Package ‘ympes’

November 1, 2022

Type Package

Title Collection of Helper Functions

Version 0.2.1

Description Provides a collection of lightweight helper functions (imps) both for interactive use and for inclusion within other packages. These include minimal assertion functions with a focus on informative error messaging for both missing and incorrect function arguments as well as other functions for visualising colour palettes, quoting user input and working with age intervals.

License GPL-2

Encoding UTF-8

RoxygenNote 7.2.1

Suggests clipr, knitr, rmarkdown, tinytest

Depends R (>= 3.5.0)

LazyData true

VignetteBuilder knitr

URL `https://timtaylor.github.io/ympes/`

BugReports `https://github.com/TimTaylor/ympes/issues`

NeedsCompilation yes

Author Tim Taylor [aut, cre, cph] (<`https://orcid.org/0000-0002-8587-7113`>)

Maintainer Tim Taylor <`tim.taylor@hiddenelephants.co.uk`>

Repository CRAN

Date/Publication 2022-11-01 13:50:02 UTC

R topics documented:

ageutils *Utilities for Age Intervals*

Description

This help page documents the utility functions provided for working with age intervals:

- `ages_to_interval()` provides categorisation of ages based on specified right-hand interval limits. The resultant groupings will span the natural numbers (from 0) and will always be closed on the left and open on the right. For example, if `limits = c(1,10,30)` the possible groupings will be "[0, 1)", "[1, 10)", "[10, 30)" and "[30, Inf)". This is roughly comparable to a call of `cut(ages, right = FALSE, breaks = c(0, limits))` but with the start and end points of the interval returned as entries in a list.
- `split_interval_counts()` splits counts within a age interval in to counts for individuals years based on a given weighting. Age intervals are specified by their lower (closed) and upper (open) bounds, i.e. intervals of the form [lower, upper).
- `aggregate_age_counts()` provides aggregation of counts across ages (in years). It is similar to a `cut()` and `tapply()` pattern but optimised for speed over flexibility. Groupings are the same as in `ages_to_interval()` and counts will be provided across all natural numbers as well as for missing values.
- `reaggregate_interval_counts()` is equivalent to, but more efficient than, calling `split_interval_counts()` and then `aggregate_age_counts()`.

Usage

```
ages_to_interval(ages, limits = c(1L, 5L, 15L, 25L, 45L, 65L))

split_interval_counts(
  lower_bounds,
  upper_bounds,
  counts,
  max_upper = 100L,
  weights = NULL
)

aggregate_age_counts(
  counts,
  ages = 0:(length(counts) - 1L),
  limits = c(1L, 5L, 15L, 25L, 45L, 65L)
)

reaggregate_interval_counts(
  lower_bounds,
  upper_bounds,
  counts,
  limits = c(1L, 5L, 15L, 25L, 45L, 65L),
  max_upper = 100L,
  weights = NULL
)
```

Arguments

`ages`	`[integerish]`. Vector of age in years. Double values will be coerced to integer prior to categorisation / aggregation. For `aggregate_age_counts()`, these must corresponding to the `counts` entry and will defaults to 0:(N-1) where `N` is the number of counts present. `ages` >= 200 are not permitted due to the internal implementation.
`limits`	`[integerish]`. 1 or more positive cut points in increasing (strictly) order. Defaults to c(1L,5L,15L,25L,45L,65L). Double values will be coerced to integer prior to categorisation.
`lower_bounds, upper_bounds`	`[integerish]`. A pair of vectors representing the bounds of the intervals. `lower_bounds` must be strictly less than `upper_bounds` and greater than or equal to zero. Missing (NA) bounds are not permitted. Double vectors will be coerced to integer.
`counts`	`[numeric]`. Vector of counts to be aggregated.
`max_upper`	`[integerish]` Represents the maximum upper bounds permitted upon splitting the data. Used to replace `Inf` upper bounds prior to splitting. If any `upper_bound` is greater than `max_upper` the function will error. Double vectors will be coerced to integer.
`weights`	`[numeric]` Population weightings to apply for individual years. If `NULL` (default) counts will be split evenly based on interval size. If specified, must be of length `max_upper` and represent weights in the range 0:(max_upper - 1).

Value

- `ages_to_interval()`. A data frame with an ordered factor column (`interval`), as well as columns corresponding to the explicit bounds (`lower_bound` and `upper_bound`).
- `split_interval_counts()`. A data frame with entries `age` (in years) and `count`.
- `aggregate_age_counts()` and `reaggregate_interval_counts()`. A data frame with 4 entries; `interval`, `lower_bound`, `upper_bound` and an associated `count`.

Examples

```
# limits are set to c(1L,5L,15L,25L,45L,65L) by default
ages_to_interval(ages = 0:9, limits = c(3L, 5L, 10L))
ages_to_interval(ages = 0:9)
ages_to_interval(ages = 0:9, limits = c(1L, 5L, 15L, 25L, 45L, 65L))

split_interval_counts(
    lower_bounds = c(0, 5, 10),
    upper_bounds = c(5, 10, 20),
    counts = c(5, 10, 30)
)

# default ages generated if only counts provided (here ages will be 0:64)
aggregate_age_counts(counts = 1:65, limits = c(1L, 5L, 15L, 25L, 45L, 65L))
aggregate_age_counts(counts = 1:65, limits = 50)

# NA ages are handled with their own grouping
ages <- 1:65;
ages[1:44] <- NA
aggregate_age_counts(
    counts = 1:65,
    ages = ages,
    limits = c(1L, 5L, 15L, 25L, 45L, 65L)
)

reaggregate_interval_counts(
    lower_bounds = c(0, 5, 10),
    upper_bounds = c(5, 10, 20),
    counts = c(5, 10, 30),
    limits = c(1L, 5L, 15L, 25L, 45L, 65L)
)
```

`assertions` *Argument assertions*

Description

Assertions for function arguments. Motivated by `vctrs::vec_assert()` but with lower overhead at a cost of less informative error messages. Designed to make it easy to identify the top level calling function whether used within a user facing function or internally.

Usage

```
imp_assert_integer(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_int(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_double(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_dbl(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_numeric(x, arg = deparse(substitute(x)), call = sys.call(-1L))
```

```
imp_assert_num(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_logical(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_lgl(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_character(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_chr(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_data_frame(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_list(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_scalar_integer(
  x,
  arg = deparse(substitute(x)),
  call = sys.call(-1L)
)

imp_assert_scalar_int(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_scalar_double(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_scalar_dbl(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_scalar_numeric(
  x,
  arg = deparse(substitute(x)),
  call = sys.call(-1L)
)

imp_assert_scalar_num(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_scalar_logical(
  x,
  arg = deparse(substitute(x)),
  call = sys.call(-1L)
)

imp_assert_scalar_lgl(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_bool(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_boolean(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_scalar_character(
  x,
  arg = deparse(substitute(x)),
  call = sys.call(-1L)
)

imp_assert_scalar_chr(x, arg = deparse(substitute(x)), call = sys.call(-1L))

imp_assert_string(x, arg = deparse(substitute(x)), call = sys.call(-1L))
```

Arguments

`x`	Argument to check.
`arg`	Name of argument being checked (used in error message).
`call`	Call to use in error message.

Value

The input argument (invisibly) if the assertion succeeds (error otherwise).

Examples

```
# Use in a user facing function
fun <- function(i, d, l, chr, b) {
    imp_assert_scalar_int(i)
    TRUE
}
fun(i=1L)
try(fun())
try(fun(i="cat"))

# Use in an internal function
internal_fun <- function(a) {
    imp_assert_string(a, arg = deparse(substitute(a)), call = sys.call(-1L))
    TRUE
}
external_fun <- function(b) {
    internal_fun(a=b)
}
external_fun(b="cat")
try(external_fun())
try(external_fun(a = letters))
```

`cc` *Quote names*

Description

`cc()` quotes comma separated names whilst trimming outer whitespace. It is intended for interactive use only.

Usage

```
cc(..., .clip = getOption("imp.clipboard", FALSE))
```

Arguments

`...`	Unquoted names (separated by commas) that you wish to quote; empty arguments (e.g. third item in `one,two,,four`) will be returned as `""`.
`.clip`	Should the code to generate the constructed character vector be copied to your system clipboard; defaults to FALSE unless the option "imp.clipboard" is set to TRUE.

Value

A character vector of the quoted input.

Note

Copying to clipboard requires the availability of package clipr.

Examples

```
cc(dale, audrey, laura, hawk)
```

`plot_palette` *Plot a colour palette*

Description

`plot_palette()` plots a palette from a vector of colour values (name or hex).

Usage

```
plot_palette(values, label = TRUE, square = FALSE)
```

Arguments

`values`	character vector of named or hex colours.
`label`	boolean. Do you want to label the plot or not? If `values` is a named vector the names are used for labels, otherwise, the values.
`square`	boolean. Display palette as square?

Value

The input (invisibly).

Examples

```
plot_palette(c("#5FE756", "red", "black"))
plot_palette(c("#5FE756", "red", "black"), square=TRUE)
```

`pop_dat` *Aggregated population data*

Description

A dataset derived from the 2021 UK census containing population for different age categories across England and Wales.

Usage

```
pop_dat
```

Format

A data frame with 200 rows and 6 variables:

area_code Unique area identifier

area_name Unique area name

age_category Left-closed and right-open age interval

value count of individ

Source

https://github.com/TimTaylor/census_pop_2021

Index