtcars, data.frame(vs = 0, am = 1))
mtcars[matching_rows, ]

matching_rows <- data_match(mtcars, data.frame(vs = 0, am = c(0, 1)))
mtcars[matching_rows, ]
```

`data_partition` *Partition data into a test and a training set*

Description

Creates a training and a test set based on a dataframe. Can also be stratified (i.e., evenly spread a given factor) using the `group` argument.

Usage

```
data_partition(data, training_proportion = 0.7, group = NULL, seed = NULL, ...)
```

Arguments

`data`	A data frame, or an object that can be coerced to a data frame.
`training_proportion`	The proportion (between 0 and 1) of the training set. The remaining part will be used for the test set.
`group`	A character vector indicating the name(s) of the column(s) used for stratified partitioning.
`seed`	A random number generator seed. Enter an integer (e.g. 123) so that the random sampling will be the same each time you run the function.
`...`	Other arguments passed to or from other functions.

Value

A list of two data frames, named `test` and `training`.

Examples

```
df <- iris
df$Smell <- rep(c("Strong", "Light"), 75)

data_partition(df)
data_partition(df, group = "Species")
data_partition(df, group = c("Species", "Smell"))
```

data_relocate *Relocate (reorder) columns of a data frame*

Description

Relocate (reorder) columns of a data frame

Usage

```
data_relocate(data, cols, before = NULL, after = NULL, safe = TRUE, ...)
```

Arguments

data	A data frame to pivot.
cols	A character vector indicating the names of the columns to move.
before, after	Destination of columns. Supplying neither will move columns to the left-hand side; specifying both is an error.
safe	If TRUE, will disregard non-existing columns.
...	Other arguments passed to or from other functions.

Value

A data frame with reordered columns.

Examples

```
# Reorder columns
head(data_relocate(iris, cols = "Species", before = "Sepal.Length"))
head(data_relocate(iris, cols = "Species", before = "Sepal.Width"))
head(data_relocate(iris, cols = "Sepal.Width", after = "Species"))
head(data_relocate(iris, cols = c("Species", "Petal.Length"), after = "Sepal.Width"))
```

data_rescale *Rescale Variables to a New Range*

Description

Rescale variables to a new range.

Usage

```
data_rescale(x, ...)

change_scale(x, ...)

## S3 method for class 'numeric'
data_rescale(x, to = c(0, 100), range = NULL, verbose = TRUE, ...)

## S3 method for class 'grouped_df'
data_rescale(
  x,
  to = c(0, 100),
  range = NULL,
  select = NULL,
  exclude = NULL,
  ...
)

## S3 method for class 'data.frame'
data_rescale(
  x,
  to = c(0, 100),
  range = NULL,
  select = NULL,
  exclude = NULL,
  ...
)
```

Arguments

x	A numeric variable.
...	Arguments passed to or from other methods.
to	New range that the variable will have after rescaling.
range	Initial (old) range of values. If NULL, will take the range of the input vector (range(x)).
verbose	Toggle warnings and messages on or off.
select	Character vector of column names. If NULL (the default), all variables will be selected.
exclude	Character vector of column names to be excluded from selection.

Value

A rescaled object.

See Also

normalize() standardize() ranktransform()

Other transform utilities: normalize(), ranktransform(), standardize()

Examples

```
data_rescale(c(0, 1, 5, -5, -2))
data_rescale(c(0, 1, 5, -5, -2), to = c(-5, 5))

# Specify the "theoretical" range of the input vector
data_rescale(c(1, 3, 4), to = c(0, 40), range = c(0, 4))

# Dataframes
head(data_rescale(iris, to = c(0, 1)))
head(data_rescale(iris, to = c(0, 1), select = "Sepal.Length"))

# One can specify a list of ranges
head(data_rescale(iris, to = list(
  "Sepal.Length" = c(0, 1),
  "Petal.Length" = c(-1, 0)
)))
```

---

data_restoretype *Restore the type of columns according to a reference data frame*

Description

Restore the type of columns according to a reference data frame

Usage

```
data_restoretype(data, reference = NULL, ...)
```

Arguments

data	A data frame to pivot.
reference	A reference data frame from which to find the correct column types.
...	Additional arguments passed on to methods.

Value

A dataframe with columns whose types have been restored based on the reference dataframe.

Examples

```
data <- data.frame(
  Sepal.Length = c("1", "3", "2"),
  Species = c("setosa", "versicolor", "setosa"),
  New = c("1", "3", "4")
)

fixed <- data_restoretype(data, reference = iris)
summary(fixed)
```

data_to_long *Reshape (pivot) data from wide to long*

Description

This function "lengthens" data, increasing the number of rows and decreasing the number of columns. This is a dependency-free base-R equivalent of `tidyr::pivot_longer()`.

Usage

```
data_to_long(
  data,
  cols = "all",
  colnames_to = "Name",
  values_to = "Value",
  rows_to = NULL,
  ...,
  names_to = colnames_to
)

data_to_wide(
  data,
  values_from = "Value",
  colnames_from = "Name",
  rows_from = NULL,
  sep = "_",
  ...,
  names_from = colnames_from
)

reshape_longer(
  data,
  cols = "all",
  colnames_to = "Name",
  values_to = "Value",
  rows_to = NULL,
  ...,
  names_to = colnames_to
)

reshape_wider(
  data,
  values_from = "Value",
  colnames_from = "Name",
  rows_from = NULL,
  sep = "_",
  ...,
```

```
  names_from = colnames_from
)
```

Arguments

data	A data frame to pivot.
cols	A vector of column names or indices to pivot into longer format.
colnames_to	The name of the new column that will contain the column names.
values_to	The name of the new column that will contain the values of the pivoted variables.
rows_to	The name of the column that will contain the row-number from the original data. If NULL, will be removed.
...	Additional arguments passed on to methods.
names_to, names_from	Same as colnames_to, is there for compatibility with tidyr::pivot_longer().
values_from	The name of the column that contains the values of the put in the columns.
colnames_from	The name of the column that contains the levels to be used as future columns.
rows_from	The name of the column that identifies the rows. If NULL, will use all the unique rows.
sep	The indicating a separating character in the variable names in the wide format.

Value

data.frame

Examples

```
wide_data <- data.frame(replicate(5, rnorm(10)))

# From wide to long
# ------------------
# Default behaviour (equivalent to tidyr::pivot_longer(wide_data, cols = 1:5))
data_to_long(wide_data)

# Customizing the names
data_to_long(wide_data,
  cols = c(1, 2),
  colnames_to = "Column",
  values_to = "Numbers",
  rows_to = "Row"
)

# From long to wide
# -----------------
long_data <- data_to_long(wide_data, rows_to = "Row_ID") # Save row number
data_to_wide(long_data,
  colnames_from = "Name",
  values_from = "Value",
  rows_from = "Row_ID"
```

```
)

# Full example
# ------------------
if (require("psych")) {
  data <- psych::bfi # Wide format with one row per participant's personality test

  # Pivot long format
  long <- data_to_long(data,
    cols = "\\d", # Select all columns that contain a digit
    colnames_to = "Item",
    values_to = "Score",
    rows_to = "Participant"
  )

  # Separate facet and question number
  long$Facet <- gsub("\\d", "", long$Item)
  long$Item <- gsub("[A-Z]", "", long$Item)
  long$Item <- paste0("I", long$Item)

  wide <- data_to_wide(long,
    colnames_from = "Item",
    values_from = "Score"
  )
  head(wide)
}
```

`data_transpose` *Transpose a dataframe*

Description

Transpose a dataframe. It's the equivalent of using `t()` but restores the data.frame class, and prints a warning if the data type is modified (see example).

Usage

```
data_transpose(data, verbose = TRUE, ...)
```

Arguments

`data`	A data frame, or an object that can be coerced to a data frame.
`verbose`	Silence warnings and/or messages by setting it to `FALSE`.
`...`	Other arguments passed to or from other functions.

Examples

```
transposed <- data_transpose(iris)
transposed[1:5]

transposed <- data_transpose(iris[1:4]) # Only numeric = no warning
```

---

demean *Compute group-meaned and de-meaned variables*

Description

demean() computes group- and de-meaned versions of a variable that can be used in regression analysis to model the between- and within-subject effect. degroup() is more generic in terms of the centering-operation. While demean() always uses mean-centering, degroup() can also use the mode or median for centering.

Usage

```
demean(
  x,
  select,
  group,
  suffix_demean = "_within",
  suffix_groupmean = "_between",
  add_attributes = TRUE,
  verbose = TRUE
)

degroup(
  x,
  select,
  group,
  center = "mean",
  suffix_demean = "_within",
  suffix_groupmean = "_between",
  add_attributes = TRUE,
  verbose = TRUE
)

detrend(
  x,
  select,
  group,
  center = "mean",
  suffix_demean = "_within",
  suffix_groupmean = "_between",
  add_attributes = TRUE,
```

```
  verbose = TRUE
)
```

Arguments

x	A data frame.
select	Character vector (or formula) with names of variables to select that should be group- and de-meaned.
group	Character vector (or formula) with the name of the variable that indicates the group- or cluster-ID.
suffix_demean, suffix_groupmean	String value, will be appended to the names of the group-meaned and de-meaned variables of x. By default, de-meaned variables will be suffixed with "_within" and grouped-meaned variables with "_between".
add_attributes	Logical, if TRUE, the returned variables gain attributes to indicate the within- and between-effects. This is only relevant when printing model_parameters() - in such cases, the within- and between-effects are printed in separated blocks.
verbose	Toggle warnings and messages.
center	Method for centering. demean() always performs mean-centering, while degroup() can use center = "median" or center = "mode" for median- or mode-centering, and also "min" or "max".

Details

Heterogeneity Bias: Mixed models include different levels of sources of variability, i.e. error terms at each level. When macro-indicators (or level-2 predictors, or higher-level units, or more general: *group-level predictors that* **vary** *within and across groups*) are included as fixed effects (i.e. treated as covariate at level-1), the variance that is left unaccounted for this covariate will be absorbed into the error terms of level-1 and level-2 (*Bafumi and Gelman 2006; Gelman and Hill 2007, Chapter 12.6.*): "Such covariates contain two parts: one that is specific to the higher-level entity that does not vary between occasions, and one that represents the difference between occasions, within higher-level entities" (*Bell et al. 2015*). Hence, the error terms will be correlated with the covariate, which violates one of the assumptions of mixed models (iid, independent and identically distributed error terms). This bias is also called the *heterogeneity bias* (*Bell et al. 2015*). To resolve this problem, level-2 predictors used as (level-1) covariates should be separated into their "within" and "between" effects by "de-meaning" and "group-meaning": After demeaning time-varying predictors, "at the higher level, the mean term is no longer constrained by Level 1 effects, so it is free to account for all the higher-level variance associated with that variable" (*Bell et al. 2015*).

Panel data and correlating fixed and group effects: demean() is intended to create group- and de-meaned variables for panel regression models (fixed effects models), or for complex random-effect-within-between models (see *Bell et al. 2015, 2018*), where group-effects (random effects) and fixed effects correlate (see *Bafumi and Gelman 2006*). This can happen, for instance, when analyzing panel data, which can lead to *Heterogeneity Bias*. To control for correlating predictors and group effects, it is recommended to include the group-meaned and de-meaned version of *time-varying covariates* (and group-meaned version of *time-invariant covariates* that are on a higher

level, e.g. level-2 predictors) in the model. By this, one can fit complex multilevel models for panel data, including time-varying predictors, time-invariant predictors and random effects.

Why mixed models are preferred over fixed effects models: A mixed models approach can model the causes of endogeneity explicitly by including the (separated) within- and between-effects of time-varying fixed effects and including time-constant fixed effects. Furthermore, mixed models also include random effects, thus a mixed models approach is superior to classic fixed-effects models, which lack information of variation in the group-effects or between-subject effects. Furthermore, fixed effects regression cannot include random slopes, which means that fixed effects regressions are neglecting "cross-cluster differences in the effects of lower-level controls (which) reduces the precision of estimated context effects, resulting in unnecessarily wide confidence intervals and low statistical power" (*Heisig et al. 2017*).

Terminology: The group-meaned variable is simply the mean of an independent variable within each group (or id-level or cluster) represented by `group`. It represents the cluster-mean of an independent variable. The regression coefficient of a group-meaned variable is the *between-subject-effect*. The de-meaned variable is then the centered version of the group-meaned variable. De-meaning is sometimes also called person-mean centering or centering within clusters. The regression coefficient of a de-meaned variable represents the *within-subject-effect*.

De-meaning with continuous predictors: For continuous time-varying predictors, the recommendation is to include both their de-meaned and group-meaned versions as fixed effects, but not the raw (untransformed) time-varying predictors themselves. The de-meaned predictor should also be included as random effect (random slope). In regression models, the coefficient of the de-meaned predictors indicates the within-subject effect, while the coefficient of the group-meaned predictor indicates the between-subject effect.

De-meaning with binary predictors: For binary time-varying predictors, there are two recommendations. First is to include the raw (untransformed) binary predictor as fixed effect only and the *de-meaned* variable as random effect (random slope). The alternative would be to add the de-meaned version(s) of binary time-varying covariates as additional fixed effect as well (instead of adding it as random slope). Centering time-varying binary variables to obtain within-effects (level 1) isn't necessary. They have a sensible interpretation when left in the typical 0/1 format (*Hoffmann 2015, chapter 8-2.I*). `demean()` will thus coerce categorical time-varying predictors to numeric to compute the de- and group-meaned versions for these variables, where the raw (untransformed) binary predictor and the de-meaned version should be added to the model.

De-meaning of factors with more than 2 levels: Factors with more than two levels are de-meaned in two ways: first, these are also converted to numeric and de-meaned; second, dummy variables are created (binary, with 0/1 coding for each level) and these binary dummy-variables are de-meaned in the same way (as described above). Packages like **panelr** internally convert factors to dummies before demeaning, so this behaviour can be mimicked here.

De-meaning interaction terms: There are multiple ways to deal with interaction terms of within- and between-effects. A classical approach is to simply use the product term of the de-meaned variables (i.e. introducing the de-meaned variables as interaction term in the model formula, e.g. `y ~ x_within * time_within`). This approach, however, might be subject to bias (see *Giesselmann & Schmidt-Catran 2020*).

Another option is to first calculate the product term and then apply the de-meaning to it. This

approach produces an estimator "that reflects unit-level differences of interacted variables whose moderators vary within units", which is desirable if *no* within interaction of two time-dependent variables is required.

A third option, when the interaction should result in a genuine within estimator, is to "double de-mean" the interaction terms (*Giesselmann & Schmidt-Catran 2018*), however, this is currently not supported by `demean()`. If this is required, the `wmb()` function from the **panelr** package should be used.

To de-mean interaction terms for within-between models, simply specify the term as interaction for the `select`-argument, e.g. `select = "a*b"` (see 'Examples').

Analysing panel data with mixed models using lme4: A description of how to translate the formulas described in *Bell et al. 2018* into R using `lmer()` from **lme4** can be found in this vignette.

Value

A data frame with the group-/de-meaned variables, which get the suffix `"_between"` (for the group-meaned variable) and `"_within"` (for the de-meaned variable) by default.

References

- Bafumi J, Gelman A. 2006. Fitting Multilevel Models When Predictors and Group Effects Correlate. In. Philadelphia, PA: Annual meeting of the American Political Science Association.
- Bell A, Fairbrother M, Jones K. 2019. Fixed and Random Effects Models: Making an Informed Choice. Quality & Quantity (53); 1051-1074
- Bell A, Jones K. 2015. Explaining Fixed Effects: Random Effects Modeling of Time-Series Cross-Sectional and Panel Data. Political Science Research and Methods, 3(1), 133–153.
- Gelman A, Hill J. 2007. Data Analysis Using Regression and Multilevel/Hierarchical Models. Analytical Methods for Social Research. Cambridge, New York: Cambridge University Press
- Giesselmann M, Schmidt-Catran, AW. 2020. Interactions in fixed effects regression models. Sociological Methods & Research, 1–28. https://doi.org/10.1177/0049124120914934
- Heisig JP, Schaeffer M, Giesecke J. 2017. The Costs of Simplicity: Why Multilevel Models May Benefit from Accounting for Cross-Cluster Differences in the Effects of Controls. American Sociological Review 82 (4): 796–827.
- Hoffman L. 2015. Longitudinal analysis: modeling within-person fluctuation and change. New York: Routledge

See Also

If grand-mean centering (instead of centering within-clusters) is required, see `center()`.

Examples

```
data(iris)
iris$ID <- sample(1:4, nrow(iris), replace = TRUE) # fake-ID
iris$binary <- as.factor(rbinom(150, 1, .35)) # binary variable

x <- demean(iris, select = c("Sepal.Length", "Petal.Length"), group = "ID")
head(x)

x <- demean(iris, select = c("Sepal.Length", "binary", "Species"), group = "ID")
head(x)


# demean interaction term x*y
dat <- data.frame(
  a = c(1, 2, 3, 4, 1, 2, 3, 4),
  x = c(4, 3, 3, 4, 1, 2, 1, 2),
  y = c(1, 2, 1, 2, 4, 3, 2, 1),
  ID = c(1, 2, 3, 1, 2, 3, 1, 2)
)
demean(dat, select = c("a", "x*y"), group = "ID")

# or in formula-notation
demean(dat, select = ~ a + x * y, group = ~ID)
```

---

describe_distribution *Describe a distribution*

Description

This function describes a distribution by a set of indices (e.g., measures of centrality, dispersion, range, skewness, kurtosis).

Usage

```
describe_distribution(x, ...)

## S3 method for class 'numeric'
describe_distribution(
  x,
  centrality = "mean",
  dispersion = TRUE,
  iqr = TRUE,
  range = TRUE,
  quartiles = FALSE,
  ci = NULL,
  iterations = 100,
  threshold = 0.1,
  verbose = TRUE,
```

```
  ...
)

## S3 method for class 'factor'
describe_distribution(x, dispersion = TRUE, range = TRUE, verbose = TRUE, ...)

## S3 method for class 'data.frame'
describe_distribution(
  x,
  centrality = "mean",
  dispersion = TRUE,
  iqr = TRUE,
  range = TRUE,
  quartiles = FALSE,
  include_factors = FALSE,
  ci = NULL,
  iterations = 100,
  threshold = 0.1,
  verbose = TRUE,
  ...
)
```

Arguments

x	A numeric vector.
...	Additional arguments to be passed to or from methods.
centrality	The point-estimates (centrality indices) to compute. Character (vector) or list with one or more of these options: "median", "mean", "MAP" or "all".
dispersion	Logical, if TRUE, computes indices of dispersion related to the estimate(s) (SD and MAD for mean and median, respectively).
iqr	Logical, if TRUE, the interquartile range is calculated (based on stats::IQR(), using type = 6).
range	Return the range (min and max).
quartiles	Return the first and third quartiles (25th and 75pth percentiles).
ci	Confidence Interval (CI) level. Default is NULL, i.e. no confidence intervals are computed. If not NULL, confidence intervals are based on bootstrap replicates (see iterations). If centrality = "all", the bootstrapped confidence interval refers to the first centrality index (which is typically the median).
iterations	The number of bootstrap replicates for computing confidence intervals. Only applies when ci is not NULL.
threshold	For centrality = "trimmed" (i.e. trimmed mean), indicates the fraction (0 to 0.5) of observations to be trimmed from each end of the vector before the mean is computed.
verbose	Toggle warnings and messages.

include_factors
: Logical, if TRUE, factors are included in the output, however, only columns for range (first and last factor levels) as well as n and missing will contain information.

Value

A data frame with columns that describe the properties of the variables.

Note

There is also a plot()-method implemented in the **see**-package.

Examples

```
describe_distribution(rnorm(100))

data(iris)
describe_distribution(iris)
describe_distribution(iris, include_factors = TRUE, quartiles = TRUE)
```

---

format_text *Convenient text formatting functionalities*

Description

Convenience functions to manipulate and format text.

Usage

```
format_text(text, sep = ", ", last = " and ", width = NULL, ...)

text_fullstop(text)

text_lastchar(text, n = 1)

text_concatenate(text, sep = ", ", last = " and ")

text_paste(text, text2 = NULL, sep = ", ", ...)

text_remove(text, pattern = "", ...)

text_wrap(text, width = NULL, ...)
```

Arguments

text, text2	A character string.
sep	Separator.
last	Last separator.
width	Positive integer giving the target column width for wrapping lines in the output. Can be "auto", in which case it will select 90\ default width.
...	Other arguments to be passed to or from other functions.
n	The number of characters to find.
pattern	Character strings.

Value

A character string.

Examples

```
# Add full stop if missing
text_fullstop(c("something", "something else."))

# Find last characters
text_lastchar(c("ABC", "DEF"), n = 2)

# Smart concatenation
text_concatenate(c("First", "Second", "Last"))

# Remove parts of string
text_remove(c("one!", "two", "three!"), "!")

# Wrap text
long_text <- paste(rep("abc ", 100), collapse = "")
cat(text_wrap(long_text, width = 50))

# Paste with optional separator
text_paste(c("A", "", "B"), c("42", "42", "42"))
```

---

nhanes_sample *Sample dataset from the National Health and Nutrition Examination Survey*

Description

Selected variables from the National Health and Nutrition Examination Survey that are used in the example from Lumley (2010), Appendix E.

References

Lumley T (2010). Complex Surveys: a guide to analysis using R. Wiley

normalize *Normalize numeric variable to 0-1 range*

Description

Performs a normalization of data, i.e., it scales variables in the range 0 -

1. This is a special case of data_rescale().

Usage

```
normalize(x, ...)

## S3 method for class 'numeric'
normalize(x, include_bounds = TRUE, verbose = TRUE, ...)

## S3 method for class 'grouped_df'
normalize(
  x,
  select = NULL,
  exclude = NULL,
  include_bounds = TRUE,
  verbose = TRUE,
  ...
)

## S3 method for class 'data.frame'
normalize(
  x,
  select = NULL,
  exclude = NULL,
  include_bounds = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

x	A numeric vector, data frame, or matrix. See details.
...	Arguments passed to or from other methods.
include_bounds	Logical, if TRUE, return value may include 0 and 1. If FALSE, the return value is compressed, using Smithson and Verkuilen's (2006) formula (x * (n -1) + 0.5) / n, to avoid zeros and ones in the normalized variables. This can be useful in case of beta-regression, where the response variable is not allowed to include zeros and ones.
verbose	Toggle warnings and messages on or off.

select	Character vector of column names. If NULL (the default), all variables will be selected.
exclude	Character vector of column names to be excluded from selection.

Details

- If x is a matrix, normalization is performed across all values (not column- or row-wise). For column-wise normalization, convert the matrix to a data.frame.
- If x is a grouped data frame (grouped_df), normalization is performed separately for each group.

Value

A normalized object.

References

Smithson M, Verkuilen J (2006). A Better Lemon Squeezer? Maximum-Likelihood Regression with Beta-Distributed Dependent Variables. Psychological Methods, 11(1), 54–71.

See Also

Other transform utilities: data_rescale(), ranktransform(), standardize()

Examples

```
normalize(c(0, 1, 5, -5, -2))
normalize(c(0, 1, 5, -5, -2), include_bounds = FALSE)

head(normalize(trees))
```

---

ranktransform *(Signed) rank transformation*

Description

Transform numeric values with the integers of their rank (i.e., 1st smallest, 2nd smallest, 3rd smallest, etc.). Setting the sign argument to TRUE will give you signed ranks, where the ranking is done according to absolute size but where the sign is preserved (i.e., 2, 1, -3, 4).

Usage

```
ranktransform(x, ...)

## S3 method for class 'numeric'
ranktransform(x, sign = FALSE, method = "average", verbose = TRUE, ...)

## S3 method for class 'grouped_df'
```

```
ranktransform(
  x,
  select = NULL,
  exclude = NULL,
  sign = FALSE,
  method = "average",
  ...
)

## S3 method for class 'data.frame'
ranktransform(
  x,
  select = NULL,
  exclude = NULL,
  sign = FALSE,
  method = "average",
  ...
)
```

Arguments

x	Object.
...	Arguments passed to or from other methods.
sign	Logical, if TRUE, return signed ranks.
method	Treatment of ties. Can be one of "average" (default), "first", "last", "random", "max" or "min". See rank() for details.
verbose	Toggle warnings and messages on or off.
select	Character vector of column names. If NULL (the default), all variables will be selected.
exclude	Character vector of column names to be excluded from selection.

Value

A rank-transformed object.

See Also

Other transform utilities: data_rescale(), normalize(), standardize()

Examples

```
ranktransform(c(0, 1, 5, -5, -2))
ranktransform(c(0, 1, 5, -5, -2), sign = TRUE)

head(ranktransform(trees))
```

rescale_weights *Rescale design weights for multilevel analysis*

Description

Most functions to fit multilevel and mixed effects models only allow to specify frequency weights, but not design (i.e. sampling or probability) weights, which should be used when analyzing complex samples and survey data. `rescale_weights()` implements an algorithm proposed by *Asparouhov (2006)* and *Carle (2009)* to rescale design weights in survey data to account for the grouping structure of multilevel models, which then can be used for multilevel modelling.

Usage

```
rescale_weights(data, group, probability_weights, nest = FALSE)
```

Arguments

`data`	A data frame.
`group`	Variable names (as character vector, or as formula), indicating the grouping structure (strata) of the survey data (level-2-cluster variable). It is also possible to create weights for multiple group variables; in such cases, each created weighting variable will be suffixed by the name of the group variable.
`probability_weights`	Variable indicating the probability (design or sampling) weights of the survey data (level-1-weight).
`nest`	Logical, if `TRUE` and `group` indicates at least two group variables, then groups are "nested", i.e. groups are now a combination from each group level of the variables in `group`.

Details

Rescaling is based on two methods: For `pweights_a`, the sample weights `probability_weights` are adjusted by a factor that represents the proportion of group size divided by the sum of sampling weights within each group. The adjustment factor for `pweights_b` is the sum of sample weights within each group divided by the sum of squared sample weights within each group (see Carle (2009), Appendix B).

Regarding the choice between scaling methods A and B, Carle suggests that "analysts who wish to discuss point estimates should report results based on weighting method A. For analysts more interested in residual between-group variance, method B may generally provide the least biased estimates". In general, it is recommended to fit a non-weighted model and weighted models with both scaling methods and when comparing the models, see whether the "inferential decisions converge", to gain confidence in the results.

Though the bias of scaled weights decreases with increasing group size, method A is preferred when insufficient or low group size is a concern.

The group ID and probably PSU may be used as random effects (e.g. nested design, or group and PSU as varying intercepts), depending on the survey design that should be mimicked.

Value

data, including the new weighting variables: pweights_a and pweights_b, which represent the rescaled design weights to use in multilevel models (use these variables for the weights argument).

References

- Carle A.C. (2009). Fitting multilevel models in complex survey data with design weights: Recommendations. BMC Medical Research Methodology 9(49): 1-13
- Asparouhov T. (2006). General Multi-Level Modeling with Sampling Weights. Communications in Statistics - Theory and Methods 35: 439-460

Examples

```
if (require("lme4")) {
  data(nhanes_sample)
  head(rescale_weights(nhanes_sample, "SDMVSTRA", "WTINT2YR"))

  # also works with multiple group-variables
  head(rescale_weights(nhanes_sample, c("SDMVSTRA", "SDMVPSU"), "WTINT2YR"))

  # or nested structures.
  x <- rescale_weights(
    data = nhanes_sample,
    group = c("SDMVSTRA", "SDMVPSU"),
    probability_weights = "WTINT2YR",
    nest = TRUE
  )
  head(x)

  nhanes_sample <- rescale_weights(nhanes_sample, "SDMVSTRA", "WTINT2YR")

  glmer(
    total ~ factor(RIAGENDR) * (log(age) + factor(RIDRETH1)) + (1 | SDMVPSU),
    family = poisson(),
    data = nhanes_sample,
    weights = pweights_a
  )
}
```

---

reshape_ci *Reshape CI between wide/long formats*

Description

Reshape CI between wide/long formats.

Usage

```
reshape_ci(x, ci_type = "CI")
```

Arguments

x	A data frame containing columns named CI_low and CI_high (or similar, see ci_type).
ci_type	String indicating the "type" (i.e. prefix) of the interval columns. Per *easystats* convention, confidence or credible intervals are named CI_low and CI_high, and the related ci_type would be "CI". If column names for other intervals differ, ci_type can be used to indicate the name, e.g. ci_type = "SI" can be used for support intervals, where the column names in the data frame would be SI_low and SI_high.

Value

A dataframe with columns corresponding to confidence intervals reshaped either to wide or long format.

Examples

```
x <- data.frame(
  Parameter = c("Term 1", "Term 2", "Term 1", "Term 2"),
  CI = c(.8, .8, .9, .9),
  CI_low = c(.2, .3, .1, .15),
  CI_high = c(.5, .6, .8, .85),
  stringsAsFactors = FALSE
)

reshape_ci(x)
reshape_ci(reshape_ci(x))
```

---

skewness *Compute Skewness and (Excess) Kurtosis*

Description

Compute Skewness and (Excess) Kurtosis

Usage

```
skewness(x, na.rm = TRUE, type = "2", iterations = NULL, verbose = TRUE, ...)

kurtosis(x, na.rm = TRUE, type = "2", iterations = NULL, verbose = TRUE, ...)

## S3 method for class 'parameters_kurtosis'
print(x, digits = 3, test = FALSE, ...)

## S3 method for class 'parameters_skewness'
print(x, digits = 3, test = FALSE, ...)
```

```
## S3 method for class 'parameters_skewness'
summary(object, test = FALSE, ...)

## S3 method for class 'parameters_kurtosis'
summary(object, test = FALSE, ...)
```

Arguments

x	A numeric vector or data.frame.
na.rm	Remove missing values.
type	Type of algorithm for computing skewness. May be one of 1 (or "1", "I" or "classic"), 2 (or "2", "II" or "SPSS" or "SAS") or 3 (or "3", "III" or "Minitab"). See 'Details'.
iterations	The number of bootstrap replicates for computing standard errors. If NULL (default), parametric standard errors are computed.
verbose	Toggle warnings and messages.
...	Arguments passed to or from other methods.
digits	Number of decimal places.
test	Logical, if TRUE, tests if skewness or kurtosis is significantly different from zero.
object	An object returned by skewness() or kurtosis().

Details

Skewness: Symmetric distributions have a skewness around zero, while a negative skewness values indicates a "left-skewed" distribution, and a positive skewness values indicates a "right-skewed" distribution. Examples for the relationship of skewness and distributions are:

- Normal distribution (and other symmetric distribution) has a skewness of 0
- Half-normal distribution has a skewness just below 1
- Exponential distribution has a skewness of 2
- Lognormal distribution can have a skewness of any positive value, depending on its parameters

(*https://en.wikipedia.org/wiki/Skewness*)

Types of Skewness: skewness() supports three different methods for estimating skewness, as discussed in *Joanes and Gill (1988)*:

- Type "1" is the "classical" method, which is g1 = (sum((x -mean(x))^3) / n) / (sum((x -mean(x))^2) / n)^1.5
- Type "2" first calculates the type-1 skewness, then adjusts the result: G1 = g1 * sqrt(n * (n -1)) / (n -2). This is what SAS and SPSS usually return
- Type "3" first calculates the type-1 skewness, then adjusts the result: b1 = g1 * ((1 -1 / n))^1.5. This is what Minitab usually returns.

Kurtosis: The kurtosis is a measure of "tailedness" of a distribution. A distribution with a kurtosis values of about zero is called "mesokurtic". A kurtosis value larger than zero indicates a "leptokurtic" distribution with *fatter* tails. A kurtosis value below zero indicates a "platykurtic" distribution with *thinner* tails (*https://en.wikipedia.org/wiki/Kurtosis*).

Types of Kurtosis: `kurtosis()` supports three different methods for estimating kurtosis, as discussed in *Joanes and Gill (1988)*:

- Type "1" is the "classical" method, which is `g2 = n * sum((x -mean(x))^4) / (sum((x -mean(x))^2)^2) -3`.
- Type "2" first calculates the type-1 kurtosis, than adjusts the result: `G2 = ((n + 1) * g2 + 6) * (n -1)/((n -2) * (n -3))`. This is what SAS and SPSS usually return
- Type "3" first calculates the type-1 kurtosis, than adjusts the result: `b2 = (g2 + 3) * (1 -1 / n)^2 -3`. This is what Minitab usually returns.

Standard Errors: It is recommended to compute empirical (bootstrapped) standard errors (via the `iterations` argument) than relying on analytic standard errors (*Wright & Herrington, 2011*).

Value

Values of skewness or kurtosis.

References

- D. N. Joanes and C. A. Gill (1998). Comparing measures of sample skewness and kurtosis. The Statistician, 47, 183–189.
- Wright, D. B., & Herrington, J. A. (2011). Problematic standard errors and confidence intervals for skewness and kurtosis. Behavior research methods, 43(1), 8-17.

Examples

```
skewness(rnorm(1000))
kurtosis(rnorm(1000))
```

---

smoothness *Quantify the smoothness of a vector*

Description

Quantify the smoothness of a vector

Usage

```
smoothness(x, method = "cor", lag = 1, iterations = NULL, ...)
```

Arguments

`x`	Numeric vector (similar to a time series).
`method`	Can be "diff" (the standard deviation of the standardized differences) or "cor" (default, lag-one autocorrelation).
`lag`	An integer indicating which lag to use. If less than 1, will be interpreted as expressed in percentage of the length of the vector.
`iterations`	The number of bootstrap replicates for computing standard errors. If `NULL` (default), parametric standard errors are computed.
`...`	Arguments passed to or from other methods.

Value

Value of smoothness.

References

https://stats.stackexchange.com/questions/24607/how-to-measure-smoothness-of-a-time-series-in-r

Examples

```
x <- (-10:10)^3 + rnorm(21, 0, 100)
plot(x)
smoothness(x, method = "cor")
smoothness(x, method = "diff")
```

---

standardize *Standardization (Z-scoring)*

Description

Performs a standardization of data (z-scoring), i.e., centering and scaling, so that the data is expressed in terms of standard deviation (i.e., mean = 0, SD = 1) or Median Absolute Deviance (median = 0, MAD = 1). When applied to a statistical model, this function extracts the dataset, standardizes it, and refits the model with this standardized version of the dataset. The `normalize()` function can also be used to scale all numeric variables within the 0 - 1 range.

For model standardization, see `effectsize::standardize.default()`

Usage

```
standardize(
  x,
  robust = FALSE,
  two_sd = FALSE,
  weights = NULL,
  verbose = TRUE,
  ...
)

standardise(
  x,
  robust = FALSE,
  two_sd = FALSE,
  weights = NULL,
  verbose = TRUE,
  ...
)
```

```
## S3 method for class 'numeric'
standardize(
  x,
  robust = FALSE,
  two_sd = FALSE,
  weights = NULL,
  verbose = TRUE,
  reference = NULL,
  center = NULL,
  scale = NULL,
  ...
)

## S3 method for class 'data.frame'
standardize(
  x,
  robust = FALSE,
  two_sd = FALSE,
  weights = NULL,
  verbose = TRUE,
  reference = NULL,
  select = NULL,
  exclude = NULL,
  remove_na = c("none", "selected", "all"),
  force = FALSE,
  append = FALSE,
  center = NULL,
  scale = NULL,
  ...
)

unstandardize(
  x,
  center = NULL,
  scale = NULL,
  reference = NULL,
  robust = FALSE,
  two_sd = FALSE,
  ...
)

unstandardise(
  x,
  center = NULL,
  scale = NULL,
  reference = NULL,
  robust = FALSE,
  two_sd = FALSE,
```

```
  ...
)
```

Arguments

x	A data frame, a vector or a statistical model (for `unstandardize()` cannot be a model).
robust	Logical, if `TRUE`, centering is done by subtracting the median from the variables and dividing it by the median absolute deviation (MAD). If `FALSE`, variables are standardized by subtracting the mean and dividing it by the standard deviation (SD).
two_sd	If `TRUE`, the variables are scaled by two times the deviation (SD or MAD depending on `robust`). This method can be useful to obtain model coefficients of continuous parameters comparable to coefficients related to binary predictors, when applied to **the predictors** (not the outcome) (Gelman, 2008).
weights	Can be `NULL` (for no weighting), or: • For model: if `TRUE` (default), a weighted-standardization is carried out. • For `data.frames`: a numeric vector of weights, or a character of the name of a column in the `data.frame` that contains the weights. • For numeric vectors: a numeric vector of weights.
verbose	Toggle warnings and messages on or off.
...	Arguments passed to or from other methods.
reference	A data frame or variable from which the centrality and deviation will be computed instead of from the input variable. Useful for standardizing a subset or new data according to another data frame.
center, scale	• For `standardize()`: Numeric values, which can be used as alternative to `reference` to define a reference centrality and deviation. If `scale` and `center` are of length 1, they will be recycled to match the length of selected variables for standardization. Else, `center` and `scale` must be of same length as the number of selected variables. Values in `center` and `scale` will be matched to selected variables in the provided order, unless a named vector is given. In this case, names are matched against the names of the selected variables. • For `unstandardize()`: `center` and `scale` correspond to the center (the mean / median) and the scale (SD / MAD) of the original non-standardized data (for data frames, should be named, or have column order correspond to the numeric column). However, one can also directly provide the original data through `reference`, from which the center and the scale will be computed (according to `robust` and `two_sd`). Alternatively, if the input contains the attributes `center` and `scale` (as does the output of `standardize()`), it will take it from there if the rest of the arguments are absent.
select	Character vector of column names. If `NULL` (the default), all variables will be selected.
exclude	Character vector of column names to be excluded from selection.

remove_na — How should missing values (NA) be treated: if "none" (default): each column's standardization is done separately, ignoring NAs. Else, rows with NA in the columns selected with select / exclude ("selected") or in all columns ("all") are dropped before standardization, and the resulting data frame does not include these cases.

force — Logical, if TRUE, forces standardization of factors and dates as well. Factors are converted to numerical values, with the lowest level being the value 1 (unless the factor has numeric levels, which are converted to the corresponding numeric value).

append — Logical or string. If TRUE, standardized variables get new column names (with the suffix "_z") and are appended (column bind) to x, thus returning both the original and the standardized variables. If FALSE, original variables in x will be overwritten by their standardized versions. If a character value, standardized variables are appended with new column names (using the defined suffix) to the original data frame.

Value

The standardized object (either a standardize data frame or a statistical model fitted on standardized data).

Note

When x is a vector or a data frame with remove_na = "none"), missing values are preserved, so the return value has the same length / number of rows as the original input.

See Also

See center() for grand-mean centering of variables.

Other transform utilities: data_rescale(), normalize(), ranktransform()

Examples

```
d <- iris[1:4, ]

# vectors
standardise(d$Petal.Length)

# Data frames
# overwrite
standardise(d, select = c("Sepal.Length", "Sepal.Width"))

# append
standardise(d, select = c("Sepal.Length", "Sepal.Width"), append = TRUE)

# append, suffix
standardise(d, select = c("Sepal.Length", "Sepal.Width"), append = "_std")

# standardizing with reference center and scale
d <- data.frame(
```

```
  a = c(-2, -1, 0, 1, 2),
  b = c(3, 4, 5, 6, 7)
)

# default standardization, based on mean and sd of each variable
standardize(d) # means are 0 and 5, sd ~ 1.581139

# standardization, based on mean and sd set to the same values
standardize(d, center = c(0, 5), scale = c(1.581, 1.581))

# standardization, mean and sd for each variable newly defined
standardize(d, center = c(3, 4), scale = c(2, 4))

# standardization, taking same mean and sd for each variable
standardize(d, center = 1, scale = 3)
```

`to_numeric` *Convert to Numeric (if possible)*

Description

Tries to convert vector to numeric if possible (if no warnings or errors). Otherwise, leaves it as is.

Usage

```
to_numeric(x)
```

Arguments

`x`	A vector to be converted.

Value

Numeric vector (if possible)

Examples

```
to_numeric(c("1", "2"))
to_numeric(c("1", "2", "A"))
```

visualisation_recipe *Prepare objects for visualisation*

Description

This function prepares objects for visualisation by returning a list of layers with data and geoms that can be easily plotted using for instance `ggplot2`. See the documentation for your object's class:

- modelbased (`estimate_means`, `estimate_contrasts`, `estimate_slopes`, `estimate_predicted`, `estimate_grouplevel`)

Usage

```
visualisation_recipe(x, ...)
```

Arguments

`x`	An `easystats` object.
`...`	Other arguments passed to other functions.

winsorize *Winsorize data*

Description

Winsorize data

Usage

```
winsorize(data, ...)

## S3 method for class 'numeric'
winsorize(data, threshold = 0.2, verbose = TRUE, ...)
```

Arguments

`data`	Dataframe or vector.
`...`	Currently not used.
`threshold`	The amount of winsorization.
`verbose`	Toggle warnings.

Details

Winsorizing or winsorization is the transformation of statistics by limiting extreme values in the statistical data to reduce the effect of possibly spurious outliers. The distribution of many statistics can be heavily influenced by outliers. A typical strategy is to set all outliers (values beyond a certain threshold) to a specified percentile of the data; for example, a 90\ to the 5th percentile, and data above the 95th percentile set to the 95th percentile. Winsorized estimators are usually more robust to outliers than their more standard forms.

Value

A dataframe with winsorized columns or a winsorized vector.

Examples

```
winsorize(iris$Sepal.Length, threshold = 0.2)
winsorize(iris, threshold = 0.2)
```

Index