Package: mStats (via r-universe)

August 24, 2024

Type Package

Title Medical Statistics & Epidemiological Analysis

Version 3.5.0

Author Myo Minn Oo <dr.myominnoo@gmail.com>

Maintainer Myo Minn Oo <dr.myominnoo@gmail.com>

Description A set of tidyverse-friendly functions for data management,
calculation of epidemiological measures, statistical analysis,
and table creation.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Depends R (>= 3.3.0)

Imports pillar (>= 1.9.0), cli, crayon, labelled, lifecycle, purrr,
rlang, scales, tidyr, tibble

Suggests covr, dplyr, haven, knitr, nycflights13, rmarkdown, testthat

Config/testthat/edition 3

VignetteBuilder knitr

URL https://myominnoo.github.io/mStats/

Repository https://myominnoo.r-universe.dev

RemoteUrl https://github.com/myominnoo/mstats

RemoteRef HEAD

RemoteSha 6c01653cbeec328cefe0d59d865d919dcd5ccdf2

Contents

`append` *Append datasets*

Description

[Deprecated]

Usage

```
append(...)
```

Arguments

`...`	Data frames to combine.

Details

append stacks multiple datasets.

Value

A data frame

See Also

Other Data Management: `codebook()`, `count_functions`, `cut()`, `tag_duplicates()`

Examples

```
append(airquality, mtcars)
```

codebook *Generate a codebook*

Description

[Stable]

The codebook function generates a codebook for the given dataset. It provides a summary of the dataset's structure and characteristics, including variable names, types, missing values, completeness percentages, unique value counts, and variable labels (if available).

Usage

```
codebook(data)
```

Arguments

`data`	The dataset for which the codebook is to be generated.

Value

The input dataset is returned invisibly, allowing `codebook()` to be used within a data pipe line.

See Also

Other Data Management: `append()`, `count_functions`, `cut()`, `tag_duplicates()`

Examples

```
codebook(mtcars)

codebook(iris)

labelled::var_label(iris) <- c(
 "sepal length", "sepal width", "petal length",
 "petal width", "species"
)
codebook(iris)
```

coder — *Apply functions to multiple variables in a data frame*

Description

The `coder()` function applies a set of functions to multiple variables in a data frame using `dplyr`'s `mutate()` and `across()` functions.

Usage

```
coder(.data, .fns)
```

Arguments

`.data`	The input data frame.
`.fns`	A set of functions to be applied to the variables in the data frame.

Value

A modified data frame with the functions applied to the variables.

Examples

```
# Apply the `mean()` function to multiple variables in the `mtcars` data frame
coder(mtcars, mean)
```

count_functions — *Count from n to N*

Description

[Stable]

Usage

```
n_(...)

N_(...)
```

Arguments

`...`	<`tidy-select`> Columns to pick. You can't pick grouping columns because they are already automatically handled by the verb (i.e. `summarise()` or `mutate()`).

Details

These functions are used for indexing observations or generating sequences of numbers.

- `n_()` generates a running counter within a group of variables and represents the number of the current observation.
- `N_()` provides the total count within each group of variables.

You can do these operations using `dplyr::n()` in this way. See examples below using iris dataset.

iris |> mutate(.N_ = n()) |> head() iris |> mutate(.n_ = 1:n()) |> head() iris |> group_by(Species) |> mutate(.n_ = 1:n()) |> slice(1:5) |> ungroup()

Value

A numeric vector representing the count from `n` to `N`.

See Also

Other Data Management: `append()`, `codebook()`, `cut()`, `tag_duplicates()`

Examples

```
# Example with a custom dataset
df <- data.frame(
  x = c(1, 1, 2, 2, 2, 3, 4, 4, 4, 4),
  y = letters[1:10]
)

library(dplyr)

# Generate a running counter for each observation within the "x" group using mutate()
mutate(df, n = n_(x))

# Generate a running counter for each observation for all columns using mutate()
mutate(df, n = n_(everything()))

# Generate the total count of observations using summarise()
reframe(df, n = n_(x))

# Generate the total count of observations within the "x" group using summarise()
mutate(df, N = N_(everything()))
mutate(df, N = N_(x))
reframe(df, N = N_(x))

# iris dataset
mutate(iris, n = n_(everything()))
mutate(iris, N = N_(everything()))
```

`cut` *Cut numeric vector into factor vector*

Description

[Stable]

Usage

```
cut(x, at, label = NULL, ...)
```

Arguments

`x`	A numeric vector to be cut into factors.
`at`	A numeric vector specifying the breakpoints or categories for cutting the vector. If a single value is provided, the function will create breaks using the same method as `[base::cut]`. If multiple values are provided, they are treated as specific breaks.
`label`	Optional labels for the resulting factor levels. If not provided, labels will be automatically generated based on the breaks.
`...`	Additional arguments to be passed to `[base::cut]` if `x` is not numeric.

Details

This function cuts a numeric vector into factors based on specified breaks or categories. If the input vector is not numeric, the function delegates to the base R `cut` function.

Value

A factor representing the cut vector with factor levels assigned based on the breaks or categories.

See Also

Other Data Management: `append()`, `codebook()`, `count_functions`, `tag_duplicates()`

Examples

```
x <- c(1, 2, 3, 4, 5)
cut(x, 2)
```

decode — *Decode variables in a data frame*

Description

The `decode()` function decodes selected variables in a data frame. It converts the variables to character type, sets labels from the original data, and returns the modified data frame.

Usage

```
decode(.data, ...)
```

Arguments

`.data`	The input data frame.
`...`	The variables to be decoded.

Value

A modified data frame with the selected variables decoded.

Examples

```
# Decode selected variables in the `mtcars` data frame
decode(mtcars, mpg, cyl)
```

egen — *Convert a continuous variable into groups*

Description

[Deprecated]

This function was deprecated because we realized that it's a special case of the `cut` function.

Usage

```
egen(data, var, at = NULL, label = NULL, new_var = NULL, ...)
```

Arguments

`data`	data.frame
`var`	existing variable
`at`	either a number or a numeric vector
`label`	Labels for the groups
`new_var`	Name of the new variable
`...`	Additional arguments to be passed to `cut`

See Also

`cut`

Examples

```
data <- data.frame(x = 1:10)
egen(data, x, at = c(3, 7), label = c("low", "medium", "high"))
egen(data, x, at = c(3, 7), label = c("low", "medium", "high"), new_var = "group")
```

`encode` *Encode variables in a data frame*

Description

The `encode()` function encodes selected variables in a data frame. It converts the variables to numeric type, sets labels from the original data, and returns the modified data frame.

Usage

```
encode(.data, ...)
```

Arguments

`.data`	The input data frame.
`...`	The variables to be encoded.

Value

A modified data frame with the selected variables encoded.

Examples

```
# Encode selected variables in the `mtcars` data frame
encode(mtcars, mpg, cyl)
```

label *Attach labels to data and variables*

Description

[Stable]

This function manipulates labels. It supports different classes of objects, including default objects, data frames, and other types.

Usage

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

Arguments

`x`	The object to which the label will be added or modified.
`label`	A character string specifying the label to be assigned to the variable.

Details

When used with `dplyr`'s `[mutate]` function, this function allows for easy labeling of variables within a data frame.

If used with a data frame, the function labels the dataset itself, and the label can be checked using the `[codebook]` function.

Value

The modified object with the updated label.

Examples

```
library(dplyr)

iris |>
  mutate(Species = label(Species, 'Species of iris flower')) |>
  codebook()

iris |>
  label("Iris dataset") |>
  codebook()
```

tag_duplicates *Tag Duplicates*

Description

[Stable]

Usage

```
tag_duplicates(..., .add_tags = TRUE)
```

Arguments

`...`	Columns to use for identifying duplicates.
`.add_tags`	logical to return three indicator columns: `.n_`, `.N_`, and `.dup_`.

Details

This function identifies and tags duplicate observations based on specified variables.

This function mimics the functionality of Stata's `duplicates` command in R. It calculates the number of duplicates and provides a report of duplicates based on the specified variables. The function utilizes the n_ and N_ functions for counting and grouping the observations.

Value

A tibble with three columns: `.n_`, `.N_`, and `.dup_`.

- `.n_` represents the running counter within each group of variables, indicating the number of the current observation.
- `.N_` represents the total number of observations within each group of variables.
- `.dup_` is a logical column indicating whether the observation is a duplicate (TRUE) or not (FALSE).

See Also

Other Data Management: `append()`, `codebook()`, `count_functions`, `cut()`

Examples

```
library(dplyr)

# Example with a custom dataset
data <- data.frame(
  x = c(1, 1, 2, 2, 3, 4, 4, 5),
  y = letters[1:8]
)

# Identify and tag duplicates based on the "x" variable
```

```
data %>% mutate(tag_duplicates(x))

# Identify and tag duplicates based on multiple variables
data %>% mutate(tag_duplicates(x, y))

# Identify and tag duplicates based on all variables
data %>% mutate(tag_duplicates(everything()))

## Not run:
## STATA example
dupxmpl <- haven::read_dta("https://www.stata-press.com/data/r18/dupxmpl.dta")
dupxmpl |> mutate(tag_duplicates(everything()))

## End(Not run)
```

Index