Package 'haven'

July 4, 2019

Title Import and Export 'SPSS', 'Stata' and 'SAS' Files

Version 2.1.1

Description Import foreign statistical formats into R via the embedded 'ReadStat' C library, <https://github.com/WizardMac/ReadStat>.

License MIT + file LICENSE

URL http://haven.tidyverse.org, https://github.com/tidyverse/haven, https://github.com/WizardMac/ReadStat

BugReports https://github.com/tidyverse/haven/issues

Depends R (>= 3.2)

Imports forcats (>= 0.2.0), hms, Rcpp (>= 0.11.4), readr (>= 0.1.0), tibble

Suggests covr, fs, knitr, rmarkdown, testthat, pillar (>= 1.1.1), cli, crayon

LinkingTo Rcpp

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

SystemRequirements GNU make

NeedsCompilation yes

Author Hadley Wickham [aut, cre], Evan Miller [aut, cph] (Author of included ReadStat code), RStudio [cph, fnd]

Maintainer Hadley Wickham <hadley@rstudio.com>

Repository CRAN

Date/Publication 2019-07-04 14:50:03 UTC

R topics documented:

`as_factor` *Convert input to a factor.*

Description

The base function `as.factor()` is not a generic, but this variant is. Methods are provided for factors, character vectors, labelled vectors, and data frames. By default, when applied to a data frame, it only affects labelled columns.

Usage

```
## S3 method for class 'data.frame'
as_factor(x, ..., only_labelled = TRUE)

## S3 method for class 'haven_labelled'
as_factor(x, levels = c("default", "labels",
  "values", "both"), ordered = FALSE, ...)

## S3 method for class 'labelled'
as_factor(x, levels = c("default", "labels", "values",
  "both"), ordered = FALSE, ...)
```

Arguments

`x`	Object to coerce to a factor.
`...`	Other arguments passed down to method.
`only_labelled`	Only apply to labelled columns?

`levels`	How to create the levels of the generated factor: • "default": uses labels where available, otherwise the values. Labels are sorted by value. • "both": like "default", but pastes together the level and value • "label": use only the labels; unlabelled values become `NA` • "values: use only the values
`ordered`	If `TRUE` create an ordered (ordinal) factor, if `FALSE` (the default) create a regular (nominal) factor.

Details

Includes methods for both class `haven_labelled` and `labelled` for backward compatibility.

Examples

```
x <- labelled(sample(5, 10, replace = TRUE), c(Bad = 1, Good = 5))

# Default method uses values where available
as_factor(x)
# You can also extract just the labels
as_factor(x, levels = "labels")
# Or just the values
as_factor(x, levels = "values")
# Or combine value and label
as_factor(x, levels = "both")
```

`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. This class makes it possible to import such labelled vectors in to R without loss of fidelity. This class provides few methods, as I expect you'll coerce to a standard R class (e.g. a `factor()`) soon after importing.

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.

Examples

```
s1 <- labelled(c("M", "M", "F"), c(Male = "M", Female = "F"))
s2 <- labelled(c(1, 1, 2), c(Male = 1, Female = 2))
s3 <- labelled(c(1, 1, 2), c(Male = 1, Female = 2),
               label="Assigned sex at birth")

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

# Other statistical software supports multiple types of missing values
s3 <- labelled(c("M", "M", "F", "X", "N/A"),
  c(Male = "M", Female = "F", Refused = "X", "Not applicable" = "N/A")
)
s3
as_factor(s3)

# Often when you have a partially labelled numeric vector, labelled values
# are special types of missing. Use zap_labels to replace labels with missing
# values
x <- labelled(c(1, 2, 1, 2, 10, 9), c(Unknown = 9, Refused = 10))
zap_labels(x)
```

---

labelled_spss *Labelled vectors for SPSS*

Description

This class is only used when `user_na = TRUE` in `read_sav()`. 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.

Examples

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

x2 <- labelled_spss(1:10, c(Good = 1, Bad = 8), na_range = c(9, Inf),
                    label = "Quality rating")
is.na(x2)

# Print data and metadata
x2
```

---

print_labels *Print the labels of a labelled vector*

Description

This is a convenience function, useful to explore the variables of a newly imported dataset.

Usage

```
print_labels(x, name = NULL)
```

Arguments

x	A labelled vector
name	The name of the vector (optional)

Examples

```
s1 <- labelled(c("M", "M", "F"), c(Male = "M", Female = "F"))
s2 <- labelled(c(1, 1, 2), c(Male = 1, Female = 2))
labelled_df <- tibble::tibble(s1, s2)

for (var in names(labelled_df)) {
  print_labels(labelled_df[[var]], var)
}
```

---

read_dta *Read and write Stata DTA files.*

Description

Currently haven can read and write logical, integer, numeric, character and factors. See labelled() for how labelled variables in Stata are handled in R.

Usage

```
read_dta(file, encoding = NULL)

read_stata(file, encoding = NULL)

write_dta(data, path, version = 14)
```

Arguments

file	Either a path to a file, a connection, or literal data (either a single string or a raw vector). Files ending in .gz, .bz2, .xz, or .zip will be automatically uncompressed. Files starting with http://, https://, ftp://, or ftps:// will be automatically downloaded. Remote gz files can also be automatically downloaded and decompressed. Literal data is most useful for examples and tests. It must contain at least one new line to be recognised as data (instead of a path) or be a vector of greater than length 1. Using a value of clipboard() will read from the system clipboard.
encoding	The character encoding used for the file. Generally, only needed for Stata 13 files and earlier. See Encoding section for details.
data	Data frame to write.
path	Path to a file where the data will be written.
version	File version to use. Supports versions 8-15.

Value

A tibble, data frame variant with nice defaults.

Variable labels are stored in the "label" attribute of each variable. It is not printed on the console, but the RStudio viewer will show it.

write_dta() returns the input data invisibly.

Character encoding

Prior to Stata 14, files did not declare a text encoding, and the default encoding differed across platforms. If encoding = NULL, haven assumes the encoding is windows-1252, the text encoding used by Stata on Windows. Unfortunately Stata on Mac and Linux use a different default encoding, "latin1". If you encounter an error such as "Unable to convert string to the requested encoding", try encoding = "latin1"

For Stata 14 and later, you should not need to manually specify encoding value unless the value was incorrectly recorded in the source file.

Examples

```
path <- system.file("examples", "iris.dta", package = "haven")
read_dta(path)

tmp <- tempfile(fileext = ".dta")
write_dta(mtcars, tmp)
read_dta(tmp)
read_stata(tmp)
```

`read_sas` *Read and write SAS files.*

Description

`read_sas()` supports both sas7bdat files and the accompanying sas7bcat files that SAS uses to record value labels. `write_sas()` is currently experimental and only works for limited datasets.

Usage

```
read_sas(data_file, catalog_file = NULL, encoding = NULL,
  catalog_encoding = encoding, cols_only = NULL)

write_sas(data, path)
```

Arguments

`data_file, catalog_file`	Path to data and catalog files. The files are processed with `readr::datasource()`.
`encoding, catalog_encoding`	The character encoding used for the `data_file` and `catalog_encoding` respectively. A value of `NULL` uses the encoding specified in the file; use this argument to override it if it is incorrect.
`cols_only`	A character vector giving an experimental way to read in only specified columns.
`data`	Data frame to write.
`path`	Path to file where the data will be written.

Value

A tibble, data frame variant with nice defaults.

Variable labels are stored in the "label" attribute of each variable. It is not printed on the console, but the RStudio viewer will show it.

`write_sas()` returns the input `data` invisibly.

Examples

```
path <- system.file("examples", "iris.sas7bdat", package = "haven")
read_sas(path)
```

read_spss *Read SPSS (.sav, .zsav, .por) files. Write .sav and .zsav files.*

Description

read_sav() reads both .sav and .zsav files; write_sav() creates .zsav files when compress = TRUE. read_por() reads .por files. read_spss() uses either read_por() or read_sav() based on the file extension.

Usage

```
read_sav(file, encoding = NULL, user_na = FALSE)

read_por(file, user_na = FALSE)

write_sav(data, path, compress = FALSE)

read_spss(file, user_na = FALSE)
```

Arguments

file	Either a path to a file, a connection, or literal data (either a single string or a raw vector). Files ending in .gz, .bz2, .xz, or .zip will be automatically uncompressed. Files starting with http://, https://, ftp://, or ftps:// will be automatically downloaded. Remote gz files can also be automatically downloaded and decompressed. Literal data is most useful for examples and tests. It must contain at least one new line to be recognised as data (instead of a path) or be a vector of greater than length 1. Using a value of clipboard() will read from the system clipboard.
encoding	The character encoding used for the file. The default, NULL, use the encoding specified in the file, but sometimes this value is incorrect and it is useful to be able to override it.
user_na	If TRUE variables with user defined missing will be read into labelled_spss() objects. If FALSE, the default, user-defined missings will be converted to NA.
data	Data frame to write.
path	Path to a file where the data will be written.
compress	If TRUE, will compress the file, resulting in a .zsav file.

Details

Currently haven can read and write logical, integer, numeric, character and factors. See labelled_spss() for how labelled variables in SPSS are handled in R.

Value

A tibble, data frame variant with nice defaults.

Variable labels are stored in the "label" attribute of each variable. It is not printed on the console, but the RStudio viewer will show it.

`write_sav()` returns the input `data` invisibly.

Examples

```
path <- system.file("examples", "iris.sav", package = "haven")
read_sav(path)

tmp <- tempfile(fileext = ".sav")
write_sav(mtcars, tmp)
read_sav(tmp)
```

`read_xpt` *Read and write SAS transport files*

Description

The SAS transport format is a open format, as is required for submission of the data to the FDA.

Usage

```
read_xpt(file)

write_xpt(data, path, version = 8, name = NULL)
```

Arguments

`file`	Either a path to a file, a connection, or literal data (either a single string or a raw vector). Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically uncompressed. Files starting with `http://`, `https://`, `ftp://`, or `ftps://` will be automatically downloaded. Remote gz files can also be automatically downloaded and decompressed. Literal data is most useful for examples and tests. It must contain at least one new line to be recognised as data (instead of a path) or be a vector of greater than length 1. Using a value of `clipboard()` will read from the system clipboard.
`data`	Data frame to write.
`path`	Path to a file where the data will be written.
`version`	Version of transport file specification to use: either 5 or 8.
`name`	Member name to record in file. Defaults to file name sans extension. Must be <= 8 characters for version 5, and <= 32 characters for version 8.

Value

A tibble, data frame variant with nice defaults.

Variable labels are stored in the "label" attribute of each variable. It is not printed on the console, but the RStudio viewer will show it.

`write_xpt()` returns the input `data` invisibly.

Examples

```
tmp <- tempfile(fileext = ".xpt")
write_xpt(mtcars, tmp)
read_xpt(tmp)
```

---

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"))
```

---

zap_empty *Convert empty strings into missing values.*

Description

Convert empty strings into missing values.

Usage

```
zap_empty(x)
```

Arguments

x	A character vector

Value

A character vector with empty strings replaced by missing values.

See Also

Other zappers: zap_formats, zap_labels, zap_label, zap_widths

Examples

```
x <- c("a", "", "c")
zap_empty(x)
```

zap_formats *Remove format attributes*

Description

To provide some mild support for round-tripping variables between Stata/SPSS and R, haven stores variable formats in an attribute: `format.stata`, `format.spss`, or `format.sas`. If this causes problems for your code, you can get rid of them with `zap_formats`.

Usage

```
zap_formats(x)
```

Arguments

x	A vector or data frame.

See Also

Other zappers: `zap_empty`, `zap_labels`, `zap_label`, `zap_widths`

zap_label *Zap label*

Description

Removes label, leaving unlabelled vectors as is. Use this if you want to simply drop all `label` attributes from a data frame.

Usage

```
zap_label(x)
```

Arguments

x	A vector or data frame

See Also

Other zappers: `zap_empty`, `zap_formats`, `zap_labels`, `zap_widths`

Examples

```
x1 <- labelled(1:5, c(good = 1, bad = 5))
x1
zap_label(x1)

x2 <- labelled_spss(c(1:4, 9), c(good = 1, bad = 5), na_values = 9)
x2
zap_label(x2)

# zap_label also works with data frames
df <- tibble::tibble(x1, x2)
df
zap_label(df)
```

`zap_labels` *Zap labels*

Description

Removes labels, leaving unlabelled vectors as is. Use this if you want to simply drop all `labels` from a data frame. Zapping labels from `labelled_spss()` also removes user-defined missing values, replacing all with NAs.

Usage

```
zap_labels(x)
```

Arguments

x	A vector or data frame

Note

This function doesn't remove any `label` attribute(s), just the `labels` attribute(s). Use `zap_label` to remove `label` attribute(s).

See Also

Other zappers: `zap_empty`, `zap_formats`, `zap_label`, `zap_widths`

Examples

```
x1 <- labelled(1:5, c(good = 1, bad = 5))
x1
zap_labels(x1)

x2 <- labelled_spss(c(1:4, 9), c(good = 1, bad = 5), na_values = 9)
x2
zap_labels(x2)
```

```
# zap_labels also works with data frames
df <- tibble::tibble(x1, x2)
df
zap_labels(df)
```

`zap_missing` *Zap special missings to regular R missings*

Description

This is useful if you want to convert tagged missing values from SAS or Stata, or user-defined missings from SPSS, to regular R `NA`.

Usage

```
zap_missing(x)
```

Arguments

`x`	A vector or data frame

Examples

```
x1 <- labelled(
  c(1, 5, tagged_na("a", "b")),
  c(Unknown = tagged_na("a"), Refused = tagged_na("b"))
)
x1
zap_missing(x1)

x2 <- labelled_spss(
  c(1, 2, 1, 99),
  c(missing = 99),
  na_value = 99
)
x2
zap_missing(x2)

# You can also apply to data frames
df <- tibble::tibble(x1, x2, y = 4:1)
df
zap_missing(df)
```

zap_widths *Remove display width attributes*

Description

To provide some mild support for round-tripping variables between SPSS and R, haven stores display widths in an attribute: `display_width`. If this causes problems for your code, you can get rid of them with `zap_widths`.

Usage

```
zap_widths(x)
```

Arguments

x	A vector or data frame.

See Also

Other zappers: `zap_empty`, `zap_formats`, `zap_labels`, `zap_label`

Index