Package ‘sjmisc’

Type Package
Encoding UTF-8
Title Miscellaneous Data Management Tools
Version 1.0.0
Date 2015-03-21
Author Daniel Lüdecke <d.luedecke@uke.de>
Maintainer Daniel Lüdecke <d.luedecke@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, stringdist
URL https://github.com/sjPlot/sjmisc
BugReports https://github.com/sjPlot/sjmisc/issues
NeedsCompilation no
Repository CRAN
Date/Publication 2015-03-22 08:07:48

R topics documented:

\begin{verbatim}
sjmisc-package ........................................... 2
chisq_gof .............................................. 3
cramer .................................................. 4
cronb ..................................................... 5
cv .......................................................... 5
dicho ...................................................... 6
efc ........................................................ 7
efc2 ....................................................... 8
efc3 ....................................................... 8
eta_sq ..................................................... 9
\end{verbatim}
Index

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.
chisq_gof

Details

Package: sjmisc
Type: Package
Version: 1.0.0
Date: 2015-03-21
License: GPL-3

Author(s)

Daniel Lüdecke <d.luedecke@uke.de>

References

- Documentation: http://www.strengejacke.de/sjPlot
- Github: https://github.com/sjPlot/sjmisc
- Bug reports: https://github.com/sjPlot/sjmisc/issues

---

chisq_gof

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

Description

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

Usage

chisq_gof(var, prob, weights = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>a numeric vector / variable.</td>
</tr>
<tr>
<td>prob</td>
<td>a vector of probabilities (indicating the population probabilities) of the same length as var’s amount of categories / factor levels. Use nrow(table(var)) to determine the amount of necessary values for prob.</td>
</tr>
<tr>
<td>weights</td>
<td>a vector with weights, used to weight var.</td>
</tr>
</tbody>
</table>

Value

(insensibly) returns the object of the computed chisq.test.

Note

This function is a convenient function for chisq.test, performing goodness-of-fit test.
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)))

cramer(tab)  # 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

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).
**Value**

The coefficient of variation of x.

**See Also**

- UCLA-FAQ: What is the coefficient of variation?
- rmse

**Examples**

data(efc)
cv(efc$e17age)

---

**dicho**  

**Dichotimize variables**

**Description**

Dichotomizes variables into dummy variables (0/1). Dichotomization is either done by median, mean or a specific value (see dichBy).

**Usage**

dicho(var, dichBy = "median", dichVal = -1, asNum = FALSE)

**Arguments**

- **var**  
The variable 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).
efc

Examples

data(efc)
summary(efc$12hour)
table(dicho(efc$12hour))
table(dicho(efc$12hour, "mean"))
table(dicho(efc$12hour, "value", 30))

Description
Sample dataset from the EUROFAMCARE project

Note
There are two further datasets, efc2 and efc3, which slightly differ in their structure. efc2 has already attached variable label attributes to each variable. In 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.uke.de/eurofamcare/

Examples

# Attach EFC-data
data(efc)

# Show structure
str(efc)

# show first rows
head(efc)

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

# show variable labels
get_var_label(efc)

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

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
sjt.frq(efc3$e15relat)
## End(Not run)
**eta_sq**

*Compute eta-squared of fitted anova*

**Description**

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

**Usage**

`eta_sq(...)`

**Arguments**

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

**Value**

The eta-squared value.

**Note**

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

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

**References**

- stack exchange 1
- stack exchange 2
- Wikipedia: Eta-squared

**Examples**

```r
# 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 SPSS-imported data frame

Description

This function retrieves the value labels of an imported SPSS data set (via read_spss) and

- if \( x \) is a data frame, returns the all variable’s value labels as 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 data (see read_spss)) or a variable (vector) with attached value labels.

Details

This package can add (and read) value and variable labels either in foreign package style (value.labels and variable.label) or in haven package style (labels and label). By default, the haven package style is used. The sjPlot package accesses these attributes to automatically read label attributes for labelling axes categories and titles or table rows and columns.

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

You can set a default label style 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 SPSS data sets with the read_spss function and can manually be applied with the set_val_labels and set_var_labels functions.

With attached value and variable labels, most functions of this 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). Use options(autoSetValueLabels = FALSE) and options(autoSetVariableLabels = FALSE) to turn off automatic label detection.
get_var_labels

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)
```

---

**get_var_labels**

Retrieve variable labels of (an SPSS-imported) data frame or of a specific variable

Description

This function retrieves the variable labels of an imported SPSS data set (via `read_spss`) and

- if `x` is a data frame, returns the all variable labels as `list` object
- or, if `x` is a vector, returns the variable label as string.

Usage

`get_var_labels(x)`

Arguments

- `x` A data frame (containing imported SPSS data or with attached variable labels) or a vector with "label" or "variable.label" attribute.

Details

This package can add (and read) value and variable labels either in foreign package style (value.labels and variable.label) or in haven package style (labels and label). By default, the haven package style is used. The sjPlot package accesses these attributes to automatically read label attributes for labelling axes categories and titles or table rows and columns.

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

You can set a default label style via `options(value_labels = "haven")` or `options(value_labels = "foreign")`. 
get_var_labels

Value

A named char vector with all variable labels from the SPSS dataset, or a simple string vector with the variable label, if x is a variable.

Note

This function only works with vectors that have value and variable labels attached. This is automatically done by importing SPSS data sets with the read_spss function and can manually be applied with the set_val_labels and set_var_labels functions.

With attached value and variable labels, most functions of this 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). 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_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)

data(efc)
# sample data set has not attached variable labels to each vector
# so we have to do this first... use 'autoAttachVarLabels' in # function 'read_spss' to automatically perform this step.
efc <- set_var_labels(efc, get_var_labels(efc))

# get variable label
get_var_labels(efc$e42dep)

# alternative way
get_var_labels(efc)["e42dep"]
```
**group_labels**  
Create labels for recoded groups

**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 scale variable, which should 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.

**Value**

A string vector containing labels based on the grouped counts of `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 `groupsize` and `rightInterval` as used in the `group_var` function if you want to create labels for the related recoded variable.

**See Also**

- `group_var`
- `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(ecf)
efc.val <- get_val_labels(ecf)
efc.var <- get_var_labels(ecf)
## Not run:
library(sjPlot)
sjp.frq(ecf$e17age,
       title = efc.var[['e17age']],
       type = "h",
       showValueLabels = FALSE)
## End(Not run)

# bar plot with EUROFAMCARE sample dataset
# grouped variable
data(ecf)
efc.val <- get_val_labels(ecf)
efc.var <- get_var_labels(ecf)
ageGrp <- group_var(ecf$e17age)
ageGrpLab <- group_labels(ecf$e17age)
## Not run:
library(sjPlot)
sjp.frq(ageGrp,
       title = efc.var[['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. The more similar two string elements are, the higher is the chance to be combined into a group.

Usage

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

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:
newstring <- group_str(oldstring)
sjt.frq(data.frame(oldstring, newstring), removeStringVectors = FALSE, autoGroupStrings = FALSE)

newstring <- group_str(oldstring, strict = TRUE)
sjt.frq(data.frame(oldstring, newstring), removeStringVectors = FALSE, autoGroupStrings = FALSE)
## End(Not run)
```

---

**group_var**

Recode count variables into grouped factors

Description

Recode count variables into grouped factors.

Usage

```r
group_var(var, groupsize = 5, asNumeric = TRUE, rightInterval = FALSE, autoGroupCount = 3@)
```
Arguments

- **var**: The count variable, which should be 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 ggplot2 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.

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)
efc.val <- get_val_labels(efc)
efc.var <- get_var_labels(efc)
## Not run:
library(sjPlot)
sjp.frq(efc$e17age,
       title = efc.var['e17age'],
       type = "h",
       showValueLabels = FALSE)
## End(Not run)

# bar plot with EUROFAMCARE sample dataset
# grouped variable
data(efc)
```
levene_test

levene_test <- get_val_labels(efc)
efc.var <- get_var_labels(efc)

ageGrp <- group_var(efc$e17age)
ageGrpLab <- group_labels(efc$e17age)

## Not run:
library(sjPlot)
sjp.fseq(ageGrp, 
  title = efc.var[['e17age']],
  axisLabels.x = ageGrpLab)

## End(Not run)

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

Examples

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

mean_n

Compute row means with min amount of valid values

Description

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

Usage

mean_n(dat, n)
### mic

**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 all added 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, built with the R-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".

**Value**

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

---

### Arguments

- `dat`: a data.frame with at least two columns, where row means are applied.
- `n`: the amount of valid values per row to calculate the row mean. If a row’s amount of valid values is less than `n`, NA will be returned as row mean value.

**Value**

A vector with row mean values of `df` for those rows with at least `n` valid values. Else, 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))
mean_n(dat, 4)  # 1 valid return value
mean_n(dat, 3)  # 2 valid return values
mean_n(dat, 2)
mean_n(dat, 1)  # all means are shown
```
Examples

```r
# Data from the EUROFAMCARE sample dataset
data(efc)
# receive first item of COPE-index scale
start <- which(colnames(efc) == "c82cop1")
# receive last item of COPE-index scale
end <- which(colnames(efc) == "c90cop9")
# create data frame with COPE-index scale
df <- as.data.frame(efc[, 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 `wilcox_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.
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:

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

Examples

```r
## Not run:
data(efc)
# Mann-Whitney-U-Tests for elder's age by elder's dependency.
mwu(efc$e17age, efc$e42dep)
## End(Not run)
```

---

### 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)
```
\textbf{read_sas} \hspace{2cm} \textit{Import SAS dataset as data frame into R}

\textbf{Description}

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

\textbf{Usage}

\begin{verbatim}
read_sas(path, path.cat = NULL, atomic.to.fac = FALSE)
\end{verbatim}

\textbf{Arguments}

- \texttt{path} \hspace{1cm} The file path to the SAS data file.
- \texttt{path.cat} \hspace{1cm} optional, the file path to the SAS catalog file.
- \texttt{atomic.to.fac} \hspace{1cm} Logical, if \texttt{TRUE}, factor variables imported from SAS (which are imported as \texttt{atomic}) will be converted to \texttt{factors}.

\textbf{Value}

A data frame containing the SAS data. Retrieve value labels with \texttt{get_val_labels} and variable labels with \texttt{get_var_labels}.

\textbf{Note}

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

\textbf{See Also}

- \texttt{read_spss}

\textbf{read_spss} \hspace{2cm} \textit{Import SPSS dataset as data frame into R}

\textbf{Description}

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

\textbf{Usage}

\begin{verbatim}
read_spss(path, enc = NA, autoAttachVarLabels = FALSE, atomic.to.fac = FALSE, option = "haven")
\end{verbatim}
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. See `set_var_labels` for details. *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. This function adds value and variable labels to the imported variables of the data frame.

With attached value and variable labels, most functions of this 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). 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
- 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**

```
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`
Recode numeric variables

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

Usage
\[ \text{rec}(x, \text{recodes}) \]

Arguments
- \( x \) a numeric variable (vector) or a \texttt{factor} with numeric levels that should be recoded.
- \( \text{recodes} \) a string with recode pairs of old and new values. See notes for details on this parameter.

Value
A numeric variable with recoded category values.

Note
The \( \text{recodes} \) string has following syntax:

- each recode pair has to be separated by a \( ; \), e.g. \texttt{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 \texttt{min} and \texttt{max}, e.g. \texttt{"min:4=1; 5:max=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 \texttt{else}, e.g. \texttt{"3=1; 1=2; else=3"} (recodes 3 into 1, 1 into 2 and all other values into 3)
- \texttt{NA} values are allowed both as old and new value, e.g. \texttt{"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)
- \texttt{"rev"} is a special token that reverses the value order (see examples)

Variable label attributes (see, for instance, \texttt{get_var_labels}) are retained, however, value label attributes are removed.

See Also
- \texttt{set_na} for setting NA values and \texttt{recode_to} for re-shifting value ranges.
Examples

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

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

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

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

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

recode_to

Recode variable categories into new values.

Description

Recodes the categories of a variables var into new category values, beginning with the lowest value specified by parameter lowest. Useful if you want to recode dummy variables with 1/2 coding to 0/1 coding, or recoding scales from 1-4 to 0-3 etc.

Usage

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

Arguments

<table>
<thead>
<tr>
<th>var</th>
<th>The variable (vector) that should be recoded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>lowest</td>
<td>Indicating the lowest category value after recoding. Default is 0, so the new variable starts with the category value 0.</td>
</tr>
<tr>
<td>highest</td>
<td>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.</td>
</tr>
</tbody>
</table>

Value

A new variable with recoded category values, where lowest indicates the lowest value.

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)
```

relia_test

Performs a reliability test on an item scale.

Description

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

Usage

```r
relia_test(df, scaleItems = FALSE, digits = 3)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>df</td>
<td>A data frame with items (from a scale)</td>
</tr>
<tr>
<td>scaleItems</td>
<td>If TRUE, the data frame’s vectors will be scaled. Recommended, when the variables have different measures / scales.</td>
</tr>
<tr>
<td>digits</td>
<td>Amount of digits for Cronbach’s Alpha and correlation values in returned data frame.</td>
</tr>
</tbody>
</table>

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

```R
# Data from the EUROFAMCARE sample dataset
data(efc)

# retrieve variable and value labels
varlabs <- 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) == "c90cop9")

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

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

## End(Not run)

# Compute PCA on COPE-Index, and perform a
# reliability check on each extracted factor.
# Not run:
factors <- sjt.pca(df)$factor.index
findex <- sort(unique(factors))
for (i in 1:length(findex)) {
  rel.df <- subset(df, 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",
                                     Cronbach's &alpha;=%.2f")
  }
}
```
**rmse**  
*Compute root mean squared error (RMSE)*

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

**Usage**  
```r
rmse(fit)
```

**Arguments**  
- `fit`  
a fitted linear model of class `lm`, `merMod` (lme4) or `lme` (nlme).

**Value**  
The root mean squared error of `fit`.

**See Also**  
`cv`

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

---

**set_na**  
*Set NA for specific variable values*

**Description**  
This function sets specific values of a variable `var` as missings (NA).

**Usage**  
```r
set_na(var, values)
```
**Arguments**

- `var` a variable where new missing values should be defined.
- `values` a numeric vector with values that should be replaced with `NA`'s.

**Value**

The `var` where each value of `values` 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**

```r
# 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)
```

---

**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 most of this package's functions, in order to automatically set values or legend labels.

**Usage**

```r
set_val_labels(x, labels)
```
set_val_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.

Details

This package can add (and read) value and variable labels either in foreign package style (value.labels and variable.label) or in haven package style (labels and label). By default, the haven package style is used. The sjPlot package accesses these attributes to automatically read label attributes for labelling axes categories and titles or table rows and columns. Furthermore, value and variable labels are used when saving data, e.g. to SPSS (see write_spss), which means that the written SPSS file contains proper labels for each variable. You can set a default label style via options(value_labels = "haven") or options(value_labels = "foreign").

Value

- x with attached value labels.

Note

With attached value and variable labels, most functions of this 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). 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
- 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)
```
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 of this package's functions can automatically retrieve the variable name to use it as axis labels or plot title (see details).

Usage

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).
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

This package can add (and read) value and variable labels either in foreign package style (value.labels and variable.label) or in haven package style (labels and label). By default, the haven package style is used. The sjPlot package accesses these attributes to automatically read label attributes for labelling axes categories and titles or table rows and columns. Furthermore, value and variable labels are used when saving data, e.g. to SPSS (see write_spss), which means that the written SPSS file contains proper labels for each variable.
You can set a default label style via options(value_labels = "haven") or options(value_labels = "foreign").

Value

x, with attached variable label attribute(s), which contains the variable name(s).

Note

With attached value and variable labels, most functions of this 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). 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
- get_var_labels

Examples

```r
# sample data set, imported from SPSS. Variable labels are attached
# as attribute to the data frame (so variables currently don't have this attribute)
data(efc)
# get variable labels
variable.labels <- get_var_labels(efc)
# set variable labels as attribute to each single variable of data frame
efc <- set_var_labels(efc, variable.labels)

## Not run:
sjt.frq(efc$s42dep)
sjt.frq(data.frame(efc$s42dep, efc$s16sex))
## 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:
sjp.frq(dummy, title = NULL)
## End(Not run)
```

---

**std_beta**

*Compute std. beta coefficients and ci of lm and mixed models*

**Description**

Returns the standardized beta coefficients and confidence intervals of a fitted linear (mixed) models, i.e. `fit` must either be of class `lm` or `lmerMod` (lme4-package).

**Usage**

```r
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

```r
# 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 variables

Usage

std_e(x)
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.
showProgressBar If TRUE, the progress bar is displayed when computing the distance matrix. Default in FALSE, hence the bar is hidden.

Examples

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

Arguments

x a (numeric) vector / variable.

Value

The standard error of variable x.
### 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,
  "sysme",
  part.dist.match = 1)

# finds nothing
str_pos("We are Sex Pistols!", "postils")
# finds partial matching of similarity
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)
```
to_fac

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**

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 `var`. See examples.

Usage

`to_fac(x)`

Arguments

**x**
A (numeric or atomic) variable.

Value

A factor variable, including variable and value labels.

Note

This function only works with vectors that have value and variable labels attached. This is automatically done by importing SPSS data sets with the `read_spss` function and can manually be applied with the `set_val_labels` and `set_var_labels` functions.
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)
# 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).

### Value

A factor variable with the associated value labels as factor levels.

### Note

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

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 vector classes.

Note

This function is currently only used to avoid possible compatibility issues with labelled class vectors. Some known issues with labelled class vectors have already been fixed, so it might be that this function will become redundant in the future.

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.

Examples

data(efc)
print(get_val_labels(efc)['c161sex'])
head(efc$c161sex)
head(to_label(efc$c161sex))

print(get_val_labels(efc)['e42dep'])
table(efc$e42dep)
table(to_label(efc$e42dep))

# structure of numeric values won't be changed
# by this function, it only applies to labelled vectors
# (typically categorical or factor variables)
str(efc$e17age)
str(to_label(efc$e17age))
to_value

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**

```r
to_value(x, startAt = 1, keep.labels = TRUE)
```

**Arguments**

- `x`: A (factor) variable.
- `startAt`: the starting index, i.e. the lowest numeric value of the variable’s value range.
- `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 from `startAt` to `startAt` + length of factor levels.

**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**

```r
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))
```
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 categories 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

```r
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 value with a different length than that from `var`.

**Usage**

```r
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**

The values of the returned vector are in sorted order, whereas the categories 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**

- `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 plot labels.

**Usage**

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

**Arguments**

- **labels**  The label(s) (i.e. character string). You can also pass several strings as vector (e.g. `labels=c("first long string", "second long string")`)
- **wrap**  The amount of chars per line (i.e. line length)
- **linesep**  By default, this parameter is `NULL` and a regular new line string 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)
```

---

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.
write_stata

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
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, 7
  efc2, 8
  efc3, 8

atomic, 21–23, 37, 38

chisq.test, 3
chisq_gof, 3
cor, 18
cramer, 4, 20
cronb, 5, 27
cv, 5, 28

data.frame, 17, 18, 29, 31
dicho, 6

efc, 7
efc2, 8
efc3, 8
eta_sq, 9

factor, 21–24, 37
ftable, 4, 20, 36

g et_val_labels, 10, 21–23, 25, 29, 30, 37
get_var_labels, 11, 21–24, 32
 group_labels, 13, 16
 group_str, 13, 14, 16, 35
 group_var, 13, 15

kruskal.test, 20

labelled, 37, 38
levene.test, 17
list, 10, 11, 30
lm, 5, 28, 33
lme, 5, 28

matrix, 17
mean_n, 17

merMod, 5, 28, 33
mic, 18
mww, 19

NA, 17, 18, 24, 29
numeric, 37

phi, 4, 20

read.spss, 22
read_dta, 23
read_sas, 21, 21
read_spss, 10–12, 21, 21, 22, 23, 36, 43
read_stata, 23, 43
rec, 24, 26, 29
recode_to, 24, 25, 29
reliab_test, 5, 26
rmse, 6, 28

set_na, 24, 26, 28
set_val_labels, 10–12, 25, 29, 36, 37, 39
set_var_labels, 10, 12, 22, 31, 36
sjmisc (sjmisc-package), 2
sjmisc-package, 2
sjp.pca, 5
sjt.pca, 5
std_beta, 32
std_e, 33
str_pos, 15, 34

table, 4, 20, 36
table_values, 35
to_fac, 36, 38, 39
to_label, 37, 37, 39
to_sjPlot, 21, 23, 38
to_value, 37, 38, 39

weight, 40, 41
weight2, 40, 41
wilcox.test, 19
wilcox_test, 19, 20
INDEX

word_wrap, 42
write_spss, 10, 11, 22, 30, 31, 42, 43
write_stata, 43

xtabs, 4, 20, 36, 41