Package ‘labelled’

May 8, 2026

Type Package

Title Manipulating Labelled Data

Version 2.16.0

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

Depends R (>= 3.2)

Imports haven (>= 2.4.1), cli, dplyr (>= 1.1.0), lifecycle, rlang (>= 1.1.0), vctrs, stringr, tidyr (>= 1.3.0), tidyselect

Suggests gt, knitr, rmarkdown, purrr, questionr, snakecase, spelling, survey, testthat (>= 3.2.0)

Enhances memisc

URL https://larmarange.github.io/labelled/, https://github.com/larmarange/labelled

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

VignetteBuilder knitr

LazyData true

RoxygenNote 7.3.3

Language en-US

Config/testthat/edition 3

Config/Needs/check memisc

NeedsCompilation no

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

Repository CRAN

Date/Publication 2025-10-22 17:00:02 UTC

Contents

copy_labels . . . 2
dictionary_to_variable_labels . . . 4
drop_unused_value_labels . . . 6
is_prefixed . . . 6
look_for . . . 7
names_prefixed_by_values . . . 10
na_values . . . 11
nolabel_to_na . . . 14
recode.haven_labelled . . . 14
recode_if . . . 16
remove_attributes . . . 17
remove_labels . . . 18
sort_val_labels . . . 19
tagged_na_to_user_na . . . 20
to_character . . . 21
to_factor . . . 23
to_labelled . . . 26
unique_tagged_na . . . 28
update_labelled . . . 29
update_variable_labels_with . . . 30
val_labels . . . 31
val_labels_to_na . . . 34
var_label . . . 35
x_haven_2.0 . . . 38

Index **39**

`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, .strict = TRUE)

copy_labels_from(to, from, .strict = TRUE)
```

Arguments

`from`	A vector or a data.frame (or survey design) to copy labels from.
`to`	A vector or data.frame (or survey design) to copy labels to.
`.strict`	When `from` is a labelled vector, `to` have to be of the same type (numeric or character) in order to copy value labels and SPSS-style missing values. If this is not the case and `.strict = TRUE`, an error will be produced. If `.strict = FALSE`, only variable label will be copied.

Details

Some base R functions like `base::subset()` drop variable and value labels attached to a variable. `copy_labels` could 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)
```

dictionary_to_variable_labels

Convert a dictionary data frame into a list of variable / value labels

Description

These helpers could be used to generate, from a data frame corresponding to a variable dictionary, a list of variable or value labels that could be passed to `set_variable_labels()` or to `set_value_labels()`.

Usage

```
dictionary_to_variable_labels(dictionary, names_from = 1, labels_from = 2)

dictionary_to_value_labels(
  dictionary,
  names_from = 1,
  values_from = 2,
  labels_from = 3,
  delim_entries = NULL,
  delim_value_label = NULL,
  data = NULL
)
```

Arguments

`dictionary`	A data frame or a tibble containing the definitions of the variable / value labels.
`names_from`	<`tidy-select`> Column containing the names of the variables.
`labels_from`	<`tidy-select`> Column containing the labels (unused if `delim_value_label` is provided).
`values_from`	<`tidy-select`> Column containing the values or the pairs of values and labels (see examples).
`delim_entries`	Optional string giving the delimiter between several pairs of value and labels if provided in a unique row. `NULL` if each pair is defined in a separate row (see examples).
`delim_value_label`	Optional string giving the delimiter between the value and the label.
`data`	A data frame or a tibble containing the raw data to identify the type (integer, numeric, character) of each variable.

Details

It should be noted that value labels could be

Examples

```
dic <- dplyr::tibble(
  variable = c("mpg", "vs", "am", "cyl"),
  label = c("miles / gallon", "Engine", "Transmission", NA),
  values = c(NA, "0:V-Shaped,1:straight", "0:automatic,1:manual", NA)
)
dic

l <- dic %>% dictionary_to_variable_labels()
l
mtcars %>%
  set_variable_labels(.labels = l) %>%
  look_for()


vl <- dic %>%
  dictionary_to_value_labels(
    values_from = values,
    delim_entries = ",",
    delim_value_label = ":",
    data = mtcars
  )
mtcars %>%
  set_value_labels(.labels = vl) %>%
  look_for()

dic2 <- dplyr::tibble(
  variable = c("am", "am", "vs", "vs"),
  labels = c("0:automatic", "1:manual", "0:V-shaped", "1:straight")
)
dic2 %>%
  dictionary_to_value_labels(
    delim_value_label = ":",
    data = mtcars
  )

dic3 <- dplyr::tibble(
  code = c(0, 1, 0, 1),
  label = c("automatic", "manual", "V-shaped", "straight"),
  var = c("am", "am", "vs", "vs")
)
dic3 %>%
  dictionary_to_value_labels(
    names_from = var,
    values_from = code,
    labels_from = label,
    data = mtcars
  )
```

drop_unused_value_labels *Drop unused value labels*

Description

Drop value labels associated to a value not present in the data.

Usage

```
drop_unused_value_labels(x)
```

Arguments

x	A vector, a data frame or a survey design.

Examples

```
x <- labelled(c(1, 2, 2, 1), c(yes = 1, no = 2, maybe = 3))
x
drop_unused_value_labels(x)
```

is_prefixed *Check if a factor is prefixed*

Description

Check if a factor is prefixed

Usage

```
is_prefixed(x)
```

Arguments

x	a factor

look_for *Look for keywords variable names and descriptions / Create a data dictionary*

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, factor levels and value labels. The command is meant to help users finding variables in large datasets.

Usage

```
look_for(
  data,
  ...,
  labels = TRUE,
  values = TRUE,
  ignore.case = TRUE,
  details = c("basic", "none", "full")
)

lookfor(
  data,
  ...,
  labels = TRUE,
  values = TRUE,
  ignore.case = TRUE,
  details = c("basic", "none", "full")
)

generate_dictionary(
  data,
  ...,
  labels = TRUE,
  values = TRUE,
  ignore.case = TRUE,
  details = c("basic", "none", "full")
)

## S3 method for class 'look_for'
print(x, ...)

look_for_and_select(
  data,
  ...,
  labels = TRUE,
  values = TRUE,
```

```
  ignore.case = TRUE
)

convert_list_columns_to_character(x, sep_value_labels = "[]", sep_other = "; ")

lookfor_to_long_format(x, sep_value_labels = "[]")

to_gt(
  x,
 column_labels = list(pos = "#", variable = "Variable", col_type = "Type", label =
  "Variable label", values = "Values", missing = "Missing values", unique_values =
    "Unique values", na_values = "User-defined missings (values)", na_range =
    "User-defined missings (range)")
)
```

Arguments

data	a data frame or a survey object
...	optional list of keywords, a character string (or several character strings), which can be formatted as a regular expression suitable for a `base::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
values	whether or not to search within values (factor levels or value labels); `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 (full details could be time consuming for big data frames, `FALSE` is equivalent to `"none"` and `TRUE` to `"full"`)
x	a tibble returned by `look_for()`
sep_value_labels	(string) for value labels, separator between value and name
sep_other	separator for other list columns
column_labels	Optional column labels

Details

When no keyword is provided, it will produce a data dictionary of the overall data frame.

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()`). If no keyword is provided, it will return all variables of `data`.

`look_for()`, `lookfor()` and `generate_dictionary()` are equivalent.

By default, results will be summarized when printing. To deactivate default printing, use `dplyr::as_tibble()`.

`lookfor_to_long_format()` could be used to transform results with one row per factor level and per value label.

Use `convert_list_columns_to_character()` to convert named list columns into character vectors (see examples).

The function `to_gt()` transform the results of `look_for()` into a nicely formatted table using `gt::gt()`. This table could be easily be exported to a file with `gt::gtsave()`. `gt::tab_header()` could be used to add a title to the table (see examples).

`look_for_and_select()` is a shortcut for selecting some variables and applying `dplyr::select()`s to return a data frame with only the selected variables.

Value

a tibble 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, Joseph Larmarange joseph@larmarange.net

Source

Inspired by the `lookfor` command in Stata.

See Also

`vignette("look_for")`

Examples

```
look_for(iris)

# Look for a single keyword.
look_for(iris, "petal")
look_for(iris, "s")
iris %>%
  look_for_and_select("s") %>%
  head()

# 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")
look_for(iris, "Pet", "sp", "width", ignore.case = FALSE)

# Look_for can search within factor levels or value labels
look_for(iris, "vers")

# Quicker search without variable details
look_for(iris, details = "none")
```

```
# To obtain more details about each variable
look_for(iris, details = "full")

# To deactivate default printing, convert to tibble
look_for(iris, details = "full") %>%
  dplyr::as_tibble()

# To convert named lists into character vectors
look_for(iris) %>% convert_list_columns_to_character()

# Long format with one row per factor and per value label
look_for(iris) %>% lookfor_to_long_format()

# Both functions can be combined
look_for(iris) %>%
  lookfor_to_long_format() %>%
  convert_list_columns_to_character()

# Labelled data
d <- dplyr::tibble(
  region = labelled_spss(
    c(1, 2, 1, 9, 2, 3),
    c(north = 1, south = 2, center = 3, missing = 9),
    na_values = 9,
    label = "Region of the respondent"
  ),
  sex = labelled(
    c("f", "f", "m", "m", "m", "f"),
    c(female = "f", male = "m"),
    label = "Sex of the respondent"
  )
)
look_for(d)
d %>%
  look_for() %>%
  lookfor_to_long_format(sep_value_labels = ":")
d %>%
  look_for() %>%
  convert_list_columns_to_character(sep_value_labels = ":", sep_other = "|")

iris %>% look_for(details = TRUE) %>% to_gt()
d %>%
  generate_dictionary() %>%
  to_gt() %>%
  gt::tab_header(gt::md("**Variable dictionary**"))
```

---

names_prefixed_by_values

Turn a named vector into a vector of names prefixed by values

---

Description

Turn a named vector into a vector of names prefixed by values

Usage

```
names_prefixed_by_values(x, sep = "[]")
```

Arguments

x	vector to be prefixed
sep	(string) separator between value and name

Examples

```
df <- dplyr::tibble(
  c1 = labelled(c("M", "M", "F"), c(Male = "M", Female = "F")),
  c2 = labelled(c(1, 1, 2), c(Yes = 1, No = 2))
)
val_labels(df$c1)
val_labels(df$c1) %>% names_prefixed_by_values()
val_labels(df)
val_labels(df) %>% names_prefixed_by_values()
val_labels(df) %>% names_prefixed_by_values(sep = ":")
```

---

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

get_na_values(x)

get_na_range(x)

set_na_values(.data, ..., .values = NA, .strict = TRUE)
```

```
set_na_range(.data, ..., .values = NA, .strict = TRUE)

is_user_na(x)

is_regular_na(x)

user_na_to_na(x)

user_na_to_regular_na(x)

user_na_to_tagged_na(x)
```

Arguments

x	A vector, a data frame, or a survey design.
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, a survey design or a vector
...	name-value pairs of missing values (see examples)
.values	missing values to be applied to the data.frame, using the same syntax as value in na_values(df) <- value or na_range(df) <- value.
.strict	should an error be returned if some labels doesn't correspond to a column of x?

Details

See haven::labelled_spss() for a presentation of SPSS's user defined missing values.

Note that base::is.na() will return TRUE for user defined missing values. It will also return TRUE for regular NA values. If you want to test if a specific value is a user NA but not a regular NA, use is_user_na(). If you want to test if a value is a regular NA but not a user NA, not a tagged NA, use is_regular_na().

You can use user_na_to_na() to convert user defined missing values to regular NA. Note that any value label attached to a user defined missing value will be lost. user_na_to_regular_na() is a synonym of user_na_to_na().

The method user_na_to_tagged_na() will convert user defined missing values into haven::tagged_na(), preserving value labels. Please note that haven::tagged_na() are defined only for double vectors. Therefore, integer haven_labelled_spss vectors will be converted into double haven_labelled vectors; and user_na_to_tagged_na() cannot be applied to a character haven_labelled_spss vector.

tagged_na_to_user_na() is the opposite of user_na_to_tagged_na() and convert tagged NA into user defined missing values.

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

get_na_values() is identical to na_values() and get_na_range() to na_range().

set_na_values() and set_na_range() could be used with **dplyr** syntax.

See Also

haven::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) # TRUE for the 6th and 10th values
is_user_na(v) # TRUE only for the 6th value

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)
user_na_to_tagged_na(v)

# it is not recommended to mix user NAs and tagged NAs
x <- c(NA, 9, tagged_na("a"))
na_values(x) <- 9
x
is.na(x)
is_user_na(x)
is_tagged_na(x)
is_regular_na(x)

if (require(dplyr)) {
  # setting value label and user NAs
  df <- tibble(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

  # example with a vector
```

```
  v <- 1:10
  v <- v %>% set_na_values(5, 6, 7)
  v
  v %>% set_na_range(8, 10)
  v %>% set_na_range(.values = c(9, 10))
  v %>% set_na_values(NULL)
}
```

---

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 dplyr::recode() method from **dplyr** to works with labelled vectors.

Usage

```
## S3 method for class 'haven_labelled'
recode(
  .x,
  ...,
  .default = NULL,
  .missing = NULL,
  .keep_value_labels = TRUE,
  .combine_value_labels = FALSE,
  .sep = " / "
)
```

Arguments

`.x` A vector to modify

`...` <dynamic-dots> 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`.

`.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`.

`.keep_value_labels` If `TRUE`, keep original value labels. If `FALSE`, remove value labels.

`.combine_value_labels` If `TRUE`, will combine original value labels to generate new value labels. Note that unexpected results could be obtained if a same old value is recoded into several different new values.

`.sep` Separator to be used when combining value labels.

See Also

`dplyr::recode()`

Examples

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

# do not keep value labels
dplyr::recode(x, `3` = 2L, .keep_value_labels = FALSE)

# be careful, changes are not of the same type (here integers),
# NA arecreated
dplyr::recode(x, `3` = 2)

# except if you provide .default or new values for all old values
dplyr::recode(x, `1` = 1, `2` = 1, `3` = 2)

# if you change the type of the vector (here transformed into character)
# value labels are lost
dplyr::recode(x, `3` = "b", .default = "a")
```

```
# use .keep_value_labels = FALSE to avoid a warning
dplyr::recode(x, `3` = "b", .default = "a", .keep_value_labels = FALSE)

# combine value labels
x <- labelled(
  1:4,
  c(
    "strongly agree" = 1,
    "agree" = 2,
    "disagree" = 3,
    "strongly disagree" = 4
  )
)
dplyr::recode(
  x,
  `1` = 1L,
  `2` = 1L,
  `3` = 2L,
  `4` = 2L,
  .combine_value_labels = TRUE
)
dplyr::recode(
  x,
  `2` = 1L,
  `4` = 3L,
  .combine_value_labels = TRUE
)
dplyr::recode(
  x,
  `2` = 1L,
  `4` = 3L,
  .combine_value_labels = TRUE,
  .sep = " or "
)
dplyr::recode(
  x,
  `2` = 1L,
  .default = 2L,
  .combine_value_labels = TRUE
)

# example when combining some values without a label
y <- labelled(1:4, c("strongly agree" = 1))
dplyr::recode(y, `2` = 1L, `4` = 3L, .combine_value_labels = TRUE)
```

`recode_if` *Recode some values based on condition*

Description

Recode some values based on condition

Usage

```
recode_if(x, condition, true)
```

Arguments

x	vector to be recoded
condition	logical vector of same length as x
true	values to use for TRUE values of condition. It must be either the same length as x, or length 1.

Value

Returns x with values replaced by true when condition is TRUE and unchanged when condition is FALSE or NA. Variable and value labels are preserved unchanged.

Examples

```
v <- labelled(c(1, 2, 2, 9), c(yes = 1, no = 2))
v %>% recode_if(v == 9, NA)
if (require(dplyr)) {
  df <- tibble(s1 = c("M", "M", "F"), s2 = c(1, 2, 1)) %>%
    set_value_labels(
      s1 = c(Male = "M", Female = "F"),
      s2 = c(A = 1, B = 2)
    ) %>%
    set_variable_labels(s1 = "Gender", s2 = "Group")

  df <- df %>%
    mutate(
      s3 = s2 %>% recode_if(s1 == "F", 2),
      s4 = s2 %>% recode_if(s1 == "M", s2 + 10)
    )
  df
  df %>% look_for()
}
```

---

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 remove

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,
  keep_var_label = FALSE,
  user_na_to_tagged_na = FALSE
)

remove_var_label(x)

remove_val_labels(x)

remove_user_na(x, user_na_to_na = FALSE, user_na_to_tagged_na = FALSE)
```

Arguments

x	A vector, a data frame or a survey design.
user_na_to_na	Convert user defined missing values into NA?
keep_var_label	Keep variable label?

`user_na_to_tagged_na`
: Convert user defined missing values into tagged NA? It could be applied only to numeric vectors. Note that integer labelled vectors will be converted to double labelled vectors.

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)`.

If you prefer to convert variables with value labels into factors, use `to_factor()` or use `unlabelled()`.

Examples

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

remove_labels(x)
remove_labels(x, user_na_to_na = TRUE)
remove_user_na(x, user_na_to_na = TRUE)
remove_user_na(x, user_na_to_tagged_na = TRUE)
```

---

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)
```

Arguments

`x`	A labelled vector or a data.frame
`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_to_user_na *Convert tagged NAs into user NAs*

Description

tagged_na_to_user_na() is the opposite of user_na_to_tagged_na() and convert tagged NA into user defined missing values (see labelled_spss()).

Usage

```
tagged_na_to_user_na(x, user_na_start = NULL)

tagged_na_to_regular_na(x)
```

Arguments

x	a vector, a data frame or a survey design
user_na_start	minimum value of the new user na, if NULL, computed automatically (maximum of observed values + 1)

Details

tagged_na_to_regular_na() converts tagged NAs into regular NAs.

Examples

```
x <- c(1:5, tagged_na("a"), tagged_na("z"), NA)
x
print_tagged_na(x)
tagged_na_to_user_na(x)
tagged_na_to_user_na(x, user_na_start = 10)

y <- c(1, 0, 1, tagged_na("r"), 0, tagged_na("d"))
val_labels(y) <- c(
  no = 0, yes = 1,
  "don't know" = tagged_na("d"),
  refusal = tagged_na("r")
)
y
tagged_na_to_user_na(y, user_na_start = 8)
tagged_na_to_regular_na(y)
tagged_na_to_regular_na(y) %>% is_tagged_na()
```

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 'double'
to_character(x, explicit_tagged_na = FALSE, ...)

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

## S3 method for class 'data.frame'
to_character(
  x,
  levels = c("labels", "values", "prefixed"),
  nolabel_to_na = FALSE,
  user_na_to_na = FALSE,
  explicit_tagged_na = FALSE,
  labelled_only = TRUE,
  ...
)

## S3 method for class 'survey.design'
to_character(
  x,
  levels = c("labels", "values", "prefixed"),
  nolabel_to_na = FALSE,
  user_na_to_na = FALSE,
  explicit_tagged_na = FALSE,
  labelled_only = TRUE,
  ...
)
```

Arguments

x	Object to coerce to a character vector.
...	Other arguments passed down to method.
explicit_tagged_na	should tagged NA be kept?
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?
user_na_to_na	user defined missing values into NA?
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`.

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

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, nolabel_to_na = TRUE)
to_character(v, "v")
to_character(v, "p")

df <- data.frame(
  a = labelled(c(1, 1, 2, 3), labels = c(No = 1, Yes = 2)),
  b = labelled(c(1, 1, 2, 3), labels = c(No = 1, Yes = 2, DK = 3)),
  c = labelled(
    c("a", "a", "b", "c"),
    labels = c(No = "a", Maybe = "b", Yes = "c")
  ),
  d = 1:4,
  e = factor(c("item1", "item2", "item1", "item2")),
  f = c("itemA", "itemA", "itemB", "itemB"),
  stringsAsFactors = FALSE
)

if (require(dplyr)) {
  glimpse(df)
  glimpse(to_character(df))
  glimpse(to_character(df, labelled_only = FALSE))
}
```

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 `haven::as_factor()` method provided by **haven** package.

`unlabelled(x)` is a shortcut for `to_factor(x, strict = TRUE, unclass = TRUE, labelled_only = TRUE)`.

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,
  explicit_tagged_na = 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,
  user_na_to_na = FALSE,
  strict = FALSE,
  unclass = FALSE,
  explicit_tagged_na = FALSE,
  ...
```

```
)

## S3 method for class 'survey.design'
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,
  user_na_to_na = FALSE,
  strict = FALSE,
  unclass = FALSE,
  explicit_tagged_na = FALSE,
  ...
)

unlabelled(x, ...)
```

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 by applying `base::unclass()`?
`explicit_tagged_na`	Should tagged NA (cf. `haven::tagged_na()`) be kept as explicit factor levels?
`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.

`unlabelled()` is a shortcut for quickly removing value labels of a vector or of a data.frame. If all observed values have a value label, then the vector will be converted into a factor. Otherwise, the vector will be unclassed. If you want to remove value labels in all cases, use `remove_val_labels()`.

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)

df <- data.frame(
  a = labelled(c(1, 1, 2, 3), labels = c(No = 1, Yes = 2)),
  b = labelled(c(1, 1, 2, 3), labels = c(No = 1, Yes = 2, DK = 3)),
  c = labelled(
    c("a", "a", "b", "c"),
    labels = c(No = "a", Maybe = "b", Yes = "c")
  ),
  d = 1:4,
  e = factor(c("item1", "item2", "item1", "item2")),
  f = c("itemA", "itemA", "itemB", "itemB"),
  stringsAsFactors = FALSE
)
if (require(dplyr)) {
  glimpse(df)
  glimpse(unlabelled(df))
}
```

`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, .quiet = FALSE, ...)
```

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`.
`.quiet`	do not display warnings for prefixed factors with duplicated codes

Details

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

`memisc_to_labelled()` converts a `memisc::data.set()]‘` object created with **memisc** package to a labelled data frame.

`foreign_to_labelled()` converts data imported with `foreign::read.spss()` or `foreign::read.dta()` from **foreign** package to a labelled data frame, i.e. using `haven::labelled()`. Factors will

not be converted. Therefore, you should use `use.value.labels = FALSE` when importing with `foreign::read.spss()` or `convert.factors = FALSE` when importing with `foreign::read.dta()`.

To convert correctly defined missing values imported with `foreign::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 `foreign::read.dta()` and could not be retrieved by `foreign_to_labelled()`.

If you convert a labelled vector into a factor with prefix, i.e. by using `to_factor(levels = "prefixed")`, `to_labelled.factor()` is able to reconvert it to a labelled vector with same values and labels.

Value

A tbl data frame or a labelled vector.

See Also

`haven::labelled()`, `foreign::read.spss()`, `foreign::read.dta()`, `memisc::data.set()`, `memisc::importer`, `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)
ds <- as.data.set(nes1948)
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))

l <- labelled(
  c(1, 1, 2, 2, 9, 2, 1, 9),
  c("yes" = 1, "no" = 2, "don't know" = 9)
)
f <- to_factor(l, levels = "p")
f
to_labelled(f)
identical(to_labelled(f), l)
```

`unique_tagged_na` *Unique elements, duplicated, ordering and sorting with tagged NAs*

Description

These adaptations of `base::unique()`, `base::duplicated()`, `base::order()` and `base::sort()` treats tagged NAs as distinct values.

Usage

```
unique_tagged_na(x, fromLast = FALSE)

duplicated_tagged_na(x, fromLast = FALSE)

order_tagged_na(
  x,
  na.last = TRUE,
  decreasing = FALSE,
  method = c("auto", "shell", "radix"),
  na_decreasing = decreasing,
  untagged_na_last = TRUE
)

sort_tagged_na(
  x,
  decreasing = FALSE,
  na.last = TRUE,
  na_decreasing = decreasing,
```

```
  untagged_na_last = TRUE
)
```

Arguments

x	a vector
fromLast	logical indicating if duplication should be considered from the last
na.last	if TRUE, missing values in the data are put last; if FALSE, they are put first
decreasing	should the sort order be increasing or decreasing?
method	the method to be used, see base::order()
na_decreasing	should the sort order for tagged NAs value be
untagged_na_last	should untagged NAs be sorted after tagged NAs? increasing or decreasing?

Examples

```
x <- c(1, 2, tagged_na("a"), 1, tagged_na("z"), 2, tagged_na("a"), NA)
x %>% print_tagged_na()

unique(x) %>% print_tagged_na()
unique_tagged_na(x) %>% print_tagged_na()

duplicated(x)
duplicated_tagged_na(x)

order(x)
order_tagged_na(x)

sort(x, na.last = TRUE) %>% print_tagged_na()
sort_tagged_na(x) %>% print_tagged_na()
```

---

update_labelled *Update labelled data to last version*

Description

Labelled data imported with **haven** version 1.1.2 or before or created with haven::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 'haven_labelled_spss'
```

```
update_labelled(x)

## S3 method for class 'haven_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.

Since haven 2.3.0, "haven_labelled" class has been evolving using now **vctrs** package.

`update_labelled()` convert labelled vectors from the old to the new classes and to reconstruct all labelled vectors with the last version of the package.

See Also

`haven::labelled()`, `haven::labelled_spss()`

---

update_variable_labels_with

Update variable/value labels with a function

Description

Update variable/value labels with a function

Usage

```
update_variable_labels_with(.data, .fn, .cols = dplyr::everything(), ...)

update_value_labels_with(.data, .fn, .cols = dplyr::everything(), ...)
```

Arguments

.data	A data frame, a data frame extension (e.g. a tibble) or a survey design
.fn	A function used to transform the variable/value labels of the selected `.cols`.
.cols	Columns to update; defaults to all columns. Use tidy selection.
...	additional arguments passed onto `.fn`.

Details

For `update_variable_labels_with()`, it is possible to access the name of the variable inside `.fn` by using `names()`, i.e. `.fn` receive a named character vector (see example). `.fn` can return `as.character(NA)` to remove a variable label.

Examples

```
df <- iris %>%
  set_variable_labels(
    Sepal.Length = "Length of sepal",
    Sepal.Width = "Width of sepal",
    Petal.Length = "Length of petal",
    Petal.Width = "Width of petal",
    Species = "Species"
  )
df$Species <- to_labelled(df$Species)
df %>% look_for()
df %>%
  update_variable_labels_with(toupper) %>%
  look_for()

# accessing variable names with names()
df %>%
  update_variable_labels_with(function(x){tolower(names(x))}) %>%
  look_for()

df %>%
  update_variable_labels_with(toupper, .cols = dplyr::starts_with("S")) %>%
  look_for()
df %>%
  update_value_labels_with(toupper) %>%
  look_for()
```

---

val_labels *Get / Set value labels*

Description

Get / Set value labels

Usage

```
val_labels(x, prefixed = FALSE)

val_labels(x, null_action = c("unclass", "labelled")) <- value

val_label(x, v, prefixed = FALSE)

val_label(x, v, null_action = c("unclass", "labelled")) <- value
```

```
get_value_labels(x, prefixed = FALSE)

set_value_labels(
  .data,
  ...,
  .labels = NA,
  .strict = TRUE,
  .overwrite = TRUE,
  .null_action = c("unclass", "labelled")
)

add_value_labels(
  .data,
  ...,
  .strict = TRUE,
  .null_action = c("unclass", "labelled")
)

remove_value_labels(
  .data,
  ...,
  .strict = TRUE,
  .null_action = c("unclass", "labelled")
)
```

Arguments

`x`	A vector, a data frame or a survey design
`prefixed`	Should labels be prefixed with values?
`null_action, .null_action`	for advanced users, if `value = NULL`, unclass the vector (default) or force/keep `haven_labelled` class (if `null_action = "labelled"`)
`value`	A named vector for `val_labels()` (see `haven::labelled()`) or a character string for `val_label()`. `NULL` to remove the labels (except if `null_action = "labelled"`). For data frames and survey design, it could also be a named list with a vector of value labels per variable.
`v`	A single value.
`.data`	a data frame or a vector
`...`	name-value pairs of value labels (see examples)
`.labels`	value labels to be applied to the data.frame, using the same syntax as `value` in `val_labels(df) <- value`.
`.strict`	should an error be returned if some labels doesn't correspond to a column of `x`?
`.overwrite`	If value labels are already defined, overwrite them?

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

`get_value_labels()` is identical to `val_labels()`.

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

`set_value_labels()` could also be applied to a vector / a data.frame column. In such case, you can provide a vector of value labels using `.labels` or several name-value pairs of value labels (see example). You could consider `dictionary_to_value_labels()` to generate a list of variable labels from a dictionary data frame. Similarly, `add_value_labels()` and `remove_value_labels()` could also be applied on vectors.

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"
v
val_label(v, 9) <- NULL
v
val_labels(v, null_action = "labelled") <- NULL
v
val_labels(v) <- NULL
v
if (require(dplyr)) {
  # setting value labels
  df <- tibble(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

  # example on a vector
  v <- 1:4
  v <- set_value_labels(v, min = 1, max = 4)
  v
  v %>% set_value_labels(middle = 3)
  v %>% set_value_labels(NULL)
  v %>% set_value_labels(.labels = c(a = 1, b = 2, c = 3, d = 4))
  v %>% add_value_labels(between = 2)
  v %>% remove_value_labels(4)
}
```

`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.

See Also

`haven::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, ...)

## S3 method for class 'data.frame'
var_label(
  x,
  unlist = FALSE,
  null_action = c("keep", "fill", "skip", "na", "empty"),
  recurse = FALSE,
  ...
)

## S3 method for class 'survey.design'
var_label(
  x,
  unlist = FALSE,
  null_action = c("keep", "fill", "skip", "na", "empty"),
  recurse = FALSE,
  ...
)

var_label(x) <- value

get_variable_labels(x, ...)

set_variable_labels(
  .data,
  ...,
  .labels = NA,
  .strict = TRUE,
  .overwrite = TRUE
)

label_attribute(x)

get_label_attribute(x)

set_label_attribute(x, value, .overwrite = TRUE)
```

```
label_attribute(x) <- value
```

Arguments

x	a vector or a data.frame
...	name-value pairs of variable labels (see examples)
unlist	for data frames and survey design, return a named vector instead of a list
null_action	for data frames and survey design, by default NULL will be returned for columns with no variable label. Use "fill" to populate with the column name instead, "skip" to remove such values from the returned list, "na" to populate with NA or "empty" to populate with an empty string ("").
recurse	if TRUE, will apply var_label() on packed columns (see tidyr::pack()) to return the variable labels of each sub-column; otherwise, the label of the group of columns will be returned.
value	a character string or NULL to remove the label For data frames, with var_label(), 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, a survey design or a vector
.labels	variable labels to be applied to the data.frame, using the same syntax as value in var_label(df) <- value.
.strict	should an error be returned if some labels doesn't correspond to a column of x?
.overwrite	if a variable label is already defined, overwrite it?

Details

get_variable_labels() is identical to var_label().

For data frames and survey design, if you are using var_label()<- and 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 be in the same order as the columns of the data frame / survey design. You could consider dictionary_to_variable_labels() to generate a list of variable labels from a dictionary data frame.

If you are using label_attribute()<- or set_label_attribute() on a data frame, the label attribute will be attached to the data frame itself, not to a column of the data frame.

If you are using packed columns (see tidyr::pack()), please read the dedicated vignette.

Value

set_variable_labels() will return an updated copy of .data.

Note

set_variable_labels() could be used with **dplyr** syntax.

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",
  Sepal.Width = NULL,
  Sepal.Length = NULL
)
var_label(iris)
var_label(iris, null_action = "fill")
var_label(iris, null_action = "skip")
var_label(iris, unlist = TRUE)

#
if (require(dplyr)) {
  # adding some variable labels
  df <- tibble(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)

  # Set labels from dictionary, e.g. as read from external file
  # One description is missing, one has no match
  description <- tibble(
    name = c(
      "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width",
      "Something"
    ),
    label = c(
      "Sepal length", "Sepal width", "Petal length", "Petal width",
      "something"
    )
  )
  var_labels <- stats::setNames(as.list(description$label), description$name)
  iris_labelled <- iris %>%
    set_variable_labels(.labels = var_labels, .strict = FALSE)
```

```
  var_label(iris_labelled)

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

  # example with a vector
  v <- 1:5
  v <- v %>% set_variable_labels("a variable label")
  v
  v %>% set_variable_labels(NULL)
}
```

`x_haven_2.0` *Datasets for testing*

Description

These datasets are used to test compatibility with foreign (spss_foreign), or haven_2.0 (x_haven_2.0, x_spss_haven_2.0) packages

Usage

```
x_haven_2.0

x_spss_haven_2.0

spss_file

dta_file
```

Format

An object of class `haven_labelled` of length 6.

An object of class `haven_labelled_spss` (inherits from `haven_labelled`) of length 10.

An object of class `list` of length 13.

An object of class `data.frame` with 47 rows and 6 columns.

Index

∗ **datasets**
x_haven_2.0, 38

add_value_labels *(val_labels)*, 31

base::as.character(), *21*
base::as.factor(), *23*
base::duplicated(), *28*
base::grep(), *8*
base::is.na(), *12*
base::order(), *28, 29*
base::sort(), *28*
base::subset(), *3*
base::unclass(), *24*
base::unique(), *28*

convert_list_columns_to_character *(look_for)*, 7
copy_labels, 2
copy_labels_from *(copy_labels)*, 2

dictionary_to_value_labels *(dictionary_to_variable_labels)*, 4
dictionary_to_value_labels(), *33*
dictionary_to_variable_labels, 4
dictionary_to_variable_labels(), *36*
dplyr::recode(), *14, 15*
dplyr::select(), *9*
drop_unused_value_labels, 6
dta_file *(x_haven_2.0)*, 38
duplicated_tagged_na *(unique_tagged_na)*, 28

foreign::read.dta(), *26, 27*
foreign::read.spss(), *26, 27*
foreign_to_labelled *(to_labelled)*, 26

generate_dictionary *(look_for)*, 7
get_label_attribute *(var_label)*, 35
get_na_range *(na_values)*, 11
get_na_values *(na_values)*, 11
get_value_labels *(val_labels)*, 31
get_variable_labels *(var_label)*, 35
gt::gt(), *9*
gt::gtsave(), *9*
gt::tab_header(), *9*

haven::as_factor(), *23*
haven::labelled(), *26, 27, 29, 30, 32*
haven::labelled_spss(), *12, 13, 30*
haven::tagged_na(), *12, 24*
haven::zap_labels(), *34*

is_prefixed, 6
is_regular_na *(na_values)*, 11
is_user_na *(na_values)*, 11

label_attribute *(var_label)*, 35
label_attribute<- *(var_label)*, 35
labelled_spss(), *20*
look_for, 7
look_for_and_select *(look_for)*, 7
lookfor *(look_for)*, 7
lookfor_to_long_format *(look_for)*, 7

memisc_to_labelled *(to_labelled)*, 26

na_range *(na_values)*, 11
na_range<- *(na_values)*, 11
na_values, 11
na_values<- *(na_values)*, 11
names_prefixed_by_values, 10
nolabel_to_na, 14

order_tagged_na *(unique_tagged_na)*, 28

print.look_for *(look_for)*, 7

recode.haven_labelled, 14
recode_if, 16
remove_attributes, 17

remove_labels, 18
remove_user_na *(*remove_labels*)*, 18
remove_val_labels *(*remove_labels*)*, 18
remove_val_labels(), *25*
remove_value_labels *(*val_labels*)*, 31
remove_var_label *(*remove_labels*)*, 18

set_label_attribute *(*var_label*)*, 35
set_na_range *(*na_values*)*, 11
set_na_values *(*na_values*)*, 11
set_value_labels *(*val_labels*)*, 31
set_value_labels(), *4*
set_variable_labels *(*var_label*)*, 35
set_variable_labels(), *4*
sort_tagged_na *(*unique_tagged_na*)*, 28
sort_val_labels, 19
spss_file *(*x_haven_2.0*)*, 38

tagged_na_to_regular_na
*(*tagged_na_to_user_na*)*, 20
tagged_na_to_regular_na(), *20*
tagged_na_to_user_na, 20
tagged_na_to_user_na(), *12*, *20*
tidyr::pack(), *36*
to_character, 21
to_factor, 23
to_factor(), *19*, *27*
to_gt *(*look_for*)*, 7
to_labelled, 26
to_labelled(), *8*

unique_tagged_na, 28
unlabelled *(*to_factor*)*, 23
unlabelled(), *19*
update_labelled, 29
update_value_labels_with
*(*update_variable_labels_with*)*,
30
update_variable_labels_with, 30
user_na_to_na *(*na_values*)*, 11
user_na_to_na(), *12, 13*
user_na_to_regular_na *(*na_values*)*, 11
user_na_to_regular_na(), *12*
user_na_to_tagged_na *(*na_values*)*, 11
user_na_to_tagged_na(), *12*, *20*

val_label *(*val_labels*)*, 31
val_label<- *(*val_labels*)*, 31
val_labels, 31
val_labels<- *(*val_labels*)*, 31
val_labels_to_na, 34
var_label, 35
var_label<- *(*var_label*)*, 35

x_haven_2.0, 38
x_spss_haven_2.0 *(*x_haven_2.0*)*, 38