

# Package ‘rtemisutils’

February 7, 2026

**Version** 0.1.0

**Title** Shared Utilities for Rtemis Packages

**Date** 2026-02-01

**Description** Includes a collection of type-checking and text-formatting utilities as well as the rtemis color system. This package is not intended for direct use by users, but rather to be imported by other rtemis packages.

**License** GPL (>= 3)

**URL** <https://www.rtemis.org>

**BugReports** <https://github.com/rtemis-org/rtemisutils/issues>

**ByteCompile** yes

**Depends** R (>= 4.1.0)

**Imports** cli, data.table, grDevices, graphics, htmltools, S7, stats,  
methods, utils

**Suggests** car, DBI, duckdb, future, gsubfn, igraph, parallelly, plotly,  
reactable, reticulate, seqinr, testthat, timeDate

**Encoding** UTF-8

**Config/testthat/edition** 3

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**NeedsCompilation** no

**Author** E.D. Gennatas [aut, cre, cph] (ORCID:  
<<https://orcid.org/0000-0001-9280-3609>>)

**Maintainer** E.D. Gennatas <gennatas@gmail.com>

## Contents

rtemisutils-package . . . . .	3
ansi256_to_hex . . . . .	4
bold . . . . .	4
check_dependencies . . . . .	5
check_inherits . . . . .	6
clean_colnames . . . . .	6
clean_int . . . . .	7

clean_names . . . . .	8
col256 . . . . .	8
col2grayscale . . . . .	9
col2hex . . . . .	10
colorgrad . . . . .	10
color_adjust . . . . .	13
color_fade . . . . .	13
color_invertRGB . . . . .	14
color_mix . . . . .	14
color_op . . . . .	15
color_txt_columns . . . . .	16
ddSci . . . . .	17
desaturate . . . . .	18
df_movecolumn . . . . .	18
drange . . . . .	19
dt_describe . . . . .	20
dt_inspect_types . . . . .	21
dt_keybin_reshape . . . . .	21
dt_merge . . . . .	22
dt_names_by_attr . . . . .	24
dt_nunique_perfeat . . . . .	24
dt_pctmatch . . . . .	25
dt_pctmissing . . . . .	26
dt_set_autotypes . . . . .	26
dt_set_cleanfactorlevels . . . . .	27
dt_set_clean_all . . . . .	28
dt_set_logical2factor . . . . .	29
export_plotly . . . . .	30
factor_NA2missing . . . . .	31
fct_describe . . . . .	31
filter_order . . . . .	32
fmt . . . . .	33
fmt_gradient . . . . .	34
getnames . . . . .	35
getnamesandtypes . . . . .	36
get_loaded_pkg_version . . . . .	36
get_mode . . . . .	37
get_output_type . . . . .	37
get_vars_from_rules . . . . .	38
graph_node_metrics . . . . .	39
gray . . . . .	39
green . . . . .	40
highlight . . . . .	40
iflengthy . . . . .	41
index_col_by_attr . . . . .	42
init_project_dir . . . . .	42
inspect_type . . . . .	43
is_constant . . . . .	44
is_discrete . . . . .	44
italic . . . . .	45
labelify . . . . .	45
list2csv . . . . .	46

lotri2edgeList . . . . .	47
make_path . . . . .	48
matchcases . . . . .	48
mgetnames . . . . .	50
msg . . . . .	51
msgdatetime . . . . .	52
msgdone . . . . .	52
msgstart . . . . .	53
muted . . . . .	53
names_by_class . . . . .	54
orange . . . . .	55
pcat . . . . .	55
previewcolor . . . . .	56
printdf . . . . .	57
printls . . . . .	58
qstat . . . . .	59
recycle . . . . .	60
repr . . . . .	60
repr_ls . . . . .	61
repr_S7name . . . . .	62
rnormmat . . . . .	63
rtemis_colors . . . . .	64
rtpalette . . . . .	65
rtversion . . . . .	65
rt_letters . . . . .	66
rt_reactable . . . . .	66
rules2medmod . . . . .	67
runifmat . . . . .	68
setdiffsym . . . . .	69
sge_submit . . . . .	69
show_pad . . . . .	71
show_range . . . . .	71
size . . . . .	72
table_column_attr . . . . .	72
thin . . . . .	73
underline . . . . .	74
uniprot_get . . . . .	74
uniquevalsperfeat . . . . .	75
vec2df . . . . .	76
xtdescribe . . . . .	76
<b>Index</b>	<b>78</b>

---

rtemisutils-package	<b>rtemisutils:</b> <i>Rtemis Utilities</i>
---------------------	---

---

## Description

Utilities for Rtemis R Packages

**Author(s)**

**Maintainer:** E.D. Gennatas <gennatas@gmail.com> (**ORCID**) [copyright holder]

**See Also**

Useful links:

- <https://www.rtemis.org>
- Report bugs at <https://github.com/rtemis-org/rtemisutils/issues>

---

ansi256\_to\_hex

*Convert ANSI 256 color code to HEX*


---

**Description**

Convert ANSI 256 color code to HEX

**Usage**

```
ansi256_to_hex(code)
```

**Arguments**

code                      Integer: ANSI 256 color code (0-255).

**Value**

Character: HEX color string.

**Author(s)**

EDG

---

bold

*Make text bold*


---

**Description**

A `fmt()` convenience wrapper for making text bold.

**Usage**

```
bold(text, output_type = c("ansi", "html", "plain"))
```

**Arguments**

text                      Character: Text to make bold  
output\_type              Character: Output type ("ansi", "html", "plain")

**Value**

Character: Formatted text with bold styling

**Author(s)**

EDG

---

check_dependencies	<b>rtemis</b> <i>internal: Dependencies check</i>
--------------------	---

---

**Description**

Checks if dependencies can be loaded; names missing dependencies if not.

**Usage**

```
check_dependencies(..., verbosity = 0L)
```

**Arguments**

...	List or vector of strings defining namespaces to be checked
verbosity	Integer: Verbosity level. Note: An error will always printed if dependencies are missing. Setting this to FALSE stops it from printing "Dependencies check passed".

**Value**

Called for side effects. Aborts and prints list of missing dependencies, if any.

**Author(s)**

EDG

**Examples**

```
# This will throw an error if "unavailable" is not installed:  
# check_dependencies("unavailable")
```

---

check_inherits	<i>Test class of object</i>
----------------	-----------------------------

---

**Description**

Test class of object

**Usage**

```
check_inherits(x, cl, allow_null = TRUE, xname = deparse(substitute(x)))
```

**Arguments**

x	Object to check.
cl	Character: class to check against.
allow_null	Logical: If TRUE, NULL values are allowed and return early.

**Details**

Exported as internal function for use by other rtemis packages.

**Value**

Called for side effects. Throws an error if checks fail.

**Author(s)**

EDG

---

clean_colnames	<i>Clean column names</i>
----------------	---------------------------

---

**Description**

Clean column names by replacing all spaces and punctuation with a single underscore

**Usage**

```
clean_colnames(x)
```

**Arguments**

x	Character vector or matrix with colnames or any object with names() method.
---	---

**Value**

Character vector.

**Author(s)**

EDG

**Examples**

```
clean_colnames(iris)
```

---

`clean_int`*Clean integer input*

---

**Description**

Clean integer input

**Usage**

```
clean_int(x, xname = deparse(substitute(x)))
```

**Arguments**

<code>x</code>	Double or integer vector to check.
<code>xname</code>	Character: Name of <code>x</code> , for error messages.

**Details**

The goal is to return an integer vector. If the input is integer, it is returned as is. If the input is numeric, it is coerced to integer only if the numeric values are integers, otherwise an error is thrown.

**Value**

Integer vector

**Author(s)**

EDG

**Examples**

```
## Not run:
clean_int(6L)
clean_int(3)
clean_int(12.1) # Error
clean_int(c(3, 5, 7))
clean_int(c(3, 5, 7.01)) # Error

## End(Not run)
```

---

clean_names	<i>Clean names</i>
-------------	--------------------

---

**Description**

Clean character vector by replacing all symbols and sequences of symbols with single underscores, ensuring no name begins or ends with a symbol

**Usage**

```
clean_names(x, prefix_digits = "V_")
```

**Arguments**

x	Character vector.
prefix_digits	Character: prefix to add to names beginning with a digit. Set to NA to skip.

**Value**

Character vector.

**Author(s)**

EDG

**Examples**

```
x <- c("Patient ID", "_Date-of-Birth", "SBP (mmHg)")
x
clean_names(x)
```

---

col256	<i>Apply 256-color formatting</i>
--------	-----------------------------------

---

**Description**

Apply 256-color formatting

**Usage**

```
col256(text, col = "79", bg = FALSE, output_type = c("ansi", "html", "plain"))
```

**Arguments**

text	Character: Text to color
col	Character or numeric: Color (ANSI 256-color code, hex for HTML)
bg	Logical: If TRUE, apply as background color
output_type	Character: Output type ("ansi", "html", "plain")

**Value**

Character: Formatted text with 256-color styling

**Author(s)**

EDG

---

col2grayscale	<i>Color to Grayscale</i>
---------------	---------------------------

---

**Description**

Convert a color to grayscale

**Usage**

```
col2grayscale(x, what = c("color", "decimal"))
```

**Arguments**

x	Color to convert to grayscale
what	Character: "color" returns a hexadecimal color, "decimal" returns a decimal between 0 and 1

**Details**

Uses the NTSC grayscale conversion:  $0.299 * R + 0.587 * G + 0.114 * B$

**Value**

Character: color hex code.

**Author(s)**

EDG

**Examples**

```
## Not run:  
col2grayscale("red")  
col2grayscale("red", "dec")  
  
## End(Not run)
```

---

col2hex	<i>Convert R color to hexadecimal code</i>
---------	--

---

**Description**

Convert a color that R understands into the corresponding hexadecimal code

**Usage**

```
col2hex(color)
```

**Arguments**

color	Color(s) that R understands
-------	-----------------------------

**Value**

Character vector of hexadecimal codes.

**Author(s)**

EDG

**Examples**

```
## Not run:  
col2hex(c("gray50", "skyblue"))  
  
## End(Not run)
```

---

colorgrad	<i>Color Gradient</i>
-----------	-----------------------

---

**Description**

Create a gradient of colors and optionally a colorbar

**Usage**

```
colorgrad(  
  n = 21,  
  colors = NULL,  
  space = c("rgb", "Lab"),  
  lo = "#18A3AC",  
  lomid = NULL,  
  mid = NULL,  
  midhi = NULL,  
  hi = "#F48024",  
  preview = FALSE,  
  colorbar = FALSE,
```

```

cb_n = 21,
cb_mar = c(1, 1, 1, 1),
cb_add = FALSE,
cb_add_mar = c(5, 0, 2, 5),
cb_axis_pos = 1.1,
cb_axis_las = 1,
cb_axis_hadj = 0,
cb_cex = 6,
bar_min = -1,
bar_mid = 0,
bar_max = 1,
cex = 1.2,
filename = NULL,
pdf_width = 3,
pdf_height = 7,
theme = getOption("rt.theme", "light"),
bg = NULL,
col_text = NULL,
plotlycb = FALSE,
plotly_width = 80,
plotly_height = 500,
return_plotly = FALSE,
margins = c(0, 0, 0, 0),
pad = 0,
par_reset = TRUE
)

```

### Arguments

n	Integer: How many distinct colors you want. If not odd, converted to $n + 1$ . Defaults to 21
colors	Character: Acts as a shortcut to defining lo, mid, etc for a number of defaults: "french", "penn", "grnblkred",
space	Character: Which colorspace to use. Option: "rgb", or "Lab". Recommendation: If mid is "white" or "black" (default), use "rgb", otherwise "Lab"
lo	Color for low end
lomid	Color for low-mid
mid	Color for middle of the range or "mean", which will result in <code>color_op(c(lo, hi), "mean")</code> . If <code>mid = NA</code> , then only lo and hi are used to create the color gradient.
midhi	Color for middle-high
hi	Color for high end
preview	Logical: Plot the colors horizontally
colorbar	Logical: Create a vertical colorbar
cb_n	Integer: How many steps you would like in the colorbar
cb_mar	Vector, length 4: Colorbar margins. Default: <code>c(1, 1, 1, 1)</code>
cb_add	Logical: If TRUE, colorbar will be added to existing plot
cb_add_mar	Vector: Margins for colorbar (See <code>par("mar")</code> )
cb_axis_pos	Float: Position of axis (See <code>axis("pos")</code> )

<code>cb_axis_las</code>	Integer {0,1,2,3}: Style of axis labels. 0: Always parallel to the axis, 1: Horizontal, 2: Perpendicular, 3: Vertical.
<code>cb_axis_hadj</code>	Float: Adjustment parallel to the reading direction (See <code>par("adj")</code> )
<code>cb_cex</code>	Float: Character expansion factor for colorbar (See <code>par("cex")</code> )
<code>bar_min</code>	Numeric: Lowest value in colorbar
<code>bar_mid</code>	Numeric: Middle value in colorbar
<code>bar_max</code>	Numeric: Max value in colorbar
<code>cex</code>	Float: Character expansion for axis
<code>filename</code>	String (Optional: Path to file to save colorbar
<code>pdf_width</code>	Float: Width for PDF output.
<code>pdf_height</code>	Float: Height for PDF output.
<code>theme</code>	Theme object.
<code>bg</code>	Color: Background color
<code>col_text</code>	Color: Colorbar text color
<code>plotlycb</code>	Logical: Create colorbar using plotly (instead of base R graphics)
<code>plotly_width</code>	Float: Width for plotly colorbar.
<code>plotly_height</code>	Float: Height for plotly colorbar.
<code>return_plotly</code>	Logical: If TRUE, return plotly object
<code>margins</code>	Vector: Plotly margins.
<code>pad</code>	Float: Padding for plotly.
<code>par_reset</code>	Logical: If TRUE (Default), reset <code>par</code> settings after running

## Details

It is best to provide an odd number, so that there is always an equal number of colors on either side of the midpoint. For example, if you want a gradient from -1 to 1 or equivalent, an  $n = 11$ , will give 5 colors on either side of 0, each representing a 20\

colors can be defined as a sequence of 3-letter color abbreviations of 2, 3, 4, or 5 colors which will correspond to values: {"lo","hi"}; {"lo", "mid", "hi"}; {"lo", "mid", "midhi", "hi"}, and {"lo", "lo-mid", "mid", "midhi", "hi"}, respectively. For example, try `colorgrad(21, "blugrnbkredyel", colorbar = TRUE)` 3-letter color abbreviations: wht: white; blk: black; red; grn: green; blu: blue; yel: yellow; rng: orange; prl: purple

## Value

Invisible vector of hexadecimal colors / plotly object if `return_plotly = TRUE`

## Author(s)

EDG

---

color_adjust	<i>Adjust HSV Color</i>
--------------	-------------------------

---

**Description**

Modify alpha, hue, saturation and value (HSV) of a color

**Usage**

```
color_adjust(color, alpha = NULL, hue = 0, sat = 0, val = 0)
```

**Arguments**

color	Input color. Any format that <code>grDevices::col2rgb()</code> recognizes
alpha	Numeric: Scale alpha by this amount. Future: replace with absolute setting
hue	Float: How much hue to add to color
sat	Float: How much saturation to add to color
val	Float: How much to increase value of color by

**Value**

Adjusted color

**Author(s)**

EDG

---

color_fade	<i>Fade color towards target</i>
------------	----------------------------------

---

**Description**

Fade color towards target

**Usage**

```
color_fade(x, to = "#000000", pct = 0.5)
```

**Arguments**

x	Color source
to	Target color
pct	Numeric (0, 1) fraction of the distance in RGBA space between x and to to move. e.g. .5 gets the mean RGBA value of the two

**Value**

Color in hex notation

**Author(s)**

EDG

---

color_invertRGB	<i>Invert Color in RGB space</i>
-----------------	----------------------------------

---

**Description**

Invert Color in RGB space

**Usage**

```
color_invertRGB(x)
```

**Arguments**

x	Color, vector
---	---------------

**Value**

Inverted colors using hexadecimal notation #RRGGBBAA

**Author(s)**

EDG

**Examples**

```
## Not run:
cols <- c("red", "green", "blue")
previewcolor(cols)
cols |>
  color_invertRGB() |>
  previewcolor()

## End(Not run)
```

---

color_mix	<i>Create an alternating sequence of graded colors</i>
-----------	--

---

**Description**

Create an alternating sequence of graded colors

**Usage**

```
color_mix(color, n = 4)
```

**Arguments**

- |       |   |
|-------|---|
| color | List: List of two or more elements, each containing two colors. A gradient will be created from the first to the second color of each element |
| n     | Integer: Number of steps in each gradient.  |

**Value**

Character vector of color hex codes.

**Author(s)**

EDG

**Examples**

```
## Not run:
color <- list(
  blue = c("#82afd3", "#000f3a"),
  gray = c("gray10", "gray85")
)
previewcolor(desaturate(color_mix(color, 6), .3))

color <- list(
  blue = c("#82afd3", "#57000a"),
  gray = c("gray10", "gray85")
)
previewcolor(desaturate(color_mix(color, 6), .3))

color <- list(
  blue = c("#82afd3", "#000f3a"),
  purple = c("#23001f", "#c480c1")
)
previewcolor(desaturate(color_mix(color, 5), .3))

## End(Not run)
```

---

color\_op

*Simple Color Operations*


---

**Description**

Invert a color or calculate the mean of two colors in HSV or RGB space. This may be useful in creating colors for plots

**Usage**

```
color_op(col, fn = c("invert", "mean"), space = c("HSV", "RGB"))
```

**Arguments**

- |       |  |
|-------|--|
| col   | Input color(s)   |
| fn    | Character: "invert", "mean": Function to perform                       |
| space | Character: "HSV", "RGB": Colorspace to operate in - for averaging only |

**Details**

The average of two colors in RGB space will often pass through gray, which is likely undesirable. Averaging in HSV space, better for most applications.

**Value**

Color

**Author(s)**

EDG

---

color_txt_columns	<i>Color columns of text art</i>
-------------------	----------------------------------

---

**Description**

This function accepts text input of 1 or more lines and two colors. It will: a) generate a color gradient between the two colors b) apply the gradient to each column of the text, creating a left to right color gradient.

**Usage**

```
color_txt_columns(
  x,
  color_left,
  color_right,
  output_type = c("ansi", "html", "plain")
)
```

**Arguments**

x	Character vector of text to colorize.
color_left	Color for the left side of the gradient.
color_right	Color for the right side of the gradient.
output_type	Character: Output type. One of "ansi", "html", "plain". Default = "ansi".

**Value**

Character vector with color formatting applied to each column.

**Author(s)**

EDG

---

`ddSci`*Format Numbers for Printing*

---

**Description**

2 Decimal places, otherwise scientific notation

**Usage**

```
ddSci(x, decimal_places = 2, hi = 1e+06, as_numeric = FALSE)
```

**Arguments**

<code>x</code>	Vector of numbers
<code>decimal_places</code>	Integer: Return this many decimal places.
<code>hi</code>	Float: Threshold at or above which scientific notation is used.
<code>as_numeric</code>	Logical: If TRUE, convert to numeric before returning. This will not force all numbers to print 2 decimal places. For example: 1.2035 becomes "1.20" if <code>as_numeric = FALSE</code> , but 1.2 otherwise. This can be helpful if you want to be able to use the output as numbers / not just for printing.

**Details**

Numbers will be formatted to 2 decimal places, unless this results in 0.00 (e.g. if input was .0032), in which case they will be converted to scientific notation with 2 significant figures. `ddSci` will return 0.00 if the input is exactly zero. This function can be used to format numbers in plots, on the console, in logs, etc.

**Value**

Formatted number

**Author(s)**

EDG

**Examples**

```
x <- .34876549
ddSci(x)
# "0.35"
x <- .00000000457823
ddSci(x)
# "4.6e-09"
```

---

desaturate	<i>Pastelify a color (make a color more pastel)</i>
------------	---

---

**Description**

Lower a color's saturation by a given percent in the HSV color system

**Usage**

```
desaturate(x, s = 0.3)
```

**Arguments**

x	Color, vector: Color(s) to operate on
s	Float: Decrease saturation by this fraction. Default = .3, which means if saturation of given color is 1, it will become .7

**Value**

List of adjusted colors

**Author(s)**

EDG

**Examples**

```
## Not run:
cols <- c("red", "green", "blue")
previewcolor(cols)
cols_d <- desaturate(cols)
previewcolor(cols_d)

## End(Not run)
```

---

df_movecolumn	<i>Move data frame column</i>
---------------	-------------------------------

---

**Description**

Move data frame column

**Usage**

```
df_movecolumn(x, colname, to = ncol(x))
```

**Arguments**

x	data.frame.
colname	Character: Name of column you want to move.
to	Integer: Which column position to move the vector to. Default = ncol(x) i.e. the last column.

**Value**

data.frame

**Author(s)**

EDG

**Examples**

```
## Not run:
ir <- df_movecolumn(iris, colname = "Species", to = 1L)

## End(Not run)
```

---

drange

*Set Dynamic Range*

---

**Description**

rtemis preproc: Adjusts the dynamic range of a vector or matrix input. By default normalizes to 0-1 range.

**Usage**

```
drange(x, lo = 0, hi = 1, byCol = TRUE)
```

**Arguments**

x	Numeric vector or matrix / data frame: Input
lo	Target range minimum. Defaults to 0
hi	Target range maximum. Defaults to 1
byCol	Logical: If TRUE: if x is matrix, drange each column separately

**Value**

Numeric vector.

**Author(s)**

EDG

**Examples**

```
## Not run:
x <- runif(20, -10, 10)
x <- drange(x)

## End(Not run)
```

---

dt_describe	<i>Describe data.table</i>
-------------	----------------------------

---

**Description**

Describe data.table

**Usage**

```
dt_describe(x, verbosity = 1L)
```

**Arguments**

x	data.table: Input data.table.
verbosity	Integer: If > 0, print output to console.

**Value**

List with three data.tables: Numeric, Categorical, and Date.

**Author(s)**

EDG

**Examples**

```
library(data.table)
origin <- as.POSIXct("2022-01-01 00:00:00", tz = "America/Los_Angeles")
x <- data.table(
  ID = paste0("ID", 1:10),
  V1 = rnorm(10),
  V2 = rnorm(10, 20, 3),
  V1_datetime = as.POSIXct(
    seq(
      1, 1e7,
      length.out = 10
    ),
    origin = origin
  ),
  V2_datetime = as.POSIXct(
    seq(
      1, 1e7,
      length.out = 10
    ),
    origin = origin
  ),
  C1 = sample(c("alpha", "beta", "gamma"), 10, TRUE),
  F1 = factor(sample(c("delta", "epsilon", "zeta"), 10, TRUE))
)
```

---

dt_inspect_types	<i>Inspect column types</i>
------------------	-----------------------------

---

**Description**

Will attempt to identify columns that should be numeric but are either character or factor by running [inspect\\_type](#) on each column.

**Usage**

```
dt_inspect_types(x, cols = NULL, verbosity = 1L)
```

**Arguments**

x	data.table: Input data.table.
cols	Character vector: columns to inspect.
verbosity	Integer: Verbosity level.

**Value**

Character vector.

**Author(s)**

EDG

**Examples**

```
library(data.table)
x <- data.table(
  id = 8001:8006,
  a = c("3", "5", "undefined", "21", "4", NA),
  b = c("mango", "banana", "tangerine", NA, "apple", "kiwi"),
  c = c(1, 2, 3, 4, 5, 6)
)
dt_inspect_types(x)
```

---

dt_keybin_reshape	<i>Long to wide key-value reshaping</i>
-------------------	---

---

**Description**

Reshape a long format data.table using key-value pairs with `data.table::dcast`

**Usage**

```
dt_keybin_reshape(
  x,
  id_name,
  key_name,
  positive = 1,
  negative = 0,
  xname = NULL,
  verbosity = 1L
)
```

**Arguments**

x	data.table object.
id_name	Character: Name of column in x that defines the IDs identifying individual rows.
key_name	Character: Name of column in x that holds the key.
positive	Numeric or Character: Used to fill id ~ key combination present in the long format input x.
negative	Numeric or Character: Used to fill id ~ key combination NOT present in the long format input x.
xname	Character: Name of x to be used in messages.
verbosity	Integer: Verbosity level.

**Value**

data.table in wide format.

**Author(s)**

EDG

**Examples**

```
library(data.table)
x <- data.table(
  ID = rep(1:3, each = 2),
  Dx = c("A", "C", "B", "C", "D", "A")
)
dt_keybin_reshape(x, id_name = "ID", key_name = "Dx")
```

---

dt\_merge

---

Merge data.tables

---

**Description**

Merge data.tables

**Usage**

```
dt_merge(
  left,
  right,
  on = NULL,
  left_on = NULL,
  right_on = NULL,
  how = "left",
  left_name = NULL,
  right_name = NULL,
  left_suffix = NULL,
  right_suffix = NULL,
  verbosity = 1L,
  ...
)
```

**Arguments**

left	data.table
right	data.table
on	Character: Name of column to join on.
left_on	Character: Name of column on left table.
right_on	Character: Name of column on right table.
how	Character: Type of join: "inner", "left", "right", "outer".
left_name	Character: Name of left table.
right_name	Character: Name of right table.
left_suffix	Character: If provided, add this suffix to all left column names, excluding on/left_on.
right_suffix	Character: If provided, add this suffix to all right column names, excluding on/right_on.
verbosity	Integer: Verbosity level.
...	Additional arguments to be passed to data.table::merge.

**Value**

Merged data.table.

**Author(s)**

EDG

**Examples**

```
library(data.table)
xleft <- data.table(ID = 1:5, Alpha = letters[1:5])
xright <- data.table(ID = c(3, 4, 5, 6), Beta = LETTERS[3:6])
xlr_inner <- dt_merge(xleft, xright, on = "ID", how = "inner")
```

---

dt_names_by_attr	<i>List column names by attribute</i>
------------------	---------------------------------------

---

**Description**

List column names by attribute

**Usage**

```
dt_names_by_attr(x, attribute, exact = TRUE, sorted = TRUE)
```

**Arguments**

x	data.table: Input data.table.
attribute	Character: name of attribute.
exact	Logical: If TRUE, use exact matching.
sorted	Logical: If TRUE, sort the output.

**Value**

Character vector.

**Author(s)**

EDG

---

dt_nunique_perfeat	<i>Number of unique values per feature</i>
--------------------	--

---

**Description**

Number of unique values per feature

**Usage**

```
dt_nunique_perfeat(x, excludeNA = FALSE, limit = 20L, verbosity = 1L)
```

**Arguments**

x	data.table: Input data.table.
excludeNA	Logical: If TRUE, exclude NA values.
limit	Integer: Print up to this many features. Set to -1L to print all.
verbosity	Integer: If > 0, print output to console.

**Value**

Named integer vector of length NCOL(x) with number of unique values per column/feature, invisibly.

**Author(s)**

EDG

**Examples**

```
library(data.table)
ir <- as.data.table(iris)
dt_nunique_perfeat(ir)
```

---

dt_pctmatch	<i>Get N and percent match of values between two columns of two data.tables</i>
-------------	---

---

**Description**

Get N and percent match of values between two columns of two data.tables

**Usage**

```
dt_pctmatch(x, y, on = NULL, left_on = NULL, right_on = NULL, verbosity = 1L)
```

**Arguments**

x	data.table: First input data.table.
y	data.table: Second input data.table.
on	Integer or character: column to read in x and y, if it is the same
left_on	Integer or character: column to read in x
right_on	Integer or character: column to read in y
verbosity	Integer: Verbosity level.

**Value**

list.

**Author(s)**

EDG

**Examples**

```
library(data.table)
x <- data.table(ID = 1:5, Alpha = letters[1:5])
y <- data.table(ID = c(3, 4, 5, 6), Beta = LETTERS[3:6])
dt_pctmatch(x, y, on = "ID")
```

---

dt_pctmissing	<i>Get percent of missing values from every column</i>
---------------	--

---

**Description**

Get percent of missing values from every column

**Usage**

```
dt_pctmissing(x, verbosity = 1L)
```

**Arguments**

x	data.frame or data.table
verbosity	Integer: Verbosity level.

**Value**

list

**Author(s)**

EDG

**Examples**

```
library(data.table)
x <- data.table(a = c(1, 2, NA, 4), b = c(NA, NA, 3, 4), c = c("A", "B", "C", NA))
dt_pctmissing(x)
```

---

dt_set_autotypes	<i>Set column types automatically</i>
------------------	---------------------------------------

---

**Description**

This function inspects a data.table and attempts to identify columns that should be numeric but have been read in as character, and fixes their type **in-place**. This can happen when one or more fields contain non-numeric characters, for example.

**Usage**

```
dt_set_autotypes(x, cols = NULL, verbosity = 1L)
```

**Arguments**

x	data.table: Input data.table. Will be modified <b>in-place</b> , if needed.
cols	Character vector: columns to work on. If not defined, will work on all columns
verbosity	Integer: Verbosity level.

**Value**

data.table, invisibly.

**Author(s)**

EDG

**Examples**

```
library(data.table)
x <- data.table(
  id = 8001:8006,
  a = c("3", "5", "undefined", "21", "4", NA),
  b = c("mango", "banana", "tangerine", NA, "apple", "kiwi"),
  c = c(1, 2, 3, 4, 5, 6)
)
str(x)
# ***in-place*** operation means no assignment is needed
dt_set_autotypes(x)
str(x)

# Try excluding column 'a' from autotyping
x <- data.table(
  id = 8001:8006,
  a = c("3", "5", "undefined", "21", "4", NA),
  b = c("mango", "banana", "tangerine", NA, "apple", "kiwi"),
  c = c(1, 2, 3, 4, 5, 6)
)
str(x)
# exclude column 'a' from autotyping
dt_set_autotypes(x, cols = setdiff(names(x), "a"))
str(x)
```

---

dt\_set\_cleanfactorlevels

*Clean factor levels of data.table* **in-place**

---

**Description**

Finds all factors in a data.table and cleans factor levels to include only underscore symbols

**Usage**

```
dt_set_cleanfactorlevels(x, prefix_digits = NA)
```

**Arguments**

**x** data.table: Input data.table. Will be modified **in-place**.  
**prefix\_digits** Character: If not NA, add this prefix to all factor levels that are numbers

**Value**

Nothing, modifies x **in-place**.

**Author(s)**

EDG

**Examples**

```
library(data.table)
x <- as.data.table(iris)
levels(x[["Species"]]) <- c("setosa:iris", "versicolor$iris", "virginica iris")
levels(x[["Species"]])
dt_set_cleanfactorlevels(x)
levels(x[["Species"]])
```

---

dt_set_clean_all	<i>Clean column names and factor levels <b>in-place</b></i>
------------------	---

---

**Description**Clean column names and factor levels **in-place****Usage**

```
dt_set_clean_all(x, prefix_digits = NA)
```

**Arguments**

`x` data.table: Input data.table. Will be modified **in-place**, if needed.

`prefix_digits` Character: prefix to add to names beginning with a digit. Set to NA to skip

**Value**Nothing, modifies `x` **in-place**.**Author(s)**

EDG

**Examples**

```
library(data.table)
x <- as.data.table(iris)
levels(x[["Species"]]) <- c("setosa:iris", "versicolor$iris", "virginica iris")
names(x)
levels(x[["Species"]])
# ***in-place*** operation means no assignment is needed
dt_set_clean_all(x)
names(x)
levels(x[["Species"]])
```

---

dt\_set\_logical2factor *Convert data.table logical columns to factors*

---

## Description

Convert data.table logical columns to factors with custom labels **in-place**

## Usage

```
dt_set_logical2factor(
  x,
  cols = NULL,
  labels = c("False", "True"),
  maintain_attributes = TRUE,
  fillNA = NULL
)
```

## Arguments

x	data.table: Input data.table. Will be modified <b>in-place</b> .
cols	Integer or character: columns to convert, if NULL, operates on all logical columns
labels	Character: labels for factor levels
maintain_attributes	Logical: If TRUE, maintain column attributes
fillNA	Character: If not NULL, fill NA values with this constant

## Value

data.table, invisibly.

## Author(s)

EDG

## Examples

```
library(data.table)
x <- data.table(a = 1:5, b = c(TRUE, FALSE, FALSE, FALSE, TRUE))
x
dt_set_logical2factor(x)
x
z <- data.table(
  alpha = 1:5,
  beta = c(TRUE, FALSE, TRUE, NA, TRUE),
  gamma = c(FALSE, FALSE, TRUE, FALSE, NA)
)
# You can use fillNA to fill NA values with a constant
dt_set_logical2factor(z, cols = "beta", labels = c("No", "Yes"), fillNA = "No")
z
w <- data.table(mango = 1:5, banana = c(FALSE, FALSE, TRUE, TRUE, FALSE))
w
dt_set_logical2factor(w, cols = 2, labels = c("Ugh", "Huh"))
```

```

w
# Column attributes are maintained by default:
z <- data.table(
  alpha = 1:5,
  beta = c(TRUE, FALSE, TRUE, NA, TRUE),
  gamma = c(FALSE, FALSE, TRUE, FALSE, NA)
)
for (i in seq_along(z)) setattr(z[[i]], "source", "Guava")
str(z)
dt_set_logical2factor(z, cols = "beta", labels = c("No", "Yes"))
str(z)

```

---

export\_plotly

*Export plotly plot to file*


---

## Description

Export plotly plot to file

## Usage

```

export_plotly(
  x,
  filename,
  width = 600,
  height = 600,
  scale = 1,
  import_kaleido = TRUE,
  verbosity = 1L
)

```

## Arguments

x	plotly object.
filename	Character: Filename to save the plot to.
width	Numeric: Width of the exported image in pixels.
height	Numeric: Height of the exported image in pixels.
scale	Numeric: Scale factor for the exported image.
import_kaleido	Logical: If TRUE, attempts to import kaleido for exporting plotly plots.
verbosity	Integer: Verbosity level.

## Author(s)

EDG

---

factor_NA2missing	<i>Factor NA to "missing" level</i>
-------------------	-------------------------------------

---

**Description**

Set NA values of a factor vector to a new level indicating missingness

**Usage**

```
factor_NA2missing(x, na_level_name = "missing")
```

**Arguments**

x	Factor.
na_level_name	Character: Name of new level to create that will be assigned to all current NA values in x.

**Value**

factor.

**Author(s)**

EDG

**Examples**

```
x <- factor(sample(letters[1:3], 100, TRUE))
x[sample(1:100, 10)] <- NA
xm <- factor_NA2missing(x)
```

---

fct_describe	<i>Describe factor</i>
--------------	------------------------

---

**Description**

Outputs a single character with names and counts of each level of the input factor.

**Usage**

```
fct_describe(x, max_n = 5, return_ordered = TRUE)
```

**Arguments**

x	factor.
max_n	Integer: Return counts for up to this many levels.
return_ordered	Logical: If TRUE, return levels ordered by count, otherwise return in level order.

**Value**

Character with level counts.

**Author(s)**

EDG

**Examples**

```
## Not run:
# Small number of levels
fct_describe(iris[["Species"]])

# Large number of levels: show top n by count
x <- factor(sample(letters, 1000, TRUE))
fct_describe(x)
fct_describe(x, 3)

## End(Not run)
```

---

filter\_order

*Filter order*


---

**Description**

Filter order

**Usage**

```
filter_order(x, id1, decreasing = FALSE)
```

**Arguments**

x	Input vector
id1	Logical vector: Index of elements to filter
decreasing	Logical: If TRUE, sort in descending order

**Author(s)**

EDG

**Examples**

```
## Not run:
x <- rnorm(10)
x
x[filter_order(x, x < 0)]

## End(Not run)
```

---

fmt	<i>Text formatting</i>
-----	------------------------

---

**Description**

Formats text with specified color, styles, and background using ANSI escape codes or HTML, with support for plain text output.

**Usage**

```
fmt(  
  x,  
  col = NULL,  
  bold = FALSE,  
  italic = FALSE,  
  underline = FALSE,  
  thin = FALSE,  
  muted = FALSE,  
  bg = NULL,  
  pad = 0L,  
  output_type = c("ansi", "html", "plain")  
)
```

**Arguments**

x	Character: Text to format.
col	Character: Color (hex code, named color, or NULL for no color).
bold	Logical: If TRUE, make text bold.
italic	Logical: If TRUE, make text italic.
underline	Logical: If TRUE, underline text.
thin	Logical: If TRUE, make text thin/light.
muted	Logical: If TRUE, make text muted/dimmed.
bg	Character: Background color (hex code, named color, or NULL).
pad	Integer: Number of spaces to pad before text.
output_type	Character: Output type ("ansi", "html", "plain").

**Details**

This function combines multiple formatting options into a single call, making it more efficient than nested function calls. It generates optimized ANSI escape sequences and clean HTML output.

**Value**

Character: Formatted text with specified styling.

**Author(s)**

EDG

## Examples

```
# Simple color
fmt("Hello", col = "red")

# Bold red text
fmt("Error", col = "red", bold = TRUE)

# Multiple styles
fmt("Warning", col = "yellow", bold = TRUE, italic = TRUE)

# With background
fmt("Highlight", col = "white", bg = "blue", bold = TRUE)
```

---

fmt_gradient	<i>Gradient text</i>
--------------	----------------------

---

## Description

Gradient text

## Usage

```
fmt_gradient(x, colors, bold = FALSE, output_type = c("ansi", "html", "plain"))
```

## Arguments

x	Character: Text to colorize.
colors	Character vector: Colors to use for the gradient.
bold	Logical: If TRUE, make text bold.
output_type	Character: Output type ("ansi", "html", "plain").

## Value

Character: Text with gradient color applied.

## Author(s)

EDG

---

getnames*Get names by string matching or class*

---

**Description**

Get names by string matching or class

**Usage**

```
getnames(  
  x,  
  pattern = NULL,  
  starts_with = NULL,  
  ends_with = NULL,  
  ignore_case = TRUE  
)
```

```
getfactornames(x)
```

```
getnumericnames(x)
```

```
getlogicalnames(x)
```

```
getcharacternames(x)
```

```
getdatenames(x)
```

**Arguments**

x	object with names() method.
pattern	Character: pattern to match anywhere in names of x.
starts_with	Character: pattern to match in the beginning of names of x.
ends_with	Character: pattern to match at the end of names of x.
ignore_case	Logical: If TRUE, well, ignore case.

**Details**

For getnames() only: pattern, starts\_with, and ends\_with are applied sequentially. If more than one is provided, the result will be the intersection of all matches.

**Value**

Character vector of matched names.

**Author(s)**

EDG

**Examples**

```
getnames(iris, starts_with = "Sepal")  
getnames(iris, ends_with = "Width")
```

---

getnamesandtypes	<i>Get data.frame names and types</i>
------------------	---------------------------------------

---

**Description**

Get data.frame names and types

**Usage**

```
getnamesandtypes(x)
```

**Arguments**

x	data.frame / data.table or similar
---	------------------------------------

**Value**

character vector of column names with attribute "type" holding the class of each column

---

get_loaded_pkg_version	<i>Get version of all loaded packages (namespaces)</i>
------------------------	--

---

**Description**

Get version of all loaded packages (namespaces)

**Usage**

```
get_loaded_pkg_version()
```

**Value**

Data frame with columns "Package\_Name" and "Version".

**Author(s)**

EDG

**Examples**

```
get_loaded_pkg_version()
```

---

get_mode	<i>Get the mode of a factor or integer</i>
----------	--

---

**Description**

Returns the mode of a factor or integer

**Usage**

```
get_mode(x, na.rm = TRUE, getlast = TRUE, retain_class = TRUE)
```

**Arguments**

x	Vector, factor or integer: Input data.
na.rm	Logical: If TRUE, exclude NAs (using <code>na.exclude(x)</code> ).
getlast	Logical: If TRUE, get the last value in case of ties.
retain_class	Logical: If TRUE, output is always same class as input.

**Value**

The mode of x

**Author(s)**

EDG

**Examples**

```
x <- c(9, 3, 4, 4, 0, 2, 2, NA)
get_mode(x)
x <- c(9, 3, 2, 2, 0, 4, 4, NA)
get_mode(x)
get_mode(x, getlast = FALSE)
```

---

get_output_type	<i>Get output type</i>
-----------------	------------------------

---

**Description**

Get output type for printing text.

**Usage**

```
get_output_type(output_type = c("ansi", "html", "plain"), filename = NULL)
```

**Arguments**

output_type	Character vector of output types.
filename	Optional Character: Filename for output.

**Details**

Exported as internal function for use by other rtemis packages.

**Value**

Character with selected output type.

**Author(s)**

EDG

**Examples**

```
get_output_type()
```

---

get_vars_from_rules	<i>Extract variable names from rules</i>
---------------------	--

---

**Description**

Extract variable names from rules

**Usage**

```
get_vars_from_rules(rules, unique = FALSE)
```

**Arguments**

rules	Character vector: Rules.
unique	Logical: If TRUE, return only unique variables.

**Value**

Character vector: Variable names.

**Author(s)**

EDG

---

graph_node_metrics	<i>Node-wise (i.e. vertex-wise) graph metrics</i>
--------------------	---

---

**Description**

Node-wise (i.e. vertex-wise) graph metrics

**Usage**

```
graph_node_metrics(x, verbosity = 1L)
```

**Arguments**

x	<b>igraph</b> network.
verbosity	Integer: Verbosity level.

**Value**

data.frame.

**Author(s)**

EDG

**Examples**

```
## Not run:
datcor <- cor(rnormmat(20, 20, seed = 2021))
datcor[sample(seq(datcor), 250)] <- 0
x <- igraph::graph_from_adjacency_matrix(
  adjmatrix = datcor,
  mode = "lower",
  weighted = TRUE,
  diag = FALSE
)

graph_node_metrics(x)

## End(Not run)
```

---

gray	<i>Gray text</i>
------	------------------

---

**Description**

A `fmt()` convenience wrapper for making text gray.

**Usage**

```
gray(x, output_type = c("ansi", "html", "plain"))
```

**Arguments**

`x` Character: Text to format  
`output_type` Character: Output type ("ansi", "html", "plain")

**Details**

Can be useful in contexts where muted is not supported.

**Value**

Character: Formatted text with gray styling

**Author(s)**

EDG

---

green	<i>Make text green</i>
-------	------------------------

---

**Description**

Make text green

**Usage**

```
green(..., bold = FALSE)
```

**Arguments**

`...` Character: Text to colorize.  
`bold` Logical: If TRUE, make text bold.

**Author(s)**

EDG

---

highlight	<i>Highlight text</i>
-----------	-----------------------

---

**Description**

A `fmt()` convenience wrapper for highlighting text.

**Usage**

```
highlight(x, pad = 0L, output_type = c("ansi", "html", "plain"))
```

**Arguments**

x	Character: Text to highlight.
pad	Integer: Number of spaces to pad before text.
output_type	Character: Output type ("ansi", "html", "plain").

**Value**

Character: Formatted text with highlight.

**Author(s)**

EDG

---

iflengthy	<i>Return object if it has length &gt; 0</i>
-----------	--

---

**Description**

Returns the input object if it has length > 0, else NULL

**Usage**

```
iflengthy(x)
```

**Arguments**

x	Object
---	--------

**Value**

x if length(x) > 0, else NULL

**Author(s)**

EDG

**Examples**

```
x <- 2:4
iflengthy(x)
y <- list()
iflengthy(y)
```

---

index_col_by_attr	<i>Index columns by attribute name &amp; value</i>
-------------------	--

---

**Description**

Index columns by attribute name & value

**Usage**

```
index_col_by_attr(x, name, value)
```

**Arguments**

x	data.frame or similar.
name	Character: Name of attribute.
value	Character: Value of attribute.

**Value**

Integer vector.

**Author(s)**

EDG

**Examples**

```
library(data.table)
x <- data.table(
  id = 1:5,
  sbp = rnorm(5, 120, 15),
  dbp = rnorm(5, 80, 10),
  paO2 = rnorm(5, 90, 10),
  paCO2 = rnorm(5, 40, 5)
)
setattr(x[["sbp"]], "source", "outpatient")
setattr(x[["dbp"]], "source", "outpatient")
setattr(x[["paO2"]], "source", "icu")
setattr(x[["paCO2"]], "source", "icu")
index_col_by_attr(x, "source", "icu")
```

---

init_project_dir	<i>Initialize Project Directory</i>
------------------	-------------------------------------

---

**Description**

Initializes Directory Structure: "R", "Data", "Results"

**Usage**

```
init_project_dir(verbosity = 1L)
```

**Arguments**

verbosity      Integer: Verbosity level.

**Value**

Character: the working directory path, invisibly.

**Author(s)**

EDG

---

inspect_type	<i>Inspect character and factor vector</i>
--------------	--

---

**Description**

Checks character or factor vector to determine whether it might be best to convert to numeric.

**Usage**

```
inspect_type(x, xname = NULL, verbosity = 1L, thresh = 0.5, na.omit = TRUE)
```

**Arguments**

x                      Character or factor vector.  
 xname                Character: Name of input vector x.  
 verbosity            Integer: Verbosity level.  
 thresh                Numeric: Threshold for determining whether to convert to numeric.  
 na.omit              Logical: If TRUE, remove NA values before checking.

**Details**

All data can be represented as a character string. A numeric variable may be read as a character variable if there are non-numeric characters in the data. It is important to be able to automatically detect such variables and convert them, which would mean introducing NA values.

**Value**

Character.

**Author(s)**

EDG

**Examples**

```
x <- c("3", "5", "undefined", "21", "4", NA)
inspect_type(x)
z <- c("mango", "banana", "tangerine", NA)
inspect_type(z)
```

---

is_constant	<i>Check if vector is constant</i>
-------------	------------------------------------

---

**Description**

Check if vector is constant

**Usage**

```
is_constant(x, skip_missing = FALSE)
```

**Arguments**

x	Vector: Input
skip_missing	Logical: If TRUE, skip NA values before test

**Value**

Logical.

**Author(s)**

EDG

**Examples**

```
## Not run:
x <- rep(9, 1000000)
is_constant(x)
x[10] <- NA
is_constant(x)
is_constant(x, skip_missing = TRUE)

## End(Not run)
```

---

is_discrete	<i>Check if variable is discrete (factor or integer)</i>
-------------	--

---

**Description**

Check if variable is discrete (factor or integer)

**Usage**

```
is_discrete(x)
```

**Arguments**

x	Input
---	-------

**Value**

Logical.

**Author(s)**

EDG

---

italic

*Make text italic*

---

**Description**

A `fmt()` convenience wrapper for making text italic.

**Usage**

```
italic(text, output_type = c("ansi", "html", "plain"))
```

**Arguments**

<code>text</code>	Character: Text to make italic
<code>output_type</code>	Character: Output type ("ansi", "html", "plain")

**Value**

Character: Formatted text with italic styling

**Author(s)**

EDG

---

labelify

*Format text for label printing*

---

**Description**

Format text for label printing

**Usage**

```
labelify(  
  x,  
  underscores_to_spaces = TRUE,  
  dotsToSpaces = TRUE,  
  toLower = FALSE,  
  toTitleCase = TRUE,  
  capitalize_strings = c("id"),  
  stringsToSpaces = c("\\$", "\\`")  
)
```

**Arguments**

<code>x</code>	Character: Input
<code>underscores_to_spaces</code>	Logical: If TRUE, convert underscores to spaces.
<code>dotsToSpaces</code>	Logical: If TRUE, convert dots to spaces.
<code>toLower</code>	Logical: If TRUE, convert to lowercase (precedes <code>toTitleCase</code> ). Default = FALSE (Good for getting all-caps words converted to title case, bad for abbreviations you want to keep all-caps)
<code>toTitleCase</code>	Logical: If TRUE, convert to Title Case. Default = TRUE (This does not change all-caps words, set <code>toLower</code> to TRUE if desired)
<code>capitalize_strings</code>	Character, vector: Always capitalize these strings, if present. Default = "id"
<code>stringsToSpaces</code>	Character, vector: Replace these strings with spaces. Escape as needed for <code>gsub</code> . Default = "\\\$", which formats common input of the type <code>data.frame\$variable</code>

**Value**

Character vector.

**Author(s)**

EDG

**Examples**

```
x <- c("county_name", "total.cost$", "age", "weight.kg")
labelify(x)
```

---

list2csv

---

*Write list elements to CSV files*


---

**Description**

Write list elements to CSV files

**Usage**

```
list2csv(x, outdir)
```

**Arguments**

<code>x</code>	List containing R objects to be written to CSV (e.g. <code>data.frames</code> , <code>matrices</code> , etc.)
<code>outdir</code>	Character: Path to output directory

**Value**

Nothing, writes CSV files to `outdir`.

**Author(s)**

EDG

**Examples**

```
## Not run:
x <- list(
  iris = iris,
  iris_normalized = as.data.frame(scale(iris[, -5]))
)
outdir <- "../exports"
list2csv(x, outdir)

## End(Not run)
```

lotri2edgeList

*Connectivity Matrix to Edge List***Description**

Turn the lower triangle of a connectivity matrix (e.g. correlation matrix or similar) to an edge list of the form: Source, Target, Weight

**Usage**

```
lotri2edgeList(A, filename = NULL, verbosity = 1L)
```

**Arguments**

A	Square matrix
filename	Character: Path for csv file. Defaults to "conmat2edgelist.csv"
verbosity	Integer: Verbosity level.

**Details**

The output can be read, for example, into gephi

**Value**

Data frame with columns: NodeA, NodeB, Weight

**Author(s)**

EDG

**Examples**

```
A <- matrix(rnorm(100), nrow = 10)
A[lower.tri(A)] <- t(A)[lower.tri(A)]
diag(A) <- 1
edgelist <- lotri2edgeList(A, filename = NULL)
head(edgelist)
```

---

make_path	<i>Expand, normalize, concatenate, clean path</i>
-----------	---

---

**Description**

Expand, normalize, concatenate, clean path

**Usage**

```
make_path(..., expand_path = TRUE)
```

**Arguments**

...	Character: Parts of path to concatenate.
expand_path	Logical: If TRUE, expand concatenated path using <a href="#">path.expand</a> .

**Value**

Character: Path.

**Author(s)**

EDG

---

matchcases	<i>Match cases by covariates</i>
------------	----------------------------------

---

**Description**

Find one or more cases from a pool data.frame that match cases in a target data.frame. Match exactly and/or by distance (sum of squared distances).

**Usage**

```
matchcases(
  target,
  pool,
  n_matches = 1,
  target_id = NULL,
  pool_id = NULL,
  exactmatch_factors = TRUE,
  exactmatch_cols = NULL,
  distmatch_cols = NULL,
  norepeats = TRUE,
  ignore_na = FALSE,
  verbosity = 1L
)
```

**Arguments**

<code>target</code>	data.frame you are matching against.
<code>pool</code>	data.frame you are looking for matches from.
<code>n_matches</code>	Integer: Number of matches to return.
<code>target_id</code>	Character: Column name in <code>target</code> that holds unique cases IDs. Default = NULL, in which case integer case numbers will be used.
<code>pool_id</code>	Character: Same as <code>target_id</code> for <code>pool</code> .
<code>exactmatch_factors</code>	Logical: If TRUE, selected cases will have to exactly match factors available in <code>target</code> .
<code>exactmatch_cols</code>	Character: Names of columns that should be matched exactly.
<code>distmatch_cols</code>	Character: Names of columns that should be distance-matched.
<code>norepeats</code>	Logical: If TRUE, cases in <code>pool</code> can only be chosen once.
<code>ignore_na</code>	Logical: If TRUE, ignore NA values during exact matching.
<code>verbosity</code>	Integer: Verbosity level.

**Value**

data.frame

**Author(s)**

EDG

**Examples**

```
## Not run:
set.seed(2021)
cases <- data.frame(
  PID = paste0("PID", seq(4)),
  Sex = factor(c(1, 1, 0, 0)),
  Handedness = factor(c(1, 1, 0, 1)),
  Age = c(21, 27, 39, 24),
  Var = c(.7, .8, .9, .6),
  Varx = rnorm(4)
)
controls <- data.frame(
  CID = paste0("CID", seq(50)),
  Sex = factor(sample(c(0, 1), 50, TRUE)),
  Handedness = factor(sample(c(0, 1), 50, TRUE, c(.1, .9))),
  Age = sample(16:42, 50, TRUE),
  Var = rnorm(50),
  Vary = rnorm(50)
)

mc <- matchcases(cases, controls, 2, "PID", "CID")

## End(Not run)
```

mgetnames

*Get names by string matching multiple patterns*

---

**Description**

Get names by string matching multiple patterns

**Usage**

```
mgetnames(  
  x,  
  pattern = NULL,  
  starts_with = NULL,  
  ends_with = NULL,  
  ignore_case = TRUE,  
  return_index = FALSE  
)
```

**Arguments**

x	Character vector or object with names() method.
pattern	Character vector: pattern(s) to match anywhere in names of x.
starts_with	Character: pattern to match in the beginning of names of x.
ends_with	Character: pattern to match at the end of names of x.
ignore_case	Logical: If TRUE, well, ignore case.
return_index	Logical: If TRUE, return integer index of matches instead of names.

**Details**

pattern, starts\_with, and ends\_with are applied and the union of all matches is returned. pattern can be a character vector of multiple patterns to match.

**Value**

Character vector of matched names or integer index.

**Author(s)**

EDG

**Examples**

```
mgetnames(iris, pattern = c("Sepal", "Petal"))  
mgetnames(iris, starts_with = "Sepal")  
mgetnames(iris, ends_with = "Width")
```

---

msg

---

*Message with provenance*


---

## Description

Print message to output with a prefix including data and time, and calling function or full call stack

## Usage

```
msg(
    ...,
    date = TRUE,
    caller = NULL,
    call_depth = 1L,
    caller_id = 1L,
    newline_pre = FALSE,
    newline = TRUE,
    format_fn = plain,
    sep = " "
)
```

```
msg0(
    ...,
    caller = NULL,
    call_depth = 1,
    caller_id = 1,
    newline_pre = FALSE,
    newline = TRUE,
    format_fn = plain,
    sep = ""
)
```

## Arguments

...	Message to print
date	Logical: if TRUE, include date and time in the prefix
caller	Character: Name of calling function
call_depth	Integer: Print the system call path of this depth.
caller_id	Integer: Which function in the call stack to print
newline_pre	Logical: If TRUE begin with a new line.
newline	Logical: If TRUE end with a new line.
format_fn	Function: Formatting function to use on the message text.
sep	Character: Use to separate objects in ...

## Details

If msg is called directly from the console, it will print [interactive>] in place of the call stack. msg0, similar to paste0, is msg(..., sep = "")

**Value**

Invisibly: List with call, message, and date

**Author(s)**

EDG

**Examples**

```
msg("Hello, world!")
x <- 42L
msg0("The answer is what you think it is (", x, ").")
```

---

msgdatetime	<i>Message datetime()</i>
-------------	---------------------------

---

**Description**

Message datetime()

**Usage**

```
msgdatetime(datetime_format = "%Y-%m-%d %H:%M:%S")
```

**Arguments**

datetime\_format

Character: Format for the date and time.

**Value**

Character: Formatted date and time.

**Author(s)**

EDG

---

msgdone	<i>msgdone</i>
---------	----------------

---

**Description**

msgdone

**Usage**

```
msgdone(caller = NULL, call_depth = 1, caller_id = 1, sep = " ")
```

**Arguments**

caller	Character: Name of calling function
call_depth	Integer: Print the system call path of this depth.
caller_id	Integer: Which function in the call stack to print
sep	Character: Use to separate objects in . . .

**Author(s)**

EDG

---

msgstart	<i>msgstart</i>
----------	-----------------

---

**Description**

msgstart

**Usage**

```
msgstart(..., newline_pre = FALSE, sep = "")
```

**Arguments**

...	Message to print
newline_pre	Logical: If TRUE begin with a new line.
sep	Character: Use to separate objects in . . .

**Author(s)**

EDG

---

muted	<i>Muted text</i>
-------	-------------------

---

**Description**

A `fmt()` convenience wrapper for making text muted.

**Usage**

```
muted(x, output_type = c("ansi", "html", "plain"))
```

**Arguments**

x	Character: Text to format
output_type	Character: Output type ("ansi", "html", "plain")

**Value**

Character: Formatted text with muted styling

**Author(s)**

EDG

---

names_by_class	<i>List column names by class</i>
----------------	-----------------------------------

---

**Description**

List column names by class

**Usage**

```
names_by_class(x, sorted = TRUE, item_format = highlight, maxlength = 24)
```

**Arguments**

x	data.frame or similar.
sorted	Logical: If TRUE, sort the output
item_format	Function: Function to format each item
maxlength	Integer: Maximum number of items to print

**Value**

NULL, invisibly.

**Author(s)**

EDG

**Examples**

```
names_by_class(iris)
```

---

orange	<i>Make text orange</i>
--------	-------------------------

---

**Description**

Make text orange

**Usage**

```
orange(..., bold = FALSE)
```

**Arguments**

...	Character: Text to colorize.
bold	Logical: If TRUE, make text bold.

**Author(s)**

EDG

---

pcat	<i>Pad-cat</i>
------	----------------

---

**Description**

Pad and concatenate two strings, with optional newline.

**Usage**

```
pcat(left, right, pad = 17, newline = TRUE)
```

**Arguments**

left	Character: Left string to pad and print.
right	Character: Right string to print after left.
pad	Integer: Total width to pad the left string to.
newline	Logical: If TRUE, print a newline after the right string.

**Author(s)**

EDG

**Examples**

```
## Not run:
{
  msg("Hello")
  pcat("super", "wow")
  pcat(NULL, "oooo")
}

## End(Not run)
```

---

previewcolor	<i>Preview color</i>
--------------	----------------------

---

## Description

Preview one or multiple colors using little rhombi with their little labels up top

## Usage

```
previewcolor(
  x,
  main = NULL,
  bg = "#333333",
  main_col = "#b3b3b3",
  main_x = 0.7,
  main_y = 0.2,
  main_adj = 0,
  main_cex = 0.9,
  main_font = 2,
  width = NULL,
  xlim = NULL,
  ylim = c(0, 2.2),
  asp = 1,
  labels_y = 1.55,
  label_cex = NULL,
  mar = c(0, 0, 0, 1),
  par_reset = TRUE,
  filename = NULL,
  pdf_width = 8,
  pdf_height = 2.5
)
```

## Arguments

x	Color, vector: One or more colors that R understands
main	Character: Title. Default = NULL, which results in <code>deparse(substitute(x))</code>
bg	Background color.
main_col	Color: Title color
main_x	Float: x coordinate for main.
main_y	Float: y coordinate for main.
main_adj	Float: adj argument to <code>mtext</code> for main.
main_cex	Float: character expansion factor for main.
main_font	Integer, 1 or 2: Weight of main 1: regular, 2: bold.
width	Float: Plot width. Default = NULL, i.e. set automatically
xlim	Vector, length 2: x-axis limits. Default = NULL, i.e. set automatically
ylim	Vector, length 2: y-axis limits.
asp	Float: Plot aspect ratio.

labels_y	Float: y coord for labels. Default = 1.55 (rhombi are fixed and range y .5 - 1.5)
label_cex	Float: Character expansion for labels. Default = NULL, and is calculated automatically based on length of x
mar	Numeric vector, length 4: margin size.
par_reset	Logical: If TRUE, reset par settings on exit.
filename	Character: Path to save plot as PDF.
pdf_width	Numeric: Width of PDF in inches.
pdf_height	Numeric: Height of PDF in inches.

**Value**

Nothing, prints plot.

**Author(s)**

EDG

**Examples**

```
## Not run:
colors <- colorgradient_x(seq(-5, 5))
previewcolor(colors)

## End(Not run)
```

---

printf

---

*Print data frame*


---

**Description**

Pretty print a data frame

**Usage**

```
printf(
  x,
  pad = 0,
  spacing = 1,
  ddSci_dp = NULL,
  transpose = FALSE,
  justify = "right",
  colnames = TRUE,
  rownames = TRUE,
  column_fmt = highlight,
  row_fmt = gray,
  newline_pre = FALSE,
  newline = FALSE
)
```

**Arguments**

x	data frame
pad	Integer: Pad output with this many spaces.
spacing	Integer: Number of spaces between columns.
ddSci_dp	Integer: Number of decimal places to print using <a href="#">ddSci</a> . Default = NULL for no formatting.
transpose	Logical: If TRUE, transpose x before printing.
justify	Character: "right", "left".
colnames	Logical: If TRUE, print column names.
rownames	Logical: If TRUE, print row names.
column_fmt	Formatting fn for printing column names.
row_fmt	Formatting fn for printing row names.
newline_pre	Logical: If TRUE, print a new line before printing data frame.
newline	Logical: If TRUE, print a new line after printing data frame.

**Details**

By design, numbers will not be justified, but using `ddSci_dp` will convert to characters, which will be justified. This is intentional for internal use.

**Author(s)**

EDG

---

printls

*Pretty print list*

---

**Description**

Pretty print a list (or data frame) recursively

**Usage**

```
printls(
  x,
  prefix = "",
  pad = 2L,
  item_format = bold,
  maxlength = 4L,
  center_title = TRUE,
  title = NULL,
  title_newline = TRUE,
  newline_pre = FALSE,
  format_fn_rhs = ddSci,
  print_class = TRUE,
  abbrev_class_n = 3L,
  print_df = FALSE,
  print_S4 = FALSE,
  limit = 12L
)
```

Arguments

x	list or object that will be converted to a list.
prefix	Character: Optional prefix for names.
pad	Integer: Pad output with this many spaces.
item_format	Formatting function for list item names.
maxlength	Integer: Maximum length of items to show using headdot() before truncating with ellipsis.
center_title	Logical: If TRUE, autopad title for centering, if present.
title	Character: Optional title to print before list.
title_newline	Logical: If TRUE, print title on new line.
newline_pre	Logical: If TRUE, print newline before list.
format_fn_rhs	Formatting function for right-hand side values.
print_class	Logical: If TRUE, print abbreviated class of object.
abbrev_class_n	Integer: Number of characters to abbreviate class names to.
print_df	Logical: If TRUE, print data frame contents, otherwise print n rows and columns.
print_S4	Logical: If TRUE, print S4 object contents, otherwise print class name.
limit	Integer: Maximum number of items to show. Use -1 for unlimited.

Details

Data frames in R began life as lists

Author(s)

EDG

---

qstat	SGE qstat
-------	-----------

---

Description

Run SGE qstat

Usage

qstat()

Details

alias for system("qstat")

Value

Called for its side effect of printing the SGE queue status.

Examples

```
## Not run:
qstat()

## End(Not run)
```

---

recycle	<i>Recycle values of vector to match length of target</i>
---------	---

---

**Description**

Recycle values of vector to match length of target

**Usage**

```
recycle(x, target)
```

**Arguments**

x	Vector to be recycled
target	Object whose length defines target length

**Details**

Used internally by many functions.

**Value**

Vector.

**Author(s)**

EDG

---

repr	<i>String representation</i>
------	------------------------------

---

**Description**

String representation

**Usage**

```
repr(x, ...)
```

**Details**

Exported as internal function for use by other rtemis packages.

**Value**

Character string representation of the object.

**Author(s)**

EDG

---

repr_ls	<i>Show list as formatted string</i>
---------	--------------------------------------

---

## Description

Works exactly like printls, but instead of printing to console with cat, it outputs a single string, formatted using mformat, so that cat(repr\_ls(x)) looks identical to printls(x) for any list x

## Usage

```
repr_ls(
  x,
  prefix = "",
  pad = 2L,
  item_format = bold,
  maxlength = 4L,
  center_title = TRUE,
  title = NULL,
  title_newline = TRUE,
  newline_pre = FALSE,
  format_fn_rhs = ddSci,
  print_class = TRUE,
  abbrev_class_n = 3L,
  print_df = FALSE,
  print_S4 = FALSE,
  limit = 12L,
  output_type = c("ansi", "html", "plain")
)
```

## Arguments

x	list or object that will be converted to a list.
prefix	Character: Optional prefix for names.
pad	Integer: Pad output with this many spaces.
item_format	Formatting function for items.
maxlength	Integer: Maximum length of items to show using headdot() before truncating with ellipsis.
center_title	Logical: If TRUE, autopad title for centering, if present.
title	Character: Title to print before list.
title_newline	Logical: If TRUE, print title on new line.
newline_pre	Logical: If TRUE, print newline before list.
format_fn_rhs	Formatting function for right-hand side of items.
print_class	Logical: If TRUE, print abbreviated class of object.
abbrev_class_n	Integer: Number of characters to abbreviate class names to.
print_df	Logical: If TRUE, print data frame contents, otherwise print n rows and columns.
print_S4	Logical: If TRUE, print S4 object contents, otherwise print class name.
limit	Integer: Maximum number of items to show.
output_type	Character: Output type for mformat ("ansi", "html", "plain").

**Details**

Exported as internal function for use by other rtemis packages.

**Value**

Character: Formatted string that can be printed with cat()

**Author(s)**

EDG

---

repr_S7name	<i>Show S7 class name</i>
-------------	---------------------------

---

**Description**

Show S7 class name

**Usage**

```
repr_S7name(x, col = col_object, pad = 0L, prefix = NULL, output_type = NULL)
```

**Arguments**

- x                   Character: S7 class name.
- col                 Color: Color code for the object name.
- pad                 Integer: Number of spaces to pad the message with.
- prefix             Character: Prefix to add to the object name.
- output\_type        Character: Output type ("ansi", "html", "plain").

**Value**

Character: Formatted string that can be printed with cat().

**Author(s)**

EDG

**Examples**

```
repr_S7name("Supervised") |> cat()
```

---

rnormmat	<i>Random Normal Matrix</i>
----------	-----------------------------

---

**Description**

Create a matrix or data frame of defined dimensions, whose columns are random normal vectors

**Usage**

```
rnormmat(  
  nrow = 10,  
  ncol = 10,  
  mean = 0,  
  sd = 1,  
  return_df = FALSE,  
  seed = NULL  
)
```

**Arguments**

nrow	Integer: Number of rows.
ncol	Integer: Number of columns.
mean	Float: Mean.
sd	Float: Standard deviation.
return_df	Logical: If TRUE, return data.frame, otherwise matrix.
seed	Integer: Set seed for rnorm.

**Value**

matrix or data.frame.

**Author(s)**

EDG

**Examples**

```
x <- rnormmat(20, 5, mean = 12, sd = 6, return_df = TRUE, seed = 2026)  
x
```

---

`rtemis_colors`*rtemis Color System*

---

## Description

A named list of colors used consistently across all packages in the rtemis ecosystem.

## Usage

```
rtemis_colors
```

## Format

A named list with the following elements:

```
rt_red "kaimana red"
```

```
rt_blue "kaimana light blue"
```

```
rt_green "kaimana medium green"
```

```
rt_orange "genlib orange"
```

```
rt_teal "rtemis teal"
```

```
rt_purple "rtemis purple"
```

```
rt_magenta "rtemis magenta"
```

```
highlight_col "rtemis teal"
```

```
col_object "rtemis teal"
```

```
col_info "lmd burgundy"
```

```
col_outer "kaimana red"
```

```
col_tuner "genlib orange"
```

## Details

Colors are provided as hex strings.

## Author(s)

EDG

## Examples

```
rtemis_colors[["rt_teal"]]
```

---

**rtpalette***Color Palettes*

---

**Description**

`rtpalette()` prints names of available color palettes. Each palette is a named list of hexadecimal color definitions which can be used with any graphics function. `rtpalette(palette_name)` returns a list of colors for a given palette.

**Usage**

```
rtpalette(palette = NULL, verbosity = 1L)
```

**Arguments**

<code>palette</code>	Character: Name of palette to return. Default = <code>NULL</code> : available palette names are printed and no palette is returned.
<code>verbosity</code>	Integer: Verbosity level.

**Value**

A list of available palettes, invisibly.

**Author(s)**

EDG

**Examples**

```
# Print available palettes
rtpalette()
# Get the Imperial palette
rtpalette("imperial")
```

---

**rtversion***Get rtemis version and system info*

---

**Description**

Get rtemis version and system info

**Usage**

```
rtversion()
```

**Value**

List: rtemis version and system info, invisibly.

**Author(s)**

EDG

**Examples**

```
rtversion()
```

---

rt_letters	<i>Construct an n-length vector of letters</i>
------------	--

---

**Description**

Returns an n-length vector of the latin alphabet, replicating for every 26 characters

**Usage**

```
rt_letters(n = 100, caps = FALSE)
```

**Arguments**

- |      |                                   |
|------|-----------------------------------|
| n    | Length of vector to return        |
| caps | Logical: If TRUE, return all caps |

---

rt_reactable	<i>View table using reactable</i>
--------------	-----------------------------------

---

**Description**

View table using reactable

**Usage**

```
rt_reactable(  
  x,  
  datatypes = NULL,  
  lightsout = TRUE,  
  bg = "#121212",  
  pagination = TRUE,  
  searchable = TRUE,  
  bordered = TRUE,  
  ...  
)
```

**Arguments**

x	data.frame, data.table or similar
datatypes	Character vector: Data types of columns in x, e.g. c("numeric", "factor", "character")
lightsout	Logical: If TRUE, use dark theme.
bg	Background color.
pagination	Logical: If TRUE, paginate table.
searchable	Logical: If TRUE, add search box.
bordered	Logical: If TRUE, add border.
...	Additional arguments passed to <code>reactable::reactable</code>

**Value**

reactable object.

**Author(s)**

E D Gennatas

**Examples**

```
## Not run:
# needs html viewer
rt_reactable(iris, datatypes = sapply(iris, class))

## End(Not run)
```

---

rules2medmod

---

*Convert rules from cutoffs to median/mode and range*


---

**Description**

Convert rules from cutoffs to median (range) and mode (range) format

**Usage**

```
rules2medmod(rules, x, .ddSci = TRUE, verbosity = 1L)
```

**Arguments**

rules	Character, vector: Input rules
x	Data frame: Data to evaluate rules
.ddSci	Logical: If TRUE, format all continuous variables using <a href="#">ddSci</a> , which will give either 2 decimal places, or scientific notation if two decimal places result in 0.00
verbosity	Integer: Verbosity level.

**Value**

Character vector.

**Author(s)**

EDG

---

runifmat

---

*Random Uniform Matrix*

---

**Description**

Create a matrix or data frame of defined dimensions, whose columns are random uniform vectors

**Usage**

```
runifmat(  
  nrow = 10,  
  ncol = 10,  
  min = 0,  
  max = 1,  
  return_df = FALSE,  
  seed = NULL  
)
```

**Arguments**

nrow	Integer: Number of rows.
ncol	Integer: Number of columns.
min	Float: Min.
max	Float: Max.
return_df	Logical: If TRUE, return data.frame, otherwise matrix.
seed	Integer: Set seed for rnorm.

**Value**

matrix or data.frame.

**Author(s)**

EDG

**Examples**

```
x <- runifmat(20, 5, min = 12, max = 18, return_df = TRUE, seed = 2026)  
x
```

---

setdiffsym	<i>Symmetric Set Difference</i>
------------	---------------------------------

---

**Description**

Symmetric Set Difference

**Usage**

```
setdiffsym(x, y)
```

**Arguments**

x	vector
y	vector of same type as x

**Value**

Vector.

**Author(s)**

EDG

**Examples**

```
setdiff(1:10, 1:5)
setdiff(1:5, 1:10)
setdiffsym(1:10, 1:5)
setdiffsym(1:5, 1:10)
```

---

sge_submit	<i>Submit expression to SGE grid</i>
------------	--------------------------------------

---

**Description**

Submit expression to SGE grid

**Usage**

```
sge_submit(
  expr,
  obj_names = NULL,
  packages = NULL,
  queue = NULL,
  n_workers = 4,
  sge_out = file.path(getwd(), "./sge_out"),
  sge_error = sge_out,
  sge_env = "#! /usr/bin/env bash",
  sge_opts = "#$ -cwd",
```

```

    R_command = NULL,
    system_command = NULL,
    h_rt = "00:25:00",
    mem_free = NULL,
    temp_dir = file.path(getwd(), ".sge_tempdir"),
    verbosity = 1L
)

```

## Arguments

expr	R expression
obj_names	Character vector: Names of objects to copy to cluster R session
packages	Character vector: Names of packages to load in cluster R session
queue	Character: Name of SGE queue to submit to
n_workers	Integer: Number of threads to request from scheduler
sge_out	Character: Path to directory to write standard out message files
sge_error	Character: Path to directory to write error message files
sge_env	Character: Shell environment for script to be submitted to SGE
sge_opts	Character: SGE options that will be written in shell script. Default = "#\$ -cwd"
R_command	Character: Optional R command(s) to run at the beginning of the R script
system_command	Character: system command to be run by shell script before executing R code. For example a command that export the R executable to use
h_rt	Character: Max time to request. Default = "00:25:00", i.e. 25 minutes
mem_free	Character: Amount of memory to request from the scheduler
temp_dir	Character: Temporary directory that is accessible to all execution nodes. Default = file.path(getwd(), ".sge_tempdir") You can use tempdir() if all execution nodes have access to the same filesystem as the submit node.
verbosity	Integer: Verbosity level.

## Value

Character, invisibly: The command that was submitted to SGE.

## Author(s)

EDG

## Examples

```

## Not run:
sge_submit({
  # Your code here
}, obj_names = c("df1", "model1"), packages = c("rtemis", "data.table"),
queue = "all.q", n_workers = 4, h_rt = "01:00:00", mem_free = "4G")

## End(Not run)

```

---

show_pad	<i>Add padding</i>
----------	--------------------

---

**Description**

Convenience function to add padding.

**Usage**

```
show_pad(pad = 2L, output_type = NULL)
```

**Arguments**

pad	Integer: Number of spaces to output - that's all.
output_type	Character: Output type ("ansi", "html", "plain").

**Author(s)**

EDG

---

show_range	<i>Print range of continuous variable</i>
------------	---

---

**Description**

Print range of continuous variable

**Usage**

```
show_range(x, ddSci = TRUE, decimal_places = 1, na.rm = TRUE)
```

**Arguments**

x	Numeric vector
ddSci	Logical: If TRUE, use <a href="#">ddSci</a> or range.
decimal_places	Integer: Number of decimal place to use if ddSci = TRUE.
na.rm	Logical: passed to <code>base::range</code>

**Value**

Called for its side effect of printing the range of x.

**Author(s)**

EDG

---

size	<i>Size of object</i>
------	-----------------------

---

### Description

Returns the size of an object

### Usage

```
size(x, verbosity = 1L)
```

### Arguments

**x** any object with `length()` or `dim()`.  
**verbosity** Integer: Verbosity level. If > 0, print size to console

### Details

If `dim(x)` is `NULL`, returns `length(x)`.

### Value

Integer vector with length equal to the number of dimensions of `x`, invisibly.

### Author(s)

EDG

### Examples

```
x <- rnorm(20)
size(x)
# 20
x <- matrix(rnorm(100), 20, 5)
size(x)
# 20 5
```

---

table_column_attr	<i>Tabulate column attributes</i>
-------------------	-----------------------------------

---

### Description

Tabulate column attributes

### Usage

```
table_column_attr(x, attr = "source", useNA = "always")
```

**Arguments**

x	data.frame or similar: Input data set.
attr	Character: Attribute to get
useNA	Character: Passed to table

**Value**

table.

**Author(s)**

EDG

**Examples**

```
library(data.table)
x <- data.table(
  id = 1:5,
  sbp = rnorm(5, 120, 15),
  dbp = rnorm(5, 80, 10),
  paO2 = rnorm(5, 90, 10),
  paCO2 = rnorm(5, 40, 5)
)
setattr(x[["sbp"]], "source", "outpatient")
setattr(x[["dbp"]], "source", "outpatient")
setattr(x[["paO2"]], "source", "icu")
setattr(x[["paCO2"]], "source", "icu")
table_column_attr(x, "source")
```

---

thin

*Make text thin/light*

---

**Description**

A `fmt()` convenience wrapper for making text thin/light.

**Usage**

```
thin(text, output_type = c("ansi", "html", "plain"))
```

**Arguments**

text	Character: Text to make thin
output_type	Character: Output type ("ansi", "html", "plain")

**Value**

Character: Formatted text with thin/light styling

**Author(s)**

EDG

---

underline	<i>Make text underlined</i>
-----------	-----------------------------

---

**Description**

A `fmt()` convenience wrapper for making text underlined.

**Usage**

```
underline(text, output_type = c("ansi", "html", "plain"))
```

**Arguments**

text	Character: Text to underline
output_type	Character: Output type ("ansi", "html", "plain")

**Value**

Character: Formatted text with underline styling

**Author(s)**

EDG

---

uniprot_get	<i>Get protein sequence from UniProt</i>
-------------	--

---

**Description**

Get protein sequence from UniProt

**Usage**

```
uniprot_get(  
  accession,  
  baseURL = "https://rest.uniprot.org/uniprotkb",  
  verbosity = 1  
)
```

**Arguments**

accession	Character: UniProt Accession number - e.g. "Q9UMX9"
baseURL	Character: UniProt rest API base URL. Default = "https://rest.uniprot.org/uniprotkb"
verbosity	Integer: Verbosity level.

**Value**

List with three elements: Identifier, Annotation, and Sequence.

**Author(s)**

E.D. Gennatas

**Examples**

```
## Not run:  
mapt <- uniprot_get("Q9UMX9")  
  
## End(Not run)
```

---

uniquevalsperfeat	<i>Unique values per feature</i>
-------------------	----------------------------------

---

**Description**

Get number of unique values per features

**Usage**

```
uniquevalsperfeat(x, excludeNA = FALSE)
```

**Arguments**

x	matrix or data frame input
excludeNA	Logical: If TRUE, exclude NA values from unique count.

**Value**

Vector, integer of length NCOL(x) with number of unique values per column/feature

**Author(s)**

EDG

**Examples**

```
## Not run:  
uniquevalsperfeat(iris)  
  
## End(Not run)
```

---

vec2df	<i>Vector to data.frame</i>
--------	-----------------------------

---

**Description**

Convert vector to 1-row data.frame, maintaining names if present

**Usage**

```
vec2df(x, col_names = NULL)
```

**Arguments**

x	Vector.
col_names	Character: Name of the vector.

**Value**

data.frame.

**Author(s)**

EDG

---

xtdescribe	<i>Describe longitudinal dataset</i>
------------	--------------------------------------

---

**Description**

This function emulates the xtdescribe function in Stata.

**Usage**

```
xtdescribe(x, id_col = 1, time_col = 2, n_patterns = 9)
```

**Arguments**

x	data.frame: Longitudinal data with ID and time variables.
id_col	Integer: The column position of the ID variable.
time_col	Integer: The column position of the time variable.
n_patterns	Integer: The number of patterns to display.

**Value**

data.frame: Summary of participation patterns, returned invisibly.

**Author(s)**

EDG

**Examples**

```
## Not run:  
# Load example longitudinal dataset  
data(xt_example, package = "rtemis")  
  
# Describe the longitudinal structure  
xtdescribe(xt_example)  
  
## End(Not run)
```

# Index

- \* **datasets**
  - rtemis\_colors, [64](#)
- \* **internal**
  - check\_inherits, [6](#)
  - highlight, [40](#)
  - printdf, [57](#)
  - printls, [58](#)
  - repr, [60](#)
  - repr\_ls, [61](#)
  - repr\_S7name, [62](#)
- ansi256\_to\_hex, [4](#)
- bold, [4](#)
- check\_dependencies, [5](#)
- check\_inherits, [6](#)
- clean\_colnames, [6](#)
- clean\_int, [7](#)
- clean\_names, [8](#)
- col256, [8](#)
- col2grayscale, [9](#)
- col2hex, [10](#)
- color\_adjust, [13](#)
- color\_fade, [13](#)
- color\_invertRGB, [14](#)
- color\_mix, [14](#)
- color\_op, [15](#)
- color\_txt\_columns, [16](#)
- colorgrad, [10](#)
- ddSci, [17](#), [58](#), [67](#), [71](#)
- desaturate, [18](#)
- df\_movecolumn, [18](#)
- drange, [19](#)
- dt\_describe, [20](#)
- dt\_inspect\_types, [21](#)
- dt\_keybin\_reshape, [21](#)
- dt\_merge, [22](#)
- dt\_names\_by\_attr, [24](#)
- dt\_nunique\_perfeat, [24](#)
- dt\_pctmatch, [25](#)
- dt\_pctmissing, [26](#)
- dt\_set\_autotypes, [26](#)
- dt\_set\_clean\_all, [28](#)
- dt\_set\_cleanfactorlevels, [27](#)
- dt\_set\_logical2factor, [29](#)
- export\_plotly, [30](#)
- factor\_NA2missing, [31](#)
- fct\_describe, [31](#)
- filter\_order, [32](#)
- fmt, [33](#)
- fmt\_gradient, [34](#)
- get\_loaded\_pkg\_version, [36](#)
- get\_mode, [37](#)
- get\_output\_type, [37](#)
- get\_vars\_from\_rules, [38](#)
- getcharacternames (getnames), [35](#)
- getdatenames (getnames), [35](#)
- getfactornames (getnames), [35](#)
- getlogicalnames (getnames), [35](#)
- getnames, [35](#)
- getnamesandtypes, [36](#)
- getnumericnames (getnames), [35](#)
- graph\_node\_metrics, [39](#)
- gray, [39](#)
- green, [40](#)
- highlight, [40](#)
- iflengthy, [41](#)
- index\_col\_by\_attr, [42](#)
- init\_project\_dir, [42](#)
- inspect\_type, [21](#), [43](#)
- is\_constant, [44](#)
- is\_discrete, [44](#)
- italic, [45](#)
- labelify, [45](#)
- list2csv, [46](#)
- lotri2edgelist, [47](#)
- make\_path, [48](#)
- matchcases, [48](#)
- mgetnames, [50](#)
- msg, [51](#)

`msg0 (msg)`, [51](#)  
`msgdatetime`, [52](#)  
`msgdone`, [52](#)  
`msgstart`, [53](#)  
`muted`, [53](#)  
  
`names_by_class`, [54](#)  
  
`orange`, [55](#)  
  
`path.expand`, [48](#)  
`pcat`, [55](#)  
`previewcolor`, [56](#)  
`printf`, [57](#)  
`printls`, [58](#)  
  
`qstat`, [59](#)  
  
`recycle`, [60](#)  
`repr`, [60](#)  
`repr_ls`, [61](#)  
`repr_S7name`, [62](#)  
`rnormmat`, [63](#)  
`rt_letters`, [66](#)  
`rt_reactable`, [66](#)  
`rtemis_colors`, [64](#)  
`rtemisutils (rtemisutils-package)`, [3](#)  
`rtemisutils-package`, [3](#)  
`rtpalette`, [65](#)  
`rtversion`, [65](#)  
`rules2medmod`, [67](#)  
`runifmat`, [68](#)  
  
`setdiffsym`, [69](#)  
`sge_submit`, [69](#)  
`show_pad`, [71](#)  
`show_range`, [71](#)  
`size`, [72](#)  
  
`table_column_attr`, [72](#)  
`thin`, [73](#)  
  
`underline`, [74](#)  
`uniprot_get`, [74](#)  
`uniquevalsperfeat`, [75](#)  
  
`vec2df`, [76](#)  
  
`xtdescribe`, [76](#)