Package ‘utile.tools’

February 20, 2022

Title Summarize Data for Publication

Version 0.2.7

Description A set of tools for preparing and summarizing data for publication purposes. Includes functions for tabulating models, means to produce human-readable summary statistics from raw data, macros for calculating duration of time, and simplistic hypothesis testing tools.

License LGPL (>= 2)

URL https://github.com/efinite/utile.tools

BugReports https://github.com/efinite/utile.tools/issues

Encoding UTF-8

Depends R (>= 3.4.0)

Imports lubridate, purrr, vctrs

Suggests survival, dplyr

RoxygenNote 7.1.2

NeedsCompilation no

Author Eric Finnesgard [aut, cre]

Maintainer Eric Finnesgard <efinite@outlook.com>

Repository CRAN

Date/Publication 2022-02-20 21:40:02 UTC

R topics documented:

`calc_chunks` *Calculate data chunk indices*

Description

Calculates chunk indices of a data object for a given chunk size (number of items per chunk).

Usage

```
calc_chunks(x, size = 10, reverse = FALSE)
```

Arguments

`x`	A data frame or vector.
`size`	An integer. The number of items (e.g. rows in a tibble) that make up a given chunk. Must be a positive integer. Caps out at data maximum.
`reverse`	A logical. Calculate chunks from back to front.

Value

An iterable list of row indices for each chunk of data.

Examples

```
# Create chunk map for a data frame
chunks <- calc_chunks(mtcars, size = 6)

# Iterate through chunks of data
for (chunk in chunks) print(paste0(rownames(mtcars[chunk,]), collapse = ', '))
```

`calc_cumsum` *Calculate cumulative summation of vector*

Description

Calculate the cumulative summation of a numeric vector with revised NA handling compared to ‘base::cumsum()‘.

Usage

```
calc_cumsum(x, na.fill = FALSE)
```

Arguments

`x`	A numeric.
`na.fill`	A logical. Impute forward for NA values.

Value

A vector of the same length and type as x.

Examples

```
x <- 1:10
x[3:4] <- NA

# Calculate cumsum & replace NA values
calc_cumsum(x, na.fill = TRUE)
```

`calc_duration` *Calculate durations of time*

Description

Calculates the duration of time between two provided date objects. Supports vectorized data (i.e. `dplyr::mutate()`).

Usage

```
calc_duration(x, y, units = NULL)
```

Arguments

`x`	A date or datetime. The start date(s)/timestamp(s).
`y`	A date or datetime. The end date(s)/timestamp(s).
`units`	A character. Units of the returned duration (i.e. 'seconds', 'days', 'years').

Value

If 'units' specified, returns numeric. If 'units' unspecified, returns an object of class '`Duration`'.

Note

Supports multiple calculations against a single time point (i.e. multiple start dates with a single end date). Note that start and end must otherwise be of the same length.

When the start and end dates are of different types (i.e. x = date, y = datetime), a lossy cast will be performed which strips the datetime data of its time components. This is done to avoid an assumption of more time passing that would otherwise come with casting the date data to datetime.

Examples

```
library(lubridate)
library(purrr)

# Dates -> duration in years
calc_duration(
  x = mdy(map_chr(sample(1:9, 5), ~ paste0('01/01/199', .x))),
  y = mdy(map_chr(sample(1:9, 5), ~ paste0('01/01/200', .x))),
  units = 'years'
)

# datetimes -> durations
calc_duration(
  x = mdy_hm(map_chr(sample(1:9, 5), ~ paste0('01/01/199', .x, ' 1', .x, ':00'))),
  y = mdy_hm(map_chr(sample(1:9, 5), ~ paste0('01/01/200', .x, ' 0', .x, ':00')))
)

# Mixed date classes -> durations
calc_duration(
  x = mdy(map_chr(sample(1:9, 5), ~ paste0('01/01/199', .x))),
  y = mdy_hm(map_chr(sample(1:9, 5), ~ paste0('01/01/200', .x, ' 0', .x, ':00')))
)
```

`chunk_data_` *Break data into chunks*

Description

Creates a factory function which returns a different chunk of a given data object with each function call.

Usage

```
chunk_data_(x, size = 10, reverse = FALSE)
```

Arguments

`x`	A data frame or vector.
`size`	An integer. The number of items (e.g. rows in a tibble) that make up a given chunk. Must be a positive integer.
`reverse`	A logical. Calculate chunks from back to front.

Value

A factory function which returns a chunk of data from the provided object with each call. Once all data has been returned, function returns NULL perpetually.

Examples

```
# Create chunk factory function
chunked_data <- chunk_data_(mtcars, size = 6)

# Chunk #1 (rows 1-6)
paste0(rownames(chunked_data()), collapse = ', ')

# Chunk #2 (rows 7-12)
paste0(rownames(chunked_data()), collapse = ', ')
```

`paste` *Concatenate strings*

Description

An augmented version of `base::paste()` with options to manage NA values.

Usage

```
paste(..., sep = " ", collapse = NULL, na.rm = FALSE)

paste0(..., collapse = NULL, na.rm = FALSE)
```

Arguments

`...`	R objects to be converted to character vectors.
`sep`	A character. A string to separate the terms.
`collapse`	A character. An string to separate the results.
`na.rm`	A logical. Whether to remove NA values from 'x'. Note that NA values are also removed from vectors.

Details

The `base::paste()` function is intentionally designed to coarce NA values to characters that appear in the concatenated character output. This behavior is not always desirable (i.e. when programatically calling paste) and there is currently no means of opting out of this behavior. These augmented functions address this deficit.

Value

Character vector of concatenated values.

See Also

`paste`

Examples

```
# Base paste() NA handling behavior
paste(
  'The', c('red', NA, 'orange'), 'fox jumped', NA, 'over the fence.',
  collapse = ' '
)

# Removal of NA values
paste(
  'The', c('red', NA, 'orange'), 'fox jumped', NA, 'over the fence.',
  collapse = ' ',
  na.rm = TRUE
)
```

`paste_efs` *Paste event-free survival*

Description

Creates a human-readable event-free-survival from a survfit object and a specified time point.

Usage

```
paste_efs(x, times, percent.sign = TRUE, digits = 1)
```

Arguments

`x`	A `survfit` object. The survival model.
`times`	A numeric. Indicates time-points of interest. Units are whatever was used to create the survival fit.
`percent.sign`	A logical. Indicates percent sign should be printed for frequencies.
`digits`	Integer. Number of digits to round to.

Value

A character vector of event free survival(s).

Examples

```
library(survival)

fit <- survfit(Surv(time, status) ~ 1, data = diabetic)
paste_efs(fit, c(1, 3, 5))
```

`paste_freq` *Paste frequency*

Description

Creates a human-readable frequency from count(able) data. Automatically tallies non-numeric data types (nrow or length) and supports vectorized data methods.

Usage

```
paste_freq(x, y, na.rm = TRUE, percent.sign = TRUE, digits = 1)
```

Arguments

`x`	A data.frame, numeric, or non-numeric. The numerator.
`y`	A data.frame, numeric, or non-numeric. The denominator. A single denominator may be used for multiple numerators or one denominator for each numerator.
`na.rm`	A logical. Whether to ignore NA's when tallying non-numeric data.
`percent.sign`	A logical. Indicates percent sign should be printed with frequencies.
`digits`	An integer. Number of digits to round to.

Value

A character vector of count(s) with frequencies.

Examples

```
# Numeric
paste_freq(20, 100)

# data.frame
df <- data.frame(x = c(1:100), y = TRUE)
paste_freq(df[1:20,], df)

# Mixed data types
paste_freq(20, df)

# Single denominator for multiple numerators
paste_freq(c(10,20,30), 100)
```

paste_mean *Paste mean*

Description

Creates a human-readable mean with standard deviation from numeric data.

Usage

```
paste_mean(x, less.than.one = FALSE, digits = 1)
```

Arguments

`x`	A numeric. Data to summarize.
`less.than.one`	A logical. Indicates a mean that rounds to 0 should be printed as <1.
`digits`	An integer. Number of digits to round to.

Value

A character vector of the mean(s) with standard deviation(s).

Examples

```
paste_mean(mtcars$mpg)
```

paste_median *Paste median*

Description

Creates a human-readable median with inter-quartile range from numeric data.

Usage

```
paste_median(x, less.than.one = FALSE, digits = 1)
```

Arguments

`x`	A numeric. Data to summarize.
`less.than.one`	A logical. Indicates a median that rounds to 0 should be printed as <1.
`digits`	An integer. Number of digits to round to.

Value

A character vector of the median(s) with interquartile range(s).

Examples

```
paste_median(mtcars$mpg)
```

`test_hypothesis` *Test the null hypothesis*

Description

Produces a p-value from parametric or non-parametric testing of the null hypothesis for stratified data.

Usage

```
test_hypothesis(x, y, parametric, digits, p.digits)
```

Arguments

`x`	A numeric or factor. Observations.
`y`	A factor or logical. Categorical variable to stratify by.
`parametric`	A logical. Indicates parametric testing should be used. See note detailing statistical tests. Defaults to FALSE.
`digits`	An integer. Number of digits to round to. Defaults to 1.
`p.digits`	An integer. Number of p-value digits to print. Note that p-values are still rounded using 'digits'. Defaults to 4.

Value

A character formatted p-value.

Note

Statistical testing used is dependent on type of 'x' data, number of levels in the factor 'y', and whether parametric/non-parametric testing is selected. For continuous 'x' data, parametric testing is Student's t-test (y, 2 lvls) or one-way ANOVA (y, >2 lvls). Non-parametric testing is Wilcoxon Rank Sum/Mann Whitney U (y, 2 lvls) or Kruskal Wallis (y, >2 lvls). For factor or logical 'x' data, "parametric" testing is Chi-squared with (x, 2 lvls) and without (x, >2 lvls) Monte Carlo simulation. Non-parametric testing is the Fisher's exact test with (x, 2 lvls) and without (x, >2 lvls) Monte Carlo simulation.

Examples

```
# Numeric data
test_hypothesis(mtcars$mpg, as.factor(mtcars$gear))

# Logical data
test_hypothesis(as.logical(mtcars$vs), as.factor(mtcars$gear))

# Factor data
test_hypothesis(as.factor(mtcars$carb), as.factor(mtcars$gear))
```

Index