Package 'labelled'

May 26, 2019

Type Package

Title Manipulating Labelled Data

Version 2.2.1

Maintainer Joseph Larmarange <joseph@larmarange.net>

Description Work with labelled data imported from 'SPSS' or 'Stata' with 'haven' or 'foreign'. This package provides useful functions to deal with ``haven_labelled'' and ``haven_labelled_spss'' classes introduced by 'haven' package.

License GPL-3

Encoding UTF-8

Imports haven (>= 2.1.0), dplyr, stats

Suggests testthat, knitr, rmarkdown, questionr, snakecase

Enhances memisc

LazyData true

URL http://larmarange.github.io/labelled/

BugReports https://github.com/larmarange/labelled/issues

VignetteBuilder knitr

RoxygenNote 6.1.1

NeedsCompilation no

Author Joseph Larmarange [aut, cre] (<https://orcid.org/0000-0001-7097-700X>), Daniel Ludecke [ctb], Hadley Wickham [ctb], Bojanowski Michal [ctb], Briatte François [ctb]

Repository CRAN

Date/Publication 2019-05-26 15:50:34 UTC

R topics documented:

copy_labels . . . 2
labelled . . . 3
labelled_spss . . . 4
look_for . . . 5
mean.haven_labelled . . . 7
na_values . . . 7
nolabel_to_na . . . 9
recode.haven_labelled . . . 9
remove_attributes . . . 10
remove_labels . . . 11
sort_val_labels . . . 12
tagged_na . . . 13
to_character . . . 14
to_factor . . . 15
to_labelled . . . 16
update_labelled . . . 18
val_labels . . . 19
val_labels_to_na . . . 21
var_label . . . 22

Index **24**

`copy_labels` *Copy variable and value labels and SPSS style missing value*

Description

This function copies variable and value labels (including missing values) from one vector to another or from one data frame to another data frame. For data frame, labels are copied according to variable names, and only if variables are the same type in both data frames.

Usage

```
copy_labels(from, to)

copy_labels_from(to, from)
```

Arguments

`from`	A vector or a data.frame (or tibble) to copy labels from.
`to`	A vector or data.frame (or tibble) to copy labels to.

Details

Some base R functions like `subset` drop variable and value labels attached to a variable. `copy_labels` coud be used to restore these attributes.

`copy_labels_from` is intended to be used with `dplyr` syntax, see examples.

Examples

```
library(dplyr)
df <- tibble(
  id = 1:3,
  happy = factor(c('yes', 'no', 'yes')),
  gender = labelled(c(1, 1, 2), c(female = 1, male = 2))
) %>%
set_variable_labels(
  id = "Individual ID",
  happy = "Are you happy?",
  gender = "Gender of respondent"
)
var_label(df)
fdf <- df %>% filter(id < 3)
var_label(fdf) # some variable labels have been lost
fdf <- fdf %>% copy_labels_from(df)
var_label(fdf)

# Alternative syntax
fdf <- subset(df, id < 3)
fdf <- copy_labels(from = df, to = fdf)
```

`labelled` *Create a labelled vector.*

Description

A labelled vector is a common data structure in other statistical environments, allowing you to assign text labels to specific values.

Usage

```
labelled(x, labels, label = NULL)

is.labelled(x)
```

Arguments

`x`	A vector to label. Must be either numeric (integer or double) or character.
`labels`	A named vector or `NULL`. The vector should be the same type as `x`. Unlike factors, labels don't need to be exhaustive: only a fraction of the values might be labelled.
`label`	A short, human-readable description of the vector.

Value

An object of class "haven_labelled" or "haven_labelled_spss".

See Also

labelled (**haven**)

haven::is.labelled() (**haven**)

Examples

```
s1 <- labelled(c("M", "M", "F"), c(Male = "M", Female = "F"))
s1
str(s1)
s2 <- labelled(c(1, 1, 2), c(Male = 1, Female = 2),
               label="Assigned sex at birth")
s2
str(s2)

# Unfortunately it's not possible to make as.factor() work for labelled objects
# so instead use to_factor(). This works for all types of labelled vectors.
to_factor(s1)
to_factor(s1, levels = "prefixed")
to_factor(s2)

# Other statistical software supports multiple types of missing values
s3 <- labelled_spss(c(1, 1, 2, 2, 8, 9),
               c(Male = 1, Female = 2, Refused = 8, "Not applicable" = 9),
               na_values = c(8, 9)

)
s3
str(s3)
to_factor(s3)
to_factor(s3, user_na_to_na = TRUE)
is.labelled(s1)
```

---

labelled_spss *Create a labelled vector with SPSS style of missing values.*

Description

It is similar to the labelled class but it also models SPSS's user-defined missings, which can be up to three distinct values, or for numeric vectors a range.

Usage

```
labelled_spss(x, labels, na_values = NULL, na_range = NULL,
  label = NULL)
```

Arguments

x	A vector to label. Must be either numeric (integer or double) or character.
labels	A named vector or NULL. The vector should be the same type as x. Unlike factors, labels don't need to be exhaustive: only a fraction of the values might be labelled.
na_values	A vector of values that should also be considered as missing.
na_range	A numeric vector of length two giving the (inclusive) extents of the range. Use -Inf and Inf if you want the range to be open ended.
label	A short, human-readable description of the vector.

See Also

labelled_spss (**haven**)

Examples

```
x1 <- labelled_spss(1:10, c(Good = 1, Bad = 8), na_values = c(9, 10))
x1
is.na(x1)
x2 <- labelled_spss(1:10, c(Good = 1, Bad = 8), na_range = c(9, Inf))
x2
is.na(x2)
```

---

look_for *Look for keywords variable names and descriptions*

Description

look_for emulates the lookfor Stata command in R. It supports searching into the variable names of regular R data frames as well as into variable labels descriptions. The command is meant to help users finding variables in large datasets.

Usage

```
look_for(data, ..., labels = TRUE, ignore.case = TRUE,
  details = FALSE)

lookfor(data, ..., labels = TRUE, ignore.case = TRUE,
  details = FALSE)
```

Arguments

data	a data frame
...	list of keywords, a character string (or several character strings), which can be formatted as a regular expression suitable for a grep pattern, or a vector of keywords; displays all variables if not specified

labels	whether or not to search variable labels (descriptions); TRUE by default
ignore.case	whether or not to make the keywords case sensitive; TRUE by default (case is ignored during matching)
details	add details about each variable (see examples)

Details

The function looks into the variable names for matches to the keywords. If available, variable labels are included in the search scope. Variable labels of data.frame imported with **foreign** or **memisc** packages will also be taken into account (see to_labelled).

look_for and lookfor are equivalent.

Value

a data frame featuring the variable position, name and description (if it exists) in the original data frame

Author(s)

François Briatte <f.briatte@gmail.com>

Source

Based on the behaviour of the lookfor command in Stata.

Examples

```
look_for(iris)
# Look for a single keyword.
look_for(iris, "petal")
look_for(iris, "s")
# Look for with a regular expression
look_for(iris, "petal|species")
look_for(iris, "s$")
# Look for with several keywords
look_for(iris, "pet", "sp")
look_for(iris, "pet", "sp", "width")
# Labelled data
## Not run: require(questionr)
data(fertility)
look_for(women)
look_for(women, "date")
# Display details
look_for(women, details = TRUE)

## End(Not run)
```

mean.haven_labelled *Arithmetic mean for numeric labelled vectors*

Description

Arithmetic mean for numeric labelled vectors

Usage

```
## S3 method for class 'haven_labelled'
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x	a numeric labelled vector.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x . before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
...	additional arguments to be passed to or from methods.

See Also

mean

na_values *Get / Set SPSS missing values*

Description

Get / Set SPSS missing values

Usage

```
na_values(x)

na_values(x) <- value

na_range(x)

na_range(x) <- value

set_na_values(.data, ...)
```

```
set_na_range(.data, ...)

user_na_to_na(x)
```

Arguments

x	A vector.
value	A vector of values that should also be considered as missing (for na_values) or a numeric vector of length two giving the (inclusive) extents of the range (for na_values, use -Inf and Inf if you want the range to be open ended).
.data	a data frame
...	name-value pairs of missing values (see examples)

Details

See labelled_spss for a presentation of SPSS's user defined missing values. Note that is.na will return TRUE for user defined misssing values. You can use user_na_to_na to convert user defined missing values to NA.

Value

na_values will return a vector of values that should also be considered as missing. na_range will return a numeric vector of length two giving the (inclusive) extents of the range.

set_na_values and set_na_range will return an updated copy of .data.

Note

set_na_values and set_na_range could be used with dplyr.

See Also

labelled_spss, user_na_to_na

Examples

```
v <- labelled(c(1,2,2,2,3,9,1,3,2,NA), c(yes = 1, no = 3, "don't know" = 9))
v
na_values(v) <- 9
na_values(v)
v
is.na(v)
user_na_to_na(v)
na_values(v) <- NULL
v
na_range(v) <- c(5, Inf)
na_range(v)
v
user_na_to_na(v)
if (require(dplyr)) {
  # setting value labels
```

```
  df <- data_frame(s1 = c("M", "M", "F", "F"), s2 = c(1, 1, 2, 9)) %>%
    set_value_labels(s2 = c(yes = 1, no = 2)) %>%
    set_na_values(s2 = 9)
  na_values(df)

  # removing missing values
  df <- df %>% set_na_values(s2 = NULL)
  df$s2
}
```

`nolabel_to_na` *Recode values with no label to NA*

Description

For labelled variables, values with no label will be recoded to `NA`.

Usage

```
nolabel_to_na(x)
```

Arguments

`x`	Object to recode.

Examples

```
v <- labelled(c(1, 2, 9, 1, 9), c(yes = 1, no = 2))
nolabel_to_na(v)
```

`recode.haven_labelled` *Recode values*

Description

Extend `recode` method from **haven** to works with labelled vectors.

Usage

```
## S3 method for class 'haven_labelled'
recode(.x, ..., .default = NULL,
  .missing = NULL)
```

Arguments

`.x`	A vector to modify
`...`	Replacements. For character and factor `.x`, these should be named and replacement is based only on their name. For numeric `.x`, these can be named or not. If not named, the replacement is done based on position i.e. `.x` represents positions to look for in replacements. See examples. When named, the argument names should be the current values to be replaced, and the argument values should be the new (replacement) values. All replacements must be the same type, and must have either length one or the same length as `.x`. These dots support tidy dots features.
`.default`	If supplied, all values not otherwise matched will be given this value. If not supplied and if the replacements are the same type as the original values in `.x`, unmatched values are not changed. If not supplied and if the replacements are not compatible, unmatched values are replaced with `NA`. `.default` must be either length 1 or the same length as `.x`.
`.missing`	If supplied, any missing values in `.x` will be replaced by this value. Must be either length 1 or the same length as `.x`.

See Also

`recode` (**dplyr**)

Examples

```
x <- labelled(1:3, c(yes = 1, no = 2))
x
dplyr::recode(x, `3` = 2L)
```

`remove_attributes` *Remove attributes*

Description

This function removes specified attributes. When applied to a data.frame, it will also remove recursively the specified attributes to each column of the data.frame.

Usage

```
remove_attributes(x, attributes)
```

Arguments

`x`	an object
`attributes`	a character vector indicating attributes to remve

Examples

```
## Not run:
library(haven)
path <- system.file("examples", "iris.sav", package = "haven")
d <- read_sav(path)
str(d)
d <- remove_attributes(d, "format.spss")
str(d)
## End(Not run)
```

`remove_labels` *Remove variable label, value labels and user defined missing values*

Description

Use `remove_var_label` to remove variable label, `remove_val_labels` to remove value labels, `remove_user_na` to remove user defined missing values (na_values and na_range) and `remove_labels` to remove all.

Usage

```
remove_labels(x, user_na_to_na = FALSE)

remove_var_label(x)

remove_val_labels(x)

remove_user_na(x, user_na_to_na = FALSE)
```

Arguments

`x`	A vector or a data frame.
`user_na_to_na`	Convert user defined missing values into `NA`?

Details

Be careful with `remove_user_na` and `remove_labels`, user defined missing values will not be automatically converted to `NA`, except if you specify `user_na_to_na = TRUE`. `user_na_to_na(x)` is an equivalent of `remove_user_na(x, user_na_to_na = TRUE)`.

Examples

```
x1 <- labelled_spss(1:10, c(Good = 1, Bad = 8), na_values = c(9, 10))
var_label(x1) <- "A variable"
x1

x2 <- remove_labels(x1)
x2
```

```
x3 <- remove_labels(x1, user_na_to_na = TRUE)
x3
x4 <- remove_user_na(x1, user_na_to_na = TRUE)
x4
```

`sort_val_labels` *Sort value labels*

Description

Sort value labels according to values or to labels

Usage

```
sort_val_labels(x, according_to = c("values", "labels"),
  decreasing = FALSE)

## S3 method for class 'haven_labelled'
sort_val_labels(x, according_to = c("values",
  "labels"), decreasing = FALSE)

## S3 method for class 'data.frame'
sort_val_labels(x, according_to = c("values",
  "labels"), decreasing = FALSE)
```

Arguments

`x`	A labelled vector.
`according_to`	According to values or to labels?
`decreasing`	In decreasing order?

Examples

```
v <- labelled(c(1, 2, 3), c(maybe = 2, yes = 1, no = 3))
v
sort_val_labels(v)
sort_val_labels(v, decreasing = TRUE)
sort_val_labels(v, 'l')
sort_val_labels(v, 'l', TRUE)
```

tagged_na *"Tagged" missing values*

Description

"Tagged" missing values 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"). When by loading a SAS and Stata file, the tagged missing values always use lower case values.

Usage

```
tagged_na(...)

na_tag(x)

is_tagged_na(x, tag = NULL)

format_tagged_na(x, digits = getOption("digits"))

print_tagged_na(x, digits = getOption("digits"))
```

Arguments

...	Vectors containing single character. The letter will be used to "tag" the missing value.
x	A numeric vector
tag	If NULL, will only return true if the tag has this value.
digits	Number of digits to use in string representation

Details

‘format_tagged_na()‘ and ‘print_tagged_na()‘ format tagged NA’s as NA(a), NA(b), etc.

Examples

```
x <- c(1:5, tagged_na("a"), tagged_na("z"), NA)

# Tagged NA's work identically to regular NAs
x
is.na(x)

# To see that they're special, you need to use na_tag(),
# is_tagged_na(), or print_tagged_na():
is_tagged_na(x)
na_tag(x)
print_tagged_na(x)
```

```
# You can test for specific tagged NAs with the second argument
is_tagged_na(x, "a")

# Because the support for tagged's NAs is somewhat tagged on to R,
# the left-most NA will tend to be preserved in arithmetic operations.
na_tag(tagged_na("a") + tagged_na("z"))
```

`to_character` *Convert input to a character vector*

Description

By default, `to_character` is a wrapper for `base::as.character()`. For labelled vector, to_character allows to specify if value, labels or labels prefixed with values should be used for conversion.

Usage

```
to_character(x, ...)

## S3 method for class 'haven_labelled'
to_character(x, levels = c("labels", "values",
  "prefixed"), nolabel_to_na = FALSE, ...)
```

Arguments

`x`	Object to coerce to a character vector.
`...`	Other arguments passed down to method.
`levels`	What should be used for the factor levels: the labels, the values or labels prefixed with values?
`nolabel_to_na`	Should values with no label be converted to 'NA'?

Details

If some values doesn't have a label, automatic labels will be created, except if `nolabel_to_na` is `TRUE`.

Examples

```
v <- labelled(c(1,2,2,2,3,9,1,3,2,NA), c(yes = 1, no = 3, "don't know" = 9))
to_character(v)
to_character(v, missing_to_na = FALSE, nolabel_to_na = TRUE)
to_character(v, "v")
to_character(v, "p")
```

to_factor *Convert input to a factor.*

Description

The base function base::as.factor() is not a generic, but this variant is. By default, to_factor is a wrapper for base::as.factor(). Please note that to_factor differs slightly from as_factor method provided by haven package.

Usage

```
to_factor(x, ...)

## S3 method for class 'haven_labelled'
to_factor(x, levels = c("labels", "values",
  "prefixed"), ordered = FALSE, nolabel_to_na = FALSE,
  sort_levels = c("auto", "none", "labels", "values"),
  decreasing = FALSE, drop_unused_labels = FALSE,
  user_na_to_na = FALSE, strict = FALSE, unclass = FALSE, ...)

## S3 method for class 'data.frame'
to_factor(x, levels = c("labels", "values",
  "prefixed"), ordered = FALSE, nolabel_to_na = FALSE,
  sort_levels = c("auto", "none", "labels", "values"),
  decreasing = FALSE, labelled_only = TRUE,
  drop_unused_labels = FALSE, strict = FALSE, unclass = FALSE, ...)
```

Arguments

x	Object to coerce to a factor.
...	Other arguments passed down to method.
levels	What should be used for the factor levels: the labels, the values or labels prefixed with values?
ordered	TRUE for ordinal factors, FALSE (default) for nominal factors.
nolabel_to_na	Should values with no label be converted to 'NA'?
sort_levels	How the factor levels should be sorted? (see Details)
decreasing	Should levels be sorted in decreasing order?
drop_unused_labels	Should unused value labels be dropped? (applied only if strict = FALSE)
user_na_to_na	Convert user defined missing values into NA?
strict	Convert to factor only if all values have a defined label?
unclass	If not converted to a factor (when strict = TRUE), convert to a character or a numeric factor?
labelled_only	for a data.frame, convert only labelled variables to factors?

Details

If some values doesn't have a label, automatic labels will be created, except if `nolabel_to_na` is `TRUE`.

If `sort_levels == 'values'`, the levels will be sorted according to the values of `x`. If `sort_levels == 'labels'`, the levels will be sorted according to labels' names. If `sort_levels == 'none'`, the levels will be in the order the value labels are defined in `x`. If some labels are automatically created, they will be added at the end. If `sort_levels == 'auto'`, `sort_levels == 'none'` will be used, except if some values doesn't have a defined label. In such case, `sort_levels == 'values'` will be applied.

When applied to a data.frame, only labelled vectors are converted by default to a factor. Use `labelled_only = FALSE` to convert all variables to factors.

Examples

```
v <- labelled(c(1,2,2,2,3,9,1,3,2,NA), c(yes = 1, no = 3, "don't know" = 9))
to_factor(v)
to_factor(v, nolabel_to_na = TRUE)
to_factor(v, 'p')
to_factor(v, sort_levels = 'v')
to_factor(v, sort_levels = 'n')
to_factor(v, sort_levels = 'l')

x <- labelled(c('H', 'M', 'H', 'L'), c(low = 'L', medium = 'M', high = 'H'))
to_factor(x, ordered = TRUE)

# Strict conversion
v <- labelled(c(1, 1, 2, 3), labels = c(No = 1, Yes = 2))
to_factor(v)
to_factor(v, strict = TRUE) # Not converted because 3 does not have a label
to_factor(v, strict = TRUE, unclass = TRUE)
```

---

to_labelled *Convert to labelled data*

Description

Convert a factor or data imported with **foreign** or **memisc** to labelled data.

Usage

```
to_labelled(x, ...)

## S3 method for class 'data.frame'
to_labelled(x, ...)

## S3 method for class 'list'
to_labelled(x, ...)
```

```
## S3 method for class 'data.set'
to_labelled(x, ...)

## S3 method for class 'importer'
to_labelled(x, ...)

foreign_to_labelled(x)

memisc_to_labelled(x)

## S3 method for class 'factor'
to_labelled(x, labels = NULL, ...)
```

Arguments

x	Factor or dataset to convert to labelled data frame
...	Not used
labels	When converting a factor only: an optional named vector indicating how factor levels should be coded. If a factor level is not found in labels, it will be converted to NA.

Details

to_labelled is a general wrapper calling the appropriate sub-functions.

memisc_to_labelled converts a data.set object created with **memisc** package to a labelled data frame.

foreign_to_labelled converts data imported with read.spss or read.dta from **foreign** package to a labelled data frame, i.e. using labelled class. Factors will not be converted. Therefore, you should use use.value.labels = FALSE when importing with read.spss or convert.factors = FALSE when importing with read.dta.

To convert correctly defined missing values imported with read.spss, you should have used to.data.frame = FALSE and use.missings = FALSE. If you used the option to.data.frame = TRUE, meta data describing missing values will not be attached to the import. If you used use.missings = TRUE, missing values would have been converted to NA.

So far, missing values defined in Stata are always imported as NA by read.dta and could not be retrieved by foreign_to_labelled.

Value

A tbl data frame or a labelled vector.

See Also

labelled (**foreign**), read.spss (**foreign**), read.dta (**foreign**), data.set (**memisc**), importer (**memisc**), to_factor.

Examples

```
## Not run:
  # from foreign
  library(foreign)
  sav <- system.file("files", "electric.sav", package = "foreign")
  df <- to_labelled(read.spss(
    sav,
    to.data.frame = FALSE,
    use.value.labels = FALSE,
    use.missings = FALSE
 ))

 # from memisc
 library(memisc)
 nes1948.por <- UnZip('anes/NES1948.ZIP', 'NES1948.POR', package='memisc')
 nes1948 <- spss.portable.file(nes1948.por)
 df <- to_labelled(nes1948)
 ds <- as.data.set(nes19480)
 df <- to_labelled(ds)

## End(Not run)

# Converting factors to labelled vectors
f <- factor(c("yes", "yes", "no", "no", "don't know", "no", "yes", "don't know"))
to_labelled(f)
to_labelled(f, c("yes" = 1, "no" = 2, "don't know" = 9))
to_labelled(f, c("yes" = 1, "no" = 2))
to_labelled(f, c("yes" = "Y", "no" = "N", "don't know" = "DK"))

s1 <- labelled(c('M', 'M', 'F'), c(Male = 'M', Female = 'F'))
labels <- val_labels(s1)
f1 <- to_factor(s1)
f1

to_labelled(f1)
identical(s1, to_labelled(f1))
to_labelled(f1, labels)
identical(s1, to_labelled(f1, labels))
```

`update_labelled` *Update labelled data to version 2.0.0*

Description

Labelled data imported with haven version 1.1.2 or before or created with `labelled` version 1.1.0 or before was using "labelled" and "labelled_spss" classes.

Usage

```
update_labelled(x)

## S3 method for class 'labelled'
update_labelled(x)

## S3 method for class 'data.frame'
update_labelled(x)
```

Arguments

x	An object (vector or data.frame) to convert.

Details

Since version 2.0.0 of these two packages, "haven_labelled" and "haven_labelled_spss" are used instead.

`update_labelled` convert labelled vectors from the old to the new classes.

`val_labels` *Get / Set value labels*

Description

Get / Set value labels

Usage

```
val_labels(x, prefixed = FALSE)

## Default S3 method:
val_labels(x, prefixed = FALSE)

## S3 method for class 'haven_labelled'
val_labels(x, prefixed = FALSE)

## S3 method for class 'data.frame'
val_labels(x, prefixed = FALSE)

val_labels(x) <- value

## S3 replacement method for class 'numeric'
val_labels(x) <- value

## S3 replacement method for class 'character'
val_labels(x) <- value
```

```
## S3 replacement method for class 'haven_labelled'
val_labels(x) <- value

## S3 replacement method for class 'haven_labelled_spss'
val_labels(x) <- value

## S3 replacement method for class 'data.frame'
val_labels(x) <- value

val_label(x, v, prefixed = FALSE)

## S3 method for class 'haven_labelled'
val_label(x, v, prefixed = FALSE)

## S3 method for class 'data.frame'
val_label(x, v, prefixed = FALSE)

val_label(x, v) <- value

## S3 replacement method for class 'haven_labelled'
val_label(x, v) <- value

## S3 replacement method for class 'numeric'
val_label(x, v) <- value

## S3 replacement method for class 'character'
val_label(x, v) <- value

## S3 replacement method for class 'data.frame'
val_label(x, v) <- value

set_value_labels(.data, ...)

add_value_labels(.data, ...)

remove_value_labels(.data, ...)
```

Arguments

x	A vector.
prefixed	Should labels be prefixed with values?
value	A named vector for `val_labels` (see `labelled`) or a character string for `val_labels`. `NULL` to remove the labels. For data frames, it could also be a named list.
v	A single value.
.data	a data frame
...	name-value pairs of value labels (see examples)

Value

val_labels will return a named vector. val_label will return a single character string.

set_value_labels, add_value_labels and remove_value_labels will return an updated copy of .data.

Note

set_value_labels, add_value_labels and remove_value_labels could be used with dplyr. While set_value_labels will replace the list of value labels, add_value_labels and remove_value_labels will update that list (see examples).

Examples

```
v <- labelled(c(1,2,2,2,3,9,1,3,2,NA), c(yes = 1, no = 3, "don't know" = 9))
val_labels(v)
val_labels(v, prefixed = TRUE)
val_label(v, 2)
val_label(v, 2) <- 'maybe'
val_label(v, 9) <- NULL
val_labels(v) <- NULL
if (require(dplyr)) {
  # setting value labels
  df <- data_frame(s1 = c("M", "M", "F"), s2 = c(1, 1, 2)) %>%
    set_value_labels(s1 = c(Male = "M", Female = "F"), s2 = c(Yes = 1, No = 2))
  val_labels(df)

  # updating value labels
  df <- df %>% add_value_labels(s2 = c(Unknown = 9))
  df$s2

  # removing a value labels
  df <- df %>% remove_value_labels(s2 = 9)
  df$s2

  # removing all value labels
  df <- df %>% set_value_labels(s2 = NULL)
  df$s2
}
```

---

val_labels_to_na *Recode value labels to NA*

Description

For labelled variables, values with a label will be recoded to NA.

Usage

```
val_labels_to_na(x)
```

Arguments

x	Object to recode.
...	Other arguments passed down to method.

See Also

zap_labels

Examples

```
v <- labelled(c(1, 2, 9, 1, 9), c(dk = 9))
val_labels_to_na(v)
```

---

var_label *Get / Set a variable label*

Description

Get / Set a variable label

Usage

```
var_label(x, unlist = FALSE)

var_label(x) <- value

set_variable_labels(.data, ..., .labels = NA)
```

Arguments

x	an object
unlist	for data frames, return a named vector instead of a list
value	a character string or NULL to remove the label For data frames, it could also be a named list or a character vector of same length as the number of columns in x.
.data	a data frame
...	name-value pairs of variable labels (see examples)
.labels	variable labels to be applied to the data.frame, using the same syntax as value in 'var_label(df) <- value'.

Details

For data frames, if value is a named list, only elements whose name will match a column of the data frame will be taken into account. If value is a character vector, labels should in the same order as the columns of the data.frame.

Value

`set_variable_labels` will return an updated copy of `.data`.

Note

`set_variable_labels` could be used with `dplyr`.

Examples

```
var_label(iris$Sepal.Length)
var_label(iris$Sepal.Length) <- 'Length of the sepal'
## Not run:
 View(iris)

## End(Not run)
# To remove a variable label
var_label(iris$Sepal.Length) <- NULL
# To change several variable labels at once
var_label(iris) <- c(
  "sepal length", "sepal width", "petal length",
  "petal width", "species"
)
var_label(iris)
var_label(iris) <- list(
  Petal.Width = "width of the petal",
  Petal.Length = "length of the petal"
)
var_label(iris)
var_label(iris, unlist = TRUE)
if (require(dplyr)) {
  # adding some variable labels
  df <- data_frame(s1 = c("M", "M", "F"), s2 = c(1, 1, 2)) %>%
    set_variable_labels(s1 = "Sex", s2 = "Yes or No?")
  var_label(df)

  # removing a variable label
  df <- df %>% set_variable_labels(s2 = NULL)
  var_label(df$s2)

  # defining variable labels derived from variable names
  if (require(snakecase)) {
    iris <- iris %>%
      set_variable_labels(.labels = to_sentence_case(names(iris)))
    var_label(iris)
  }
}
```

Index

add_value_labels *(*val_labels*)*, 19
as_factor, *15*

base::as.character(), *14*
base::as.factor(), *15*

copy_labels, 2
copy_labels_from *(*copy_labels*)*, 2

data.set, *17*

foreign_to_labelled *(*to_labelled*)*, 16
format_tagged_na *(*tagged_na*)*, 13

haven::is.labelled(), *4*

importer, *17*
is.labelled *(*labelled*)*, 3
is_tagged_na *(*tagged_na*)*, 13

labelled, 3, *4*, *17*, *20*
labelled_spss, 4, *5*, *8*
look_for, 5
lookfor *(*look_for*)*, 5

mean, *7*
mean.haven_labelled, 7
memisc_to_labelled *(*to_labelled*)*, 16

na_range *(*na_values*)*, 7
na_range<- *(*na_values*)*, 7
na_tag *(*tagged_na*)*, 13
na_values, 7
na_values<- *(*na_values*)*, 7
nolabel_to_na, 9

print_tagged_na *(*tagged_na*)*, 13

read.dta, *17*
read.spss, *17*
recode, *9, 10*
recode.haven_labelled, 9

remove_attributes, 10
remove_labels, 11
remove_user_na *(*remove_labels*)*, 11
remove_val_labels *(*remove_labels*)*, 11
remove_value_labels *(*val_labels*)*, 19
remove_var_label *(*remove_labels*)*, 11

set_na_range *(*na_values*)*, 7
set_na_values *(*na_values*)*, 7
set_value_labels *(*val_labels*)*, 19
set_variable_labels *(*var_label*)*, 22
sort_val_labels, 12
subset, *2*

tagged_na, 13
tidy dots, *10*
to_character, 14
to_factor, 15, *17*
to_labelled, *6*, 16

update_labelled, 18
user_na_to_na, *8*
user_na_to_na *(*na_values*)*, 7

val_label *(*val_labels*)*, 19
val_label<- *(*val_labels*)*, 19
val_labels, 19, *20*
val_labels<- *(*val_labels*)*, 19
val_labels_to_na, 21
var_label, 22
var_label<- *(*var_label*)*, 22

zap_labels, *22*