Package: lest (via r-universe)

September 6, 2024

Type Package

Title Vectorised Nested if-else Statements Similar to CASE WHEN in 'SQL'

Version 1.1.0

Maintainer Stefan Fleck <stefan.b.fleck@gmail.com>

Description Functions for vectorised conditional recoding of variables. case_when() enables you to vectorise multiple if and else statements (like 'CASE WHEN' in 'SQL'). if_else() is a stricter and more predictable version of ifelse() in 'base' that preserves attributes. These functions are forked from 'dplyr' with all package dependencies removed and behave identically to the originals.

License MIT + file LICENSE

Suggests testthat

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.0.1.9000

Repository https://s-fleck.r-universe.dev

RemoteUrl https://github.com/s-fleck/lest

RemoteRef HEAD

RemoteSha 2a0bb22ea7ffad5b1d07bb8eb092ff300a7f0dcf

Contents

case_when *A general vectorised if*

Description

This function allows you to vectorise multiple `if` and `else if` statements. It is an R equivalent of the SQL `CASE WHEN` statement.

Usage

```
case_when(...)
```

Arguments

`...`	A sequence of two-sided formulas. The left hand side (LHS) determines which values match this case. The right hand side (RHS) provides the replacement value. The LHS must evaluate to a logical vector. The RHS does not need to be logical, but all RHSs must evaluate to the same type of vector. Both LHS and RHS may have the same length of either 1 or `n`. The value of `n` must be consistent across all cases. The case of `n == 0` is treated as a variant of `n != 1`.

Value

A vector of length 1 or `n`, matching the length of the logical input or output vectors, with the type (and attributes) of the first RHS. Inconsistent lengths or types will generate an error.

Examples

```
x <- 1:50
case_when(
  x %% 35 == 0 ~ "fizz buzz",
  x %% 5 == 0 ~ "fizz",
  x %% 7 == 0 ~ "buzz",
  TRUE ~ as.character(x)
)

# Like an if statement, the arguments are evaluated in order, so you must
# proceed from the most specific to the most general. This won't work:
case_when(
  TRUE ~ as.character(x),
  x %%  5 == 0 ~ "fizz",
  x %%  7 == 0 ~ "buzz",
  x %% 35 == 0 ~ "fizz buzz"
)

# All RHS values need to be of the same type. Inconsistent types will throw an error.
# This applies also to NA values used in RHS: NA is logical, use
```

```
# typed values like NA_real_, NA_complex, NA_character_, NA_integer_ as appropriate.
case_when(
  x %% 35 == 0 ~ NA_character_,
  x %% 5 == 0 ~ "fizz",
  x %% 7 == 0 ~ "buzz",
  TRUE ~ as.character(x)
)
case_when(
  x %% 35 == 0 ~ 35,
  x %% 5 == 0 ~ 5,
  x %% 7 == 0 ~ 7,
  TRUE ~ NA_real_
)
# This throws an error as NA is logical not numeric
try({
case_when(
  x %% 35 == 0 ~ 35,
  x %% 5 == 0 ~ 5,
  x %% 7 == 0 ~ 7,
  TRUE ~ NA
)
})
dat <- iris[1:5, ]
dat$size <- case_when(
  dat$Sepal.Length < 5.0 ~ "small",
  TRUE ~ "big"
)
dat
```

`cumall` *Cumulative all and any*

Description

Cumulative all and any

Usage

```
cumall(x)

cumany(x)
```

Arguments

`x`	a `logical` vector.

Value

a `logical` vector

Examples

```
cumall(c(TRUE, TRUE, NA, TRUE, FALSE))
cumany(c(FALSE, FALSE, NA, TRUE, FALSE))
```

`exceeds_tumbling_sum` *Check When the Tumbling Sum of a Vector Exceeds a Threshold*

Description

The tumbling sum is calculated as the partial cumulative sum of a vector until a threshold is exceeded. Once this happens, the tumbling sum is calculated from zero again. `exceeds_tumbling_sum()` returns `TRUE` whenever this threshold is hit/exceeded and `FALSE` otherwise.

Usage

```
exceeds_tumbling_sum(x, threshold, inclusive = TRUE)
```

Arguments

`x`	a `numeric` vector
`threshold`	a `numeric` scalar
`inclusive`	a `logical` scalar. If `TRUE` inclusive bounds are used (i.e. the threshold is checked with >=), otherwise exclusive

Details

This is for example useful if you have high frequency GPS positions and want to keep only points that are at least `x` seconds apart.

Value

a `logical` vector of the same length as `x` that is `TRUE` whenever `threshold` was exceeded and `FALSE` otherwise

See Also

`MESS::cumsumbinning()` does something very similar, but returns group indices instead of a logical vector.

Examples

```
exceeds_tumbling_sum(c(1, 3, 3, 3), 4)
```

`if_else` *Vectorised if*

Description

Compared to the base `ifelse()`, this function is more strict. It checks that `true` and `false` are the same type. This strictness makes the output type more predictable, and makes it somewhat faster.

Usage

```
if_else(condition, true, false, missing = NULL)
```

Arguments

`condition`	Logical vector
`true`, `false`	Values to use for `TRUE` and `FALSE` values of `condition`. They must be either the same length as `condition`, or length 1. They must also be the same type: `if_else()` checks that they have the same type and same class. All other attributes are taken from `true`.
`missing`	If not `NULL`, will be used to replace missing values.

Value

Where `condition` is `TRUE`, the matching value from `true`, where it's `FALSE`, the matching value from `false`, otherwise `NA`.

Examples

```
x <- c(-5:5, NA)
if_else(x < 0, NA_integer_, x)
if_else(x < 0, "negative", "positive", "missing")

# Unlike ifelse, if_else preserves types
x <- factor(sample(letters[1:5], 10, replace = TRUE))
ifelse(x %in% c("a", "b", "c"), x, factor(NA))
if_else(x %in% c("a", "b", "c"), x, factor(NA))
# Attributes are taken from the `true` vector,
```

Index