Package 'declared'

May 8, 2026

Title Functions for Declared Missing Values

Version 0.26

URL https://github.com/dusadrian/declared

BugReports https://github.com/dusadrian/declared/issues

Description A zero dependency package containing functions to declare labels and missing values, coupled with associated functions to create (weighted) tables of frequencies and various other summary measures.
Some of the base functions have been rewritten to make use of the specific information about the missing values, most importantly to distinguish between empty NA and declared NA values.
Some functions have similar functionality with the corresponding ones from packages ``haven" and ``labelled". The aim is to ensure as much compatibility as possible with these packages, while offering an alternative in the objects of class ``declared".

License GPL (>= 3)

Language en-US

Encoding UTF-8

NeedsCompilation yes

Author Adrian Dusa [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-3525-9253>)

Maintainer Adrian Dusa <dusa.adrian@unibuc.ro>

Repository CRAN

Date/Publication 2026-04-02 10:30:02 UTC

Contents

`as.declared` *Labelled vectors with declared missing values*

Description

The labelled vectors are mainly used to analyse social science data, and the missing values declaration is an important step in the analysis.

Usage

```
as.declared(x, ...)

declared(
  x,
  labels = NULL,
  na_values = NULL,
  na_range = NULL,
  label = NULL,
  measurement = NULL,
  llevels = FALSE,
  ...
)

is.declared(x)

anyNAdeclared(x)
```

Arguments

`x`	A numeric vector to label, or a declared labelled vector (for `undeclare`)
`...`	Other arguments used by various other methods
`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
`measurement`	Optional, user specified measurement level
`llevels`	Logical, when `x` is a factor only use those levels that have labels

Details

The `declared` objects are very similar to the `haven_labelled_spss` objects from package **haven**. It has exactly the same arguments, but it features a fundamental difference in the treatment of (declared) missing values.

In package **haven**, existing values are treated as if they were missing. By contrast, in package **declared** the NA values are treated as if they were existing values.

This difference is fundamental and points to an inconsistency in package **haven**: while existing values can be identified as missing using the function `is.na()`, they are in fact present in the vector and other packages (most importantly the base ones) do not know these values should be treated as missing.

Consequently, the existing values are interpreted as missing only by package **haven**. Statistical procedures will use those values as if they were valid values.

Package **declared** approaches the problem in exactly the opposite way: instead of treating existing values as missing, it treats (certain) NA values as existing. It does that by storing an attribute containing the indices of those NA values which are to be treated as declared missing values, and it refreshes this attribute each time the declared object is changed.

This is a trade off and has important implications when subsetting datasets: all declared variables get this attribute refreshed, which consumes some time depending on the number of variables in the data.

The generic function `as.declared()` attempts to coerce only the compatible types of objects, namely `haven_labelled` and `factors`. Dedicated class methods can be written for any other type of object, and users are free to write their own. To end of with a declared object, additional metadata is needed such as value labels, which values should be treated as missing etc.

The measurement level is optional and, for the moment, purely aesthetic. It might however be useful to (automatically) determine if a declared object is suitable for a certain statistical analysis, for instance regression requires quantitative variables, while some declared objects are certainly categorical despite using numbers to denote categories.

It distinguishes between `"categorical"` and `"quantitative"` types of variables, and additionally recognizes `"nominal"` and `"ordinal"` as categorical, and similarly recognizes `"interval"`, `"ratio"`, `"discrete"` and `"continuous"` as quantitative.

Value

`declared()` and `as.declared()` return labelled vector of class "declared". When applied to a data frame, `as.declared()` will return a corresponding data frame with declared variables. `is.declared()` and `anyNAdeclared()` return a logical value.

Examples

```
x <- declared(
    c(1:5, -1),
    labels = c(Good = 1, Bad = 5, DK = -1),
    na_values = -1
)

x
```

```
is.na(x)

x > 0

x == -1

# Values are actually placeholder for categories,
# so labels work as if they were factors:
x == "DK"


# when newly added values are already declared as missing,
# they are automatically coerced
c(x, 2, -1)

# switch NAs with their original values
undeclare(x)

as.character(x)

# Returning values instead of categories
as.character(x, values = TRUE)

anyNAdeclared(x) # contains declared missing values

anyNAdeclared(c(1:5, NA)) # no declared missing values
```

`as.haven` *Coerce to haven / labelled objects*

Description

Convert declared labelled objects to haven labelled objects

Usage

```
as.haven(x, ...)
```

Arguments

`x`	A declared labelled vector
`...`	Other arguments used by various methods

Details

This is a function that reverses the process of `as.declared()`, making a round trip between `declared` and `haven_labelled_spss` classes.

Value

A labelled vector of class "haven_labelled_spss".

Examples

```
x <- declared(
    c(1:5, -1),
    labels = c(Good = 1, Bad = 5, DK = -1),
    na_values = -1
)

x

as.haven(x)
```

`drop_undeclare` *Drop information / undeclare labelled objects*

Description

A function to obtain a version of the object with all information about declared missing values, dropped

Usage

```
undeclare(x, drop = FALSE, ...)

drop_na(x, drop_labels = TRUE)
```

Arguments

`x`	A labelled object with declared missing values
`drop`	Logical, drop all attributes
`...`	Other internal arguments
`drop_labels`	Logical, drop the labels for the declared missing values

Details

The function `undeclare()` replaces the NA entries into their original numeric values, and drops all attributes related to missing values: `na_values`, `na_range` and `na_index`, and it preserves the labels referring to the missing values.

The result can be a regular vector (dropping all attributes, including the class "declared") by activating the argument `drop`.

Function `drop_na()` transforms the declared missing values in regular empty NAs, and the labels referring to the missing values are deleted by default.

Function `drop()` deletes all attributes.

Value

A declared labelled object.

See Also

Other labelling functions: `labels()`, `measurement()`

Examples

```
x <- declared(
    c(-2, 1:5, -1),
    labels = c("Good" = 1, "Bad" = 5, "DK" = -1),
    na_values = c(-1, -2),
    label = "Test variable"
)

x

undeclare(x)

drop_na(x)

drop(x)

undeclare(x, drop = TRUE)

# similar to:
drop(undeclare(x))
```

`is.empty` *Test the presence of empty (undeclared) missing values*

Description

Functions that indicate which elements are empty NA missing values, in contrast to declared missing values.

Usage

```
is.empty(x)

anyNAempty(x)
```

Arguments

x	A vector

Details

All missing values, declared or undeclared, as stored as regular NA values, therefore the base function is_na() does not differentiate between them.

These functions are specifically adapted to objects of class "declared", to return a truth value only for those elements that are completely missing with no reason.

Value

A logical vector.

Examples

```
x <- declared(
    c(1:2, -91),
    labels = c(Good = 1, Bad = 2, Missing = -91),
    na_values = -91
)

x

is.empty(x) # FALSE FALSE FALSE

anyNAempty(x) # FALSE

x <- c(x, NA)

is.empty(x) # FALSE FALSE FALSE  TRUE

anyNAempty(x) # TRUE
```

---

labels *Get / Declare value labels*

Description

Functions to extract information about the declared variable / value labels, or to declare such values if they are present in the data.

Usage

```
label(x)

label(x, ...) <- value

labels(x) <- value
```

Arguments

x	Any vector of values that should be declared as missing (for `labels`) or a numeric vector of length two giving the (inclusive) extents of the range of missing values (for `label`).
...	Other arguments, for internal use.
value	The variable label, or a list of (named) variable labels

Details

The function `labels()` is a adaptation of the base function to the objects of class `declared`. In addition to the regular arguments, it has two additional (logical) arguments called `prefixed` (FALSE by default), to retrieve the value labels prefixed with their values, and `print_as_df` (TRUE by default) to print the result as a data frame.

Value

`labels()` will return a named vector.

`label()` will return a single character string.

See Also

Other labelling functions: `drop_undeclare`, `measurement()`

Examples

```
x <- declared(
    c(-2, 1:5, -1),
    labels = c("Good" = 1, "Bad" = 5, "DK" = -1),
    na_values = c(-1, -2),
    label = "Test variable"
)
x

labels(x)

labels(x, prefixed = TRUE)

labels(x) <- c("Good" = 1, "Bad" = 5, "DK" = -1, "Not applicable" = -2)

label(x)

label(x) <- "This is a proper label"

x
```

`measurement` *Get / Set measurement levels for declared objects*

Description

Functions to extract information about the measurement levels of a variable (if already present), or to specify such measurement levels.

Usage

```
measurement(x)

measurement(x) <- value
```

Arguments

`x`	A declared vector.
`value`	A single character string of measurement levels, separated by commas.

Details

This function creates an attribute called `"measurement"` to a declared object. This is an optional feature, at this point for purely aesthetic reasons, but it might become useful in the future to (automatically) determine if a declared object is suitable for a certain statistical analysis, for instance regression requires quantitative variables, while some declared objects are certainly categorical despite using numbers to denote categories.

It distinguishes between `"categorical"` and `"quantitative"` types of variables, and additionally recognizes `"nominal"` and `"ordinal"` as categorical, and similarly recognizes `"interval"`, `"ratio"`, `"discrete"` and `"continuous"` as quantitative.

The words `"qualitative"` is treated as a synonym for `"categorical"`, and the words `"metric"` and `"numeric"` are treated as synonyms for `"quantitative"`, respectively.

Value

A character vector.

See Also

Other labelling functions: `drop_undeclare`, `labels()`

Examples

```
x <- declared(
    c(-2, 1:5, -1),
    labels = c(Good = 1, Bad = 5, DK = -1),
    na_values = c(-1, -2),
    label = "Test variable"
```

```
)

x

measurement(x)

# automatically recognized as categorical
measurement(x) <- "ordinal"

measurement(x)

# the same with
measurement(x) <- "categorical, ordinal"

set.seed(1890)
x <- declared(
    sample(c(18:90, -91), 20, replace = TRUE),
    labels = c("No answer" = -91),
    na_values = -91,
    label = "Respondent's age"
)

# automatically recognized as quantitative
measurement(x) <- "discrete"

measurement(x)

# the same with
measurement(x) <- "metric, discrete"
```

`missing_range` *Get / Declare missing values*

Description

Functions to extract information about the declared missing values, or to declare such values if they are present in the data.

Usage

```
missing_range(x)

missing_range(x) <- value

missing_values(x)

missing_values(x) <- value
```

Arguments

x	A vector.
value	Any vector of values that should be declared as missing (for missing_values) or a numeric vector of length two giving the (inclusive) extents of the range of missing values (for missing_range).

Value

missing_values() will return a vector of one or more values.

missing_range() will return a numeric vector of length 2.

Examples

```
x <- declared(c(-2, 1:5, -1),
    labels = c(Good = 1, Bad = 5, DK = -1, NotApplicable = -2),
    na_values = c(-1, -2)
)
x

missing_values(x)

missing_range(x) <- c(-10, -7)

missing_range(x)
```

---

wIQR *Compute weighted summaries for declared objects*

Description

Functions to compute weighted tables or summaries, based on a vector of frequency weights. These are reimplementations of various existing functions, adapted to objects of class "declared" (see Details below)

Usage

```
wIQR(x, wt = NULL, na.rm = FALSE, ...)

wfivenum(x, wt = NULL, na.rm = FALSE)

wmean(x, wt = NULL, trim = 0, na.rm = TRUE)

wmedian(x, wt = NULL, na.rm = TRUE, ...)

wmode(x, wt = NULL)
```

```
wquantile(x, wt = NULL, probs = seq(0, 1, 0.25), na.rm = TRUE, ...)

wsd(x, wt = NULL, method = NULL, na.rm = TRUE)

wstandardize(x, wt = NULL, na.rm = TRUE)

wsummary(x, wt = NULL, ...)

wtable(
  x,
  y = NULL,
  wt = NULL,
  values = TRUE,
  valid = TRUE,
  observed = TRUE,
  vlabel = FALSE
)

wvar(x, wt = NULL, method = NULL, na.rm = TRUE)
```

Arguments

x	A numeric vector for summaries, or declared / factor for frequency tables
wt	A numeric vector of frequency weights
na.rm	Logical, should the empty missing values be removed?
...	Further arguments passed to or from other methods.
trim	A 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.
probs	Numeric vector of probabilities with values in [0,1]
method	Character, specifying how the result is scaled, see 'Details' below.
y	An optional variable, to create crosstabs; must have the same length as x
values	Logical, print the values in the table rows
valid	Logical, print separate percent distribution for valid values, if any missing values are present; for cross tables, use valid values only
observed	Logical, print the observed categories only
vlabel	Logical, print the variable label, if existing

Details

Weighted summaries

A frequency table is usually performed for a categorical variable, displaying the frequencies of the respective categories. Note that general variables containing text are not necessarily factors, despite having a small number of characters.

A general table of frequencies, using the base function `table()`, ignores the defined missing values (which are all stored as NAs). The reimplementation of this function in `wtable()` takes care of this detail, and presents frequencies for each separately defined missing values. Similar reimplementations for the other functions have the same underlying objective.

It is also possible to perform a frequency table for numerical variables, if the number of values is limited (an arbitrary and debatable upper limit of 15 is used). An example of such variable can be the number of children, where each value can be interpreted as a class, containing a single value (for instance 0 meaning the category of people with no children).

Objects of class `declared` are not pure categorical variables (R factors) but they are nevertheless interpreted as if they were factors, to allow producing frequency tables. Given the high similarity with package `haven`, objects of class `haven_labelled_spss` are automatically coerced to objects of class `declared` and treated accordingly.

The argument `values` makes sense only when the input is of family class `declared`, otherwise for regular (base R) factors the values are just a sequence of numbers.

The later introduced argument `observed` is useful in situations when a variable has a very large number of potential values, and a smaller subset of actually observed ones. As an example, the variable "Occupation" has hundreds of possible values in the ISCO08 codelist, and not all of them might be actually observed. When activated, this argument restricts the printed frequency table to the subset of observed values only.

The argument `method` can be one of `"unbiased"` or `"ML"`.

When this is set to `"unbiased"`, the result is an unbiased estimate using Bessel's correction. When this is set to `"ML"`, the result is the maximum likelihood estimate for a Gaussian distribution.

The argument `wt` refers only to frequency weights. Users should be aware of the differences between frequency weights, analytic weights, probability weights, design weights, post-stratification weights etc. For purposes of inferential testing, Thomas Lumley's package `survey` should be employed.

If no frequency weights are provided, the result is identical to the corresponding base functions.

The function `wquantile()` extensively borrowed ideas from packages `stats` and `Hmisc`, to ensure a constant interpolation that would produce the same quantiles if no weights are provided or if all weights are equal to 1.

Other arguments can be passed to the stats function `quantile()` via the three dots `...` argument, and their extensive explanation is found in the corresponding stats function's help page.

For all functions, the argument `na.rm` refers to the empty missing values and its default is set to TRUE. The declared missing values are automatically eliminated from the summary statistics, even if this argument is deactivated.

The function `wmode()` returns the weighted mode of a variable. Unlike the other functions where the prefix `w` signals a weighted version of the base function with the same name, this has nothing to do with the base function `mode()` which refers to the storage mode / type of an R object.

Value

A vector of (weighted) values.

Author(s)

Adrian Dusa

Examples

```
set.seed(215)

# a pure categorical variable
x <- factor(sample(letters[1:5], 215, replace = TRUE))
wtable(x)


# simulate number of children
x <- sample(0:4, 215, replace = TRUE)
wtable(x)

# simulate a Likert type response scale from 1 to 7
values <- sample(c(1:7, -91), 215, replace = TRUE)
x <- declared(values, labels = c("Good" = 1, "Bad" = 7))
wtable(x)


# Defining missing values
missing_values(x) <- -91
wtable(x)


# Defined missing values with labels
values <- sample(c(1:7, -91, NA), 215, replace = TRUE)
x <- declared(
    values,
    labels = c("Good" = 1, "Bad" = 7, "Don't know" = -91),
    na_values = -91
)

wtable(x)

# Including the values in the table of frequencies
wtable(x, values = TRUE)


# An example involving multiple variables
DF <- data.frame(
    Area = declared(
        sample(1:2, 215, replace = TRUE, prob = c(0.45, 0.55)),
        labels = c(Rural = 1, Urban = 2)
    ),
    Gender = declared(
        sample(1:2, 215, replace = TRUE, prob = c(0.55, 0.45)),
        labels = c(Males = 1, Females = 2)
    ),
    Age = sample(18:90, 215, replace = TRUE),
    Children = sample(0:5, 215, replace = TRUE)
)

wtable(DF$Gender)
```

```
wsd(DF$Age)


# Weighting: observed proportions
op <- proportions(with(DF, table(Gender, Area)))

# Theoretical proportions: 53% Rural, and 50.2% Females
tp <- rep(c(0.53, 0.47), each = 2) * rep(c(0.498, 0.502), 2)

# Corrections by strata
fweights <- tp / op

DF$fweight <- fweights[match(10 * DF$Area + DF$Gender, c(11, 12, 21, 22))]

with(DF, wtable(Gender, wt = fweight))

with(DF, wmean(Age, wt = fweight))

with(DF, wquantile(Age, wt = fweight))
```

Index