Package: rcamisc (via r-universe)

November 14, 2024

Type Package

Title rcamisc: Plot, wrangle and stay sane

Version 0.3.0

Maintainer Ruben C. Arslan <rubenarslan@gmail.com>

Description Miscellaneous helpers for plotting and staying sane.

Encoding UTF-8

LazyData true

Language en_GB

URL https://github.com/rubenarslan/rcamisc

BugReports https://github.com/rubenarslan/rcamisc/issues

License MIT + file LICENSE

Depends R (>= 3.0.2)

Imports dplyr, utils, ggplot2, pryr, reshape2, stringr, rio, renv, rmarkdown, rstudioapi

Suggests knitr, roxygen2

RoxygenNote 7.1.2

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Remotes rstudio/renv

Config/pak/sysreqs make libicu-dev libssl-dev libx11-dev zlib1g-dev

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

RemoteUrl https://github.com/rubenarslan/rcamisc

RemoteRef HEAD

RemoteSha 2bb97b274267ee17375572c6266e8e17d617a8e3

Contents

`aggregate2sources` *aggregates two variables from two sources into one*

Description

Takes two variables with different missings and gives one variable with values of the second variable substituted where the first had missings.

Usage

```
aggregate2sources(
  df,
  new_var = NULL,
  var1 = NULL,
  var2 = NULL,
  remove_old_variables = TRUE
)
```

Arguments

`df`	data.frame or variable
`new_var`	new variable name
`var1`	first source. Assumed to be new_var.x (default suffixes after merging)
`var2`	second source. Assumed to be new_var.y (default suffixes after merging)
`remove_old_variables`	Defaults to not keeping var1 and var2 in the resulting df.

Examples

```
cars$dist.x = cars$dist
cars$dist.y = cars$dist
cars$dist.y[2:5] = NA
cars$dist.x[10:15] = NA # sprinkle missings
cars$dist = NULL # remove old variable
cars = aggregate2sources(cars, 'dist')
```

`amigoingmad` *Am I going mad?*

Description

It's easy to attach packages that overwrite functions from other packages. Especially dplyr has a lot of conflicts with base packages, MASS and plyr. Because some of these conflicts do not always lead to error messages, sometimes just incorrect behaviour, this function exists. Don't trust your faulty memory, just check whether dplyr's (or any other package's) functions are 'on top' if you so desire.

Usage

```
amigoingmad(package = "dplyr", fix = TRUE, iteration = 0)
```

Arguments

`package`	the package you want to be on top (loaded last), defaults to dplyr
`fix`	defaults to true. Detaches the desired package (without unloading) and loads it again. Won't work for base packages and can't overwrite functions that you defined yourself.
`iteration`	for internal use only, if set to 0 the function will call itself to check that it worked, if set to 1, it won't.

Examples

```
amigoingmad(fix = FALSE, package = 'rcamisc')
```

`bibliography` *build a bibliography bibtex file from your lockfile*

Description

Renv helps you maintain consistent package versions for a project. To be able to give due credit in a way that academics understand, it's helpful to be able to generate citations.

Usage

```
bibliography(
  overwrite_bib = FALSE,
  silent = FALSE,
  cite_only_directly_called = TRUE,
  lockfile_path = "renv.lock",
  bibliography_path = "bibliography.bibtex",
  cite_renv = !cite_only_directly_called
)
```

Arguments

`overwrite_bib`	whether to overwrite an existing bibtex file of the same name
`silent`	defaults to false. whether to cat out a nocite string to use in your header
`cite_only_directly_called`	whether to call only the packages you called yourself (default) or also their dependencies
`lockfile_path`	path to the packrat lock file to use
`bibliography_path`	path to the bibtex file to generate
`cite_renv`	whether to cite renv even if it's not loaded explicitly, defaults to the reverse of cite_only_directly_called

`geom_shady_smooth` *iterate adding ribbons to a ggplot2 plot at varying confidence levels to shade by confidence. Horribly inefficient, because smooth stat is computed every time, but flexible.*

Description

iterate adding ribbons to a ggplot2 plot at varying confidence levels to shade by confidence. Horribly inefficient, because smooth stat is computed every time, but flexible.

Usage

```
geom_shady_smooth(
  mapping = NULL,
  data = NULL,
  stat = "smooth",
  method = "auto",
  formula = y ~ x,
  se = TRUE,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  levels = c(0.6, 0.8, 0.95),
  base_alpha = 1,
  fill_gradient = NULL,
  fill = "black",
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	defaults to smooth
method	Smoothing method (function) to use, accepts either NULL or a character vector, e.g. "lm", "glm", "gam", "loess" or a function, e.g. MASS::rlm or mgcv::gam, stats::lm, or stats::loess. "auto" is also accepted for backwards compatibility. It is equivalent to NULL. For method = NULL the smoothing method is chosen based on the size of the largest group (across all panels). stats::loess() is used for less than 1,000 observations; otherwise mgcv::gam() is used with formula = y ~ s(x, bs = "cs") with method = "REML". Somewhat anecdotally, loess gives a better appearance, but is $O(N^2)$ in memory, so does not work for larger datasets. If you have fewer than 1,000 observations but want to use the same gam() model that method = NULL would use, then set method = "gam", formula = y ~ s(x, bs = "cs").
formula	Formula to use in smoothing function, eg. y ~ x, y ~ poly(x, 2), y ~ log(x). NULL by default, in which case method = NULL implies formula = y ~ x when

	there are fewer than 1,000 observations and `formula = y ~ s(x, bs = "cs")` otherwise.
`se`	Display confidence interval around smooth? (`TRUE` by default, see `level` to control.)
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`show.legend`	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.
`levels`	the confidence levels that are supposed to be displayed, defaults to 0.6, 0.8, 0.95
`base_alpha`	divided by length(levels)
`fill_gradient`	a vector of colors that has at least the same length as levels. Color each ribbon differently
`fill`	a single color for the ribbon
`...`	everything else is passed to and documented in `ggplot2::geom_smooth()`

Examples

```
data(beavers)
plot = ggplot2::ggplot(beaver1, ggplot2::aes(time, temp))
plot + geom_shady_smooth() + ggplot2::facet_wrap(~ day)
plot + geom_shady_smooth(fill = 'blue', levels = seq(0.05,0.95,0.1))
plot + geom_shady_smooth(size = 0.1, fill = '#49afcd', levels = seq(0.1,0.8,0.01))
plot + geom_shady_smooth(fill_gradient = c('red', 'orange', 'yellow'), base_alpha = 3)
```

`missingness_patterns` *missingness patterns*

Description

this function shows how common possible missingness patterns are. Emulates misschk in stata.

1. excludes any variables that don't have any missings, so as not to clutter output. Disable using omit_complete
2. sorts variables by number of missings, so that the usual suspects show up at the front.
3. displays number of missings accounted for by each pattern

Usage

```
missingness_patterns(
  df,
  min_freq = ifelse(relative, 1/nrow(df), 1),
  long_pattern = FALSE,
  print_legend = ifelse(long_pattern, FALSE, TRUE),
  show_culprit = TRUE,
  relative = FALSE,
  omit_complete = TRUE
)
```

Arguments

df	dataset
min_freq	show only patterns that occur at least this often. Defaults to 1 observation.
long_pattern	by default (FALSE) only shows column indices for space and legibility reasons.
print_legend	prints a legend for the column indices, defaults to FALSE if long_pattern is set
show_culprit	defaults to TRUE. In case a missingness pattern boils down to one variable, it will be shown here.
relative	defaults to FALSE. If true, percentages are shown (relative to total before excluding minimum frequency).
omit_complete	defaults to TRUE. Columns that don't have any missings are excluded.

Examples

```
data(ChickWeight)
ChickWeight[1:2,c('weight','Chick')] = NA
ChickWeight[3:5,'Diet'] = NA
names(ChickWeight); nrow(ChickWeight)
missingness_patterns(ChickWeight)
```

---

mtmm *multi trait multi method matrix*

Description

renders a MTMM using ggplot2. This function will split the variable names in a correlation matrix, or a data.frame. The first part will be used as the trait, the second as the method. Correlations are displayed as text, with the font size corresponding to absolute size. You can optionally supply a data frame of reliabilites to show in the diagonal.

Usage

```
mtmm(variables = NULL, reliabilities = NULL, split_regex = "_", cors = NULL)
```

Arguments

variables	data frame of variables that are supposed to be correlated
reliabilities	data frame of reliabilties: column 1: scale, column 2: rel. coefficient
split_regex	regular expression to separate construct and method from the variable name, splits on '.' by default
cors	you can also supply a (named) correlation matrix

Examples

```
data.mtmm = data.frame(
`Ach_self_report` = rnorm(200), `Pow_self_report` = rnorm(200), `Aff_self_report`= rnorm(200),
`Ach_peer_report` = rnorm(200),`Pow_peer_report`= rnorm(200),`Aff_peer_report` = rnorm(200),
`Ach_diary` = rnorm(200), `Pow_diary` = rnorm(200),`Aff_diary` = rnorm(200))
reliabilities = data.frame(scale = names(data.mtmm), rel = stats::runif(length(names(data.mtmm))))
mtmm(data.mtmm, reliabilities = reliabilities)
```

---

qplot_waffle *Waffle plot*

Description

Pass in a a variable and get a waffle plot. Useful to display simple counts or if the variable has different values, a square pie chart. If the variable has a length that makes the individual squares hard to see, consider showing hundreds, thousands etc.

Usage

```
qplot_waffle(
  x,
  shape = 15,
  rows = NULL,
  cols = NULL,
  drop_shadow_h = -0.3,
  drop_shadow_v = 0.3
)
```

Arguments

x	a variable with not too many unique values
shape	defaults to a filled square
rows	defaults to the rounded up square root of the number of values
cols	defaults to the rounded down square root of the number of values
drop_shadow_h	horizontal offset of the drop shadow, tinker with this to get a proper shadow effect
drop_shadow_v	vertical offset of the drop shadow

Details

To avoid the Hermann grid illusion, don't use dark colours.

Examples

```
qplot_waffle(rep(1:2,each=5))
```

`qplot_waffle_text` *Waffle plot (text)*

Description

Pass in a a variable and get a waffle plot. Useful to display simple counts or if the variable has different values, a square pie chart. If the variable has a length that makes the individual squares hard to see, consider showing hundreds, thousands etc.

Usage

```
qplot_waffle_text(
  x,
  symbol = fontawesome_square,
  rows = NULL,
  cols = NULL,
  drop_shadow_h = -0.9,
  drop_shadow_v = 0.9,
  font_family = "FontAwesome",
  font_face = "Regular",
  font_size = round(140/sqrt(length(x)))
)
```

Arguments

`x`	a variable with not too many unique values
`symbol`	pass a unicode symbol from FontAwesome here. Defaults to a square with rounded edges
`rows`	defaults to the rounded up square root of the number of values
`cols`	defaults to the rounded down square root of the number of values
`drop_shadow_h`	horizontal offset of the drop shadow, tinker with this to get a proper shadow effect
`drop_shadow_v`	vertical offset of the drop shadow
`font_family`	defaults to FontAwesome
`font_face`	defaults to Regular
`font_size`	defaults to round(140/sqrt(length(x)))

Details

This functions is like waffle_plot but it allows you to specify custom symbols from FontAwesome. Copypaste them from here: http://fontawesome.io/cheatsheet

To avoid the Hermann grid illusion, don't use dark colours.

Examples

```
## Not run:
qplot_waffle_text(rep(1:2,each=30), rows = 5)

## End(Not run)
```

---

qplot_waffle_tile *Waffle plot (tile)*

Description

Pass in a a variable and get a waffle plot. Useful to display simple counts or if the variable has different values, a square pie chart. If the variable has a length that makes the individual squares hard to see, consider showing hundreds, thousands etc.

Usage

```
qplot_waffle_tile(x, rows = NULL, cols = NULL)
```

Arguments

x	a variable with not too many unique values
rows	defaults to the rounded up square root of the number of values
cols	defaults to the rounded down square root of the number of values

Details

This function allows and requires the least tinkering, but also does not drop shadows. To avoid the Hermann grid illusion, don't use dark colours.

adapted from http://shinyapps.stat.ubc.ca/r-graph-catalog/ who adapted it from http://www.techques.com/question/17-17842/How-to-make-waffle-charts-in-R who adapted it from http://ux.stackexchange.com/a/46543/56341

Examples

```
qplot_waffle_tile(rep(1:2,each=500))
```

render_job *render an rmarkdown file in background using RStudio Jobs*

Description

if you want to

Usage

```
render_job(input, params = NULL, output_file = NULL)
```

Arguments

input	.Rmd document to be knitted
params	params to pass to the .Rmd
output_file	name of the output_file (and the job)

Examples

```
## Not run:
   render_job("document.Rmd", list(dataset = "df1"), "summary_df1.html")

## End(Not run)
```

repeat_last *repeat last non-NA value*

Description

Will repeat the last non-NA value. This is also known as carrying the last observation forward/backward. It's faster than zoo::na.locf http://rpubs.com/rubenarslan/repeat_last_na_locf and other alternatives. By specifying maxgap, you can choose not to bridge overly long gaps. By specifying forward = FALSE, you can carry the last observation backward.

Usage

```
repeat_last(x, forward = TRUE, maxgap = Inf, na.rm = FALSE)
```

Arguments

x	vector to be repeated
forward	carry last observation forward? or backward (FALSE)
maxgap	bridge only up to x NAs (defaults to Inf)
na.rm	whether to omit NAs at the beginning (defaults to FALSE)

Examples

```
x = c(NA,NA,1,NA,NA,NA,NA,NA,NA,NA,NA,2,3,4,NA,NA,NA,NA,NA,5, NA)
data.frame(x,
   repeat_last(x),
   repeat_last(x, forward = FALSE),
   repeat_last(x, maxgap = 5),
check.names = FALSE)
```

---

take_nonmissing *take only nonmissing*

Description

this function takes a subset of a dataset, omitting all cases with missings in variables specified in 'keep' and omitting all variables that still have missings after that. Good to see how large your dataset for a certain analysis will be and which covariates are 'free' in terms of sample size.

Usage

```
take_nonmissing(df, keep = c())
```

Arguments

df	dataset
keep	defaults to empty vector

Examples

```
data(ChickWeight)
ChickWeight[1:2,c('weight','Chick')] = NA
ChickWeight[3:4,'Diet'] = NA
names(ChickWeight); nrow(ChickWeight)
ChickWeight2 = take_nonmissing(ChickWeight, keep = c('weight'))
names(ChickWeight2); nrow(ChickWeight2)
```

---

view_in_excel *Open in Excel*

Description

Simple helper, so I don't complain about the slugginess of RStudio's View so much

Usage

```
view_in_excel(x)
```

Arguments

x	a dataframe to open in Excel

Examples

```
## Not run:
view_in_excel(Titanic)

## End(Not run)
```

Index