Package: ggrrr (via r-universe)

October 11, 2024

Title Addressing Annoyances and Irritations

Version 0.0.0.9024

Description Visualisation hacks, tabular data helpers, fonts, caching, tidy data functions. It is an swiss army knife, jack of all trades.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Biarch false

Imports devtools, dplyr, glue, grid, gridExtra, gtable, huxtable, openxlsx, rlang, tibble, tidyr, digest, rappdirs, fs, ggplot2, magrittr, flextable, htmltools, officer, patchwork, purrr, pillar, readr, rstudioapi, stringr, tidyselect, systemfonts, extrafont, scales, colorspace, svglite, rsvg, utils, stats, graphics, grDevices, knitr, base64enc, ragg, lifecycle, pdftools

Suggests tidyverse, rmarkdown, gt, rJava, testthat (>= 3.0.0), Rttf2pt1 (>= 1.3.12), Cairo

VignetteBuilder knitr

RoxygenNote 7.3.1

URL https://terminological.github.io/ggrrr/index.html, https://github.com/terminological/ggrrr

BugReports https://github.com/terminological/ggrrr/issues

Config/testthat/edition 3

SystemRequirements Google Chrome or other Chromium-based browser. chromium: chromium (rpm) or chromium-browser (deb)

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

RemoteUrl https://github.com/terminological/ggrrr

RemoteRef 0.0.0.9024

RemoteSha 389aaaf72370f643a85728edebce72be36737afe

Contents

.substitute_fonts *Pick a locally installed font family that matches requested*

Description

Pick a locally installed font family that matches requested

Usage

```
.substitute_fonts(family, quiet = TRUE)
```

Arguments

family	the font family requested
quiet	do not print warnings.

Examples

```
try({
.substitute_fonts(c("Roboto","Arial","Kings","Unmatched"))
})
```

as.character.rendered_plot *Convert a rendered_plot object to a character*

Description

Convert a rendered_plot object to a character

Usage

```
## S3 method for class 'rendered_plot'
as.character(x, ...)
```

Arguments

x	the rendered_plot
...	not used

Value

a named vector

as.character.rendered_table *Convert a rendered_table object to a character*

Description

Convert a rendered_table object to a character

Usage

```
## S3 method for class 'rendered_table'
as.character(x, ...)
```

Arguments

x	the rendered_table
...	not used

Value

a named vector

Examples

```
hux = iris %>% hux_default_layout()
tmp = hux %>% hux_save_as(tempfile())
as.character(tmp)
```

as.long_format_table *Convert a table to long format*

Description

Converts a square display table format to a long format suitable for applying as a sequence of formatting operations in a google doc or as a ggplot. Currently only plain dataframes and huxtables are supported but flextables look very doable. Only a limited subset of formatting features is implemented at present as supported by roogledocs. The output format is a simple dataframe with the following columns:

Usage

```
as.long_format_table(table, ...)
```

Arguments

table	the input table (e.g. a huxtable)
...	passed onto subclass methods

Details

- Character: label - non blank text (a single space is OK but not an empty string) - Integer: row - must be an integer, 1-based from top left - Integer: col - must be an integer, 1-based from top left - Integer: rowSpan - must be an integer, minimum value 1 - Integer: colSpan - must be an integer, minimum value 1 - Character: fontName - font name as seen in font drop down of google docs e.g "Roboto","Arial","Times New Roman", unrecognised values will be displayed as Arial - Character: fontFace - one of "bold", "bold.italic", "italic", "plain" - Numeric: fontSize - in points - Character: fillColour - as a hex string e.g. "#aaaaaa". N.b. British English spelling (sorry) - Numeric: leftBorderWeight - border weight in points - minimum size that appears in google docs is 0.5 - Numeric: rightBorderWeight - Numeric: topBorderWeight - Numeric: bottomBorderWeight - Character: alignment - one of "START","CENTER","END" - Character: valignment - one of "TOP","MIDDLE","BOTTOM"

It also has an attribute ‘colWidths‘ which is a vector the same length as the width of the table containing the relative widths of the columns. The overall table width is decided on rendering.

So not supported at the moment are border line types, border colours, control of padding, row height control, alignment on a decimal point, complex content / markup in cells.

Value

a format that is considered valid for roogledocs::RoogleDocs$updateTable()

`as.struct_list` *Cast to a list of structures*

Description

TODO: testing this empty input (must specify .class) struct_list - identity simple struct - wrapped list of struct_lists is flattened list of structs is wrapped nested plain lists / struct_lists are flattened

Usage

```
as.struct_list(x, .class = NULL)
```

Arguments

`x`	a list
`.class`	the type of structures in the list

Value

a structured list

as_vars *Reuse tidy-select syntax outside of a tidy-select function*

Description

Reuse tidy-select syntax outside of a tidy-select function

Usage

```
as_vars(tidyselect, data = NULL)
```

Arguments

tidyselect	a tidyselect syntax which will be evaluated in context by looking for a call in the call stack that includes a dataframe as the first argument
data	(optional) a specific dataframe with which to evaluate the tidyselect

Value

a list of symbols resulting from the evaluation of the tidyselect in the context of the current call stack (or a provided data frame)

bind_rows_with_factors *Bind rows for colums with factors*

Description

Bind_rows works until there are factors with a set of different levels then it throws a wobbly. This handles that particular situation by combining factor levels.

Usage

```
bind_rows_with_factors(...)
```

Arguments

...	a list of dataframes

Value

the union of those dataframes. Factor levels are combined with a superset of all levels

Examples

```
library(tidyverse)
bind_rows_with_factors(iris,
 ggplot2::diamonds %>% dplyr::rename(Species = cut)) %>%
 dplyr::pull(Species) %>%
 levels()
```

---

breaks_log1p *A scales breaks generator for log1p scales*

Description

A scales breaks generator for log1p scales

Usage

```
breaks_log1p(n = 5, base = 10)
```

Arguments

n	the number of breaks
base	the base for the breaks

Value

a function for ggplot scale breaks

Examples

```
library(tidyverse)
ggplot2::ggplot(diamonds, ggplot2::aes(x=price))+
  ggplot2::geom_density()+
  ggplot2::scale_x_continuous(trans="log1p", breaks=ggrrr::breaks_log1p())
```

---

cached *A simple pass-through cache for complex or long running operations*

Description

executes expr and saves the output as an RDS file indexed by has of code in expr and the hash of input variables (which should contain any variable inputs)

Usage

```
cached(
  .expr,
  ...,
  .nocache = getOption("cache.disable", default = FALSE),
  .cache = getOption("cache.dir", default = rappdirs::user_cache_dir("ggrrr")),
  .prefix = getOption("cache.item.prefix", default = "cached"),
  .stale = getOption("cache.stale", default = Inf)
)
```

Arguments

.expr	the code the output of which requires caching. Other than a return value this should not create side effects or change global variables.
...	inputs that the code in expr depends on and changes in which require the code re-running, Could be Sys.Date()
.nocache	an option to defeat the caching which can be set globally as options("cache.disable"=TRUE)
.cache	the location of the cache as a directory. May get its value from options("cache.dir") or the default value of rappdirs::user_cache_dir("ggrrr")
.prefix	a name of the operation so that you can namespace the cached files and do selective clean up operations on them
.stale	the length of time in days to keep cached data before considering it as stale. can also be set by options("cache.stale")

Value

the output of .expr which will usually be a value

Examples

```
colName = "Petal.Width"
{
  iris[[colName]]
} %>% cached(iris, colName, .prefix="example", .cache=tempdir())
```

---

cache_clear *Clear data from the passthrough cache for complex or long running operations*

Description

Clear data from the passthrough cache for complex or long running operations

Usage

```
cache_clear(
  .cache = getOption("cache.dir", default = rappdirs::user_cache_dir("ggrrr")),
  .prefix = ".*",
  interactive = TRUE
)
```

Arguments

.cache	the location of the cache as a directory. May get its value from options("ggrrr.cache.dir") or the default value of rappdirs::user_cache_dir("ggrrr")
.prefix	a regular expression matching the prefix of the cached item, so that do selective clean up operations. defaults to everything.
interactive	suppress 'are you sure¿ warning with a FALSE value (defaults to TRUE)

Value

nothing. called for side effects

Examples

```
cache_clear(.prefix="example", .cache=tempdir(), interactive=FALSE)
```

---

cache_delete_stale *Delete stale files in a cache*

Description

Staleness is determined by the number of days from 2am on the current day in the current timezone. A item cached for only one day becomes stale at 2am the day after it is cached. The time is configurable and option(cache.time_day_starts = 0) would be midnight. Automated analysis using caches and updated data should ensure that analysis does not cross this time point otherwise it may end up using old data.

Usage

```
cache_delete_stale(
  .cache = getOption("cache.dir", default = rappdirs::user_cache_dir("ggrrr")),
  .prefix = ".*",
  .stale = getOption("cache.stale", default = Inf)
)
```

Arguments

.cache	the location of the cache as a directory. May get its value from options("cache.dir") or the default value of rappdirs::user_cache_dir("ggrrr")
.prefix	a name of the operation so that you can namespace the cached files and do selective clean up operations on them
.stale	the length of time in days to keep cached data before considering it as stale.

Value

nothing. called for side effects.

---

cache_download *Download a file into a local cache.*

Description

This function copies a remote file to a local cache once and makes sure it is reused.

Usage

```
cache_download(
  url,
  ...,
  .nocache = getOption("cache.disable", default = FALSE),
  .cache = getOption("cache.download.dir", default =
    rappdirs::user_cache_dir("ggrrr-download")),
  .stale = getOption("cache.stale", default = Inf),
  .extn = NULL
)
```

Arguments

url	the url to download
...	passed to ‘utils::download.file()‘
.nocache	if set to TRUE all caching is disabled
.cache	the location of the downloaded files
.stale	how long to leave this file before replacing it.
.extn	the file name extension

Value

the path to the downloaded file

check_font *Ensures a font is available.*

Description

This checks to see if a font exists. If missing it will try and install from ‘google fonts‘ or ‘brick.io‘. If nothing can be done it will suggest alternatives from ‘fonts_available()‘. In all cases this will make the font available to ‘systemfonts‘ (for ‘ragg‘ and ‘svg‘ devices), and ‘extrafonts‘ (for ‘pdf‘ etc). Webfonts are automatically downloaded into the users font directory and from there will be picked up by ‘cairo‘ devices in theory, and system pdf/svg viewers. In practice this is a bit hit and miss.

Usage

```
check_font(family)
```

Arguments

family	a font family name or names

Value

the font family name if it can be located or an alternative if not.

Examples

```
check_font(c("Roboto","Arial","Kings","EB Garamond"))
extrafont::fonts()
fonts_available(c("Roboto","Arial","Kings","EB Garamond"))

plot = ggplot2::ggplot(ggplot2::diamonds, ggplot2::aes(x=carat,y=price,color = color))+
  ggplot2::theme_minimal(base_family="Roboto")+
  ggplot2::geom_point()+
  ggplot2::annotate("label",x=2,y=10000,label="Hello \u2014 world", family="Kings")+
  ggplot2::labs(tag = "A")+
  ggplot2::xlab("Carat\u2082")+
  ggplot2::ylab("price\u2265")


if (FALSE) {

  # font but no unicode support
  tmp = tempfile(fileext = ".pdf")
  pdf(tmp)
  plot
  dev.off()
  utils::browseURL(tmp)

  # font and unicode support
```

```
  tmp = tempfile(fileext = ".pdf")
  cairo_pdf(tmp)
  plot
  dev.off()
  utils::browseURL(tmp)

  # font and unicode support
  tmp = tempfile(fileext = ".png")
  png(tmp)
  plot
  dev.off()
  utils::browseURL(tmp)

  # font and unicode support
  tmp = tempfile(fileext = ".png")
  ragg::agg_png(tmp)
  plot
  dev.off()
  utils::browseURL(tmp)

  # font and unicode support
  tmp = tempfile(fileext = ".svg")
  svglite::svglite(tmp)
  plot
  dev.off()
  utils::browseURL(tmp)

  # Does not work - "family 'Roboto' not included in postscript() device"
  # however:  names(grDevices::postscriptFonts()) includes Roboto
  tmp = tempfile(fileext = ".eps")
  postscript(tmp)
  plot
  dev.off()
  utils::browseURL(tmp)

  # This does work but rasterises output at low fidelity
  tmp = tempfile(fileext = ".eps")
  cairo_ps(tmp)
  plot
  dev.off()
  utils::browseURL(tmp)

  # This fully works
  tmp = tempfile(fileext = ".ps")
  Cairo::CairoPS(tmp)
  plot
  dev.off()
  utils::browseURL(tmp)

}
```

code_snip *Display a code snippet*

Description

Pulls out a code snippet based on a start and end tags as comments within the code

Usage

```
code_snip(
  type,
  filename,
  startMatches = "START",
  endMatches = "END",
  includeStart = FALSE,
  includeEnd = FALSE,
  sep = "\n...\n"
)
```

Arguments

type	the code type
filename	the source code file
startMatches	a regex that matched start lines
endMatches	a regex that matches end lines
includeStart	is the regex inclusive of the start line or not
includeEnd	is the regex inclusive of the end line or not
sep	a seperator

Value

a text string of the selected code

code_snip_by_line *Display a code snippet*

Description

Pulls out a code snippet based on a vector of start and end lines.

Usage

```
code_snip_by_line(type, filename, starts = 1, ends = Inf, sep = "\n...\n")
```

Arguments

type	The code type (as understood by the minted latex plugin)
filename	The source code filename
starts	a vector of start indices (as line numbers)
ends	a vector of end indices (as line numbers)
sep	a seperator

Value

a formatted string based on the file

---

cran *Make sure packages available on CRAN are installed*

Description

Make sure packages available on CRAN are installed

Usage

```
cran(cran_deps)
```

Arguments

cran_deps	a vector of package names

Value

nothing

Examples

```
# cran("tidyverse")
```

cut_date *Places a set of dates withing a regular time series*

Description

where the periodicity of the time series is expressed as numbers of days, weeks, months quarters, or years.

Usage

```
cut_date(dates, full_seq = dates, factor = FALSE, ...)
```

Arguments

dates	a set of dates
full_seq	a full sequence of allowable dates as created by 'full_seq_dates()'. Alternatively a vector of dates will some regular periodicity, that will be used as an input for 'full_seq_dates()', if missing this will be derived from the data itself.
factor	return the result as an ordered factor with the date ranges as a label. if false this returns a date vector where the date is
...	if full_seq is not give, or a plain vector of dates, other options for 'full_seq_dates()' can be set here. E.g. ('fmt="%d/%m/%Y", period="1 week")

Value

a set of dates, representing the start (or end) of the period the date falls into, where the period is defined by 'full_seq' - which is usually defined by 'full_seq_dates()'

Examples

```
# dates = as.Date(c("2020-01-01","2020-02-01","2020-01-15","2020-02-03",NA))
# fs = full_seq_dates(dates, "2 days")
# dates - cut_date(dates, fs)
# cut_date(dates,fs,TRUE)

# A weekly set of dates:
# dates2 = Sys.Date() + floor(stats::runif(50,max=10))*7

# in this specific situation the final date is not truncated because the
# input data is seen as an exact match for the whole output period.
# cut_date(dates2, fmt = "%d/%b", factor = TRUE)

# if the input dates don't line up with the output dates
# there may be incomplete coverage of the first and last category.
# where the cutting results in short periods. In this
# instance the first and last periods are truncated to prevent them
# being counted as complete when they are in fact potentially missing a few days worth of data:
# cut_date(dates2, fmt = "%d/%b", factor = TRUE, period = "-2 weeks", anchor="sun")
```

`cut_integer` *Cut and label an integer valued quantity*

Description

Deals with some annoying issues classifying integer data sets, such as ages, into groups. where you want to specify just the change over points as integers and clearly label the resulting ordered factor.

Usage

```
cut_integer(
  x,
  cut_points,
  glue = "{label}",
  lower_limit = -Inf,
  upper_limit = Inf,
  ...
)
```

Arguments

`x`	a vector of integer valued numbers, e.g. ages, counts
`cut_points`	a vector of integer valued cut points which define the lower boundaries of conditions
`glue`	a glue spec that may be used to generate a label. It can use {low}, {high}, {next_low}, or {label} as values.
`lower_limit`	the minimum value we should include (this is inclusive for the bottom category) (default -Inf)
`upper_limit`	the maximum value we should include (this is also inclusive for the top category) (default Inf)
`...`	not used

Value

an ordered factor of the integer

Examples

```
cut_integer(stats::rbinom(20,20,0.5), c(5,10,15))
cut_integer(floor(stats::runif(100,-10,10)), cut_points = c(2,3,4,6), lower_limit=0, upper_limit=10)
```

cut_time *Places a set of dates withing a regular time series*

Description

where the periodicity of the time series is expressed as numbers of days, weeks, months quarters, or years.

Usage

```
cut_time(
  timepoints,
  full_seq = timepoints,
  unit = attr(timepoints, "unit"),
  day_zero = attr(timepoints, "day_zero"),
  factor = FALSE,
  ...
)
```

Arguments

timepoints	a set of times (defined by count of periods from a zero day - see 'date_to_time()')
full_seq	a full sequence of allowable dates as created by 'full_seq_dates()'. Alternatively a vector of dates will some regular periodicity, that will be used as an input for 'full_seq_dates()', if missing this will be derived from the data itself.
unit	the unit of the timepoints in terms of "1 week"
day_zero	the origin of the timepoints
factor	return the result as an ordered factor with the date ranges as a label. if false this returns a date vector where the date is
...	if full_seq is not give, or a plain vector of dates, other options for 'full_seq_dates()' can be set here. E.g. ('fmt="%d/%m/%Y", period="1 week")

Value

a set of dates, representing the start (or end) of the period the date falls into, where the period is defined by 'full_seq' - which is usually defined by 'full_seq_dates()'

Examples

```
#dates = as.Date(c("2020-01-01","2020-02-01","2020-01-15","2020-02-03",NA))
#fs = full_seq_dates(dates, "2 days")
#dates - cut_date(dates, fs)
#cut_date(dates,fs,TRUE)

# A weekly set of dates:
# dates2 = Sys.Date() + floor(stats::runif(50,max=10))*7
```

```
#times2 = date_to_time(dates2)

# in this specific situation the final date is not truncated because the
# input data is seen as an exact match for the whole output period.
#cut_time(times2, fmt = "%d/%b", factor = TRUE)

# if the input dates don't line up with the output dates
# there may be incomplete coverage of the first and last category.
# where the cutting results in short periods. In this instance
# the first and last periods are truncated to prevent them
# being counted as complete when they are in fact potentially missing a few days worth of data:
#cut_time(times2, fmt = "%d/%b", factor = TRUE, period = "-2 weeks", anchor="sun")
#times2 - cut_time(times2, fmt = "%d/%b", factor = FALSE, period = "-2 weeks", anchor="sun")
```

`data_supplement` *Create a function list that allows for supplementary tables (as huxtables) to be added to a XLSX output file.*

Description

This function encapsulates a excel output file as a destination for data tables. With the output of this function you can add extra data to the supplement as a new sheet, or you can write the spreadsheet to disk. When each data table is written either the table can be written silently or returned so that it is included in a knitr document. This is controlled by ‘option("hide.supplementary.tables"=TRUE)‘.

Usage

```
data_supplement(
  ...,
  filename = "supplementary-material.xlsx",
  out = ggrrr::outputter(...),
  nameGlue = "Supplementary Table {index}"
)
```

Arguments

`...`	output location options will be passed to outputter(...) to define the location of the file
`filename`	the xlsx filename
`out`	an outputter (defaults to a default outputter)
`nameGlue`	What will the tables be named

Value

a list of 2 functions. $add_table(hux, caption, footnote, index), which takes a huxtable, caption, and index a writes the huxtable into a supplementary. $write() which writes the collection of tables to the excel file.

date_to_time *Convert a set of dates to numeric timepoints*

Description

Using a day_zero and a unit specification or a full sequence of dates (see 'full_seq_dates()')

Usage

```
date_to_time(
  dates,
  unit = .day_interval(dates),
  day_zero = getOption("day_zero", "2019-12-29")
)
```

Arguments

dates	a vector of dates to convert
unit	a specification of the unit of the resulting time series. Will be determined from periodicity of dates if not specified
day_zero	the origin of the conversion. Defaults to the beginning of the COVID pandemic

Value

a sequence of numeric time points as the number of periods since day zero

Examples

```
# DEPRECATED
# times = date_to_time(as.Date("2019-12-29")+0:100, "1 week")
# dates = time_to_date(times)
```

drawDetails.watermark *Internal function for drawing watermark on ggplots*

Description

Internal function for drawing watermark on ggplots

Usage

```
## S3 method for class 'watermark'
drawDetails(x, recording)
```

Arguments

x	A grid grob.
recording	A logical value indicating whether a grob is being added to the display list or redrawn from the display list.

Value

a grid object

---

fit_col_widths *Estimate column content widths*

Description

Widths are based on dataframe or huxtable content ignoring rowspans and potential for wrapping.

Usage

```
fit_col_widths(table)
```

Arguments

table	a table to get column content widths for.

Value

a vector of column widths

Examples

```
library(tidyverse)
iris %>% fit_col_widths()
```

---

fonts_available *Which fonts are available on this system without hitting webfonts.*

Description

Which fonts are available on this system without hitting webfonts.

Usage

```
fonts_available(family)
```

Arguments

family	a font family name or names

Value

the font family name if it can be located or an empty list otherwise

Examples

```
fonts_available(c("Arial","sdfdsfsd"))
```

`full_seq_dates` *Expand a data vector to the full range*

Description

Convert a vector of observation dates to a ordered sequence of every day in the time series

Usage

```
full_seq_dates(dates, period = "1 day", anchor = "start", fmt = "%d %b")
```

Arguments

`dates`	a vector of dates, possibly including NA values
`period`	the gap between observations as a number or, a negative number means the resulting sequence defines a end of time periods, a positive defines the beginning. may be an integer number of days, or a text string like '2 weeks', '-1 month', etc.
`anchor`	defines the day of week the periods start or end. either "start", "end", a day of the week, or a date
`fmt`	a strptime formatting string for date range labels.

Value

a vector of dates for complete between the minimum and maximum of dates, on the day of week of the anchoring date

Examples

```
# full_seq_dates(c("2020-01-01","2020-02-01","2020-01-15","2020-02-01",NA), "2 days")
```

full_seq_times *Generate a full regular timepoint sequence*

Description

Generate a full regular timepoint sequence

Usage

```
full_seq_times(
  timepoints,
  period = unit,
  unit = attr(timepoints, "unit"),
  day_zero = attr(timepoints, "day_zero"),
  ...
)
```

Arguments

timepoints	a set of timepoints relating to data
period	the desired interval between time points, e.g. "1 day". negative periods define the intervals as closed on the left
unit	the unit of the timepoints in terms of "1 week"
day_zero	the origin of the timepoints
...	passed to 'full_seq_dates()', paricularly anchor, and fmt, to define the day of week of the new sequence and the format of the labels.

Value

a complete set of timepoints on the same scale as the original but with potentially different frequency. This will probably involve non integer times

Examples

```
# DEPRECATED
# times = date_to_time(as.Date("2019-12-29")+0:100, "1 week")
# tmp = full_seq_times(times)
```

get_value_sets *Get a value set list of a dataframe*

Description

This function examines a dataframe and returns a list of the columns with sub-lists as all the options for factors. This provides programmatic access (and autocomplete) to the values available in a dataframe, and throws and early error if we try and access data by a variable that does not exist.

Usage

```
get_value_sets(df)
```

Arguments

df	a dataframe to examine

Value

a list of lists with the column name and the factor levels as list, as a ‘checked list‘.

gg_find_webfonts *Find webfonts for a ggplot*

Description

Resolves any missing fonts in a ggplot and tries to resolve them against webfont providers (Brick and Google Fonts) locally cacheing them. This function is the default for ‘gg_save_as‘

Usage

```
gg_find_webfonts(plot)
```

Arguments

plot	a ggplot

Value

a list of css webfont specifications

gg_formatted_table *Display a long format table as a ggplot object.*

Description

This is useful if you want to combine a formatted table with a plot in a multi-panel patchwork.

Usage

```
gg_formatted_table(
  longFormatTable,
  colWidths = NULL,
  tableWidthInches = 5.9,
  font = "Roboto",
  ...
)
```

Arguments

longFormatTable	a table - usually converted using as.long_format_table()
colWidths	(optional) the relative widths of the columns.
tableWidthInches	the maximum desired width of the plot. Text will be scaled to fit this width.
font	the default font family
...	passed to as.long_format_table if and only if the input is not already in that format.

Value

a ggplot object containing the table as a ggplot.

gg_hide_legend *Hide the legend of a plot*

Description

Hide the legend of a plot

Usage

```
gg_hide_legend()
```

Value

a theme

gg_hide_X_axis *Hide the x axis of a plot*

Description

Hide the x axis of a plot

Usage

```
gg_hide_X_axis()
```

Value

a theme

gg_hide_Y_axis *Hide the y axis of a plot*

Description

Hide the y axis of a plot

Usage

```
gg_hide_Y_axis()
```

Value

a theme

gg_label_size *Convert a label size from points to ggplot units*

Description

Labels like geom_text are in a random unit size which is only mysteriously connected to the size of text on axes

Usage

```
gg_label_size(pts)
```

Arguments

pts	label size in points

Value

a ggplot size aesthetic for labels

---

gg_narrow *Make a plot narrower*

Description

Make a plot narrower

Usage

```
gg_narrow(ang = 90)
```

Arguments

ang	the angle for the x labels

Value

a theme

---

gg_pedantic *An opinionated set of defaults for plots*

Description

This is a set of styles with a focus on making plots compact, and minimally fussy, and ensuring fonts are consistent between axes and labels. It sets default sizes for line widths and point sizes. It also switched the default png renderer in knitr to 'ragg::ragg_png' to allow for modern font support.

Usage

```
gg_pedantic(
  lineSize = 0.25,
  fontSize = 8,
  font = "Roboto",
  size = lineSize * 2,
  ...
)
```

Arguments

lineSize	the default line and shape size in ggplot units
fontSize	the base font size
font	the default font name.
size	the size of points (the default size aesthetic)
...	passed to ‘ggplot2::theme‘

Value

nothing

---

gg_resize_legend *Make the legend smaller*

Description

Make the legend smaller

Usage

```
gg_resize_legend(pointSize = 0.75, textSize = 6, spaceLegend = 0.75)
```

Arguments

pointSize	- the ggplot size of lines or points
textSize	- the size in pts of the text
spaceLegend	- degree of spacing between items in the scale (defines overall size)

Value

a theme

`gg_save_as` *Save a plot to multiple formats*

Description

Saves a ggplot object to disk at a set physical size. Allows specific maximum dimensions with an optional target aspect ratio to fit into specific configurations for publication. e.g. a half page plot or a third of a 2 column page. Allows output in pdf for journal publication or png for inclusion in documents, and makes sure that the outputs are near identical.

Usage

```
gg_save_as(...)
```

Arguments

`...`	Arguments passed on to `.gg_save_as` `filename` base of target filename (excluding extension). `plot` a ggplot `size` a standard size see 'std_size' `maxWidth` maximum width in inches `maxHeight` maximum height in inches `aspectRatio` defaults to maxWidth/maxHeight `formats` some of svg, png, pdf, Rdata, eps, ... `webfontFinder` a function that takes a plot and returns a properly formatted css specification for webfonts in the plot. This is for internal use and does not need to be changed.

Details

For maximum cross platform reproducibility we are using the combination of 'systemfonts' for font management, 'svglite' to render the canonical output 'rsvg' to convert that to pdf, and 'ragg' to for bitmap formats. In some situations 'rsvg' fails in which case we fall back to rendering in a headless chrome instance. This rather complicated pipeline ensures modern webfont support, and editable SVG or PDF.

Value

the output is an sensible default object that can be displayed given the context it is called in, for example if knitting an RMarkdown document a link to the png file for embedding, if latex a link to the pdf file.

Examples

```
try({
.gg_pedantic(fontSize = 6)
p = ggplot2::ggplot(mtcars, ggplot2::aes(mpg, wt, colour=as.factor(cyl))) +
  ggplot2::geom_point()
# p %>% .gg_save_as(filename="~/tmp/plot_example",maxWidth=4,maxHeight=4)
p %>% .gg_save_as(filename=tempfile(),maxWidth=2,maxHeight=1.5)

plot = ggplot2::ggplot(ggplot2::diamonds, ggplot2::aes(x=carat,y=price,color = color))+
  ggplot2::geom_point()+
  ggplot2::annotate("label",x=2,y=10000,label="Hello \u2014 world", family="Kings")+
  ggplot2::labs(tag = "A")+
  ggplot2::xlab("Carat\u2082")+
  ggplot2::ylab("price\u2265")

# plot %>% .gg_save_as(filename="~/tmp/plot_example_2")
res = plot %>% .gg_save_as(filename=tempfile(), formats=c("png","eps"))
as.character(res)
res
})
```

---

gg_set_size_defaults *Set sizes in ggplot uniformly*

Description

Set the default sizes of lines, points and fonts in ggplot geoms, and text labels in ggplot axes to get a single consistent look and feel.

Usage

```
gg_set_size_defaults(
  lineSize = 0.5,
  fontSizePts = 4 + lineSize * 8,
  font = "Roboto",
  size = lineSize * 2
)
```

Arguments

lineSize	the width of lines
fontSizePts	the size of labels and other on plot text in pts.
font	the font family name
size	the size of points (the default size aesthetic)

Value

nothing

Examples

```
library(tidyverse)
gg_set_size_defaults(lineSize = 0.25)
```

`gg_set_X_angle` *Set the angle of the x axis labels of a plot*

Description

Also sets horizontal and vertical alignment correctly, and does top and bottom axes.

Usage

```
gg_set_X_angle(ang = 60)
```

Arguments

`ang`	the angle for the x labels

Value

a theme

`gg_simple_table` *A simple table as a ggplot patchwork object, no customisation allowed*

Description

A simple table as a ggplot patchwork object, no customisation allowed

Usage

```
gg_simple_table(df, pts = 8, font = "sans", unwrapped = FALSE)
```

Arguments

`df`	the dataframe with the table data. Column names will become headings
`pts`	text size in points
`font`	the font family
`unwrapped`	- set this to TRUE if you want to add to a patchwork and use patchwork::wrap_plots(p,list(table))

Value

A gtable object (i.e. a grob) optionally wrapped as a patchwork plot.

Examples

```
if (FALSE) {
  gg_simple_table(tibble::tibble(x=c(1,2,3),y=c(5,4,3)),pts=10)
}
```

`gg_tiny_theme` *A space saving ggplot theme*

Description

A ggplot theme with minimal fluff and with the defaults set small.

Usage

```
gg_tiny_theme(baseSize = 8, font = "Roboto")
```

Arguments

`baseSize`	the size of the base font.
`font`	the font family name

Value

a ggplot theme

Examples

```
if (interactive()) {
  ggplot2::ggplot(ggplot2::diamonds,
    ggplot2::aes(x=carat,y=price,color=color))+
    ggplot2::geom_point()+
    gg_tiny_theme()
}
```

`gg_watermark` *Add in a watermark to plots*

Description

Add in a watermark to plots

Usage

```
gg_watermark(
  lab = "DRAFT",
  disable = getOption("ggrrr.disable.watermark", default = FALSE)
)
```

Arguments

lab	the watermark label (DRAFT)
disable	- global option to disable all watermarks options("ggrrr.disable.watermark"=TRUE)

Value

a watermark layer

---

here *Drop in replacement for 'here' ('here' pkg)*

Description

Drop in replacement for 'here' ('here' pkg)

Usage

```
here(..., projRoot = .locate_project())
```

Arguments

...	the relative path within the project
projRoot	the project root - defaults to '.locate_project()'

Value

a path

Examples

```
try(.here("vignettes"))
```

---

html_pdf_converter *Convert html to pdf depending on what is available on the platform*

Description

If html2pdfr is installed it will use that by default, if not it will fall back to a headless chrome instance.

Usage

```
html_pdf_converter(html, filename, maxWidth, maxHeight)
```

Arguments

html	the html fragment
filename	the pdf filename
maxWidth	the maximum page width in inches
maxHeight	the maximum page height in inches

Value

nothing called for side effects

---

hux_auto_widths *Calculate a sensible column and table width for a huxtable based on its content.*

Description

Calculate a sensible column and table width for a huxtable based on its content.

Usage

```
hux_auto_widths(hux, target = "html", including_headers = FALSE)
```

Arguments

hux	the huxtable
target	the expected output (could be "docx"/"odt", "xlsx") which are the only options that matter
including_headers	Should we try and fit the header contents as well (TRUE) or let those wrap (FALSE).

Value

the huxtable with the width options set.

hux_bind_rows *Bind rows for huxtables*

Description

Sometimes vanilla bind_rows gets confused.

Usage

```
hux_bind_rows(...)
```

Arguments

...	a list of huxtables

Value

a single huxtable

hux_default_layout *A tidy article theme for huxtables*

Description

The main aim is to get something that works with google docs when you copy and paste.

Usage

```
hux_default_layout(
  hux,
  defaultFontSize = 8,
  defaultFont = "Roboto",
  headerRows = 1
)
```

Arguments

hux	a huxtable object
defaultFontSize	default size of font in points (8)
defaultFont	the font family name
headerRows	the number of rows that are headers

Value

the formatted huxtable.

Examples

```
library(tidyverse)
hux = iris %>% hux_default_layout()
```

`hux_insert_start` *Insert row at start maintaining format*

Description

Insert row at start maintaining format

Usage

```
hux_insert_start(hux, ..., fill = "", colspan = 1)
```

Arguments

`hux`	a huxtable
`...`	stuff to insert into cells
`fill`	padding for empty cells.
`colspan`	how far to span first inserted cell?

Value

a huxtable with row inserted at start in the same format

`hux_nest_group` *Make a huxtable narrower*

Description

Converts row spanning columns into column spanning header rows making a table narrower but longer. The column that is being moved is retained to allow for the appearance of indentation.

Usage

```
hux_nest_group(t, col = 1)
```

Arguments

`t`	the huxtable
`col`	the column index you want to nest into the row above

Value

a narrower huxtable

hux_save_as *Save a table to a variety of formats*

Description

depending on the context return the correct format for a document. The basic output here is to use HTML as an output if possible and convert it to an image or a PDF that can then be included into a latex document for example.

Usage

```
hux_save_as(...)
```

Arguments

...	Arguments passed on to .hux_save_as hux the huxtable to save filename the filename, which may omit the extension size a 'std_size' list entry maxWidth or the maximum width in inches maxHeight and either the maximum height in inches aspectRatio or the minimum allowed aspect ratio formats if the extension is omitted, all the formats described here will be saved. Currently supported outputs are "html","png","pdf","docx","xlsx" defaultFontSize the default font size sheetname if saving as an xlsx file. pdfConverter a function that takes an HTML fragment and returns a pdf file. webfontFinder a function that takes a set of font families and returns a css webfonts directive

Value

the output depends on if the function is called in a knitr session. It maybe the HTML or a link to the pdf output for example.

Examples

```
try({
 hux = iris %>% huxtable::as_hux() %>%
   huxtable::theme_mondrian(font="Roboto")
 out = .hux_save_as(hux, tempfile())
 # browseURL(out$html)

 out2 = .hux_save_as(hux, tempfile(), formats=c("pdf","png"))
 as.character(out2)
```

```
 # The resulting pdf has fonts embedded & is multipage.
})
```

`hux_set_caption` *Set a huxtable caption as a first row*

Description

Keeps the same formatting as the rest of the table

Usage

```
hux_set_caption(hux, caption)
```

Arguments

`hux`	a huxtable
`caption`	caption text

Value

a huxtable with first row caption

`hux_set_font` *Set the font family and size in a huxtable globally*

Description

Set the font family and size in a huxtable globally

Usage

```
hux_set_font(hux, defaultFontSize = 8, defaultFont = "Roboto")
```

Arguments

`hux`	a huxtable table
`defaultFontSize`	the desired font size
`defaultFont`	the desired font

Value

the altered huxtable

hux_set_footer *Add a footer row as a final row in a huxtable*

Description

Keeps the same formatting as the rest of the table except for borders

Usage

```
hux_set_footer(hux, footer)
```

Arguments

hux	a huxtable
footer	footer text

Value

a huxtable with last row footer

hux_sprintf *A sprintf alternative that handles NA values gracefully (ish)*

Description

A sprintf alternative that handles NA values gracefully (ish)

Usage

```
hux_sprintf(fmt, ..., na.text = "–")
```

Arguments

fmt	sprintf format string
...	sprintf inputs
na.text	an string to replace NA values with.

Value

a string value

hux_tidy *Convert a dataframe to a huxtable with nested rows and columns.*

Description

The assumption here is that the input data is a long format tidy dataframe with both rows and columns specified by values of the ‘rowGroupVars‘ and ‘colGroupVars‘ columns. The long format (sparse) table is translated into a nested tree of rows (using ‘rowGroupVars‘) and a nested tree of columns (from ‘colGroupVars‘). Individual data items are placed in the cell intersecting these two trees. If there are multiple matches an additional layer of grouping is added to the columns.

Usage

```
hux_tidy(
  tidyDf,
  rowGroupVars,
  colGroupVars,
  missing = "—",
  na = "—",
  displayRedundantColumnNames = FALSE,
  ...
)
```

Arguments

tidyDf	A dataframe with row groupings (as a set of columns) and column groupings (as a set of columns) and data, where the data is in a tidy format with a row per "cell" or cell group.
rowGroupVars	A dplyr::vars(...) column specification which will define how rows are grouped
colGroupVars	A dplyr::vars(...) column specification with defines how columns will be grouped
missing	If there is no content for a given rowGroup / colGroup combination then this character will be used as a placeholder
na	If there are NA contents then this character will be used.
displayRedundantColumnNames	if there is one column per column group the name of that column may be irrelevant (e.g. if there is a ‘col_name‘, ‘value‘ fully tidy format) and ‘col_name‘ is in the ‘colGroupVars‘ list then the name of the column ‘value‘ is redundant and not displayed by default. However sometimes you want to display this if you have named it as something specific e.g. including the units. If there is more than one column per ‘colGroup‘ the column titles are needed and kept.
...	passed to ‘hux_default_layout()‘

Value

a huxtable table

hux_to_ggplot *Convert a huxtable to a ggplot object*

Description

Useful if you need to include a formatted table in a figure with a plot

Usage

```
hux_to_ggplot(hux, width = 5.9)
```

Arguments

hux	the huxtable
width	the desired ggplot width

Value

a ggplot object of the right width

intersecting_group_by *Create a dataframe with groups mathing a range of predicates*

Description

Create a new data frame including duplicate rows where the rows fulfil a potentially overlapping set of conditions specified as named predicates (as formulae)

Usage

```
intersecting_group_by(.data, ..., .colname)
```

Arguments

.data	a data frame
...	a set of predicates specified like case_whens syntax, such as mpg < 5 ~ "gas guzzlers"
.colname	the name of the new group

Value

a new dataframe containing the overlapping groups which may create duplicates of individual rows.

Examples

```
library(tidyverse)
iris %>% dplyr::group_by(Species) %>% intersecting_group_by(
  Sepal.Length > mean(Sepal.Length) ~ "Long",
  Sepal.Width > mean(Sepal.Width) ~ "Wide"
)
```

`knit_print.rendered_plot` *Knit a rendered_plot object*

Description

Knit a rendered_plot object

Usage

```
## S3 method for class 'rendered_plot'
knit_print(x, options, ...)
```

Arguments

`x`	the rendered_plot
`options`	the chunk options
`...`	not used

Value

nothing - used for side effects

`knit_print.rendered_table` *Knit a rendered_table object*

Description

Knit a rendered_table object

Usage

```
## S3 method for class 'rendered_table'
knit_print(x, ...)
```

Arguments

x	the rendered_table
...	not used

Value

nothing - used for side effects

`knit_versioned` *Knit to a versioned file in a sub-directory of the project*

Description

used in a knitr preamble to direct the output to a subdirectory of the project — title: "Analysis 1" output: html_document knit: ggrrr::knit_versioned("output/analysis-1") —

Usage

```
knit_versioned(
  directory = NULL,
  ...,
  datedFile = !datedSubdirectory,
  datedSubdirectory = FALSE
)
```

Arguments

directory	the root of the output - can be an absolute path or a relative path interpreted as relative to the root of the project.
...	ignored
datedFile	do you want the filename to have the date appended (defaults TRUE)?
datedSubdirectory	do you want the files to be placed in a dated subdirectory (defaults FALSE)?

Details

This can only work when deployed as a library and hence no standalone version of it exists, because the fully qualified packagename has to be used.

Value

nothing. called for side effects

logit_trans *logit scale*

Description

Perform logit scaling with right axis formatting. To not be used directly but with ggplot (e.g. ggplot2::scale_y_continuous(trans = "logit")

Usage

```
logit_trans(n = 5, ...)
```

Arguments

n	number of breas
...	not used

Value

A scales object

Examples

```
tibble::tibble(pvalue = c(0.001, 0.05, 0.1), fold_change = 1:3) %>%
  ggplot2::ggplot(ggplot2::aes(fold_change , pvalue)) +
  ggplot2::geom_point() +
  ggplot2::scale_y_continuous(trans = "logit")
```

map2_struct *Map over two inputs*

Description

These functions are variants of `map()` that iterate over two arguments at a time.

Usage

```
map2_struct(.x, .y, .f, ..., .progress = FALSE)
```

Arguments

`.x, .y`	A pair of vectors, usually the same length. If not, a vector of length 1 will be recycled to the length of the other.
`.f`	A function, specified in one of the following ways: • A named function. • An anonymous function, e.g. `\(x, y) x + y` or `function(x, y) x + y`. • A formula, e.g. `~ .x + .y`. You must use `.x` to refer to the current element of `x` and `.y` to refer to the current element of `y`. Only recommended if you require backward compatibility with older versions of R.

`...`	Additional arguments passed on to the mapped function.
	We now generally recommend against using `...` to pass additional (constant) arguments to `.f`. Instead use a shorthand anonymous function:
	`# Instead of`
	`x |> map(f, 1, 2, collapse = ",")`
	`# do:`
	`x |> map(\(x) f(x, 1, 2, collapse = ","))`
	This makes it easier to understand which arguments belong to which function and will tend to yield better error messages.

`.progress`	Whether to show a progress bar. Use `TRUE` to turn on a basic progress bar, use a string to give it a name, or see progress_bars for more details.

Value

a 'struct_list'

`map_struct` *Apply a function to each element of a vector*

Description

The map functions transform their input by applying a function to each element of a list or atomic vector and returning an object of the same length as the input.

- `map()` always returns a list. See the `modify()` family for versions that return an object of the same type as the input.
- `map_lgl()`, `map_int()`, `map_dbl()` and `map_chr()` return an atomic vector of the indicated type (or die trying). For these functions, `.f` must return a length-1 vector of the appropriate type.
- `map_vec()` simplifies to the common type of the output. It works with most types of simple vectors like Date, POSIXct, factors, etc.
- `walk()` calls `.f` for its side-effect and returns the input `.x`.

Usage

```
map_struct(.x, .f, ..., .progress = FALSE)
```

Arguments

`.x`	A list or atomic vector.
`.f`	A function, specified in one of the following ways: • A named function, e.g. `mean`. • An anonymous function, e.g. `\(x) x + 1` or `function(x) x + 1`. • A formula, e.g. `~ .x + 1`. You must use `.x` to refer to the first argument. Only recommended if you require backward compatibility with older versions of R. • A string, integer, or list, e.g. `"idx"`, `1`, or `list("idx", 1)` which are shorthand for `\(x) pluck(x, "idx")`, `\(x) pluck(x, 1)`, and `\(x) pluck(x, "idx", 1)` respectively. Optionally supply `.default` to set a default value if the indexed element is `NULL` or does not exist.

`...`	Additional arguments passed on to the mapped function.
	We now generally recommend against using `...` to pass additional (constant) arguments to `.f`. Instead use a shorthand anonymous function:
	`# Instead of`
	`x |> map(f, 1, 2, collapse = ",")`
	`# do:`
	`x |> map(\(x) f(x, 1, 2, collapse = ","))`
	This makes it easier to understand which arguments belong to which function and will tend to yield better error messages.

`.progress`	Whether to show a progress bar. Use `TRUE` to turn on a basic progress bar, use a string to give it a name, or see progress_bars for more details.

Value

a ‘struct_list‘

---

non_cran *Make sure github packages are installed.*

Description

Use a locally checked out version if available.

Usage

```
non_cran(name, github, force = FALSE, subdir = "", ...)
```

Arguments

name	the name of the package
github	something like "github-repo/project-name"
force	will only update a loaded package if TRUE (defaults to FALSE)
subdir	if the package is in a subdirectory of the github repo
...	passed to devtools::install_github

Value

nothing

Examples

```
# non_cran("patchwork", "thomasp85/patchwork")
```

---

optional_fn *Get an optional function without triggering a CRAN warning*

Description

You want to use a function if it is installed but don't want it to be installed as part of your package and you don't want to reference it as part of the Imports or Suggests fields in a package DESCRIPTION.

Usage

```
optional_fn(
  pkg,
  name,
  alt = function(...) {
     stop("function `", pkg, "::", name, "(...)` not available")

    }
)
```

Arguments

pkg	the package name
name	the function you wish to use
alt	a function that can be used instead

Value

the function you want if available or the alternative

Examples

```
fn = .optional_fn("openssl", "md5", digest::digest)
as.character(fn(as.raw(c(1,2,3))))
```

---

outputter *Generate a versioned file name in a subdirectory.*

Description

This function generates a function that resolves a file path fragment to a specific file location, accounting for a versioning strategy involving the current date. The defaults create a naming strategy that places an file in the "output" sub-directory of the current project with a filename suffix including the date.

Usage

```
outputter(
  directory = .here("output"),
  ...,
  datedFile = !datedSubdirectory,
  datedSubdirectory = FALSE
)
```

Arguments

directory	the root of the output
...	not used must be empty
datedFile	do you want the filename to have the date appended?
datedSubdirectory	do you want the files to be placed in a dated subdirectory?

Value

a function that takes a filename and boolean delete parameter. When called with a filename component this function will return the absolute path of a file which is versioned with date. If the file exists and delete=TRUE it is deleted. (allowing for libraries that refuse to overwrite existing files)

Examples

```
out = outputter("~/output",datedSubdirectory=TRUE)
out("file.png")
```

percent_trans *percentage scale*

Description

display a 0-1 scale as 0-100

Usage

```
percent_trans()
```

Details

‘r lifecycle::badge(stage = "deprecated")‘

Value

A scales object

pmap_struct *Map over multiple input simultaneously (in "parallel")*

Description

These functions are variants of `map()` that iterate over multiple arguments simultaneously. They are parallel in the sense that each input is processed in parallel with the others, not in the sense of multi-core computing, i.e. they share the same notion of "parallel" as `base::pmax()` and `base::pmin()`.

Usage

```
pmap_struct(.l, .f, ..., .progress = FALSE)
```

Arguments

`.l`	A list of vectors. The length of `.l` determines the number of arguments that `.f` will be called with. Arguments will be supply by position if unnamed, and by name if named. Vectors of length 1 will be recycled to any length; all other elements must be have the same length. A data frame is an important special case of `.l`. It will cause `.f` to be called once for each row.
`.f`	A function, specified in one of the following ways: • A named function. • An anonymous function, e.g. `\(x, y, z) x + y / z` or `function(x, y, z) x + y / z`

- A formula, e.g. `~ ..1 + ..2 / ..3`. This syntax is not recommended as you can only refer to arguments by position.

`...` Additional arguments passed on to the mapped function.

We now generally recommend against using `...` to pass additional (constant) arguments to `.f`. Instead use a shorthand anonymous function:

```
# Instead of
x |> map(f, 1, 2, collapse = ",")
# do:
x |> map(\(x) f(x, 1, 2, collapse = ","))
```

This makes it easier to understand which arguments belong to which function and will tend to yield better error messages.

`.progress` Whether to show a progress bar. Use `TRUE` to turn on a basic progress bar, use a string to give it a name, or see progress_bars for more details.

Value

a ‘struct_list‘

`pmixnorm` *Mixture of normal distribution PDFs*

Description

‘r lifecycle::badge("deprecated")‘

Usage

```
pmixnorm(q, means, sds, weights = rep(1, length(means)), na.rm = FALSE)
```

Arguments

`q`	vector of quantiles.
`means`	a vector of normal distribution means
`sds`	a vector of normal distribution sds
`weights`	a vector of weights
`na.rm`	remove distributions which have NA for mean or sd

Value

the pdf of the mixture distribution.

Examples

```
pmixnorm(q=c(2,20), means=c(10,13,14), sds=c(1,1,2), weights=c(2,2,3))
```

print.rendered_plot *Print a rendered_plot object*

Description

Print a rendered_plot object

Usage

```
## S3 method for class 'rendered_plot'
print(x, ...)
```

Arguments

x	the rendered_plot
...	not used

Value

nothing - used for side effects

print.rendered_table *Print a rendered_table object*

Description

Print a rendered_table object

Usage

```
## S3 method for class 'rendered_table'
print(x, ...)
```

Arguments

x	the rendered_table
...	not used

Value

nothing - used for side effects

qmixnorm *Mixture of normal distribution quantiles*

Description

‘r lifecycle::badge("deprecated")‘

Usage

```
qmixnorm(p, means, sds, weights = rep(1, length(means)), na.rm = FALSE)
```

Arguments

p	vector of probabilities.
means	a vector of normal distribution means
sds	a vector of normal distribution sds
weights	a vector of weights
na.rm	remove distributions with NA values for mean or sd

Value

the value of the yth quantile

Examples

```
qmixnorm(p=c(0.025,0.5,0.975), means=c(10,13,14), sds=c(1,1,2))
```

rebuild_fonts *Rebuild font caches*

Description

This repopulates ‘systemfonts‘ from the webfont cache and then ‘extrafont‘ from ‘systemfonts‘. This does a full rebuild and will be slow (depending a bit)

Usage

```
rebuild_fonts()
```

Value

nothing

reset_fonts *Reset any custom fonts*

Description

This wipes a lot of cached font data.

Usage

```
reset_fonts(
  confirm = utils::askYesNo(msg = "Are you sure?", default = FALSE),
  web = FALSE,
  fonts = FALSE
)
```

Arguments

confirm	set to TRUE to automatically confirm
web	also clear webfont cache? (default FALSE)
fonts	also clear any downloaded fonts or converted afm files? (default FALSE)

Value

nothing

rowwise_mutate *Create new data in a strictly row-wise fashion without vectorisation*

Description

Applies an expression to each row and assignes it to a new column. Per-row failures are handled with default values (NAs) or can be intercepted by the user with a tryCatch(...) expression. There are many other ways to do a similar thing in ‘dplyr‘ and ‘purrr‘ but they are all more complicated than I expect them to be.

Usage

```
rowwise_mutate(.data, ..., .onerror = function(e, ...) NA)
```

Arguments

.data	a dataframe. grouping is ingnored
...	a named list of expressions similar to mutate but where the expressions to be evaluated are evaluated in only in the context of the current row - and are not vectorised. This does not support [dplyr::accross] syntax.
.onerror	a function that is called for

Value

a dataframe the same length as input with additional or altered columns

Examples

```
# calculations are scoped only to current row. Hence max(x) == x always:
iris %>% rowwise_mutate(
  widths = Sepal.Width+max(Petal.Width),
  lengths = Sepal.Length+max(Petal.Length),
  tmp = tibble::tibble(a=1, b=2)) %>%
dplyr::glimpse()

# This is different to standard dplyr behaviour when the additional tibble
# column is considered. standard dplyr rowwise does something unexpected:
iris %>% dplyr::rowwise() %>% dplyr::mutate(
  widths = Sepal.Width+max(Petal.Width),
  lengths = Sepal.Length+max(Petal.Length),
  tmp = tibble::tibble(a=1, b=2)) %>%
dplyr::glimpse()

# As expressions are not vectorised we can use normal if ... else ... statements
# and errors can be handled and default values provided.
suppressWarnings(
iris %>% rowwise_mutate(
  tmp = if (Petal.Width > 2.0) stop("error message: ",Petal.Width) else Petal.Width,
  .onerror = function(e) -Petal.Width
) %>%
dplyr::glimpse()
)

# The default values
# are evaluated in the same context as the original expression, but only are
# defaults for all the columns so makes most sense when a default value is given

suppressWarnings(
iris %>% rowwise_mutate(
  tmp = if (Petal.Width > 2.0) stop("too wide petals: ",Petal.Width) else Petal.Width,
  tmp2 = if (Sepal.Width > 4) stop("too wide sepals: ",Sepal.Width) else Sepal.Width,
  .onerror = function(e) Inf
) %>%
dplyr::glimpse()
)
```

---

scale_colour_subtype *A discrete colour scale for dividing where there is a natural ordered subgrouping into groups and subgroups*

Description

This is intended to combine with 'scale_fill_subtype' when we want to divide major groupings differently to minor groups

Usage

```
scale_colour_subtype(
  subclasses,
  class_colour = "black",
  subclass_colour = "grey50",
  na.value = "grey50",
  aesthetics = "color",
  ...
)
```

Arguments

subclasses	a vector containing the count of the subcategories, e.g. c(2,3,4) defines 3 major categories and a total of 9 sub-categories
class_colour	the colour for major group divisions
subclass_colour	the colour for sub group divisions
na.value	missing value colour
aesthetics	this only really makes sense for color scales.
...	passed on to ggplot2::discrete_scale()

Value

a ggplot scale

Examples

```
library(tidyverse)

# prep some data:
data = ggplot2::diamonds %>%
  dplyr::mutate(color_cut = sprintf("%s (%s)",color,cut)) %>%
  dplyr::group_by(color,cut,color_cut) %>%
  dplyr::count() %>%
  dplyr::ungroup() %>%
  dplyr::mutate(color_cut = ordered(color_cut))

# work out the number of subgroups for each group:
subgroups = data %>%
  dplyr::select(color,cut) %>%
  dplyr::distinct() %>%
  dplyr::group_by(color) %>%
  dplyr::count() %>%
  dplyr::pull(n)

# plot as a horizontal stacked bar chart using color brewer as the main
# colour axis. N.b. having enough different colours here is important
ggplot2::ggplot(data, ggplot2::aes(y=1,x=n, fill=color_cut, color=color_cut))+
  ggplot2::geom_bar(stat="identity",orientation = "y")+
```

```
ggrrr::scale_fill_subtype(.palette = scales::brewer_pal,
  palette="Accent", subclasses = subgroups)+
ggrrr::scale_colour_subtype(subclasses=subgroups)+
ggrrr::gg_hide_Y_axis()+
ggrrr::gg_narrow()
```

`scale_fill_subtype` *Discrete fill or colour scale where there is a natural ordered subgrouping*

Description

If you have a categorical variable defining colour or fill and it has a natural grouping you can use this to have a colour scale involving major colors defining the major groupings, and these are progressively lightened for each of the subcategories.

Usage

```
scale_fill_subtype(
  .palette,
  subclasses,
  ...,
  undefined = "#606060",
  lighten = NA,
  na.value = "grey50",
  aesthetics = "fill"
)
```

Arguments

`.palette`	the palette for the major groupings, either as a function e.g. 'scales::viridis_pal', or as a manual set of colors e.g. 'c("#FF0000","#00FF00","#0000FF")'. if a function can be either discrete or continuous palette.
`subclasses`	a vector containing the count of the subcategories, e.g. c(2,3,4) defines 3 major categories and a total of 9 sub-categories
`...`	additional options to be passed to the major palette function, e.g. 'option="magma"', or to 'ggplot2::discrete_scale()', e.g. 'alpha=0.5'
`undefined`	If the number of sub-categories in the data is longer than defined in 'subclasses', the extra categories are assumed to be an set of "other" categories, which will be coloured using this base colour
`lighten`	The factor by which to lighten the colour at each step of the subgrouping. If left blank this will calculate a fraction based on the number of levels of the subgroup. Otherwise if, e.g. 0.5 the first sub category will be the full saturation, the second 0.5 saturation, the third 0.25 saturation, the fourth 0.125 and so on.
`na.value`	what colour for NA values.
`aesthetics`	this is a fill scale by default but can be used for colour by setting this to "color" or both as c("fill","color")

Value

a ggplot scale

Examples

```
library(tidyverse)

# prep some data:
data = ggplot2::diamonds %>%
  dplyr::mutate(color_cut = sprintf("%s (%s)",color,cut)) %>%
  dplyr::group_by(color,cut,color_cut) %>%
  dplyr::count() %>%
  dplyr::ungroup() %>%
  dplyr::mutate(color_cut = ordered(color_cut))

# work out the number of subgroups for each group:
subgroups = data %>%
  dplyr::select(color,cut) %>%
  dplyr::distinct() %>%
  dplyr::group_by(color) %>%
  dplyr::count() %>%
  dplyr::pull(n)

# plot as a horizontal stacked bar chart using color brewer as the main
# colour axis. N.b. having enough different colours here is important
ggplot2::ggplot(data, ggplot2::aes(y=1,x=n, fill=color_cut, color=color_cut))+
  ggplot2::geom_bar(stat="identity",orientation = "y")+
  scale_fill_subtype(.palette = scales::brewer_pal,
    palette="Accent", subclasses = subgroups)+
  scale_colour_subtype(subclasses=subgroups)+
  gg_hide_Y_axis()+
  gg_narrow()
```

---

scale_x_log1p *A log1p x scale*

Description

A log1p x scale

Usage

```
scale_x_log1p(..., n = 5, base = 10, sf = 2)
```

Arguments

...	Other arguments passed on to scale_(x|y)_continuous()
n	the number of major breaks
base	the base for the logarithm
sf	significant figures

Value

a ggplot scale

---

scale_x_logit *A logit x scale*

Description

A logit x scale

Usage

```
scale_x_logit(..., n = 5, sf = 2)
```

Arguments

...	Other arguments passed on to scale_(x|y)_continuous()
n	the number of major breaks
sf	significant figures

Value

a ggplot scale

---

scale_x_percent *A percentage x scale*

Description

A percentage x scale

Usage

```
scale_x_percent(..., sf = 2)
```

Arguments

...	Other arguments passed on to scale_(x|y)_continuous()
sf	significant figures

Value

a ggplot scale

scale_y_log1p *A log1p y scale*

Description

A log1p y scale

Usage

```
scale_y_log1p(..., n = 5, base = 10, sf = 2)
```

Arguments

...	Other arguments passed on to scale_(x|y)_continuous()
n	the number of major breaks
base	the base for the logarithm
sf	significant figures

Value

a ggplot scale

scale_y_logit *A logit y scale*

Description

A logit y scale

Usage

```
scale_y_logit(..., n = 5, sf = 2)
```

Arguments

...	Other arguments passed on to scale_(x|y)_continuous()
n	the number of major breaks
sf	significant figures

Value

a ggplot scale

Examples

```
tibble::tibble(pvalue = c(0.001, 0.05, 0.1), fold_change = 1:3) %>%
 ggplot2::ggplot(ggplot2::aes(fold_change , pvalue)) +
 ggplot2::geom_point() +
 scale_y_logit(n=8)
```

---

scale_y_percent *A percentage y scale*

Description

A percentage y scale

Usage

```
scale_y_percent(..., sf = 2)
```

Arguments

...	Other arguments passed on to scale_(x|y)_continuous()
sf	significant figures

Value

a ggplot scale

Examples

```
tibble::tibble(pvalue = c(0.001, 0.05, 0.1), fold_change = 1:3) %>%
 ggplot2::ggplot(ggplot2::aes(fold_change , pvalue)) +
 ggplot2::geom_point() +
 scale_y_percent()
```

---

sprintf_list *Sprintf with a list input*

Description

A variant of sprintf that work well with inputs that are in the format of a list. Good examples of which are the quantile functions

Usage

```
sprintf_list(format, params, na.replace = "-")
```

Arguments

format	the format string
params	the inputs as a list (rather than as a set of individual numbers)
na.replace	a value to replace NA values with.

Value

the formatted string

Examples

```
# generate a mixture confidence interval from a set of distributions
sprintf_list("%1.2f [%1.2f\u2013%1.2f]",
 qmixnorm(p=c(0.5,0.025,0.975),
 means=c(10,13,14), sds=c(1,1,2)))
```

---

std_size *Standard image and paper sizes*

Description

The width and height of images to fit scientific publication standards.

Usage

```
std_size
```

Format

A list with width and height in inches

---

struct_flatten *Ensure 'struct_list' is a flat list of 'structs'*

Description

Unlike 'purrr::list_flatten' this is recursive for one reason which is that a 'struct_list' must only contain 'structs'.

Usage

```
struct_flatten(x)
```

Arguments

x	a potentially nested list of 'struct_lists'.

Value

a flat ‘struct_list‘ of ‘structs‘

`subset-struct-list` *Manipulate structured lists*

Description

These functions allow generic list behaviour.

Usage

```
## S3 method for class 'struct_list'
c(...)

## S3 method for class 'struct_list'
rep(x, ...)

## S3 method for class 'struct_list'
x$y

## S3 method for class 'struct_list'
x[...]

## S3 replacement method for class 'struct_list'
x[...] <- value

## S3 method for class 'struct_list'
x[[...]]

## S3 replacement method for class 'struct_list'
x[[...]] <- value
```

Arguments

`...`	generic support
`x`	a ‘struct_list‘
`y`	item to retrieve
`value`	the value

Value

a ‘struct_list‘ with all the items

Functions

- `c(struct_list)`: Repeat a 'struct_list'
- `rep(struct_list)`: Repeat a 'struct_list'
- `$`: Subset a 'struct_list'
- `[`: Subset a 'struct_list'
- `` `[`(struct_list) <- value ``: Assign a subset to a 'struct_list'
- `[[`: get a value from a 'struct_list'
- `` `[[`(struct_list) <- value ``: set a single value in a 'struct_list'

Examples

```
x = struct(a=1,b=2,c=1:3,.class="test")
y = struct(a=4,b=5,c=1:3,.class="test")
z= tibble::tibble(a= 1:10, b=rep(c(x,y),5))

z$b

c(x,y)
c(rep(x,5),y)
class(c(rep(x,5),rep(y,5))[[1]])

as.struct_list(list(x,y))

#' x = struct(a=1,b=2,c=1:3,.class="test")
class(rep(c(x,y),5)[[1]]) == "test"
class(rep(x,5))

a = (rep(c(x,y),5))
a[[1]] = y
a
```

`summarise_with_totals` *Summarise a subgroup and create a summary row*

Description

Summarise and include a total row, or a row including the summary for the whole group, into a factor list. This looks and feels like a natural summarisation step, but applies the summarisation both to the subgroups and to the data ungrouped by one level. The additional group result is included as a new row. allows for a natural grouped and ungrouped summarisation

Usage

```
summarise_with_totals(
  .data,
  ...,
  .groups = NULL,
  .total = "Total",
  .total_first = FALSE
)
```

Arguments

.data	a dataframe
...	the summarisation specification
.groups	what to do with the grouping after summarisation (same as dplyr::summarise)
.total	name of the total row which will be added into a factor list.
.total_first	should the total be before or after the groups

Value

a summarised dataframe with the additional totals or group row

Examples

```
library(tidyverse)
diamonds %>%
 dplyr::group_by(color,cut) %>%
 summarise_with_totals(
    mpg = sprintf("%1.1f \u00B1 %1.1f", mean(price), stats::sd(price)),
    .total = "Overall"
 )
```

---

time_to_date *Convert a set of timepoints to dates*

Description

Convert a set of timepoints to dates

Usage

```
time_to_date(
  timepoints,
  unit = attr(timepoints, "unit"),
  day_zero = getOption("day_zero", "2019-12-29")
)
```

Arguments

timepoints	a set of numeric time points
unit	the period / unit of the time points, which will be extracted from timepoints if possible
day_zero	the zero day of the time series, will be extracted from timepoints if possible

Value

a vector of dates

Examples

```
# DEPRECATED
# times = date_to_time(as.Date("2019-12-29")+0:100, "1 week")
# dates = time_to_date(times)
```

---

unstable *Get unstable version of package*

Description

reboot a library reloading from a local development copy if it exists locally alternatively get the most up to date github package.

Usage

```
unstable(pkg = "ggrrr", org = "terminological")
```

Arguments

pkg	the package to load the unstable version of
org	the github organisation

Value

nothing

Examples

```
# ggrrr::unstable()
```

`$.checked_list` *Checked list accessor*

Description

A checked list is a sub class of list access operator that throws an error if you attempt to access a value that does not exist (rather than returning NULL) the point of this is to throw errors early if the data changes.

Usage

```
## S3 method for class 'checked_list'
x$y
```

Arguments

`x`	the list
`y`	the item

Value

the value of the list item or an error if it does not exist

Index