Package 'sjlabelled'

May 9, 2026

Type Package

Encoding UTF-8

Title Labelled Data Utility Functions

Version 1.2.0

Maintainer Daniel Lüdecke <d.luedecke@uke.de>

Description Collection of functions dealing with labelled data, like reading and writing data between R and other statistical software packages like 'SPSS', 'SAS' or 'Stata', and working with labelled data. This includes easy ways to get, set or change value and variable label attributes, to convert labelled vectors into factors or numeric (and vice versa), or to deal with multiple declared missing values.

License GPL-3

Depends R (>= 3.4)

Imports insight, datawizard, stats, tools, utils

Suggests dplyr, haven (>= 1.1.2), magrittr, sjmisc, sjPlot, knitr, rlang, rmarkdown, snakecase, testthat

URL https://strengejacke.github.io/sjlabelled/

BugReports https://github.com/strengejacke/sjlabelled/issues

RoxygenNote 7.1.2

VignetteBuilder knitr

NeedsCompilation no

Author Daniel Lüdecke [aut, cre] (ORCID: <https://orcid.org/0000-0002-8895-3206>), avid Ranzolin [ctb], Jonathan De Troye [ctb]

Repository CRAN

Date/Publication 2022-04-10 09:30:02 UTC

Contents

`sjlabelled-package` *Labelled Data Utility Functions*

Description

Purpose of this package

Collection of miscellaneous utility functions (especially intended for people coming from other statistical software packages like 'SPSS', and/or who are new to R), supporting following common tasks when working with labelled data:

- Reading and writing data between R and other statistical software packages like 'SPSS', 'SAS' or 'Stata'
- Easy ways to get, set and change value and variable label attributes, to convert labelled vectors into factors (and vice versa), or to deal with multiple declared missing values etc.

Author(s)

Daniel Lüdecke <d.luedecke@uke.de>

`add_labels` *Add, replace or remove value labels of variables*

Description

These functions add, replace or remove value labels to or from variables.

Usage

```
add_labels(x, ..., labels)

replace_labels(x, ..., labels)

remove_labels(x, ..., labels)
```

Arguments

`x`	A vector or data frame.
`...`	Optional, unquoted names of variables that should be selected for further processing. Required, if `x` is a data frame (and no vector) and only selected variables from `x` should be processed. You may also use functions like `:` or tidyselect's select-helpers. See 'Examples'.
`labels`	**For** `add_labels()` A named (numeric) vector of labels that will be added to `x` as label attribute. **For** `remove_labels()` Either a numeric vector, indicating the position of one or more label attributes that should be removed; a character vector with names of label attributes that should be removed; or a `tagged_na()` to remove the labels from specific NA values.

Details

`add_labels()` adds `labels` to the existing value labels of `x`, however, unlike `set_labels`, it does *not* remove labels that were *not* specified in `labels`. `add_labels()` also replaces existing value labels, but preserves the remaining labels.

`remove_labels()` is the counterpart to `add_labels()`. It removes labels from a label attribute of `x`.

`replace_labels()` is an alias for `add_labels()`.

Value

`x` with additional or removed value labels. If `x` is a data frame, the complete data frame `x` will be returned, with removed or added to variables specified in `...`; if `...` is not specified, applies to all variables in the data frame.

See Also

set_label to manually set variable labels or get_label to get variable labels; set_labels to add value labels, replacing the existing ones (and removing non-specified value labels).

Examples

```
# add_labels()
data(efc)
get_labels(efc$e42dep)

x <- add_labels(efc$e42dep, labels = c(`nothing` = 5))
get_labels(x)

if (require("dplyr")) {
  x <- efc %>%
    # select three variables
    dplyr::select(e42dep, c172code, c161sex) %>%
    # only add new label to two of those
    add_labels(e42dep, c172code, labels = c(`nothing` = 5))
  # see data frame, with selected variables having new labels
  get_labels(x)
}

x <- add_labels(efc$e42dep, labels = c(`nothing` = 5, `zero value` = 0))
get_labels(x, values = "p")

# replace old value labels
x <- add_labels(
  efc$e42dep,
  labels = c(`not so dependent` = 4, `lorem ipsum` = 5)
)
get_labels(x, values = "p")

# replace specific missing value (tagged NA)
if (require("haven")) {
  x <- labelled(c(1:3, tagged_na("a", "c", "z"), 4:1),
                c("Agreement" = 1, "Disagreement" = 4, "First" = tagged_na("c"),
                  "Refused" = tagged_na("a"), "Not home" = tagged_na("z")))
  # get current NA values
  x
  # tagged NA(c) has currently the value label "First", will be
  # replaced by "Second" now.
  replace_labels(x, labels = c("Second" = tagged_na("c")))
}


# remove_labels()

x <- remove_labels(efc$e42dep, labels = 2)
get_labels(x, values = "p")

x <- remove_labels(efc$e42dep, labels = "independent")
```

```
get_labels(x, values = "p")

if (require("haven")) {
  x <- labelled(c(1:3, tagged_na("a", "c", "z"), 4:1),
                c("Agreement" = 1, "Disagreement" = 4, "First" = tagged_na("c"),
                  "Refused" = tagged_na("a"), "Not home" = tagged_na("z")))
  # get current NA values
  get_na(x)
  get_na(remove_labels(x, labels = tagged_na("c")))
}
```

---

as_character *Convert variable into factor with associated value labels*

Description

as_label() converts (replaces) values of a variable (also of factors or character vectors) 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. as_character() does the same as as_label(), but returns a character vector.

Usage

```
as_character(x, ...)

to_character(x, ...)

## S3 method for class 'data.frame'
as_character(
  x,
  ...,
  add.non.labelled = FALSE,
  prefix = FALSE,
  var.label = NULL,
  drop.na = TRUE,
  drop.levels = FALSE,
  keep.labels = FALSE
)

as_label(x, ...)

to_label(x, ...)

## S3 method for class 'data.frame'
as_label(
  x,
  ...,
```

```
  add.non.labelled = FALSE,
  prefix = FALSE,
  var.label = NULL,
  drop.na = TRUE,
  drop.levels = FALSE,
  keep.labels = FALSE
)
```

Arguments

x	A vector or data frame.
...	Optional, unquoted names of variables that should be selected for further processing. Required, if x is a data frame (and no vector) and only selected variables from x should be processed. You may also use functions like : or tidyselect's select-helpers. See 'Examples'.
add.non.labelled	Logical, if TRUE, values without associated value label will also be converted to labels (as is). See 'Examples'.
prefix	Logical, if TRUE, the value labels used as factor levels or character values will be prefixed with their associated values. See 'Examples'.
var.label	Optional string, to set variable label attribute for the returned variable (see vignette Labelled Data and the sjlabelled-Package). If NULL (default), variable label attribute of x will be used (if present). If empty, variable label attributes will be removed.
drop.na	Logical, if TRUE, tagged NA values with value labels will be converted to regular NA's. Else, tagged NA values will be replaced with their value labels. See 'Examples' and get_na.
drop.levels	Logical, if TRUE, unused factor levels will be dropped (i.e. droplevels will be applied before returning the result).
keep.labels	Logical, if TRUE, value labels are preserved This allows users to quickly convert back factors to numeric vectors with as_numeric().

Details

See 'Details' in get_na.

Value

A factor with the associated value labels as factor levels. If x is a data frame, the complete data frame x will be returned, where variables specified in ... are coerced to factors; if ... is not specified, applies to all variables in the data frame. as_character() returns a character vector.

Note

Value label attributes (see get_labels) will be removed when converting variables to factors.

Examples

```
data(efc)
print(get_labels(efc)['c161sex'])
head(efc$c161sex)
head(as_label(efc$c161sex))

print(get_labels(efc)['e42dep'])
table(efc$e42dep)
table(as_label(efc$e42dep))

head(efc$e42dep)
head(as_label(efc$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$e17age)
str(as_label(efc$e17age))


# factor with non-numeric levels
as_label(factor(c("a", "b", "c")))

# factor with non-numeric levels, prefixed
x <- factor(c("a", "b", "c"))
x <- set_labels(x, labels = c("ape", "bear", "cat"))
as_label(x, prefix = TRUE)


# create vector
x <- c(1, 2, 3, 2, 4, NA)
# add less labels than values
x <- set_labels(
  x,
  labels = c("yes", "maybe", "no"),
  force.labels = FALSE,
  force.values = FALSE
)

# convert to label w/o non-labelled values
as_label(x)

# convert to label, including non-labelled values
as_label(x, add.non.labelled = TRUE)


# create labelled integer, with missing flag
if (require("haven")) {
  x <- labelled(
    c(1:3, tagged_na("a", "c", "z"), 4:1, 2:3),
    c("Agreement" = 1, "Disagreement" = 4, "First" = tagged_na("c"),
```

```
      "Refused" = tagged_na("a"), "Not home" = tagged_na("z"))
  )

  # to labelled factor, with missing labels
  as_label(x, drop.na = FALSE)

  # to labelled factor, missings removed
  as_label(x, drop.na = TRUE)

  # keep missings, and use non-labelled values as well
  as_label(x, add.non.labelled = TRUE, drop.na = FALSE)
}

# convert labelled character to factor
dummy <- c("M", "F", "F", "X")
dummy <- set_labels(
  dummy,
  labels = c(`M` = "Male", `F` = "Female", `X` = "Refused")
)
get_labels(dummy,, "p")
as_label(dummy)

# drop unused factor levels, but preserve variable label
x <- factor(c("a", "b", "c"), levels = c("a", "b", "c", "d"))
x <- set_labels(x, labels = c("ape", "bear", "cat"))
set_label(x) <- "A factor!"
x
as_label(x, drop.levels = TRUE)

# change variable label
as_label(x, var.label = "New variable label!", drop.levels = TRUE)


# convert to numeric and back again, preserving label attributes
# *and* values in numeric vector
x <- c(0, 1, 0, 4)
x <- set_labels(x, labels = c(`null` = 0, `one` = 1, `four` = 4))

# to factor
as_label(x)

# to factor, back to numeric - values are 1, 2 and 3,
# instead of original 0, 1 and 4
as_numeric(as_label(x))

# preserve label-attributes when converting to factor, use these attributes
# to restore original numeric values when converting back to numeric
as_numeric(as_label(x, keep.labels = TRUE), use.labels = TRUE)


# easily coerce specific variables in a data frame to factor
# and keep other variables, with their class preserved
as_label(efc, e42dep, e16sex, c172code)
```

as_factor *Convert variable into factor and keep value labels*

Description

This function converts a variable into a factor, but preserves variable and value label attributes.

Usage

```
as_factor(x, ...)

to_factor(x, ...)

## S3 method for class 'data.frame'
as_factor(x, ..., add.non.labelled = FALSE)
```

Arguments

x	A vector or data frame.
...	Optional, unquoted names of variables that should be selected for further processing. Required, if x is a data frame (and no vector) and only selected variables from x should be processed. You may also use functions like : or tidyselect's select-helpers. See 'Examples'.
add.non.labelled	Logical, if TRUE, non-labelled values also get value labels.

Details

as_factor converts numeric values into a factor with numeric levels. as_label, however, converts a vector into a factor and uses value labels as factor levels.

Value

A factor, including variable and value labels. If x is a data frame, the complete data frame x will be returned, where variables specified in ... are coerced to factors (including variable and value labels); if ... is not specified, applies to all variables in the data frame.

Note

This function is intended for use with vectors that have value and variable label attributes. Unlike as.factor, as_factor converts a variable into a factor and preserves the value and variable label attributes.

Adding label attributes 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_labels and set_label.

Examples

```
if (require("sjmisc") && require("magrittr")) {
  data(efc)
  # normal factor conversion, loses value attributes
  x <- as.factor(efc$e42dep)
  frq(x)

  # factor conversion, which keeps value attributes
  x <- as_factor(efc$e42dep)
  frq(x)

  # create partially labelled vector
  x <- set_labels(
    efc$e42dep,
    labels = c(
      `1` = "independent",
      `4` = "severe dependency",
      `9` = "missing value"
   ))

  # only copy existing value labels
  as_factor(x) %>% head()
  get_labels(as_factor(x), values = "p")

  # also add labels to non-labelled values
  as_factor(x, add.non.labelled = TRUE) %>% head()
  get_labels(as_factor(x, add.non.labelled = TRUE), values = "p")


  # easily coerce specific variables in a data frame to factor
  # and keep other variables, with their class preserved
  as_factor(efc, e42dep, e16sex, c172code) %>% head()

  # use select-helpers from dplyr-package
  if (require("dplyr")) {
    as_factor(efc, contains("cop"), c161sex:c175empl) %>% head()
  }
}
```

---

as_labelled *Convert vector to labelled class*

Description

Converts a (labelled) vector of any class into a `labelled` class vector, resp. adds a `labelled` class-attribute.

Usage

```
as_labelled(
  x,
  add.labels = FALSE,
  add.class = FALSE,
  skip.strings = FALSE,
  tag.na = FALSE
)
```

Arguments

x	Variable (vector), data.frame or list of variables that should be converted to labelled()-class objects.
add.labels	Logical, if TRUE, non-labelled values will be labelled with the corresponding value.
add.class	Logical, if TRUE, x preserves its former class-attribute and labelled is added as additional attribute. If FALSE (default), all former class-attributes will be removed and the class-attribute of x will only be labelled.
skip.strings	Logical, if TRUE, character vector are not converted into labelled-vectors. Else, character vectors are converted to factors vector and the associated values are used as value labels.
tag.na	Logical, if TRUE, tagged NA values are replaced by their associated values. This is required, for instance, when writing data back to SPSS.

Value

x, as labelled-class object.

Examples

```
data(efc)
str(efc$e42dep)

x <- as_labelled(efc$e42dep)
str(x)

x <- as_labelled(efc$e42dep, add.class = TRUE)
str(x)

a <- c(1, 2, 4)
x <- as_labelled(a, add.class = TRUE)
str(x)

data(efc)
x <- set_labels(efc$e42dep,
                labels = c(`1` = "independent", `4` = "severe dependency"))
x1 <- as_labelled(x, add.labels = FALSE)
x2 <- as_labelled(x, add.labels = TRUE)
```

```
str(x1)
str(x2)

get_values(x1)
get_values(x2)
```

`as_numeric` *Convert factors to numeric variables*

Description

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

Usage

```
as_numeric(x, ...)

to_numeric(x, ...)

## S3 method for class 'data.frame'
as_numeric(x, ..., start.at = NULL, keep.labels = TRUE, use.labels = FALSE)
```

Arguments

`x`	A vector or data frame.
`...`	Optional, unquoted names of variables that should be selected for further processing. Required, if `x` is a data frame (and no vector) and only selected variables from `x` should be processed. You may also use functions like `:` or tidyselect's select-helpers. See 'Examples'.
`start.at`	Starting index, i.e. the lowest numeric value of the variable's value range. By default, this argument is `NULL`, hence the lowest value of the returned numeric variable corresponds to the lowest factor level (if factor levels are numeric) or to `1` (if factor levels are not numeric).
`keep.labels`	Logical, if `TRUE`, former factor levels will be added as value labels. For numeric factor levels, values labels will be used, if present. See 'Examples' and `set_labels` for more details.
`use.labels`	Logical, if `TRUE` and `x` has numeric value labels, the values defined in the labels (right-hand side of `labels`, for instance `labels = c(null = 0, one = 1)`) will be set as numeric values (instead of consecutive factor level numbers). See 'Examples'.

Value

A numeric variable with values ranging either from `start.at` to `start.at` + length of factor levels, or to the corresponding factor levels (if these were numeric). If `x` is a data frame, the complete data frame `x` will be returned, where variables specified in `...` are coerced to numeric; if `...` is not specified, applies to all variables in the data frame.

Examples

```
data(efc)
test <- as_label(efc$e42dep)
table(test)

table(as_numeric(test))
hist(as_numeric(test, start.at = 0))

# set lowest value of new variable to "5".
table(as_numeric(test, start.at = 5))

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

# do not drop unused factor levels
dummy <- ordered(c(rep("No", 5), rep("Maybe", 3)),
                 levels = c("Yes", "No", "Maybe"))
as_numeric(dummy)

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

# for numeric factor levels, value labels will be used, if present
dummy1 <- factor(c("3", "4", "6"))
dummy1 <- set_labels(dummy1, labels = c("first", "2nd", "3rd"))
dummy1
as_numeric(dummy1)

# for non-numeric factor levels, these will be used.
# value labels will be ignored
dummy2 <- factor(c("D", "F", "H"))
dummy2 <- set_labels(dummy2, labels = c("first", "2nd", "3rd"))
dummy2
as_numeric(dummy2)


# easily coerce specific variables in a data frame to numeric
# and keep other variables, with their class preserved
data(efc)
efc$e42dep <- as.factor(efc$e42dep)
efc$e16sex <- as.factor(efc$e16sex)
efc$e17age <- as.factor(efc$e17age)

# convert back "sex" and "age" into numeric
head(as_numeric(efc, e16sex, e17age))

x <- factor(c("None", "Little", "Some", "Lots"))
x <- set_labels(x,
  labels = c(None = "0.5", Little = "1.3", Some = "1.8", Lots = ".2")
```

```
)
x
as_numeric(x)
as_numeric(x, use.labels = TRUE)
as_numeric(x, use.labels = TRUE, keep.labels = FALSE)
```

---

convert_case *Generic case conversion for labels*

Description

This function wraps to_any_case() from the **snakecase** package with certain defaults for the sep_in and sep_out arguments, used for instance to convert cases in term_labels.

Usage

```
convert_case(lab, case = NULL, verbose = FALSE, ...)
```

Arguments

lab	Character vector that should be case converted.
case	Desired target case. Labels will automatically converted into the specified character case. See to_any_case() for more details on this argument.
verbose	Toggle warnings and messages on or off.
...	Further arguments passed down to to_any_case(), like sep_in or sep_out.

Details

When calling to_any_case() from **snakecase**, the sep_in argument is set to "(?<!\\d)\\.", and the sep_out to " ". This gives feasible results from variable labels for plot annotations.

Value

lab, with converted case.

Examples

```
data(iris)
convert_case(colnames(iris))
convert_case(colnames(iris), case = "snake")
```

copy_labels *Copy value and variable labels to (subsetted) data frames*

Description

Subsetting-functions usually drop value and variable labels from subsetted data frames (if the original data frame has value and variable label attributes). This function copies these value and variable labels back to subsetted data frames that have been subsetted, for instance, with subset.

Usage

```
copy_labels(df_new, df_origin = NULL, ...)
```

Arguments

df_new	The new, subsetted data frame.
df_origin	The original data frame where the subset (df_new) stems from; use NULL, if value and variable labels from df_new should be removed.
...	Optional, unquoted names of variables that should be selected for further processing. Required, if x is a data frame (and no vector) and only selected variables from x should be processed. You may also use functions like : or tidyselect's select-helpers. See 'Examples'.

Value

Returns df_new with either removed value and variable label attributes (if df_origin = NULL) or with copied value and variable label attributes (if df_origin was the original subsetted data frame).

Note

In case df_origin = NULL, all possible label attributes from df_new are removed.

Examples

```
data(efc)

# create subset - drops label attributes
efc.sub <- subset(efc, subset = e16sex == 1, select = c(4:8))
str(efc.sub)

# copy back attributes from original dataframe
efc.sub <- copy_labels(efc.sub, efc)
str(efc.sub)

# remove all labels
efc.sub <- copy_labels(efc.sub)
str(efc.sub)
```

```
# create subset - drops label attributes
efc.sub <- subset(efc, subset = e16sex == 1, select = c(4:8))
if (require("dplyr")) {
  # create subset with dplyr's select - attributes are preserved
  efc.sub2 <- select(efc, c160age, e42dep, neg_c_7, c82cop1, c84cop3)
  # copy labels from those columns that are available
  copy_labels(efc.sub, efc.sub2) %>% str()
}

# copy labels from only some columns
str(copy_labels(efc.sub, efc, e42dep))
str(copy_labels(efc.sub, efc, -e17age))
```

`drop_labels` *Drop, add or convert (non-)labelled values*

Description

For (partially) labelled vectors, `zap_labels()` will replace all values that have a value label attribute with `NA`; `zap_unlabelled()`, as counterpart, will replace all values that *don't* have a value label attribute with `NA`.

`drop_labels()` drops all value labels for unused values, i.e. values that are not present in a vector. `fill_labels()` is the counterpart to `drop_labels()` and adds value labels to a partially labelled vector, i.e. if not all values are labelled, non-labelled values get labels.

Usage

```
drop_labels(x, ..., drop.na = TRUE)

fill_labels(x, ...)

zap_labels(x, ...)

zap_unlabelled(x, ...)
```

Arguments

`x`	(partially) `labelled()` vector or a data frame with such vectors.
`...`	Optional, unquoted names of variables that should be selected for further processing. Required, if `x` is a data frame (and no vector) and only selected variables from `x` should be processed. You may also use functions like `:` or tidyselect's select-helpers. See 'Examples'.
`drop.na`	Logical, whether existing value labels of tagged NA values (see `tagged_na`) should be removed (`drop.na = TRUE`, the default) or preserved (`drop.na = FALSE`). See `get_na` for more details on tagged NA values.

Value

- For `zap_labels()`, `x`, where all labelled values are converted to `NA`.
- For `zap_unlabelled()`, `x`, where all non-labelled values are converted to `NA`.
- For `drop_labels()`, `x`, where value labels for non-existing values are removed.
- For `fill_labels()`, `x`, where labels for non-labelled values are added.

If `x` is a data frame, the complete data frame `x` will be returned, with variables specified in `...` being converted; if `...` is not specified, applies to all variables in the data frame.

Examples

```
if (require("sjmisc") && require("dplyr")) {

  # zap_labels() ----

  data(efc)
  str(efc$e42dep)

  x <- set_labels(
    efc$e42dep,
    labels = c("independent" = 1, "severe dependency" = 4)
  )
  table(x)
  get_values(x)
  str(x)

  # zap all labelled values
  table(zap_labels(x))
  get_values(zap_labels(x))
  str(zap_labels(x))

  # zap all unlabelled values
  table(zap_unlabelled(x))
  get_values(zap_unlabelled(x))
  str(zap_unlabelled(x))

  # in a pipe-workflow
  efc %>%
    select(c172code, e42dep) %>%
    set_labels(
      e42dep,
      labels = c("independent" = 1, "severe dependency" = 4)
    ) %>%
    zap_labels()


  # drop_labels() ----

  rp <- rec_pattern(1, 100)
  rp
```

```
# sample data
data(efc)
# recode carers age into groups of width 5
x <- rec(efc$c160age, rec = rp$pattern)
# add value labels to new vector
x <- set_labels(x, labels = rp$labels)

# watch result. due to recode-pattern, we have age groups with
# no observations (zero-counts)
frq(x)
# now, let's drop zero's
frq(drop_labels(x))

# drop labels, also drop NA value labels, then also zap tagged NA
if (require("haven")) {
  x <- labelled(c(1:3, tagged_na("z"), 4:1),
                c("Agreement" = 1, "Disagreement" = 4, "Unused" = 5,
                  "Not home" = tagged_na("z")))
  x
  drop_labels(x, drop.na = FALSE)
  drop_labels(x)
  zap_na_tags(drop_labels(x))

  # fill_labels() ----

  # create labelled integer, with tagged missings
  x <- labelled(
    c(1:3, tagged_na("a", "c", "z"), 4:1),
    c("Agreement" = 1, "Disagreement" = 4, "First" = tagged_na("c"),
      "Refused" = tagged_na("a"), "Not home" = tagged_na("z"))
    )
  # get current values and labels
  x
  get_labels(x)

  fill_labels(x)
  get_labels(fill_labels(x))
  # same as
  get_labels(x, non.labelled = TRUE)
}
}
```

---

efc *Sample dataset from the EUROFAMCARE project*

Description

A SPSS sample data set, imported with the read_spss function.

Examples

```
# Attach EFC-data
data(efc)

# Show structure
str(efc)

# show first rows
head(efc)

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

# show variable labels
get_label(efc)

# plot efc-data frame summary
sjt.df(efc, altr.row.col = TRUE)
## End(Not run)
```

`get_label` *Retrieve variable label(s) of labelled data*

Description

This function returns the variable labels of labelled data.

Usage

```
get_label(x, ..., def.value = NULL, case = NULL)
```

Arguments

`x`	A data frame with variables that have label attributes (e.g. from an imported SPSS, SAS or STATA data set, via `read_spss`, `read_sas` or `read_stata`); a variable (vector) with variable label attribute; or a `list` of variables with variable label attributes. See 'Examples'.
`...`	Optional, names of variables, where labels should be retrieved. Required, if either data is a data frame and no vector, or if only selected variables from `x` should be used in the function. Convenient argument to work with pipe-chains (see 'Examples').
`def.value`	Optional, a character string which will be returned as label if `x` has no label attribute. By default, `NULL` is returned.
`case`	Desired target case. Labels will automatically converted into the specified character case. See `to_any_case()` for more details on this argument.

Value

A named character vector with all variable labels from the data frame or list; or a simple character vector (of length 1) with the variable label, if x is a variable. If x is a single vector and has no label attribute, the value of `def.value` will be returned (which is by default `NULL`).

Note

`var_labels` is an alternative way to set variable labels, which follows the philosophy of tidyvers API design (data as first argument, dots as value pairs indicating variables)

See Also

See vignette Labelled Data and the sjlabelled-Package for more details; `set_label` to manually set variable labels or `get_labels` to get value labels; `var_labels` to set multiple variable labels at once.

Examples

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

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

# retrieve value labels
# mydat.val <- get_labels(mydat)

data(efc)

# get variable lable
get_label(efc$e42dep)

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

# 'get_label()' also works within pipe-chains
library(magrittr)
efc %>% get_label(e42dep, e16sex)

# set default values
get_label(mtcars, mpg, cyl, def.value = "no var labels")

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

# get labels from multiple variables
get_label(list(efc$e42dep, efc$e16sex, efc$e15relat))
```

```
# use case conversion for human-readable labels
data(iris)
get_label(iris, def.value = colnames(iris))
get_label(iris, def.value = colnames(iris), case = "parsed")
```

`get_labels` *Retrieve value labels of labelled data*

Description

This function returns the value labels of labelled data.

Usage

```
get_labels(
  x,
  attr.only = FALSE,
  values = NULL,
  non.labelled = FALSE,
  drop.na = TRUE,
  drop.unused = FALSE
)
```

Arguments

`x`	A data frame with variables that have value label attributes (e.g. from an imported SPSS, SAS or STATA data set, via `read_spss`, `read_sas` or `read_stata`); a variable (vector) with value label attributes; or a `list` of variables with values label attributes. If `x` has no label attributes, factor levels are returned. See 'Examples'.
`attr.only`	Logical, if `TRUE`, labels are only searched for in the the vector's `attributes`; else, if `attr.only = FALSE` and `x` has no label attributes, factor levels or string values are returned. See 'Examples'.
`values`	String, indicating whether the values associated with the value labels are returned as well. If `values = "as.name"` (or `values = "n"`), values are set as `names` attribute of the returned object. If `values = "as.prefix"` (or `values = "p"`), values are included as prefix to each label. See 'Examples'.
`non.labelled`	Logical, if `TRUE`, values without labels will also be included in the returned labels (see `fill_labels`).
`drop.na`	Logical, whether labels of tagged NA values (see `tagged_na()`) should be included in the return value or not. By default, labelled (tagged) missing values are not returned. See `get_na` for more details on tagged NA values.
`drop.unused`	Logical, if `TRUE`, unused labels will be removed from the return value.

Value

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

See Also

See vignette Labelled Data and the sjlabelled-Package for more details; set_labels to manually set value labels, get_label to get variable labels and get_values to retrieve the values associated with value labels.

Examples

```
# import SPSS data set
# mydat <- read_spss("my_spss_data.sav")

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

# retrieve value labels
# mydat.val <- get_labels(mydat)

data(efc)
get_labels(efc$e42dep)

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

# include associated values
get_labels(efc$e42dep, values = "as.name")

# include associated values
get_labels(efc$e42dep, values = "as.prefix")

# get labels from multiple variables
get_labels(list(efc$e42dep, efc$e16sex, efc$e15relat))


# create a dummy factor
f1 <- factor(c("hi", "low", "mid"))
# search for label attributes only
get_labels(f1, attr.only = TRUE)
# search for factor levels as well
get_labels(f1)

# same for character vectors
c1 <- c("higher", "lower", "mid")
# search for label attributes only
get_labels(c1, attr.only = TRUE)
```

```
# search for string values as well
get_labels(c1)


# create vector
x <- c(1, 2, 3, 2, 4, NA)
# add less labels than values
x <- set_labels(x, labels = c("yes", "maybe", "no"), force.values = FALSE)
# get labels for labelled values only
get_labels(x)
# get labels for all values
get_labels(x, non.labelled = TRUE)


# get labels, including tagged NA values
library(haven)
x <- labelled(c(1:3, tagged_na("a", "c", "z"), 4:1),
              c("Agreement" = 1, "Disagreement" = 4, "First" = tagged_na("c"),
                "Refused" = tagged_na("a"), "Not home" = tagged_na("z")))
# get current NA values
x
get_labels(x, values = "n", drop.na = FALSE)


# create vector with unused labels
data(efc)
efc$e42dep <- set_labels(
  efc$e42dep,
  labels = c("independent" = 1, "dependent" = 4, "not used" = 5)
)
get_labels(efc$e42dep)
get_labels(efc$e42dep, drop.unused = TRUE)
get_labels(efc$e42dep, non.labelled = TRUE, drop.unused = TRUE)
```

`get_na` *Retrieve tagged NA values of labelled variables*

Description

This function retrieves tagged NA values and their associated value labels from a labelled vector.

Usage

```
get_na(x, as.tag = FALSE)
```

Arguments

`x`	Variable (vector) with value label attributes, including tagged missing values (see `tagged_na()`); or a data frame or list with such variables.
`as.tag`	Logical, if `TRUE`, the returned values are not tagged NA's, but their string representative including the tag value. See 'Examples'.

Details

Other statistical software packages (like 'SPSS' or 'SAS') allow to define multiple missing values, e.g. *not applicable*, *refused answer* or "real" missing. These missing types may be assigned with different values, so it is possible to distinguish between these missing types. In R, multiple declared missings cannot be represented in a similar way with the regular missing values. However, `tagged_na()` values can do this. Tagged NAs work exactly like regular R missing values except that they store one additional byte of information: a tag, which is usually a letter ("a" to "z") or character number ("0" to "9"). This allows to indicate different missings.

Furthermore, see 'Details' in `get_values`.

Value

The tagged missing values and their associated value labels from `x`, or `NULL` if `x` has no tagged missing values.

Examples

```
library(haven)
x <- labelled(c(1:3, tagged_na("a", "c", "z"), 4:1),
              c("Agreement" = 1, "Disagreement" = 4, "First" = tagged_na("c"),
                "Refused" = tagged_na("a"), "Not home" = tagged_na("z")))
# get current NA values
x
get_na(x)
# which NA has which tag?
get_na(x, as.tag = TRUE)

# replace only the NA, which is tagged as NA(c)
if (require("sjmisc")) {
  replace_na(x, value = 2, tagged.na = "c")
  get_na(replace_na(x, value = 2, tagged.na = "c"))

  # data frame as input
  y <- labelled(c(2:3, 3:1, tagged_na("y"), 4:1),
                c("Agreement" = 1, "Disagreement" = 4, "Why" = tagged_na("y")))
  get_na(data.frame(x, y))
}
```

---

get_values *Retrieve values of labelled variables*

Description

This function retrieves the values associated with value labels from `labelled` vectors. Data is also labelled when imported from SPSS, SAS or STATA via `read_spss`, `read_sas` or `read_stata`.

Usage

```
get_values(x, sort.val = TRUE, drop.na = FALSE)
```

Arguments

x	Variable (vector) with value label attributes; or a data frame or list with such variables.
sort.val	Logical, if TRUE (default), values of associated value labels are sorted.
drop.na	Logical, if TRUE, tagged NA values are excluded from the return value. See 'Examples' and get_na.

Details

labelled vectors are numeric by default (when imported with read-functions like read_spss) and have variable and value labels attributes. The value labels are associated with the values from the labelled vector. This function returns the values associated with the vector's value labels, which may differ from actual values in the vector (e.g. if not all values have a related label).

Value

The values associated with value labels from x, or NULL if x has no label attributes.

See Also

get_labels for getting value labels and get_na to get values for missing values.

Examples

```
data(efc)
str(efc$e42dep)
get_values(efc$e42dep)
get_labels(efc$e42dep)

library(haven)
x <- labelled(c(1:3, tagged_na("a", "c", "z"), 4:1),
              c("Agreement" = 1, "Disagreement" = 4, "First" = tagged_na("c"),
                "Refused" = tagged_na("a"), "Not home" = tagged_na("z")))
# get all values
get_values(x)
# drop NA
get_values(x, drop.na = TRUE)

# data frame as input
y <- labelled(c(2:3, 3:1, tagged_na("y"), 4:1),
              c("Agreement" = 1, "Disagreement" = 4, "Why" = tagged_na("y")))
get_values(data.frame(x, y))
```

is_labelled *Check whether object is of class "labelled"*

Description

This function checks whether x is of class labelled.

Usage

```
is_labelled(x)
```

Arguments

x	An object.

Value

Logical, TRUE if x inherits from class labelled, FALSE otherwise.

label_to_colnames *Use variable labels as column names*

Description

This function sets variable labels as column names, to use "labelled data" also for those functions that cannot cope with labelled data by default.

Usage

```
label_to_colnames(x, ...)
```

Arguments

x	A data frame.
...	Optional, unquoted names of variables that should be selected for further processing. Required, if x is a data frame (and no vector) and only selected variables from x should be processed. You may also use functions like : or tidyselect's select-helpers. See 'Examples'.

Value

x with variable labels as column names. For variables without variable labels, the column name is left unchanged.

Examples

```
data(iris)

iris <- var_labels(
  iris,
  Petal.Length = "Petal length (cm)",
  Petal.Width = "Petal width (cm)"
)

colnames(iris)
plot(iris)

colnames(label_to_colnames(iris))
plot(label_to_colnames(iris))
```

---

read_spss *Import data from other statistical software packages*

Description

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

Usage

```
read_spss(
  path,
  convert.factors = TRUE,
  drop.labels = FALSE,
  tag.na = FALSE,
  encoding = NULL,
  verbose = FALSE,
  atomic.to.fac = convert.factors
)

read_sas(
  path,
  path.cat = NULL,
  convert.factors = TRUE,
  drop.labels = FALSE,
  encoding = NULL,
  verbose = FALSE,
  atomic.to.fac = convert.factors
)

read_stata(
  path,
  convert.factors = TRUE,
```

```
  drop.labels = FALSE,
  encoding = NULL,
  verbose = FALSE,
  atomic.to.fac = convert.factors
)

read_data(
  path,
  convert.factors = TRUE,
  drop.labels = FALSE,
  encoding = NULL,
  verbose = FALSE,
  atomic.to.fac = convert.factors
)
```

Arguments

path	File path to the data file.
convert.factors	Logical, if TRUE, categorical variables imported from the dataset (which are imported as atomic) will be converted to factors. Variables are considered as categorical if they have at least the same number of value labels as unique values. This prevents that ranges of continuous variables, where - for instance - the minimum and maximum values are labelled only, will also be converted to factors.
drop.labels	Logical, if TRUE, unused value labels are removed. See drop_labels.
tag.na	Logical, if TRUE, missing values are imported as tagged_na values; else, missing values are converted to regular NA (default behaviour).
encoding	The character encoding used for the file. This defaults to the encoding specified in the file, or UTF-8. Use this argument to override the default encoding stored in the file.
verbose	Logical, if TRUE, a progress bar is displayed that indicates the progress of converting the imported data.
atomic.to.fac	Deprecated, please use ‘convert.factors‘ instead.
path.cat	Optional, the file path to the SAS catalog file.

Details

These read-functions behave slightly differently from **haven**'s read-functions:

- The vectors in the returned data frame are of class atomic, not of class labelled. The labelled-class might cause issues with other packages.
- When importing SPSS data, variables with user defined missings *won't* be read into labelled_spss objects, but imported as *tagged NA values*.

The convert.factors option only converts those variables into factors that are of class atomic and which have value labels after import. Atomic vectors without value labels are considered as continuous and not converted to factors.

Value

A data frame containing the imported, labelled data. Retrieve value labels with `get_labels` and variable labels with `get_label`.

Note

These are wrapper functions for **haven**'s `read_*`-functions.

See Also

Vignette Labelled Data and the sjlabelled-Package.

Examples

```
## Not run:
# import SPSS data set. uses haven's read function
mydat <- read_spss("my_spss_data.sav")

# use haven's read function, convert atomic to factor
mydat <- read_spss("my_spss_data.sav", convert.factors = TRUE)

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

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

`remove_all_labels` *Remove value and variable labels from vector or data frame*

Description

This function removes value and variable label attributes from a vector or data frame. These attributes are typically added to variables when importing foreign data (see `read_spss`) or manually adding label attributes with `set_labels`.

Usage

```
remove_all_labels(x)
```

Arguments

`x`	Vector or `data.frame` with variable and/or value label attributes

Value

`x` with removed value and variable label attributes.

See Also

See vignette Labelled Data and the sjlabelled-Package, and `copy_labels` for adding label attributes (subsetted) data frames.

Examples

```
data(efc)
str(efc)
str(remove_all_labels(efc))
```

`remove_label` *Remove variable labels from variables*

Description

Remove variable labels from variables.

Usage

```
remove_label(x, ...)
```

Arguments

x	A vector or data frame.
...	Optional, unquoted names of variables that should be selected for further processing. Required, if x is a data frame (and no vector) and only selected variables from x should be processed. You may also use functions like : or tidyselect's select-helpers. See 'Examples'.

Value

x with removed variable labels

See Also

`set_label` to manually set variable labels or `get_label` to get variable labels; `set_labels` to add value labels, replacing the existing ones (and removing non-specified value labels).

Examples

```
data(efc)
x <- efc[, 1:5]
get_label(x)
str(x)

x <- remove_label(x)
get_label(x)
str(x)
```

set_label *Add variable label(s) to variables*

Description

This function adds variable labels as attribute (named "label") to the variable x, resp. to a set of variables in a data frame or a list-object. var_labels() is intended for use within pipe-workflows and has a tidyverse-consistent syntax, including support for quasi-quotation (see 'Examples').

Usage

```
set_label(x, label)

set_label(x) <- value

var_labels(x, ...)
```

Arguments

x	Variable (vector), list of variables or a data frame where variables labels should be added as attribute. For var_labels(), x must be a data frame only.
label	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 label = "" to remove labels-attribute from x, resp. set any value of vector label to "" to remove specific variable label attributes from a data frame's variable.
value	See label.
...	Pairs of named vectors, where the name equals the variable name, which should be labelled, and the value is the new variable label.

Value

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

See Also

See vignette Labelled Data and the sjlabelled-Package for more details; set_labels to manually set value labels or get_label to get variable labels.

Examples

```
# manually set value and variable labels
dummy <- sample(1:4, 40, replace = TRUE)
dummy <- set_labels(dummy, labels = c("very low", "low", "mid", "hi"))
dummy <- set_label(dummy, label = "Dummy-variable")
```

```
# or use:
# set_label(dummy) <- "Dummy-variable"

# auto-detection of value labels by default, auto-detection of
# variable labels if argument "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_label(dummy, c("Variable A", "Variable B", "Variable C"))
str(dummy)

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

# setting same variable labels to multiple vectors

# create a set of dummy variables
dummy1 <- sample(1:4, 40, replace = TRUE)
dummy2 <- sample(1:4, 40, replace = TRUE)
dummy3 <- sample(1:4, 40, replace = TRUE)
# put them in list-object
dummies <- list(dummy1, dummy2, dummy3)
# and set variable labels for all three dummies
dummies <- set_label(dummies, c("First Dummy", "2nd Dummy", "Third dummy"))
# see result...
get_label(dummies)


# use 'var_labels()' to set labels within a pipe-workflow, and
# when you need "tidyverse-consistent" api.
# 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)
)

library(magrittr)
dummy %>%
  var_labels(a = "First variable", c = "third variable") %>%
  get_label()

# with quasi-quotation
library(rlang)
```

```
v1 <- "First variable"
v2 <- "Third variable"
dummy %>%
  var_labels(a = !!v1, c = !!v2) %>%
  get_label()

x1 <- "a"
x2 <- "c"
dummy %>%
  var_labels(!!x1 := !!v1, !!x2 := !!v2) %>%
  get_label()
```

`set_labels` *Add value labels to variables*

Description

This function adds labels as attribute (named `"labels"`) to a variable or vector `x`, resp. to a set of variables in a data frame or a list-object. A use-case is, for instance, the **sjPlot**-package, which supports labelled data and automatically assigns labels to axes or legends in plots or to be used in tables. `val_labels()` is intended for use within pipe-workflows and has a tidyverse-consistent syntax, including support for quasi-quotation (see 'Examples').

Usage

```
set_labels(
  x,
  ...,
  labels,
  force.labels = FALSE,
  force.values = TRUE,
  drop.na = TRUE
)

val_labels(x, ..., force.labels = FALSE, force.values = TRUE, drop.na = TRUE)
```

Arguments

`x`	A vector or data frame.
`...`	For `set_labels()`, Optional, unquoted names of variables that should be selected for further processing. Required, if `x` is a data frame (and no vector) and only selected variables from `x` should be processed. You may also use functions like `:` or tidyselect's select-helpers. For `val_labels()`, pairs of named vectors, where the name equals the variable name, which should be labelled, and the value is the new variable label. `val_labels()` also supports quasi-quotation (see 'Examples').

labels — (Named) character vector of labels that will be added to x as "labels" or "value.labels" attribute.

- if labels is **not** a *named vector*, its length 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 length of labels is intended to differ from length of unique values of x, a warning is given. You can still add missing labels with the force.labels or force.values arguments; see 'Note'.
- if labels **is** a *named vector*, value labels will be set accordingly, even if x has a different length of unique values. See 'Note' and 'Examples'.
- if x is a data frame, labels may also be a list of (named) 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, labels will be applied to each column of x.

Use labels = "" to remove labels-attribute from x.

force.labels — Logical; if TRUE, all labels are added as value label attribute, even if x has less unique values then length of labels or if x has a smaller range then length of labels. See 'Examples'. This parameter will be ignored, if labels is a named vector.

force.values — Logical, if TRUE (default) and labels has less elements than unique values of x, additional values not covered by labels will be added as label as well. See 'Examples'. This parameter will be ignored, if labels is a named vector.

drop.na — Logical, whether existing value labels of tagged NA values (see tagged_na) should be removed (drop.na = TRUE, the default) or preserved (drop.na = FALSE). See get_na for more details on tagged NA values.

Value

x with value label attributes; or with removed label-attributes if labels = "". If x is a data frame, the complete data frame x will be returned, with removed or added to variables specified in ...; if ... is not specified, applies to all variables in the data frame.

Note

- if labels is a named vector, force.labels and force.values will be ignored, and only values defined in labels will be labelled;
- if x has less unique values than labels, redundant labels will be dropped, see force.labels;
- if x has more unique values than labels, only matching values will be labelled, other values remain unlabelled, see force.values;

If you only want to change partial value labels, use add_labels instead. Furthermore, see 'Note' in get_labels.

See Also

See vignette Labelled Data and the sjlabelled-Package for more details; set_label to manually set variable labels or get_label to get variable labels; add_labels to add additional value labels without replacing the existing ones.

Examples

```
if (require("sjmisc")) {
  dummy <- sample(1:4, 40, replace = TRUE)
  frq(dummy)

  dummy <- set_labels(dummy, labels = c("very low", "low", "mid", "hi"))
  frq(dummy)

  # assign labels with named vector
  dummy <- sample(1:4, 40, replace = TRUE)
  dummy <- set_labels(dummy, labels = c("very low" = 1, "very high" = 4))
  frq(dummy)

  # force using all labels, even if not all labels
  # have associated values in vector
  x <- c(2, 2, 3, 3, 2)
  # only two value labels
  x <- set_labels(x, labels = c("1", "2", "3"))
  x
  frq(x)

  # all three value labels
  x <- set_labels(x, labels = c("1", "2", "3"), force.labels = TRUE)
  x
  frq(x)

  # create vector
  x <- c(1, 2, 3, 2, 4, NA)
  # add less labels than values
  x <- set_labels(x, labels = c("yes", "maybe", "no"), force.values = FALSE)
  x
  # add all necessary labels
  x <- set_labels(x, labels = c("yes", "maybe", "no"), force.values = TRUE)
  x

  # set labels and missings
  x <- c(1, 1, 1, 2, 2, -2, 3, 3, 3, 3, 3, 9)
  x <- set_labels(x, labels = c("Refused", "One", "Two", "Three", "Missing"))
  x
  set_na(x, na = c(-2, 9))
}


if (require("haven") && require("sjmisc")) {
  x <- labelled(
    c(1:3, tagged_na("a", "c", "z"), 4:1),
```

```
  c("Agreement" = 1, "Disagreement" = 4, "First" = tagged_na("c"),
    "Refused" = tagged_na("a"), "Not home" = tagged_na("z"))
)
# get current NA values
x
get_na(x)
# lose value labels from tagged NA by default, if not specified
set_labels(x, labels = c("New Three" = 3))
# do not drop na
set_labels(x, labels = c("New Three" = 3), drop.na = FALSE)


# set labels via named vector,
# not using all possible values
data(efc)
get_labels(efc$e42dep)

x <- set_labels(
  efc$e42dep,
  labels = c(`independent` = 1,
             `severe dependency` = 2,
             `missing value` = 9)
  )
get_labels(x, values = "p")
get_labels(x, values = "p", non.labelled = TRUE)

# labels can also be set for tagged NA value
# create numeric vector
x <- c(1, 2, 3, 4)
# set 2 and 3 as missing, which will automatically set as
# tagged NA by 'set_na()'
x <- set_na(x, na = c(2, 3))
x
# set label via named vector just for tagged NA(3)
set_labels(x, labels = c(`New Value` = tagged_na("3")))

# setting same value labels to multiple vectors
dummies <- data.frame(
  dummy1 = sample(1:4, 40, replace = TRUE),
  dummy2 = sample(1:4, 40, replace = TRUE),
  dummy3 = sample(1:4, 40, replace = TRUE)
)

# and set same value labels for two of three variables
test <- set_labels(
  dummies, dummy1, dummy2,
  labels = c("very low", "low", "mid", "hi")
)
# see result...
get_labels(test)
}

# using quasi-quotation
```

```
if (require("rlang") && require("dplyr")) {
  dummies <- data.frame(
    dummy1 = sample(1:4, 40, replace = TRUE),
    dummy2 = sample(1:4, 40, replace = TRUE),
    dummy3 = sample(1:4, 40, replace = TRUE)
  )

  x1 <- "dummy1"
  x2 <- c("so low", "rather low", "mid", "very hi")

  dummies %>%
    val_labels(
      !!x1 := c("really low", "low", "a bit mid", "hi"),
      dummy3 = !!x2
    ) %>%
    get_labels()

  # ... and named vectors to explicitly set value labels
  x2 <- c("so low" = 4, "rather low" = 3, "mid" = 2, "very hi" = 1)
  dummies %>%
    val_labels(
      !!x1 := c("really low" = 1, "low" = 3, "a bit mid" = 2, "hi" = 4),
      dummy3 = !!x2
    ) %>% get_labels(values = "p")
}
```

---

set_na *Replace specific values in vector with NA*

Description

This function replaces specific values of variables with NA.

Usage

```
set_na(x, ..., na, drop.levels = TRUE, as.tag = FALSE)
```

Arguments

x	A vector or data frame.
...	Optional, unquoted names of variables that should be selected for further processing. Required, if x is a data frame (and no vector) and only selected variables from x should be processed. You may also use functions like : or tidyselect's select-helpers. See 'Examples'.
na	Numeric vector with values that should be replaced with NA values, or a character vector if values of factors or character vectors should be replaced. For labelled vectors, may also be the name of a value label. In this case, the associated values for the value labels in each vector will be replaced with NA. na can also be a named vector. If as.tag = FALSE, values will be replaced only in those variables that are indicated by the value names (see 'Examples').

drop.levels — Logical, if TRUE, factor levels of values that have been replaced with NA are dropped. See 'Examples'.

as.tag — Logical, if TRUE, values in x will be replaced by tagged_na, else by usual NA values. Use a named vector to assign the value label to the tagged NA value (see 'Examples').

Details

set_na() converts all values defined in na with a related NA or tagged NA value (see tagged_na()). Tagged NAs work exactly like regular R missing values except that they store one additional byte of information: a tag, which is usually a letter ("a" to "z") or character number ("0" to "9").

Different NA values for different variables

If na is a named vector *and* as.tag = FALSE, the names indicate variable names, and the associated values indicate those values that should be replaced by NA in the related variable. For instance, set_na(x, na = c(v1 = 4, v2 = 3)) would replace all 4 in v1 with NA and all 3 in v2 with NA.

If na is a named list *and* as.tag = FALSE, it is possible to replace different multiple values by NA for different variables separately. For example, set_na(x, na = list(v1 = c(1, 4), v2 = 5:7)) would replace all 1 and 4 in v1 with NA and all 5 to 7 in v2 with NA.

Furthermore, see also 'Details' in get_na.

Value

x, with all values in na being replaced by NA. If x is a data frame, the complete data frame x will be returned, with NA's set for variables specified in ...; if ... is not specified, applies to all variables in the data frame.

Note

Labels from values that are replaced with NA and no longer used will be removed from x, however, other value and variable label attributes are preserved. For more details on labelled data, see vignette Labelled Data and the sjlabelled-Package.

Examples

```
if (require("sjmisc") && require("dplyr") && require("haven")) {
  # 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, na = c(1, 8))
  # show value distribution, including missings
  table(dummy, useNA = "always")

  # add named vector as further missing value
  set_na(dummy, na = c("Refused" = 5), as.tag = TRUE)
```

```
# see different missing types
print_tagged_na(set_na(dummy, na = c("Refused" = 5), as.tag = TRUE))


# 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))
# set value 2 and 4 as missings
dummy %>% set_na(na = c(2, 4)) %>% head()
dummy %>% set_na(na = c(2, 4), as.tag = TRUE) %>% get_na()
dummy %>% set_na(na = c(2, 4), as.tag = TRUE) %>% get_values()

data(efc)
dummy <- data.frame(
  var1 = efc$c82cop1,
  var2 = efc$c83cop2,
  var3 = efc$c84cop3
)
# check original distribution of categories
lapply(dummy, table, useNA = "always")
# set 3 to NA for two variables
lapply(set_na(dummy, var1, var3, na = 3), table, useNA = "always")


# if 'na' is a named vector *and* 'as.tag = FALSE', different NA-values
# can be specified for each variable
set.seed(1)
dummy <- data.frame(
  var1 = sample(1:8, 10, replace = TRUE),
  var2 = sample(1:10, 10, replace = TRUE),
  var3 = sample(1:6, 10, replace = TRUE)
)
dummy

# Replace "3" in var1 with NA, "5" in var2 and "6" in var3
set_na(dummy, na = c(var1 = 3, var2 = 5, var3 = 6))

# if 'na' is a named list *and* 'as.tag = FALSE', for each
# variable different multiple NA-values can be specified
set_na(dummy, na = list(var1 = 1:3, var2 = c(7, 8), var3 = 6))


# drop unused factor levels when being set to NA
x <- factor(c("a", "b", "c"))
x
set_na(x, na = "b", as.tag = TRUE)
set_na(x, na = "b", drop.levels = FALSE, as.tag = TRUE)

# set_na() can also remove a missing by defining the value label
# of the value that should be replaced with NA. This is in particular
# helpful if a certain category should be set as NA, however, this category
```

```
  # is assigned with different values accross variables
  x1 <- sample(1:4, 20, replace = TRUE)
  x2 <- sample(1:7, 20, replace = TRUE)
  x1 <- set_labels(x1, labels = c("Refused" = 3, "No answer" = 4))
  x2 <- set_labels(x2, labels = c("Refused" = 6, "No answer" = 7))

  tmp <- data.frame(x1, x2)
  get_labels(tmp)
  table(tmp, useNA = "always")

  get_labels(set_na(tmp, na = "No answer"))
  table(set_na(tmp, na = "No answer"), useNA = "always")

  # show values
  tmp
  set_na(tmp, na = c("Refused", "No answer"))
}
```

`term_labels` *Retrieve labels of model terms from regression models*

Description

This function retrieves variable labels from model terms. In case of categorical variables, where one variable has multiple dummies, variable name and category value is returned.

Usage

```
term_labels(
  models,
  mark.cat = FALSE,
  case = NULL,
  prefix = c("none", "varname", "label"),
  ...
)

get_term_labels(
  models,
  mark.cat = FALSE,
  case = NULL,
  prefix = c("none", "varname", "label"),
  ...
)

response_labels(models, case = NULL, multi.resp = FALSE, mv = FALSE, ...)

get_dv_labels(models, case = NULL, multi.resp = FALSE, mv = FALSE, ...)
```

Arguments

models	One or more fitted regression models. May also be glm's or mixed models.
mark.cat	Logical, if TRUE, the returned vector has an attribute with logical values, which indicate whether a label indicates the value from a factor category (attribute value is TRUE) or a term's variable labels (attribute value is FALSE).
case	Desired target case. Labels will automatically converted into the specified character case. See to_any_case() for more details on this argument.
prefix	Indicates whether the value labels of categorical variables should be prefixed, e.g. with the variable name or variable label. May be abbreviated. See 'Examples',
...	Further arguments passed down to to_any_case(), like preprocess or postprocess.
mv, multi.resp	Logical, if TRUE and models is a multivariate response model from a brmsfit object, then the labels for each dependent variable (multiple responses) are returned.

Details

Typically, the variable labels from model terms are returned. However, for categorical terms that have estimates for each category, the value labels are returned as well. As the return value is a named vector, you can easily use it with **ggplot2**'s scale_*() functions to annotate plots.

Value

For term_labels(), a (named) character vector with variable labels of all model terms, which can be used, for instance, as axis labels to annotate plots.

For response_labels(), a character vector with variable labels from all dependent variables of models.

Examples

```
# use data set with labelled data
data(efc)

fit <- lm(barthtot ~ c160age + c12hour + c161sex + c172code, data = efc)
term_labels(fit)

# make "education" categorical
if (require("sjmisc")) {
  efc$c172code <- to_factor(efc$c172code)
  fit <- lm(barthtot ~ c160age + c12hour + c161sex + c172code, data = efc)
  term_labels(fit)

  # prefix value of categorical variables with variable name
  term_labels(fit, prefix = "varname")

  # prefix value of categorical variables with value label
  term_labels(fit, prefix = "label")
```

```
  # get label of dv
  response_labels(fit)
}
```

`tidy_labels` *Repair value labels*

Description

Duplicated value labels in variables may cause troubles when saving labelled data, or computing cross tabs (cf. `sjmisc::flat_table()` or `sjPlot::plot_xtab()`). `tidy_labels()` repairs duplicated value labels by suffixing them with the associated value.

Usage

```
tidy_labels(x, ..., sep = "_", remove = FALSE)
```

Arguments

`x`	A vector or data frame.
`...`	Optional, unquoted names of variables that should be selected for further processing. Required, if `x` is a data frame (and no vector) and only selected variables from `x` should be processed. You may also use functions like `:` or tidyselect's select-helpers. See 'Examples'.
`sep`	String that will be used to separate the suffixed value from the old label when creating the new value label.
`remove`	Logical, if `TRUE`, the original, duplicated value label will be replaced by the value (i.e. the value is not the suffix of the value label, but will become the value label itself). The `sep`-argument will be ignored in such cases.

Value

`x`, with "repaired" (unique) value labels for each variable.

Examples

```
if (require("sjmisc")) {
  set.seed(123)
  x <- set_labels(
    sample(1:5, size = 20, replace = TRUE),
    labels = c("low" = 1, ".." = 2, ".." = 3, ".." = 4, "high" = 5)
  )
  frq(x)

  z <- tidy_labels(x)
  frq(z)

  z <- tidy_labels(x, sep = ".")
```

```
  frq(z)

  z <- tidy_labels(x, remove = TRUE)
  frq(z)
}
```

`unlabel` *Convert labelled vectors into normal classes*

Description

This function converts `labelled` class vectors into a generic data format, which means that simply all `labelled` class attributes will be removed, so all vectors / variables will most likely become `atomic`.

Usage

```
unlabel(x, verbose = FALSE)
```

Arguments

`x`	A data frame, which contains `labelled` class vectors or a single vector of class `labelled`.
`verbose`	Logical, if `TRUE`, a progress bar is displayed that indicates the progress of converting the imported data.

Value

A data frame or single vector (depending on `x`) with common object 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.

`write_spss` *Write data to other statistical software packages*

Description

These functions write the content of a data frame to an SPSS, SAS or Stata-file.

Usage

```
write_spss(x, path, drop.na = FALSE, compress = FALSE)

write_stata(x, path, drop.na = FALSE, version = 14)

write_sas(x, path, drop.na = FALSE)
```

Arguments

x	A data frame that should be saved as file.
path	File path of the output file.
drop.na	Logical, if TRUE, tagged NA values with value labels will be converted to regular NA's. Else, tagged NA values will be replaced with their value labels. See 'Examples' and get_na.
compress	Logical, if TRUE and a SPSS-file should be created, saves x in zsav (i.e. compressed SPSS) format.
version	File version to use. Supports versions 8-14.

---

zap_na_tags *Convert tagged NA values into regular NA*

Description

Replaces all tagged_na() values with regular NA.

Usage

```
zap_na_tags(x, ...)
```

Arguments

x	A labelled() vector with tagged_na values, or a data frame with such vectors.
...	Optional, unquoted names of variables that should be selected for further processing. Required, if x is a data frame (and no vector) and only selected variables from x should be processed. You may also use functions like : or tidyselect's select-helpers. See 'Examples'.

Value

x, where all tagged_na values are converted to NA.

Examples

```
if (require("haven")) {
  x <- labelled(
    c(1:3, tagged_na("a", "c", "z"), 4:1),
    c("Agreement" = 1, "Disagreement" = 4, "First" = tagged_na("c"),
      "Refused" = tagged_na("a"), "Not home" = tagged_na("z"))
  )
  # get current NA values
  x
  get_na(x)
  zap_na_tags(x)
  get_na(zap_na_tags(x))

  # also works with non-labelled vector that have tagged NA values
  x <- c(1:5, tagged_na("a"), tagged_na("z"), NA)
  haven::print_tagged_na(x)
  haven::print_tagged_na(zap_na_tags(x))
}
```

Index