Package ‘sjmisc’

May 18, 2015

Type Package
Encoding UTF-8
Title Miscellaneous Data Management Tools
Version 1.0.2
Date 2015-05-16
Author Daniel Lüdecke <d.luedeke@uke.de>
Maintainer Daniel Lüdecke <d.luedeke@uke.de>
Description Collection of several utility functions for reading or writing data, recoding and la-
     belling variables and some frequently used statistical tests.
License GPL-3
Imports MASS
Suggests coin, foreign, haven, lme4, nlme, stringdist
URL https://github.com/sjPlot/sjmisc
BugReports https://github.com/sjPlot/sjmisc/issues
NeedsCompilation no
Repository CRAN
Date/Publication 2015-05-18 00:50:30

R topics documented:

  sjmisc-package ...................................................... 3
  chisq_gof .......................................................... 3
  cod ............................................................... 5
  cramer ............................................................. 6
  cronb ............................................................. 6
  cv ................................................................. 7
  dicho .............................................................. 8
  efc ............................................................... 9
  efc2 ............................................................. 10
  efc3 ............................................................. 10
topics documented:

- eta_sq .......................................................... 11
- get_val_labels .............................................. 12
- get_var_labels .............................................. 14
- group_labels ................................................ 15
- group_str .................................................... 17
- group_var .................................................... 18
- hoslem_gof .................................................. 19
- icc ............................................................ 20
- is_crossed .................................................... 22
- is_nested ..................................................... 23
- is_num_fac ................................................... 24
- levene_test .................................................. 25
- mean_n ......................................................... 25
- mic ............................................................. 26
- mwu ............................................................. 27
- phi .............................................................. 28
- pseudo_r2 ..................................................... 29
- read_sas ....................................................... 30
- read_spss ..................................................... 31
- read_stata .................................................... 32
- rec ............................................................. 33
- recode_to ..................................................... 34
- reliab_test ................................................... 36
- rmse ............................................................ 37
- set_na .......................................................... 39
- set_val_labels ............................................... 40
- set_var_labels ............................................... 41
- std_beta ....................................................... 43
- std_e ........................................................... 44
- str_pos ........................................................ 44
- table_values .................................................. 46
- to_fac ........................................................ 47
- to_label ....................................................... 48
- to_sjPlot ...................................................... 49
- to_value ....................................................... 50
- trim ............................................................ 50
- weight ........................................................ 51
- weight2 ....................................................... 52
- word_wrap ..................................................... 54
- write_spss .................................................... 54
- write_stata ................................................... 55

Index .......................................................... 56
sjmisc-package

*Miscellaneous Data Management Tools*

**Description**

Collection of several utility functions for reading or writing data, recoding and labelling variables and some frequently used statistical tests.

**Details**

- **Package**: sjmisc
- **Type**: Package
- **Version**: 1.0.2
- **Date**: 2015-05-16
- **License**: GPL-3

**Author(s)**

Daniel Lüdecke <d.luedecke@uke.de>

**References**

- **Documentation**: [http://www.strengejacke.de/sjPlot](http://www.strengejacke.de/sjPlot)
- **Github**: [https://github.com/sjPlot/sjmisc](https://github.com/sjPlot/sjmisc)
- **Bug reports**: [https://github.com/sjPlot/sjmisc/issues](https://github.com/sjPlot/sjmisc/issues)

**chisq_gof**

*Performs a Chi-square goodness-of-fit-test*

**Description**

This method performs a Chi-square goodness-of-fit-test (GOF) either on a numeric vector against probabilities, or a Goodness-of-fit tests for *glm* for binary data.

**Usage**

`chisq_gof(x, prob = NULL, weights = NULL)`
Arguments

- `x`: a numeric vector / variable, or a `glm`-object.
- `prob`: a vector of probabilities (indicating the population probabilities) of the same length as `x`'s amount of categories / factor levels. Use `nrow(table(x))` to determine the amount of necessary values for `prob`. Only used, when `x` is a vector, and not a `glm`-object.
- `weights`: a vector with weights, used to weight `x`.

Value

For vectors, (insensibly) returns the object of the computed `chisq.test`.

For `glm`-objects, an object of class `chisq.gof` with following values:

- `p.value` the p-value for the goodness-of-fit test
- `z.score` the standardized z-score for the goodness-of-fit test
- `RSS` the residual sums of squares term
- `X2` the pearson chi-squared statistic

Note

For vectors, this function is a convenient function for the `chisq.test`, performing goodness-of-fit test.

For `glm`-objects, this function performs a goodness-of-fit test based on the `X2GOFtest` function of the `binomTools` package. A well-fitting model shows no significant difference between the model and the observed data, i.e. the reported p-values should be greater than 0.05.

Examples

data(efc)
# differing from population
chisq_gof(efc$e42dep, c(0.3, 0.2, 0.22, 0.28))
# equal to population
chisq_gof(efc$e42dep, prop.table(table(efc$e42dep)))

# goodness-of-fit test for logistic regression
efc$services <- dicho(efc$tot_sc_e, "v", 0, asNum = TRUE)
fit <- glm(services ~ neg_c_7 + c161sex + e42dep, 
    data = efc, 
    family = binomial(link = "logit"))
chisq_gof(fit)
Description

This method calculates the Coefficient of Discrimination $d$ for \texttt{glm}s or \texttt{glmer}s for binary data. It is an alternative to other Pseudo-R-squared values like Nakelkerke’s R2 or Cox-Snell R2.

Usage

cod(x)

Arguments

x a fitted \texttt{glm} or \texttt{glmer}.

Value

The $d$ Coefficient of Discrimination, also known as Tjur’s R-squared value.

Note

The Coefficient of Discrimination $d$ can be read like any other (Pseudo-)R-squared value.

References


Examples

data(efc)

# Tjur's R-squared value
efc$services <- dicho(efc$tot_sc_e, "v", 0, asNum = TRUE)
fit <- glm(services ~ neg_c_7 + c161sex + e42dep,
  data = efc,
  family = binomial(link = "logit"))
cod(fit)
cramer  

Cramer’s V for a contingency table

Description

Compute Cramer’s V for a table with more than 2x2 fields.

Usage

cramer(tab)

Arguments

tab  
A simple table or ftable. Tables of class xtabs and other will be coerced to ftable objects.

Value

The table’s Cramer’s V.

See Also

phi

Examples

```r
   tab <- table(sample(1:2, 30, TRUE), sample(1:3, 30, TRUE))
cramer(tab)
```

cronb  

Calculates Cronbach’s Alpha for a matrix

Description

This function calculates the Cronbach’s alpha value for each column of a data frame or matrix.

Usage

cronb(df)

Arguments

df  
A data frame or matrix with more than 2 columns.

Value

The Cronbach’s alpha value for df.
Note

See examples from sjp.pca and sjt.pca.

See Also

reliab_test

---

### cv

#### Compute coefficient of variation

**Description**

Compute coefficient of variation for single variables (standard deviation divided by mean) or for fitted linear (mixed effects) models (root mean squared error (RMSE) divided by mean of dependent variable).

**Usage**

`cv(x)`

**Arguments**

- `x`: a (numeric) vector / variable or a fitted linear model of class `lm`, `merMod` (lme4) or `lme` (nlme).

**Details**

The advantage of the `cv` is that it is unitless. This allows coefficient of variation to be compared to each other in ways that other measures, like standard deviations or root mean squared residuals, cannot be (source: UCLA-FAQ).

**Value**

The coefficient of variation of `x`.

**References**

UCLA-FAQ: What is the coefficient of variation?

**See Also**

`rmse`
**Examples**
```
data(efc)
cv(efc$e17age)

fit <- lm(neg_c_7 ~ e42dep, data = efc)
cv(fit)

library(lme4)
fit <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
cv(fit)

library(nlme)
fit <- lme(distance ~ age, data = Orthodont) # random is ~ age
cv(fit)
```

**dicho**

**Dichotomize variables**

**Description**

Dichotomizes variables into dummy variables (0/1). Dichotomization is either done by median, mean or a specific value (see `dichby`). Either single vectors or complete data frames can be dichotomized.

**Usage**
```
dicho(x, dichBy = "median", dichVal = -1, asNum = FALSE)
```

**Arguments**
- `x` The variable (vector) or data frame that should be dichotomized.
- `dichBy` Indicates the split criterion where the variable is dichotomized.
  - By default, `var` is split into two groups at the median (`dichBy = "median"` or `dichBy = "md"`).
  - `dichBy = "mean"` (or `dichBy = "m"`) splits `var` into two groups at the mean of `var`.
  - `dichBy = "value"` (or `dichBy = "v"`) splits `var` into two groups at a specific value (see `dichVal`).
- `dichVal` numeric, indicates a value where `var` is dichotomized when `dichBy = "value"`. *Note that dichVal is inclusive*, i.e. `dichVal = 10` will split `var` into one group with values from lowest to 10 and another group with values greater than 10.
- `asNum` logical, if TRUE, return value will be numeric, not a factor.

**Value**

A dichotomized factor (or numeric, if `asNum = TRUE`) variable (0/1-coded), respectively a data frame of dichotomized factor (or numeric) variables.
Examples

data(efc)
summary(efc$e12hour)
table(dicho(efc$e12hour))
table(dicho(efc$e12hour, "mean"))
table(dicho(efc$e12hour, "value", 30))

# sample data frame, values from 1-4
head(efc[, 6:10])
# dichotomized values (1 to 2 = 0, 3 to 4 = 1)
head(dicho(efc[, 6:10], "v", 2))

Description

A SPSS sample data set, read with the read_spss function and "converted" with to_sjPlot.

Note

There are two further datasets, efc2 and efc3, which slightly differ in their structure.

efc2 was read with the read_spss function and also has attached variable label attributes to each variable.

efc3 categorical variables have been converted to labelled factors, i.e. value labels are set as factor levels. However, factors in efc3 no longer have variable label attributes.

References

http://www.uk.de/eurofamcare/

Examples

# Attach EFC-data
data(efc)

# Show structure
str(efc)

# show first rows
head(efc)

# show variables
## Not run:
library(sjPlot)
view_spss(efc)

# show variable labels
get_var_labels(efc)

# plot efc-data frame summary
sjt.df(efc, alternateRowColor=TRUE)
## End(Not run)

efc2

Sample dataset from the EUROFAMCARE project

Description
Sample dataset from the EUROFAMCARE project

References
http://www.uke.de/eurofamcare/

Examples
# Attach EFC-data
data(efc2)

## Not run:
# show variables
view_spss(efc2)

# print frq of dependency
sjt.frq(efc2$e42dep)
## End(Not run)

efc3

Sample dataset from the EUROFAMCARE project

Description
Sample dataset from the EUROFAMCARE project

References
http://www.uke.de/eurofamcare/
Examples

# Attach EFC-data
data(efc3)

str(efc3$e15relat)
table(efc3$e15relat)

## Not run:
# print frq of relationships
sji.frg(efc3$e15relat)
## End(Not run)

\section*{eta_sq}

\textit{Compute eta-squared of fitted anova}

\section*{Description}

Returns the eta-squared value for one-way-anovas.

\section*{Usage}

\begin{verbatim}
eta_sq(...)\end{verbatim}

\section*{Arguments}

\begin{verbatim}
... \end{verbatim} 

A fitted one-way-anova model or a dependent and grouping variable (see examples).

\section*{Value}

The eta-squared value.

\section*{Note}

Interpret eta-squared like r-squared or R-squared; a rule of thumb (Cohen):

- .02 ~ small
- .13 ~ medium
- .26 ~ large

\section*{References}

- How to compute eta-sq in ANOVA by hand?
- How to interpret and report eta squared?
- Wikipedia: Eta-squared
get_val_labels

Examples

# load sample data
data(efc)

# fit linear model
fit <- aov(c12hour ~ as.factor(e42dep), data = efc)

# print eta squared
eta_sq(fit)

# grouping variable will be converted to factor automatically
eta_sq(efc$c12hour, efc$e42dep)

get_val_labels Retrieve value labels of a variable or an imported data frame

Description

This function retrieves the value labels of an imported SPSS, SAS or STATA data set (via \texttt{read_spss}, \texttt{read_sas} or \texttt{read_stata}) and

- if \( x \) is a data frame, returns the all variable’s value labels as \texttt{list} object
- or, if \( x \) is a vector, returns the label as string.

Usage

get_val_labels(\( x \))

Arguments

\( x \)

a data frame with variables that have attached value labels (e.g. from an imported SPSS, SAS or STATA data set, via \texttt{read_spss}, \texttt{read_sas} or \texttt{read_stata}) or a variable (vector) with attached value labels.

Details

This package can add (and read) value and variable labels either in foreign package style (\texttt{value.labels} and \texttt{variable.label}) or in haven package style (\texttt{labels} and \texttt{label}). By default, the haven package style is used.

The \texttt{sjPlot} package accesses these attributes to automatically read label attributes for labelling axes categories and titles or table rows and columns.

When working with other packages, you can, e.g., use \texttt{get_var_labels} or \texttt{get_val_labels} to get a vector of value and variable labels, which can then be used with other functions like \texttt{barplot} etc. See 'Examples' from \texttt{get_val_labels}.

Furthermore, value and variable labels are used when saving data, e.g. to SPSS (see \texttt{write_spss}).
which means that the written SPSS file contains proper labels for each variable.

You can set a default label style (i.e. the names of the label attributes, see above) via `options(value_labels = "haven")` or `options(value_labels = "foreign")`.

**Value**

Either a list with all value labels from the data frame’s variables, a string with the value labels, if `x` is a variable, or `NULL` if no value label attribute was found.

**Note**

This function only works with vectors that have value and variable labels attached. This is automatically done by importing data sets with the `read_spss`, `read_sas` or `read_stata` function or labels can manually be added using the `set_val_labels` and `set_var_labels` functions.

With attached value and variable labels, most functions of the sjPlot package automatically detect labels and uses them as axis, legend or title labels in plots (sjp.-functions) respectively as column or row headers in table outputs (sjt.-functions). See online manual for more details.

Use `options(autoSetValueLabels = FALSE)` and `options(autoSetVariableLabels = FALSE)` to turn off automatic label detection.

**See Also**

- sjPlot manual: data initialization
- sjPlot manual: inspecting (SPSS imported) data frames
- `set_val_labels`

**Examples**

```r
# import SPSS data set
# mydat <- read.spss("my_spss_data.sav", enc="UTF-8")

# retrieve variable labels
# mydat.var <- get.var_labels(mydat)

# retrieve value labels
# mydat.val <- get.val_labels(mydat)

data(efc)
get_val_labels(efc$e42dep)

# simple barplot
barplot(table(efc$e42dep))
# get value labels to annotate barplot
barplot(table(efc$e42dep),
       names.arg = get_val_labels(efc$e42dep),
       main = get_var_labels(efc$e42dep))
```
**get_var_labels**

Retrieves variable labels of a data frame or a variable

---

**Description**

This function retrieves the value labels of an imported SPSS, SAS or STATA data set (via `read_spss`, `read_sas` or `read_stata`) and

- if `x` is a data frame, returns the all variable labels as names character vector of length `ncol(x)`.
- or, if `x` is a vector, returns the variable label as string.

**Usage**

`get_var_labels(x)`

**Arguments**

- `x` A data frame or a vector with "label" or "variable.label" attribute.

**Details**

See 'Details' in `get_val_labels`

**Value**

A named char vector with all variable labels from the data frame, or a simple char vector (of length 1) with the variable label, if `x` is a variable.

**Note**

See 'Note' in `get_val_labels`

**See Also**

- sjPlot manual: data initialization
- sjPlot manual: inspecting (SPSS imported) data frames
- `set_var_labels`

**Examples**

```r
# import SPSS data set
# mydat <- read_spss("my_spss_data.sav", enc="UTF-8")

# retrieve variable labels
# mydat.var <- get_var_labels(mydat)

# retrieve value labels
# mydat.val <- get_val_labels(mydat)
```
group_labels

Create labels for recoded groups

data(efc)

# get variable label
get_var_labels(efc$e42dep)

# alternative way
get_var_labels(efc)["e42dep"]

# simple barplot
barplot(table(efc$e42dep))
# get value labels to annotate barplot
barplot(table(efc$e42dep),
       names.arg = get_val_labels(efc$e42dep),
       main = get_var_labels(efc$e42dep))

---

**Description**

Creates the related labels for the grouped variable created by the `group_var` function.

**Usage**

```r
group_labels(var, groupsize = 5, rightInterval = FALSE, 
             autoGroupCount = 30)
```

**Arguments**

- `var`  
  The variable, which should be recoded into groups.

- `groupsize`  
  The group-size, i.e., the range for grouping. By default, for each 5 categories new group is built, i.e., `groupsize=5`. Use `groupsize="auto"` to automatically resize a variable into a maximum of 30 groups (which is the ggplot-default grouping when plotting histograms). Use parameter `autoGroupCount` to define the amount of groups.

- `rightInterval`  
  If TRUE, grouping starts with the lower bound of `groupsize`. In this case, groups cover the ranges from 50-54, 55-59, 60-64 etc. If FALSE (default), grouping starts with the upper bound of `groupsize`. In this case, groups cover the ranges from 51-55, 56-60, 61-65 etc.

- `autoGroupCount`  
  Sets the maximum number of groups that are built when auto-grouping is on (`groupsize="auto"`). Default is 30. If `groupsize` is not set to "auto", this parameter will be ignored.

**Details**

See 'Details' in `group_var`.
Value

A string vector containing labels based on the grouped counts of \texttt{var}, formatted as "from lower bound to upper bound", e.g. "10–19" "20–29" "30–39" etc. See example below.

Note

Usually you should use the same values for \texttt{groupsize} and \texttt{rightInterval} as used in the \texttt{group_var} function if you want to create labels for the related recoded variable.

See Also

• \texttt{group_var}

• \texttt{group_str}

Examples

```r
age <- abs(round(rnorm(100, 65, 20)))
age.grp <- group_var(age, 10)
hist(age)
hist(age.grp)

age.grpvar <- group_labels(age, 10)
table(age.grp)
print(age.grpvar)

# histogram with EUROFAMCARE sample dataset
# variable not grouped
data(efc)
## Not run:
library(sjPlot)
sjp.frq(efc$e17age, 
    title = get_var_labels(efc$e17age), 
    type = "h", 
    showValueLabels = FALSE)
## End(Not run)

# bar plot with EUROFAMCARE sample dataset
# grouped variable
data(efc)
ageGrp <- group_var(efc$e17age)
ageGrpLab <- group_labels(efc$e17age)
## Not run:
library(sjPlot)
sjp.frq(ageGrp, 
    title = get_var_labels(efc$e17age), 
    axisLabels.x = ageGrpLab)
## End(Not run)
```
group_str

Group near elements of string vectors

Description

This function groups elements of a string vector (character or string variable) according to the element’s distance ('similarity'). The more similar two string elements are, the higher is the chance to be combined into a group.

Usage

```r
group_str(strings, maxdist = 2, method = "lv", strict = FALSE,
trim.whitespace = TRUE, remove.empty = TRUE, showProgressBar = FALSE)
```

Arguments

- `strings`: a character vector with string elements
- `maxdist`: the maximum distance between two string elements, which is allowed to treat two elements as similar or equal.
- `method`: Method for distance calculation. The default is "lv". See `stringdist` package for details.
- `strict`: if TRUE, value matching is more strictly. See examples for details.
- `trim.whitespace`: if TRUE (default), leading and trailing white spaces will be removed from string values.
- `remove.empty`: if TRUE (default), empty string values will be removed from the character vector `strings`.
- `showProgressBar`: If TRUE, the progress bar is displayed when computing the distance matrix. Default in FALSE, hence the bar is hidden.

Value

A character vector where similar string elements (values) are recoded into a new, single value.

See Also

`str_pos`

Examples

```r
## Not run:
library(sjPlot)
oldstring <- c("Hello", "Helo", "Hole", "Apple",
              "Ape", "New", "Old", "System", "Systemic")
newstring <- group_str(oldstring)
```
Recode count variables into grouped factors

Description
Recode count variables into grouped factors.

Usage
```
group_var(var, groupsize = 5, asNumeric = TRUE, rightInterval = FALSE, autoGroupCount = 30)
```

Arguments
- **var**: The count variable, which should recoded into groups.
- **groupsize**: The group-size, i.e. the range for grouping. By default, for each 5 categories a new group is defined, i.e. groupsize=5. Use groupsize="auto" to automatically resize a variable into a maximum of 30 groups (which is the ggplot-default grouping when plotting histograms). Use autoGroupCount to determine the amount of groups.
- **asNumeric**: If TRUE (default), the recoded variable will be returned as numeric vector. If FALSE, a factor is returned.
- **rightInterval**: If TRUE, grouping starts with the lower bound of groupsize. In this case, groups cover the ranges from 50-54, 55-59, 60-64 etc.
  If FALSE (default), grouping starts with the upper bound of groupsize. In this case, groups cover the ranges from 51-55, 56-60, 61-65 etc.
- **autoGroupCount**: Sets the maximum number of groups that are defined when auto-grouping is on (groupsize="auto"). Default is 30. If groupsize is not set to "auto", this parameter will be ignored.

Details
If groupsize is set to a specific value, the variable is recoded into several groups, where each group has a maximum range of groupsize. Hence, the amount of groups differ depending on the range of var.

If groupsize = "auto", the variable is recoded into a maximum of autoGroupCount groups. Hence, independent from the range of var, always the same amount of groups are created, so the range within each group differs (depending on var’s range).
hoslem_gof

**Value**

A grouped variable, either as numeric or as factor (see parameter `asNumeric`).

**See Also**

- `group_labels`
- `group_str`

**Examples**

```r
age <- abs(round(rnorm(100, 65, 20)))
age.grp <- group_var(age, 10)
hist(age)
hist(age.grp)

# histogram with EUROFAMCARE sample dataset
# variable not grouped
data(efc)
## Not run:
library(sjPlot)
sjp.frq(efc$e17age,
    title = get_var_labels(efc$e17age),
    type = "h",
    showValueLabels = FALSE)
## End(Not run)

# bar plot with EUROFAMCARE sample dataset
# grouped variable
data(efc)
ageGrp <- group_var(efc$e17age)
ageGrpLab <- group_labels(efc$e17age)
## Not run:
library(sjPlot)
sjp.frq(ageGrp,
    title = get_var_labels(efc$e17age),
    axisLabels.x = ageGrpLab)
## End(Not run)
```

---

**hoslem_gof**

*Performs a Hosmer-Lemeshow Goodness-of-fit-test*

**Description**

This method performs a Hosmer-Lemeshow goodness-of-fit-test for `glm`s or `glmer`s for binary data.

**Usage**

```r
hoslem_gof(x, g = 10)
```
Arguments

\( x \) a fitted \texttt{glm} or \texttt{glmer}.

\( g \) number of bins to divide the data Default is 10.

Value

An object of class \texttt{hoslem_test} with following values:

- \texttt{chisq} the Hosmer-Lemeshow chi-squared statistic
- \texttt{df} degrees of freedom
- \texttt{p.value} the p-value for the goodness-of-fit test

Note

A well-fitting model shows no significant difference between the model and the observed data, i.e. the reported p-value should be greater than 0.05.

See Also

\texttt{pseudo_r2}

Examples

data(efc)

\begin{verbatim}
# goodness-of-fit test for logistic regression 
efc$services <- dichoto(efc$tot_sc_e, "v", 0, asNum = TRUE)
fit <- glm(services ~ neg_c_7 + c161sex + e42dep, 
data = efc, 
family = binomial(link = "logit")
hoslem_gof(fit)
\end{verbatim}

---

\textbf{icc} \hspace{1cm} \textit{Compute Intra-Class-Correlation}

Description

This function calculates the intraclass-correlation (icc) for random intercepts of mixed effects models. Currently, only \texttt{merMod} objects are supported.

Usage

\texttt{icc(x)}

Arguments

\( x \) a fitted mixed effects model (\texttt{merMod}-class).
Value

A numeric vector with all random intercept intraclass-correlation-coefficients.

Note

Why ICC is useful

1. It can help you determine whether or not a linear mixed model is even necessary. If you find
   that the correlation is zero, that means the observations within clusters are no more similar
   than observations from different clusters. Go ahead and use a simpler analysis technique.

2. It can be theoretically meaningful to understand how much of the overall variation in the
   response is explained simply by clustering. For example, in a repeated measures psychological
   study you can tell to what extent mood is a trait (varies among people, but not within a person
   on different occasions) or state (varies little on average among people, but varies a lot across
   occasions).

3. It can also be meaningful to see how the ICC (as well as the between and within cluster
   variances) changes as variable are added to the model.

(Grace-Martin K: The Intraclass Correlation Coefficient in Mixed Models, web)

The calculation of the ICC for generalized linear mixed models is based on Wu et al. (2012).

References

  correlation coefficient for binary responses in cancer prevention cluster randomized trials.
  Contempory Clinical Trials 33: 869-880 (doi:10.1016/j.cct.2012.05.004)
- CrossValidated (2012) Intraclass correlation (ICC) for an interaction?
- CrossValidated (2014) Interpreting the random effect in a mixed-effect model
- CrossValidated (2014) how to partition the variance explained at group level and individual
  level

Examples

```r
## Not run:
library(lme4)
fit <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
icc(fit)

sleepstudy$mygrp <- sample(1:45, size = 180, replace = T)
fit <- lmer(Reaction ~ Days + (1|mygrp) + (Days | Subject), sleepstudy)
icc(fit)
## End(Not run)
```
is_crossed  

Check whether two factors are crossed

Description

This function checks whether two factors are crossed, i.e. if each level of one factor occurs in combination with each level of the other factor.

Usage

is_crossed(f1, f2)

Arguments

f1  
a numeric vector or factor.
f2  
a numeric vector or factor.

Value

Logical, TRUE if factors are crossed, FALSE otherwise.

References

Grace, K. The Difference Between Crossed and Nested Factors. (web)

See Also

is_nested

Examples

# crossed factors, each category of  
# x appears in each category of y
x <- c(1,4,3,2,3,2,1,4)
y <- c(1,1,1,2,2,1,2,2)  
# show distribution
table(x, y)  
# check if crossed
is_crossed(x, y)

# not crossed factors
x <- c(1,4,3,2,3,2,1,4)
y <- c(1,1,1,2,1,1,2,2)  
# show distribution
table(x, y)  
# check if crossed
is_crossed(x, y)
is_nested

Check whether two factors are nested

Description

This function checks whether two factors are nested, i.e. if each category of the first factor co-occurs with only one category of the other.

Usage

is_nested(f1, f2)

Arguments

- **f1**: a numeric vector or factor.
- **f2**: a numeric vector or factor.

Value

Logical, TRUE if factors are nested, FALSE otherwise.

Note

If factors are nested, a message is displayed to tell whether f1 is nested within f2 or vice versa.

References

Grace, K. The Difference Between Crossed and Nested Factors. (web)

See Also

is_crossed

Examples

# nested factors, each category of
# x appears in one category of y
x <- c(1,2,3,4,5,6,7,8,9)
y <- c(1,1,1,2,2,2,3,3,3)
# show distribution
table(x, y)
# check if nested
is_nested(x, y)
is_nested(y, x)

# not nested factors
x <- c(1,2,3,4,5,6,7,8,9,1,2)
y <- c(1,1,1,2,2,2,3,3,3,2,3)
# show distribution
is_num_fac

Description

This function checks whether a factor has only numeric or any non-numeric factor levels.

Usage

is_num_fac(x)

Arguments

x a factor.

Value

Logical, TRUE if factor has numeric factor levels only, FALSE otherwise.

Examples

# numeric factor levels
f1 <- factor(c(NA, 1, 3, NA, 2, 4))
is_num_fac(f1)

# not completeley numeric factor levels
f2 <- factor(c(NA, "C", 1, 3, "A", NA, 2, 4))
is_num_fac(f2)

# not completeley numeric factor levels
f3 <- factor(c("Justus", "Bob", "Peter"))
is_num_fac(f3)
**levene_test**

Plot Levene-Test for One-Way-Anova

**Description**

Plot results of Levene’s Test for Equality of Variances for One-Way-Anova.

**Usage**

levene_test(depVar, grpVar)

**Arguments**

- **depVar**
  The dependent variable. Will be used with following formular: aov(depVar ~ grpVar)

- **grpVar**
  The grouping variable, as unordered factor. Will be used with following formular: aov(depVar ~ grpVar)

**Examples**

data(efc)
levene_test(efc$cl2hour, efc$e42dep)

---

**mean_n**

Compute row means with min amount of valid values

**Description**

This function is similar to the SPSS `mean.n` function and computes row means from a `data.frame` or `matrix` if at least `n` values of a row are valid (and not `NA`).

**Usage**

mean_n(dat, n, digits = 2)

**Arguments**

- **dat**
  A `data.frame` with at least two columns, where row means are applied.

- **n**
  May either be
  - a numeric value that indicates the amount of valid values per row to calculate the row mean;
  - or a value between 0 and 1, indicating a proportion of valid values per row to calculate the row mean (see details).

  If a row’s amount of valid values is less than `n`, `NA` will be returned as row mean value.

- **digits**
  Numeric value indicating the number of decimal places to be used for rounding mean value. Negative values are allowed (see ‘Details’).
Details

Rounding to a negative number of digits means rounding to a power of ten, so for example `mean_n(df, 3, digits = -2)` rounds to the nearest hundred.

For n, must be a numeric value from 0 to `ncol(dat)`. If a row in dat has at least n non-missing values, the row mean is returned. If n is a non-integer value from 0 to 1, n is considered to indicate the proportion of necessary non-missing values per row. E.g., if n = .75, a row must have at least `ncol(dat) * n` non-missing values for the row mean to be calculated. See examples.

Value

A vector with row mean values of df for those rows with at least n valid values. Else, \texttt{NA} is returned.

References

- candrea's blog
- r4stats.com

Examples

```r
dat <- data.frame(c1 = c(1,2,NA,4),
                  c2 = c(NA,2,NA,5),
                  c3 = c(NA,4,NA,NA),
                  c4 = c(2,3,7,8))

# needs at least 4 non-missing values per row
mean_n(dat, 4) # 1 valid return value

# needs at least 3 non-missing values per row
mean_n(dat, 3) # 2 valid return values

# needs at least 2 non-missing values per row
mean_n(dat, 2)

# needs at least 1 non-missing value per row
mean_n(dat, 1) # all means are shown

# needs at least 50% of non-missing values per row
mean_n(dat, .5) # 3 valid return values

# needs at least 75% of non-missing values per row
mean_n(dat, .75) # 2 valid return values
```

---

\textbf{mic} \hspace{1cm} \textit{Compute mean inter-item-correlation}
Description

This function calculates a mean inter-item-correlation, i.e. a correlation matrix of data will be computed (unless data is already a cor-object) and the mean of the sum of all item’s correlation values is returned. Requires either a data frame or a computed cor-object.

Usage

```r
mic(data, corMethod = "pearson")
```

Arguments

- `data`: A correlation object (see cor-function), or a data frame which correlations should be calculated.
- `corMethod`: Indicates the correlation computation method. May be one of "spearman" (default), "pearson" or "kendall". You may use initial letter only.

Value

The value of the computed mean inter-item-correlation.

Examples

```r
# Data from the EUROFAMCARE sample dataset
# create data frame with COPE-index scale
df <- data.frame(ecf[, c(start:end)])
mic(df)
```

---

**mwu**  
*Performs a Mann-Whitney-U-Test*

Description

This function performs a Mann-Whitney-U-Test (or wilcoxon rank sum test, see wilcox.test and wilcoxon.test) for the variable var, which is divided into groups indicated by grp (so the formula var ~ grp is used). If grp has more than two categories, a comparison between each two groups is performed.

The function reports U, p and Z-values as well as effect size r and group-rank-means.
Usage

```r
mwu(var, grp, distribution = "asymptotic", weights = NULL)
```

Arguments

- `var`: A numeric vector / variable, where the Mann-Whitney-U-Test should be applied to.
- `grp`: The grouping variable indicating the groups that should be used for comparison.
- `distribution`: Indicates how the null distribution of the test statistic should be computed. May be one of `exact`, `approximate` or `asymptotic` (default). See `wilcox_test` for details.
- `weights`: Defining integer valued weights for the observations. By default, this is NULL.

Value

(Invisibly) returns a data frame with U, p and Z-values for each group-comparison as well as effect-size r; additionally, group-labels and groups’ n’s are also included.

Note

This function calls the `wilcox_test` with formula. If `grp` has more than two groups, additionally a Kruskal-Wallis-Test (see `kruskal.test`) is performed.

Interpretation of effect sizes, as a rule-of-thumb:

- small effect >= 0.1
- medium effect >= 0.3
- large effect >= 0.5

Examples

```r
data(efc)
# Mann-Whitney-U-Tests for elder's age by elder's dependency.
mwu(efc$e17age, efc$e42dep)
```

---

### Description

Compute Phi value for a contingency table.

Usage

```r
phi(tab)
```
**Arguments**

`tab` A simple `table` or `ftable`. Tables of class `xtabs` and other will be coerced to `ftable` objects.

**Value**

The table’s Phi value.

**See Also**

cramer

**Examples**

```r
tab <- table(sample(1:2, 30, TRUE), sample(1:2, 30, TRUE))
phi(tab)
```

---

**Description**

This method calculates Nagelkerke’s and Cox-Snell’s pseudo-r-squared-values of `glm` for binary data.

**Usage**

`pseudo_r2(x)`

**Arguments**

`x` a fitted `glm`.

**Value**

An object of class `pseudo_r2` with following values:

- **CoxSnell** Cox-Snell’s pseudo-r-squared-value
- **Nagelkerke** Nagelkerke’s pseudo-r-squared-value

**See Also**

cod
Examples

```r
data(efc)

# Pseudo-R-squared values
efc$services <- dicho(efc$tot_sc_e, "v", 0, asNum = TRUE)
fit <- glm(services ~ neg_c_7 + c161sex + e42dep,
          data = efc,
          family = binomial(link = "logit"))
pseudo_r2(fit)
```

---

**read_sas**

*Import SAS dataset as data frame into R*

**Description**

Imports data from SAS (.sas7bdat), including NA's, value and variable labels.

**Usage**

```r
read_sas(path, path.cat = NULL, atomic.to.fac = FALSE)
```

**Arguments**

- `path` The file path to the SAS data file.
- `path.cat` optional, the file path to the SAS catalog file.
- `atomic.to.fac` Logical, if TRUE, factor variables imported from SAS (which are imported as *atomic*) will be converted to *factors*.

**Value**

A data frame containing the SAS data. Retrieve value labels with `get_val_labels` and variable labels with `get_var_labels`.

**Note**

This is a wrapper function for `read_sas` function of the haven package. This function converts the imported data into a sjPlot friendly format (see `to_sjPlot`).

**See Also**

`read_spss`
**Description**

Import data from SPSS, including NA's, value and variable labels.

**Usage**

```r
read_spss(path, enc = NA, autoAttachVarLabels = FALSE, atomic.to.fac = FALSE, option = "haven")
```

**Arguments**

- `path` The file path to the SPSS dataset.
- `enc` The file encoding of the SPSS dataset. *Not needed if option = "haven" (default).*
- `autoAttachVarLabels` If TRUE, variable labels will automatically be attached to each variable as "variable.label" attribute. Use this parameter if option = "foreign", where variable labels are attached as list-attribute to the imported data frame. *Not needed if option = "haven" (default).*
- `atomic.to.fac` Logical, if TRUE, factor variables imported from SPSS (which are imported as atomic) will be converted to factors.
- `option` String, indicating which package will be used to read the SPSS data file. By default, option = "haven", which means, the read_spss function from the haven package is used. Use option = "foreign" to use foreign's `read_spss` function. Use `options(read_spss = "foreign")` to make this function always use the foreign-package `read_spss` function.

**Value**

A data frame containing the SPSS data. Retrieve value labels with `get_val_labels` and variable labels with `get_var_labels`.

**Note**

This is a wrapper function for `read_spss` of the haven package and `read.spss` of the foreign package. This function adds value and variable labels as attributes to the imported variables of the data frame.

With attached value and variable labels, most functions of the sjPlot package automatically detect labels and uses them as axis, legend or title labels in plots (sjp.-functions) respectively as column or row headers in table outputs (sjt.-functions). See online manual for more details.

When working with other packages, you can, e.g., use `get_var_labels` or `get_val_labels` to
get a vector of value and variable labels, which can then be used with other functions like `barplot` etc. See 'Examples' from `get_val_labels`.

**See Also**
- sjPlot manual: data initialization
- sjPlot manual: inspecting (SPSS imported) data frames
- `write_spss`

**Examples**

```r
## Not run:
# import SPSS data set. uses haven's read function
# by default
mydat <- read.spss("my.spss.data.sav")

# use foreign's read function
mydat <- read.spss("my.spss.data.sav",
  enc = "UTF-8",
  option = "foreign")

# use haven's read function, convert atomic to factor
mydat <- read.spss("my.spss.data.sav", atomic.to.fac = TRUE)

# retrieve variable labels
mydat.var <- get.var.labels(mydat)

# retrieve value labels
mydat.val <- get.val.labels(mydat)
## End(Not run)
```

---

**read_stata**

*Import STATA dataset as data frame into R*

**Description**

Imports data from STATA dta-files, including NA’s, value and variable labels.

**Usage**

```r
read_stata(path, atomic.to.fac = FALSE)
```

**Arguments**

- `path` The file path to the STATA data file.
- `atomic.to.fac` Logical, if TRUE, factor variables imported from STATA (which are imported as atomic) will be converted to factors.
Value

A data frame containing the STATA data. Retrieve value labels with `get_val_labels` and variable labels with `get_var_labels`.

Note

This is a wrapper function for `read_dta` function of the haven package. This function converts the imported data into a sjPlot friendly format (see `to_sjPlot`).

See Also

`read_spss`

---

**rec**

*Recode numeric variables*

**Description**

Recodes the categories of a (numeric) variable `x` into new category values.

**Usage**

```r
rec(x, recodes)
```

**Arguments**

- `x` a numeric variable (vector) or a `factor` with numeric levels that should be recoded; or a data frame with such vectors.
- `recodes` a string with recode pairs of old and new values. See details for examples.

**Details**

The `recodes` string has following syntax:

- each recode pair has to be separated by a ;, e.g. `recodes = "1=1; 2=4; 3=2; 4=3"`
- multiple old values that should be recoded into a new single value may be separated with comma, e.g. "1,2=1; 3,4=2"
- a value range is indicated by a colon, e.g. "1:4=1; 5:8=2" (recodes all values from 1 to 4 into 1, and from 5 to 8 into 2)
- minimum and maximum values are indicates by `min` and `max`, e.g. "min:4=1; max:5=2" (recodes all values from minimum values of `x` to 4 into 1, and from 5 to maximum values of `x` into 2)
- all other values except specified are indicated by `else`, e.g. "3=1; 1=2; else=3" (recodes 3 into 1, 1 into 2 and all other values into 3)
- *NA* values are allowed both as old and new value, e.g. "NA=1; 3:5=NA" (recodes all NA from old value into 1, and all old values from 3 to 5 into NA in the new variable)
- "rev" is a special token that reverses the value order (see examples)
Value

A numeric variable with recoded category values, or a data frame with recoded categories for all variables.

Note

Please note following behaviours of the function:

• Non-matching values will be set to NA.
• Variable label attributes (see, for instance, get_var_labels) are retained, however, value label attributes are removed.
• If x is a data frame, all variables of the data frame should have the same categories resp. value range (else, see first bullet, NAs are produced).

See Also

set_na for setting NA values and recode_to for re-shifting value ranges.

Examples

data(efe)
table(efe$e42dep, exclude = NULL)

# replace NA with 5
table(rec(efe$e42dep, "1=1;2=2;3=3;4=4;NA=5"), exclude = NULL)

# recode 1 to 2 into 1 and 3 to 4 into 2
table(rec(efe$e42dep, "1,2=1;3,4=2"), exclude = NULL)

# recode 1 to 3 into 4 into 2
table(rec(efe$e42dep, "min:3=1;4=2"), exclude = NULL)

# recode 2 to 1 and all others into 2
table(rec(efe$e42dep, "2=1;else=2"), exclude = NULL)

# reverse value order
table(rec(efe$e42dep, "rev"), exclude = NULL)

# recode variables with same categorie in a data frame
head(efe[, 6:9])
head(rec(efe[, 6:9], "1=10;2=20;3=30;4=40"))
Usage

recode_to(x, lowest = 0, highest = -1)

Arguments

- **x**: A variable (vector) or a data frame that should be recoded.
- **lowest**: Indicating the lowest category value after recoding. Default is 0, so the new variable starts with the category value 0.
- **highest**: If specified and larger than `lowest`, all category values larger than `highest` will be set to NA. Default is -1, i.e. this parameter is ignored and no NA's will be produced.

Value

A new variable with recoded category values, where `lowest` indicates the lowest value; or a data frame where variables have been recoded as described.

Note

Value and variable label attributes (see, for instance, `get_val_labels` or `set_val_labels`) are retained.

See Also

`rec` for general recoding of variables and `set_na` for setting NA values.

Examples

```r
# recode 1-4 to 0-3
dummy <- sample(1:4, 10, replace = TRUE)
recode_to(dummy)

# recode 3-6 to 0-3
# note that numeric type is returned
dummy <- as.factor(3:6)
recode_to(dummy)

# lowest value starting with 1
dummy <- sample(11:15, 10, replace = TRUE)
recode_to(dummy, 1)

# lowest value starting with 1, highest with 3
# all others set to NA
dummy <- sample(11:15, 10, replace = TRUE)
recode_to(dummy, 1, 3)
```
reliab_test

Performs a reliability test on an item scale

Description
This function calculates the item discriminations (corrected item-total correlations for each item of x with the remaining items) and the Cronbach’s alpha for each item, if it was deleted from the scale.

Usage
reliab_test(x, scaleItems = FALSE, digits = 3)

Arguments
x A data frame with items (from a scale)
scaleItems If TRUE, the data frame’s vectors will be scaled. Recommended, when the variables have different measures / scales.
digits Amount of digits for Cronbach’s Alpha and correlation values in returned data frame.

Value
A data frame with the corrected item-total correlations (item discrimination) and Cronbach’s alpha (if item deleted) for each item of the scale, or NULL if data frame had too less columns.

Note
This function is similar to a basic reliability test in SPSS. The correlations in the Item-Total-Statistic are a computed correlation of each item against the sum of the remaining items (which are thus treated as one item).

See Also
cronb

Examples
# --------------------------------------------------------
# Data from the EUROFAMCARE sample dataset
# --------------------------------------------------------
data(efc)

# retrieve variable and value labels
dv <- get_var_labels(efc)

# receive first item of COPE-index scale
start <- which(colnames(efc) == "c82cop1")
# receive last item of COPE-index scale
end <- which(colnames(efc) == "c9@cop9")

# create data frame with COPE-index scale
x <- data.frame(efc[, c(start:end)])
colnames(x) <- varlabs[c(start:end)]

## Not run:
library(sjPlot)
sjt.df(reliab_test(x),
       describe = FALSE,
       showCommentRow = TRUE,
       commentString = sprintf("Cronbach's \&alpha;=\%.2f",
                               crnb(x)))

## End(Not run)

# ----------------------------------------
# Compute PCA on Cope-Index, and perform a
# reliability check on each extracted factor.
# ----------------------------------------
## Not run:
factors <- sjt.pca(x)$factor.index
findex <- sort(unique(factors))
library(sjPlot)
for (i in 1:length(findex)) {
  rel.df <- subset(x, select = which(factors == findex[i]))
  if (ncol(rel.df) >= 3) {
    sjt.df(reliab_test(rel.df),
           describe = FALSE,
           showCommentRow = TRUE,
           useViewer = FALSE,
           title = "Item-Total-Statistic",
           commentString = sprintf("Scale's overall Cronbach's \&alpha;=\%.2f",
                                    crnb(rel.df)))
  }
}
## End(Not run)

---

**rmse**

*Compute root mean squared error (RMSE)*

**Description**

Compute root mean squared error of fitted linear (mixed effects) models.

**Usage**

```r
rmse(fit, normalized = FALSE)
```
Arguments

- `fit`: a fitted linear model of class `lm`, `merMod` (lme4) or `lme` (nlme).
- `normalized`: logical, use TRUE if normalized rmse should be returned.

Value

The root mean squared error of `fit`; or the normalized root mean squared error of `fit` if `normalized = TRUE`.

Note

The RMSE is the square root of the variance of the residuals and indicates the absolute fit of the model to the data (difference between observed data to model’s predicted values). "RMSE can be interpreted as the standard deviation of the unexplained variance, and has the useful property of being in the same units as the response variable. Lower values of RMSE indicate better fit. RMSE is a good measure of how accurately the model predicts the response, and is the most important criterion for fit if the main purpose of the model is prediction." (Grace-Martin K: Assessing the Fit of Regression Models).

The normalized RMSE is the proportion of the RMSE related to the range of the response variable. Hence, lower values indicate less residual variance.

References

- Wikipedia: RMSD
- Grace-Martin K: Assessing the Fit of Regression Models

See Also

cv

Examples

data(efc)
fit <- lm(barthtot ~ c160age + c12hour, data = efc)
rmse(fit)

library(lme4)
fit <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
rmse(fit)

# normalized RMSE
library(nlme)
fit <- lme(distance ~ age, data = Orthodont) # random is ~ age
rmse(fit, normalized = TRUE)
set_na

Set NA for specific variable values

Description
This function sets specific values of a variable or data frame as missings (NA).

Usage
set_na(x, values)

Arguments
x a variable (vector) or a data frame where new missing values should be defined. If x is a data frame, each column is assumed to be a new variable, where missings should be defined.
values a numeric vector with values that should be replaced with NA’s. Thus, for each variable in x, values are replaced by NA’s.

Value
The variable or data frame x, where each value of values is replaced by an NA.

Note
Value and variable label attributes (see, for instance, get_val_labels or set_val_labels) are retained.

See Also
rec for general recoding of variables and recode_to for re-shifting value ranges.

Examples
# create random variable
dummy <- sample(1:8, 100, replace = TRUE)
# show value distribution
table(dummy)
# set value 1 and 8 as missings
dummy <- set_na(dummy, c(1, 8))
# show value distribution, including missings
table(dummy, exclude = NULL)

# create sample data frame
dummy <- data.frame(var1 = sample(1:8, 100, replace = TRUE),
                    var2 = sample(1:10, 100, replace = TRUE),
                    var3 = sample(1:6, 100, replace = TRUE))
# show head of data frame
head(dummy)
# set value 2 and 4 as missings
dummy <- set_na(dummy, c(2, 4))
# show head of new data frame
head(dummy)

set_val_labels

Attach value labels to a variable or vector

Description

This function attaches character labels as "value.labels" attribute to a variable or vector "x", resp. to all variables of a data frame if "x" is a data.frame. These value labels will be accessed by functions of the sjPlot package, in order to automatically set values or legend labels.

Usage

set_val_labels(x, labels)

Arguments

x  
a variable (vector) or a data frame where labels should be attached. Replaces former value labels.

labels  
A character vector of labels that will be attached to x by setting the "labels" or "value.labels" attribute. The length of this character vector must equal the value range of x, i.e. if x has values from 1 to 3, labels should have a length of 3. If x is a data frame, labels may also be a list of character vectors. If labels is a list, it must have the same length as number of columns of x. If labels is a vector and x is a data frame, the labels will be applied to each column of x. Use labels = "" to remove labels-attribute from x.

Details

See 'Details' in get_val_labels

Value

x with attached value labels; or with removed label-attribute if labels = "".

Note

See 'Note' in get_val_labels

See Also

- sjPlot manual: data initialization
- sjPlot manual: inspecting (SPSS imported) data frames
- get_val_labels
Examples

```r
## Not run:
library(sjPlot)
dummy <- sample(1:4, 40, replace=TRUE)
sjp.frq(dummy)

dummy <- set_val_labels(dummy, c("very low", "low", "mid", "hi"))
sjp.frq(dummy)
## End(Not run)
```

Description

This function sets variable labels to a single variable or to a set of variables in a data frame. To each variable, the attribute "label" or "variable.label" with the related variable name is attached. Most functions of the sjPlot package can automatically retrieve the variable name to use it as axis labels or plot title (see details).

Usage

```r
set_var_labels(x, lab, attr.string = NULL)
```

Arguments

- `x`: A single variable (vector) or data frame with variables.
- `lab`: If `x` is a vector (single variable), use a single character string with the variable label for `x`. If `x` is a `data.frame`, use a vector with character labels of same length as `ncol(x)`. Use `lab = ""` to remove labels-attribute from `x`, resp. set any value of vector `lab` to "" to remove specific variable label attributes from a data frame's variable.
- `attr.string`: The attribute string for the variable label. To ensure compatibility to the foreign-package, use the default string "variable.label". If you want to save data with the haven package, use `attr.string = "label"`. There is a wrapper function `write_spss` to save SPSS files, so you don’t need to take care of this.

Details

See 'Details' in `get_val_labels`

Value

`x`, with attached variable label attribute(s), which contains the variable name(s); or with removed label-attribute if `lab = ""`. 

---

**set_var_labels**

*Attach variable label(s) to a single variable or data frame*

**Description**

This function sets variable labels to a single variable or to a set of variables in a data frame. To each variable, the attribute "label" or "variable.label" with the related variable name is attached. Most functions of the sjPlot package can automatically retrieve the variable name to use it as axis labels or plot title (see details).

**Usage**

```r
set_var_labels(x, lab, attr.string = NULL)
```

**Arguments**

- `x`: A single variable (vector) or data frame with variables.
- `lab`: If `x` is a vector (single variable), use a single character string with the variable label for `x`. If `x` is a `data.frame`, use a vector with character labels of same length as `ncol(x)`. Use `lab = ""` to remove labels-attribute from `x`, resp. set any value of vector `lab` to "" to remove specific variable label attributes from a data frame's variable.
- `attr.string`: The attribute string for the variable label. To ensure compatibility to the foreign-package, use the default string "variable.label". If you want to save data with the haven package, use `attr.string = "label"`. There is a wrapper function `write_spss` to save SPSS files, so you don’t need to take care of this.

**Details**

See 'Details' in `get_val_labels`

**Value**

`x`, with attached variable label attribute(s), which contains the variable name(s); or with removed label-attribute if `lab = ""`. 

---

**Examples**

```r
## Not run:
library(sjPlot)
dummy <- sample(1:4, 40, replace=TRUE)
sjp.frq(dummy)

dummy <- set_val_labels(dummy, c("very low", "low", "mid", "hi"))
sjp.frq(dummy)
## End(Not run)
```
Note

See 'Note' in `get_val_labels`

See Also

- sjPlot manual: data initialization
- sjPlot manual: inspecting (SPSS imported) data frames
- `get_var_labels`

Examples

```r
# sample data set, imported from SPSS.
data(efc)

## Not run:
library(sjPlot)
sjt.frq(efc$e42dep)
sjt.frq(data.frame(efc$e42dep, efc$e16sex))
## End(Not run)

# manually set value and variable labels
# ----------------------------------------
dummy <- sample(1:4, 40, replace=TRUE)
dummy <- set_val_labels(dummy, c("very low", "low", "mid", "hi"))
dummy <- set_var_labels(dummy, "Dummy-variable")
# auto-detection of value labels by default, auto-detection of
# variable labels if parameter "title" set to NULL.
## Not run:
library(sjPlot)
sjp.frq(dummy, title = NULL)
## End(Not run)

# ----------------------------------------
# Set variable labels for data frame
# ----------------------------------------
dummy <- data.frame(a = sample(1:4, 10, replace = TRUE),
                   b = sample(1:4, 10, replace = TRUE),
                   c = sample(1:4, 10, replace = TRUE))
dummy <- set_var_labels(dummy,
                        c("Variable A",
                           "Variable B",
                           "Variable C"))
str(dummy)

# remove one variable label
dummy <- set_var_labels(dummy,
                        c("Variable A",
                           "",
                           "Variable C"))
str(dummy)
```
std_beta

Compiles standardized beta coefficients and confidence intervals of linear (mixed) models.

Description

Returns standardized beta coefficients and confidence intervals of a fitted linear (mixed) model, i.e. fit must either be of class lm or merMod.

Usage

std_beta(fit, include.ci = FALSE)

Arguments

- fit: A fitted linear (mixed) model of class lm or merMod (lme4-package).
- include.ci: logical, if TRUE, a data frame with confidence intervals will be returned, when fit is of class lm. If fit is a lmerMod object (lme4-package), always returns standard error instead of confidence intervals (hence, this parameter will be ignored when fit is a lmerMod object).

Value

A vector with standardized beta coefficients of the fitted linear model, or a data frame with standardized confidence intervals, if include.ci = TRUE.

Note

"Standardized coefficients refer to how many standard deviations a dependent variable will change, per standard deviation increase in the predictor variable. Standardization of the coefficient is usually done to answer the question of which of the independent variables have a greater effect on the dependent variable in a multiple regression analysis, when the variables are measured in different units of measurement (for example, income measured in dollars and family size measured in number of individuals)." (Source: Wikipedia)

References

Wikipedia: Standardized coefficient

Examples

# fit linear model
fit <- lm(airquality$Ozone ~ airquality$Wind + airquality$Temp + airquality$Solar.R)
# print std. beta coefficients
std_beta(fit)

# print std. beta coefficients and ci
std_beta(fit, include.ci = TRUE)
**std_e**  
*Compute standard error for variables*

**Description**  
Compute standard error for variable or for all variables of a data frame.

**Usage**  
std_e(x)

**Arguments**  

x  
a (numeric) vector / variable or a data frame.

**Value**  
The standard error of variable x, or for each variable if x is a data frame.

**Examples**  
std_e(rnorm(n = 100, mean = 3))

data(efc)
std_e(efc[, 1:3])

**str_pos**  
*Find partial matching and close distance elements in strings*

**Description**  
This function finds the element indices of partial matching or similar strings in a character vector. Can be used to find exact or slightly mistyped elements in a string vector.

**Usage**  
str_pos(searchString, findTerm, maxdist = 2, part.dist.match = 0, showProgressBar = FALSE)
Arguments

- searchString: a character vector with string elements
- findTerm: the string that should be matched against the elements of searchString.
- maxdist: the maximum distance between two string elements, which is allowed to treat them as similar or equal.
- part.dist.match: activates similar matching (close distance strings) for parts (substrings) of the searchString. Following values are accepted:
  - 0 for no partial distance matching
  - 1 for one-step matching, which means, only substrings of same length as findTerm are extracted from searchString matching
  - 2 for two-step matching, which means, substrings of same length as findTerm as well as strings with a slightly wider range are extracted from searchString matching

Default value is 0. See 'Details' for more information.

- showProgressBar: If TRUE, the progress bar is displayed when computing the distance matrix. Default in FALSE, hence the bar is hidden.

Details

For part.dist.match = 1, a substring of length(findTerm) is extracted from searchString, starting at position 0 in searchString until the end of searchString is reached. Each substring is matched against findTerm, and results with a maximum distance of maxdist are considered as "matching". If part.dist.match = 2, the range of the extracted substring is increased by 2, i.e. the extracted substring is two chars longer.

Value

A numeric vector with index position of elements in searchString that partially match or are similar to findTerm. Returns -1 if no match was found.

Note

This function does not return the position of a matching string inside another string, but the element's index of the searchString vector, where a (partial) match with findTerm was found. Thus, searching for "abc" in a string "this is abc" will not return 9 (the start position of the substring), but 1 (the element index, which is always 1 if searchString only has one element).

See Also

group_str
Examples

```r
## Not run:
str_pos(string, "hel")  # partial match
str_pos(string, "stem")  # partial match
str_pos(string, "R")  # no match
str_pos(string, "saste")  # similarity to "System"

# finds two indices, because partial matching now
# also applies to "Systemic"
str_pos(string,
   "sytsme",
   part.dist.match = 1)

# finds nothing
str_pos("We are Sex Pistols!", "postils")
str_pos("We are Sex Pistols!", "postils", part.dist.match = 1)
## End(Not run)
```

---

### table_values

**Compute expected and relative table values**

**Description**

This function calculates a table’s cell, row and column percentages as well as expected values and returns all results as lists of tables.

**Usage**

```r
table_values(tab, digits = 2)
```

**Arguments**

- `tab`: A simple `table` or `ftable` of which cell, row and column percentages as well as expected values are calculated. Tables of class `xtabs` and other will be coerced to `ftable` objects.
- `digits`: The amount of digits for the table percentage values.

**Value**

(invisibly) returns a list with four tables:

1. `cell` a table with cell percentages of `tab`
2. `row` a table with row percentages of `tab`
3. `col` a table with column percentages of `tab`
4. `expected` a table with expected values of `tab`
to_fac

Examples

```r
tab <- table(sample(1:2, 30, TRUE), sample(1:3, 30, TRUE))
# show expected values
table_values(tab)$expected
# show cell percentages
table_values(tab)$cell
```

---

to_fac  

*Convert variable into factor and keep value labels*

Description

This function converts a variable into a factor, but keeps variable and value labels, if these are attached as attributes to the variable. See examples.

Usage

```r
to_fac(x)
```

Arguments

- `x`: A (numeric or atomic) variable or a data frame with (numeric or atomic) variables.

Value

A factor variable, including variable and value labels, respectively a data frame with factor variables (including variable and value labels) if `x` was a data frame.

Note

This function is intended for use with vectors that have value and variable labels attached. Unlike `as.factor`, `to_fac` converts a variable into a factor and retains the value and variable label attributes.

Attaching labels is automatically done by importing data sets with one of the `read_*`-functions, like `read_spss`. Else, value and variable labels can be manually added to vectors with `set_val_labels` and `set_var_labels`.

See Also

- `to_value` to convert a factor into a numeric value and `to_label` to convert a value into a factor with labelled factor levels.
Examples

```r
## Not run:
data(efc)
library(sjPlot)
# normal factor conversion, loses value attributes
efc$e42dep <- as.factor(efc$e42dep)
sjt.frq(efc$e42dep)

data(efc)
# factor conversion, which keeps value attributes
efc$e42dep <- to_fac(efc$e42dep)
sjt.frq(efc$e42dep)
## End(Not run)
```

---

### to_label

**Converts variable into factor and replaces values with associated value labels**

#### Description

This function converts (replaces) variable values (also of factors) with their associated value labels. Might be helpful for factor variables. For instance, if you have a Gender variable with 0/1 value, and associated labels are male/female, this function would convert all 0 to male and all 1 to female and returns the new variable as `factor`.

#### Usage

```r
to_label(x)
```

#### Arguments

- **x**
  A variable of type `numeric, atomic, factor` or `labelled` (see haven package) with associated value labels (see `set_val_labels`), respectively a data frame with such variables.

#### Value

A factor variable with the associated value labels as factor levels, or a data frame with such factor variables (if `x` was a data frame).

#### Note

Value and variable label attributes (see, for instance, `get_val_labels` or `set_val_labels`) will be removed when converting variables to factors.

#### See Also

- `to_fac` to convert a numeric variable into a factor (and retain labels) and `to_value` to convert a factor into a numeric variable.
to_sjPlot

Convert a haven-imported data frame to sjPlot format

Description

This function converts a data frame, which was imported with any of haven’s read functions and contains labelled class vectors or a single vector of type labelled into an sjPlot friendly data frame format, which means that simply all labelled class attributes will be removed, so all vectors / variables will most likely become atomic. Additionally, tbl_df and tbl class attributes will be removed from data frames. See ’Note’.

Usage

to_sjPlot(x)

Arguments

x a data frame, which contains labelled class vectors or a single vector of class labelled.

Value

a data frame or single vector (depending on x) with ’sjPlot’ friendly object classes.
Note

This function is currently only used to avoid possible compatibility issues with labelled class vectors and tbl_df resp. tbl class attributes for data frames. Some known issues with labelled class vectors have already been fixed, so it might be that this function will become redundant in the future. Currently, data frames with tbl_df and tbl class attributes may cause difficulties when indexing columns like `data.frame[, colnr]` - only `data.frame[[colnr]]` seems to be safe when accessing data frame columns from within function calls.

See Also

sjPlot manual: data initialization

to_value

Converts factors to numeric variables

Description

This function converts (replaces) factor values with the related factor level index number, thus the factor is converted to a numeric variable.

Usage

to_value(x, startAt = NULL, keep.labels = TRUE)

Arguments

- **x**: A (factor) variable or a data frame with (factor) variables.
- **startAt**: the starting index, i.e. the lowest numeric value of the variable’s value range. By default, this parameter is NULL, hence the lowest value of the returned numeric variable corresponds to the lowest factor level (if factor is numeric) or to 1 (if factor levels are not numeric).
- **keep.labels**: logical, if TRUE, former factor levels will be attached as value labels. See `set_val_labels` for more details.

Value

A numeric variable with values ranging either from startAt to startAt + length of factor levels, or to the corresponding factor levels (if these were numeric). Or a data frame with numeric variables, if x was a data frame.

See Also

to_label to convert a value into a factor with labelled factor levels and to_fac to convert a numeric variable into a factor (and retain labels)
Examples

data(efc)
test <- to_label(efc$e42dep)
table(test)

table(to_value(test))
hist(to_value(test, 0))

# set lowest value of new variable
# to "5".
table(to_value(test, 5))

# numeric factor keeps values
dummy <- factor(c("3", "4", "6"))
table(to_value(dummy))

# non-numeric factor is converted to numeric
# starting at 1
dummy <- factor(c("D", "F", "H"))
table(to_value(dummy))

trim

Trim leading and trailing whitespaces from strings

Description

Trim leading and trailing whitespaces from strings

Usage

trim(x)

Arguments

x a character vector or string

Value

Trimmed x, i.e. with leading and trailing spaces removed.

Examples

trim("white space at end ")
trim(" white space at start and end ")
weight

Weight a variable

Description

This function weights the variable var by a specific vector of weights.

Usage

weight(var, weights)

Arguments

- var: The (unweighted) variable
- weights: A vector with same length as var, which contains weight factors. Each value of var has a specific assigned weight in weights.

Value

The weighted var.

Note

The values of the returned vector are in sorted order, whereas the values' order of the original var may be spread randomly. Hence, var can't be used, for instance, for further cross tabulation. In case you want to have weighted contingency tables or (grouped) box plots etc., use the weightBy parameter of most functions.

See Also

weight2

Examples

v <- sample(1:4, 20, TRUE)
table(v)
w <- abs(rnorm(20))
table(weight(v, w))
weight2

Weight a variable

Description
This function weights the variable `var` by a specific vector of `weights`. It's an alternative weight calculation to `weight`, though `weight` usage is recommended. This function sums up all weights values of the associated categories of `var`, whereas the `weight` function uses a `xtabs` formula to weight cases. Thus, this function may return a vector of different length than `var`.

Usage
weight2(var, weights)

Arguments
- `var`: The (unweighted) variable
- `weights`: A vector with same length as `var`, which contains weight factors. Each value of `var` has a specific assigned weight in `weights`.

Value
The weighted `var`.

Note
See 'Note' in `weight`

See Also
weight

Examples
```r
v <- sample(1:4, 20, TRUE)
table(v)
w <- abs(rnorm(20))
table(weight2(v, w))
```
### word_wrap

**Insert line breaks in long labels**

**Description**

Insert line breaks in long character strings. Useful if you want to wordwrap labels / titles for plots or tables.

**Usage**

```r
word_wrap(labels, wrap, linesep = NULL)
```

**Arguments**

- **labels**: the label(s) (i.e. character string) where a line break should be inserted. You can also pass several strings as vector (e.g. `labels = c("first long string", "second long string")`).
- **wrap**: the maximum amount of chars per line (i.e. line length).
- **linesep**: by default, this parameter is NULL and a regular new line string (`"\n"`) is used. For HTML-needs, for instance, `linesep` could be "<br>".

**Value**

New label(s) with line breaks inserted at every `wrap`’s position.

**Examples**

```r
word_wrap(c("A very long string", "And another even longer string!"), 10)
message(word_wrap("Much too long string for just one line!", 15))
```

---

### write_spss

*Write content of data frame to SPSS sav-file*

**Description**

This function saves the content of a data frame to an SPSS sav-file.

**Usage**

```r
write_spss(x, path)
```

**Arguments**

- **x**: data frame that should be saved as SPSS sav-file.
- **path**: file path to the SPSS dataset.
Note

You don’t need to take care whether variables have been imported with the `read_spss` function from this package or from haven or even the foreign package, or if you have imported SPSS data and created new variables. This function does all necessary data preparation to write a properly labelled SPSS sav file.

See Also

- sjPlot manual: data initialization
- sjPlot manual: inspecting (SPSS imported) data frames
- read_spss

---

**write_stata**  
*Write content of data frame to STATA dta-file*

Description

This function saves the content of a data frame to an STATA dta-file.

Usage

```r
write_stata(x, path)
```

Arguments

- `x`: data frame that should be saved as STATA-file.
- `path`: file path to the STATA dataset.

Note

You don’t need to take care whether variables have been imported with the `read_stata` function from this package or from haven, or if you have imported STATA data and created new variables. This function does all necessary data preparation to write a properly labelled STATA file.

See Also

write_spss
Index

*Topic data
  efc, 9
efc2, 10
efc3, 10

as.factor, 47
atomic, 30–32, 48, 49

barplot, 12, 32
chisq.test, 4
chisq_gof, 3
cod, 5, 29
cor, 27
cramer, 6, 29
cronb, 6, 36
cv, 7, 38
data.frame, 25, 40, 41
dicho, 8
efc, 9
efc2, 10
efc3, 10
eta_sq, 11

factor, 22–24, 30–33, 48
ftable, 6, 29, 46

get_val_labels, 12, 12, 14, 30–33, 35, 39–42, 48
get_var_labels, 12, 14, 30, 31, 33, 34, 42
glm, 3–5, 19, 20, 29
glmer, 5, 19, 20
group_labels, 15, 19
group_str, 16, 17, 19, 45
group_var, 15, 16, 18

hoslem_gof, 19

icc, 20

is_crossed, 22, 23
is_nested, 22, 23
is_num_fac, 24

kruskal.test, 28

labelled, 48–50
levene_test, 25
list, 12, 40
lm, 7, 38, 43
lme, 7, 38

matrix, 25
mean_n, 25
merMod, 7, 20, 38, 43
mic, 26

mwu, 27

NA, 25, 26, 33–35, 39
numeric, 48, 50

phi, 6, 28
pseudo_r2, 20, 29

read.spss, 9, 31
read_dta, 33
read_sas, 12–14, 30, 30
read_spss, 9, 12–14, 30, 31, 33, 33, 47, 55
read_stata, 12–14, 32, 55
rec, 33, 35, 39
recode_to, 34, 34, 39
reliab_test, 7, 36

rmse, 7, 37

set_na, 34, 35, 39
set_val_labels, 13, 35, 39, 40, 47, 48, 50
set_var_labels, 13, 14, 41, 47

sjmisc (sjmisc-package), 3
sjmisc-package, 3

sjp.pca, 7

sjt.pca, 7
std_beta, 43
std_e, 44
str_pos, 17, 44
stringdist, 17

table, 6, 29, 46
table_values, 46
to_fac, 47, 48, 50
to_label, 47, 48, 50
to_sjPlot, 9, 30, 33, 49
to_value, 47, 48, 50
trim, 51

weight, 52, 53
weight2, 52, 53
wilcox.test, 27
wilcox_test, 27, 28
word_wrap, 54
write_spss, 12, 32, 41, 54, 55
write_stata, 55

xtabs, 6, 29, 46, 53